

YILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

Bir Endüst. İma. Sis. Veri Yapı.
Ana. ve Uyg.

Yüksek Lisans Tezi

Kaya Hamat

YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

Kot : R 371
Alındığı Yer : Y.T.Ü.
Tarih : 12.9.1995
Fatura : -
Fiyatı : 350.000 TL.
Ayniyat No : 1-15
Kayıt No : 51642
UDC :
Ek :

Y. T. Ü.
KÜTÜPHANE DOK. DAİ. BAŞKANLIĞI

YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ



BİR ENDÜSTRİYEL İMALAT SİSTEMİNDE VERİ
YAPILARININ ANALİZİ VE UYGULAMASI



YÜKSEK LİSANS TEZİ
END.MÜH. KAYA HAMAT

İSTANBUL 1991

V DİLLERİNDE YORDAMLAR ARASI

A. ANLAY ALGORİTMA DİLİ İLE

B. ALGORİTMA ETKİNLİĞİ

1. Anlay dilinde yordamlar arası

1

2. Anlay dilinde yordamlar arası

3. Anlay dilinde yordamlar arası

1

4. Anlay dilinde yordamlar arası

4

5. Anlay dilinde yordamlar arası

5

6. Anlay dilinde yordamlar arası

7. Anlay dilinde yordamlar arası

8

8. Anlay dilinde yordamlar arası

9. Anlay dilinde yordamlar arası

11

10. Anlay dilinde yordamlar arası

11

11. Anlay dilinde yordamlar arası

12. Anlay dilinde yordamlar arası

13

Bu tezi, hazırladığım sırada vefat eden ANNEANNE' me ithaf ediyorum.

Minnet ve Saygılarımla.

13. Anlay dilinde yordamlar arası

29

14. Anlay dilinde yordamlar arası

33

İÇİNDEKİLER

SAYFA NO

ÖZET

I

SUMMARY

III

1. ÜST DÜZEY PROGRAMLAMA DİLLERİNDE YORDAMLAR ARASI

VERİ-DENETİM İLETİŞİMİ VE ANLAT ALGORİTMA DİLİ İLE
ALGORİTMA KARMAŞIKLIĞI VE ALGORİTMA ETKİNLİĞİ

1.1. Üst Düzey Programlama Dillerinde yordamlar arası

1

veri- denetim iletişimi

1.1.1. Yordamların veri alanı sıra düzeni

1

1.1.2. Dış yordamlar arası veri iletişim türleri

4

1.1.2.1. Değer-Sonuç ile Çağırma Yöntemi

5

ve veri iletişimi

1.1.2.2. Kaynak ile çağırma yöntemi ve

8

veri iletişimi

1.2. Türkçe Algoritma Dili " ANLAT " Tanımı

11

1.2.1. Algoritma iletişimi için Seçenekler

11

Nelerdir ?

1.2.2. ANLAT Algoritma Dilinin Belgeleme

13

Amacıyla Kullanımı

1.2.3. ANLAT Algoritma Dili Komut Yapıları

14

1.2.4. Örnek Bir Algoritma

27

1.3. Algoritma Karmaşıklığı Ve Algoritma Etkinliği

29

Tahminleri

1.3.1. Algoritma Karmaşıklığı

29

1.3.2. Algoritma Etkinliği

33

İÇİNDEKİLER

SAYFA NO

1.3.3. Sistem Çözümleme Ve Yazılım Geliştirmede	37
Bu Bilgi Nasıl Uygulanabilir ?	74
2. DİZİ VERİ YAPISI VE ÖZEL TÜRLERİ İLE BAĞLAQSIZ YIGIT	83
VERİ YAPISI VE ÖZ YINELİ ALGORİTMALAR	83
2.1. Dizi Veri Yapısı	39
2.1.1. Dizi Veri Yapısı Nasıl İncelenir ?	39
2.1.2. Veri Yapısı Tanımı	40
2.1.3. Veri Yapısı Gösterimi	40
2.1.4. Veri Yapısı Denetim Grubuna Erişim	40
2.1.5. Veri Ögesine Erişim	42
2.2. Dizi Veri Yapısı Özel Türleri	45
2.2.1. Üçgen ve Kuşak Matrisler	45
2.2.1.1. Üçgen Matrisler	47
2.2.1.2. Kuşak Matrisler	52
2.2.2. Seyrek Matrisin Diziler ile Gösterimi	58
2.2.2.1. Seyrek Matris Tanımı	58
2.2.2.2. Seyrek Matris Gösterimi	58
(Üç Dizi ile Gösterim)	
2.2.2.3. Seyrek Matrisin Başlangıcına Ve	
Seyrek Matrisin Ögesine Erişim	60
2.2.2.4. Seyrek Matris Veri Yapısı Üzerinde	
Temel Algoritmalar	61
2.3. Bağlaqsız Yığıt	70
2.3.1. Yığıt Veri Yapısı Tanımı	70
2.3.2. Yığıt Veri Yapısının Dizi Üzerinde Gösterimi	71

İÇİNDEKİLER

SAYFA NO

2.3.3. Yığıt'a Ve Öğelerine Erişim	72
2.4. Özyineli Algoritma	74
3. LİSTE, KUYRUK, ÇİZGE VE AĞAC VERİ YAPILARI	83
3.1. Tek Bağlaçlı Doğrusal Liste	83
3.2. Çift Yönlü Bağlaçlı Doğrusal Ve Döngülü Listeler	100
3.3. Alt Liste Paylaşımsız Liste Yapıları	105
3.4. Çift Bağlı Döngülü Listenin Bilgisayar Sistem Yazılımında Kullanılışı	113
3.5. Bağlaçsız Döngülü Kuyruk	124
3.6. Dizilerin Bağlaçlı Yerleşimleri	128
3.7. Çizge Veri Yapısı	139
3.8. Ağaç Veri Yapısı	147
3.9. İkili Ağaç Veri Yapısı	151
4. BELLEK İÇİ SIRALAMA, BELLEK İÇİ ARAMA VE ANAHTARLAMA YÖNTEMLERİ	162
4.1. Bellek İçi Sıralama Yöntemleri	162
4.2. Bellek İçi Arama Yöntemleri	177
4.3. Anahtarlama Yöntemleri	191
5. UYGULAMA	201
6. SONUÇ	221
KAYNAKLAR	
ÖZGEÇMİŞ	

T E Ş E K K Ü R

Bu çalışma esnasında değerli fikirleri ve tavsiye-
leriyle bana yön veren hocam Sayın Yrd. Doç. Dr. Hüseyin
BAŞLIGİL'e teşekkürü bir borç bilirim.

Ayrıca bu çalışma esnasında benden maddi ve manevi
yardımını esirgemiyen Anş.Gör.A.Kürşad TÜRKER'e ve tezin
yazımında büyük emegi geçen End.Müh. B. Murat BAYIK'a
teşekkür ederim.

Ö Z E T

Günümüzde bilgisayar gelişen teknoloji ile beraber yaşantımızın hemen her safhasında yerini almıştır. Böyle güncel ve işlerliği olan bir konunun daha dinamik ve verimli bir kullanıma dönüştürülebilmesi bir problem olarak karşımıza çıkmaktadır. Problemin çözümü ise veri yapıları konusuna tamamen hakim olunarak işletmelerde dinamik bir veri tabanı oluşturulması ile sağlanabilmektedir. Daha verimli bir şekilde kullanılmalarını sağlamak için tasarlanmıştır.

Buradaki çalışma dört bölümde incelenmiştir. İlk bölümde veri yapıları konusuna genel giriş ve veri yapıları konusunun öğrenilmesinde çok büyük kolaylık sağlayan Türkçe algoritma dili "ANLAT"ın tanımı yapılmaktadır. Buna bağlı olarak algoritma karmaşıklığı ile algoritma etkinliği kavramları açıklanmaktadır.

Diğer bölümler ise anlat algoritma dilinin kullanıldığı ve esas veri yapılarının incelendiği kısımlardır.

Dizi veri yapısı ile bu yapının özel türleri olan üçgen, kuzak ve seyrek matris veri yapıları ile bunların özel gösterimleri ikinci bölümde incelenmiştir. Ayrıca bu bölümde yığıt veri yapısıyla özyineli algoritmalara da değinilmiştir.

Günümüzde en çok kullanılan veri yapısı türü olan liste veri yapısı ile daha çok mühendislik problemlerinin tanımı ve çözümünde kullanılan çizge ve ağaç veri yapıları üçüncü bölümde incelenmiştir. Aynı zamanda ağaç veri yapısının özel bir türü olan ikili ağaç veri yapısında sıralama problemlerine getirdiği yeni boyuttan dolayı bu bölümde incelenmiştir.

II

Bilgi işlem alanında sıralama, arama ve anahtarlama önemli yer tutar. Tipik bilgi işlem merkezinde işlem gücünün $1/4$ ile $1/3$ 'ü arasındaki kesimin sıralama ve aramaya ayrıldığı düşünülebilir. Bu nedenle değişik sıralama ve arama algoritmaları ile anahtarlama yöntemleri bilgi işlem açısından önem taşımaktadır. Bu yöntemler ise çalışmanın dördüncü bölümünde incelenmiştir.

Bu dört bölümde incelediğimiz veri yapıları konusu bilgisayara daha verimli bir şekilde kullanılmasını sağlamak için tasarlanan modern veri tabanı sistemlerine geçişte bilgi işlem etkinliği açısından büyük önem taşımaktadır.

III

S U M M A R Y

Together with developing technology, computers have took it's place at every stage of our todays life. We're facing with a problem of converting such daily subject into more dynamic and efficient use the solution of the problem may be obtained by having full knowledge on data structure and followingly construct a dynamic data base in organizations.

The study provided here in is examined under four chapters. A basic introduction have been made to data structures, and a Turkish algorithm language " ANLAT " which profoundly facilitates the learning of data structure have been presented, in the first chapter. Computational complexity algorithm and algorithm efficiency terms pertaining to this have been explained.

The other chapters are explaining the use of " ANLAT " algorithm language and the basic data structures.

Sequenced data structure and special types of this structure such as triangular band and sparse matrix data structures and special illustration of these have been examined under the second chapter. Furthermore, stack data structure and recursive algorithm have been summarized under this chapter.

List data structure which is the most extensively used type of data and graph and tree structure which is mostly used on describing and solving engineering problems are examined in chapter three. Binary tree data structure which is special type of tree data structure is examined under this chapter as well due to it's contrubution to sorting problems.

Sorting, searching and hashing plays an important role on data processing $1/4$ to $1/3$ of processing power in a typical data processing center is dedicated to sorting and searching. For this reason, various sorting and searching algorithms and hashing methods are important in spite of data processing these methods are examined in chapter four of the study.

The data structure examined under four chapters carries a big importance in spite of data processing efficiency on conveyance to modern data base systems which provides more efficient use of computers.

1. ÜST DÜZEY PROGRAMLAMA DİLLERİNDE YORDAMLAR ARASI VERİ-DENETİM İLETİŞİMİ VE ANLAT ALGORİTMA DİLİ İLE ALGORİTMA KARMASIKLIĞI VE ALGORİTMA ETKİNLİĞİ.

1.1 ÜST DÜZEY PROGRAMLAMA DİLLERİNDE YORDAMLAR ARASI VERİ-DENETİM İLETİŞİMİ.

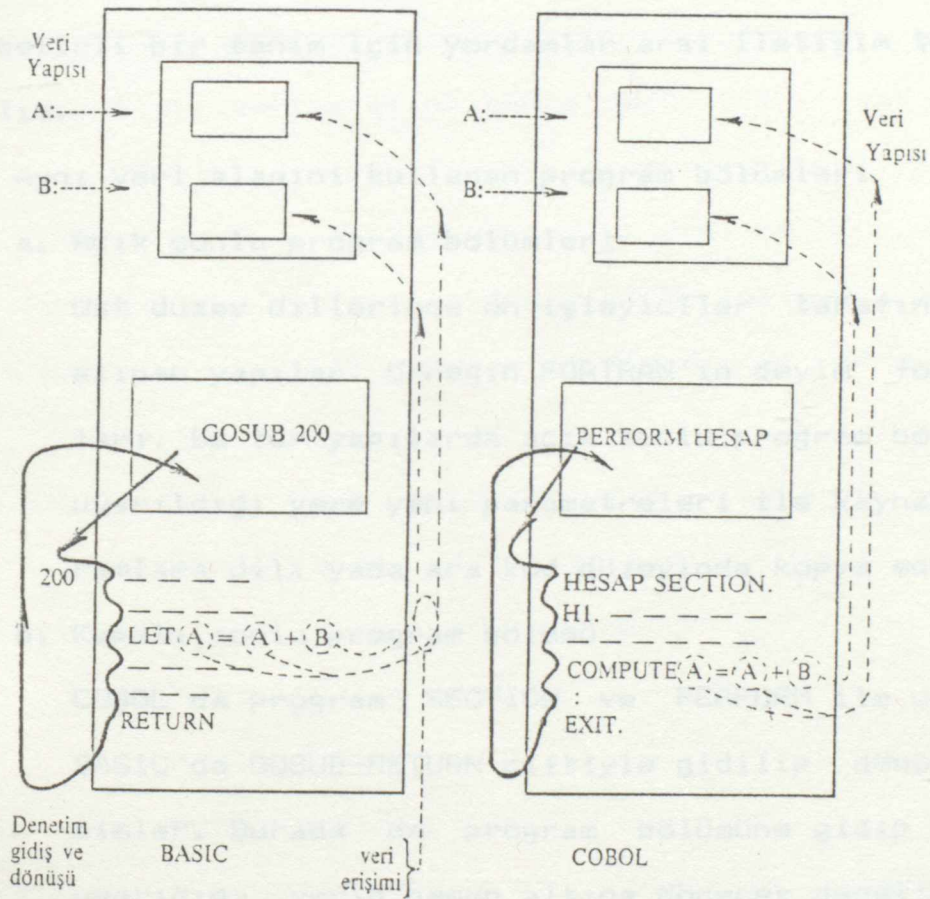
1.1.1 Yordamların Veri Alanı Sıra Düzeni.

COBOL, FORTRAN, BASIC, PL/1, PASCAL gibi gerçek programlama dillerinde yordamlar arası veri ve denetim iletişimi çok az farklılıklar gösterir. Bu konuda temel ayırım yordamın veri alanlarının sınıflanmasıyla başlar.

- a. Standart tanımında tek ve ortak bir veri alanı olan programlama dilleri (BASIC ve COBOL)
- b. İç içe yordam yapısı olmayan ancak her yordamın ayrı birer veri alanı olan programlama dilleri (FORTRAN, C)
- c. Her yordamın iki düzeyli olarak iç yapı ve dış yapı bilgilerine ulaşılabilirdiği veri yapısı soyutlaması (ANLAT algoritma dili)
- d. Yordam içinde yordam tanımlanabilen ve yordamın sıra düzeni (hiyerarşik) bir yapıda tüm atalarının veri yapılarına ulaşılabilirdiği yapıdaki programlama dilleri (ALGOL, PL/1 ve PASCAL)

Bu kavramlara kısaca göz atacak olursak genişletilmemiş standart BASIC dilinde yazılmış tüm programların tek bir veri yapısı olduğunu görürüz. BASIC programı iç kesimlerine gidilip geri

dönülse bile aynı veri alanını aynı mantıksal veri adlarını (değişken adlarını) kullanıyor durumdadır. Aynı durum COBOL dili için de geçerlidir. Programda bölümler kullanılmış ve bunlara uygula (perform) komutu ile gidiş dönüş yapılıyor da olsa COBOL programının veri yapısı tek ve kesiksiz bir yapıdadır. Öteki deyişle aynı değişken adına tüm program bölümleri kısıtlamasız ulaşabilirler.



Sekil - 1 COBOL ve BASIC'de Veri Erişim ve Değişken Adının Tanım Alanı

Öte yandan FORTRAN dilinde her bir yordamın kendi veri alanı vardır ve bir yordamın öteki yordamın veri alanındaki değişkene

erişimi alt yordam tanımla (SUBROUTINE DEFINATION) ve alt yordam çağır (CALL SUBROUTINE) komutlarında yapma argümanlar ile gerçek argümanların yazılış sırası yönünden eşleşmeleri ile yapılır. İki ayrı FORTRAN yordamında aynı değişkenler bulunsada dahi bunlar birbiriyle ilgisi olmayan değişken adlarıdır, çünkü ayrı veri alanları elemanlarıdır. Aralarında ilişki gerekiyorsa, ancak FORTRAN'ın COMMON komutu ile yada yapma-gerçek argüman arası veri iletişimi ile sağlanabilir.

Daha belirli bir tanım için yordamlar arası iletişim türlerini tekrarlıyalım.

i. Aynı veri alanını kullanan program bölümleri

a. Açık sonlu program bölümleri

Üst düzey dillerinde ön işleyiciler tarafından ele alınan yapılar. Örneğin FORTRAN'ın deyim fonksiyonları. Bu tür yapılarda açık sonlu program bölümü her uyarıldığı yere yeni parametreleri ile kaynak programlama dili yada ara kod düzeyinde kopya edilir.

b. Kapalı sonlu program bölümü

COBOL'da program SECTION ve PERFORM ile uyarılışı BASIC'da GOSUB-RETURN çiftiyle gidilip dönülen kesimler. Burada da program bölümüne gidip yordamın uyarıldığı yerin hemen altına dönecek denetim yapısı kurulmuştur, ancak ayrı bir veri alanı ve ayrı bir veri alanı ile veri iletişimi yoktur. Bu nedenle bu tür program kesimleri de yordam niteliği taşımazlar. Bu tür program bölümlerinin bellekte yalnızca bir

kopyası bulunur ve denetim yapısı bu bölüme gider ve geri döner.

ii. DIŞ YORDAMLAR

Herbir yordamın ayrı veri alanı vardır. Yordamlar arası hem denetim yapısı hemde veri iletişimi kurulmak zorundadır. Bu yordam içinde yordam tanımlanamaz, FORTRAN yordamları bu türdendir.

iii. SIRA DÜZEN İÇ YORDAMLAR

Yordamlar gerekli sırada iç içe kurulur. Bir yordam hem kendi veri alanına hemde sırasıyla içinde tanımlandığı tüm atalarının veri alanına erişebilir.

PASCAL ve PL/1'de dış yordamlar yanında sıra düzen iç yordamlarda kullanılmaktadır. ALGOL'da ise tüm programı sıra düzen iç yordamlar biçiminde kurmak alışkanlık olmuştur.

1.1.2 Dış Yordamlar Arası Veri İletişim Türleri

Dış yordamlar arası veri iletişiminde belli başlı üç ayrı iletişim yöntemi söz konusudur:

- i. Değer-Sonuç ile çağırma
- ii. Kaynak ile çağırma
- iii. Ad ile çağırma

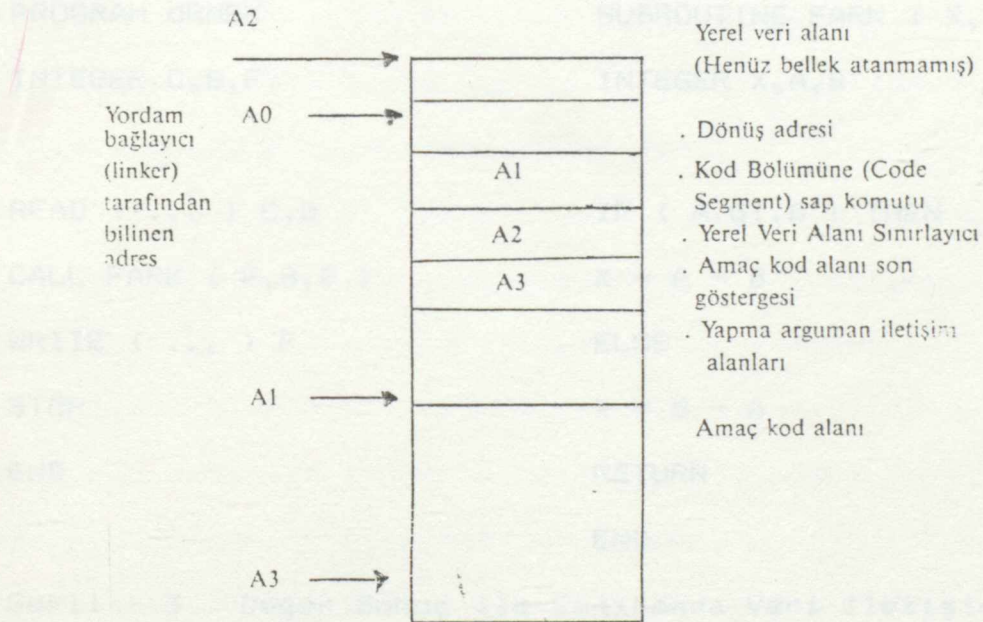
Yapma-Gerçek argümanlar arası veri iletişim yöntemlerinden üçüncüsü (Ad ile çağırma) yüksek düzeyde esneklik sağlamasına karşılık bilgisayarda uygulanma etkinliğinin düşük, programcı tarafından anlaşılmasının zor ve yan etkilerinin fazla ve gözden kaçabilir oluşu nedeniyle pratik hayatta hemen hemen hiç kullanılmaz.

maktadır. Şimdi diğer iki yöntemi açıklamadan önce yordamın veri iletişim denetim yapı iskeletinden söz edelim:

Üst düzey bir programlama dilindeki bir yordam derlenip çalışmaya hazır hale getirilirken bu yordam için derleyici iki parça üretir. Bu parçalar;

- i. Veri-İletişim-Denetim-Yapı iskeleti ve yerel veri alanı
- ii. Uygulanabilir (yada yer değiştirilerek uygulanabilir) amaç kod olarak alınır.

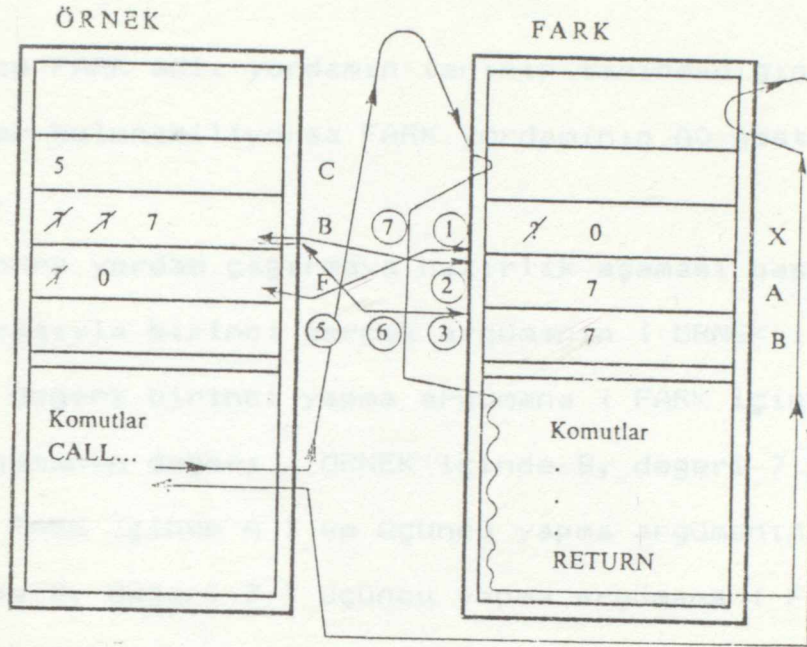
Aşağıda derlenerek makina koduna dönüştürülmüş bir dış yordam modellenmiştir.



Sekil - 2 Bir Veri İletişim Denetim Yapısı İskeleti Örneği

1.1.2.1 DEĞER-SONUÇ ile Çağırma Yöntemi ve Veri İletişimi

Düz çizgiler deger kopya etme ilişkisini gösterir, üzerindeki rakamlar işlem sırasıdır.



PROGRAM ÖRNEK

INTEGER C,B,F

READ (...) C,B

CALL FARK (F,B,B)

WRITE (...) F

STOP

END

SUBROUTINE FARK (X,A,B)

INTEGER X,A,B

IF (A.GT.B) THEN

X = A - B

ELSE

X = B - A

RETURN

END

Sekil - 3 Deger Sonuç ile Çağırma Veri İletişim Örneği

Varsayalımki yukarıdaki örnekte işlemler ÖRNEK programında oku (READ) komutunda C ve B ' ye 5 ve 7 degerleri aktarılarak başlasın uygulama ÖRNEK yordamı içinde CALL FARK komutuna geldiğinde sırasıyla şu işlemler yapılır.

i. Önce FARK adlı yordamın tanınıp tanınmadığına bakılır.

Eğer bulunabiliyorsa FARK yordamının A0 göstergesi hatırlanır.

ii. Sonra yordam çağırmaya hazırlık aşaması başlar. Bu aşamada sırasıyla birinci gerçek argümanın (ÖRNEK içinde F, değeri belirsiz) değeri birinci yapma argümana (FARK içinde X), ikinci gerçek argümanın değeri (ÖRNEK içinde B, değeri 7) ikinci yapma argümana (FARK içinde A) ve üçüncü yapma argümanın değeri (ÖRNEK içinde yine B, değeri 7) üçüncü yapma argümana (FARK içinde B) değer olarak kopya edilir.

iii. Daha sonra işlemler ÖRNEK yordamı içinden FARK yordamının ikinci komutundaki sapma komutu ile yönlendirilerek FARK yordamının uygulanır komutlarına ulaşılır.

iv. FARK yordamının komutlarının işleyişi sırasında gerekli değerler bu yordamın veri alanı içindedir. ÖRNEK yordamının veri alanına hiçbir dolaylı erişim yoktur. (Değer-Sonuç ile çağırmada)

v. Çağırılan yordamdan dönüş hazırlık aşamasında fark yordamı içinde oluşan değerler (yani X değişkeni içindeki 0 ve diğerleri) yapma-gerçek argüman eşleşmeleri nedemiyle çağırılan yordam olan ÖRNEK'in F,B ve B değişkenlerine sırasıyla taşınır.

vi. En son olarak FARK yordamı içinde dönüş adresi konumuna saklanmış olan adres alınıp çağırılan yordamda uygun konuma (çağırılan komutun hemen altı) dönüş yönlendirilir.

1.1.2.2 Kaynak ile Çağırma Yöntemi ile Veri İletişimi

PROGRAM ÖRNEK INTEGER FUNCTION ORTALA (K,ARR)

INTEGER DİZİ(40) INTEGER ARR(1),TOPL

TOPL = 0

DO 10 N = 1, K

READ (...) N TOPL = TOPL + ARR (N)

IF (N.LE.40) THEN 10 CONTINUE

AVG = ORTALA (N,DİZİ) ORTALA = REAL (TOPL) / K

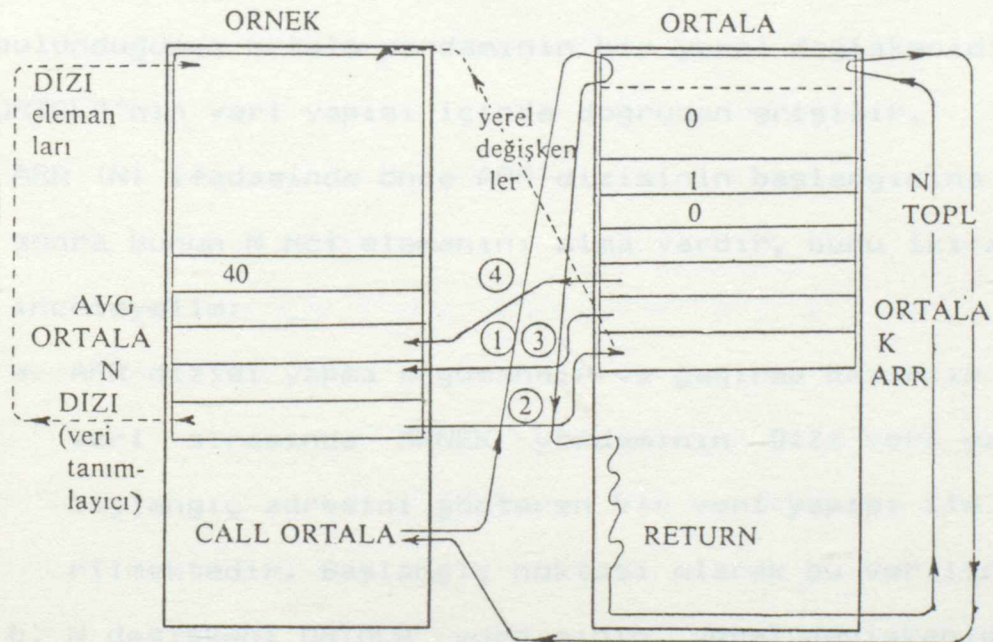
WRITE (...) AVG RETURN

ELSE END

END IF

STOP

Düz çizgiler değer kopya etmeyi
kesirli çizgiler gösterge ile
dolaylı erişimi ifade ederler.



Sekil - 4 Kaynak ile Çağırma Yöntemi ile Veri İletişim Örneği



Bu örneğimizde hem Deger-Sonuç ile çağırma hemde kaynak ile çağırma örneklenmiştir. Basit değişkenlerin veri iletişimini (genellikle Deger-Sonuç) biraz önce inceledik. Şimdi dizilerin yordamlar arası veri iletişimini inceleyelim. Bu birçok programlama dilinde kaynak ile çağırmadır. Bu yöntemde çağırılan program içinde gerçek argüman olan dizinin elemanları için fiziksel bellek yeri ayrılmaz ve dizi öğeleri çağırılan yordam içine kopya edilmez. Bunun yerine çağırılan yordamda, çağırma hazırlık işlemi sırasında gerçek argüman olan dizinin veri tanımlayıcısı karşılık gelen yapma argüman içine kopya edilir.

Örneğimizde ORTALA yordamının ikinci gerçek argümanı olan DİZİ tek boyutlu dizisinin veri tanımlayıcısını kopya etmiş biçimi görülmektedir. Bu noktadan sonra ORTALA yordamı içindeki

$TOPL = TOPL + ARR(N)$ komutunun nasıl işlediğini görelim.

- i. Ifadenin sağ tarafındaki TOPL değişkeni şimdi içinde bulunduğumuz ortalama yordamının bir yerel değişkenidir ve ORTALA'nın veri yapısı içinde doğrudan erişilir.
- ii. ARR (N) ifadesinde önce ARR dizisinin başlangıcına erişim sonra bunun N nci elemanını alma vardır, bunu iki aşamada inceleyelim:
 - a. ARR dizisi yapma argümandır ve çağırma hazırlık işlemleri sırasında ÖRNEK yordamının DİZİ veri yapısını başlangıç adresini gösteren bir veri yapısı ile gösterilmektedir. Başlangıç noktası olarak bu verilir.
 - b. N değişkeni ORTALA yordamının yerel değişkenidir ve ÖRNEK yordamındaki N değişkeninden ayrık bir yerdir.

Buradaki erişimimiz ORTALA içinde DO döngüsü indisi olarak kullanılan yerel değişkenedir.

iii. Bu biçimde bulunmuş olan TOPL + ARR(N) aritmetik ifadesinin değeri yerel değişken TOPL'a ya aktarılır.

Ortala yordamının işleyişinin sonlarına yaklaşırken döngünün sona erişinde $ORTALA = REAL (TOPL) / K$ işlemine ulaşıldığında işlem; ORTALA yordamı içindeki TOPL yerel değişkeni ve ORTALA yordamının birinci yapma argümanı K arasında yapılır. Oluşan sonuç ORTALA yordamı içindeki ORTALA adlı özel değişken alanına konur. Çağırılan yordamdan dönüş hazırlık kesimi RETURN komutuna ulaşıldığında şu işlemleri yapar:

- i. ORTALA içindeki K'nın değeri eşleşen yapma argüman olan ORNEK içindeki N'ye taşınır.
- ii. ORTALA yordamı alanındaki özel bilgi alanı ORTALA'nın içindeki değer ORNEK yordamı içindeki ORTALA değişkenine (fonksiyon adı özel biçimde ele alınır) taşınır.
- iii. ARR dizisi ve DIZI arasında hiçbir ek işleme gerek yoktur, çünkü veriler kaynak ile çağırma nedeniyle zaten DIZI içinde oluşmuştur.

Çağırma işlemini sonuçlandıran RETURN komutunun ele alınışındaki son kesim dönüş kesimidir. Bu aşamada çağırılan yordam içinde saklanmış olan dönüş adresi (Çağırılan yordamda çağırma yapan komutun hemen altındaki komutun adresi) alınarak çağırılan yordamın işletimi bitirilir ve çağırılan yordamda

$AVG = ORTALA (N, DIZI)$

komutuna dönlür. Bu aşamada dönüş yerini eşit işareti ile ORTALA

adı arası bir nokta olarak düşünebiliriz. Çünkü artık ORTALA değeri ORNEK yordamı içinde yerel bir değişken değeri gibi olmuştur ve bu bölüm için yerel değişken olan AVG'ye aktarılabilir.

1.2 TÜRKÇE ALGORİTMA DİLİ (ANLAT) TANIMI

Algoritma dili genellikle doğal dile benzemekle birlikte bazı kalıplara uyan ve tasarımcıyı giriş çıkış işlemlerinin ayrıntıları ile uğraştırmayan bir bilgisayar programı tanımlama yöntemidir. Bilişimciler arasında bir algoritma diline duyulan gereksinme ALGOL programlama dilinin tasarım çalışmaları sırasında duyulmuş, ALGOL'ün tasarım amaçları arasında buda belirtilmiştir. Ancak ne ALGOL nede başka bir dil Türkiye'de genel bir iletişim aracı olamamıştır. ANLAT algoritma dili 1982 sonbaharında Ümit Karakaş ve Mehmet Aksayan tarafından bu alandaki boşluğu doldurmak amacıyla tasarlanmıştır. Hacettepe Üniversitesinde başarı ile uygulanmaktadır.

Tasarımın iki temel amacı şöyle belirlenebilir.

- i. Yazılım belgelemede program bölümleri arası ilişki çizimi ile üst düzey programlama dillerindeki programlar arasında bir belgeleme aracı olarak,
- ii. Veri yapıları ve algoritmalar derslerinde algoritmanın kısa ve belirli bir biçimde yazılması için.

1.2.1. Algoritma İletişimi İçin Seçenekler Nelerdir?

Birinci seçenek, algoritma üzerine konuşacak yada algoritmayı farklı zamanlarda inceleyecek bilişimcilerin doğal dilini (bizim

için Türkçe) sınırlama ve ek biçimler / veri yapıları olmaksızın kullanmaktır. Bu yaklaşımda kullanılan dilin düzeyi öteki seçeneklere göre en yüksek olduğundan anlatım kısa olacak fakat algoritmanın önemli özelliklerinden belirlilik garanti edilemeyecek, algoritma karmaşıklığı ve algoritma etkinliği düzenli bir biçimde incelenip belirlenemeyecektir.

İkinci seçenek İngilizce eğitim yapan birçok üniversitede veri yapıları ders kitabı olarak kullanılmış olan Horowitz-Sahni'nin SPARKS algoritma dili kullanılmaktadır. Bu seçenek biri evrensel, diğeri bize özel iki sakınca içermektedir. Evrensel sakıncası SPARKS'ta grup değişken yapısı bulunmadığından, her algoritmanın yanında bir sayfa çizim ve yazılı açıklamalar taşıma gereğidir. Bize özel sakınca ise İngilizce anahtar sözcüklü yapısı nedeniyle Türkiye'nin bilgi işlem merkezlerine kullanımı pratikte yaygınlaşan bir dil olmamakta yalnızca İngilizce hazırlık okulu bulunan bilgisayar mühendisliği bölümlerinde öğretim amacıyla kullanılabilmektedir.

Üçüncü seçenek bir programlama dilinin (PD) algoritma diliymişçesine ikinci amacı gerçekleştirmek için kullanmaktır. Bu durumda birinci amaç tümüyle ortadan kalkmaktadır. PD bağımlılıkları yanında bilgisayar mimarisine eldeki bilgisayarın görünümüne işletim sistemine bağımlılıklar artmaktadır. Ancak öğretim bazı kavramlarda çok belirgin yapılabildiği için bu yaklaşımı benimseyen PASCAL yada PL/1 tabanlı veri yapıları kitapları çoğalmaktadır.

Dördüncü seçenek, veri yapıları ve algoritmalar alanında bir anıt eser niteliği taşıyan Knuth' un ' THE ART OF PROGRAMMING ' dizisindeki kavramsal birleştirici dili (MIX) dir. Gelişen üst düzey programlama dilleri nedeniyle veri yapıları ve algoritmalar öğretiminde bu yaklaşım önemini yitirmiştir.

1.2.2. ANLAT 'ın Belgeleme Amacıyla Kullanımı

Yazılım geliştirmede önemli yöntemlerden biri " Adım adım çözümleme " yöntemidir. Bu yöntem düzenli ve güvenilir bir programı geliştirmek için anlat algoritma dili ile iç içe kullanılmalıdır. Bunu sağlamak için aşağıdaki kurallara uyulması tavsiye edilir.

KURAL 1 : Adım adım çözümlemenizi yazılımınızı kendi bilgi düzeyinizde kişilerin okuyabileceğini düşünerek kesmeyiniz. Bilgisayar dalında öğrenim yılı yada iş deneyimi sizden bir yıl daha az kişiler için belgeleyiniz. Eger daha alt düzeyde bilgi / deneyimli kişileri düşünerek belgeleme yaparsanız bu belgeleme değil ders kitabı olacaktır. Eger yalnızca kendinizin o andaki düzeyiniz için çözümler ve yetersiz belgellerseniz iki sakıncası vardır. Birincisi bir süre sonra eski düşünce sürecinizi yada kuramsal bilginizi unutursunuz. İkincisi bilgi işlem merkezlerinde işler genellikle eş bilgi düzeyindeki kişilerle değil, bir alt düzeyde bilgi/deneyimi olan kişilere devredilir. Eger iyi belgeleme yaparsanız eski programınızın işletmenliğinden kurtulup yeni atılımlar yapamayabilirsiniz.

KURAL 2 : Öbegin içeriği tasarımcı, algoritmanın bu günkü

okuyucusu ve gelecekteki okuyucuları için hiçbir yoruma açık olmayacak biçimde kesin ise o öbeğin adım adım çözümlenmesi ANLAT algoritma dili ile algoritma yazılmadan sonlanabilir.

" NOTLAR vektöründeki N adet değerin ortalamasını bul, ORTA'ya aktar."

" NOTLAR vektöründeki N adet değeri kabarcıksıralama yöntemi ile artan sıraya koy."

" ORTA := ORTALA (N,NOTLAR) "

" KABARCIKSIRALA (N,NOTLAR) "

Yukarıdaki örneklerden ilk ikisi başlangıç düzeyi öğrencisinin ; diğer ikisi veri yapıları konusunu bilen kişinin daha sonra daha alt düzeyde bilgili bir kişinin programlarını okuyacağını planladığı durumda adım adım çözümlemeyi kesebileceği düzeylerdir.

KURAL 3 : Adım adım çözümlemede oluşan diğer öbekleri programa dönüştürürken veri türünün ondalık basamak duyarlılığı dışında hiçbir belirsizliğe neden olmayacak biçimde anlat dilinde yazınız.

1.2.3. ANLAT Algoritma Dili Komut Yapıları

Bu çalışmanın okuyucusunun program okuma yeteneği kazanmış olduğu düşünülerek önce bir örnek verip, ardından giderek ayrıntılara inecek biçimde ANLAT algoritma dilini açıklayalım. Aşağıda bir DİZİ üzerinde yerleşmiş N adet sayının ENBÜYÜK DEĞERİNİ bulan bir yordam yer almaktadır.

```

1  BÜYÜK (N, DİZİ) YORDAMI ;
    /* BU YORDAM DİZİ ADLI DİZİNLİ DEĞİŞKENİN */
    /* İÇİNDEKİ EN BÜYÜK DEĞERİ BULUR */
2  DİŞ YAPILAR ;
3  N TAMSAYI ;
4  DİZİ (*) TUR ( TAMSAYI, KESİRLİ ) ;
5  BÜYÜK TUR ( DİZİ ) ;
    /* DİZİ YA TAMSAYILI YADA KESİRLİDİR */
    /* BÜYÜK ADLI DEĞİŞKEN DİZİ İLE AYNI TÜRDEDİR */
6  İÇ YAPILAR ;
7      I DİZİ ÜZERİNDE GÖSTERGE, SINIR ( 1:N ) ;
8  KOMUTLAR ;
9      BÜYÜK := DİZİ ( 1 ) ;
10     SAYARAK YİNELE ( I:= 2, I>N, I:=I+1 ) ;
11         [ EĞER DİZİ ( I ) > BÜYÜK
12             İSE BÜYÜK := DİZİ ( I )
13         ]
14  BÜYÜK BİTTİ ;

```

1.2.3.1. Yordamın Nitelikleri

ANLAT dilinde yordamlar ayrı ayrı tanımlanır. Bir yordam içinde başka bir yordam tanımlanamaz.

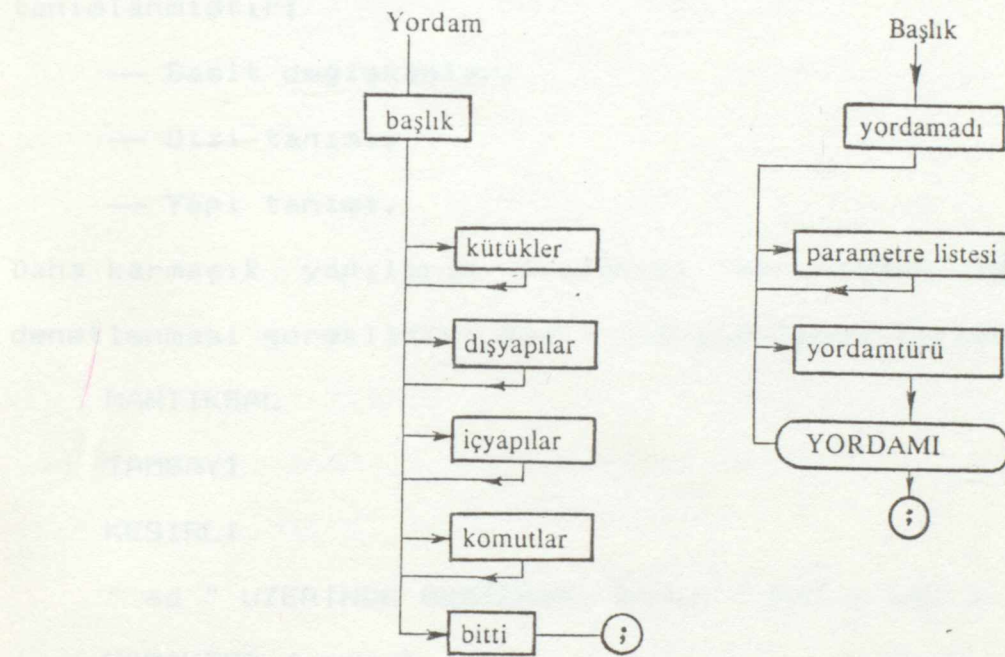
ANLAT yordamları özyineli yapıdadır öteki deyişle bir yordam doğrudan yada dolaylı olarak kendini çağırabilir.

ANLAT yordamı adı içinde daima bir değer döndürecek yapıdadır. Yordamın adı için bir tür tanımı yapılmamışsa döndürülen

değer MANTIKSAL kabul edilir.

ANLAT yordamları arasında veri iletişimi basit değişkenler için değer sonuç ile çağırma dizinli değişkenler için kaynak ile çağırma. Bu kural özyineli yordam uyarılışı başka bir iletişim yöntemine zorlanmadığı sürece geçerlidir.

Aşağıda ANLAT yordamının bazı kesimleri tanımlanmıştır.



Sekil - 5 ANLAT Algoritma Dili Yapısı

1.2.3.2. Yordamın Uyarılması

Yordam, adı ve parantez içinde tanım sırasını uygun biçimde verilmiş gerçek argümanlar listesiyle uyarılır.

ANLAT dilinde yazılmış bir yordam genellikle yordamın tanımlanmış işlevinimi yaptığını yoksa bir yanlış koşulumu döndürdüğünü sınavan bir koşul komutu içinde uyarılır.

Örneğin ; EGER MATRİSTOPLA (M,N,K,L,A,B,C)

ISE MATRİSYAZ (M,N,C)

DEĞİLSE ['MATRİS TOPLAMA YAPILAMADI' YAZ ; DÖN]

KATSAYI (1 : 13 , 1 : 9) TAMSAYI ;

1.2.3.3. Yordamın İç Veri Yapıları

ANLAT algoritma dilinde temel düzeyde üç tür veri yapısı tanımlanmıştır;

— Basit değişkenler,

— Dizi tanımı,

— Yapı tanımı,

Daha karmaşık yapıların kullanıcı tarafından tanımlanması ve denetlenmesi gereklidir. Basit değişkenlerin türleri şunlardır ;

MANTIKSAL

TAMSAYI

KESİRLİ

" ad " ÜZERİNDE GÖSTERGE, SINIR (ALT : ÜST)

KARAKTER (nnn)

Basit değişkenlerin tanımı değişken listesini izleyen " tür " anahtar sözcüğü ile yapılır.

Örnek ; A, B, C, TAMSAYI ;

DEGER, TOPLAM KESİRLİ ;

Dizi tanımı iç yapılar arasında tanımlandığında tüm boyutları tam sayılar ile belirtilmelidir. Bir dizinin her boyutunda iki nokta üst üste ile ayrılmış biçimde o boyutun alt ve üst sınırları verilebilir. Alt sınır verilmezse 1 olarak varsayılır.

Aynı türden basit ve dizinli değişkenler bir liste biçiminde yazılabilirler.

Örnek : ARTIŞ (1980 : 1987) KESİRLİ ;

KATSAYI (1 : 15 , 1 : 9) TAMSAYI ;

TEST, İLİSKİ (4) MANTIKSAL ;

ANLAT dilinde yapı tanımı birbirleriyle anlamsal yönden, kullanım yönünden ilişkili farklı türden verileri bir grup olarak birleştirir. Yapının kaç kez yineleneceği dizinli değişken tanımı gibi verilir.

Örnek : EKONOMİ (1980 : 1987) (TOPTANESYA, TÜKETİCİİNDİSİ

ÜCRETLER KESİRLİ) YAPI ;

Yapı tanımı içinde bir başka yapı tanımı da yer alabilir. Yapı tanımı ile belirtilmiş bir grup değişkene erişebilmek için tüm tanımın doğru indisler kullanılarak belirtilmesi gereklidir.

Örnek : YIL := 1980

T := EKONOMİ (YIL). TOPTANESYA

Eksik tanımı ile yada indisi tanımladığı yerden farklı yerde kullanılarak değişkene erişim yapılması bazı programlama dillerinde serbest iken ANLAT'ta amaçlanarak yasaklanmıştır. Aşağıdaki kullanımlar ANLAT dili için doğru yazım biçimi değildir.

TOPTANESYA (1985)

EKONOMİ . TOPTANESYA (1985)

Bir yordamın iç yapıları arasında tanımlanmış bir değişken o yazılım grubundaki hemen hemen tüm yordamlarda kullanılacak olabilir. Bu yordamda o değişkenin / yapının tanımının arkasına EVRENSEL sözcüğü eklenir. Bu durumda yordamlardan bu veri yapısına erişim

için hiçbir tanım yapmaya gerek yoktur. Bu veri yapısı içinde tüm değişkenler aynı ad ve tanımlarla , tanınmak üzere o uygulama grubundaki her yordama açıktır.

- 1 VERİLER YORDAMI ;
/* BU YORDAM YALNIZCA KAYNAK YAPISINI İÇERİR */
- 2 İÇYAPILAR ;
- 3 KAYNAK (200) (ADSOYAD KARAKTER (20),
VERGİ KESİRLİ,
BAĞ KAYNAK ÜZERİNDE GÖSTERGE ,
SINIR (1 : 200)) YAPI, EVRENSEL ;
- 4 SERBEST KAYNAK ÜZERİNDE GÖSTERGE, SINIR (1 : 200), EV-
RENSEL;
/* EVRENSEL SÖZCÜĞÜ İLE BURADA TANIMLANANIN BAŞKA YERDE
KULLANILACAGI BELİRTİMEKTEDİR */
- 5 VERİLER BİTTİ ;

1.2.3.4. Yordamın Dış Yapıları

Bir yordamın dış ortam ile iletişim yaptığı değişkenin türü bazen yordamın belirttiği algoritma açısından önemsizdir. Örneğin; kabarcık sıralama yönteminde sıralanan sayıların tamsayı yada kesirli olmalarının algoritma yönünden hiçbir önemi yoktur. Bu değişkenler için tür tanımı yerine muhtemel tür listesi yer alır.

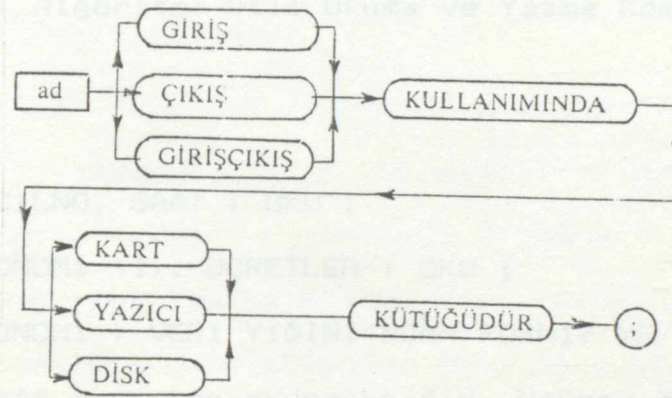
Örnek ; DİZİ (*) TÜR (TAMSAYI, KESİRLİ, KARAKTER (*)) ;

Yordamlarda dış ortam ile iletişim yapan değişkenlerden

bazıları yalnızca yordama giriş bilgisi taşımak, bazıları yalnızca yordamdan dış ortama çıkış bilgisi götürmek için kullanılır. Bir kesimide üzerinde değişiklikler yapılacak biçimdedir. Bu grup giriş-çıkış türü olarak belirtilir.

1.2.3.5. Kütük Tanımları

ANLAT dilinin bu tanımında (ANLAT/ VERİ YAPILARI) kütük tanımları şöyledir.



Sekil - 6 ANLAT Algoritma Dili Kütük Yapıları

Kütüğün sona eriş koşulu VERİBİTTİ (ad) mantıksal yordamını kullanarak bir koşul komutu ile sınanır.

Örnek :

FAZLAMESAİ GİRİŞ KULLANIMINDA KART KÜTÜĞÜDÜR ;

LISTE ÇIKIŞ KULLANIMINDA YAZICI KÜTÜĞÜDÜR ;

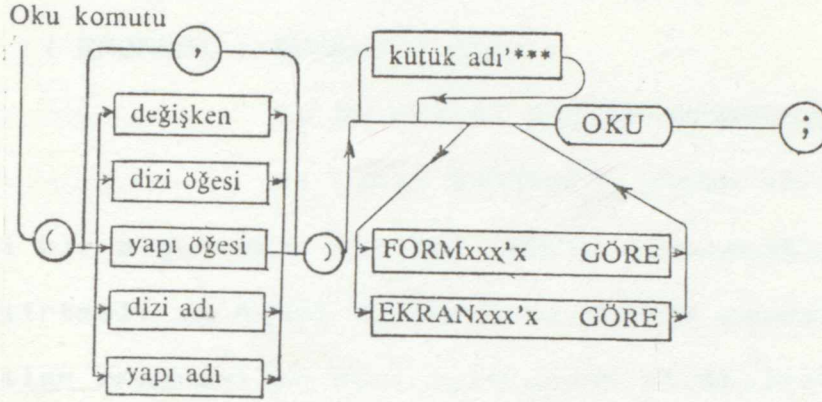
KADROLAR GİRİŞ KULLANIMINDA DISK KÜTÜĞÜDÜR ;

EGER VERİ BİTTİ (FAZLAMESAİ)

İSE

1.2.3.6. Okuma ve Yazma Komutları

ANLAT dilinde Oku ve Yaz komutlarının yapıları şöyledir



Sekil - 7 ANLAT Algoritma Dili Okuma ve Yazma Komutları

Örnek :

(SICILNO, SAAT) OKU ;

(EKONOMİ (I). ÜCRETLER) OKU ;

(EKONOMİ) VERİ YIGINI'NDAN FORM17'YE GÖRE OKU ;

İlk örnekteki basit değişken okunmaktadır. Üçüncü örnekte EKONOMİ adlı 1980' den 1987' ye kadar yineleyen her bir öyesi üç basit değişkenden oluşan yapıya 8*3 değer okunması istenmektedir.

Yapılmış olan sistem çözümleme çalışması ile ilgili bağlantılar FORMxxx ve EKTRANxxx sözcükleriyle kurulmakta, algoritma, veri giriş çıkışın ayrıntılarından soyutlanmaktadır.

Yaz komutu, değişmezlerinde yazılabilmesi ve ekrana yazışta satır sütun belirtilebilmesi dışında Oku komutuna yapı yönünden benzemektedir.

Örnek :

```
( SICILNO, SAAT ) YAZ ;
( SICILNO, SAAT ) ( SA = 10 , SU = 1,9 ) YAZ ;
( EKONOMI ) EKRAN8'E GÖRE ;
```

```
/* BU KESİMİ SİSTEM TASARIMINDAKİ */
```

```
/* ÇIKIŞ BİÇİMİNDE UYGUN YAZINIZ */
```

ANLAT dili bir algoritma dilidir. Neyin okunacağını ve yazılacağını belirtmeli ve nasıl okunacağını sistem çözümleme çalışması ve kullanılan programlama dili için esnek bırakılmalıdır.

1.2.3.7. Aritmetik Deyimler

ANLAT dili için gösterge türü değişken tamsayıların bir alt kümesidir. Aritmetik deyim içinde yer alabilir. Mantıksal degerde deyim içinde tamsayı gibi (0 = yanlış, 1 = doğru) kullanılabilir.

ANLAT dilinde üs alma işleci ** dır. Çarpı ve bölü işareti, artı ve eksiye göre önceliklidir. Aynı öncelikteki üs işaretinin önceliği mutlaka parantez çiftleri ile belirtilir. Bu önlem kullanıcının farklı programlama dillerindeki birbiri ile çelişen alışkanlıkları nedeniyle önerilmiştir.

Kullanıcı aktarma işlecini := yada <- olarak kullanabilir.

1.2.3.8. Mantıksal Deyimler

Mantıksal deyim .K., .KE., .B., .BE., =, ≠, >, >, <, <, >=, <=, 'den oluşan ilişki işlemleri ile öncelik sırasına göre DEĞİL, VE, YADA mantıksal işlemleri ile aritmetik ve karakter türü deyimler arasında kurulur.

Örnek :

ANLAT algoritmalarında ve yeni üst düzey dillerde denetli yapı (A > 0 VE (B = 1 YADA B = 2)) DEĞİL

1.2.3.9. Koşul Komutu

EGER 'koşul'

ISE 'komut';

EGER 'koşul'

ISE 'komut1'

DEĞİLSE 'komut2';

Koşul komutunun iki muhtemel yapısı yukarıda verilmiştir. Komut koşul komutu dışındaki tek bir komut olabileceği gibi BAŞLA... BİTTİ [] çiftleri arasında herhangi bir komut dizisinden oluşan birleşik komut da olabilir. ANLAT'ta koşul komutuna koşul birleştirme yazılım güvenilirliğini tehlikeye düşürdüğü için amaçlanarak yasaklanmış, bu durum ancak birleşik komut içinde sağlanmıştır.

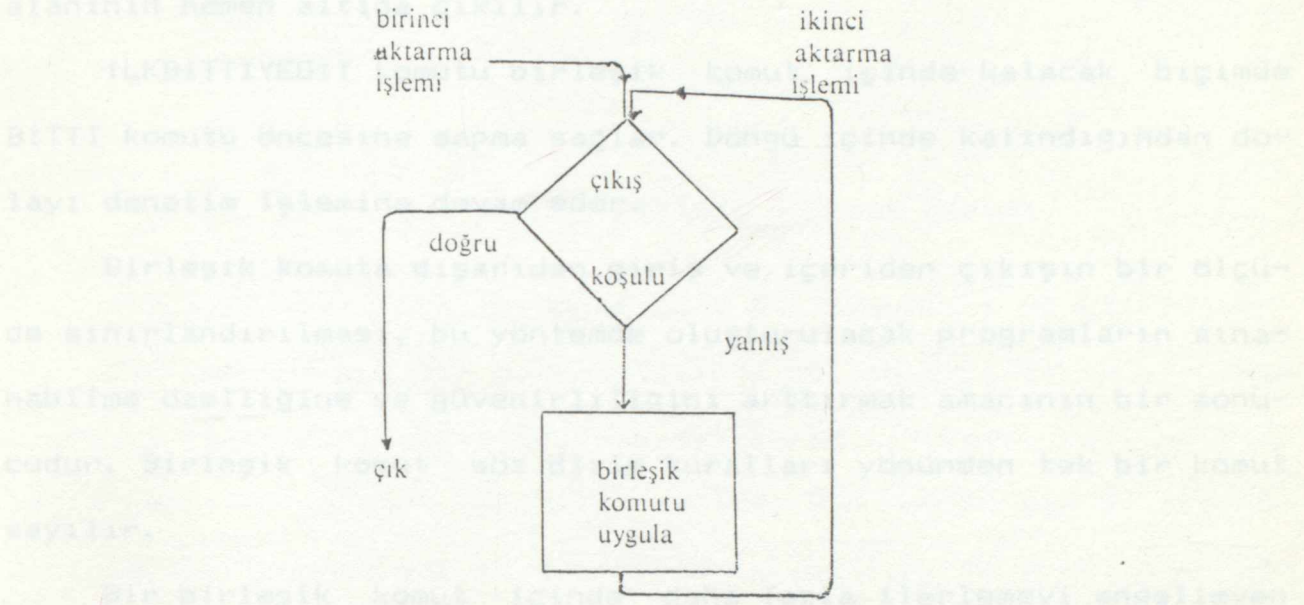
1.2.3.10. Yineleme Komutları

Yineleme komutu bir denetim kesimi ve bunu izleyen bir birleşik komuttan oluşur. Denetim kesimi yapılarından biri şöyledir :

SAYARAK YİNELE (Birinci aktarma işlemi, çıkış koşulu, ikinci aktarma işlemi);

İlk aktarma işlemi başlangıç değerine aktarmak için kullanılır. Sonraki aktarma işlemi bir sonraki yineleme başlamadan önce yapılacak değişkenliği gösterir. ANLAT dilinde çıkış koşulunun sağlanıp sağlanmadığı döngü alanına girmeden önce denetlenir. Gerçekleştirmenin yapılacağı programlama dili bu özelliği sağlamıyor ise algoritmalarda ayrıca önlem alınmalıdır.

Aşağıda ANLAT algoritmalarında ve yeni üst düzey dillerdeki denetim yapısı görülmektedir.



Şekil - 8 ANLAT Algoritma Dili Döngü Yapısı

Örnek :

SAYARAK YINELE (Q := L, Q = 0, Q := D (Q). BAĞ);

Öteki yineleme komutu denetim yapıları şunlardır :

- ÖNCE YINELE, koşul DOĞRUYSA ÇIK ;
- ÖNCE YINELE, koşul YANLISSA ÇIK ;
- ÖNCE koşul SINA, DOĞRUYSA YINELE ;
- ÖNCE koşul SINA, YANLISSA YINELE ;

1.2.3.11. Birleşik Komut ve Denetimli İç Sapma

Birleşik komut; BAŞLA ve BİTTİ, yada [] çiftleri arasında bir dizi komuttur. Birleşik komut içindeki bir komuta birleşik komutun dışından sapma yapılamaz. Birleşik komutun içinden de dışardaki

bir noktaya SAP'ma yapılmaması önerilir. İLKBITTİYİAS komutu birleşik komutun BITTİ komutunu izleyen komuta ulaşmayı sağlar, döngü alanının hemen altına çıkılır.

İLKBITTİYEGİT komutu birleşik komut içinde kalacak biçimde BITTİ komutu öncesine sapma sağlar. Döngü içinde kalındığından dolayı denetim işlemine devam eder.

Birleşik komuta dışarıdan giriş ve içeriden çıkışın bir ölçüde sınırlandırılması, bu yöntemde oluşturulacak programların sınırlanabilme özelliğine ve güvenilirliğini arttırmak amacının bir sonucudur. Birleşik komut söz dizim kuralları yönünden tek bir komut sayılır.

Bir birleşik komut içinde daha fazla ilerlemeyi engelleyen bir koşul oluşuyorsa birleşik komutun dışına SAP'ma komutu ile çıkıp durumu orada ele almak yerine ya birleşik komut içinde yerel olarak ele alınır yada bir hata yordamı çağırarak durum kontrol altına alınır.

DDN komutu yordamın her yerinde birleşik komut içinde de kullanılabilir.

1.2.3.12. Seçenek Yapısı

Bir komut ikiden fazla seçenekten yalnız biri oluşabilecek ve yalnızca bu koşula ilişkin komutlar uygulanacaksa kullanılır.

(ŞEKİL-9' da ANLAT dili seçenek yapısı görülmektedir.)

1.2.3.15. Hazır işlevler, Programlama Dili ile ilişkilendirme ANLAT dili sistem çözümleme aşaması ile programlama arasındaki boşluğu doldurmak amacıyla geliştirilmiştir. Bir ucunda programlama diline ulaşmalıdır. Bir programlama dilinin ANLAT dilinde karşılığı olmayan bir yapısı (Örneğin COBOL'un EXAMINE komutu), bir hazır yordam yada işletim için kullanıcı desteği istenirse bu durum algoritma içinde açıkça belirlenmelidir. Bu belgelemede programlama diline bağlı açıklamaların /*!, kullanılan bilgisayara/işletim sistemine bağımlı olanların /*!!, ile başlayan açıklamalarla verilmesi alışkanlık edilmelidir. Kullanılan bilgisayarın donanım (Yapı) özelliklerine bağlı durumlarda /* ile başlayan açıklamalarla belgelenmelidir.

Örnek :

```

11      BÜYÜK := -99999 /*!! BU BİLGİSAYARDA TANIMLANABİLECEK */
12
13      /*!! EN KÜÇÜK TAMSAYI */
14
15      EĞER I/2*I = I          /*! FORTRAN DİLİNDE */
16
17      İSE ( 'SAYI ÇİFT' ) YAZ ; /*! TAMSAYI BÖLME ÖZELLİĞİ */

```

1.2.4. Bir Örnek Algoritma

```

1  MATRİSTOPLA ( N,K,L,A,B,C ) MANTIKSAL YORDAMI ;
   /* BU YORDAM NxM BOYUTUNDAKİ A MATRİSİ */
   /* İLE KxL BOYUTUNDAKİ B MATRİSİNİ */
   /* KOŞULLAR TANIMA UYGUNSA TOPLAR VE */
   /* NxM BOYUTLU C MATRİSİNİ OLUŞTURUR */
2  DIŞYAPILAR ;

```



```

1  A ( *,* ) , B ( *,* ) TÜR ( TAMSAYI, KESİRLİ ) , GİRİŞ ;
2  C ( *,* ) TÜR ( A ) , ÇIKIŞ ;
3  N,M,K,L TAMSAYI, GİRİŞ ;
4  İÇYAPILAR ;
5  I A ÜZERİNDE GÖSTERGE , SINIR ( 1 : N ) ;
6  J A ÜZERİNDE GÖSTERGE , SINIR ( 1 : M ) ;
7  KOMUTLAR ;
8  EĞER N ≠ K YADA M ≠ L
9  İSE MATRİSTOPLA := YALNIS
10 DEĞİLSE BAŞLA
11 MATRİSTOPLA := DOĞRU ;
12 SAYARAK YİNELE ( I := 1, I > N, I := I+1 ) ;
13 [ SAYARAK YİNELE ( J := 1, J > M, J := J+1 ) ;
14 [ C ( I, J ) := A ( I, J ) + B ( I, J ) ]
15 ]
16 BİTTİ ;
17 MATRİSTOPLA BİTTİ ;

```

Yukarıdaki algoritma veri yapıları ve algoritmalar düzeyinin aşıldığı kabul edildiğinde şu şekilde yazılabilirdi ;

```

MATRİSTOPLA ( N,M,K,L,A,B,C )

```

```

/* A NxM , B KxL BOYUTLUDUR, BOYUT UYUSMAZLIĞINDA */

```

```

/* MATRİSTOPLA YALNIŞTIR AKSI HALDE DOĞRUDUR */

```

Fakat bu algoritmanın işletim etkinliği düşüktür algoritma etkinliği konusu ileride ele alınacaktır.

1.3. ALGORİTMA KARMAŞIKLIĞI VE ALGORİTMA ETKİNLİĞİ TAHMİNLERİ

Bu bölümde aynı sorunu çözen algoritmaları karşılaştırabilmek için gerekli kriter tabanı oluşturmak için algoritma karmaşıklığı ve algoritma etkinliği kavramları incelenecektir.

1.3.1. Algoritma Karmaşıklığı

Algoritma karmaşıklığı bir sorunu çözebilecek algoritmanın, giriş verilerinin o işletimde ençok işlemi gerektirecek (en olumsuz) dağılımı gösterdiği varsayılarak, işlemler için üst sınır fonksiyonu bulmaktır. Algoritma karmaşıklığının incelenmesinin temel nedeni n1 adet giriş verisi için T1 adet işlem gerektiren bir algoritmanın n2 adet giriş verisi için ne kadar işlem gerektireceğini tahmin edebilmektir.

Önce sorunu kolay bir biçimde ele alalım :

Varsayalımki 20x20'lik A ve B matrisleri olsun, bunların toplamını bulan ve C matrisini oluşturan bir yordam bu işlemi 1 sn'de yapsın

Şimdi varsayalımki aynı hazır yordam ile 100x100 boyutunda C ve D matrislerinin toplamı E matrisini oluşturan bir işlem yapmak isteyelim.

Buna göre i. ilk işletimde $20 \times 20 = 400$ kez A ve B ögesi toplanıp C ögesine aktarılıyordu. Buna göre bir C ögesi için 1/400 sn'ye gerekiyordu.

ii. İkinci işletimde $100 \times 100 = 10.000$ adet C ögesi bulunacak ve bu yaklaşık $10.000 * (1/400) = 25$ sn'ye olacaktır.

Simdi bu tür tahminlere daha karmaşık birçok algoritma için yapabilmek amacıyla kuramsal tanımları verelim.

TANIM : Bir algoritmanın giriş verileri sayısı n , işletim zamanı $f(n)$ ile gösterildiğinde

$$|f(n)| < c \cdot |g(n)| \quad c < 0, n > n_0 > 0$$

için sağlanıyorsa algoritma karmaşıklığı tanımlıdır ve $c.O(g(n))$ biçiminde gösterilir. Bu tanıma kolay bir örnek verelim.

Bu örnek FIBONACCI dizisi olarak bilinen ilk iki rakamı 0 ve 1 diğerleri kendinden önce gelen iki rakamın toplamına eşit diziyi oluşturan algoritmadır.

```

1  FIBONACCI (N) MANTIKSAL YORDAMI ;
2  DIŞYAPILAR ;
3    N TAMSAYI ;
4  İÇYAPILAR ;
5    SAYI, ÖNCE, DAHAÖNCE TAMSAYI ;
6    SAYAC TAMSAYI ;
7  KOMUTLAR ;
8    EĞER N . KE. 1
9      İSE [ FIBONACCI := YANLIŞ ; DÖN ]
10     DEĞİLSE BAŞLA
11       FIBONACCI := DOĞRU ;
12       DAHAÖNCE := 0 ; ÖNCE := 1 ;
13       ( DAHAÖNCE ) YAZ ;
14       ( ÖNCE ) YAZ ;
15     SAYARAK YİNELE ( SAYAC := 2, SAYAC > N, SAYAC := SAYAC+1);

```

```

16      [ SAYI := ÖNCE + DAHAÖNCE ;
17      ( SAYI ) YAZ ;
18      DAHAÖNCE := ÖNCE ;
19      ÖNCE := SAYI ]
20      BİTTİ ;
21      FIBONACCI BİTTİ ;

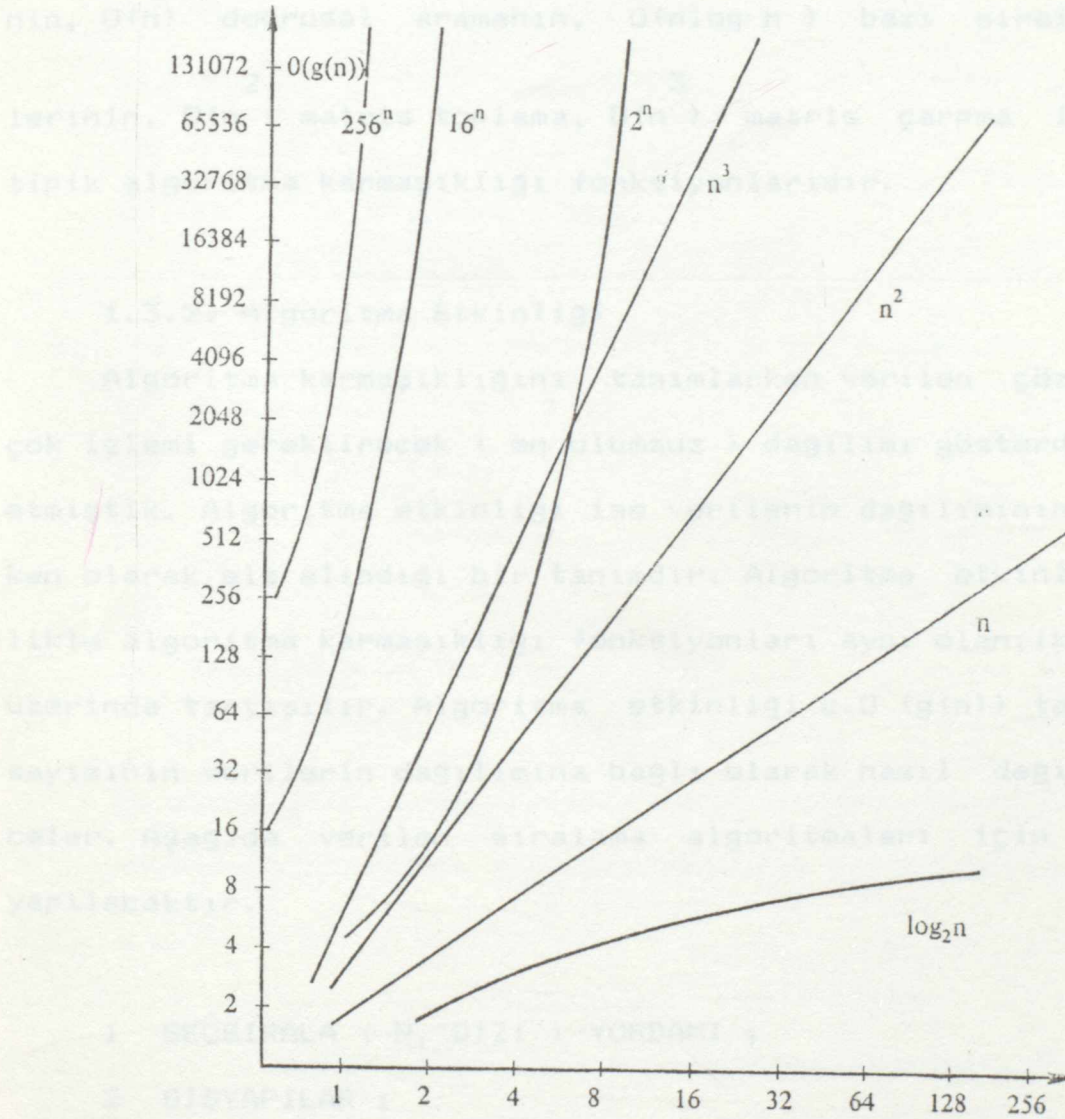
```

Bu algoritmada giriş bilgisi N'nin değeri arttıkça 16-19'ncu satırlar arasındaki işlem grubunun tekrarlanma sayısı da artmaktadır. Bu ilişki doğrusal (linear) dır, öteki deyişle N ilk işle-timine göre 3 kez artmışsa yineleme sayısında 3 kez artmaktadır. Algoritma karmaşıklığında N büyürken olacakları incelediğimizde döngü denetiminin 2'den başlaması önemsenmemekte ve bu algoritma-nın karmaşıklığı $c.O(n)$, $c = 1$ olarak belirtilmektedir. Algoritma karmaşıklığı incelenirken $g(n)$ fonksiyonu bazı standart fonksiyon-lara dönüştürülür (yaklaştırılır yada üstten bağlanır).

Örneğin :

$$g(n) = \frac{1}{2} \cdot n^2 + 3n$$

Fonksiyonu elde edildiyse n büyük değerlere ulaşırken n^2 ya-nında $3n$ önemsenmez ve algoritma karmaşıklığı $c.O(n^2)$, $c = 0.5$ biçiminde yazılır. Sık kullanılan algoritma karmaşıklığı fonksi-yonları aşağıda verilmiştir.



Sekil - 10 Algoritma Karmaşıklığı Fonksiyonları Grafik Gösterimi

Herbiri karşılaştırma amacıyla $c=1$ için aynı çizim üzerinde gösterilmiştir. Bu algoritma karmaşıklığı fonksiyonlarından $O(1)$ sabit sayıda işlem gerektiren program kesimlerinin, $O(\log n)$ daima ² iki eşit kümeye bölüp bunlardan birinin içinde aramaya devam etmenin, $O(n)$ doğrusal aramanın, $O(n \log n)$ bazı sıralama yöntemlerinin, $O(n^2)$ matris toplama, $O(n^3)$ matris çarpma işlemlerinde tipik algoritma karmaşıklığı fonksiyonlarıdır.

1.3.2. Algoritma Etkinliği

Algoritma karmaşıklığını tanımlarken verilen çözüm için en çok işlemi gerektirecek (en olumsuz) dağılımı gösterdiğini kabul etmiştik. Algoritma etkinliği ise verilerin dağılımının da bir etken olarak ele alındığı bir tanımdır. Algoritma etkinliği genellikle algoritma karmaşıklığı fonksiyonları aynı olan, iki algoritma üzerinde tartışılır. Algoritma etkinliği $c.O(g(n))$ tanımındaki c sayısının verilerin dağılımına bağlı olarak nasıl değiştiğini inceler. Aşağıda verilen sıralama algoritmaları için bu inceleme yapılacaktır.

- 1 SEÇSIRALA (N, DİZİ) YORDAMI ;
- 2 DISYAPILAR ;
- 3 DİZİ (*) TÜR (TAMSAYI, KESİRLİ) ;
- 4 N TAMSAYI ;
- 5 İÇYAPILAR ;
- 6 SAKLA TÜR (DİZİ) ; /* SAKLA DİZİ ile aynı türde olmalı*/
- 7 I, J, K DİZİ ÜZERİNDE GÖSTERGE SINIR (1:N) ;


```

8  KOMUTLAR ;
9  SAYARAK YINELE ( I := 1, I = N, I := I+1 ) ;
    [ /* Bu kesimde Dizi adlı yapının I'nci yerinde bulunan
        ögeyi sıralayınız */
10  J := I ;    /* J indisi hatırlanacak */
11  SAYARAK YINELE ( K := I+1, K > N, K := K+1 ) ;
12  [ EGER Dizi ( K ) > Dizi ( J )
13  İSE J := K ]
        /* Dizi'nin I'nci yerindeki öge ile yer değiştirecek en
        küçük öge Dizi'nin J nci yerindedir. Şimdi değiştiriniz. */
14  SAKLA := Dizi ( I ) ;
15  Dizi ( I ) := Dizi ( J ) ;
16  Dizi ( J ) := SAKLA ;
17  ]
18  SEÇSIRALA BİTTİ ;

```

Bu algoritmanın en çok tekrarlanan kesimi 11-13'ncü komutları arasındaki kesimdir. Bu kesimin işleyiş sayısı şöyle özetlenebilir.

KOMUTLAR :		Komut 11-13
I-değeri		işleyiş sayısı
1		n-1
2		n-2
3		n-3
.		.
.		.
n-2		2
n-1		1
n		0

Yukarıdaki dizi matematik birleştirmeler ile

$$\frac{1}{2}n^2 - \frac{1}{2}n \quad \text{olarak hesaplanır.}$$

Komut 9-10 ve 14-16' nında N kez uygulanacağı düşünülürse

$$g(n) = \frac{1}{2}n^2 - \frac{1}{2}n + n$$

biçiminde belirlenebilir, n büyük değerlere giderken yalnızca $\frac{1}{2}$

n teriminin dikkate alınarak bu algoritmanın algoritma karmaşıklığını ve verilerin dağılımından bağımsız olan algoritma etkinliğini

$$c.O(n^2), \quad c = \frac{1}{2} \quad \text{olarak belirleyebiliriz.}$$

Şimdi kabarcık sıralama olarak bilinen bir başka algoritma yazalım.

```

1 KABARCIKSIRALA ( N, Dizi ) YORDAMI ;
2 DIŞYAPILAR ;
3 Dizi ( * ) TÜR ( TAMSAYI, KESİRLİ ), GİRİŞ-ÇIKIS ;
4 N TAMSAYI, GİRİŞ ;
5 İÇYAPILAR ;
6 DEĞİŞTİ MANTIKSAL, BAŞLANGIÇ := DOGRU ;
7 I Dizi ÜZERİNDE GÖSTERGE, SINIR ( 1 : N ) ;
8 SAKLA TÜR ( Dizi ) ;
9 /* SAKLA DEĞİŞKENİ Dizi İLE AYNI TÜRDE */
9 KOMUTLAR ;
10 ÖNCE DEĞİŞTİ SINA, DOGRUYSA YİNELE ;
```



```

11  [ DEĞİSTİ := YANLIŞ ;
12  SAYARAK YİNELE ( I := 1, I = N, I := I+1 ) ;

13  EĞER DİZİ ( I ) > DİZİ ( I + 1 )
14  İSE [ DEĞİSTİ := DOĞRU ;
15      SAKLA := DİZİ ( I )
16      DİZİ ( I ) := DİZİ ( I + 1 )
17      DİZİ ( I + 1 ) := SAKLA ]
18  ]
19  ]
20  KABARCIKSIRALA BİTTİ ;

```

Bu algoritmanın 11-19' ncu komutları 10' ncu komuta ulaşılan herkes için $n-1$ yineleme gerektirir. Eğer o anda ele alınan iki değer değiştirme gerektirmiyorsa 14-17 o anda işleme girmez, ancak bu bu aşamada önemsizdir. Bu algoritmanın 10 ncu satırında başlayan dıştaki döngü ise son geçişte yerdeğiştirme olmuşsa bir daha yinelenir, aksi halde sona erer. Küçükten büyüğe sıralama yapan bu algoritmada yeri uygun olmayan küçük değerler her geçişte yerine doğru bir adım (biryer) yaklaşır. Bu nedenle bu döngü, sırası en aykırı olan ögenin gerçek yerine uzaklığından bir fazla işlem yapar. Bu nedenle komut 10-11 (ve dolayısıyla 12'ye ulaşma sayısı) 1 ile $(N - 1) + 1$ arasında değişkendir. Algoritma karmaşıklığını incelerken en çok işlem sayısı ile ilgilendiğimiz için bu sayıyı

n olarak alırız ve

$$g(n) = n \cdot (n-1) \cdot c \cdot 0(n), \quad c = 1$$

olarak belirleriz.

Algoritma etkinliği ise verilerin o işletimdeki dağılımını da dikkate alarak işletim zamanını tahmin etmeye çalışır. Buna göre ilk algoritmanın (seçsırala) algoritma etkinliği verilerin dağılımından bağımsız olarak

$$c \cdot 0(n), \quad c = 0.5$$

ikinci algoritma (kabarcıksırala) algoritma etkinliği ise $c \cdot 0(n), \quad c = 1$ ile $c \cdot 0(n), \quad c = 1$

arasında değişkendir.

Bu nedenle birçok ögesi sıralı durumda, sırasızların da sonuçta olacakları yere uzak olmadıkları durumda KABARCIKSIRALA yordamı daha etkindir. Öte yandan tümüyle rastgele sayıları sıralamada KABARCIKSIRALA algoritması SEÇSIRALA' ya göre iki kat fazla zaman gerektirebilir.

1.3.3 SİSTEM ÇÖZÜMLEME VE YAZILIM GELİŞTİRMEDE BU BİLGİ NASIL UYGULANABİLİR ?

Geliştirilmiş bir programın işletim zamanı birçok etkene bağlıdır. Örneğin :

- Hangi bilgisayarda işletiliyor ,
- Hangi programlama dili ve bu dilin derleyicisi mi yoksa yorumlayıcısı mı kullanılıyor ,
- Derleyicinin kod enuygunlaştırma niteliği kullanılıyor mu ?
- Programcı program etkinliğini sağlayacak noktalara ne

ölçüde önem veriyor ,

- Sorunu çözecek algoritmalarından hangisi kullanılıyor, ve benzeri etkenlerdir.

2.1 DİĞİ VERİ YAPISI

Bazı durumlarda en önemli etken sonuncusudur. İlk dört maddede sayılan özelliklerin bazıları onlarca kat bazıları ise yüzde 5 - 10 etki yapabilir. Oysa bazı durumlarda uygun algoritmayı seçmemek n kat yada $\log n$ kat fazla işletim zamanına neden olabilir.

Görüldüğü gibi n sayısının yüzler dolayında bulunduğu durumlarda bu algoritmaların bilinmiş ve kullanılmış olması çok önemli etkinlik farkları doğurur. Bu nedenle bir yazılım geliştirecek bilişim elemanının kullanılabilecek veri yapıları ile muhtemel algoritmaları, bunların karmaşıklık ve etkinlik fonksiyonlarını çözüm için masa başına oturmadan önce bilmesi, tartışabilmesi hem profesyonel alanda hem akademik alanda önemlidir.

(i) Veri yapısına erişim.

(ii) Kullanım ve önemli algoritmalar.

Yukarıda belirtilen edimler her önemli ve sık kullanılan veri yapısı için belirlenmelidir. Bu kavramları dışı, ayrıca gibi 211. bölümde bir veri yapısı üzerinde örneklerle açıklığa kavuşturulmuştur.

2. DİZİ VERİ YAPISI VE ÖZEL TÜRLERİ İLE BAĞLAÇSIZ YIGIT

VERİ YAPISI VE ÖZYİNELİ ALGORİTMALAR

2.1 DİZİ VERİ YAPISI

Tek boyutlu dizi (vektör), çift boyutlu dizi (matris) ve daha çok boyutlu olan diziler her bilgisayar programcısının kullanmayı alışkanlık edindiği veri yapılarıdır. Bu veri yapıları COBOL, FORTRAN, PASCAL, PL/1 gibi dillerin derleyicilerince sağlanan yapılardır.

2.1.1 DİZİ VERİ YAPISI NASIL İNCELENİR?

Bir veri yapısı (data structure) incelenirken şu adımlardan geçerek belirleme hem kuramsal açıklık sağlayacak, hem de güvenilir program yazmaya yöneltecektir.

- i) Veri yapısı tanımı,
- ii) Veri yapısı gösterimi ,
- iii) Veri yapısı denetim grubuna erişim,
- iv) Veri ögesine erişim,
- v) Kullanım ve önemli algoritmalar.

Yukarıda belirtilen adımlar her önemli ve sık kullanılan veri yapısı için belirlenmelidir. Bu kavramları dizi, matris gibi bilinen bir veri yapısı üzerinde örnekleyerek açıklığa kavuşturalım:

2.1.2 VERİ YAPISI TANIMI

Veri yapısı tanımı ; Veri yapısının kullanıcı algoritması tarafından görülen soyut biçimidir. Veri yapısının zorunlu başlangıç değerleri (varsa) ve geçerlilik göstergeleri (varsa) bu düzeyde tanımlanır.

Örnek: A (0 : 2 , -1 : +1) kesirli.

A	-1	0	+1
0			
1			
2			

Kullanıcı bu veri yapısı tanımından herbirne birer kesirli değışken yerleştirebilecegi görüntüsü yanda gizilmiş yapıyı algılar.

Sekil 11 Veri Yapısı Tanımı Örneği

2.1.3 VERİ YAPISI GÖSTERİMİ

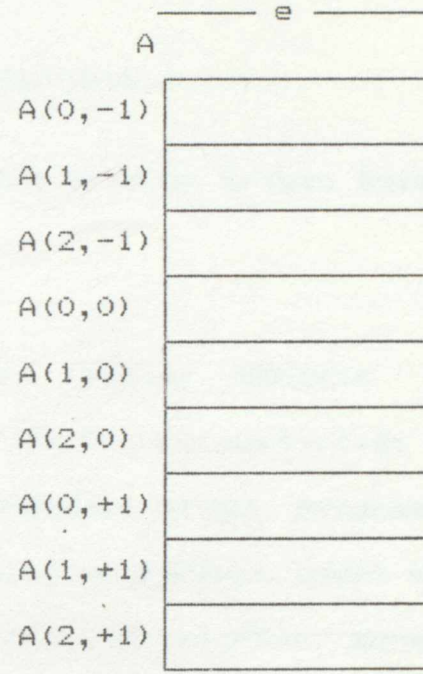
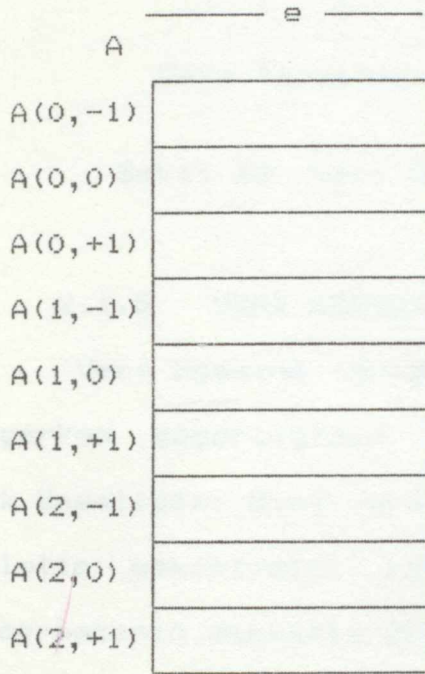
Veri yapısı gösterimi' veri yapısının bilgisayar belleğine yerleşim biçimidir. Kullanılan bilgisayarın ' bellek sözcük boyu dikkate alınarak bellekte uygun bir yerleşim düzenlenir.

Örnek: A (0 : 2 , -1 : +1) kesirli Tanımının üç temel yaklaşımı vardır. Bunlar; satır önce yerleşim, sütun önce yerleşim ve sıra düzen diziler ile yerleşimdir. Bu yerleşim türleri ile ilgili çizimler şekil 12' de görölmektedir.

2.1.4 VERİ YAPISI DENETİM GRUBUNA ERİŞİM

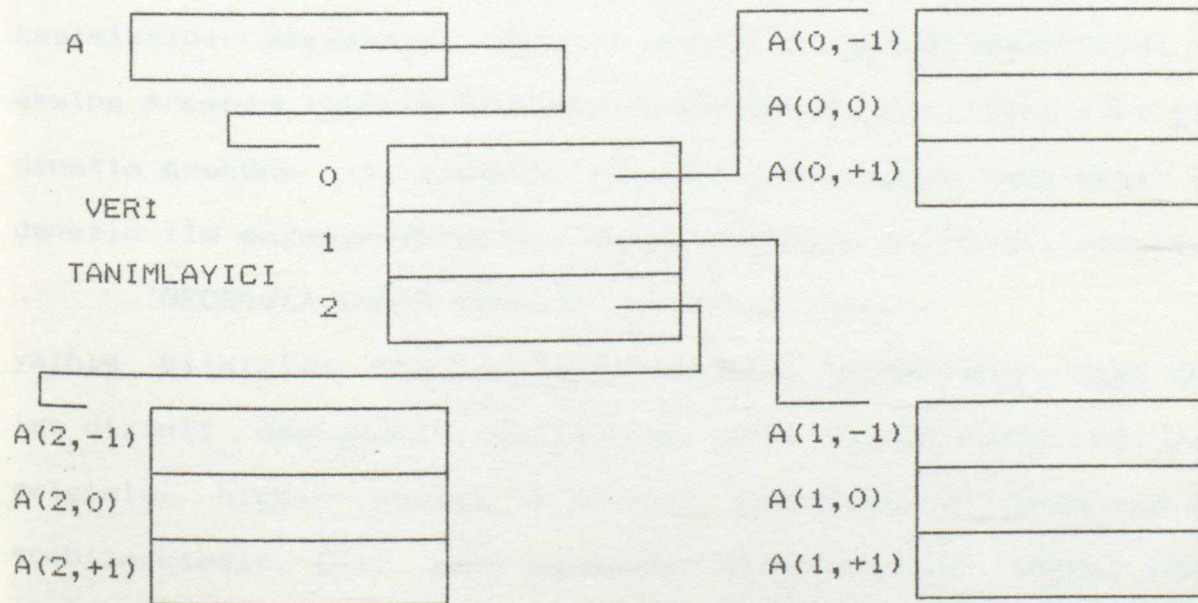
'Veri yapısı denetim grubuna erişim', bilgisayar belleğinin bir bölgesinde yerleştirilmiş olan veri yapısı gösteriminin

başlangıcına erişimdir. Öteki deyişle veri yapısının sembolik adı ile bellekteki yerleşim adresinin eşleştirilmesidir.

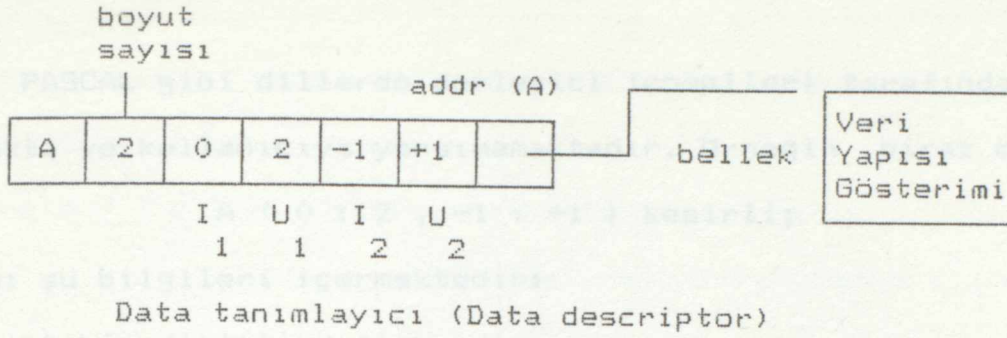


Şekil 12-A Satır Önce Yerleşim

Şekil 12-B Sütun Önce Yerleşim



Şekil 12-C Sıradüzen Diziler ile Yerleşim



Şekil 13 Veri Yapısı Denetim Grubuna Erişim Düzeni

2.1.5 VERİ ÖGESİNE ERİŞİM

'Veri ögesine erişim' in veri yapısı denetim grubundan geçerken geçerliliğinin sağlanması yazılım güvenilirliği açısından çok önemlidir. Dizi veri yapısını sağlayan birçok programlama dili işletim etkinliğini kaybetmemek için bu noktayı ihmal ederler ve programcının dikkatli program yazmasına bırakırlar. Buradaki yaklaşım bir veri ögesine erişim geçerli değilse ve bu durum 'Veri yapısı denetim grubu'da incelenerek bulunabiliyorsa bu yanlış erişimi hemen ve açıklıkla belirterek erişimi engelleyecek program kesimlerini yazmaktır. Birçok derleyici gerçekleştiricisi bunun aksine program işletim hızından kaybetmemek için 'dizi veri yapısı denetim grubunu' ya tümüyle ortadan kaldırmakta yada eksik bilgi denetim ile düzenlemektedir. Bunun sonucuda bulunması oldukça güç

'GEÇERSİZ İNDİS YANLIŞI' (invalid index)

yalnış bildirisi ile kesilen programlar artmaktadır. Daha kötüsü ise dizinli değişkenin yerleştiği alan içinde kalan bazı yanlış erişimler hiçbir yanlış bildirisi üretmeksizin işlemlere devam edebilmektedir. Dizi veri yapısının denetim grubu COBOL, FORTRAN

PL/1, PASCAL gibi dillerde derleyici (compiler) tarafından oluşturulmakta ve kullanıcıya yansımamaktadır. Örneğin biraz önce verilen

$$A (0 : 2 , -1 : +1) \text{ kesirli;}$$

tanımı şu bilgileri içermektedir:

- . A bir dizinli değişkendir.
- . A'nın içerigi kesirlidir.
- . A'nın iki boyutu vardır.
- . A'nın birinci boyutu 0, 1, 2 indisleri için geçerlidir.
- . A'nın ikinci boyutu -1, 0, +1 indisleri için geçerlidir.

Kullanılan üst düzey programlama dilinin derleyicisi kendisi için tanımlı olan dizi tanımının sözdizim çözümlemesi (syntax analysis) yapınca belirtilen parametreleri oluşturup dizi veri yapısının denetim grubunu kurmaktır. Veri ögesine erişimde, veri yapısı denetim grubunun güvenlik denetiminden geçmiş olan erişimin fiziksel bellekte doğru yere ulaşmasının sağlanmasıdır. Bu yer verinin içeriğini saklar ve okuma, yazma, değiştirme işlemleri bu alan üzerinde yapılır.

Yine önceki örneğimize dönelim

$$A (0 : 2 , -1 : +1) \text{ kesirli;}$$

tanımı

$$A (\begin{matrix} I & : & U \\ 1 & & 1 \end{matrix} , \begin{matrix} I & : & U \\ 2 & & 2 \end{matrix}) \text{ tür;}$$

biçiminde genelleştirilebilir. İşletim zamanında ise $A (i , j)$ biçiminde belirli bir ögesine erişim istenir. Bir çok programlama dilinde i ve j yerine aritmetik ifadeler kullanılabilir. Şimdi daha önce sözü edilen gösterimlerden matris ve çok boyutlu

Dizilerin satır önce yerleşim yönteminin kabul edildiğini varsayarak ögeye erişimi inceleyelim:

$$A(i, j) = A \text{ dizisinin başlangıç adresi} \\ + (i - I) \text{ adet satırdaki ögenin alacağı yer} \\ + i' \text{ nci satırdaki } (j - I) \text{ ögenin alacağı yer}$$

Öteki deyişle daha kesin ifadesi ile

$$A(i, j) = \text{adres}(A) + (i - I) * (U - I + 1) * e + (j - I) * e$$

olacaktır. Burada e herbir ögenin kaç karakter (byte) yer alacağını gösteren bir tamsayıdır.

Yukarıdaki ifadede bazı değişkenler ' derleme aşaması'nda belirlenmekte bazıları programın belleğe yerleştirilişi sırasında belirlenmektedir. (Adres(A) böyledir) Bu nedenle işletim aşaması öncesinde yapılabilecek bazı etkileşimler vardır:

$$A(i, j) = \text{Adres}(A) - (I * (U - I + 1) * e + I * e) \\ + i * (U - I + 1) \\ + j * e$$

olarak elde edilir. Derleme zamanında hesaplanabilecek

$$c = I * (U - I + 1) * e + I * e \\ c = U - I + 1 \\ c = e$$

değerleri ile bu ifade

$$A(i, j) = \text{adres}(A) - c + i * c + j * c \quad \text{olacaktır.}$$

Bu erişim 'birleştirici' dili 'düzeyinde' işletim zamanında en az iki çarpma üç toplama/çıkarma ve yedi yükleme komutu ile yapılabilir. Birer yükleme işlemi Adres(A), i ve j değişkenleri içindir. Küçük bir tamsayı değeri olan c_1, c_2, c_3 , bazen komutun içine de yerleşebilir. Yerleşmiyor ise onlar için de erişim hesaplanmalıdır. Ayrıca bilgisayarın işlem biriminin yapısı nedeniyle adres hesaplanırken ara sonuçların saklanması yükle / sakla işlerini arttıracaktır. Özetle bir dizinli değişken ögesine erişim için üretilecek komut grubu şöyledir;

birinci indis ifadesini hesapla i' yi oluştur
j'yi oluştur
Dizi A denetim grubunu kullanarak geçerli erişim sına
erişilecek ögenin adresini hesapla
eriş ve işlem yap

2.2 DİZİ VERİ YAPISI ÖZEL TÜRLERİ

2.2.1 ÜÇGEN VE KUŞAK MATRİSLER

Özel matrisler bazı uygulama alanlarında oldukça sık raslanan sıfırdan farklı öğelerin kümelenişi yönünden belirgin biçimde tanınabilen matrislerdir. Fizik, Yöneylem Araştırması, Üretim Planlaması gibi gerçek konular ile Matematik gibi uygulamaya destek

Bir matrisin özel matris görüntüsü gösterdiği durumda o uygulama için özel bir veri yapısı geliştirmeden önce şu karar noktalarını incelemek gereklidir:

- i) Özelve veri yapısı kurulduğunda elde edilecek bellek kazancı ne kadardır? Eğer bellek kazancınız üç-beş yüz karakter ise özel veri yapısı için emek harcamaya gerek yoktur. Çünkü bu kazancımızın bir kısmı da özel veri yapısını denetleyen algoritmalara harcanacaktır.
- ii) Özel veri yapısı kurulduğu takdirde sıkça uygulayacağınız algoritmanın, genel matris yapısı kullandığınızda uygulanacak algoritmaya göre 'Algoritma Karmaşıklığı' ve 'algoritma Etkinliği' nedir? Önemli bir işletim dezavantajı getiriyor ise vazgeçiniz.

Bu karar noktalarındaki tartışma sizi özel matris yapısı kurmaya yöneltirse gerçekleştirmek için algoritma yazmaya yönelmek gerekecektir. Şimdi en kolayından başlayarak özel matrisler için özel veri yapılarını inceleyelim.

2.2.1.1 ÜÇGEN MATRİSLER

Sayısal çözümleme problemlerinde sıkça karşılaşılan özel matrislerden biri de 'Alt üçgen matris (lover triangular matrix)'yada 'Üst üçgen matris (upper triangular matrix)'dir.

n

x	0	0	0	0	0
x	x	0	0	0	0
x	x	x	0	0	0
x	x	x	x	0	0
x	x	x	x	x	0
x	x	x	x	x	x

Bir alt üçgen matris örneği yanda verilmiştir.

Bir Üçgen matris $n \times n$ kare matristir. Köşegen elemanları sıfırdan farklı olabilir. Köşegenin ya üstü sıfırdan farklıdır (üst üçgen matris) yada alt kesimi (alt üçgen matris) sıfırdan farklıdır. Buna göre sıfırdan farklı olabilen öge sayısı

$$k = \sum_{i=1}^n i \longrightarrow \frac{n(n+1)}{2} \text{ olarak belirlenir.}$$

Sıfırdan farklı öge sayısıyarıdan fazla olduğuna göre bu matrisin seyrek matris olarak gösteriminin bellek kazancı sağlayacağı söylenemez. Ancak normal matris yerleşiminde veri ögelerine erişim için kullanılan erişim yordamı yerine birbaşka erişim yordamı yazılarak bellek kazancı sağlanabilir. Aşağıda yerleşim örneği verilmiştir.

A	a	a	a	...	a	a		a	a			
	11	21	22	31		33	41		44	51		
ALT												
	1	2	3	4	5	6	7	8	9	10	11	12

$A(i,j)$ ögeleri $ALT(K)$ üzerinde yerleştirilmiştir. Alt üçgen matrisin ögelerine erişim fonksiyonu $k = f(i,j)$ olacaktır.

Şimdi bu fonksiyonu bulalım:

$$k = \sum_{t=1}^{I-1} t + j$$

Bu satırdan önce yerleşmiş ögeler

bu satır içinde bu elemanın yeri

$I \geq J$
 $I = 1$
 1 İÇİN
 $I = 1$
 2

$$k = \frac{(I-1) \cdot I}{2} + J \quad \text{Olacaktır.}$$

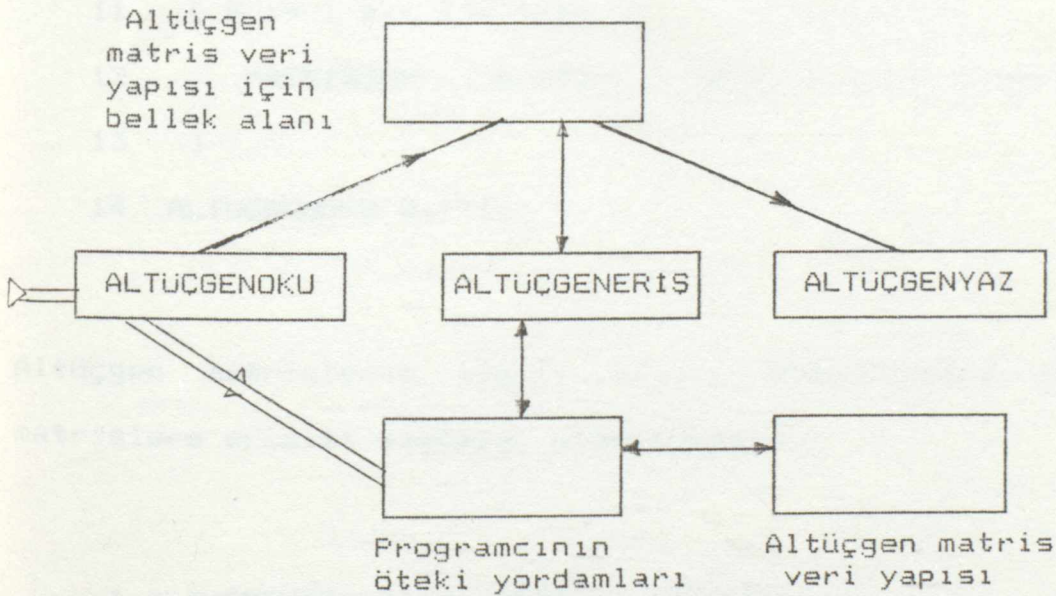
İki boyutlu dizi (A matrisi) ögesine erişim (2 çarpma 3 toplama) yerine önce uygun indis hesaplanacak (2 çarpma, 2 toplama). Erişimde biraz yavaşlamaya karşılık yüzde 50 'ye varan bellek kazancı sağlanmıştır. Tam fonksiyon aşağıda verilmiştir.

EGER I ≥ J

İSE K := ((I-1) * I / 2 + J)

DEĞİLSE K := 0

Görüldüğü gibi alt üçgen matris için veri yapısı oluşturma üç kolay yordamın buluşması ile gerçekleştirilebilmektedir.



Sekil 16 Altüçgen Özel Veri Yapısı İçin Yordam Kümesi


```

1 ALTÜÇGENOKU(N) YORDAMI;
2 KUTUKLER;
3   MATRİS GİRİŞ KULLANIMINDA KART KUTUGUDUR;
4 DISYAPILAR;
5   N TAMSAYI GİRİŞ;
6 A(N) TÜR (TAMSAYI, KESİRLİ), GİRİŞ;
7 İÇYAPILAR;
8   ALT(N*(N+1)/2) TÜR (TAMSAYI, KESİRLİ), EVRENSEL;
   /* PROGRAMA DÖNÜŞTÜRÜRKEN ALT İÇİN ÖGE SAYISI BELİR- */
   /* LEMEK ZORUNDA KALIRSANIZ N*(N+1)/2 DEĞERİNİN BU Sİ- */
   /* NİRİ ASMAĞINI DENETLEYEN BİR KOMUT EKLEYİNİZ      */
9 KOMUTLAR;
10  SAYARAK YİNELE (I := 1, I   N, I := I+1);
11  [ K := 1 * ( I-1 )/2;
12    MATRİSDEN ( A(K+J), J := 1, J > I, J := J + 1 ) OKU;
13  ]
14 ALTÜÇGENOKU BİTTİ;

```

Altüçgen matrislerle ilgili ikinci algoritmamız ise altüçgen matrislere erişimi sağlayan algoritmadır.

```

1 ALTÜÇGENERİŞ(I,J) TAMSAYI YORDAMI;
2 DISYAPILAR;
3   ALT(*); /* ALT DİZİ DAHA ÖNCE TANIMLANDIĞI GİBİDİR */
4 KOMUTLAR;

```

```

5  EGER  I ≥ J
6  ISE [ K := ( I-1 ) * I / 2 + J ]
7  DEĞİLSE [ K := 0 ]
8  ALTUÇGENERİS := K;
9  ALTUÇGENERİS BİTTİ;

```

Yukarıda yazılan Altüçgeneriştir yordamında denetim eksikliği vardır. Şimdi daha güvenilir bir Altüçgeneriştir yordamı yazalım.

Bu yordamın amacı, bir altüçgenin altı kenarının toplamı ile iç kenarlarının toplamı arasında bir ilişki kurmaktır. Bu ilişki, altüçgenin alanı ile de ilişkilidir.

```

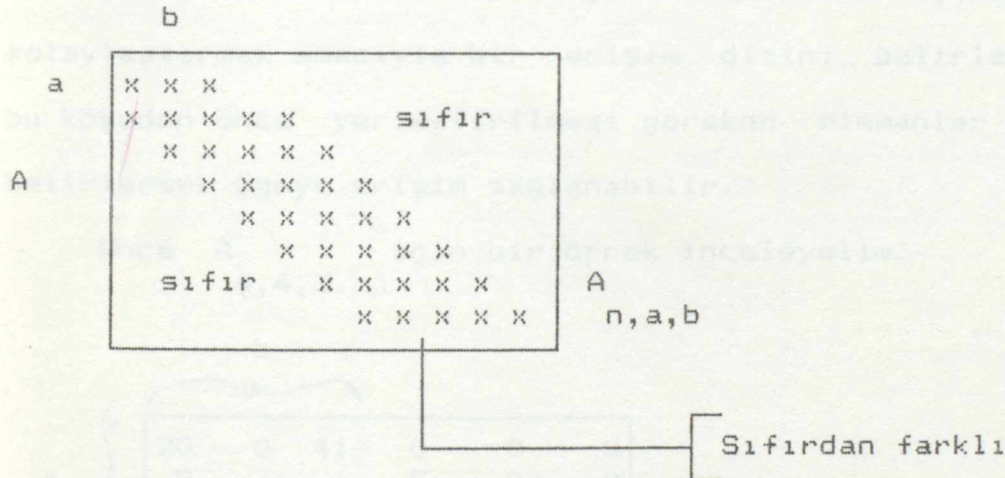
1  ALTUÇGENERİS (I,J) TAMSAYI YORDAMI;
2  DISYAPILAR;
3  ALT(*), N;
   /* YUKARIDAKİ İFADE ALT VE N DEĞİŞKENLERİ          */
   /* TANIMLANDIKLARI YERDEKİ GİBİDİR ANLAMINDADIR    */
4  KOMUTLAR;
5  EGER I < 0 YADA I > N YADA
6      J < 0 YADA J > N
7  ISE [ 'GECERSİZ İNDİS' YAZ ]
8  DEĞİLSE [ EGER I ≥ J
9      ISE [ K := ( I - 1 ) * I / 2 + J ]
10     DEĞİLSE [ K := 0 ]
11     ]
12 ALTUÇGENERİS BİTTİ;

```


Görüldüğü gibi üçgen matris özel veri yapısı kullandığınız programlama dilinin derleyicisinde hiçbir değişiklik yapmaksızın kullanıcı programında yazacağınız üç küçük yordam ile sağlayabiliriz. Veri yapısı bu niteliği gösterdiğinde bu olanak kullanmak alışkanlık haline getirilmelidir.

2.2.1.2 KUSAK MATRİSLER

Bir başka sıkça karşılaşılan matris türü 'kuşak matris (band matrix)' türleridir. En genel görünümüyle bir kuşak matris aşağıda verilmiştir.



Eğer $a = b$ ise A olarak yazılır. Bu durumda köşegen, köşegenin alt ve üstündeki $(a - 1)$ köşegen sıfırdan farklı sıfırdan farklı öğeler içerebilir, öteki öğeler sıfırdır.

Eğer $a = b = 2$ ise yalnızca köşegen, alt ve üstünde birer köşegen daha sıfırdan farklı olabilir. Bu matris 'üç köşegen matrisi' olarak anılır.

Şimdi kuşak matrislerin en genel biçimi için bir veri yapısı görünümü tasarlayalım.

bu formülü

$$n.a - 1 - 2 \dots - a + n.b - 1 - 2 \dots - b + n$$

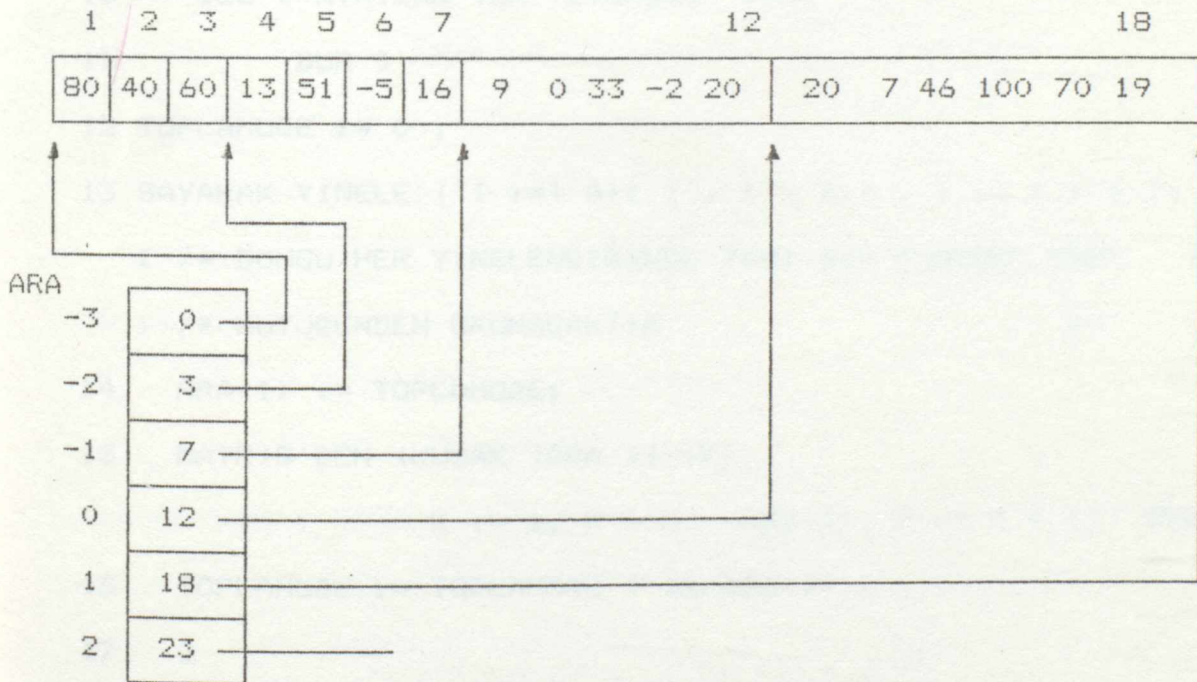
olarak düşürsek

$$\frac{n^2}{2} - (n(a+b-1)) - \frac{(a+1)a}{2} + \frac{(b-1)b}{2}$$

Matriste sıfır olan alttaki ve üstteki bölgelerdeki toplam sıfır sayısı t' dir. Bu değer uygulamamızda önemli bir kazanç olduğunda aşağıdaki yordamları oluşturmaliyiz.

KUSAKMKUR : Kuşak matris kur

KUSAKMERİS : Kuşak matris'e eriş



Sekil 18 Kuşak Matris Yerleşim Örneği

Şimdi bu özel veri yapısı ile ilgili algoritmaları yazımına geçelim. İlk algoritmamız kuşak matris özel veri yapısını oluşturan

KUŞAK MATRİS KUR algoritmasıdır.

```

1  KUŞAKMKUR(N,A,B)YORDAMI;
2  KUTÜKLER;
3  MATRİS GİRİŞ KULLANIMINDA KART KUTUGUDUR;
4  İÇYAPILAR;
5  T TAMSAYI, DEĞER = 1000 ;
6  KUŞAK(T) TÜR (TAMSAYI,KESİRLİ)EVRENSEL;
7  ARA(-A+1 , B-1) TAMSAYI EVRENSEL;
8  KOMUTLAR;
9  EĞER N*(A+B-1)-((A+1) * A+(B+1) * B)/2 > T
10 İSE ['AYRILAN YER YETERSİZ' YAZ;
11     DUR ]
12 TOPLAMÖGE := 0 ;
13 SAYARAK YİNELE ( I :=( A+1 ) , I > B-1 , I := I + 1 );
    [ /* DÖNGÜ HER YİNELENDİĞİNDE YENİ BİR KÖŞEĞEN KART */
    /* KUTUGUNDEN OKUNACAKTIR */
14  ARA(I) := TOPLAMÖGE;
15  MATRİS'DEN (KUŞAK (ARA (I)+K),
16  DEĞER K := 1, K > N - ABS(I), K := K + 1)) OKU;
16  TOPLAMÖGE := TOPLAMÖGE + (N-ABS(I));
17 ]
18 KUŞAKMKUR BİTTİ ;

```

Bu algoritmada belki ilk bakışta anlaşılmayacak nokta

$N - \text{ABS}(I)$ köşegeni içinde bulunan A

ifadesinin iki yerde kullanımıdır. Köşegen numarası olan I' nin ana köşegen için sıfır olduğunda köşegenin altında -1 köşegeninde bir eksik eleman, ana köşegen üzerinde $+1$ köşegeninde bir eksik eleman olduğunu ve köşegen ana köşegenden uzaklaştıkça bu etkinin devam ettiğine dikkat edilmelidir.

Şimdi ise ikinci özel algoritmamız olan KUSAK MATRİS'E ERİŞ algoritmasını yazalım.

2 DİŞYAPILAR;

1 KUSAKMERİS(N,A,B,I,J)TAMSAYI YORDAMI;

2 DİŞYAPILAR;

3 KUSAK(*), ARA(*);

4 KOMUTLAR;

5 EĞER $I > J$ YADA $I < 1$ YADA

$J > N$ YADA $J < 1$

6 İSE ['İNDİS YANLIŞI' YAZ; DÖN]

7 EĞER $J > I$

8 İSE [EĞER $J - 1 < B$

9 İSE KUSAKMERİS := ARA(J-1) + I

10 DEĞİLSE KUSAKMERİS := 0]

11 DEĞİLSE [EĞER $I - J < A$

12 İSE KUSAKMERİS := ARA(J-I) + J

13 DEĞİLSE KUSAKMERİS := 0]

14 KUSAKMERİS BİTTİ;

Bu algoritmada dikkat edilmesi gereken bir nokta erişim formülünün ana köşegenin üzerinde 'köşegen içinde eleman no' olarak I' yı

yani özgün matrisin satır numarasını ana köşegenin altında ise J' yi içerdigidir. Bu algoritma da beş parametresi ile programı okuyan kişiye gereksiz düşünce yükü getirmektedir. Bilgi gizleme kavramını kullanan bir başka algoritma aşağıda yazılmıştır.

```

1 X(I,J) YORDAMI;
2 DIŞYAPILAR;
3   KUŞAK(*), ARA(*);
4   N, A, B, TAMSAYI;
5 İÇYAPILAR;
6   X TUR (KUŞAK) ;
   /* X YORDAM TURU SAKLANAN VERİ TURU İLE          */
   /* UYUMLU OLMALI                                  */
7 KOMUTLAR;
8   EGER I > N YADA I < 1 YADA
9       J > N YADA J < 1
10   İSE [ 'İNDİS YANLIŞI' YAZ; DÖN ]
11   EGER J > I
12   İSE [ EGER J - I < B
13       İSE X := KUŞAK(ARA(J-I) + I)
14       DEĞİLSE X := 0 ]
15   DEĞİLSE [ EGER I - J < A
16       İSE X := KUŞAK(ARA(J-I) + J)
17       DEĞİLSE X := 0 ]
18 X BİTTİ;

```


2.2.2 SEYREK MATRİSİN DİZİLER İLE GÖSTERİMİ

Seyrek matris (Sparse matrix) mühendislik, fizik, yöneylem araştırmaları, bilgisayar yazılımında derleyiciler ve benzeri konularda sıkça karşılaşılan bir veri türüdür. Gerçek uygulamalarda yüz satır, birkaç yüz sütünlük matrisler karşılaşılabilecek tipik boyutlardır. Seyrek matrisde sıfırdan farklı öğeler bir kümelenme göstermezler, bu nedenle sıfırdan farklı olan değer yanında bu değerlerin yer aldığı satır ve sütun bilgisi de saklanmalıdır. Aşağıda ilk bakışta seyrek matris olabileceği izlenimi veren bir matris verilmiştir. Ancak seyrek matris olduğu kararı 'seyrek matris özel veri yapısı' nın bellek gereksinimi hesap edilip bellek yönünden kazançlı ise verilebilir.

2.2.2.1 SEYREK MATRİS TANIMI

Bilgisayar kullanıcısı matrisi alıştığı iki boyutlu yapı içinde düşünecektir. İç yapının (gösterimin) değişik olması matris, dizi veri yapısının soyut tanımını ve üzerinde yapılabilir işlemleri etkilemeyecektir.

Sütun	1	2	3	4	5	6	7
Satır 1	46	0	0	0	-13	0	0
2	0	0	80	1	0	0	0
3	0	-40	0	0	0	4	0
4	0	0	0	0	0	0	26
5	11	0	0	0	0	0	0

2.2.2.2 SEYREK MATRİS GÖSTERİMİ (Uç Dizi ile Gösterim)

Bu matris ilk bakışta elemanlarının bir çoğunun sıfır,

oldukça az sayıda elemanın sıfırdan farklı olması ile 'Seyrek matris (sparse matrix)' izlenimi vermektedir. Ancak bir matrisin seyrek matris gösterimini hesaba dayalı bir işlemdir.

Örneğin yukarıdaki matris için bir gösterim şöyle olabilir;

	satır sayısı	sütun sayısı	
0	5	7	8
1	1	1	46
2	1	5	-13
3	2	3	82
4	2	4	1
5	3	2	-40
6	3	6	4
7	4	7	26
8	5	1	0

sıfırdan farklı
değer sayısı

Deger
Sütun
Satır

Üçlüler

Sekil 19 Bir Seyrek Matrisin Üç Vektör ile Gösterimi

Bilgisayarların birçoğu en etkin biçimde bellek sözcük boylarının 1, 1/2 ve 2 katı uzunluktaki bellek sözcükleri üzerinde işlem yaparlar. Bu nedenle satır ve sütun belirten tamsayı değişkenler genellikle 2 sekizli (byte) olarak hesaplanabilir.

i) Matrisin türü tamsayı olduğunda

normal yerleştirme: $n*m*2$ sekizli

üç vektör ile gösterim: $(k+1)*(2+2+2)$ sekizli k =sıfırdan farklı değer sayısı

Eğer $6.(k+1) < 2.n.m$

ise bu durumda üç vektör ile gösterim bellek gösterimi yönünden avantajlıdır.

ii) Matrisin türü kesirli olursa

$(k+1).(2+2+4) < 4.n.m$

eşitsizliği sağlanırsa seyrek matristen söz edilebilir.

Buraya kadar matrisin içeriği seyrek matris olursa ne zaman gösterimi değiştirebileceğimizi inceledik. Şimdi bu yapıda kurulmuş olan seyrek matrise erişimi ve veri ögelerine erişimi inceleyelim.

2.2.2.3 SEYREK MATRİSİN BAŞLANGICINA VE SEYREK MATRİSİN

ÜĞESİNE ERİŞİM

Seyrek matrisin başlangıcına erişim bu yapının yerleştiği adrese bir gösterge ile olur ve normal yerleşimden etkinlik yönünden farketmez.

Üğeye erişim rasgele erişimle yada adres hesaplamayla yapılmaz. İstenen satır ve sütun üğlüler içinde aranarak bulunur. İşletim zamanında bu gösterimdeki bir seyrek matrise yeni bir sıfırdan

farklı bir değer eklemek birçok kaydirmayı gerektiren zaman alıcı bir işlemdir. Eleman çıkarma az sayıda ise değeri sifıra eşitleyerek yapılabilir. Anlaşıldığı gibi seyrek matris gösterimi koşulu sağlanırsa ve işlemler sırasında koşul geçerliliğini sürdürürse bellek kazandıran fakat veri ögesine erişim, ekleme, çıkarma işlemlerinde özel kısıtlamalar getiren bir gösterimdir. Eger bu kısıtlamalara uyumlu algoritmalar bulunabilirse belirlenmiş işler için kullanılabilirler.

2.2.2.4 SEYREK MATRİS VERİ YAPISI ÜZERİNDE TEMEL ALGORİTMALAR

Bu grupta ele alacağımız önemli algoritmalar seyrek matrisin üç vektör ile gösterimi üzerinedir. Bunlar arasında Seyrek matris kurma (SEYREKMKUR), Seyrek matris ögesine erişme (SEYREKMERİŞ), Seyrek matrisin devrighi (Transpoze) (SEYREKM DEVRİK) ve Seyrek matris toplamı (SEYREKMTOPLA) vardır.

1 SEYREKMKUR(N,M,A) YORDAMI;

/* KARTLARA SATIR İNDİSİNE VE İKİNCİ ANAHTAR OLARAK */

/* SÜTUN İNDİSİNE GÖRE SİRALANMIŞ SIFIRDAN FARKLI */

/* MATRİS ÖGELERİ YAZILMIŞTIR */

2 KUTUKLER;

3 MATRİS GİRİŞ KULLANIMINDA KART KUTUGUDUR;

4 DİŞYAPILAR;

5 T TAMSAYI , DEĞER = 500;

6 A(0:T)(SA,SU) TAMSAYI;

7 DEĞER TUR (TAMSAYI, KESİRLİ) YAPI;


```

8  ICYAPILAR;
9  BUDEGER TUR(DEGER);
10 SAYAC TAMSAYI;
11 KOMUTLAR;
12 SAYAC := 1;
13 ÖNCE VERİBİTTİ(MATRİS)= YANLIŞ SINA, SONRA YİNELE;
14 [ MATRİS'DEN (SATIR,SÜTÜN,BUDEGER) OKU;
15   A(SAYAC).SA := SATIR ;
16   A(SAYAC).SU := SÜTÜN ;
17   A(SAYAC).DEGER := BUDEGER ;
18   EGER SAYAC < T
19     İSE SAYAC := SAYAC + 1;
20   DEĞİLSE [ 'AYRILAN YER DOLDU'YAZ ;
21   DÖN ]
22 ]
23 A(0).SA := N; A(0).SU := M; A(0).DEGER := SAYAC - 1 ;
24 SEYREKMKUR BİTTİ;

```

Bu algoritmadan sonra yazacağımız SEYREKMERİS algoritması kuruluşu sağlanmış yapı üzerinde işlem yapacaktır.

```

1  SEYREKMERİS(I,J) TAMSAYI YORDAMI;
   /* BU YORDAM EGER MATRİSİN İLGİLİ ÖGESİ SIFIRDAN */
   /* FARKLI İSE A DİZİSİNİN BU ÖGEYİ BULUNDURAN ALANIN */
   /* İNDİSİNİ VERİR AKSİ HALDE SIFIR DEĞERİ DÖNDÜRÜLÜR */
2  KOMUTLAR;

```

```

3  SAYARAK YINELE(W := 1, W > A(0).DEGER, W := W + 1);
4  [ EGER A(W).SA > I ISE [ SEYREKMERIS := 0;
5                                DON ; /* BU DEGER SIFIRDIR */]
6  EGER A(W).SA = I
7  ISE [ EGER SU(W) = J
8        ISE [ SEYREKMERIS := W;
9              DON ; /* BULUNDU */ ]
10 ]
11 ] SEYREKMERIS := 0 ;
12 SEYREKMERIS BITTI;

```

Yukarıda algoritmanın ' algoritma karmaşıklığı ' nın incelenmesi bize $c.O(k)$ yapısını gösterecektir. (k matristeki sıfırdan farklı öğelerin sayısı) Algoritma etkinliği ise ortalama olarak $1/2.c.O(k)$ olarak belirecektir. Normal matrislerde bir öğeye erişim zamanı $c.O(1)$ iken bu çok yüksek bir erişim zamanıdır.

Şimdi matris devrighi (transpoze) işlemini ele alalım. Ağıştığımız normal matrisler üzerinde bu algoritmanın $c.O(n.m)$ biçiminde bir algoritma karmaşıklığı ve algoritma etkinliği olduğu kolayca görülebilir. Üzerinde çalıştığımız üçlüler ile gösterimde matris devrighi 1'den k 'ya degin öğelerin her defasında bir sütundaki öğeleri seçip bunları yeni matrisin bir sütunu olarak oluşturacak m geçiş ile oluşturulabilir. Bu durumda algoritma karmaşıklığı $c.O(m.k)$ olur. $k \leq n.m$ dir, ancak $k = m.n$ olduğunda algoritma karmaşıklığı $c.O(n.m.m)$ olur. Bu algoritma karmaşıklığı kullandığımız üçlüler ile seyrek matris gösteriminin - başka bir önlem

almazsak - matris devriginde alystıgımız işlem zamanlarının m katı fazla zaman gerekebileceğini belirtmektedir.

Şimdi bu süreyi azaltmak için bulabileceğimiz yolu düşünelim:

1. Biraz önce düşündüğümüz algoritmanın zayıf noktası eldeki üçlülerin m kez ele alınma gereği idi. Bu gerek ele alınan ögenin yeni matrisin hangi noktasına yerleşeceğini bilmeyişimizden kaynaklanıyordu. Şimdi sorunu iki eşit parçaya bölelim.
2. Birinci aşamada eldeki matrisin her sütununda kaç eleman olduğu bulunacak, bu bilgi kullanılarak yeni oluşturulacak matris devriginin her satırının hangi indisten başlayacağı bulunacaktır.
3. İkinci aşama birinci aşamada oluşturulan göstergeleri de kullanarak bir geçiş ile matris devrigini oluşturacaktır.

1 SEYREKMDEVRIK(N,M,A,AT) YORDAMI;

2 DISYAPILAR;

3 A(0:T) (SA,SU TAMSAYI,

DEGER TUR(TAMSAYI, KESIRLI) YAPI;

4 AT(0:T) (SA,SU TAMSAYI,

DEGERAT TUR(DEGER) YAPI;

5 İCYAPILAR;

6 ÜGESAY(1:M) TAMSAYI;

7 YERİNDİSİ(1:M+1) TAMSAYI;

```

8   W, A DİZİSİ ÜZERİNDE GÖSTERGE, SINIR(1:T);
9   V, YERİNDİSİ ÜZERİNDE GÖSTERGE, SINIR(1:M+1);
10  KOMUTLAR;
11  /* DGESAY İÇERİĞİ OLUŞTURULUYOR */
12  DGESAY(*) := 0;
13  SAYARAK YİNELE(W := 1, W > A(0).DEGER, W := W + 1);
14  [ DGESAY (A(W).SU := DGESAY ( A(W).SU + 1 ; ]
15  /* DGESAY İÇERİKLERİ OLUŞTU, ŞİMDİ EKLEMELİ */
16  /* OLARAK YERİNDİSLERİ BULUNUYOR */
17  YERİNDİSİ (1) := 1;
18  SAYARAK YİNELE ( V := 2, V > M + 1, V := V + 1 );
19  [ YERİNDİSİ(V) := YERİNDİSİ(V-1) + DGESAY(V-1) ]
20  /* AT DİZİSİ ÜZERİNDE HER SATIRIN BAŞLANGIÇ */
21  /* NOKTALARI BULUNDU */
22  SAYARAK YİNELE (W := 1, W > A(0).DEGER, W := W + 1);
23  [ K := YERİNDİSİ ( A(W).SU );
24  YERİNDİSİ ( A(W).SU ) := YERİNDİSİ ( A(W).SU ) + 1;
25  AT(K).SA := A(W).SU;
26  AT(K).SU := A(W).SA;
27  AT(K).DEGER := AT(W).DEGER; ]
28  AT(0).SA := A(0).SU; AT(0).SU := A(0).SA;
29  AT(0).DEGERAT := A(0).DEGER;
30  SEYREKMDEVREK BİTTİ;

```

Yukarıda oluşturulan algoritmanın birinci aşaması $k+m$ kez ikinci aşamasıda k kez işlenmektedir. Buna göre algoritma karmaşıklığı

$c.O(m+2k)$ olarak belirlenir. k değeri sıfırdan farklı matris öğelerinin sayısıdır. Buna göre $k \leq n.m$ 'dir. Pratikte seyrek matris gösterimi seçildiğine göre $k \leq 1/2 n.m$ 'dir. Bu nedenle $c.O(m + n.m)$ bu algoritmanın algoritma karmaşıklığı olarak belirlenir. Algoritma karmaşıklığı normal matris gösterimine göre aynı derecededir.

Öte yandan normal matris gösteriminde devrik matris algoritmasının algoritma etkinliği daima $c.O(n.m)$ 'dir. Bazı öğelerin sıfır oluşu nedeni ile işlem zamanı kazancı yoktur. Oluşturduğumuz üçlüler ile seyrek matris gösterimi ve yukarıdaki algoritma ile algoritma etkinliği

$$c.O(m + 2.k) \text{ ve } k \leq \frac{1}{2} n.m$$

Bu kesimde son oluşturacağımız algoritma iki seyrek matrisin toplamını bulan algoritmadır. A ve B toplanacak matrisler, C ise toplamı içerecek matris olarak belirlensin. PA , PB ve PC 'yi bu matrisleri oluşturan üçlüler dizisi üzerinde gösterge olarak kullanalım. Algoritmaya başlamadan önce dikkat etmemiz gereken bir başka nokta da KC , C matrisindeki eleman sayısını gösteriyorsa $KC \leq KA + KB$ ilişkisidir. Öteki deyişle toplama sırasında ters işaretli aynı mutlak değerli bazı terimler sıfır toplam verip yok olabilirler.

```

1 SEYREKMTOPLA(A,B,C) MANTIKSAL YORDAMI;
2
3     /* BU YORDAM A VE B MATRİSLERİ TOPLAMA İÇİN BOYUT      */
4     /* KÖŞÜLÜNÜ SAĞLAMİYOR İSE YANLIŞ, AKSİ HALDE DOĞRU    */
5     /* DEĞERİNİ DÖNDÜRÜR.                                     */
6
7 DISYAPILAR;
8
9     A(0 : TA)(SA,SU TAMSAYI,
10        DEĞER TUR(TAMSAYI, KESİRLİ)) YAPI ;
11
12     B(0 : TB)(SA,SU TAMSAYI,
13        DEĞER TUR(A(*).DEĞER)) YAPI ;
14
15     C(0 : TC)(SA,SU TAMSAYI,
16        DEĞER TUR(A(*).DEĞER)) YAPI ;
17
18 İÇYAPILAR;
19
20     PA A ÜZERİNDE GÖSTERGE, SINIR (0 : TA);
21     PB B ÜZERİNDE GÖSTERGE, SINIR (0 : TB);
22     PC C ÜZERİNDE GÖSTERGE, SINIR (0 : TC);
23
24     TA, TB, TC TAMSAYI, DEĞER = 500;
25
26 KOMUTLAR;
27
28     EĞER (A(0).SA = B(0).SA VE A(0).SU = B(0).SU) DEĞİL
29     İSE [SEYREKMTOPLA := YANLIŞ;
30
31         'BOYUT UYUŞMAZLIĞI' YAZ;
32
33         DÖN ]
34
35     PA := 1; PB := 1; PC := 1;
36
37     ÖNCE PA > A(0).DEĞER VE PB > B(0).DEĞER SİNA DEĞİLSE YİNELE
38
39     [ SEÇENEK
40
41         BAŞLA
42
43         PA > A(0).DEĞER : [BDENCYETASI (PB,PC) ÇAĞIR ];

```



```

19  PB > B(0).DEGER : [ADANCYETASI (PA,PC) ÇAGIR ] ;
20  A(PA).SA < B(PB).SA : [ ADANCYETASI (PA,PC) ÇAGIR ] ;
21  A(PA).SA > B(PB).SA : [ BDENCYETASI (PB,PC) ÇAGIR ] ;
22  YOKSA : [ /* BU DURUMDA DATIR DEGERLERİ EŞİTTİR */
23  SEÇENEK 1 = B(PB).DEGER ;
24  BAŞLA 1 ;
25  A(PA).SU < B(PB).SU : [ADANCYETASI (PA,PC) ÇAGIR] ;
26  A(PA).SU > B(PB).SU : [BDENCYETASI (PB,PC) ÇAGIR] ;
27  YOKSA : [ /* SUTUN DEGERLERİ EŞİTTİR */
28  ADANCYETASI (PA, PC) TOPLAM := A(PA).DEGER + B(PB) DEGER ;
29  SONUTUN : EGER TOPLAM = 0
30  C(PC).SA := A(PA).SA ISE [ C(PC).SA := A(PA).SA ;
31  C(PC).SU := A(PA).SU C(PC).SU := A(PA).SU ;
32  C(PC).DEGER := A(PA).DEGER C(PC).DEGER := TOPLAM ;
33  PC := PC + 1 ]
34  PA := PA + 1 ; PB := PB + 1 ]
35  BİTTİ ;
36  BİTTİ ;
37  EGER PC > TC
38  ISE [ 'AÇILAN YER YETERSİZ' YAZ ;
39  DÖN ]
40  ] /* Burası asıl döngünün sonu */
41  SEYREKMTOPLA := DOGRU ;
42  SEYREKMTOPLA BİTTİ ;

```

Simdi bu algoritmanın alt yordam olarak kullandığı iki kısa algoritmayı yazalım.

1 BDENCYETASI (PB,PC) YORDAMI;

2 KOMUTLAR ;

3 C(PC).SA := B(PB).SA ;

4 C(PC).SU := B(PB).SU ;

5 C(PC).DEGER := B(PB).DEGER ;

6 PB := PB + 1 ;

7 PC := PC + 1 ;

8 BDENCYETASI BITTI ;

1 ADANCYETASI (PA,PC) YORDAMI ;

2 KOMUTLAR ;

3 C(PC).SA := A(PA).SA ;

4 C(PC).SU := A(PA).SU ;

5 C(PC).DEGER := A(PA).DEGER ;

6 PA := PA + 1 ;

7 PC := PC + 1 ;

8 ADANCYETASI BITTI;

Matris toplama için oluşturduğumuz bu algoritmanın algoritma karmaşıklığı ve algoritma etkinliği $c.O(k_a + k_b)$ olacaktır. $k_a \leq n.m$ ve $k_b \leq n.m$ ' dir. Ancak öğelerin eşleşmesi nedeniyle üst sınırın yinede $c.O(n.m)$ olduğu görülür. Sıfırdan farklı olan öğelerin sayısı azaldıkça algoritma etkinliği de olumlu yönde etkilendir. Normal matris gösterimi ile matris toplama işleminde bu etkinlik artışı söz konusu değildir.

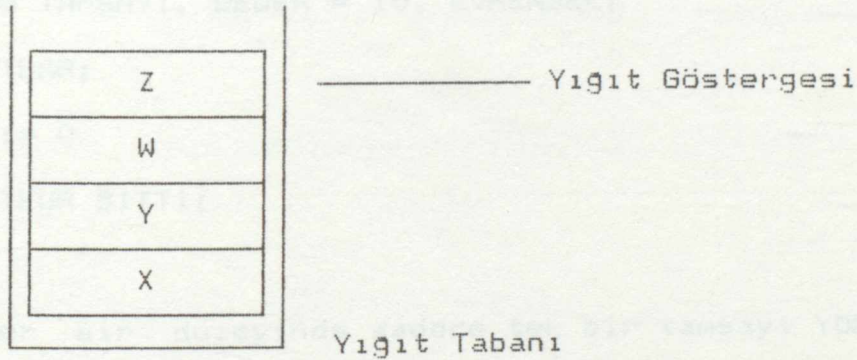
2.3 BAĞLAÇSIZ YIGIT

Veri yapılarında olduğu gibi gerçek yaşantımızda da enson yerleştirdiğimiz birimin en önce alındığı yapılar vardır.

- i) Bir ucu kapalı lokomotif manevra yolu
- ii) Sarjörlü tabanca mermileri
- iii) Üst üste yığılan kafeterya tepsileri gibi

Bu tür yapılar bilişim dalında 'Son Giren Önce Çıkar (Last In First out)'olarak alınır. Bu yapılara 'Yığın (Stack)' adı verilir. Şimdi bu veri yapılarını ve ilişkin algoritmaları inceleyelim.

2.3.1 YIGIT VERİ YAPISI TANIMI



Sekil 20 Yığın Veri Yapısı Gösterimi

Yığın veri yapısının tanımı soyut düzeyde sıfır ile sonsuz arasında 1'er ögelik adımlarla artan ve eksilen bir yığın göstergesi ile yönetilen bilgi alanlarıdır. Bir anda yalnız bir yığın ögesine erişilebilir, bu da enson olandır. Boş olan yığıttan eleman çıkarmaya (okumaya) çalışmak alttaşma (underflow) oluşturur.

2.3.2 YIGIT VERİ YAPISININ DİZİ ÜZERİNDE GÖSTERİMİ

Yığıt veri yapısı bilgisayar belleğinin bir bölgesini kaplayacaktır. Bu durumda yığıtın kullanımı için atanmış bellek yerinin başlangıç ve sonucu işaretlenmiş olmalıdır. Bu durumda yığıtın tüm elemanları dolu iken yeni eleman eklemeye çalışmak üsttaşma (overflow) oluşturur. Şimdi önce bir dizi üzerinde oluşturulmuş tek bir yığıt (stack) ile çalışalım.

```

1 YIGITKUR YORDAMI ;
2 İÇYAPILAR;
3   Y(10)(YÖN) TAMSAYI, EVRENSEL;
4   S TAMSAYI , EVRENSEL;
5   TOS TAMSAYI, DEĞER = 10, EVRENSEL;
6 KOMUTLAR;
7   S := 0
8 YIGITKUR BİTTİ;

```

Yukarıda her bir düzeyinde sadece tek bir tamsayı YÖN değişkeni bulunan 10 elemanlı bir yığıt tanımı verilmiştir. Aşağıda ise her düzeyinde 3 X 3 'lük bir matris ve SONKURAL adlı bir tamsayı değişken bulunan bir yapı verilmiştir.

```

1 YIGITKUR YORDAMI;
2 İÇYAPILAR;
3   Y(20) (DURUM (3,3) TAMSAYI;
4           SONKURAL TAMSAYI) YAPI, EVRENSEL;

```



```

5  S TAMSAYI, EVRENSEL;
6  TOS TAMSAYI, DEGER = 20, EVRENSEL;
7  KOMUTLAR;
8  S := 0 ;
9  YIGITKUR BITTI;

```

2.3.3 YIGIT'A VE ÖGELERİNE ERİŞİM

Eğer bir adet yığıtımız var ise yığıtın denetim grubuna erişim YIGIYAEKLE, YIGITIOKU algoritmalarında yığıtın üzerinde kurulmuş olduğu dizinin adı geçtiği ve kullandığımız programlama dili bu eşleştirmeyi yaptığı için programcı tarafından farkedilmeden halledilir. Eğer birden fazla yığıt var ise söz konusu yığıtın başlangıcına erişim için yığıt seçimi kesimi olması gereklidir.

Yığıttaki temel işlevlerimiz YIGITAEKLE, YIGITIOKU; YIGITTANAL işlemlerimizdir.

```

1  YIGITAEKLE(YENI) MANTIKSAL YORDAMI;
2  DISYAPILAR;
3  Y(*), S, TOS;
4  KOMUTLAR;
5  EGER S ≠ TOS
6    ISE [ S := S + 1 ;
7          Y(S) := YENI;
8          YIGITAEKLE := DOGRU ]
9  DEĞİLSE [ YIGITAEKLE := YANLIS ]
10 YIGITAEKLE BITTI;

```

Burada yığıtın üst düzey dil ile desteklenen bir dizi veri yapısı üzerine kurulmuş olması nedeniyle 1 arttırılan S göstergesinin niteliğine dikkat edilmelidir. Algoritmik (mantıksal) düzeyde 1 birim arttırılan gösterge fiziksel düzeyde ilk tanımladığımız öğesi bir tek YDN olan yığıt için fiziksel düzeyde büyük olasılıkla 2 byte, daha sonra tanımladığımız yığıt için ise büyük olasılıkla 20 byte ($3 \times 3 \times 2 + 2$) artış gerektirmektedir.

1 YIGITOKU (OKUNAN) MANTIKSAL YORDAMI;

2 DIŞYAPILAR;

3 Y(*), S, TOS ;

4 KOMUTLAR;

5 EGER S = 0

6 ISE [YIGITOKU := YANLIŞ ;

7 'YIGIT BOS, OKUNAMADI' YAZ]

8 DEĞİLSE [OKUNAN := Y(S)]

9 YIGITOKU BİTTİ ;

1 YIGITTANAL (ALINAN) MANTIKSAL YORDAMI ;

2 DIŞYAPILAR;

3 Y(*), S, TOS ;

4 KOMUTLAR;

5 EGER S = 0

6 ISE [YIGITTANAL := YANLIŞ]

7 DEĞİLSE [ALINAN := Y(S) ;


```

8           S:= S - 1]
9 YIGITTANAL BİTTİ;

```

Yukarıdaki algoritmaların her ikisinde de alttaşıma olsılığı vardır ve dikkate alınmıştır. Yığıttan - oku yordamı okuduğu elemanı silmemekte, Yığıttanal ise silmektedir.

2.4 ÖZYİNELİ ALGORİTMA

Bazı sorunlar veya tanımlar özyineli (recursive) nitelik gösterirler. Özyineli nitelik, bir tanımlı yaparken tanımlanan problemin daha az bir karmaşıklık düzeyinde tanımlayanlar içinde yer almasıdır. Bazı özyineli sorunlar yaygın biçimde bilindiği halde bunlara özyineli çözümler bulmak alışkanlık olmamıştır.

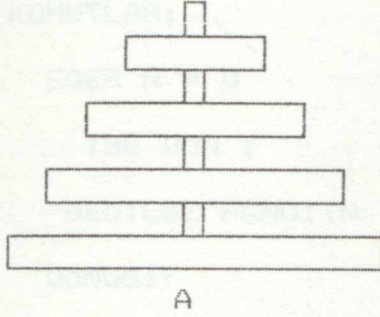
Örneğin: FAKTÖRİYEL TANIMI aşağıda görüldüğü gibi

$$N! = N.(N - 1)!$$

$$0! = 1$$

Özyineli bir tanımdır. Ancak faktöriyel bulmak için özyineli program yazma şu nedenlerle alışkanlık olmamıştır.

- i) Faktöriyel hesabı gerektirebilecek matematiksel işlemlerde çok kullanılan FORTRAN dili özyineli altyordam kullanmaz
- ii) Özyineli bir programın işletim zamanı davranışını anlamak daha üst düzeyde bilgi gerektirir ve adımların dikkatlice izlenmesi gerekir. Ancak bazı sorunların özyineli özellik kullanmaksızın çözümü güç anlaşılır hatta çözülemez nitelikte olabilir.



Sekil 21 Hanoi Kuleleri Problemi

A kulesindeki değişik çaplı dört disk, geniş çaplı disk hiçbir zaman daha küçüğünün üzerine oturtmaksızın B kulesine nasıl taşırırsınız?

```

1 HANOIKULELERI YORDAMI;
2 KOMUTLAR;
3   ( M ) OKU; /* kaç disk olduğunu oku */
4   ( M, 'DISK İLE HANOİ KULELERİ PROBLEMİ ÇÖZÜMÜ' ) YAZ;
5   HANOİ(M, 'A', 'B', 'C') ÇAGIR;
6   ANA;
7 HANOIKULELERİ BİTTİ;

```

İlk yordam problemin boyutunu okudu ve sorunu çözecek yordamı uyardı, şimdi özyineli programı inceleyelim:

```

1 HANOİ (N, KAYNAK, ARA, AMAC) YORDAMI;
2 İÇYAPILAR;

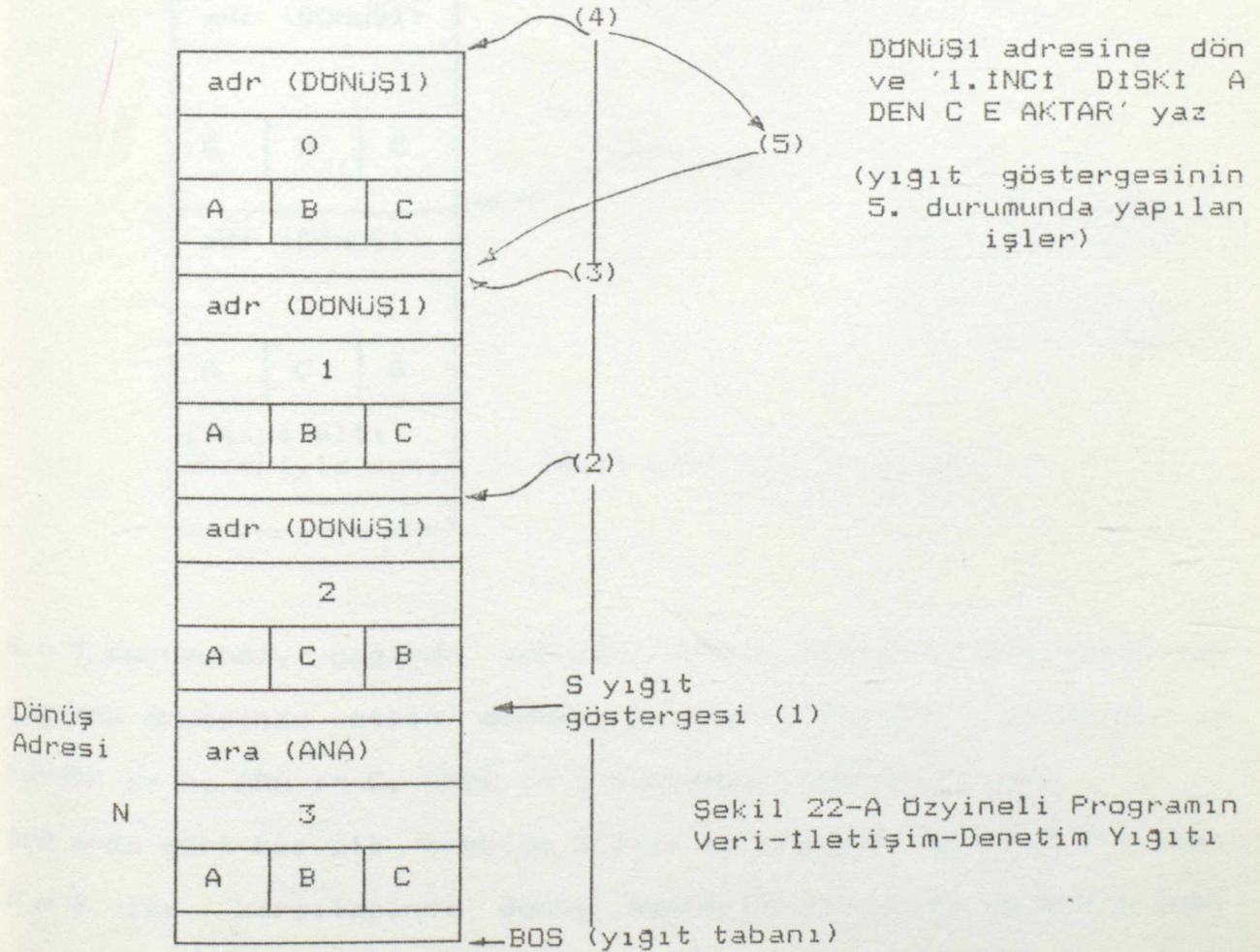
```



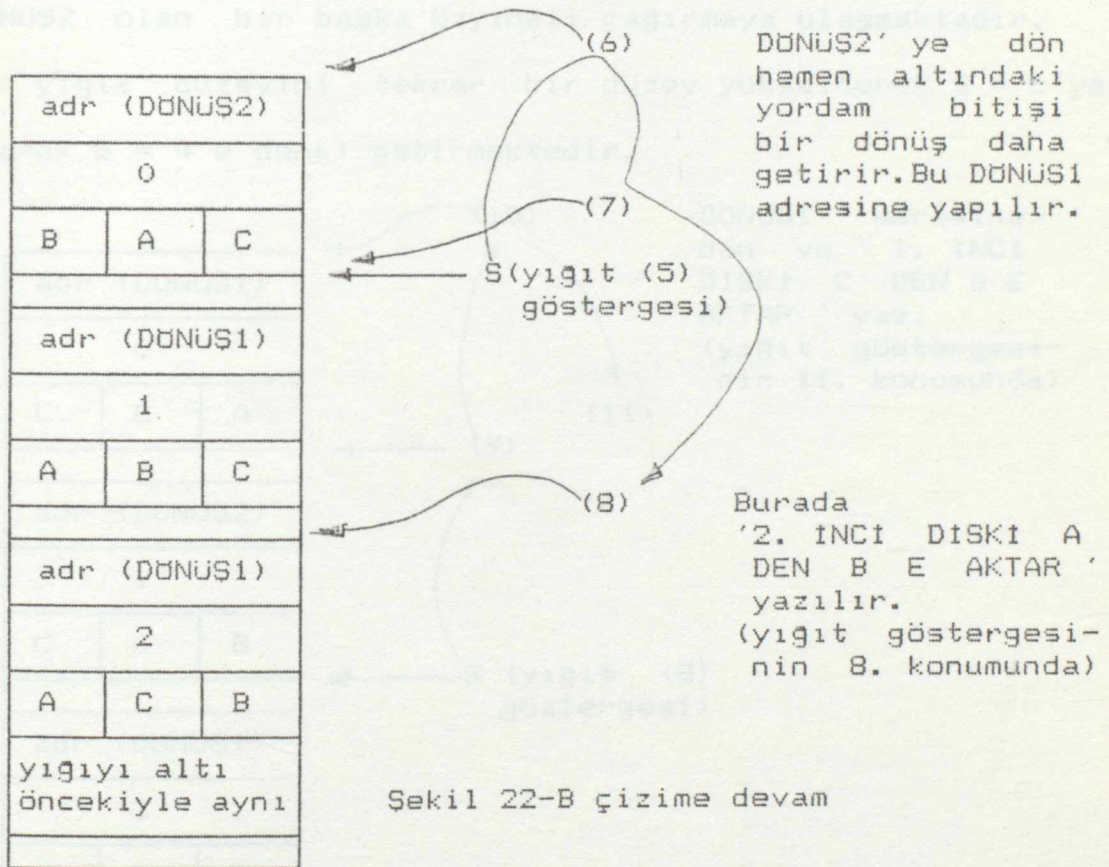
```

3  N TAMSAYI;
4  KAYNAK, ARA, AMAC KARAKTER(1);
5  KOMUTLAR;
6  EGER N = 0
7      ISE DON ;
8  DEGI LSE HANOI(N - 1, KAYNAK, AMAC, ARA) CAGIR ;
9  DONUS1:
10 (N; '1. INCI DISKI'; KAYNAK; 'DEN'; AMAC; 'E AKTAR') YAZ,
11 HANOI(N - 1, ARA, KAYNAK, AMAC) CAGIR;
12 DONUS2:
13 HANOI BITTI;

```



Özyineli programın veri - iletişim - denetim yığıtı adlı çizimde üzerinde üç disk bulunan Hanoi kuleleri bulmacası için programın işletim zamanı davranışı özetlenmektedir. İlk çizimde yığıttaki ilk beş hareket görülmektedir. $S = 1$ durumunun ana program içindeki parametrelerle oluştuğuna $KAYNAK := A$, $ARA := B$, $AMAC := C$ eşlemesine dikkat edilmelidir.



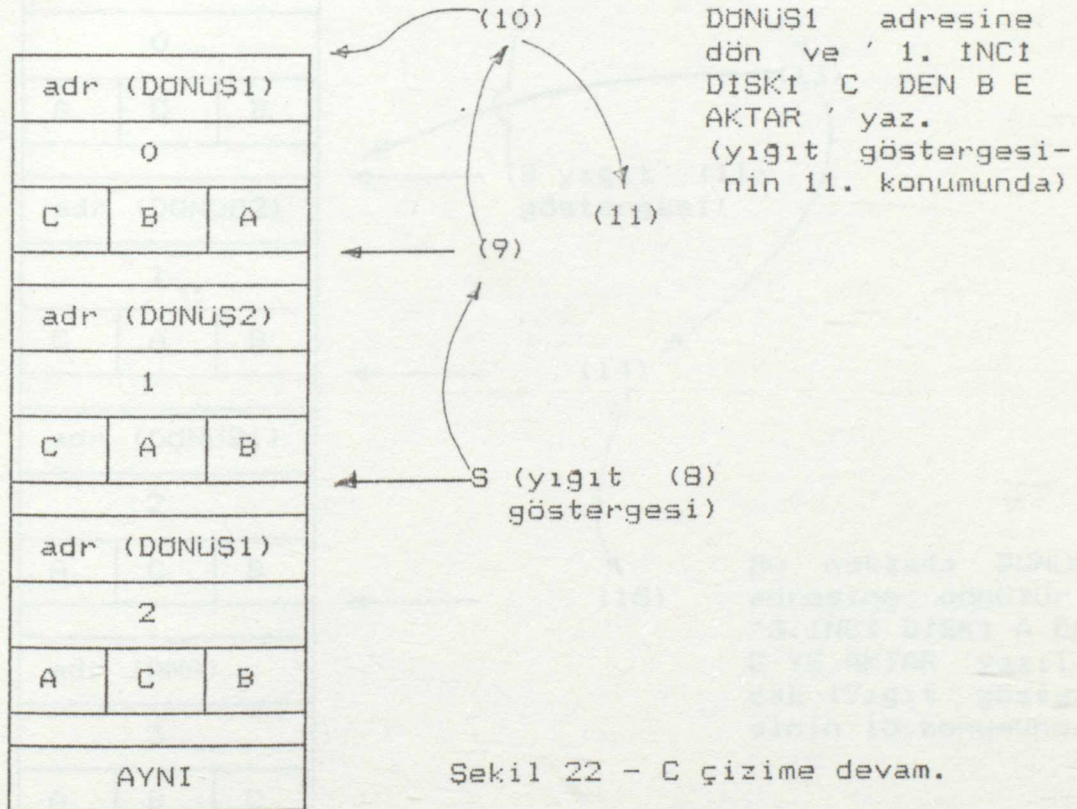
$S = 2$ durumunda, çağırma yerinin DÖNÜŞ1 öncesi olduğu yerde ise çağırma öncesinde etkin durumda bulunan düzey $S = 1$ düzeyidir ve $KAYNAK := A$, $ARA := C$, $AMAC := B$ eşleşmesi sağlanmaktadır.

Ard arda dört kez alt problem üreten bu çağırma işlemi $S = 4$ iken $N = 0$ ile karşılaşınca dönüş aşamasına girmekte ve $S = 4$ iken

bulunan dönüş adresi DÖNÜS1'e yığıttan bir çağırıcı düzeyini silerek (yani $S = 5$ düzeyine inerek) ulaşmaktadır. Konum yönünden $S = 3$ 'e eşit olan $S = 5$ yığıt göstergesi durumunda yordamın işletiminde yazma komutuna gelindiğinden kullanıcıya;

1.İNCİ DİSKİ A'DAN C'YE AKTAR

direktifi gönderilmekte ve bu yazma komutunun altındaki dönüş adresi DÖNÜS2 olan bir başka özyineli çağırmaya ulaşmaktadır. Bu çağırıcı yığıt düzeyini tekrar bir düzey yükselterek $S = 6$ 'ya (konum olarak $S = 4$ 'e denk) getirmektedir.



Şekil 22 - C çizime devam.

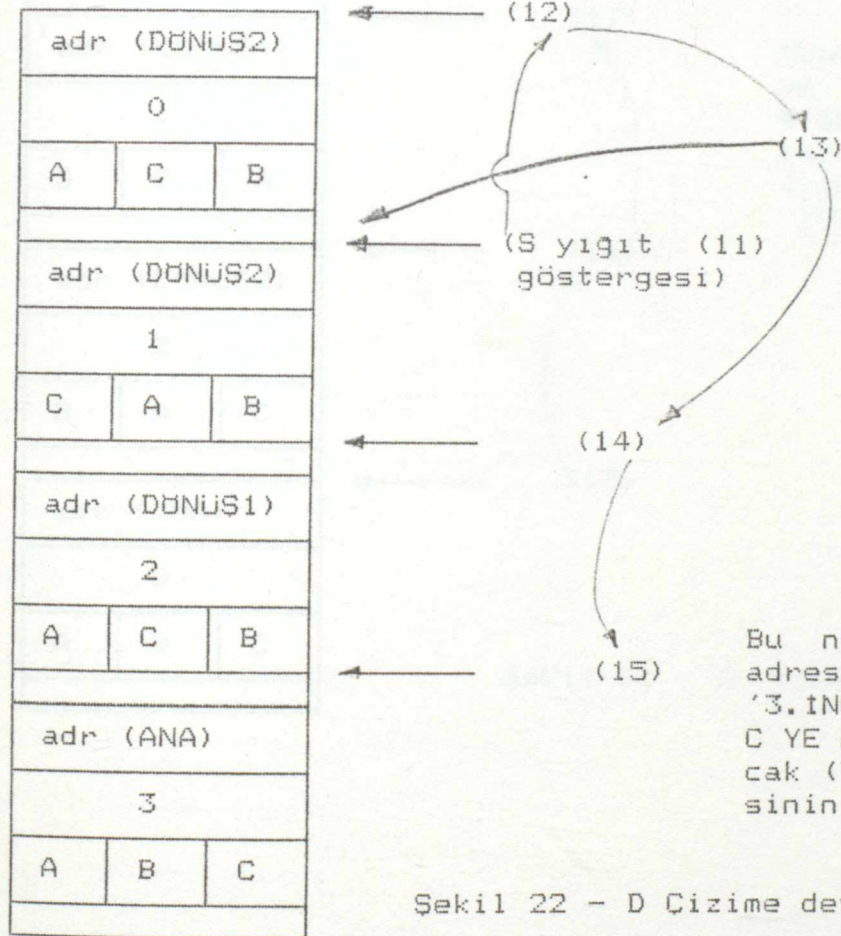
$S = 6$ düzeyinde $N = 0$ 'ın fark edilmesiyle başlayan zincirleme iki dönüş vardır. Şimdi bunları inceleyelim:

i) $S = 6$ düzeyindeki dönüş adresi DÖNÜS2' yi alarak yapılır ve

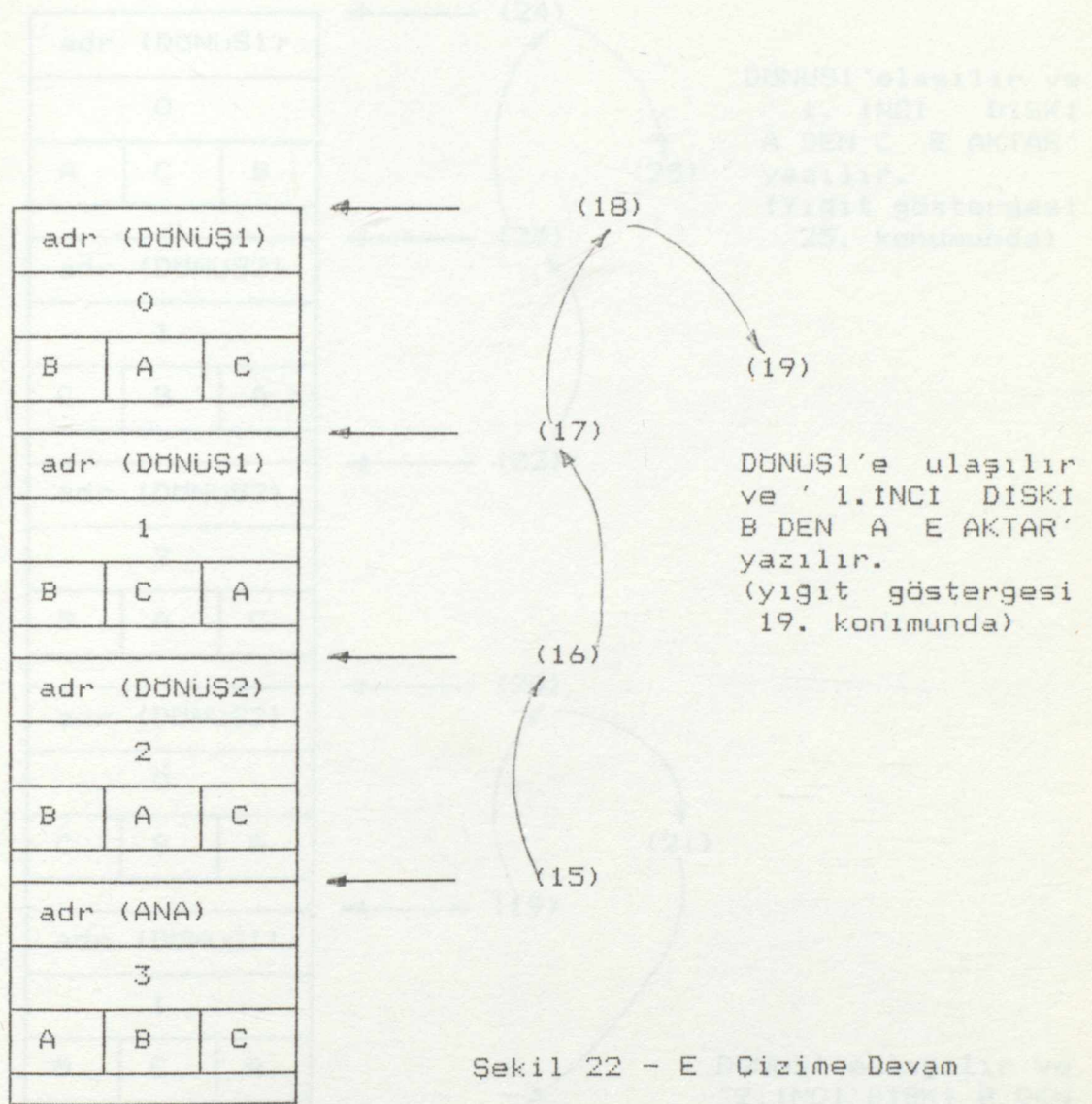
- ii) $S = 7$ konumuna geçilir. ($S = 5$ ve $S = 3$ ile eşdeğer konum) ulaşır. Bu noktada ulaşılan komutta $S = 7$ çağırılma düzeyinde kullanıcıya

2.İNCİ DİSKİ A'DAN B'YE AKTAR

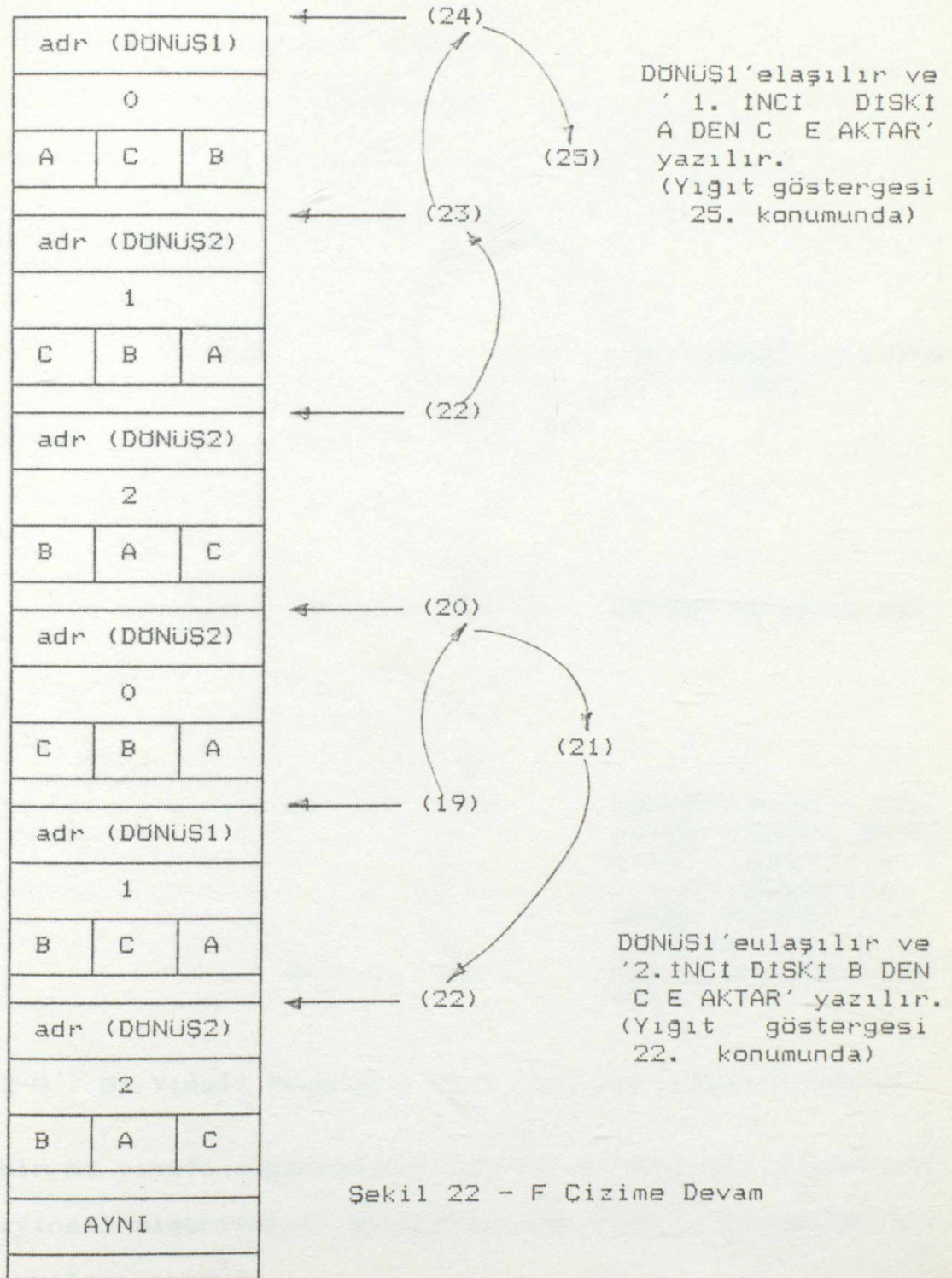
mesajı verilir.

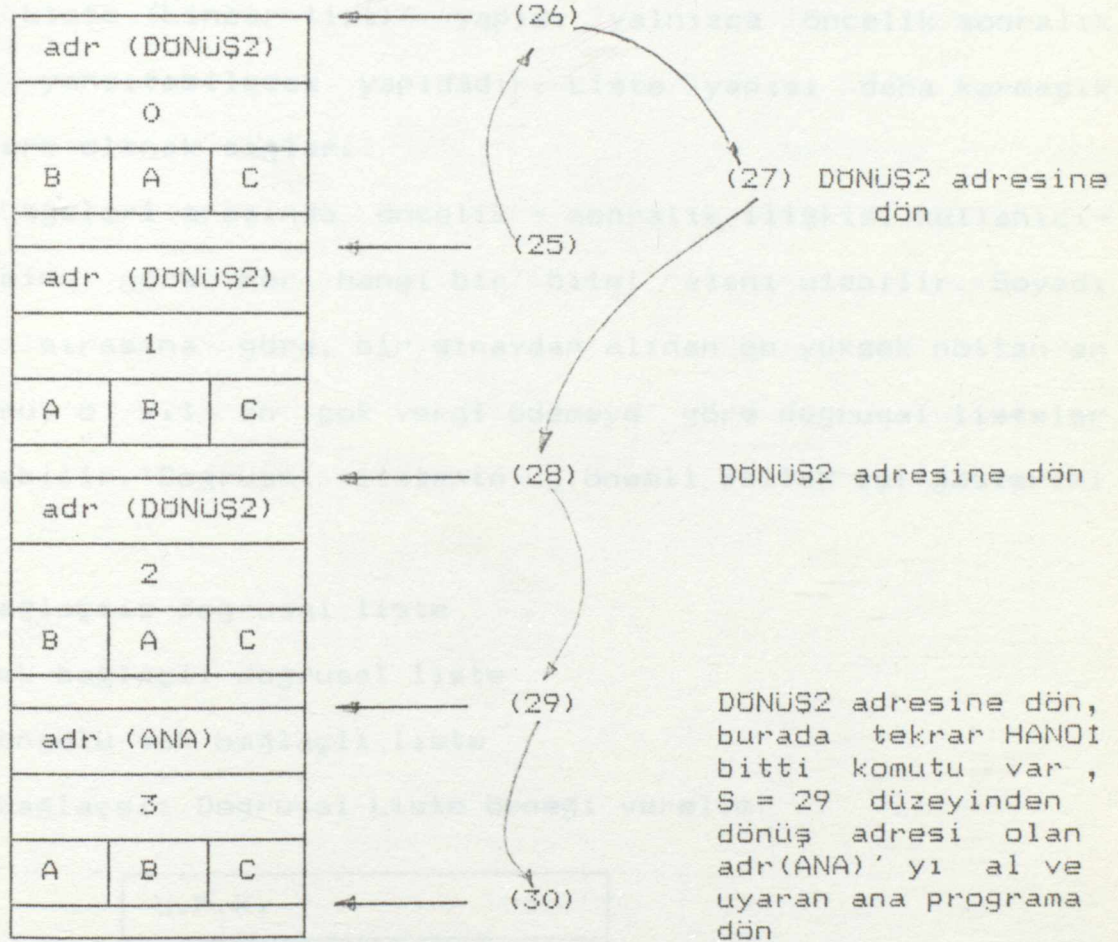


Şekil 22 - D Çizime devam



Sekil 22 - E Çizime Devam





Sekil 2-6 Öz Yineli Programın Veri İletişim- Denetim Yığıtı

Yukarıda bir öz yineli algoritmanın işletim sırasındaki dinamiğini gördük. Özyineli algoritmalar genellikle çok kısa algoritmalardır, ancak sınanmaları zordur.

3 LİSTE, KUYRUK, ÇİZGE VE AĞAC VERİ YAPILARI

3.1 TEK BAĞLAÇLI DOĞRUSAL LİSTE

'Liste (List)' sözcüğü aralarında bir biçimde öncelik sonralık yada altlık üstlük ilişkisi bulunan veri ögeleri arasında kurulur. 'Doğrusal Liste (Linear list)' yapısı yalnızca öncelik sonralık ilişkisini yansıtabilecek yapıdadır. Liste yapısı daha karmaşık gösterimlere olanak sağlar.

Veri ögeleri arasında öncelik - sonralık ilişkisi kullanıcının seçimine göre her hangi bir bilgi alanı olabilir. Soyadı alfabetik sırasına göre, bir sınavdan alınan en yüksek nottan en azına doğru, o yıl en çok vergi ödemeye göre doğrusal listeler oluşturulabilir. Doğrusal listenin üç önemli bellek içi gösterimi vardır.

- i) Bağlaçsız doğrusal liste
- ii) Tek bağlaçlı doğrusal liste
- iii) Döngülü tek bağlaçlı liste

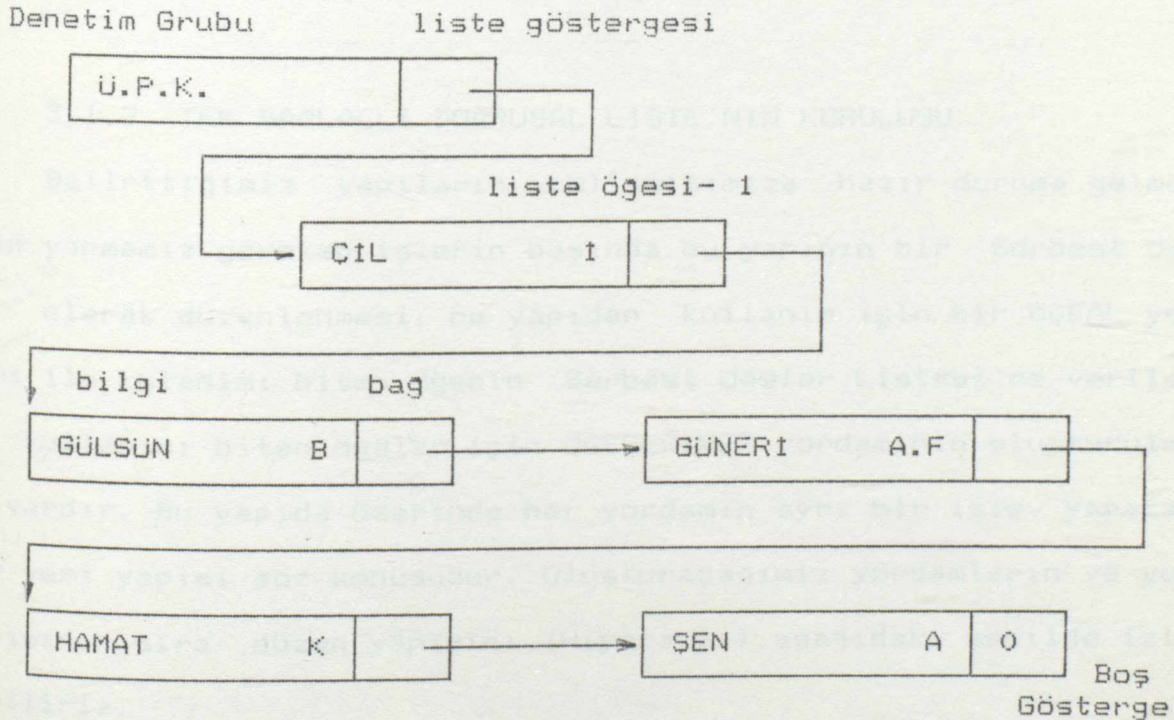
Önce bir Bağlaçsız Doğrusal Liste örneği verelim:

Ü.P.K.	
ÇİL	İ
GÜLSÜN	B
GÜNERİ	A.F
HAMAT	K
SEN	A
TÜRKER	A.K

Yukarıda bir yüksek lisans dersinin soyadı alfabetik sırasında öğrenci listesini görüyoruz. Simdi sırasıyla şu işlemleri yapmaya çalışalım:

- i) A.K. TURKER bu dersi bıraksın
- ii) E.SARITAŞ bu derse sonradan kaydolsun

Bu yapının güçlüğü hemen görülecektir. Bir silme yapılsa silinecek öğeden sonra gelen tüm öğelerin yukarıya doğru birer hane yer değiştirmesini, bir ekleme yapılması ekleme yapılacak noktadan sonraki tüm öğelerin birer aşağıya kaymasını gerektirir. Bu gösterim biçimi yani doğrusal sıralı listeyi oluşturmak için altyapı olarak dizinin kullanımı, doğrusal sıralı listenin olası gösterimlerinden biridir. Ancak yukarıda belirtilen nedenle uygulamada sık kullanılmaz. İzleyen bölümde tek bağlaçlı doğrusal listeyi inceleyeceğiz.



Sekil 23 Tek Bağlaçlı Doğrusal Liste Örneği

3.1.1 TEK BAĞLAÇLI DOĞRUSAL LİSTE ÖRNEĞİ

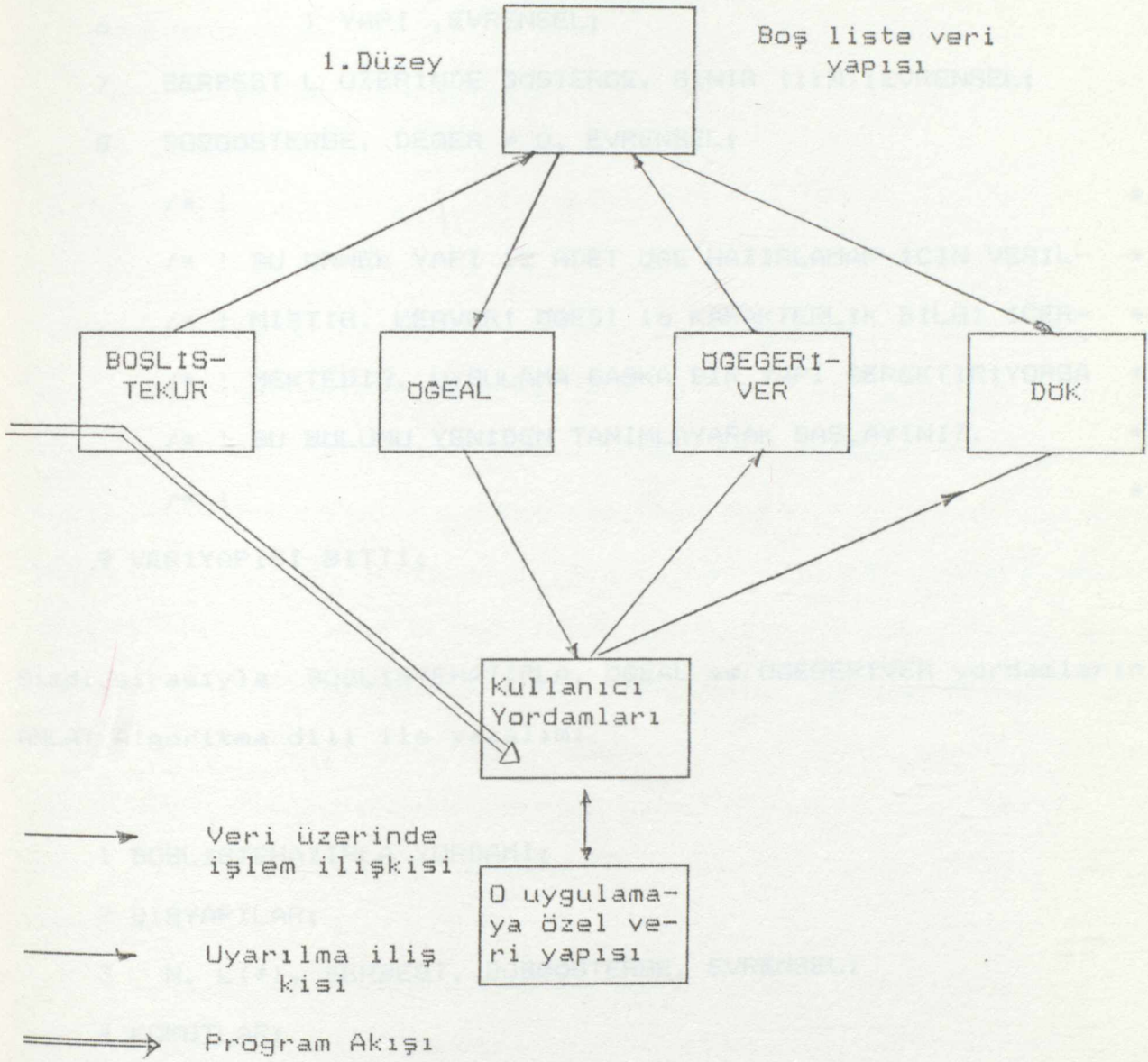
Tek bağlaçlı doğrusal liste, öğelerinin arasındaki mantıksal ilişkiye göre bir sonraki öğenin bellekte yerleştiği yerin bir gösterge ile gösterildiği yapıdır. Kavramsal yapısı için örnek Şekil 23 ' te verilmiştir.

Örnekteki yapıda ' Veri Yapısı Denetim Grubu olarak ' Liste adı 'Liste göstergesi' kesimleri görülmektedir. Örneğimiz üzerinde açıklamaya devam edersek, üniversitenin öğretime açılışı sırasında önce o yarı yılda hangi derslerin verileceği belirtilir, ancak hiçbir öğrenci o derse kayıtlı değildir. Öteki deyişle bir ' boş liste ' söz konusudur. Bu durum liste göstergesi kesimine boş gösterge anlamında '0' işaretini koyarak belirtilir.

Liste öğeleri ise bir yada birçok bilgi alanının birleşmesinden oluşan bir bilgi alanı ile kendisinden sonra gelen öğeyi işaret edecek bir bağ alanından oluşmaktadır.

3.1.2 TEK BAĞLAÇLI DOĞRUSAL LİSTE 'NİN KURULUŞU

Belirttiğimiz yapıların kullanımımıza hazır duruma gelmesi için yapmamız gereken işlerin başında bu yapının bir 'Serbest Öğeler' olarak düzenlenmesi, bu yapıdan kullanım için bir DGEAL yordamı ile kullanımı biten öğenin 'Serbest Öğeler Listesi'ne verilmesi, kullanımı biten öğeler için DGEGERIVER yordamının oluşturulması vardır. Bu yapıda üzerinde her yordamın ayrı bir işlev yapacağı bir veri yapısı söz konusudur. Oluşturacağımız yordamların ve veri yapısının sıra düzen yapısını (Hiyerarşi) aşağıdaki şekilde izleyebiliriz.



Sekil 24 Tek Bağlı Liste İçin Destek Yordamları

- 1 VERİYAPISI YORDAMI;
- 2 İÇYAPILAR;
- 3 N TAMSAYI, EVRENSEL, DEGER = 12;
- 4 L(1:N) (BİLGİ KARAKTER(16),

```

5          BAG L ÜZERİNDE GÖSTERGE, SINIR (1:N)
6          ) YAPI ,EVRENSEL;
7  SERBEST L ÜZERİNDE GÖSTERGE, SINIR (1:N);EVRENSEL;
8  BOŞGÖSTERGE, DEĞER = 0, EVRENSEL;

  /* !                                                                    */
  /* ! BU ÖRNEK YAPI 12 ADET ÖGE HAZIRLAMAK İÇİN VERİL- */
  /* ! MİSTİR. HERVERİ ÖGESİ 16 KARAKTERLİK BİLGİ İÇER- */
  /* ! MEKTEDİR. UYGULAMA BAŞKA BİR YAPI GEREKTİRİYORSA */
  /* ! BU BÖLÜMÜ YENİDEN TANIMLAYARAK BAŞLAYINIZ.      */
  /* !                                                                    */

9 VERİYAPISI BİTTİ;

```

Şimdi sırasıyla BOŞLISTEHAZIRLA, ÖGEAL ve ÖGEGERİVER yordamlarını ANLAT Algoritma dili ile yazalım:

```

1 BOŞLISTEHAZIRLA YORDAMI;
2 DİŞYAPILAR;
3  N, L(*), SERBEST, BOŞGÖSTERGE, EVRENSEL;
4 KOMUTLAR;
5  SAYARAK YİNELE (I := 1, I = N, I := I + 1);
6  [ L(I).BAG := I + 1 ;]

  /*                                                                    */
  /* N'İNCİ ÖGE DİŞİNDAKİLER ARDISIK ÖGEYİ GÖSTERECEK */
  /* BİÇİMDE BAGLANDI. SON ÖGENİN BAG ALANINA BOS GÖS- */
  /* TERGE YERLEŞTİR, SERBEST GÖSTERGESİNİ ZİNCİRİN BAS-*/
  /* LANGICINI GÖSTERECEK BİÇİMDE BELİRLE */

```



```

/*
7 BOŞGÜSTERGE := 0;
8 L(N).BAĞ := BOŞGÜSTERGE;
9 SERBEST := 1;
10 BOSLISTEHAZIRLA BİTTİ;

```

Tek bağlı liste kullanımında bir kez çalıştırılacak olan bu yordamın algoritma karmaşıklığı 6.komutun n kez işleyişi nedeniyle $O(N)$ ' dir. DGEAL yordamı, serbest ögeler listesinden bir öge alarak kullanıcı yordamına verecektir. Bu ögenin kullanıcının denetimindeki bir P göstergesi ile işaret edildiğini kabul edelim. DGEAL'ma işlemi eğer alınabilecek öge varsa bu ögeyi P ile işaret edilecek biçimde kullanıma sunar. Geri kalan serbest ögeler listenin doğru biçimde bağlanmış durumda bırakılması garanti edilme-lidir. Eger alınabilecek öge yoksa bu durumda işaret edilir. Bu durum (öge yok) işaret edildikten sonra yapılacak işlemin seçimi önemlidir:

- i) Tüm programın işletimi bu noktada durdurulabilir,
- ii) öge isteyen yordama öge verilmediği bilgisi verilir ve denetim çağıran yordama bırakılır.

Aşağıda ikinci seçeneğin benimsendiği bir yordam örneği verilmiştir.

```

1 DGEAL(P) MANTIKSAL YORDAMI;
/* !
/* ! BU DGEAL YORDAMI - ÖGE BULUNMADIĞI DURUMDA
/* ! DENETİMİ ÇAĞIRAN YORDAMA BIRAKIR

```

```

/* !                                     */
2 KOMUTLAR;
3  EGER SERBEST = BOŞGÜSTERGE
4      İSE DGEAL := YANLIS
5      DEĞİLSE [ P := SERBEST;
6                SERBEST := L(SERBEST).BAĞ;
7                DGEAL := DOGRU ]
8 DGEAL BİTTİ;

```

Yukarıdaki örnek için ilk seçenegin benimsendigi durumda da işletimi durdurmadan önce vereceğimiz bilgi değişiklik gösterir. Olası yaklaşımlar şunlardır:

- i) 'SERBEST DGELELER BİTTİ' bildirisi verilir ve işletim durur,
- ii) 'SERBEST DGELELER BİTTİ'

'KULLANICI VERİ YAPISI LİSTE BAŞLIKLARI'

.....

.....

'VERİ YAPISI İÇERİĞİ DOKÜMÜ'

.....

.....

.....

.....

Bu biçimde verilebilir. İkinci tür ayrıntılı bilgi vererek kesiliş, özellikle program sinama aşamasında yada programın eğitim / öğretim amacıyla gerçekleştirilişinde daha önemlidir. İyi programcılığın bu tür program güvenilirliğini ve programın kullanıcıya yardım ediş düzeyini yükselten küçük küçük önlemlerin birleşimi ile oluştuğuda akılda tutulmalıdır.

Bir sonraki yordamımız DGEGERİVER yordamıdır:

```

1 DGEGERİVER(P) YORDAMI;

  /* BU YORDAM P İLE İŞARET EDİLEN ÖGEYİ SERBEST          */
  /* ÖGELER LİSTESİNİN BAŞLANGICINA EKLER                */

2 DİŞYAPILAR;

3   P GÖSTERGE;

4 KOMUTLAR;

5   L(P).BAĞ := SERBEST;

6   SERBEST := P;

7 DGEGERİVER BİTTİ;

```

Bu yordamın uygulanışı ile kullanıcının o uygulamaya özel veri yapısından uygun biçimde daha önceki komutlar ile ayrılmış olan, P ile işaret edilen öge serbest ögeler listesine katılır. DGEAL ve DGEGERİVER yordamları döngü içermeyen algoitma karmaşıklıkları $O(1)$ olan etkin ve kısa yordamlardır.

3.1.3 TEK BAĞLAÇLI DOĞRUSAL LİSTEDE ERİŞİM

i) Tek Bağlaçlı Doğrusal Liste Başlığına Erişim:

Tek bağlaçlı doğrusal listeye 'liste göstergesi' ile erişilir. Eğer liste göstergesi boşgösterge (0) ise hiçbir öge yoktur, değilse liste ögelerine erişim söz konusudur.

ii) Tek Bağlaçlı Doğrusal Liste Ögelerine Erişim:

Tek bağlaçlı doğrusal listede rasgele erişim olanığı yoktur. Bu nedenle erişilecek öge ya listedeki sıra numarası ile belirtilir veya içeriği ile belirtilir. Önce listedeki sıra numarası ile belirtilen durumu, sonra içeriği belirtilen ögeye erişim

algoritmasını yazalım.

```

1  DGEYEERISN(AD,N,X) MANTIKSAL YORDAMI;
2  DIŞYAPILAR;
3  D(*) (BİLGİ KARAKTER, BAĞ GÖSTERGE) YAPI;
4  AD, X, GÖSTERGE;
5  N TAMSAYI;
6  KOMUTLAR;
7  DGEYEERISN := YANLIŞ; I := 0;
8  SAYARAK YİNELE (X := AD, X = 0, X := D(X).BAG;
9    [ I := I + 1;
10     EĞER I = N
11       İSE [ DGEYEERISN := DOĞRU; DÖN ]
12    ] /* ARANAN DGE BULUNAMADI */
13 DGEYEERISN := YANLIŞ;
14 ('VERİLEN LİSTEDE'; N'ADET DGE YOK';
15 'LİSTE ADI = ';AD) YAZ;
16 DGEYEERIS BİTTİ;

```

```

1  DGEYEERISBİLGİ(AD,ARA,X) MANTIKSAL YORDAMI;
2  DIŞYAPILAR;
3  D(*) (BİLGİ KARAKTER, BAĞ GÖSTERGE
4  AD, X GÖSTERGE ;
5  ARA TÜR(D(*).BİLGİ); /* ÖNCEKİ ÖRNEĞE GÖRE */
6  KOMUTLAR;
7  DGEYEERISBİLGİ := YANLIŞ;

```



```

8  EGER AD =0 DEĞİLSE [ 'AD KALMADI' YAZ;
9      İSE X := 0;
10     DEĞİLSE [ X := AD ;
11         ÖNCE X = 0 SINA DEĞİLSE YINELE;
12         [ EGER ARA = D(X).BİLGİ
13             İSE [ DGEYEERİSBİLGİ := DOGRU ; DÖN ]
14             X := D(X).BAĞ;
15         ]
16         ('SIRASIZ TEK BAĞLI LİSTEDE ARANAN AD BULUNAMADI')YAZ;
17     ]
18 DGEYEERİSBİLGİ BİTTİ;

```

Şimdi aşağıdaki programın ve verilerin verilmiş olduğunu varsayalım ve liste üzerinde EKLE/SİL işlemlerini birlikte ele alan bir algoritmayı inceleyelim:

```

1  LİSTEİŞLE YORDAMI,
2  KOMUTLAR;
3  BOŞLİSTEHAZIRLA ÇAĞIR;
4  ÖNCE VERİBİTTİ(1) SINA DEĞİLSE YINELE;
5  [( İSKODU, ADAY ) OKU;
6      EGER İSKODU = 'EKLE'
7      İSE [ EGER DGEBUL(SERBEST,X)
8          İSE [ D(X).BİLGİ := ADAY;
9              D(X).BAĞ := AD;
10             AD := X ]

```

```

11 1. Listenin ilk DEĞİLSE [ ('ÜGE KALMADI' YAZ ;
12 2. Liste ]
13 3. EGER ISKODU = 'StL'
14 4. ise [ EGER AD = 0
15 5. ise ('LISTE BOS, İSLEM YANLIŞ' YAZ)
16 6. DEĞİLSE [ X := AD ;
17 7. AD := D(X).BAG;
18 8. ÖGEGERİVER(SERBEST,X)ÇAGIRI
19 9. ]
20 ]
21 LISTEİŞLE BİTTİ;

```

Yukarıdaki örnekte de görüldüğü gibi ekleme ve çıkarmalar bu listenin başlangıcına yapılmıştır. Ekleme ve çıkarmalar ile listenin sıralı olup olmayışına göre birkaç belirgin kullanımı ayrıca söyleyelim:

- i) Bağlaçlı Yığıt
- ii) Bağlaçlı Kuyruk
- iii) Tek Bağlaçlı Sıralı Doğrusal Liste
- iv) Tek Bağlaçlı Döngülü Liste

Örneğimize başlarken bir yüksek lisans sınıfının öğrenci listesini ele almıştık. Gerçek uygulamalar daha karmaşıktır ve birçok liste (sınıf listesi) içerebilir. İlk erişim sorunu istenen listenin 'Liste Denetim Grubu' nu bulmaktır. 'Liste göstergesi' bulunduktan sonra ögeye erişim iki biçimde olabilir:

i) Listenin ilk ögesine erişim istenir,
 ii) 0 liste öğelerinden belirli birine erişim istenebilir.
 ilk erişim hemen sağlanabilir. İkinci tür erişim ise bir DGEARA yordamının oluşturulmasını gerektirir. Bu yordamı yazmadan önce bu yordamın işletiminden sonra yapılması olası işleri düşünelim.

- i) Bulunan öğe listeden silinecektir,
- ii) Aranılan öğenin listede bulunmaması durumunda bildiri verilecektir,
- iii) Aranılan öğe listede yok ise durumuna uygun öğeler arasına eklenecektir.

DGEARA yordamının Q göstergesi ile arana öğenin yerleştiği yeri işaret edeceğini belirlesek, sonra yapılacak işlemler için bu öğeden önce gelen öğesinde bir gösterge ile işaret edilmesinin gerekeceğini anlarız. Bu göstergede P diyelim ancak bazı özel durumları da dikkate almalıyız, örneğin Q göstergesi listenin ilk ögesi ise P göstergesi bundan önce gelen öğeyi gösteremez. Dört olası durum aşağıda özetlenmiştir:

P	Q	Anlamı
0	0	Liste boş hiç arama yapılmadı
0	$\neq 0$	Aranılan öğe liste göstergesi ile gösteriliyor
$\neq 0$	$\neq 0$	Aranılan öğe listede bulundu
0	$\neq 0$	Aranılan öğe bulunamadı, P uygun ekleme yerini gösteriyor

```

1  ÜGEARA(İLK,AD,P,Q) YORDAMI;
    /* P ARANAN ÜGEYİ İŞARET EDEN ÜGEYİ, Q ARANAN ÜGEYİ */
    /* İŞARET EDER */
2  DİŞYAPILAR;
3  P, Q GÖSTERGE ;
4  AD TÜR(L(*).BİLGİ);
5  KOMUTLAR;
6  P := İLK;
7  Q := 0; /* P LİSTE GÖSTERGESİNİ ALDI */
8  EĞER P = 0
9  İSE DÖN ; /* LİSTE BOS */
10 Q := P ;
11 P := 0 ;
12 EĞER L(Q).BİLGİ = AD
13 YADA L(Q).BİLGİ > AD
14 İSE DÖN ; /* ARANAN İLK ÜGEDİR */
15 ÖNCE Q = 0 SİNA, SONRA YİNELE;
16 [ P := Q ; Q := L(Q).BAĞ ;
17 EĞER L(Q).BİLGİ := AD
18 İSE DÖN ; /* ARANAN BULUNDU */
19 EĞER L(Q).BİLGİ > AD
20 İSE [ Q := 0 ; DÖN ] /* ARANAN SİRALI LİSTEDE BULUNA-*/
    /* MADI UYGUN EKLEME YERİ BULUNDU */
21 ]
    /*SİRALI LİSTE KULLANILMAZSA İKİNCİ KOSUL YAZILMAMALIDIR*/
22 ÜGEARA BİTTİ;

```


3.1.4 TEK BAĞLAÇLI DOĞRUSAL LİSTENİN KULLANIMI

Tek bağlaçlı doğrusal liste sıralı listedeki öge sayısının 20 - 30 'u fazla aşmadığı durumlarda çok kullanışlı bir yapıdır. Aynı tür BİLGİ alanı kullanan bir çok liste tek bir SERBEST ögeler listesinden öge olarak gerçekleştirilebilir.

Bu veri yapısında bir ögenin yanına birleştirilen BAĞ alanı nedeniyle getirilen yük genellikle önemsizdir. Örneğin 20 karakter BİLGİ iki karakter BAĞ alanı olan bir yapıda getirilen ek bellek gereksinmesi % 10' dur. Bu yapı genellikle bir çok listenin birarada kullanıldığı bir yapıdır. Varsayalımki aşağıdaki kayıt yapısında 4 adet ders için yer ayrılmış olsun.

Su anda Ü.P.K. dersi 6 öğrenci ile açık olsun ve bu dersten K. HAMAT' ın silme ve H. BARAÇLI' yı bu derse ekleme işlemlerini aşağıdaki bellek gösterimi üzerinde ayrı ayrı inceleyelim:

İSLEM 1:

K. HAMAT' ın silinmesi

Uyarılacak Yordamlar

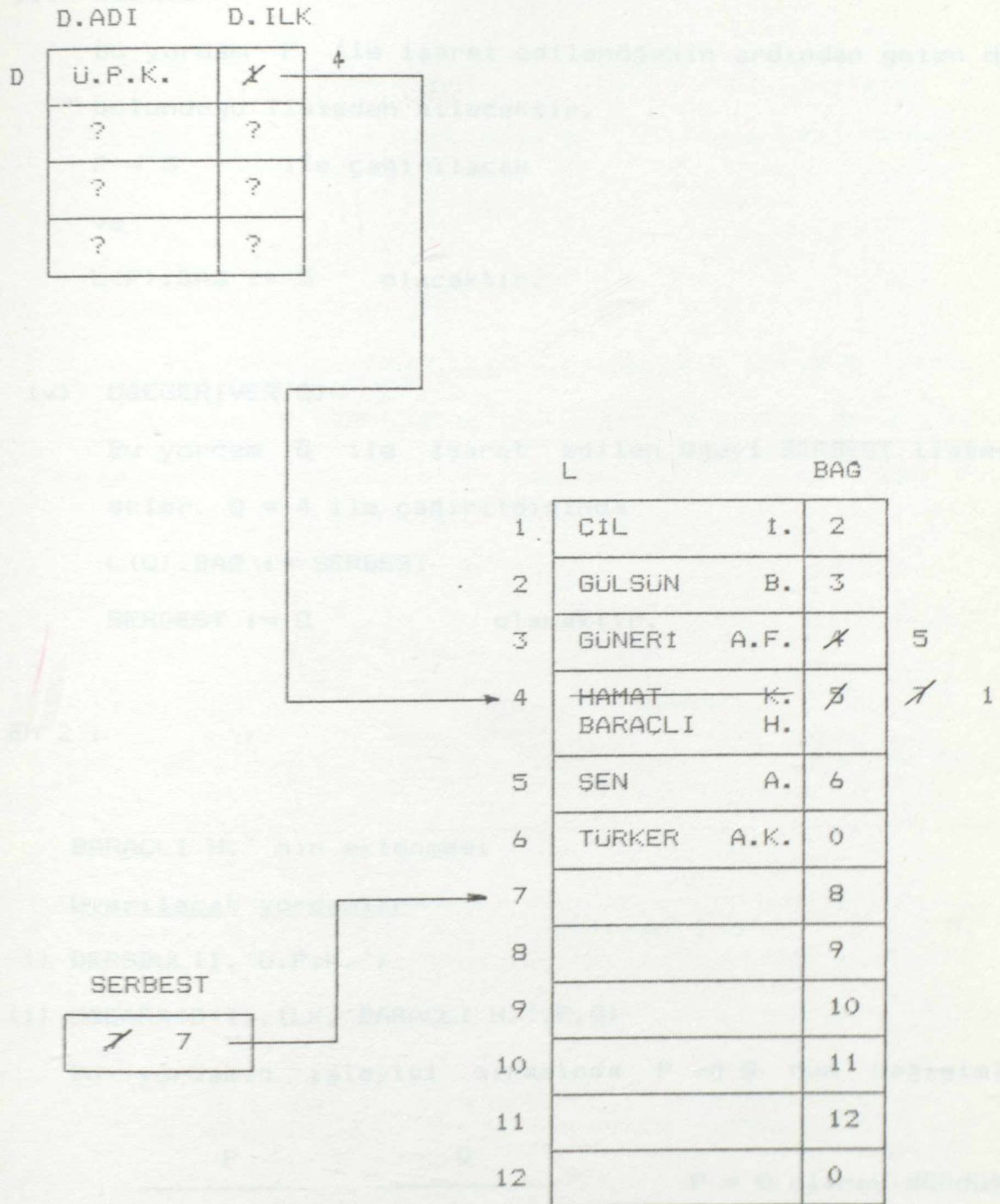
i) DERSEBUL(I, 'Ü.P.K.')

ii) DGEARA(D(I).İLK, 'K. HAMAT', P, Q)

Bu yordamın işleyişi sırasında P ve Q nun değişimleri aşağıdaki gibi olacaktır.

P	Q
1	0
0	1
1	2
2	3
3	4

Bularak döndü Q = 4



Sekil 25 Birden Fazla Tek Bağlı Doğrusal Liste Örneği

iii) ÖGESİL(P)

Bu yordam P ile işaret edilen ögenin ardından gelen ögeyi bulunduđu listeden silecektir.

P = 3 ile çağırılacak

ve

L(P).BAG := 5 olacaktır.

iv) ÖGEGERİVER(Q)

Bu yordam Q ile işaret edilen ögeyi SERBEST listesine ekler. Q = 4 ile çağırıldığında

L(Q).BAG := SERBEST

SERBEST := Q olacaktır.

İŞLEM 2 :

BARAÇLI H.'nin eklenmesi

Uyarılacak yordamlar

i) DERSEBUL(I, 'Ü.P.K.')

ii) ÖGEARA(D(I).İLK, 'BARAÇLI H.', P, Q)

Bu yordamın işleyişi sırasında P ve Q nun değışimleri

P	Q
1	0
0	1

P = 0 olarak döndüğüne göre başlangıca ekleme yapılacaktır.

iii) ÖGEAL(T)

4 nolu yer T göstergesi ile kullanıma verilecek ve ilk SERBEST öge bunun bağlacından 7 olarak belirlenecektir.

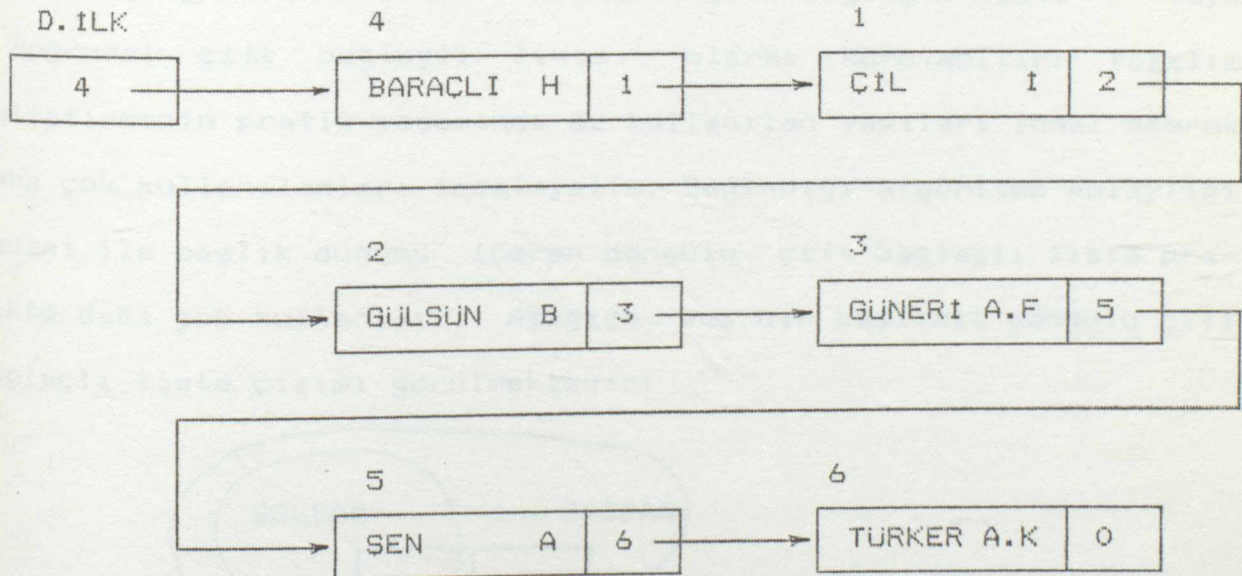
iv) ÖGEKLE(D(I).İLK, P, T)

Bu yordam $P = 0$ ise liste göstergesi önemli olmayacak T göstergesi P ile gösterilen ögenin arkasına eklenecektir. Eger ekleme listenin başlangıcına yapılacak ise $P = Q$ olmalıdır.

$D(I).İLK := 4$ $I = 1$

ve içeriği 'BARAÇLI H.' oluşu bu aşamadır.

Neticede oluşacak liste yapısı aşağıdaki gibi olacaktır.



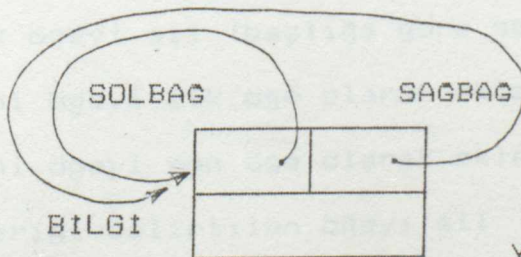
Sekil 26 Tek Bağlılı Doğrusal Liste Uygulaması Sonuç Çizimi

3.2 ÇİFT YÖNLÜ BAĞLAÇLI DOĞRUSAL VE DÖNGÜLÜ LİSTELER

Bu aşamaya değin kullandığımız bağlaçlı listeler her düğümde bir bağ alanı gerektiriyor ve yalnız bir yöne doğru listenin öğeleri üzerinde kolay hareket sağlanıyordu. Bazı uygulamalarda üzerinde bulunulan öğeye göre iki yönde de kolay hareket edilebilmesi istenebilir. Bu tür uygulamalardan birkaçı şunlardır:

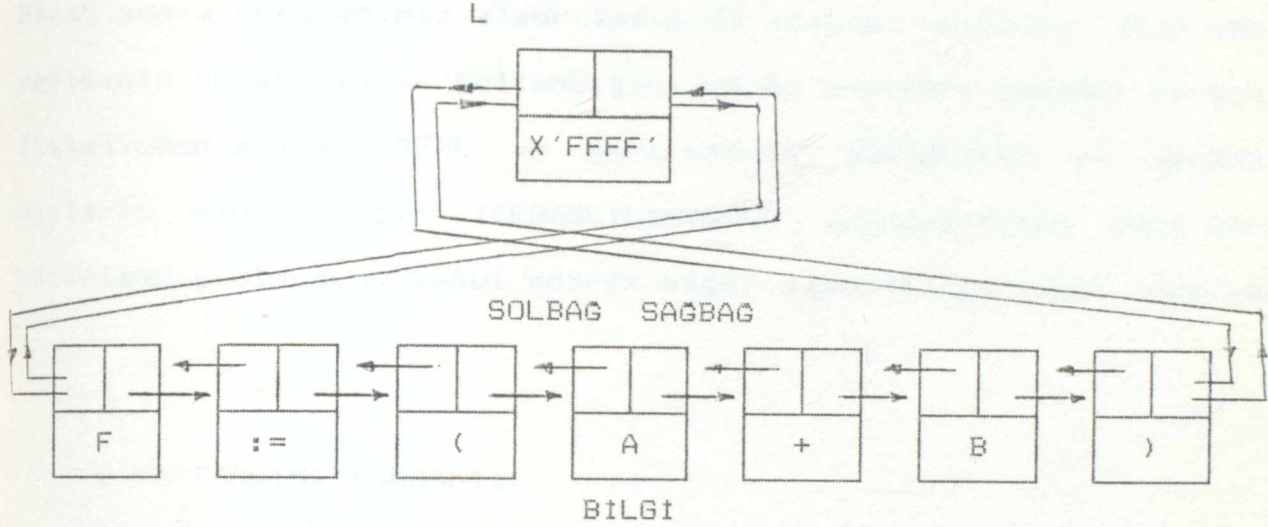
- i) Hem azalan sırada, hem aratan sırada listelenmesi anlamlı olabilecek bir BİLGİ alanı listenin düzenini belirliyorsa (örneğin sınıftaki genel sınav notları kistesinin düzenini belirleyen bilgi alanı ise)
- ii) Bilgisayar ana belleğinin değişken boyutlu parçalar ile birçok programın kullanımına dinamik olarak açabilmek gerekli yada tasarlanmış ise

Çift bağlaçlı bir liste 'döngülü çift bağlaçlı liste' veya 'doğrusal çift bağlaçlı liste' olarak kurulabilir. Yazılım geliştirmenin pratik yaşamında az kullanılan yapıları ihmal ederek daha çok kullanılanları inceleyelim. Sağladığı algoritma kolaylığı nedeni ile başlık düğümü içeren döngülü çift bağlaçlı liste pratikte daha çok kullanılır. Aşağıda boş bir başlıklı döngülü çift bağlaçlı liste çizimi görülmektedir:



Sekil 27 Boş Bir Çift
Yönlü Bağlaçlı Döngülü Liste

şimdi de elemanları dolu olana bir örnek verelim. Aşağıda 7 ögeli ve başlıklı döngülü çift bağlaçlı liste görülmektedir:



Şekil 28 Çift Yönlü Bağlaçlı Döngülü Liste Örneği

Bir çift bağlaçlı döngülü listede her düğüm için aşağıdaki işlem çiftinin sağlanması gereklidir:

$X := \text{SAGBAG} (\text{SOLBAG} (X))$

$X := \text{SOLBAG} (\text{SAGBAG} (X))$

Bu koşulun başlık düğümüne göre birinci ve sonuncu düğümler, hatta boş liste için de sağlandığını görmekteyiz. Bu yeni yapı içinde listeye (çift bağlaçlı) ekleme ve çıkarma yapmak daha fazla dikkat gerektirir.

- i) İlk ögeyi sil (başlığa göre soldaki ögeyi sil)
- ii) Yeni ögeyi ilk öge olarak ekle (başlığa göre sola ekle)
- iii) Yeni ögeyi son öge olarak ekle (başlığa göre sağa ekle)
- iv) İçerigi belirtilen ögeyi sil

- v) Belirtilen X ögesini belirtilen P ögesinin soluna ekle
- vi) Belirtilen X ögesini belirtilen P ögesinin sağına ekle

Biraz sonra vereceğimiz algoritmalarda aşağıda verilmiş olan veri yapısının ortak olarak kullandığını ve bu yapıdaki ögeleri serbest listesinden alacak ÖGEAL ve geri verecek ÖGEGERİVER ve serbest ögelerin oluşumu için TEKBAGLILISTEKUR yordamlarının daha önce hazırlanmış olduğunu kabul ederek diğer algoritmalarımızı yazalım.

Yapılandırma ve BİTİT:

X := D(L).SOLBAG

- 1 YAPITANIMI YORDAMI;
- 2 İÇYAPILAR;
- 3 SERBEST GÖSTERGE, EVRENSEL;
- 4 D(200) (SOLBAG, SAGBAG GÖSTERGE,
- 5 BİLGİ KARAKTER (4) YAPI) EVRENSEL;
- 6 YAPITANIMI BİTTİ;

Şimdi sırasıyla diğer algoritmalarımızı yazalım:

Yapılandırma ve BİTİT:

- 1 ÇİFTBİTİL(L) MANTIKSAL YORDAMI;
- 2 DİSYAPILAR,
- 3 D(*) , SERBEST ;
- 4 KOMUTLAR;
- 5 X := D(L).SAGBAG; /* başlık yapısına dikkat */
- 6 EĞER X = L
- 7 İSE [/* başlıktan başka öge yok silinmez */

```

8      ÇİFTBSOLSİL := YANLIŞ ]
9      DEĞİLSE [ D(D(X).SOLBAG).SAGBAG := D(X).SAGBAG;
10         D(D(X).SAGBAG).SOLBAG := D(X).SOLBAG;
11      ÇİFTBSOLSİL DOĞERİVER(X) ÇAGIR;
12      ÇİFTBSOLSİL := DOĞRU ]
13 ÇİFTBSOLSİL BİTTİ;

```

ÇİFTBSAGSİL yordamı yukarıdaki yordama işe başlayışında farklılık oluşturacak ve 5. komutta

```
X := D(L).SOLBAG
```

işlemini yaparak liste başlığına göre en soldaki ögeyi silmeye yönelecektir. Diğer komutlar aynen geçerlidir. Şimdi bu iki örnekten sonra çift bağlı döngülü listeden bağlaçlarla dolaşım yönünden başlık ögesinin hem listenin solunda hem de sağında varmış gibi etki oluşturduğu açıkça görülecektir. Ancak bu durum bağlaçlar üzerinden ilerleyiş için söz konusudur. Eğer başlık ögesinin aramalar sırasında bir emniyet tamponu gibi görev yapmasını ister ve başlık ögesinin BİLGİ alanına bunu sağlayacak bir bilgi yerleştirmek istersek bir tercih yapmamız gereklidir. Olası seçimlerimiz aşağıdaki tercihlerden biri olabilir:

- i) Liste içeriği alfabetik sıralı iken başlığın bilgi alanına ' boşluk ' karakterleri yerleştirilir.
- ii) Liste içeriği alfabetik sıralı iken Başlığın bilgi alanına ' ZZZZ ' karakterleri yerleştirilir.
- iii) Başlığın bilgi alanına liste ögelerinin içeriği olamayacak özel bir bilgi yerleştirilir ve bunu arama arama algorit-

ması içinde kontrol ettirilir.

Simdi yeni algoritmamız CİFTBSOLEKLE algoritmasını yazalım:

```

1 CİFTBSOLEKLE(X,L) YORDAMI;
   /* L listesinin bu yordam uyarıldığı zaman en azından
   /* bir başlık ögesi ile var olduğu kabul edilmiştir
2 KOMUTLAR;
3   D(X).SAGBAG := D(L).SAGBAG;
4   D(X).SOLBAG := L;
5   D(D(L).SAGBAG).SOLBAG := X
6   D(L).SAGBAG := X
7 CİFTBSOLEKLE BİTTİ;

```

Bu algoritmada dikkat edilmesi gereken komutlar özellikle 5. ve 6. komutlardır. Beşinci komutta önce içerdeki parantzin yorumlandığına ve 5. ve 6. komutların yerlerinin değişmemesi gerektiğine dikkat etmek gerekir. Simdi bilgi içeriği verilen bir ögeyi çift bağlı döngülü listede bulup silecek bir algoritmayı inceleyelim.

Varsayalım ki listemiz sağ tarafı tamponlayan bir başlık içerigine sahip olsun.

```

1 CİFTBARAVESİL(L,KİMİSİL) MANTIKSAL YORDAMI;
2 DISYAPILAR;
3   SERBEST, D(*); /* daha önce tanımlandıkları gibi */
4   KİMİSİL KARAKTER(4);
5 KOMUTLAR;

```

```

6  X := D(L).SAGBAG; /* listenin sol tarafından girildi */
7  ÖNCE X ≠ L SİNA SONRA YİNELE;
8  [ EĞER D(X).BİLGİ = KİMİSİL
9      İSE [ D(D(X).SOLBAG).SAGBAG := D(X).SAGBAG;
10         D(D(X).SAGBAG).SOLBAG := D(X).SOLBAG;
11         DEĞERİVER(X) ÇAĞIR;
12         ÇİFTBARAVESİL := DOĞRU;
13     DÖN ]
14     DEĞİLSE [ X := D(X).SAGBAG ]
15 ] /* Arama döngüsü sonu */
16 ÇİFTBARAVESİL := YANLIŞ;
17 ( ' SİLİNECEK ÖĞE BULUNAMADI ' ; KİMİSİL ) YAZ;
18 ÇİFTBARAVESİL BİTTİ;

```

3.3 ALT LİSTE PAYLAŞIMSIZ LİSTE YAPILARI

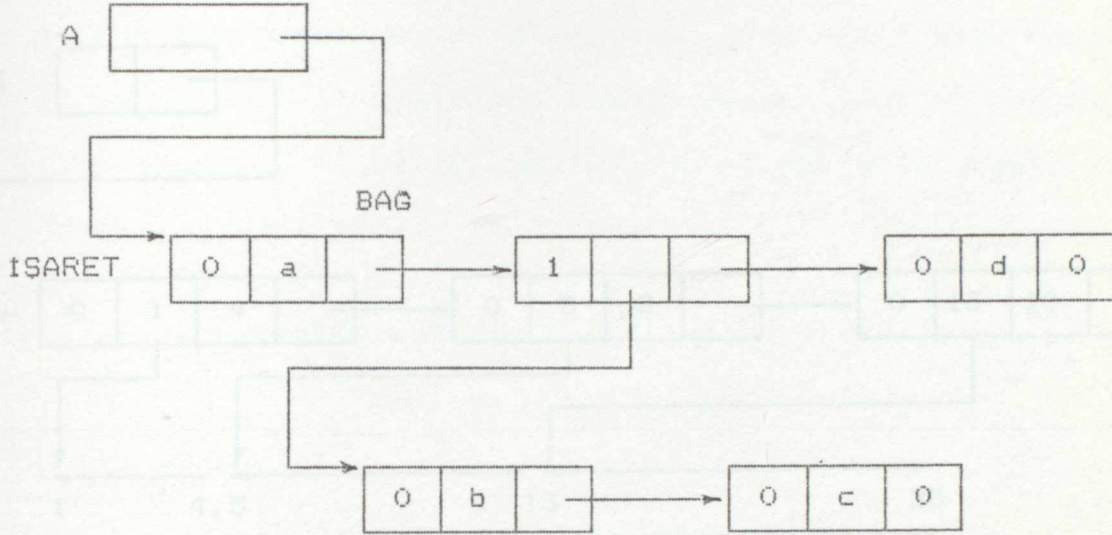
Tek bağlaçlı doğrusal liste basit bir yapıdır. Bu yapıda her bir öge basit bir atomdur. Atom, doğrudan veri içeren en alt düzeydeki liste ögesine verilen addır. Eğer listenin öğelerinin bir atom veya birbirinden ayrık (paylaşımsız) birer liste olarak alırsak oluşan yapı altliste paylaşımsız bir liste yapısıdır.

Tanımı : $A = (a_1, a_2, \dots, a_n)$ $a_i = a_j$ eğer $i = j$

a_1 : liste başı (head of list)

$a_2 \dots a_n$: liste kuyruğu (tail of a list) olarak anılır.

Örnek : $A = (a, (b, c), d)$ gösterimi ;



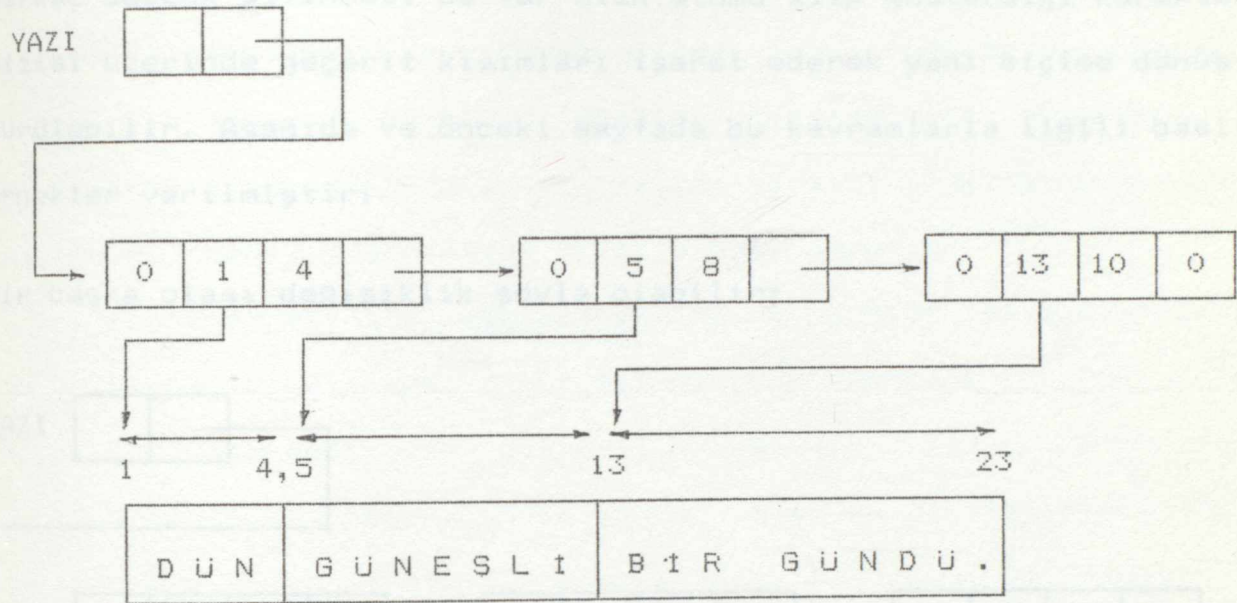
Sekil 29 Alt Liste Paylaşımsız Liste Yapısına Örnek

Veri yapısına erişim : Liste göstergesi ile yapılır.

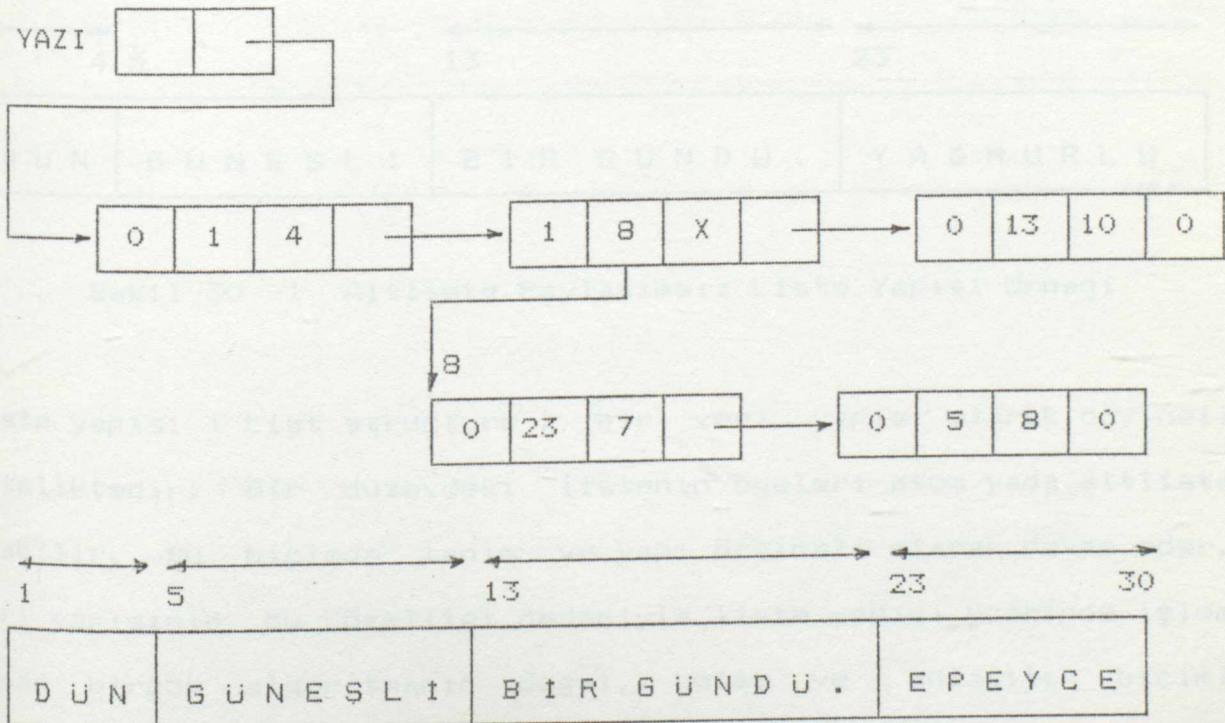
Veri ögesine erişim : Algoritmanın gereğine göre bağlaçlar aracılığı ile yapılır.

Alt liste paylaşımsız liste yapıları bilgisayar alanında yazılım-
 evi niteliğinde çalışan kuruluşlarda sıkça karşılaşılan bir veri
 yapısıdır. Özellikle yazı düzenleyiciler (text editors) ve bilgi
 belge erişim sistemleri (information retrieval systems) gibi
 destek yada hizmet yazılımları ile derleyiciler gibi sistem yazı-
 lımları (system software) bu veri yapısının kullanımı ile
 karşılaşılabilecek tipik alanlardır. Örneğin bir yazı düzenleyici-
 de liste yapısının en üst düzeyi tüm yazıyı, birinci düzeydeki
 liste ögeleri ana bölüm başlıklarını, bir sonraki düzeyi çeşitli

karakter dizilerini işaret eden ögeleri içerebilir.



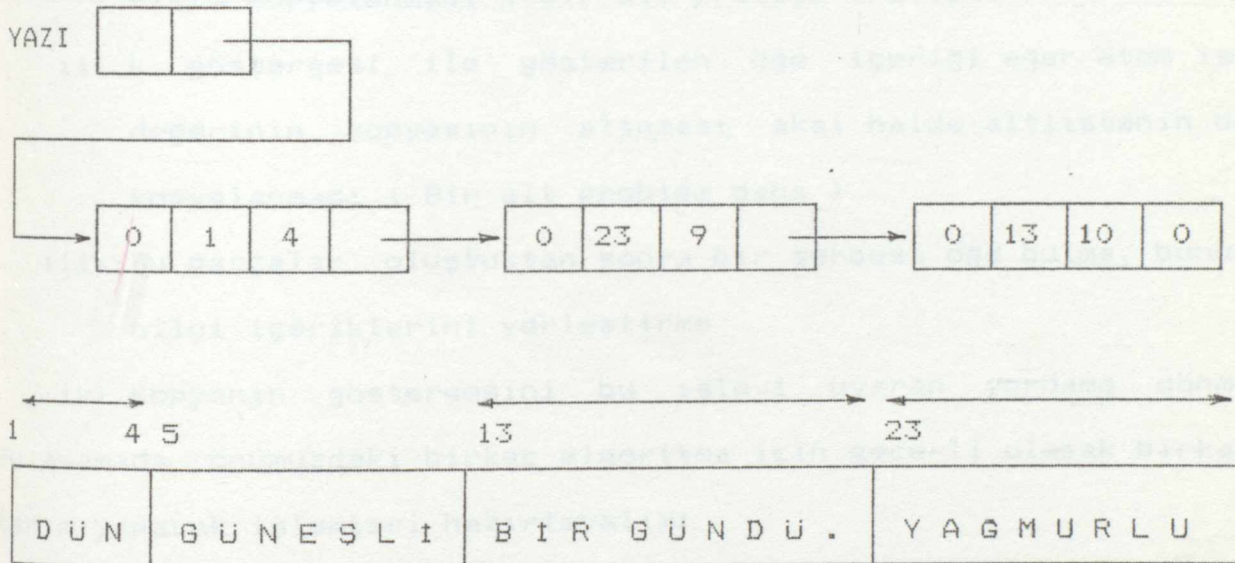
Şimdi bu yapıyı kısmen değiştirelim.



Şekil 3 Altliste Paylaşımsız Liste Yapısı Örneği

Daha önce var olan ve bir paragrafı işaret eden atom kolayca bir alt listeye dönüştürülebilir. Öte yandan bir paragraf içinden birkaç sözcük silinmesi de var olan atomu alıp gösterdiği karakter dizisi üzerinde geçerli kısımları işaret ederek yeni biçime dönüştürülebilir. Aşağıda ve önceki sayfada bu kavramlarla ilgili basit örnekler verilmiştir:

Bir başka olası değişiklik şöyle olabilir;



Sekil 30 -1 Altliste Paylaşımsız Liste Yapısı Örneği

Liste yapısı (List structure) bir veri yapısı olarak özyineli niteliktedir. Bir düzeydeki listenin öğeleri atom yada altliste olabilir. Bu biçimde tanım ve yapı özyineli olarak devam eder. Veri yapısının bu özelliği nedeniyle liste yapısı üzerinde işlem yapan birçok algoritmanın doğal, kolay ve anlaşılır biçimi özyineli algoritmalarlardır. Daha öncede işlendiği gibi özyineli

yordam sorunu alt parçalara bölerek çözüm getirir. Eger sorun daha alt parçalara bölünmeyecek kadar küçülüp ' o sorunun en basit yapısı ' na indirgenmiş ise basit bir işlem yapılır ve bu kez parça parça çözümlerin geri dönüş aşamasında birleşerek tüm çözümü oluşturması aşaması başlar. Aşağıda verilen ilk algoritmada paylaşımsız listenin kopyalanması işlemi vardır. Şimdi bu problemi başarılması gereken dört adım işlem olarak görelim.

- i) L göstergesi ile gösterilen ögenin sagbağ ile gösterilen kısım kopyalanması (Bir alt problem üretildi)
- ii) L göstergesi ile gösterilen öge içeriği eğer atom ise değerinin kopyasının alınması aksi halde altlistenin de kopyalanması (Bir alt problem daha)
- iii) Bu parçalar oluştuktan sonra bir serbest öge bulma, bunun bilgi içeriklerini yerleştirme
- iv) Kopyanın göstergesini bu işlevi uyaran yordama dönme

Bu aşamada önümüzdeki birkaç algoritma için geçerli olacak birkaç tanım yaparak işlemleri hazırlayalım:

- a) Paylaşımsız listeyi işleyen yordam adlarını PZLİS, Altlist-
te 'yide PALİS olarak kısaltalım.
- b) Paylaşımsız liste için eleman alacağımız serbest listesi-
nin ögeyapısı aşağıdaki gibi olsun.
- c) Y yapısının tek bağlı bir serbest ögeler listesi olarak kurulması, ÖGEBUL ve ÖGEVER yordamları hazır olsun.


```

1 ORTAKVERİYAPISI YORDAMI;
2 İÇYAPILAR;
3   Y(400) (İSARET TAMSAYI,
4           BİLGİ TÜR (TAMSAYI GÖSTERGE),
5           BAĞ GÖSTERGE ) YAPI, EVRENSEL;
6   SERBEST GÖSTERGE , EVRENSEL;
7 ORTAKVERİYAPISI BİTTİ;

```

```

1 PZLİSKOPYA(L) YORDAMI;
2 DISYAPILAR;
3   Y(*)/* Bu veri yapısı daha önce tanımlanmış yapısında */
4   SERBEST GÖSTERGE;
5 İÇYAPILAR;
6 KOMUTLAR;
7   P := 0
8   EĞER L ≠ 0
9     İSE [ EĞER Y(L).İSARET = 0
10          İSE [ Q := Y(L).BİLGİ ]
11          DEĞİLSE [ Q := PZLİSKOPYA(Y(L).BİLGİ)];
12          /* L ile gösterilen ögenin alt listesinin kop_*/
13          /* yasıyada bilgi içeriği Q içinde */
14          R := PZLİSKOPYA (Y(L).BAĞ );
15          /* L ile gösterilen listenin kuyruk kısmının */
16          /* kopyası R içinde */
17   EĞER DĞEBUL(SERBEST,P)
18     İSE [ Y(P).BİLGİ := Q;

```

```

15          Y(P).BAĞ := R;
16          Y(P).İŞARET := Y(L).İŞARET ]
17 DEĞİLSE [ ' ÖGE BİTTİ ' YAZ; DÖN ]
18 ]
19 PZLİSKOPYA := P
20 PZLİSKOPYA BİTTİ;

```

Bu algoritmamızda da S ve T göstergeleri ile gösterilen iki paylaşımsız liste yapısının eşitliği sınanacaktır. Bu özyineli algoritmamızda şu adımların sağlanması ile kurulabilir:

- i) T ve S her ikiside boş gösterge ise eşittir.
- ii) T ve S 'den biri boş gösterge öteki değil ise eşit değildir.
- iii) T ve S 'den her ikiside boş gösterge değil ise
 - a) Ögelerin işaretleri aynı ve sıfır durumunda, atomların eşit olmaması durumunda eşit değildir.
 - b) Ögelerin işaretleri aynı ve sıfır değilse altlistelerin eşit olmaması durumunda eşit değildir.
- iv) Altliste eşitliği sınanmasından başarılı olarak geçen algoritma T ve S ile gösterilen ögelerin eşitlik sınaması ile devam eder.

```

1 PZLİŞEŞİTMİ(S,T) MANTIKSAL YORDAMI;
2 DIŞYAPILAR;
3   Y(*); /* Y yapısı önceki tanım gibidir */
4 İÇYAPILAR;

```



```

5  TEST MANTIKSAL;
6  KOMUTLAR;
7  TEST := YANLIS;
8  SECENEK;
9  [ S = 0 VE T = 0 : [ TEST := DOGRU ];
10 S ≠ 0 VE T ≠ 0 :
11   [ EGER Y(S).ISARET = Y(T).ISARET
12     ISE [ EGER Y(S).ISARET = 0
13         ISE [ EGER Y(S).BILGI = Y(T).BILGI
14             ISE TEST := DOGRU;
15             DEGI LSE TEST := YANLIS ]
16         DEGI LSE [ EGER TEST := DOGRU
17             ISE [ TEST := PZLISESITMI(Y(S).BILGI, Y(T).BILGI) ]
18             ] /* Y(.) .ISARET eřitligi incelendi*/
19     ]
20   DEGI LSE TEST := YANLIS ;
21 ] /* Y(.) liste bařı S yada T sıfır deęilken incelendi*/
22 ] /* liste bařı incelendi */
23 EGER TEST = DOGRU
24   ISE [ TEST := PZLISESITMI (Y(S).BAG, Y(T).BAG) ]
25 PZLISESITMI := TEST ;
26 PZLISESITMI BITTI;

```

3.4 CİFT BAĞLI DÖNGÜLÜ LİSTENİN BİLGİSAYAR SİSTEM

YAZILIMINDA KULLANIMI

Cift bagli döngülü liste bilgisayar yazılımında çok sık kullanılan bir yapı değildir. Çünkü çift yönlü bagli listenin bir gerek ortaya çıkması için tek yönlü baglaçlı listenin yeteneklerinin aşılması, bu yapı ile sağlanamayan bazı özelliklerin bulunması lazımdır. Daha önce tek yönlü listenin bir anahtara göre sıralandığını ve bu şekilde sıralı listenin ters yönde sıralı bilgiyi elde etmek için üç biçimde kullanılabileceğini görmüştük.

Hatırlayacağımız gibi bu yöntemler:

- i) Tek bagli listeyi özyineli algoritma kullanarak ters yöne çevirme
- ii) Tek bagli listeyi üç gösterge kullanarak ters yöne çevirme
- iii) Tek bagli listeyi baştan sona geçerken öğelerin indislerini bir yığita doldurma ve yığıtı boşaltırken öğe içeriklerini yazdırma.

olarak özetlenebilir. Bu yöntemlerden asıl ikisi asıl listenin yön ilişkisini bozmakta üçüncüsü ise tüm liste üzerinde işlem yapmakta algoritma karmaşıklığı olarak $2.O(n)$ yapısını göstermektedir. Cift bagli döngülü listenin kullanımı, öğenin önceki ve sonraki komşusunun bulunması gereği çok sık ortaya çıkıyor ise sözkonusudur. Bu tür kullanım örneği en tipik biçimiyle bilgisayar sistem yazılımında vardır. Kullanıcının bilgisayarı kendi programlarının işletimi için kolay ve güvenilir biçimde kullanmasını sağlayan yazılım ' sistem yazılımı (system software)' olarak anılır. Sistem yazılımı kendi içinde donanıma en yakın düzeyde işletim sistemi,

bir üst düzeyde derleyiciler ve hizmet yordamları biçiminde gruplanabilir. Eger bilgisayar tek kullanıcılı bir mikro bilgisayar ise o anda var olan tüm ana bellek tek kullanıcıya aittir ve kullanıcılar arasında paylaşılamaz. Ancak bilgisayar çok kullanıcı-
lı yapıda kullanılıyor ise bilgisayarın ana belleğinin kullanıcı-
lar arasında paylaşıyor olması gerekir. Bu paylaşımı etkin ve güvenilir bir biçimde sağlama sorumluluğu işletim sisteminin bir alt kesimi olan bellek yönetimi yazılımındadır. İşletim sistemleri konusunda ayrıntılı biçimde incelenen bellek yönetimi felsefeleri şöyle özetlenebilir.

- i) Sabit boylu bellek sayfaları ile bellek yönetimi
- ii) Değişken boylu mantıksal bellek bölümleri ile bellek yönetimi
- iii) Bellek sayfalama ile desteklenmiş bellek bölümlere yöntemi

Aşağıda çift bağlı döngülü listenin kullanım örneği olarak vereceğimiz yöntem (ii) ile işaretlenmiş değişken boylu mantıksal bellek bölümleri ile bellek yönetimi örneğidir. Bu yöntem başka işletim sistemleri yanında son yıllarda kullanımı artan başarılı sistemlerden UNIX' in de kullandığı bellek yönetimi felsefesidir. Bu yöntem içinde sistem o gün ilk açıldığında tüm bellek yalnızca iki bölgeden oluşacak biçimde başlangıç durumuna getirilir. Bu işlem İLKOLUSTUR adlı kısa bir programla sağlanmalıdır. Birinci bölgede sadece göstergelerin bulunduğu özel bir başlık, ikinci bölgede ise tüm belleği boş bir alan olarak kabul eden kesimdir. Aktif hale gelen bir kullanıcı programı ne kadar

bellek istediğini belirterek denetimi işletim sistemine bırakır, işletim sistemi de alt yordam olarak kullanmak üzere şimdi incelediğimiz ' bellek yönetimi ' örnek yordamını uyarır. Etkin hale gelen BELLEKATA yordamı eğer uygun bellek alanı bulabilir ise bulduğu alanı gösteren P göstergesini işletim sistemine verir ve o program için atadığı serbest bellek alanını serbest listesinden algoritmada görüldüğü gibi çıkarır. Öte yandan eğer kullanıcı programının işletimi sona ererse bu bilgi önce işletim sistemi tarafından ele alınır. İşletim sistemi P göstergesinide vererek BELLEKBIRAK yordamını uyarır. Bu yordamın işlevlerinden biri de serbest bırakılan bellek bölgesinin bitişik altındaki yada bitişik üstündeki bellek bölgelerinin serbest olup olmadığının denetimidir. Eğer komşu bölgelerden serbest olanlar varsa yeni serbest bölge bunlarla birleştirilerek daha büyük serbest bölgeler elde edilir. Aşağıda bu algoritmalar verilmiştir.

- 1 İLKOLUŞTUR (BAŞLA, BOYUT) YORDAMI;
- 2 İÇYAPILAR; GÖSTERGEBOYU = 4;
- 3 P, SERBEST GÖSTERGE, EVRENSEL;
- 4 GÖSTERGEBOYU TAMSAYI, DEĞER = 4, EVRENSEL;
 - /* bellek boyu 64 Kbyte'dan büyük olduğunda */
 - /* gösterge boyu 4 byte (32 ikil) alınır */
- 5 BLOKBOYU TAMSAYI, DEĞER = 256, EVRENSEL;
 - /* Bu algoritma 256 byte 'dan küçük kırpıntı bellek */
 - /* alanlarını ihmal eder */
- 6 İSARET, BOY, SOLBAĞ, SAĞBAĞ GÖSTERGE, EVRENSEL;


```

/* Bu algoritma büyük olasılıkla alt düzey dil */
/* (Assembler, C gibi) ile yazılacağından PASCAL */
/* PL/1, ANLAT 'da bulunan yapılar kullanılmamıştır */
8 KOMUTLAR;

9  P := BASLA /* işletim sistemi kendi yerleştiği yerden */
/* sonraki bellek adresini BASLA içinde */
/* bellek yönetimi programına verir */

10  ISARET(P) := 1; /* ilk blok kullanımında, boyu 0 */
11  BOY(P) := 0;
12  SOLBAG(P) := P + 6 * GÖSTERGE BOYU; /* her ikisi de aynı*/
13  SAGBAG(P) := P + 6 * GÖSTERGE BOYU; /* bloğu gösteriyor */
14  ISARET(P + 4 * GÖSTERGEBOYU) := 1;
15  USTBAG(P) := P;

/* Başlık (header) bloğu düzenlendi. Şimdi tüm geri */
/* kalan belleği bir boş kesim (segment) gibi görecektir */
/* kesim yazılıyor */

16  SERBSET := P;
17  P := P + 6 * GÖSTERGEBOYU ;
18  ISARET(P) := 0;
19  BOY(P) := BOYUT - 12 * GÖSTERGEBOYU,

/* bu yapıda şimdilik 12 gösterge alanı haricinde*/
/* her yer kullanılabilir durumda, BOYUT bilgisi */
/* işletim sistemince sağlanıyor */

20  SOLBAG(P) := SERBSET; /* her ikiside başlığı gösteriyor */
21  SAGBAG(P) := SERBEST;

/* şimdi bölgenin en alt kesimi düzenleniyor */

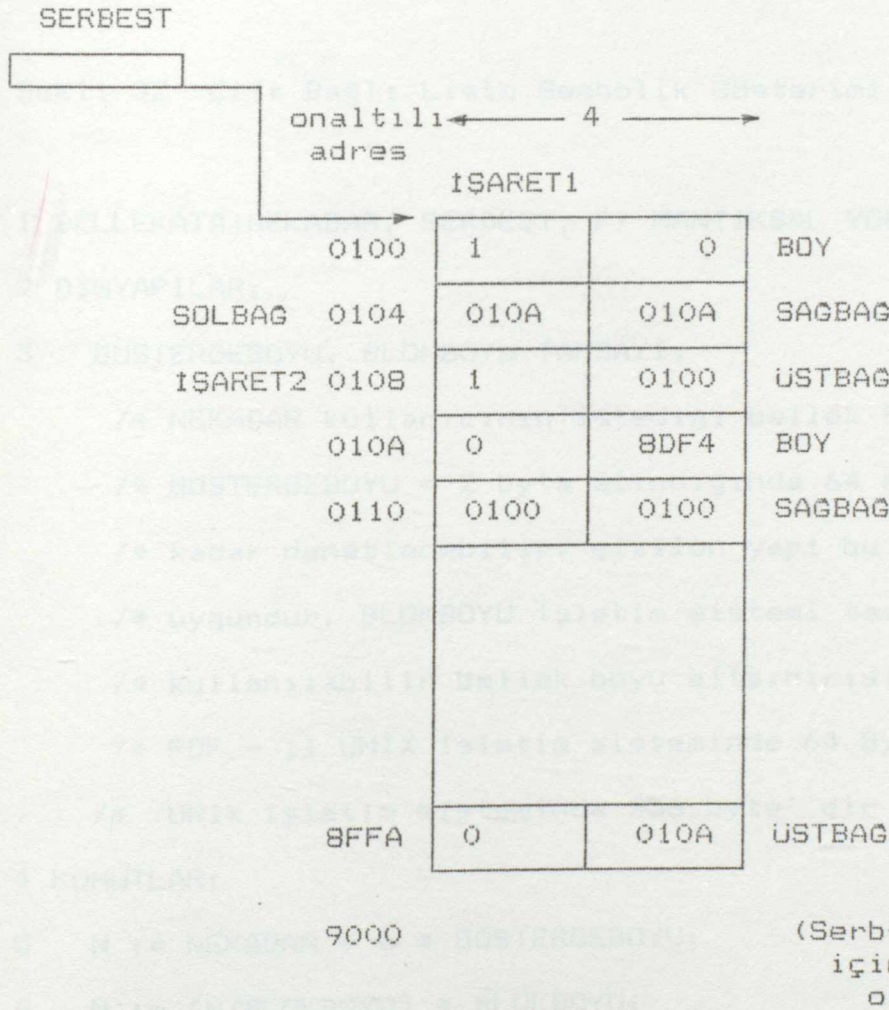
```

```

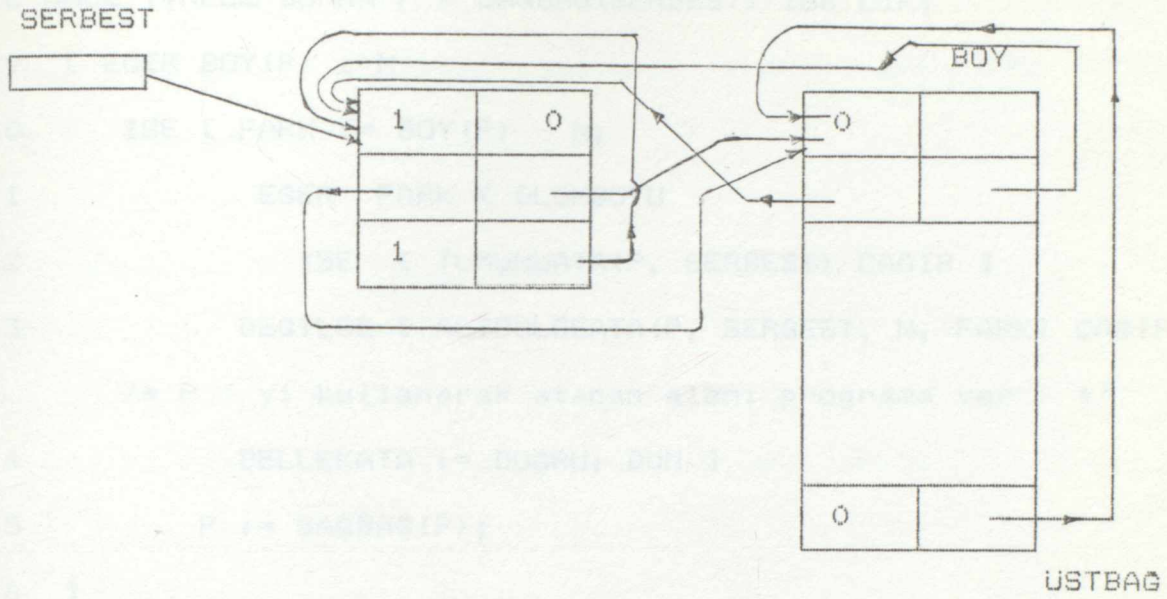
22 İSARET(P + 4 * GÖSTERGEBOYU + BOY(P)) := 0 ;
23 ÜSTBAĞ(P + 4 * GÖSTERGEBOYU + BOY(P)) := P;
24 İLKOLUŞTUR BİTTİ;

```

Bu algoritmada SOLBAĞ, SAĞBAĞ gibi alanlar yapı haline getirilmeden kullanılmıştır. İlk oluştur yordamını yukarıda gördükten sonra şimdi bu yordamın oluşturduğu bellek görünümünü inceleyelim ve daha sonrada BELLEKATA ve BELLEKBIRAK yordamlarını ele alalım.



Sekil 31 Çift Bağlı Liste Yapısının Fiziksel Bellek Gösterimi



Sekil 32 Çift Bağlı Liste Sembolik Gösterimi

1 BELLEKATA(NEKADAR, SERBEST, P) MANTIKSAL YORDAMI;

2 DISYAPILAR;

3 GÖSTERGEBOYU, BLOKBOYU TAMSAYI;

/* NEKADAR kullanıcının istediği bellek boyudur. */

/* GÖSTERGEBOYU = 2 byte alındığında 64 Kbyte belleğe */

/* kadar denetlenebilir. çizilen yapı bu duruma */

/* uygundur. BLOKBOYU işletim sistemi tasarımcısının */

/* kullanılabilir bellek boyu altsınırıdır. Bu sayı */

/* PDP - 11 UNIX işletim sisteminde 64 Byte VAX - 780 */

/* UNIX işletim sisteminde 256 byte' dir */

4 KOMUTLAR;

5 N := NEKADAR + 6 * GÖSTERGEBOYU;

6 N := [N/BLOKBOYU] * BLOKBOYU;

7 P := SAGBAG(SERBEST);

```

8 ÖNCE YİNELE SONRA P = SAGBAG(SERBEST) İSE ÇIK;
9 [ EGER BOY(P) ≥ N
10     İSE [ FARK := BOY(P) - N;
11         EGER FARK < BLOKBOYU
12             İSE [ TUMUNUATA(P, SERBEST) ÇAĞIR ]
13             DEĞİLSE [ ALTBÖLGEATA(P, SERBEST, N, FARK) ÇAĞIR ]
14                 /* P ' yi kullanarak atanan alanı programa ver */
15                 BELLEKATA := DOĞRU; DÖN ]
16 ]
17 ( 'HİCBİR BLOK YETERİNCE BÜYÜK DEĞİL' YAZ;
18 BELLEKATA := YANLIS;
19 BELLEKATA BİTTİ;

```

```

1 TUMUNUATA(P, SERBEST) YORDAMI;
2 KOMUTLAR;
3     SOLBAG(SAGBAG(P)) := SOLBAG(P);
4     SAGBAP(SOLBAG(P)) := SAGBAG(P);
5     İŞARET(P) := 1; /* atanmış olan */
6     İŞARET(P + BOY(P) - 2 * GÖSTERGEBOYU) := 1;
7     SERBEST := SOLBAG(P);
8 TUMUNUATA BİTTİ;

```

Yukarıda seçilmiş olan bellek bölgesinin gereksinmeye ancak yettiği için tümünün atandığı bir algoritmayı inceledik. Aşağıda seçilmiş bellek bölgesinin gereksinimden büyük olması durumunda

bulunan serbest alanın altbölgesini atamayı inceleyeceğiz. Altbölgeyi atamanın avantajı serbest listesinden eleman koparmadan işlem yapmanın basitliğidir.

```

1  ALTBÖLGEATA(P, N, FARK, SERBEST) YORDAMI;
2  KOMUTLAR;
3      ÜSTBAG(P + BOY(P) - 2 * GÖSTERGEBOYU) := P + FARK;
      /* Atanan alt kesim için üstbag belirlendi */
4      İŞARET(P - BOY(P) - 2 * GÖSTERGEBOYU) := 1;
5      İŞARET(P + FARK) := 1;
      /* Atanan alt kesimlerin işareti kondu */
6      ÜSTBAG(P + FARK - GÖSTERGEBOYU) := P;
7      İŞARET(P + FARK - 2 * GÖSTERGEBOYU) := 0
      /* Alt kesime bitişik iki gösterge alanı belirlendi */
8      BOY(P) := FARK ; /* geri kalan kesimin boy bilgisi */
9      SERBEST := P; /* belirlendi. Bir sonraki arama için*/
      /* serbest göstergesi belirlendi ve */
      /* P göstergesi yerel kullanım için */
      /* bağımsızlaştı. */
10     P := P + FARK;
11     BOY(P) := N - FARK;
12  ALTBÖLGEATA BİTTİ;

```

Bu algoritmalarda dikkatimizi çekecek bir nokta P göstergesinin bellek atamadan sonra ilk başlangıç noktasına (başlık ögesi) götürülmemesidir. Bu önlem bellek atamalar nedeniyle küçülen

bellek bölümlerinin listenin başlangıcında toplanmaması içindir. İkinci dikkat çekici nokta algoritmanın kullanıcı programı için istenen bellek boyunu sağlayan 'ilk serbest alanı bul' durumunda işletim sistemine atama için bu alanı önermesidir. Gereksinen belleği sağlayan en küçük, 'en uygun serbest alanı bul' algoritması kullanılmamıştır. Bu algoritmaların tartışma ve çözümlemesi işletim sistemi alt alanının konusudur, bu kapsamda yalnızca çift bağlı döngülü liste veri yapısı, veri yapıları kapsamı içinde ele alınmaktadır.

```

1 BELLEKBIRAK(P, SERBEST) YORDAMI;
2 KOMUTLAR;
3   N := BOY(P);
4   EGER İŞARET(P + N) := 1 ;/* Alttaki komşu serbest değil*/
5   İSE [ EGER İŞARET(P - 2 * GÖSTERGEBOYU) := 1
6       İSE BUALANIBIRAK(P, SERBEST) ÇAĞIR;
          /* Üstteki komşu da serbest değil */
7       DEĞİLSE SOLKOMŞUYLA(P, SERBEST) ÇAĞIR ]
8   DEĞİLSE [ EGER İŞARET(P - 2 * GÖSTERGEBOYU) := 1
9       İSE SAĞKOMŞUYLA(P, SERBEST) ÇAĞIR
10      DEĞİLSE HERİKİKOMŞUYLA(P, SERBEST) ÇAĞIR ]
11 BELLEKBIRAK BİTTİ;

```

```

1 BUALANIBIRAK(P, SERBEST) YORDAMI;
2 KOMUTLAR;
3   İŞARET(P) := 0;

```



```

4  İSARET(P + BOY(P) - 2 * GÖSTERGEBOYU) := 0;
5  ÜSTBAĞ(P + BOY(P) - GÖSTERGEBOYU) := P;
6  ÇİFTBİSAGEKLE(P, SERBEST) ÇAĞIR;
7  BUALANIBIRAK BİTTİ;

```

```

1  SOLKOMSUYLA(P, SERBEST) YORDAMI;
2  KOMUTLAR;
3  Q := ÜSTBAĞ(P - GÖSTERGEBOYU) ;
4  BOY(Q) := BOY(Q) + BOY(P) ;
5  ÜSTBAĞ(P + BOY(P) - GÖSTERGEBOYU) := Q;
6  İSARET(P + BOY(P) - 2 * GÖSTERGEBOYU) := 0;
   /* Serbest listesine yeni öge eklenmiyor, eski ögeler- */
   /* den birinin boyu büyüyor.                               */
7  SOLKOMSUYLA BİTTİ;

```

Yukarıdaki SOLKOMSUYLA yordamı en pratik biçimde yeni alanı var olan altına eklemektedir. Bu işlem için yeni alanın alttaki işareti ve üstbağı değişmeli, buna komşu alanın da boy bilgisi belirlenmelidir. Yukarıdaki algoritmalarda Q ise sadece ara değişkendir.

```

1  SAĞKOMSUYLA(P, SERBEST) YORDAMI;
2  KOMUTLAR;
3  Q := P + BOY(P);
4  ÇİFTBÖLGESİL(Q, SERBEST) /* SAĞ komşuyu SERBEST */
   /* listesinden sil .    */

```

```

5  ISARET(P) := 0,
6  USTBAG(Q + BOY(Q) - GOSTERGEBOYU) := P;
7  BOY(P) := BOY(P) + BOY(Q) ;
8  CIFTBSOLEKLE(P, SERBEST) /*Simdi bütünleşmiş blogu ekle*/
9  SAGKOMSUYLA BITTI;

```

SAGKOMSUYLA algoritması sağdaki (alttaki) komşuyla birleştirebilmek için önce bu eleman Q geçici değişkeni ile işaret edilmekte daha sonra serbest listesinden silinmektedir. Simdi ikiside serbest listesine bağlı olmayan biri P diğeri Q ile işaret edilen bu alanları birleştirme işlemi yapılacaktır. Üstteki alanın işaretinin sıfır (serbest) duruma getirilmesi, alttaki (daha önce serbest kalan) alanın üst bağının bu yeni alanın başına yöneltilmesi ve yeni alanın boy bilgisinin belirlenmesi bu aradaki işlemlerdir. Öte yandan HERİKİKOMSUYLA yordamı alttaki komşunun serbest listesinden silinmesini ve üçlü birleşimin zaten var olan sol (üstteki) komşunun bağlaç alanlarıyla ulaşılmasını sağlamaktadır. Ayrıca son komutta da göreceğimiz gibi alanların birleşmesi yanın- da 12 gösterge alanı daha kazanç sağlamaktadır.

```

1  HERİKİKOMSUYLA(P, SERBEST) YORDAMI;
2  KOMUTLAR;
3  Q := P + BOY(P); /* Sağ komşuya göre gösterge */
4  R := USTBAG(P - GOSTERGEBOYU); /* Sol komşuya gösterge */

```



```
5 ÇİFTBÖGESİL(Q, SERBEST);/* Sağ komşuyu serbest listesinden
                               sil */
```

```
6 BOY(R) := BOY(R) + BOY(P) + BOY(Q) + 12 * GÖSTERGEBOYU;
```

```
7 HERİKİKOMŞUYLA BİTTİ;
```

3.5 BAĞLACSIZ DÖNGÜLÜ KUYRUKLAR

İnsan yaşamında kuyruklar sık karşılaşılan olaylardır. Maç bileti kuyruğu, telefon faturası ödeme kuyruğu, jetonlu telefon kuyruğu yaşamımızın bir parçasıdır. Bilgisayar işletim sisteminde ve sistem yazılımının öteki bölümlerinde de kuyruklar sıkça kullanılır. Ancak insan kuyrukları ve yazılım ile benzetimleri arasında önemli bir fark vardır. İnsan kuyruklarında öndeki birey beklediği servisi alıp kuyruktaki yerinden ayrılınca kuyruktakiler birer adım öne kolayca, istekli biçimde yürürler yazılım içinde oluşturulmuş olan kuyruklarda bu kendiliğinden ve kolay uygulanır nitelik olmadığından iki olasılık akla gelir.

- i) Her kuyruk ögesini bir öndeki yere kaydırmak
- ii) Kuyruk önünü işaretleyen levhayı (Göstergeyi) bir öge geriye çekmek

Birinci seçeneğin sakıncası kuyruktaki ögelerin sayısı kadar kaydırma işleminin kuyruktan her ayrılma olayında yapılmasıdır. Kuyruğa giren ögelerin, ortalama kuyruk boyu 1 ise ($n.1$) kadar öge kaydırma işlemi yapılması gerekir. Bir ögeye ilişkin bilgiler bir dizi kayıt ta olduğunda bu işlem zaman alıcıdır. Bu tür bir algoritmanın almagoritma karmaşıklığı her eleman için $O(n)$ olarak belirlenir.

ikinci seçenekte kuyruğun ön göstergesi devamlı arkaya kayacak, başka birimler alınmazsa, bu kuyruğu belirlemek için kullanılan tek boyutlu dizinin sonuna kadar ulaşabilecektir. Kullanılabilir yerler tükenirken göstergenin önünde kullanılmayan alanlar oluşacaktır. Ancak algoritmanın algoritma karmaşıklığı $O(1)$ yani diğer algoritmaya göre n kat hızlıdır.

ikinci seçeneğin sakıncasını ortadan kaldırmak daha kolaydır. Kuyruğun (Queue) ÖN (front) ve ARKA (rear) göstergeleri bu kuyruk için ayrılmış tek boyutlu dizi üzerinde döngülü olarak kullanılabilir. Şimdi bu kuyruğu tanımlayalım:

TEKİL BAĞLAÇSIZ DÖNGÜLÜ KUYRUK

a) Bağlaçsız Döngülü Kuyruk Yapısı Tanımı:

Tanım 1 : ÖN = ARKA Boş kuyruk tanımıdır.

Tanım 2 : Kuyruk ilk kez kullanıma açılırken
ÖN = ARKA = 1 ile başlanır.

Aşağıda örnek bir kuruluş algoritması verilmiştir.

```

1 BDZKUYRUKKUR(N) YORDAMI;
2 İÇYAPILAR;
3 K(N) TÜR(TAMSAYI, KARAKTER), EVRENSEL;
4 ÖN, ARKA K ÜZERİNDE GÖSTERGE, EVRENSEL;
5 KOMUTLAR;
6 ÖN := 1;
7 ARKA := 1;
8 BDZKUYRUKKUR BİTTİ;
```


Tanım 3 : DN göstergesi dolu olan ögenin bir önündeki yeri işaret eder.

Tanım 4 : DN ve ARKA göstergeleri her ikisi de saat kadranı yönünde hareket eder. Kuyruğun önündeki ögeyi almak için şu algoritma uygulanır.

```

1 BDZKUYRUKTANAL(DN, VERİ) MANTIKSAL YORDAMI;
2 DISYAPILAR;
3   K(*); DN, ARKA; /* Bu değişkenler daha önce tanımları */
                      /* gibidir. */
4 KOMUTLAR;
5   EGER DN = ARKA
6     ISE [ /* kuyrukta okunabilecek geçerli bilgi yok */
7       BDZKUYRUKTANAL := YANLIS ]
8   DEĞİLSE [ DN := DN + 1;
9     EGER DN > N - 1
10     ISE [ DN := 0 ]
11     VERİ := K(DN);
12     BDZKUYRUKTANAL := DOGRU ]
13 BDZKUYRUKTANAL BİTTİ;

```

Tanım 5 : $ARKA + 1 = DN$ durumu kullanılabilir tüm ögeleri dolu kuyruk belirtilir ($ARKA = N$ oluncaya kadar kullansa idik taşma durumundaki kuyruk ile boş kuyruğu ayıramazdık). Buna göre ($0 : n - 1$) indisleri ile belirtilen dizi üzerinde n öge vardır, fakat ençok $n - 1$ 'i kuyruk için kullanılabilir.

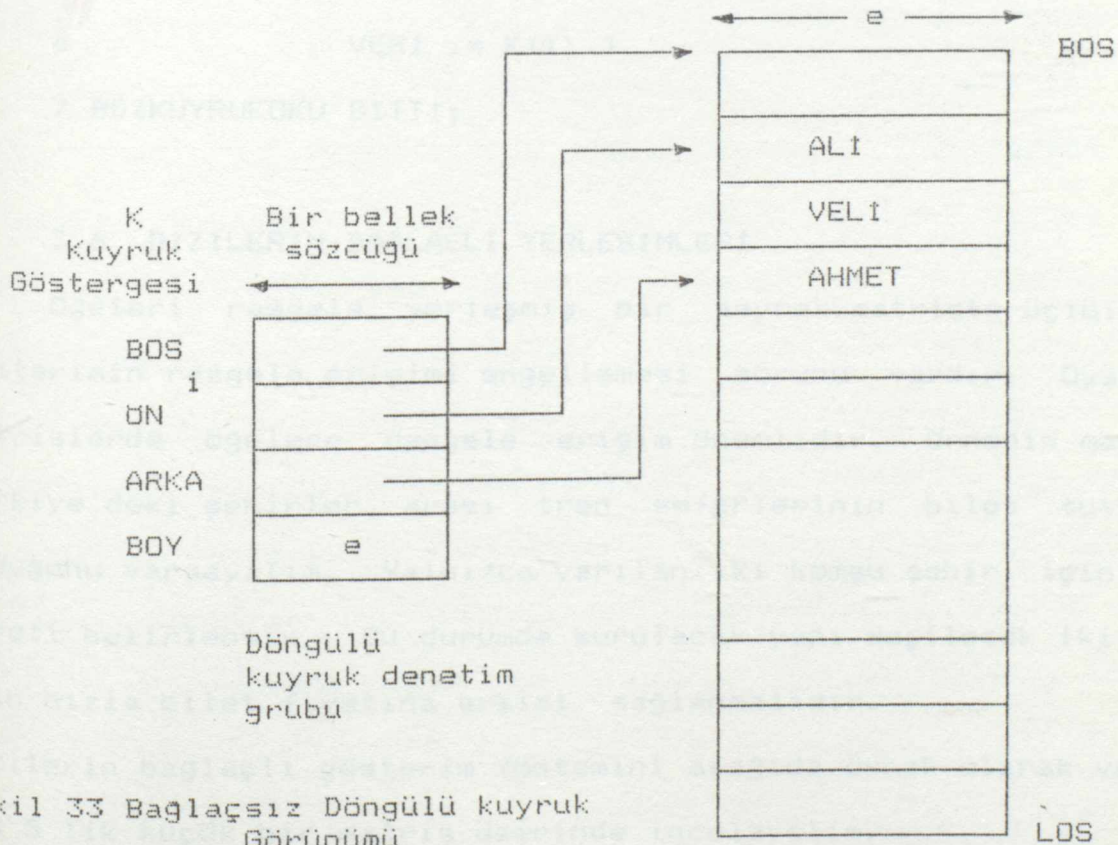
şimdi kuyruğa ekleme algoritmasını yazalım:

```

1 BDZKUYRUGAEKLE (ARKA, VERİ) YORDAMI;
2 KOMUTLAR;
3   EĞER DN = MOD(ARKA + 1, N)
4     İSE [ /* kuyruk dolu yeni eleman almaz */
5       BDZKUYRUGAEKLE := YANLIŞ; ]
6   DEĞİLSE [ ARKA := MOD (ARKA + 1, N);
7     VERİ := K( ARKA )
8     BDZKUYRUGAEKLE := DOĞRU ]
9 BDZKUYRUGAEKLE BİTTİ;

```

b) Bağlaçsız Döngülü Kuyruk Yapısı Gösterimi:



Şekil 33 Bağlaçsız Döngülü kuyruk Görünümü

c) Bağlaçsız döngülü kuyruk erişimi önce kuyruk göstergesi ile döngülü kuyruk denetim grubuna yapılır. Burada erişimin geçerlilik sınaması yapılır. Erişim geçerli ise kuyruk ögesine erişilir.

d) Bağlaçsız döngülü kuyruk ögesine erişim ÖN (front) ve ARKA (rear) göstergeleri yardımıyla yapılır. Aşağıda kuyruğa ekleme yada çıkarma yapmayan sadece ön ögeyi okuyan bir algoritma vardır.

```
1 BDZKUYRUKOKU(ÖN, VERİ) MANTIKSAL YORDAMI;
```

```
2 KOMUTLAR;
```

```
3 EĞER ÖN = ARKA
```

```
4     İSE [ BDZKUYRUKOKU := YANLIŞ;
```

```
        /* Kuyruk boş */
```

```
5     DEĞİLSE [ T := MOD(ÖN + 1, N)
```

```
6         VERİ := K(T) ]
```

```
7 BDZKUYRUKOKU BİTTİ;
```

3.6 DİZİLERİN BAĞLAÇLI YERLEŞİMLERİ

Ögeleri rasgele yerleşmiş bir seyrek matriste üçlüler ile gösterimin rasgele erişimi engellemesi sorunu vardır. Oysa bazı matrislerde ögelere rasgele erişim önemlidir. Örneğin matrisin Türkiye'deki şehirler arası tren seferlerinin bilet tutarları olduğunu varsayalım. Yalnızca varılan iki komşu şehir için bilet ücreti belirlensin. Bu durumda kurulacak yapı seçilecek iki şehir için hızla bilet fiyatına erişimi sağlanmalıdır.

Dizilerin bağlaçlı gösterim Yöntemini aşağıda örnek olarak verilen 5 x 5 lik küçük bir matris üzerinde inceleyelim.

a) Veri Yaapısı Tanımı

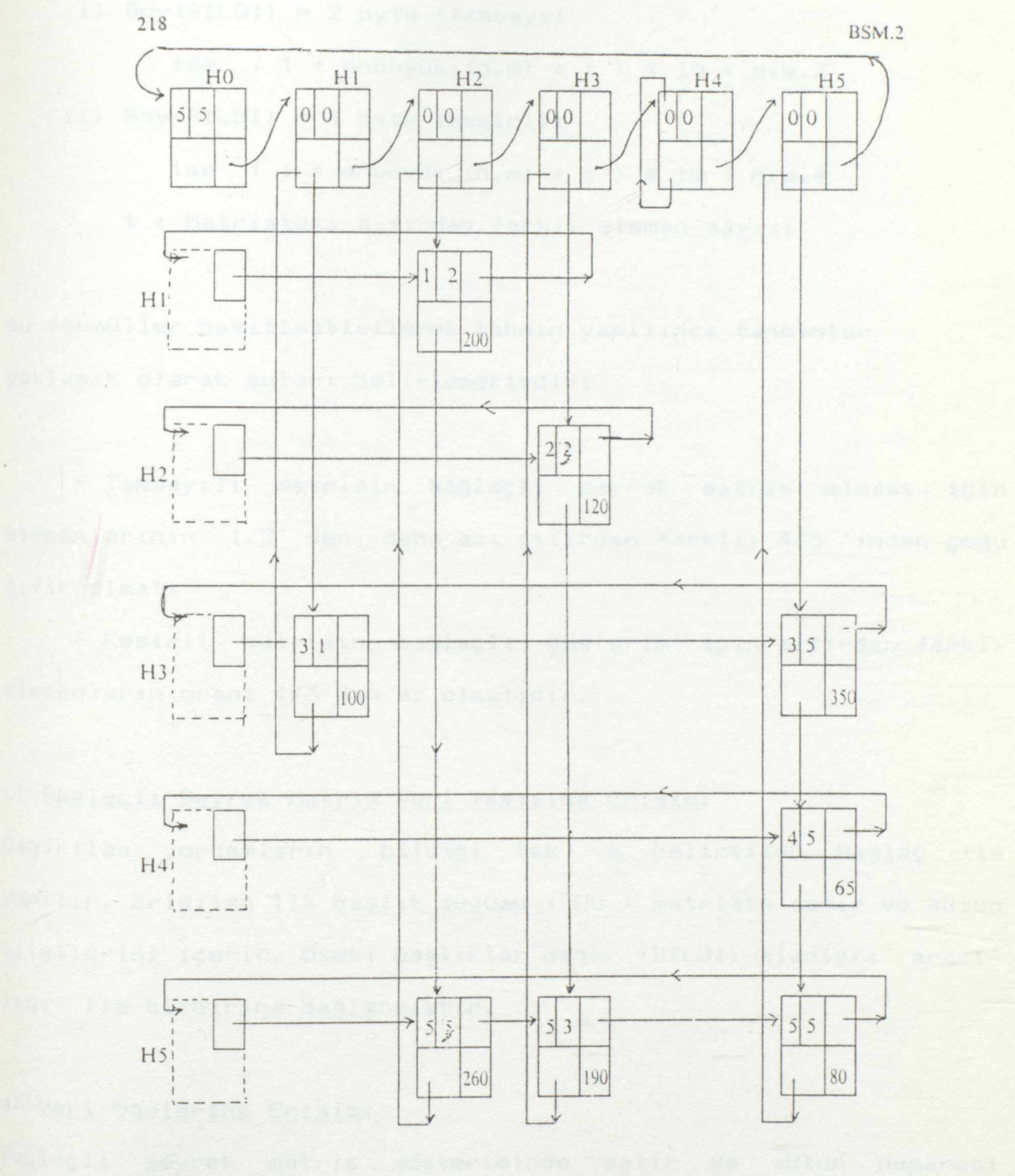
0	200	0	0	0
0	0	120	0	0
100	0	0	0	350
0	0	0	0	165
0	260	190	0	180

Seyrek matrisin programcı tarafından kavramsal düzeyde algılanışı normal matris tanımından farklı değildir.

b) Veri Yapısının Gösterimi

Bağlaçlı seyrek matris gösterimi her veri ögesinin DEGER'i yanında SATIR, SÜTUN indislerini, aynı satırdaki ögeleri birbirine bağlamak için SAĞBAĞ gösterge alanını, aynı sütundaki ögeleri bağlamak üzere ASAGIBAĞ gösterge alanını içeren elemanlardan oluşan bir yapıdır. Bu biçimde bağlanmış yapıya ya satır sütun için gösterge dizilerinden yada aynı yapı içindeki başlık ögelerinden erişilebilir. Her satır listesi ve her sütun listesi kendi içinde döngülü liste biçiminde düzenlenmiştir. başlık için kullanılan ögeler de kendi içinde o grup içinde kullanılmayan DEGER alanları ile döngülü liste yapısında örgütlenmiştir. Ayrıca matrisin kaç satır ve kaç sütünlük olduğu eklenen fazla dam bir H0 başlığı içinde gösterilmiştir. Aşağıda bir düzey indirgenmiş örnek setrek matris gösterimi bağlaçları ile verilmiştir. Burada dikkatimizi çekecek nokta sol taraftaki başlıklar ile üst taraftaki başlıkların sembolik adlarla (H1, H2, H3,...) ile gösterilmesidir. Gerçekte aynı sembolik adı taşıyan birtek başlık vardır, Başlığın iki kez çizilmesi yapının kolay canlandırılması içindir.

Özellik tanımları ve konuları için belirlenmiş yapılar aynı yapıya göre tanımlanmıştır.



Sekil 34 Bir Seyrek Matrisin Bağlaçlı Gösterimi

Şimdi tamsayılı ve kesirli veri için bellek kazancını ayrı ayrı inceleyelim:

i) Boy(BİLGİ) = 2 byte (tamsayı)

$$\text{ise } (1 + \text{enbüyük } (n,m) + t) * 10 < n.m.2$$

ii) Boy(BİLGİ) = 4 byte (kesirli)

$$\text{ise } (1 + \text{enbüyük } (n,m) + t) * 12 < n.m.4$$

t : Matristeki sıfırdan farklı eleman sayısı

Bu formüller basitleştirilerek tahmin yapılınca tanminler yaklaşık olarak şuları belirlemektedir:

- Tamsayılı matrisin bağlaçlı seyrek matris olması için elemanlarının $1/5$ ' den daha azı sıfırdan farklı, $4/5$ 'inden çoğu sıfır olmalı

- Kesirli matrisin bağlaçlı gösterimi için sıfırdan farklı elemanların oranı $1/3$ den az olmalıdır.

c) Bağlaçlı Seyrek Matris Veri Yapısına Erişim:

Çağırılan yordamların bildiği tek ve belirtilen bağlaç ile yapılır. Erişilen ilk başlık düğümü (H0) matrisin satır ve sütun bilgilerini içerir. Öteki başlıklar değer (BİLGİ) alanları aracılığı ile birbirine bağlanmıştır.

d) Veri Ügelerine Erişim:

Bağlaçlı seyrek matris gösteriminde satır ve sütun numarası belirtilen (i,j) elemanının değerine erişim şöyle yapılabilir:

Eğer $i < j$

ise H_i başlık ögesini ara, sonra 'aşağıbağ' alanlarını kullanarak istenen elemanı bul ;

değilse H_j başlık ögesini ara, sonra 'sağbağ' alanlarını kullanarak istenen ögeyi bul ;

Bu erişim rasgele değilse de yaklaşık $1/2 \cdot \text{enbüyük}(n,m) + 1/2 \cdot k$ arama ile (k satır yada sütundaki ortalama öge sayısı) istenen ögeye ulaşılabilir.

Eğer başlık düğümlerinin adresleri tek boyutlu bir dizi içinde tutulursa bu arama sayısı $1 + 1/2 \cdot (t/n)$ ($n > m$ için) tamsayı bilgi düzeyine indirilebilir. Üçlüler ile gösterimde ortalama arama zamanı olan $t/2$ ye göre bu oldukça hızlı bir erişim zamanıdır. Mühendislik türü yazılım ile uğraşan yazılım evlerinde (software house) büyük boyutlu seyrek matris sık sık gündeme gelir. Boyutu 300×100 , 400×200 , 150×250 gibi seyrek matrisler olağandır.

Örnek :

$n = 160$, $m = 80$, $t = 1650$ veri türü kesirli

Üçlüler ile gösterimde ortalama arama zamanı $1650 \cdot (1/2)$

karşılaştırma' dır.

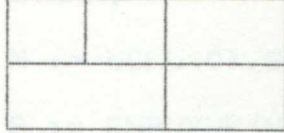
Vektörlü bağlaçlı seyrek matris gösteriminde

$$k = \frac{t}{\text{enbüyük}(n,m)} = 10.4$$

Ortalama arama = $1 + (1/2) \cdot 10.4 = 6$ karşılaştırma süresi kadardır.

Bağlaçlı seyrek matris gösterimi için kullanılacak veri yapısının her düğümü aşağıdaki gibi düzenlenmelidir:

SATIR SUTUN



SAGBAG

BİLGİ

AŞAĞIBAG

Eğer seyrek matris gösterimi önerilen matris $n \times m$ boyutlu ise ve t adet sıfırdan farklı bilgi içeriyorsa bağlaçlı gösterimde kullanılacak düğüm sayısı şöyle belirlenir:

$$1 + \text{enbüyük}(n, m) + t$$

Her bir düğümün boyu $n, m < 255$ ise 3 gösterge alanı + BİLGİ alanı uzunludur. Eğer $n, m \geq 255$ ile üstten bağlı değilse (veya makine etkinliği için karakter gösterimi tercih edilmiyorsa) 4 gösterge alanı ve BİLGİ alanının uzunluğu düğümün boyunu oluşturur. Bağlaçlı seyrek matrisin bellek yönünden avantajlı olması bu nedenle aşağıdaki koşulun sağlanmasına bağlıdır:

$$(1 + \text{enbüyük}(n, m) + t) * (4 * 2 + \text{boy}(\text{BİLGİ})) < n * m * \text{boy}(\text{BİLGİ})$$

Şimdi seyrek matrisin üçlü gösterimini vektörlü bağlaçlı seyrek matrise dönüştürecek algoritmayı yazalım:

- 1 UÇLUSEYREKDÖNÜŞÜMÜ(UÇLU, A, VEKTÖR) MANTIKSAL YORDAMI;
- 2 DİŞYAPILAR;
- 3 VEKTÖR(*) GÖSTERGE ;
- 4 UÇLU(0:*)(SA, SU, DEĞERÜÇLÜ TAMSAYI) YAPI;
- 5 K(*) (SAT, SUT, SAGBAG, AŞAĞIBAG, DEĞER TAMSAYI) YAPI;


```

6   SERBEST GÖSTERGE,
7   İÇYAPILAR:
8   YARDIMCI(*) GÖSTERGE;
9   KOMUTLAR;
10  N := ÜÇLÜ(0).SA; M := ÜÇLÜ(0).SU; R := ÜÇLÜ(0).DEĞER;
11  P := ENBÜYÜK(N,M); ÜÇLÜSEYREKDÖNÜŞÜMÜ := DOĞRU;
12  SAYARAK YİNELE (I := 1, I > P, I := I + 1);
13  [ EĞER DÖĞEBUL(SERBEST, X)
14      İSE [ VEKTÖR(I) := X;
15          K(X).SAT := 0; K(X).SUT := 0;
16          K(X).SAGBAG := X;
17          K(X).ASAGIBAG := X ]
18  DEĞİLSE [ 'DÖĞE BİTTİ' YAZ; ]
19          ÜÇLÜSEYREKDÖNÜŞÜMÜ := YANLIŞ; DÖN ]
20  ]
21  EĞER DÖĞEBUL(SERBEST, A)
22      İSE [ K(A).SAT := N;
23          K(A).SUT := M;
24          K(A).DEĞER := VEKTÖR(1);
25          SAYARAK YİNELE (I := 1, I = P, I := I + 1);
26          [ K(VEKTÖR(I).DEĞER := VEKTÖR(I + 1) ]
27          K(VEKTÖR(P)).DEĞER := A;
28          ] /* Başlık düğümleri düzenlendi ve kendi içinde */
           /* bağlandı */
29  DEĞİLSE [ 'DÖĞE BİTTİ ' YAZ;
30          ÜÇLÜSEYREKDÖNÜŞÜMÜ := YANLIŞ; DÖN ]

```

```

13      YARDIMCI(*) := VEKTÖR(*);

      /* Yardımcı o sütundaki son ögeyi gösterir. Bu aşamada*/
      /* Üçlüler dizisini birinci anahtar olarak satır,      */
      /* ikinci anahtar olarak sütun numarası ile sıralı      */
      /* ögeler alınacak                                     */

32      ŞİMDİKİSATIR := 1;

33      SON := VEKTÖR(ŞİMDİKİSATIR); /* Bu satırdaki son öge */
34      SAYARAK YINELE (I := 1, I > R, I := I + 1 );

35      [ EGER ÜÇLÜ(I).SA > ŞİMDİKİSATIR
36          İSE [ K(SON).SAGBAG := VEKTÖR(ŞİMDİKİSATIR);
37              ŞİMDİKİSATIR := ÜÇLÜ(I).SA;
38              SON := VEKTÖR(ŞİMDİKİSATIR);

      /* Bu satır sona erdi bu listeyi kapat, yenisine başla */
39      ]

40      EGER ÖĞEBUL(SERBEST, X)

41          İSE [ K(X).SAT := ÜÇLÜ(I).SA;
42              K(X).SUT := ÜÇLÜ(I).SU;
43              K(X).DEGER := ÜÇLÜ(I).DEGER;
44              K(SON).SAGBAG := X; /* aynı satırda sağa bağla*/
45              SON := X;

      /* Şimdi yeni eklenen son eleman oldu */

46          T := YARDIMCI(ÜÇLÜ(I).SU);
47          K(T).AŞAGIBAG := X; /* İçinde olduğu sütunda */
                                   /* aşağı bağla */
48          YARDIMCI(ÜÇLÜ(I).SU) := X;

49      ] /* bulunan öge bağlandı */

```



```

50     DEĞİLSE [ ÜÇLÜSEYREKDONUŞUMU := YANLIS; DÖN ]
51     ] /* R adet deger üçlüler biçiminden bağlaçlı yapıya*/
        /* dönüştü: Şimdi son satırı kapat */
52 EGER R ≠ 0
53     İSE [ K(SON).SAGBAG := VEKTÖR(SİMDİKİSATIR);
        /* Tüm kolonları kapat */
54     SAYARAK YİNELE (I := 1, I > M, I := I + 1);
55     [ T := YARDIMCI(I);
56     K(T).ASAGIBAG := VEKTÖR(I) ]
57     ] /* Son bağlaç alanlarıda kapandı */

/* Bu algoritma öğrenimi kolaylaştırmak amacıyla YARDIMCI
dizisini kullanmıştır. Eger başlık öğelerinin DEĞER
alanlarını algoritma başlangıcında belirlemezsek bu
alanı YARDIMCI dizisi yerine o sütundaki son öğeyi
göstermek üzere kullanabiliriz.*/

/* Algoritma etkin erişim amacıyla VEKTÖR dizisini
kullanıyor. Bu alan kullanılmadan, Başlık düğümlerinin
DEĞER alanlarından iz sürerek de erişim yapılabilir */
58 ÜÇLÜSEYREKDONUŞUMU BİTTİ;

```

Öte yandan var olan bir seyrek matrise yeni bir öge eklemek, toplama yada çıkarma işlemi yapmak her bir eleman için ayrı ayrı işleyen aşağıdaki algoritma ile yapılabilir:

```

1 SEYREKMATRİSEGEKLE(I, J, EKDEĞER, A) YORDAMI;
2 DISYAPILAR;

```

```

3  SERBEST;
4  K(*);
5  EKDEGER TAMSAYI;
6  I, J TAMSAYI;
7  ICYAPILAR;
8  P, Q, R GÖSTERGE;
9  SAY GÖSTERGE;
10 KOMUTLAR;
11  EGER K(A).SATIR > I
12      İSE K(A).SATIR := I; /* Bu algoritma yeni eklenen */
13  EGER K(A).SUTUN > J /* eleman önceki matrisin dışın-*/
14      İSE K(A).SUTUN := J; /* da ise matrisin boyutunu */
15  SAY := 0; P := A; PSUT := 0; PSAT := 0; /* değiştirir */
16  ÖNCE YİNELE SONRA (PSUT = 0 VE PSAT = 0) İSE ÇIK;
17  [ EGER K(P).DEGER = A
18      İSE [ EGER DÜĞEBUL(SERBEST,X)
19          İSE [ K(P).DEGER := X;
20              K(X).DEGER := A;
21              K(X).SATIR := 0;
22              K(X).SUTUN := 0;
23              K(X).ASAGIBAG := X;
24              K(X).SAGBAG := X ]
25  DEĞİLSE [ ('SERBEST YOK') YAZ; DÖN ]
26  ]
27  P := K(P).DEGER;
28  SAY := SAY + 1;

```



```

29     EGER SAY = I İSE PSAT := P;
30     EGER SAY = J İSE PSUT := P;
31     J /* bu noktada PSAT ve PSUT göstergeleri ilişkili */
        /* başlık düğümlerini işaret eder durumdadır */
32     Q := PSAT; /* Q uygun satırdan girip sağa ilerliyor */
33     ÖNCE (K(Q).SAGBAG = PSAT YADA
34         J > K(K(Q).SAGBAG).SUTUN) DEĞİL SINA SONRA YİNELE;
35     [ Q := K(Q).SAGBAG ]
36     R := PSUT; /* R uygun sütundan girip aşağıya iniyor */
37     ÖNCE (K(R).ASAGIBAG = PSUT YADA
38         I > K(K(R).ASAGIBAG).SATIR) DEĞİL SINA SONRA YİNELE;
39     [ R := K(R).ASAGIBAG ];
        /* Q ve P göstergeleri ekleme yapılacak ögeden önceki */
        /* düğümleri işaret edecek yere getirildi. ancak aynı */
        /* indisi taşıyan öge var ise değer arttırılacak, yok */
        /* ise yeni öge eklenecektir */
40     EGER J = K(K(Q).SAGBAG).SUTUN VE
41         I = K(K(R).ASAGIBAG).SATIR
42         İSE [ W := K(Q).SAGBAG;
        /* Aynı imdislere sahip eleman bulundu */
43         T := K(W).DEĞER + EKDEĞER ; ]
44     DEĞİLSE T := EKDEĞER;
        /* Her iki durumda da T değişkeni I,J indisi
        /* son değerini saklar
45     EGER T = 0
46     İSE [ K(Q).SAGBAG := K(K(Q).SAGBAG).SAGBAG;

```

```

47      K(R).ASAGIBAG := K(K(R).ASAGIBAG).ASAGIBAG;
48      DGEGERIVER (W, SERBEST) CAGIR I
49      DEGIILSE ( EGER DGEBUL(SERBSET, X)
50          ISE I K(X).ASAGIBAG := K(R).ASAGIBAG;
51              K(X).SAGBAG := K(Q).SAGBAG;
52              K(R).ASAGIBAG := X;
53              K(Q).SAGBAG := X;
54              K(X).SATIR := I;
55              K(X).SUTUN := J;
56              K(X).DEGER := T; I
57 SEYREKMATRİSEÜGEKLE BİTTİ;

```

Son verilen algoritma seyrek matrisin kuruluşu için değil az sayıda öge değişikliği ekleme, silme, değer farklılaştırması (toplama, çıkarma) işlemleri için uygundur.

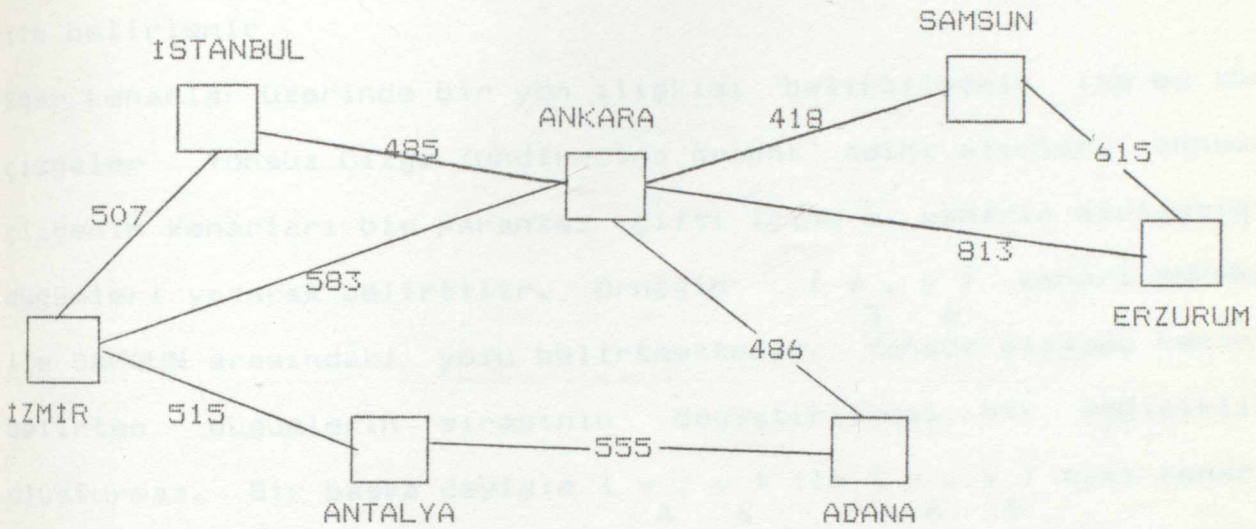
3.7 ÇİZGE VERİ YAPISI

Yönlü çizge, yönsüz çizge, ağaç ve ikili ağaç veri yapıları bilgisayar yazılımında çok kullanılan veri yapılarıdır. Bunlardan çizge veri yapısı bilgisayar alanı dışında öteki mühendislik ve iş yönetimi ile yöneylem alanında da ele alınan konulardır. Konunun genişliği nedeni ile bu bölümde tanım ve gösterimler verilecektir.

3.7.1 YÖNSÜZ ÇİZGE

Çizge (graph) yapısı hem günlük yaşantımızda hem de işyönetimi mühendislik ve benzeri dalların uygulamasında sıkça karşılaşılan

bir yapıdır. Örneğin Türkiye'de bazı illerin karayolu bağlantıları ile yolun uzunluğunu gösteren yapı bir yönsüz çizgedir.



Şekil 35 Yönsüz Çizge Örneği

İlk şekilde verilen yapıyı çizgenin teknik terimleri ile belirttiğimizde iki il arasındaki yollar 'kenar (edge)' olarak anılır. İki kenarın birleştiği yer 'düğüm (vertex)' olarak anılır. herhangi bir düğümü küçük harf ve o düğümün indisi ile örneğin v_4 biçiminde göstereceğiz. Çizgeyi oluşturan düğümlerin sırası önemli olmaksızın yalnız bir kez yazıldığı küme (set) 'yi de büyük harf V ile belirteceğiz. Örneğin yukarıdaki çizgede;

v_1 : İSTANBUL
 v_2 : İZMİR
 v_3 : ANKARA
 v_4 : ANTALYA

v_5 : ADANA
 v_6 : SAMSUN
 v_7 : ERZURUM

olarak belirlenirse.

$V(G) = \{ v_1, v_2, v_3, v_4, v_5, v_6, v_7 \}$ dir. Tek bir kenarı küçük

harf e ile kenarların oluşturduğu ' küme (set) ' ise büyük harf E ile belirlenir

Eğer kenarlar üzerinde bir yön ilişkisi belirtilmemiş ise bu tür çizgeler ' Yönsüz Çizge (undirected graph) ' adını alırlar. Yönsüz çizgenin kenarları bir parantez çifti içine o kenarın birleştiği düğümleri yazarak belirtilir. Örneğin (v_3, v_6) kenarı ANKARA ile SAMSUN arasındaki yolu belirtmektedir. Yönsüz çizgede kenarı belirten düğümlerin sırasının değiştirilmesi bir değişiklik oluşturmaz. Bir başka deyişle (v_4, v_6) ile (v_6, v_4) aynı kenardır. Ayrıca veri yapısının kolay kuruluşunu sağlamak ve veri yapısını kurarken gözden kaçan bir kenar olmadığını daha kolay denetlemek için indisi daha küçük olan düğüm önce yazılır. Bu yazım biçimine göre yukarıdaki örnekte kenar listesi şöyledir:

$$E = \{ (v_1, v_2, 507), (v_1, v_3, 458), (v_2, v_3, 583), (v_2, v_4, 515), (v_3, v_5, 486), (v_3, v_6, 418), (v_3, v_7, 883), (v_4, v_5, 555), (v_6, v_7, 665) \}$$

Bu ilişkiyi bilgisayar belleğinde gösterim biçimlerinden biri matris olabilir. Bu durumda matris simetrik olduğundan bir üst üçgen matris kesimi yerleştirmek ve alt üçgen kesimindeki bir öge

yerine simetrik ögeye erişmek yeterlidir. Örnekte verilen ' yönsüz çizge ' için aşağıdaki veri yapısı önerilebilir.

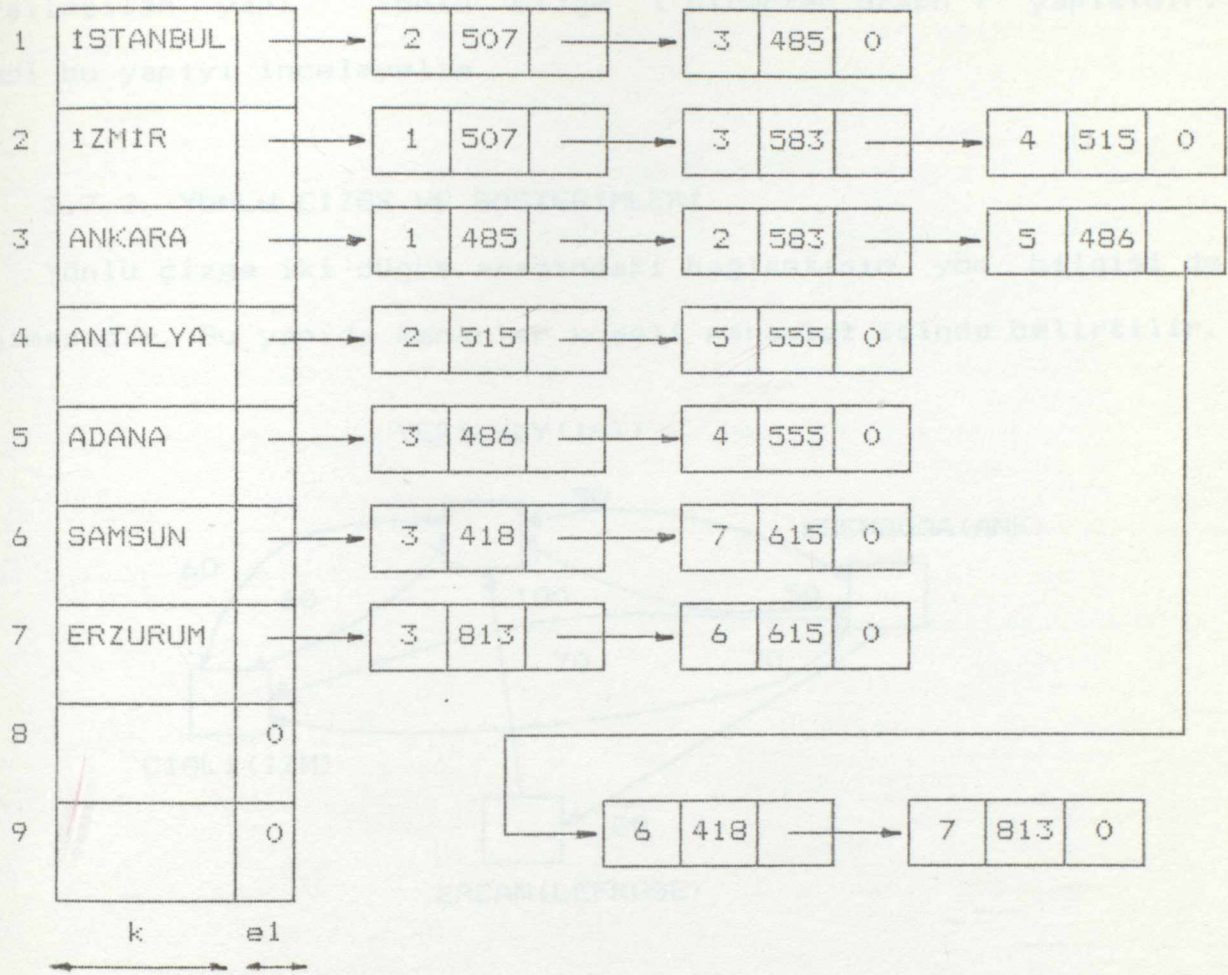
DÜĞÜMADİ		1	2	3	4	5	6	7
1	İSTANBUL	1	0	505	485	0	0	0
2	İZMİR	2		0	583	515	0	0
3	ANKARA	3			0	0	486	418
4	ANTALYA	4				0	555	0
5	ADANA	5					0	0
6	SAMSUN	6						0
7	ERZURUM	7						

k

N

Şekil 36 Yönsüz Çizgenin Maliyet Matrisi ile Gösterimi

Hatırlayacağımız gibi altüçgen yada üstüçgen matris tek boyutlu bir dizi üzerine rastgele erişim niteliği kaybedilmeksizin yerleştirilebilir. Yönsüz çizge'nin üzerinde kolay işlem yapılabilecek ikinci gösterimi her kenarın (edge) ikikez yer aldığı ' komşuluk - Maliyet listeleri' yöntemidir. Aşağıda bu yöneme göre gösterim verilmiştir. Bu yöntemde de birinci düğüm no ikinciden büyük olduğunda bunların yerini değiştirerek erişim yapılabilir. Bu durumda ögelerin yarısı azaltılabilir. Bu yeni gösterimde rastgele erişim olanagı ortadan kalkmıştır. Ancak her düğümden az sayıda kenarın çıktığı çizgelerde fazla zaman kaybı olmadan, birkaç karşılaştırma ile istenen bilgiye ulaşılabilir.



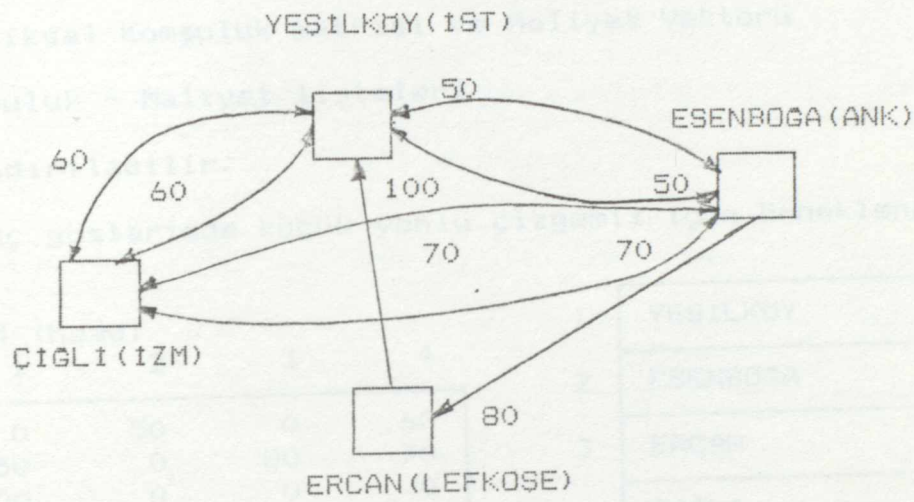
Sekil 37 Yönsüz Çizgenin Komşuluk Maliyet Listeleri ile Gösterimi

Bu veri yapısında da $n * (k + e1) + 2 * t * (2 * e1 + e2)$ Karakterlik bellek yeri alacaktır. Burada t çizgedeki kenar sayısıdır. Bu yapılarda $e1 = 2$, $e2 = 4$ karakterlik alanlar olarak alınırsa ($e1$ bağ alanı boyu, $e2$ bilgi alanı boyu) olduğu durumlarda üçgen matris gösterimi avantajlıdır. Pratikte bir düğümden çıkan kenarların ortalama sayısı düğüm sayısının dörtte birini geçerse üst üçgen matrisin avantajlı olacağı kabul edilebilecektir. Öte yandan gerçek uygulamalarda daha çok

karşılaşılan yapı 'Yönlü Çizge (Directed Graph)' yapısıdır. Şimdi bu yapıyı inceleyelim.

3.7.2 YÖNLÜ ÇİZGE VE GÖSTERİMLERİ

Yönlü çizge iki düğüm arasındaki bağlantının yön bilgisi de taşımasıdır. Bu yapıda kenarlar köşeli parantez içinde belirtilir.



Şekil 38 Yönlü Çizge Tanım Örneği

$$\begin{aligned}
 E(G) = \{ & \langle 1,2 : 50 \rangle, \quad \langle 1,4 : 60 \rangle, \\
 & \langle 2,1 : 50 \rangle, \quad \langle 2,3 : 80 \rangle, \\
 & \langle 2,4 : 70 \rangle, \quad \langle 3,1 : 100 \rangle, \\
 & \langle 4,1 : 60 \rangle, \quad \langle 4,2 : 70 \rangle \}
 \end{aligned}$$

$$V(G) = \{ 1, 2, 3, 4 \}$$

Verilmiş olan yönlü çizge ESENBOĞA - YEŞİLKÖY uçuşunun her iki yönde de olduğunu ve 50 dakika sürdüğünü ESENBOĞA - ERCAN' ın 80 dakikalık tek yönlü bağlantısını ve öteki bağlantıları

göstermektedir. Yönlü çizge tanımında kenar kuyruk (tail) ve ön (head) kenarın < kuyruk,ön > biçiminde yazılışı ile belirtilmektedir.

3.7.2.1 Yönlü Çizgenin Olası Gösterimleri:

Kavramsal yapısı yukarıda verilen yönlü çizgenin belli başlı üç bellek içi gösterimi vardır. Bu yöntemler ;

i) Komşuluk - Maliyet Matrisi

ii) Mantıksal Komşuluk matrisi ve Maliyet Vektörü

iii) Komşuluk - Maliyet listeleri

olarak adlandırılabilir.

Aşağıda bu üç gösterimde küçük yönlü çizgemiz için örneklenmiştir.

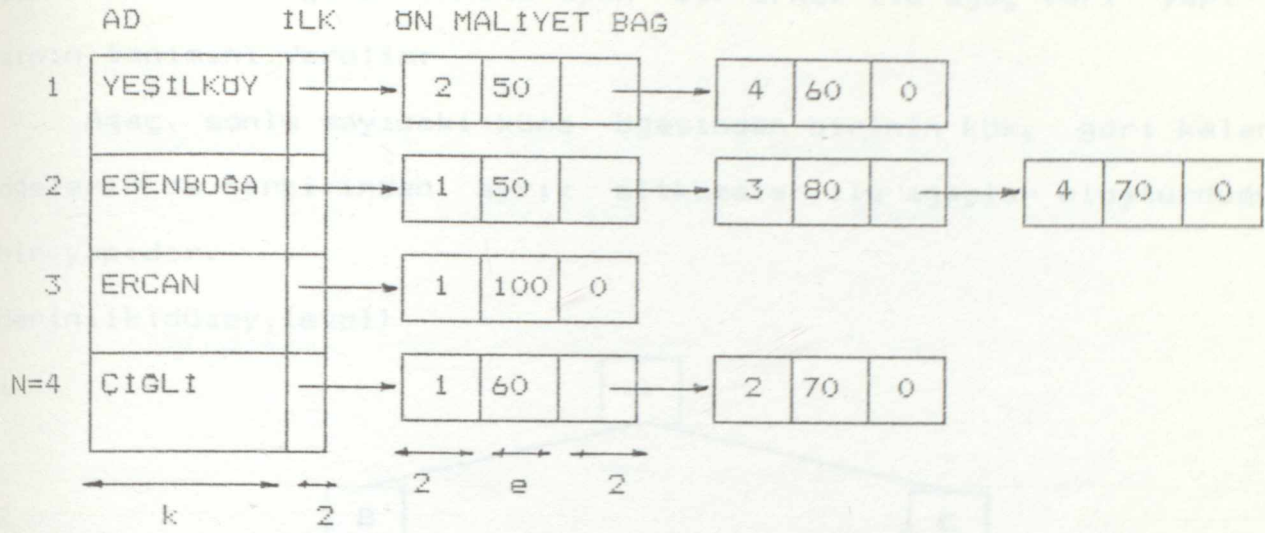
		ÖN (head)					
		1	2	3	4		
KUYRUK (tail)	1	0	50	0	60	1	YEŞİLKÖY
	2	50	0	80	70	2	ESENBÖĞA
	3	100	0	0	0	3	ERCAN
	4	60	70	0	0	4	ÇİĞLİ

Sekil 38 - 1 Yönlü Çizgenin Maliyet Matrisi ile Gösterimi

Gerçek uygulamalarda Komşuluk - Maliyet Matrisleri genellikle seyrek matris niteliği gösterir. Tipik boyutlar 100 x 200, 300 x 300, 150 x 300 gibi komşuluk maliyet matrisini genellikle tercih ettirmeyecek boyutlardır.

3.7.2.2 Mantıksal Komşuluk Matrisi ve Maliyet Vektörü

G çizgesinde n adet düğüm ve t adet kenar var ise önceki yöntem $k.n + e.n$ lik bellek gerektirir ve rastgele (random) erişim sağlar. Bir başka yöntem şudur:



Sekil 38 - 3 Yönlü Cizgenin Komşuluk Maliyet Listeleri ile Gösterimi

Yönlü çizgede bazı çözümlemelerin yapılabilmesi için her düğüm için 'gelen kenar sayısı' ve 'giden kenar sayısı' belirlenir. Örnek için bu tablo aşağıdadır.

	Düğüm	Gelen	Giden
YEŞİLKÖY	1	3	2
ESENBÖĞA	2	2	3
ERCAN	3	1	1
ÇİĞLİ	4	2	2

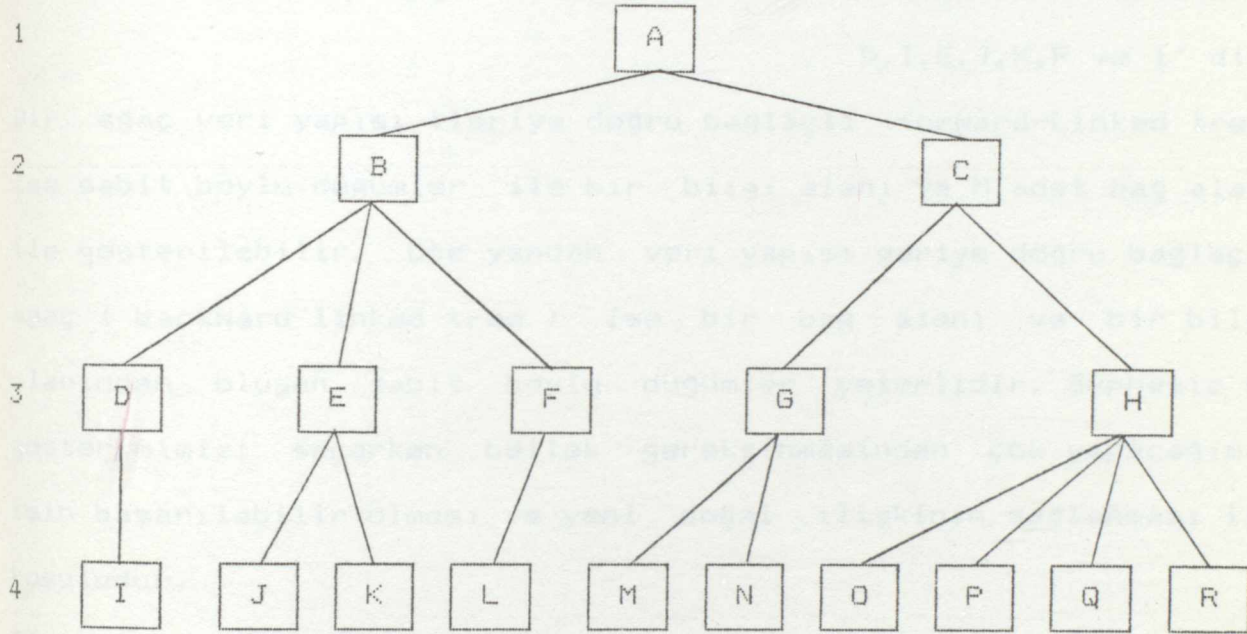
3.8 AĞAÇ VERİ YAPISI

Ağaç veri yapısı (tree structure) tanımı yada ikili ağaçlar (binary tree) ile oyun ve bulmaca ağaçları olmak üzere üç tür ağaç veri yapısı söz konusudur.

şimdi bunlardan genel tanıma uyan bir örnek ile ağaç veri yapısının tanımını verelim:

Ağaç, sonlu sayıdaki küme ögesinden birinin kök, geri kalan öğelerin de birbirinden ayrık altkümeler ile ağaçlar oluşturduğu bir yapıdır.

derinlik(düzey, level)



Sekil 39 Ağaç Veri Yapısı Örneği

Yukarıdaki örnekte 4 derinliğinde bir ağaç örneği verilmiştir. Bu yapıya C' den K' ya bir kenar eklense idi ayrık küme özelliği bozulacak ve yapı ağaç değil çizge olacaktı. Bir ağaçta her düğümün bir derecesi (degree) vardır. Bu derece o düğümün alt düğümleri sayısıdır. Yukarıdaki örnekte A 2, B 3, H 4' üncü dereceden düğümlerdir. Ağacın derecesi, bir ağacın düğümlerinin en çok derecesi olarak belirlenir. Buna M' diyelim. Burada ağacın derecesi H düğümünün derecesi olan 4 nedeniyle dört olarak belirlenir.

Agacın düğümleri arasındaki bazı ilişkiler şöyle belirlenir.

- i) Oglu (begin a son of) : B düğümü A'nın oğludur.
- ii) Babası (begin a father of) : B düğümü E'nin babasıdır.
- iii) Atası (begin a predecessor of) : K düğümünün ataları E, B ve A' dir.
- iv) varisleri (begin a successor of) : B düğümünün varisleri D, I, E, J, K, F ve I' dir.

Bir ağaç veri yapısı ileriye doğru bağlaçlı (forward-Linked tree) ise sabit boylu düğümler ile bir bilgi alanı ve M adet bağ alanı ile gösterilebilir. Öte yandan veri yapısı geriye doğru bağlaçlı ağaç (backward linked tree) ise bir bağ alanı ve bir bilgi alanından oluşan sabit boylu düğümler yeterlidir. Şüphesiz ki gösterimimizi seçerken bellek gereksinmesinden çok yapacağımız işin başarılabilir olması ve yeni doğal ilişkinin sağlanması ilk koşuldur.

Ağaç veri yapısında bazı düğümlerin özel adları vardır:

- i) Kök (root of the tree)
- ii) uç düğümler (terminal nodes)

Eğer ağaç veri yapısı ileriye doğru bağlaçlı gösterimde ise kök hariç her düğüme yalnız 1 gelen kenar vardır. Kök için gelen kenar sayısı sıfırdır.

Eğer ağaç veri yapısı geriye doğru bağlaçlı gösterimde ise kök hariç her düğümden yalnız 1 adet giden kenar vardır. Kök için giden kenar sayısı sıfırdır. Bir ağaç veri yapısında bir köke bağlı alt ağaçların yerleri değiştirildiğinde yapı yine ağaçtır.

Şimdi bu ağacı iki yöntemle gösterelim:

T



1	F	13	0	0	0
2	B	11	6	1	0
3	H	18	16	15	17
4	A	2	7	0	0
5	N	0	0	0	0
6	E	10	14	0	0
7	C	8	2	0	0
8	G	9	5	0	0
9	M	0	0	0	0
10	J	0	0	0	0
11	D	12	0	0	0
12	I	0	0	0	0
13	L	0	0	0	0
14	K	0	0	0	0
15	Q	0	0	0	0
16	P	0	0	0	0
17	R	0	0	0	0
18	O	0	0	0	0

İleriye Doğru Bağlanmış
Ağaç Veri Yapısı Gösterimi

UÇDİZİ

5	15	12	9	10	17	16	18	14	13
---	----	----	---	----	----	----	----	----	----

Uç düğümler gösterge dizisi

1	F	2
2	B	4
3	H	7
4	A	0
5	N	8
6	E	2
7	C	4
8	G	7
9	M	8
10	J	6
11	D	2
12	I	11
13	L	1
14	K	16
15	Q	3
16	P	3
17	R	3
18	O	3

! KÖK DÜĞÜMÜ

Geriye Doğru Bağlanmış
Ağaç Veri Yapısı
Gösterimi

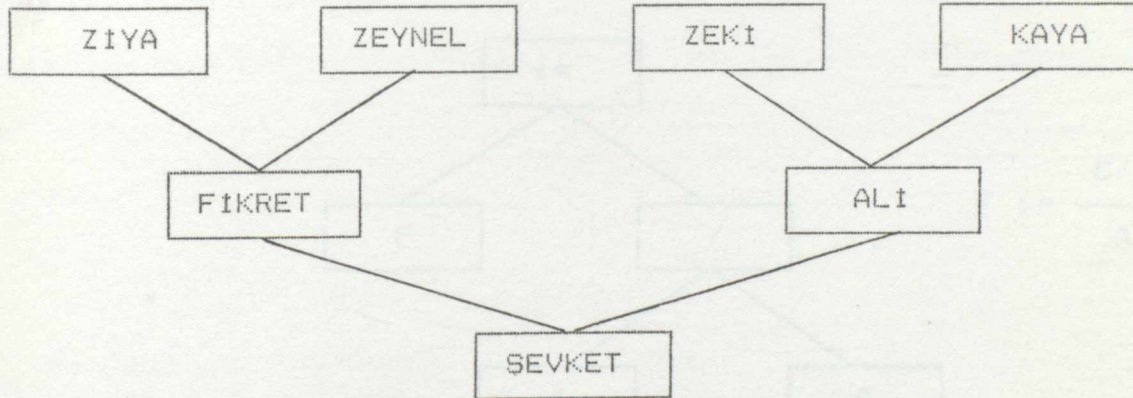
Şekil 40 Ağaç Veri Yapısı Olası Gösterimleri

3.9 İKİLİ AĞAC VERİ YAPISI

İkili ağaç (binary tree) veri yapısının yoğun biçimde kullanılır oluşu iki önemli uygulama alanı ile ilişkilidir. Sıralama algoritmaları ile derleyicilerde sözdizim çözümleme (syntax analysis), kod geliştirme ve amaç kod üretme algoritmaları bu veri yapılarını kullanır. Ayrıca kodlama kuramı, turnuva düzenleme, aile ağacı gösterimi ve benzeri tekil uygulamaları da sözkonusudur.

3.9.1 İkili Ağaç Veri Yapısı Tanımı

İkili ağaç veri yapısı kavramsal tanım düzeyinde, bellek içi gösterimden bağımsız olarak şöyle düşünülebilir:

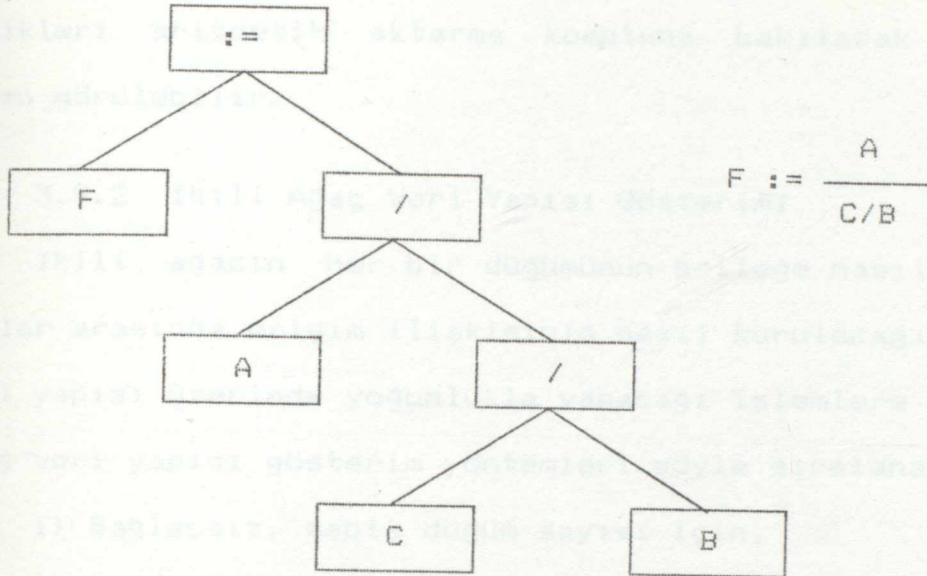


Sekil 41 Aile Ağacı Örneği

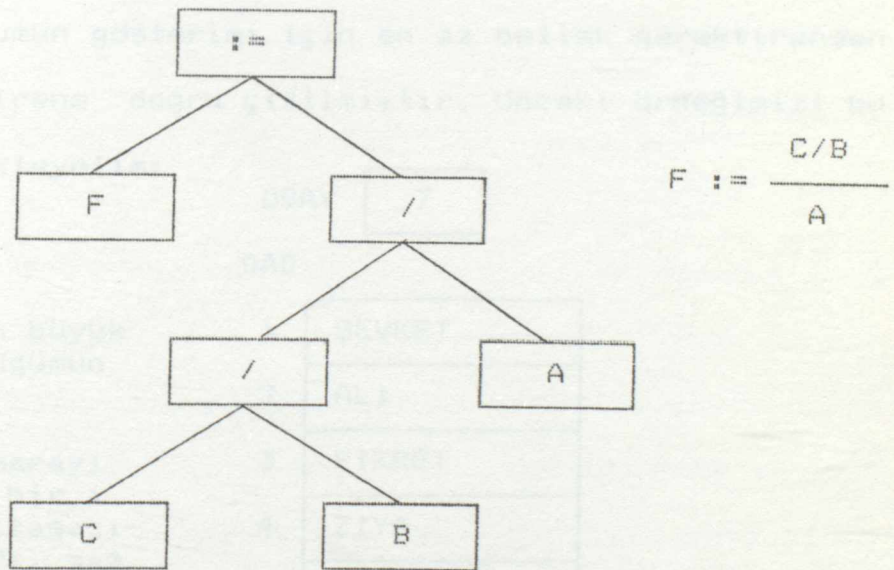
İkili ağaç veri yapısının tanımında ağaç veri yapısının tüm tanımları geçerlidir farklı olan iki nokta ise şunlardır:

- i) Ağacın derecesi (degree of tree) 2' dir.
- ii) Sol ve sağ alt ağacın yer değiştirmesiyle çizilen ağaç aynı ağaç değildir.

Simdi bunlari inceleyelim:



Sekil 42 - 1 Sözdizim Ağacı



Sekil 42 - 2 Önceki Sözdizim Ağacının Alt Ağaçlarının Yer Değiştirmiş Durumu

Şekil 42 - 2, şekil 42 - 1' in altağaçları yer değiştirilerek çizilmiştir. Tanımımıza göre bu iki ağaç özdeş değildir. İfade ettikleri aritmetik aktarma komutuna bakılacak olursa bu durum hemen görülebilir.

3.9.2 İkili Ağaç Veri Yapısı Gösterimi

İkili ağacın her bir düğümünün belleğe nasıl yerleşeceği ve bunlar arasında erişim ilişkisinin nasıl kurulacağı programcının o veri yapısı üzerinde yoğunlukla yapacağı işlemlere bağlıdır. İkili ağaç veri yapısı gösterim yöntemleri şöyle sıralanabilir.

- i) Bağlaçsız, sabit düğüm sayısı için,
- ii) Geriye doğru bağlanmış,
- iii) İleriye doğru bağlanmış,
- iv) Çift yönlü bağlanmış,

Bu yöntemler bir düğümün gösterimi için en az bellek gerektirenden en çok bellek gerektirene doğru çizilmiştir. Önceki örneğimizi bu dört yöntem ile örnekleyelim:

DSAY

7

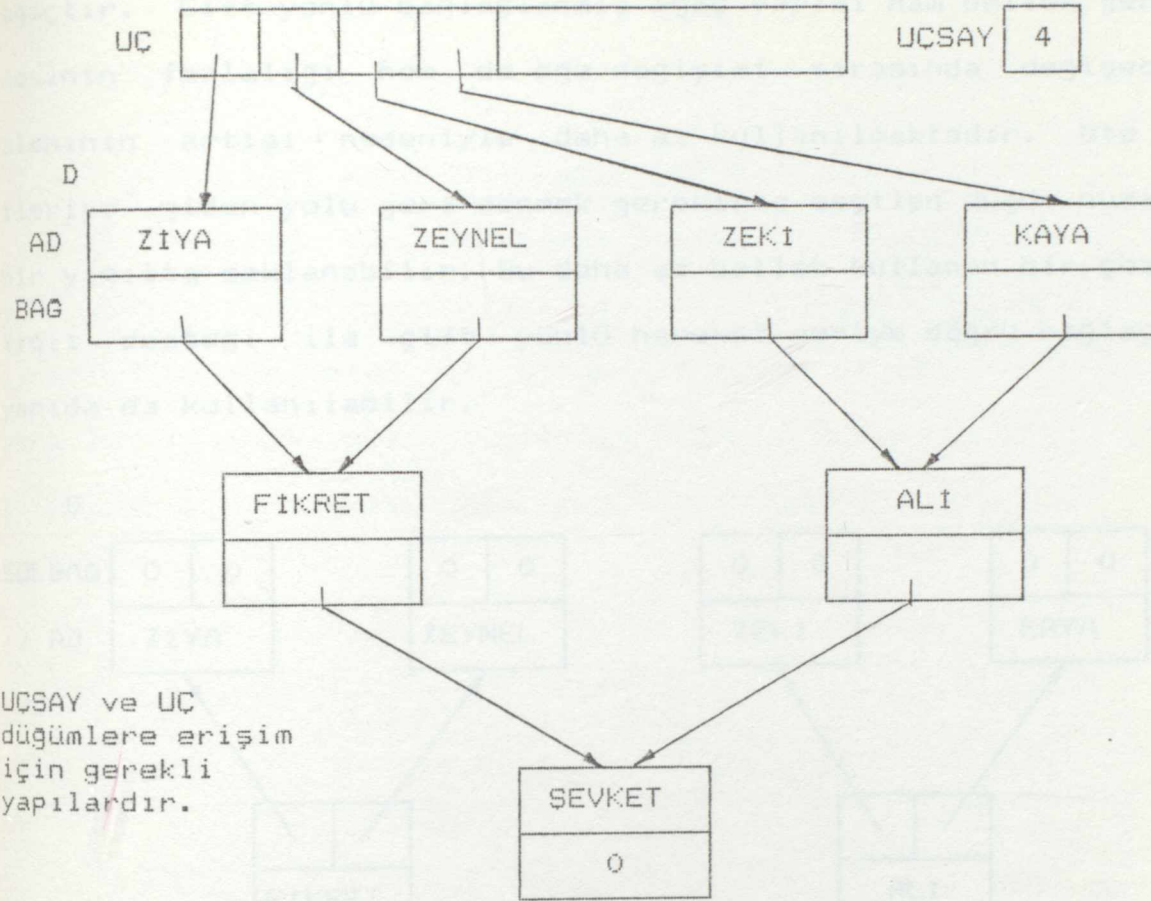
DAD

Not: DSAY ağaçtaki en büyük numarayı alan düğümün numarasıdır.

Not: Kök düğümü 1 numarayı alır. Herhangi bir i düğümünün sol altağacının kök düğümü $2i$, sağ altağacının kök düğümü $2i+1$ konumuna yerleşir.

1	SEVKET
2	ALİ
3	FIKRET
4	ZİYA
5	ZEYNEL
6	ZEKİ
7	KAYA

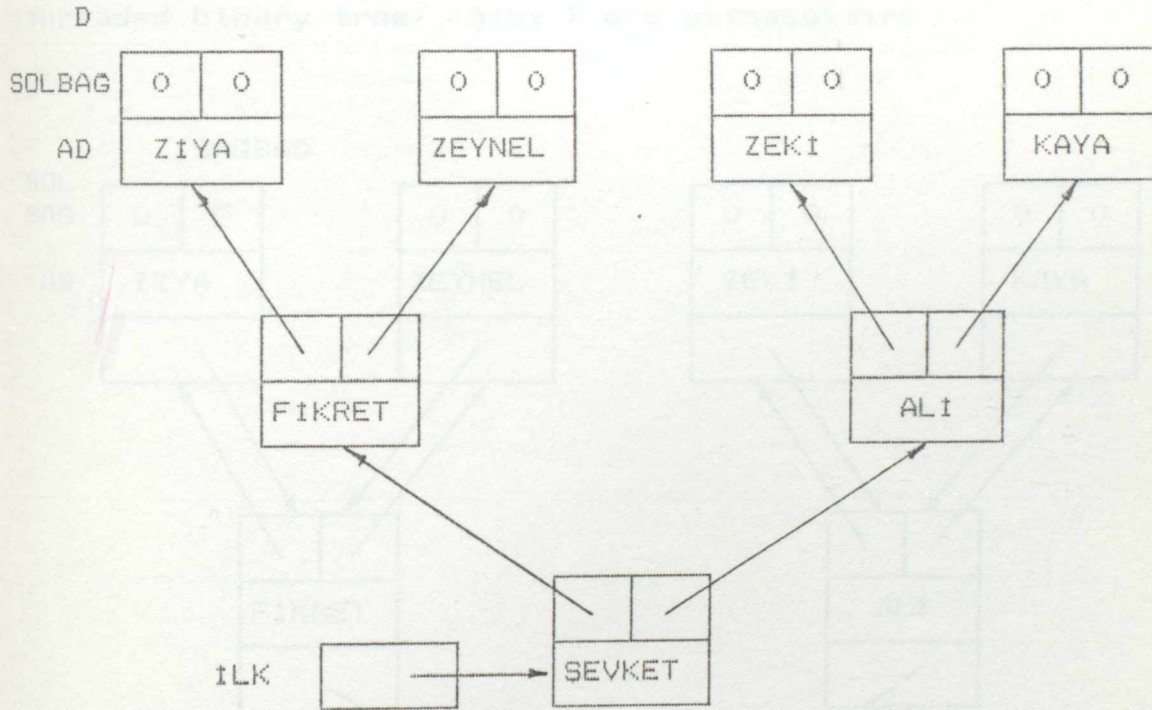
Şekil 43 İkili Ağacın Bağlaçsız Gösterimi



Sekil 44 Geriye Dogru Baglaçlanmış İkili Ağaç Gösterimi

Yukarıda örneği verilen yapılardan bağıksız gösterimde düğüm no 1 ile en büyük düğüm sayısı arasındaki oluşmamış için de yer ayrılır. Bu nedenle bağıksız yapı fazla öge değişimi göstermeyen ve tam dolu ikili ağaç (full binary tree) görünümüne oldukça yaklaşan ağaç yapıları için uygundur. Tam dolu ikili ağaç bir k derinliğine kadar tüm sol ve sağ oğulları dolu olan ikili ağaçtır. Bu ağaçta kök düğümü 1 olarak numaralanır. Herhangi bir i düğümünün sol oğlu $2i$ sağ oğlu $2i+1$ numarasını alır. Tam ikili ağaç düğümlerinde 1 ile en çok düğüm no arasında boşluk bulunmayan

agaçtır. Çift yönlü bağlanmış ağaç yapısı hem bellek gereksiniminin fazlalığı hem de öge değişimi sırasında değişecek bağ alanının artışı nedeniyle daha az kullanılmaktadır. Öte yandan ileriye giden yolu geri dönmek gerekirse seçilen düğüm numaraları bir yığıtta saklanabilir. Bu daha az bellek kullanan bir çözümdür. Yığıt desteği ile çift yönlü hareket geriye doğru bağlanmış yapıda da kullanılabilir.



Sekil 45 İleriye Doğru Bağlanmış İkili Ağaç Gösterimi

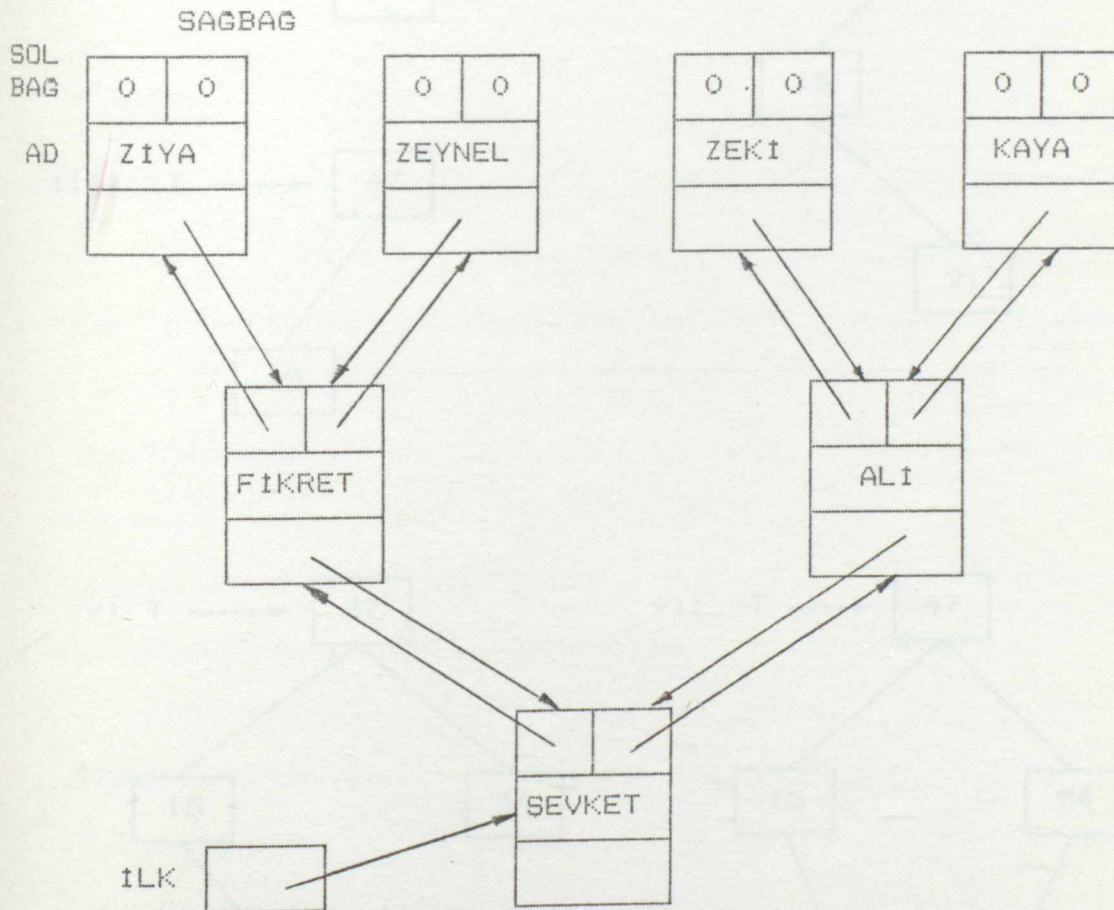
3.9.3 İkili Ağaç Veri Yapısında Erişim:

İkili ağaç veri yapısında bilgi içeriği ve ağaç veri yapısının erişim bağlaçları dağıtımını bir yapı gösterirler. Herhangi bir bilgi içeriği bilinen bir öge de kök düğümünden girilerek bağlaç zincirlerini izleyerek alınır. Yada bir ağacın tüm düğümlerini

ele alan dizi içinde (örneğin kök önce ikili ağaç işleme) ele alınır.

3.9.4 İkili Ağaç veri Yapısının Kullanımı

İkili ağaç veri yapısının önemli kullanımlarından biri sayı yada alfabetik bilgi sıralamasıdır. İkili ağaç kullanarak sıralama yaparken şu ana değin ele alınmamış bazı kavramlar (Yığın (heap), dengelenmiş ikili ağaç (balanced tree), kılavuzlanmış ikili ağaç (threaded binary tree) gibi) ele alınacaktır.



Şekil 46 Çift Yönlü Bağlanmış İkili Ağaç Gösterimi

Tanım: Bir ikili ağaçta her kök düğümünün bilgi içeriği alfabetik (yada sayısal) yönden sol alt ağacındaki tüm değerlerden büyük, sağ alt ağacındaki tüm değerlerden küçük yada eşitse bu ikili ağaca sıralama ağacı denir.

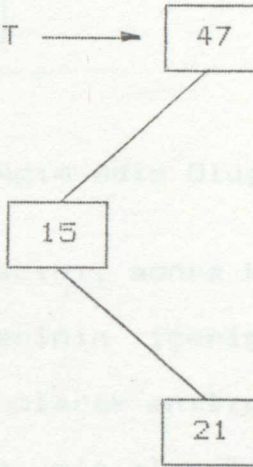
Örnek : 47, 15, 21, 54, 51, 54 olsun.

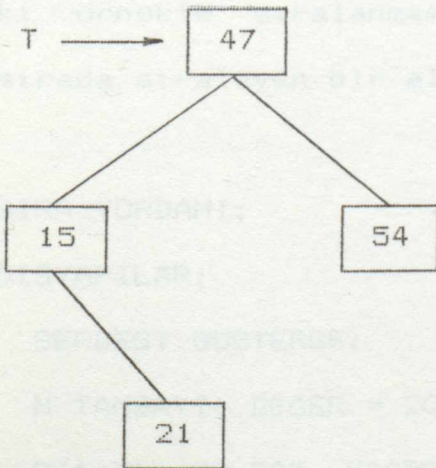
Sıralama ağacının aşamaları.

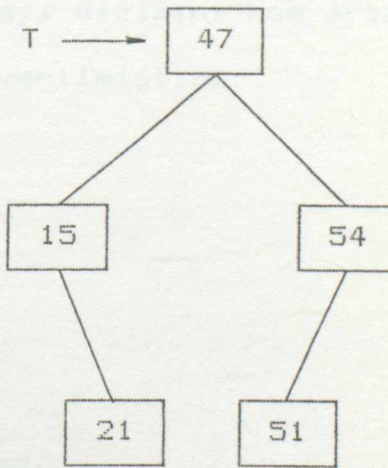
i) $T \rightarrow 0$

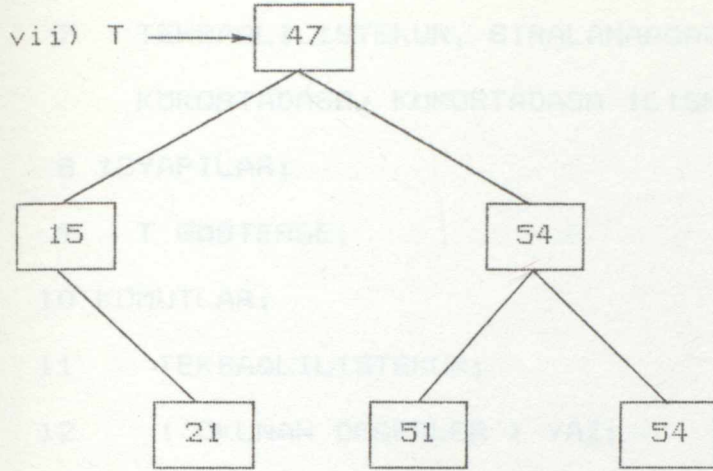
ii) $T \rightarrow 47$

iii) $T \rightarrow 47$


iv) $T \rightarrow 47$


v) $T \rightarrow 47$


vi) $T \rightarrow 47$




Sekil 47 Sıralama Ağacının Adım Adım Oluşumu

Tanım: Bir ikili ağacın önce bir alt ağacını, sonra kök düğümünün, daha sonra öteki alt ağacının düğümlerinin içerigini alma ve işleme 'kök ortada ikili ağaç dolaşma' olarak anılır. Kök ortada dolaşma yönteminde hangi alt ağacın önce ele alınacağı önemlidir. Bu nedenle 'sağ alt ağaç öncelikli kök ortada ikili dolaşımı' ile 'sol alt ağaç öncelikli kök ortada ikili dolaşım' farklı etkiler oluştururlar.

Aşağıdaki örnekte sıralanmamış bir sayı dizisini hem artan hemde azalan sırada sıralayan bir algoritma verilmiştir.

- 1 SIRA YORDAMI;
- 2 DISYAPILAR;
- 3 SERBEST GÜSTERGE;
- 4 N TAMSAYI, DEĞER = 200;
- 5 D(1:N) (SOLBAG, SAGBAG GÜSTERGE;

```

6   T := D; BILGI TUR (TAMSAYI, KESIRLI, KARAKTER))YAPI;
7   TEKBAGLILISTEKUR, SIRALAMAAGACIKUR, DGEAL, DGEVER,
8   KOKORTADASO, KOKORTADASA ILISKILI YORDAMLARDIR;
9   ICYAPILAR;
10  T GOSTERGE;
11  KOMUTLAR;
12  TEKBAGLILISTEKUR;
13  ('OKUNAN DEGERLER') YAZ;
14  K := 0, /* siralama agaci once bos */
15  EGER SIRALAMA AGACIKUR(T)
16      ISE [ ('KUCUKTEN BUYUGE SIRA') YAZ;
17            KOKORTADASO(T);
18            ANA1;
19            ('BUYUKTEN KUCUGE SIRA') YAZ;
20            KOKORTADASA(T);
21            ANA2: ]
22  DEGILSE [ ('ONGORULEN';N; 'YER YETERLI DEGIL') YAZ;
23            ('PROGRAMDA DIZI BOYUTUNU ARTTIRIN') YAZ ]
24  SIRA BITTI;

```

```

1  SIRALAMAAGACIKUR(T) MANTIKSAL YORDAMI;
2  ICYAPILAR;
3  P,Q,X,T GOSTERGE;
4  SAYI TUR(D(*).BILGI; /* Asil veri yapısının türünde */
5  YON KARAKTER(3);
6  KOMUTLAR;

```



```

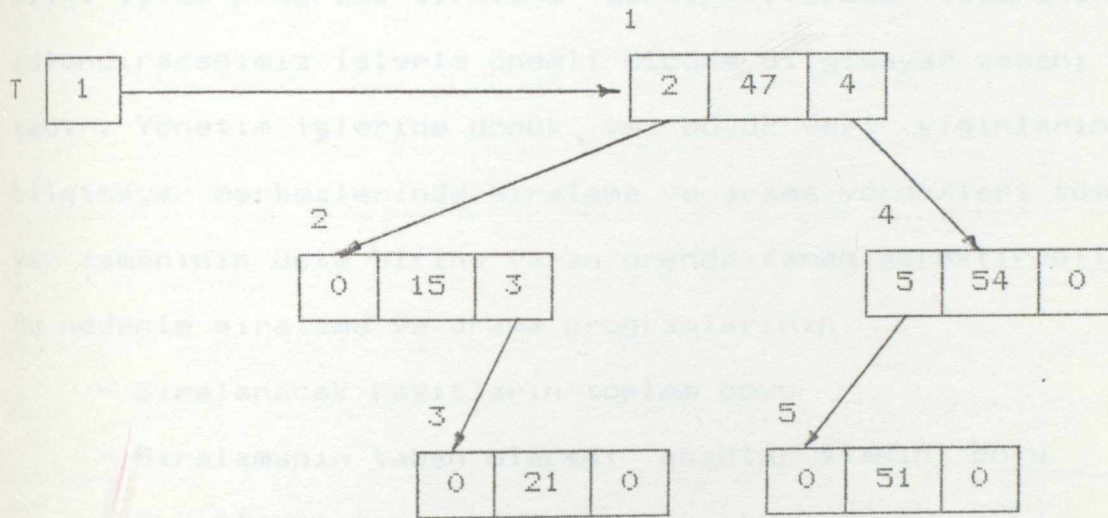
7   T := 0; /* BOS GÖSTERGE DEĞERİ SIFIR ATANDI      */
8   ÖNCE VERİBİTTİ(GİRİŞ) DEĞİL SİNA SONRA YİNELE;
9   [ SAYI'YI OKU;
10  EĞER DĞEBUL(X)
11  İSE [ D(X).BİLGİ := SAYI; D(X).SOLBAĞ := 0;
12      D(X).SAGBAĞ := 0;
13      EĞER T = 0 İSE [ T := X ]
14      DEĞİLSE [ /* UYGUN YER ARAKEN P GÖSTERGESİ      */
                  /* İLERİDEDİR. Q GÖSTERGESİ BUNU      */
                  /* BİR ADIM ARKADAN İZLER.             */
15                  P := T;
16                  ÖNCE P ≠ 0 SİNA SONRA YİNELE;
17                  [ EĞER D(X).BİLGİ < D(P).BİLGİ
18                      İSE [ Q := P ; YÖN := 'SOL';
19                          P := D(P).SOLBAĞ ]
20                      DEĞİLSE [ Q := P; YÖN := 'SAG';
21                          P := D(P).SAGBAĞ ]
22                  ] /* BU NOKTADA YENİ DĞENİN EKLENECEĞİ */
                  /* DÜĞÜM Q' DA BELİRLİ                */
23                  EĞER YÖN = 'SOL';
24                      İSE D(Q).SOLBAĞ := X;
25                      DEĞİLSE D(Q).SAGBAĞ := X;
26                  ]
27      DEĞİLSE [ SIRALAMAAGACIKUR := YANLIŞ; DÖN ]
28  ] /* BU NOKTA TÜM SAYILARIN SIRALAMA AGACINA      */
      /* GİRDİĞİ YERDİR                                */

```

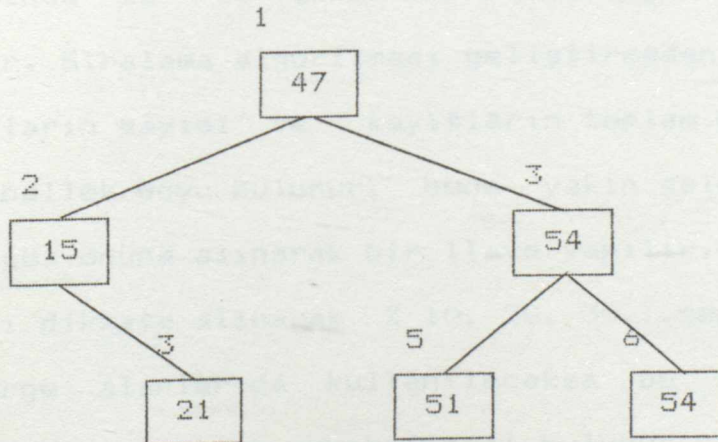
29 SIRALAMAAGACIKUR := DOGRU;

30 SIRALAMAAGACIKUR BITTI;

Yukarıda verilen algoritmanın işleyişini sıralama ağacına eklenen altıncı değer için izleyelim:



X	P	Q	YÖN
6	?	?	?
6	1	?	?
6	1	1	'SAG' ilk düğüm sağa dönülerek geçiliyor
6	4	1	'SAG'
6	4	4	'SAG' Ulaşılan düğüm sağa dönerek geçilir
6	0	4	'SAG' Burada D(Q).SAGBAĞ := X yapılıyor.



Şekil 48 Sıralama Ağacı Oluşumunda Bir Adım

4 BELLEK İÇİ SIRALAMA, BELLEK İÇİ ARAMA VE ANAHTARLAMA YÖNTEMLERİ

4.1 BELLEK İÇİ SIRALAMA YÖNTEMLERİ

Bilgisayar merkezlerinde yönetim işlerine destek veren birçok bilgi işlem programı sıralama (sorting), arama (searching) olarak adlandıracağımız işlerle önemli ölçüde bilgisayar zamanı harcamaktadır. Yönetim işlerine dönük ve büyük veri yığınlarını işleyen bilgisayar merkezlerinde sıralama ve arama yordamları tüm bilgisayar zamanının üçte birine varan oranda zaman gerektirebilir.

Bu nedenle sıralama ve arama programlarının

- Sıralanacak kayıtların toplam boyu
- Sıralamanın taban olacağı 'anahtar alanın' boyu
- Kayıtların toplam sayısı
- Sırası bozuk olan kayıtların toplam kayıt sayısına oranı
- Sırası bozuk kayıtların gerçek yerlerine uzaklıklarının dağılımı

incelenerek en uygun yöntemin bulunması istenir. Günümüzde bilgisayar yazılımında 25 - 30 farklı tür bellek içi sıralama ve arama yöntemi vardır. Sıralama algoritması geliştirmeden önce 'sıralanacak kayıtların sayısı' ve 'kayıtların toplam boyu' çarpılarak tüm tablonun bellek boyu bulunur, buna yakın gelecekte verilerin artabileceği göz önüne alınarak bir ilave yapılır. Bu ek o sorunun özel noktaları dikkate alınarak % 10, 20, 30,...seçilir. Sıralama için ek gösterge alanları da kullanılacaksa bu alanların toplam boyu da eklenerek iç bellek gereksinmesi bulunur. Eğer iç bellek gereksinmesi bu sıralama algoritması için planlanan bellek

boyu içinde ise bellek içi sıralama yöntemi seçiminin incelenmesine yönelinir. İç bellek alanı yeterli değilse 'bellek dışı sıralama ve birleştirme' yöntemi seçimi incelenir.

Bellek içi sıralama yönteminde sıralanacak kayıtlar temel olarak şu üç veri yapısı düzeninde bulunurlar:

- i) Tablo veya dizi
- ii) Tek bağlaçlı liste veya gösterge dizisi eklenmiş dizi
- iii) Bağlaçlı veya bağlaçsız ikili ağaç yapısı

Tablo veya dizi biçiminde düzenlenmiş verile sıralama için mutlaka kayıtların yer değiştirmesini yaparlar. Eğer kayıt boyu uzun ise kayıtları yerinden kaydırmak yerine bir gösterge dizisinde uygun sırayı gösteren göstergeler oluşturma önerilir. Bu durumda sıralamada tek bağlı liste yapısı kullanılmış olur. Bu durumda ek bellek alanı dikkate alınmalıdır. Gösterge ile erişim önceki tablo yöntemine göre ek bir yük getirmez, çünkü tablo oluştuğundaysa ek bir indeks hesaplayarak erişim söz konusudur. Ayrıca sıralama işlemlerinde bağlaçlı ve bağlaçsız biçiminde ikili ağaç kullanılabilir. İkili ağacın bağlaçsız gösterimi içim 'Tam dolu ikili ağaç' oluşturulur. Bu durumda baba ve oğul düğümlerine dizi indisi hesaplanarak erişilebilir. Şimdi bazı sıralama yöntemlerini inceleyelim:

4.1.1 Çabuk - Sıralama Algoritması

Çabuk - Sıralama algoritması (Quick - sort) sıralanacak kümeyi her aşamada alt kümelere bölmeye giderek daralan altkümeler ile sıralama yapmaya dayanır.

şimdi bu yöntemin birinci aşamasını aşağıdaki örnekte inceleyelim:

DİZİ

41
31
57
25
52
46
67
38
70
63
19
49

```

1 İKİKÜMEYEBÖL(DİZİ,I,J) YORDAMI;
2 DİŞYAPILAR;
3   DİZİ(*);
4 KOMUTLAR;
5   EĞER I = J
6     İSE DÖN;
7   EĞER I > J
8     İSE [ ('I.İNDİS J DEN BÜYÜK') YAZ;
9           ('YANLIŞ KULLANIM') YAZ ; DÖN ]
10  YÖN := -1  /* J geriye doğru */
11  ÖNCE I = J SİNA SONRA YİNELE;
12  [ EĞER DİZİ(I) > DİZİ(J)
13    İSE [ SAKLA := DİZİ(I);
14          DİZİ(I) := DİZİ(J);
15          DİZİ(J) := SAKLA
16          YÖN := - YÖN ];
17  EĞER YÖN := -1;
18    İSE J := J - 1;
19    DEĞİLSE I := I + 1;
20  ] /* Bu noktada asıl döngü kapanıyor.*/
21 İKİKÜMEYEBÖL BİTTİ;

```

Şimdi ilk aşamanın işleyişini adım adım izleyelim:

I				I		J	ETKİ	YÖN
1	1	41	19	1	12			-1
	2	31		1	11	$DIZI(1) \leftrightarrow DIZI(11)$		+1
	3	51	41 38	2	11			+1
	4	25		3	11	$DIZI(3) \leftrightarrow DIZI(11)$		-1
	5	52	41	3	10			-1
	6	46		3	9			-1
	7	67		3	8	$DIZI(3) \leftrightarrow DIZI(8)$		+1
	8	38	41 52	4	8			+1
	9	70		5	8	$DIZI(5) \leftrightarrow DIZI(8)$		-1
	10	63		5	7			-1
J	11	11	41 57	5	6			-1
	12	49		5	5			

dön

Görüldüğü gibi İKİKÜMEYEBÖL algoritması sonunda $DIZI(I)$ içinde sıralanmamış bir dizi (küme $S1$)'den sonra gelmekte ve sıralanmamış bir dizi ile (küme $S2$) izlemektedir. Dikkate değer bir nokta küme $S1$ içindeki tüm öğelerin $DIZI(I)$ 'den küçük veya eşit, küme $S2$ deki öğelerin de $DIZI(I)$ 'den büyük veya eşit oluşudur. Bu durumda sıralamanın geri kalan kesiminde $DIZI(I)$ ögesi yer değiştirmeyecektir. Böyleyse problem elde ettiğimiz iki alt kümenin sıralanmasına indirgenmiştir.

Şimdi alt kümenin ALT ve ÜST indislerini saklayacak bir yığıt kurarak çabuk - sıralama algoritmasını yazalım:


```

1 CABUKSIRALA (N,DIZI) YORDAMI
2   DIZI(0:N+1) TUR(TAMSAYI, KESIRLI, KARAKTER);
3 DISYAPILAR;
4   YIGIT(100) (ALT, UST TAMSAYI)YAPI;
5 ICYAPILAR;
6 KOMUTLAR;
7   S := 1; YIGIT(S).ALT := 1; YIGIT(S).UST := N;
8   ONCE S = 0 SINASONRA YINELE;
9     [ II := YIGIT(S).ALT;
10      JJ := YIGIT(S).UST;
11      S := S - 1; I := II; J := JJ;
12      IKIKUMEYEBOL(DIZI,I,J) CAGIR;
13      EGER II < I - 1
14        ISE [ S:= S + 1;
15              YIGIT(S).ALT := II;
16              YIGIT(S).UST := I-1 ]
17      EGER JJ > I + 1
18        ISE [ S := S + 1;
19              YIGIT(S).ALT := I + 1;
20              YIGIT(S).UST := JJ ]
21      ]
22 CABUKSIRALA BITTI;

```

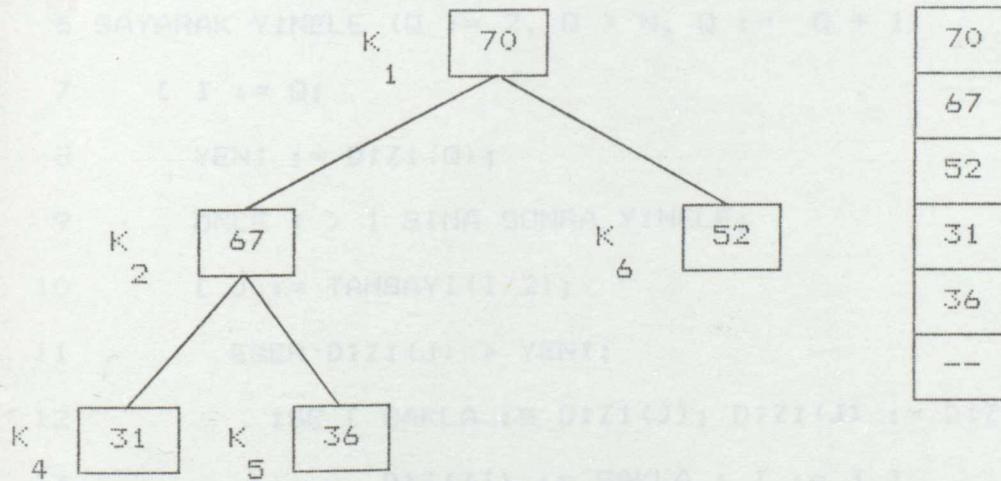
Cabuk sıralama algoritması kabarcık sıralama algoritmasının etkin olmadığı daha büyük tablolar üzerinde kullanılır. Bu algoritmanın algoritma karmaşıklığı $O(n)$ 'dir, ancak algoritma etkinliği $n \log n$

olarak hesaplanmaktadır. Bu algoritma için en olumsuz giriş verisi düzeni sıralanmış dizidir. Bu durumda S1 kümesi boş küme, S2 kümesi $n - 1$ ögeli bir küme olacağından algoritma karmaşıklığı $O(n^2)$ olarak belirir.

4.1.2 İkili Ağaç Yapısı ile sıralama Yöntemleri

Ağaç yapısı kullanılarak sıralama yöntemlerinden biri daha önce işlenen sıralama ağacı oluşturmak ve bunu kök ortada yöntemi ile dolaşmaktır. Kök ortada yöntemi ile dolaşma sırasında kullanılan yığıt'ı ortadan kaldırmak için kök ortada kılavuzlanmış ikili ağaç yapısında kullanılabilir.

Önce ikili ağacın bağlaçsız gösterimini inceleyelim:



Şekil 49 Bağlaçsız İkili Ağaç Gösterimi

K_i ögesi dizinin i 'inci ögesine, K_i 'nin sol ve sağ oğulları ise $2i$ ve $2i + 1$ yerlerine yerleşir. Bir ögenin yerleştiği dizi yerinin indisi J ise bu ögenin babası durumunda olan öge $[J/2]$ konumunda olacaktır.

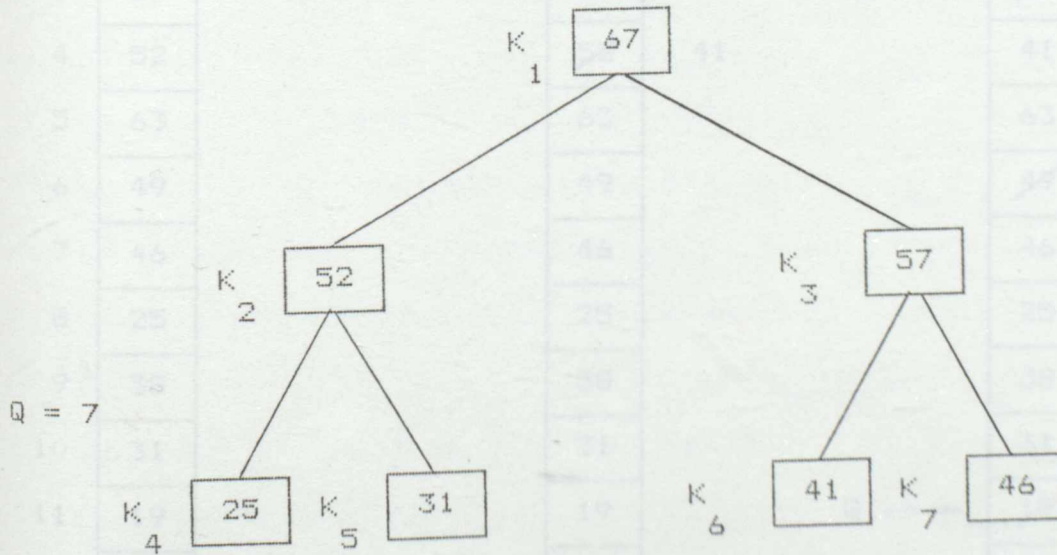
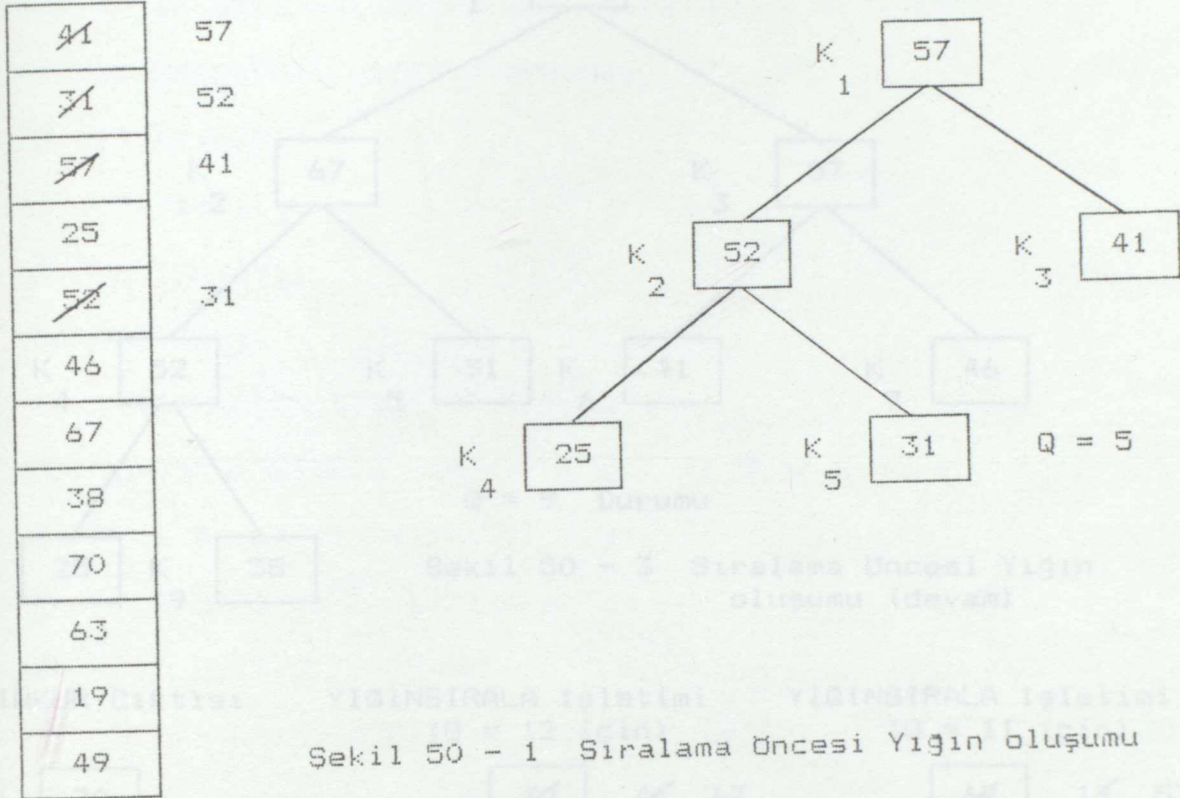
4.1.2.1 Yığın Sıralama (Heap - Sort) Yöntemi

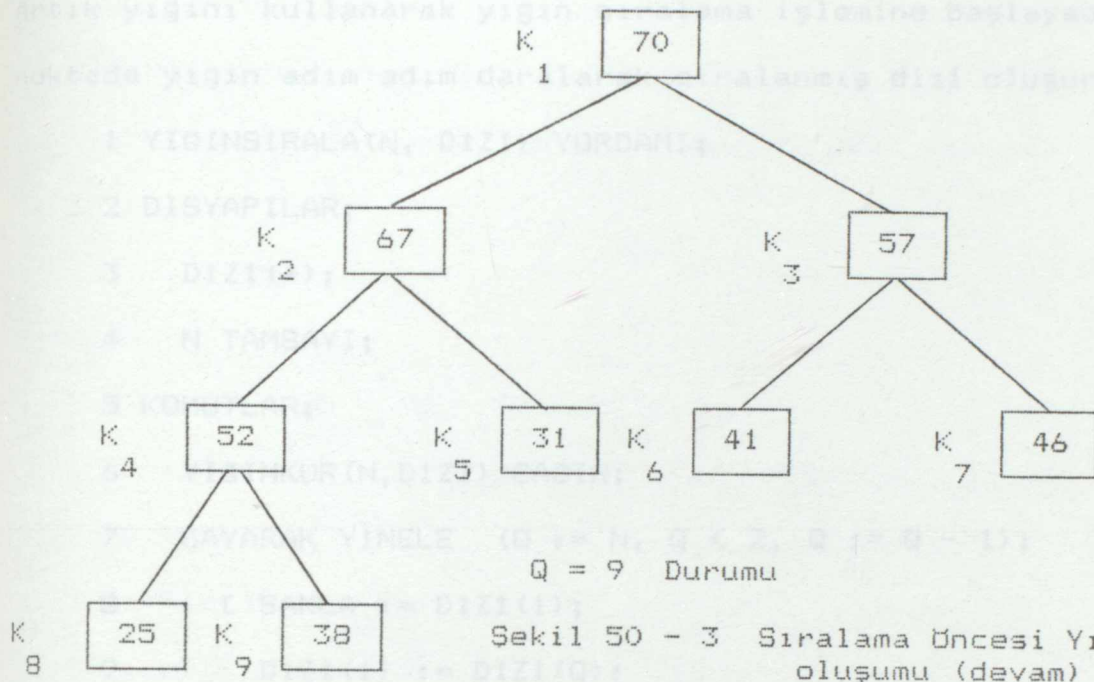
Yığın sıralama da iki aşamalı bir algoritmadır. Birinci aşamada bir yığın (Heap) kurulur. Küçükten büyüğe doğru yapılacak bir yığın sıralama için kurulacak yığın kök düğümünde en büyük değeri bulundurur, ve uç dallardan köke doğru gidilirken değerler devamlı büyür. Önce aşağıdaki algoritmayı inceleyelim. Bu aşamada yığın adım adım genişleyerek tüm diziyi kapsar.

```

1 YIGINKUR( N, Dizi ) YORDAMI;
2 DISYAPILAR;
3   Dizi(*);
4   N   TAMSAYI;
5 KOMUTLAR;
6 SAYARAK YINELE (Q := 2, Q > N, Q := Q + 1)
7   [ I := Q;
8     YENI := Dizi(Q);
9     ÖNCE I > 1 SİNA SONRA YINELE;
10    [ J := TAMSAYI(I/2);
11      EĞER Dizi(J) > YENI;
12        İSE [ SAKLA := Dizi(J); Dizi(J) := Dizi(I);
13             Dizi(I) := SAKLA ; I := J ]
14        DEĞİLSE [ A1' E GİT ]
15    ]   A1 : /* Q ADET ELEMAN İÇİN YIGIN OLUSTU      */
16  ]
17 YIGINKUR BİTTİ;

```





YIGINKUR Çıktısı

YIGINSIRALA İşletimi
(Q = 12 için)YIGINSIRALA İşletimi
(Q = 11 için)

1	70	70	67	57
2	67	67	52	52
3	57	57	49	49
4	52	52	41	41
5	63	63	63	63
6	49	49	49	19
7	46	46	46	46
8	25	25	25	25
9	38	38	38	38
10	31	31	31	31
11	19	19	19	67
12	41	41	70	70

Q →

Sekil 50 - 4 Yiginsirala Algoritması İkinci Aşama

Artık yığını kullanarak yığın sıralama işlemine başlayabiliriz. Bu noktada yığın adım adım daralarak sıralanmış dizi oluşur.

```

1 YIGINSIRALA(N, DIZI) YORDAMI;
2 DISYAPILAR;
3   DIZI(*);
4   N TAMSAYI;
5   KOMUTLAR;
6   YIGINKUR(N,DIZI) CAGIR;
7   SAYARAK YINELE (Q := N, Q < 2, Q := Q - 1);
8   [ SAKLA := DIZI(1);
9     DIZI(1) := DIZI(Q);
10    DIZI(Q) := SAKLA;
11    /* Kök ile DIZI(Q)'yu yer değiştir. */
12    I := 1; J := 2;
13    ÖNCE J < Q - 1 SINA SONRA YINELE;
14    [ EGER DIZI(J + 1) > DIZI(J);
15      İSE J := J + 1;
16      EGER DIZI(J) > DIZI(I);
17      İSE [ SAKLA := DIZI(I);
18            DIZI(I) := DIZI(J);
19            DIZI(J) := SAKLA;
20            I := J ; J := 2 * I ]
21    DEĞİLSE [ A2'YE GİT ]
22  ]
23 YIGINSIRALA BİTTİ;

```


D1	D2	172	D4	D5
----	----	-----	----	----

4.1.3 Birleştirme Sıralaması

Birleştirme sıralaması (Merge sorting) büro işlerinde kayıt kartları üzerinde zaman zaman insan eliyle yapılan bir sıralamadır.

Örneğin

* D1 : 19 25 38 49 57
 * D2 : 31 41 46 52 63 67 70
 D0 : boş

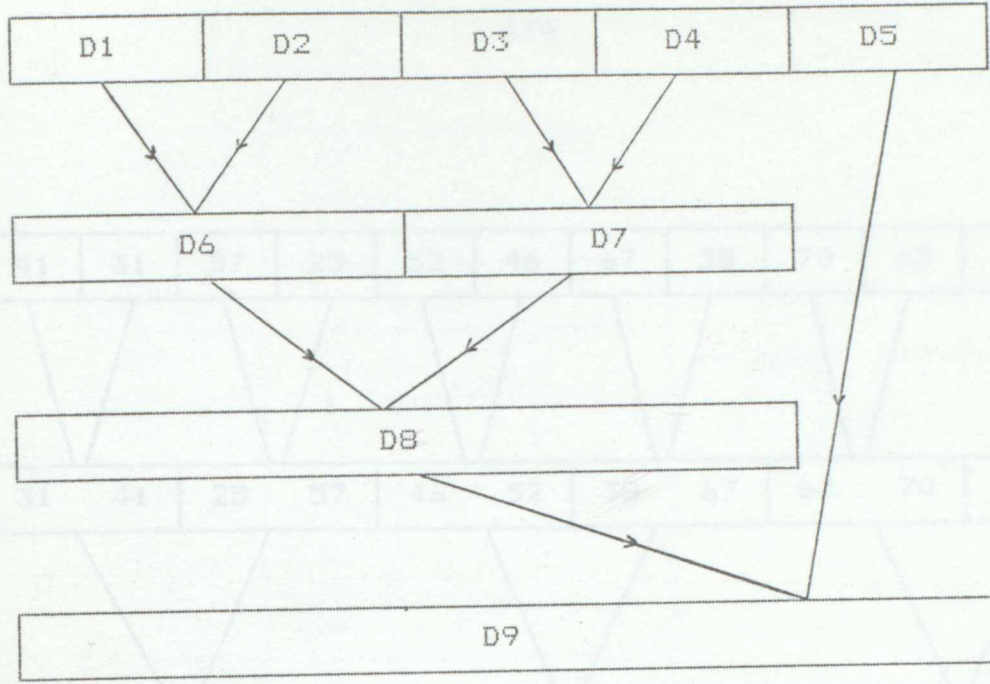
Üç iterasyon sonra

* D1 : 38 49 57
 * D2 : 41 46 52 63 67 70
 D0 : 19 25 31

Dokuz adım sonra

* D1 : boş
 * D2 : 63 67 70
 D0 : 19 25 31 38 41 46 49 52 57

Yukarıda örneği verilen algoritma 'iki yönlü birleştirme (two way merge sorting) sıralaması' olarak bilinir. Birleştirme sıralaması yapılacak dizilerin sayısı K ise ($K \geq 2$) diziler ikişer ikişer ele alınıp iki yönlü birleştirme sıralamasını eldeki dizilere ve ara dizilere ağaç yapısını uygulayarak sıralanmış diziyi oluşturmak gerekir. Aşağıda buna örnek verilmiştir:

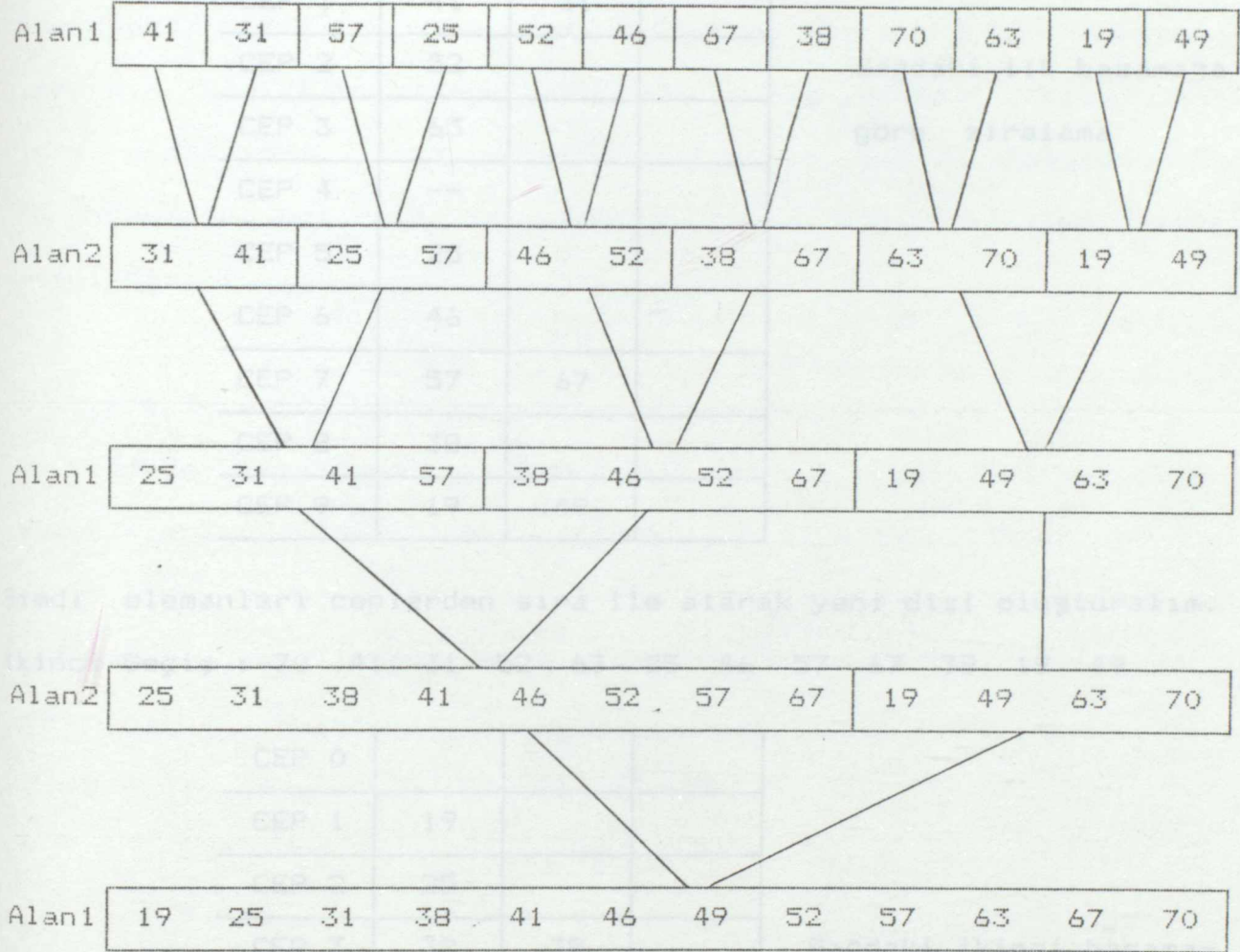


Şekil 51 İki Yönlü Birleştirme Sıralaması Ağaç Yapısı

Bu algoritma en çok $\log_k n$ derinliğinde birleştirme ağacı oluşturur. Bu durum başlangıçta her altdizinin 1 öğeden oluştuğu kabul edildiğinde söz konusudur. Eğer başlangıçta içinde toplam n adet öğe olan r adet dizi varsa ($r \leq n$) bu durumda algoritma karmaşıklığı $O(n \log_k r)$ olarak belirlenir.

Bu algoritma sıralanmamış verilerin yerleştiği n ögeli alana eşit boyda ikinci bir bellek alanı gerektirir. Bu nedenle bellek alanı artışı dikkate alınmalıdır. Algoritma karmaşıklığı $n \log_2 n$ düzeyindedir. Kaydırma işlemlerinin yoğunluğu nedeniyle kayıt alan boyunun kısa olduğu bellek içi sıralama problemlerinde yada ana belleğe sığmayacak boyutta dış sıralama yöntemlerinde kullanılır.

Birleştirme sıralamasının işleyişini aşağıdaki örnekte izleyelim:



Sekil 52 Birleştirme Sıralamasını ilerleyişi

4.1.4 Basamak Sıralaması

Basamak sıralamasım (radix sort) bilgisayarın üretilmesinden önce kullanılan mekanik sıralayıcıların çalışma yapısına dayalı olarak yapılır.

Aşağıdaki örneğimizin giriş verileri şunlardır:

Giriş : 41 31 57 25 52 46 67 38 70 63 19 49

CEP 0	70		
CEP 1	41	31	
CEP 2	52		
CEP 3	63		
CEP 4	--		
CEP 5	25		
CEP 6	46		
CEP 7	57	67	
CEP 8	38		
CEP 9	19	49	

Sağdaki ilk basamaga
göre sıralama

Şimdi elemanları ceplerden sıra ile alarak yeni dizi oluşturalım.

İkinci Geçiş : 70 41 31 52 63 25 46 57 67 38 19 49

CEP 0			
CEP 1	19		
CEP 2	25		
CEP 3	32	38	
CEP 4	41	46	49
CEP 5	52	57	
CEP 6	63	67	
CEP 7	70		
CEP 8			
CEP 9			

Sağdaki ikinci basama-
ga göre sıralama.

İkinci Geçiş Sonu: 19 25 31 38 41 46 49 52 57 63 67 70

iki basamaklı sayılar ikinci geçiş sonunda sıralanmış olacaktır. Basamak sıralaması yöntemi sıralama anahtarının kısa olduğu durumlarda hızlı bir sıralamadır. Eger sıralama anahtarı M basamak ise $M \cdot n$ kez işlem yapmak gereklidir. Bu yöntemin gerçekleşmesi için olası veri yapısı tek bağlı liste yapısı olabileceğine, Bu durumda belleğin iki katına çıkmayacağı noktasına dikkat etmelidir.

4.1.5 Adres Hesaplama Sıralaması

Adres hesaplama sıralaması (address - calculation sort) anahtarlama yöntemi (hash coding)' e dayalıdır. Eger n adet kayıt bir anahtarlama fonksiyonu ile m kümeye oldukça dengeli biçimde bölünebiliyorsa ve $m = \text{karekök}(n)$ dolaylarında ise adres hesaplaması uygulanabilir.

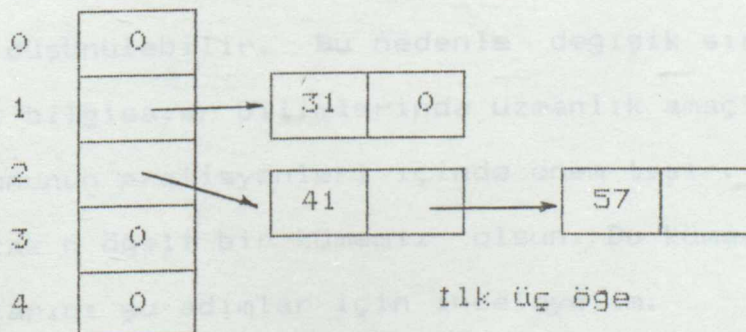
Giriş : 41 31 57 25 52 46 67 38 70 63 19 49

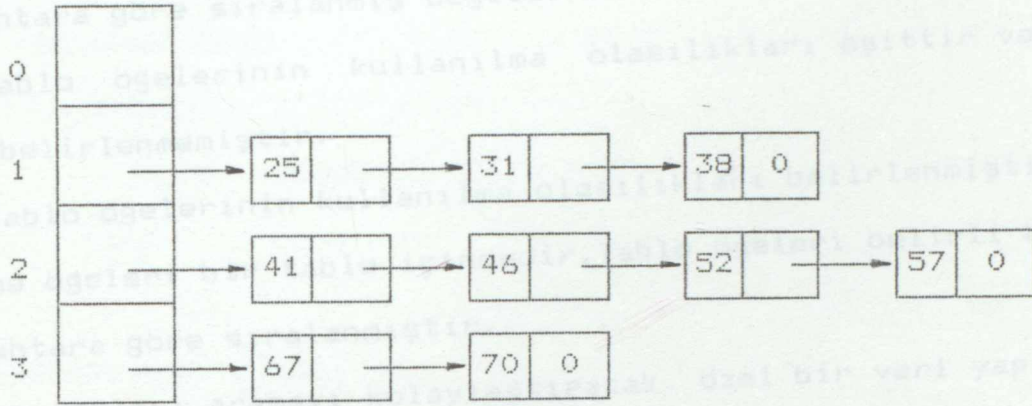
$$n = 12$$

$$m = 4$$

Sayıların iki basamaklı verildiği bu örnekte bilindiğinden önerilen H anahtarlama fonksiyonu aşağıdaki gibi öngörülmüştür.

$$H(X_i) = \text{tamsayı}(X_i/20)$$





Sekil 54 Adres hesaplama sıralamasında Tekbağlı Listeler

Bu algoritmanın algoritma karmaşıklığı $O(n)^2$ dir. Bu durum tüm öğeler kümelerden birine yayıldığı zaman oluşur. Knuth'a göre anahtarlama fonksiyonu oldukça iyi olduğunda algoritma etkinliği $O(n)$ ' e yaklaşmaktadır. Bir sıralama yönteminde $O(n)$ her bir elemanın bir kez ele alınma gereği nedeniyle olabilecek en az hesaplama sınırı (teorik minimum) dur.

4.2 BELLEK İÇİ ARAMA YÖNTEMLERİ

Bilgi işlem alanında sıralama ve arama önemli yer tutar. Tipik bir bilgi işlem merkezinde merkezi işlem gücünün dörtte biri ila üçte biri arasında önemli bir kısmı sıralama ve arama işlemlerine kullanıldığı düşünülebilir. Bu nedenle değişik sıralama ve arama algoritmaları bilgisayar bilimlerinde uzmanlık amaçlayan kişilerin yanında bu konunun pratisyenleri içinde önem taşır.

Arama yapılacak n ögeli bir kümemiz olsun. Bu kümede arama zamanı ve algoritmalarını şu adımlar için inceleyelim.

- i) Küme öğeleri tablo içindedir. Tablo içeriği belirli bir anahtara göre sıralanmış değildir.
- a) Tablo öğelerinin kullanılma olasılıkları eşittir veya belirlenmemiştir.
- b) Tablo öğelerinin kullanılma olasılıkları belirlenmiştir.
- ii) Küme öğeleri bir tablo içindedir. Tablo öğeleri belirli bir anahtara göre sıralanmıştır.
- iii) Küme öğeleri aramayı kolaylaştıracak özel bir veri yapısı içine yerleştirilmiştir. Bu özel veri yapısı ;
- a) Dengelenmiş ikili ağaç
- b) Anahtarlama Tablosu
- olarak düzenlenmiştir.

4.2.1 Doğrusal Arama

Doğrusal arama, genellikle dizi üzerinde (bazan tek bağlı liste üzerinde) arama yapılacak küme öğeleri bir tablo içinde bir anahtara göre sıralanmamış olduğu durumda uygulanan bir yöntemdir. Aşağıda algoritma etkinliği, algoritma karmaşıklığı ve program etkinliği yönünden karşılaştırmamız gereken iki algoritma vardır.

- 1 DOGRUSALARAI(Dizi,N,YENI) TAMSAYI YORDAMI;
- 2 DISYAPILAR;
- 3 Dizi(N) KESIRLI;
- 4 YENI (*) KESIRLI;
- 5 N TAMSAYI
- 6 KOMUTLAR;

```

7 DOGRUSALARA1 := 0
8 SAYARAK YINELE (I := 1, I > N, I := I + 1)
9 DIZI[ EGER DIZI(I) = YENI
10 DIZI ISE (DOGRUSALARA1 := I
11 DIZI YAPILARI DONI
12 N, J YARISAYI
13 DOGRUSALARA1 BITTI;
14 DIZI(I) := YENI

```

Bu algoritmada

- i) Algoritma karmaşıklığı : $O(n)$,
- ii) Algoritma etkinliği : $1/2 O(n)$,
(Ortalama olarak tablonun yarısında bulacaktır.)
- iii) Program etkinliği şöyle özetlelebilir.

Bu algoritmada

i) $\frac{1}{2} n$ kez $DIZI(I)$, $YENI$, 8 Erişim

ii) algoritma etkinliği I , N , I , I , 1 olarak çarpılmıştır.

Bu iki algoritmanın algoritmik etkinliği 2 toplama

iii) de aynı fonksiyonla modellenmiştir. Ancak program etkin-

liğinde bazı farklılıklar 2 Koşul sinama

iii) Program etkinliği $(DIZI(I) = YENI$ 2 koşul

ve $I > N$)

1 toplama

1 toplama

1 aktarma

1 aktar

Özet : 8 erişim + 1 Çarp + 3 Topla + 2 Koşul sına + 1 Aktar

1 aktar

1 aktar

Öte yandan ikinci algoritmamız şöyledir:

1 DOGRUSALARA2(DİZİ, N, YENİ) TAMSAYI YORDAMI;

2 DİŞYAPILAR;

3 DİZİ(0:N), YENİ KESİRLİ;

4 İÇYAPILAR;

5 N, İ TAMSAYI;

6 KOMUTLAR;

7 DİZİ(0) := YENİ;

8 İ := N

9 ÖNCE DİZİ(İ) $\neq$ YENİ SİNA SONRA YİNELE;

10 [İ := İ - 1]

11 DOGRUSALARA2 := İ

12 DOGRUSALARA2 BİTTİ;

Bu algoritmada

i) algoritma karmaşıklığı : $O(n)$

ii) algoritma etkinliği : $1/2 O(n)$ olarak belirlenmiştir.

Bu iki algoritmanın algoritma karmaşıklığı da ve algoritma etkinliği de aynı fonksiyonla modellenmiştir. Ancak program etkinliğinde bazı farklılıklar vardır.

iii) Program etkinliği şöyle özetlenebilir.

1	keş	DİZİ(İ), YENİ,	6 erişim
2	keş	İ, İ, 1 Erişimi	1 çarp
			2 topla
		1 koşul sinama	1 koşul
		1 çıkarma (topla)	1 topla
		1 aktar	1 aktar

Özet : 6 erişim + 1 çarp + 3 topla + 1 koşul + 1 aktar

Bu iki algoritmanın program düzeyinde etkinlikleri açısından yapılan çözümlemeye göre % 10 - 20 arasında işletim zamanı farkedebilir. Teknik yönden iyi yetişmiş bir yazılımcı bunları ihmal etmemelidir.

Doğrusal aramanın küçük boyutlu ($n < 20$) tekbaglı listelerde pratik ve kolay bir yöntem olduğu unutulmamalıdır.

4.2.2 Önyükleme ile Doğrusal Arama

İçinde arama yapılacak n ögenin kullanım sıklıkları birbirinden farklı olabilir ve bu olasılıklar belirlenmiş durumdadır. Örneğin programlama dillerinin anahtar sözcüklerinin kullanımı bu özelliği gösterir. Aşağıdaki tabloyu COBOL programlama dilinin anahtar sözcüklerinin 100 COBOL satırında geçiş sayısı tablosunun bir kesimi olduğunu var sayalım.

ADD	10	Bu durumda kümenin kullanım olasılığı
COMPUTE	30	yüksek öğelerin tablonun başlan-
DIVIDE	40	gıcına yerleştirmek doğrusal arama
ELSE	09	algoritmasının etkinliğini arttıra-
EXAMINE	00.5	caktır. Bu yönteme önyükleme adı
IF	15	verilir. Önyükleme ile doğrusal ara-
INPUT	23	ma küçük boyutlu tablolarda (20-25)
MULTIPLY	05	kullanılabilecek, algoritma etkinli-
OUTPUT	06	ği $1/2 O(n)$ olan bir yöntemdir.
SUBTRACT	03	
THEN	15	

4.2.3 Fiziksel Olarak Sıralanabilir Dizile Üzerinde İkili

21 İKİLİARAMA DİZİ: Arama Yöntemi

İkili arama yöntemi (binary search) uygulanabilmesi için arama yapılacak kümenin n ögesinin belirli bir anahtara göre sıralanmış olması gerekir. Bu anahtarın artan sırada düzenlenmiş olduğunu var sayarak bir ikili arama algoritmasını inceleyelim:

```

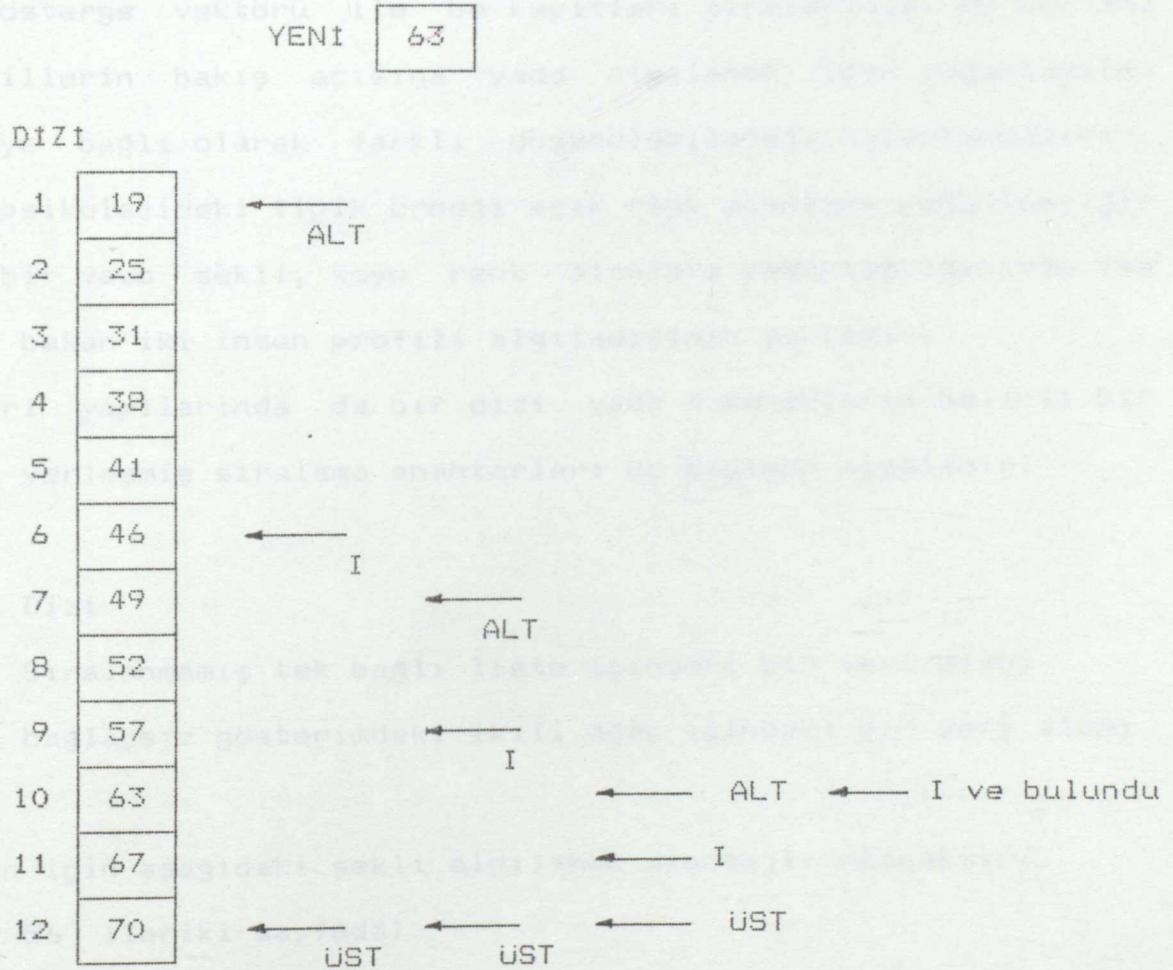
1 İKİLİARAMA(DİZİ, N, YENİ) TAMSAYI YORDAMI;
2 DISYAPILAR;
3   DİZİ(N), YENİ KESİRLİ;
4 İÇYAPILAR;
5   N, ALT, UST, I TAMSAYI;
6 KOMUTLAR;
7   İKİLİARAMA := 0;
8   I := 1; UST := N;
9   ÖNCE ALT ≠ UST SİNA SONRA YİNELE;
10  [ I := TAMSAYI((ALT + UST)/2);
11    EĞER YENİ < DİZİ(I)
12    İSE [ UST := I - 1 ]
13          DEĞİLSE [ EĞER YENİ > DİZİ(I)
14                İSE [ ALT := I + 1 ]
15                      DEĞİLSE [ İKİLİARAMA := I;
16                                /* Aranan öge bulundu */; DÖN ]
17          ]
18    ]
19  EĞER YENİ = DİZİ(I)

```

20 İSE [İKİLİARAMA := İ]

21 İKİLİARAMA BİTTİ;

Şimdi aşağıda oniki ögeli bir DİZİ üzerinde bir arama örneği verelim :



Şekil 55 İkili Arama Algoritması

Bu arama algoritmasında algoritma karmaşıklığı $O(\log_2 n)$ olarak belirlenir algoritma etkinliği de $O(\log_2 n)$ dir. Kümeye eleman ekleme ve çıkarma kaydırma işlemleri veya yeniden sıralama gerektirir.

4.2.4 Fiziksel Olarak Sıralanamayan Diziler Üzerinde İkili

Arama Yöntemi

Yerinden kaydıramayacağımız 400 adet kayıt (TUT) olduğunu varsayınız. TUT içindeki DÖRENCİNO'yu anahtar olarak kullanarak DİZİ gösterge vektörü ile bu kayıtları sıralayınız. Bu noktada bazı şekillerin bakış açısına yada algılamak için yoğunlaşılacak düşünceye bağlı olarak farklı düşünülebileceği hatırlanmalıdır. bunun psikolajideki tipik örneği açık renk alanlara yoğunlaşmışınızda bir vazod şekli, koyu renk alanlara yoğunlaşmışınızda ise yüzyüze bakan iki insan profili algıladığımız şekildedir.

Veri yapılarında da bir dizi yada tutanakların belirli bir alanına yerleşmiş sıralama anahtarları üç biçimde algılanır.

- i) Dizi
- ii) Sıralanmamış tek bağılı liste içindeki bir veri alanı
- iii) bağılsız gösterimdeki ikili ağaç içindeki bir veri alanı

Sekil 56 Sıralama Tablosu İndis Dizisi ile Sıralama

Bu sorun için aşağıdaki şekli algılamak avantajlı olacaktır.

(Sekil 56 ileriki sayfada)

Bu durumda önceki bölümde verilen algoritmada karşılaştırma

DİZİ(I) İLE YENİ

arasında yapılırken bu algoritmada karşılaştırma

TUT(DİZİ(I)).DÖRENCİNO İLE YENİ

Bilgi alanları arasında olacaktır. Şüphesiz ki bu aramayı yapabilmek için gerekli olan indis dizisi bir adım öncesinde, sıralama aşamasında kurulmuş olmalıdır.

Dizi		Anahtar TUT Alanı		Öteki bağlı alanlar	
1	3	1	V28114	13	
2	4	2	U30512	10	
3	5	3	P27304	8	
4	6	4	D11404	6	
5	7	5	R18483	6	
6	8	6	R21301	6	
7	9	7	S15713	6	
8	12	8	S26414	6	
9	11	9	T16813	6	
10	10	10	U12707	5	
11	1	11	U11430	4	
12	2	12	T21114	4	

Şekil 56 Sırasız Tabloda Indis Dizisi ile Sıralama

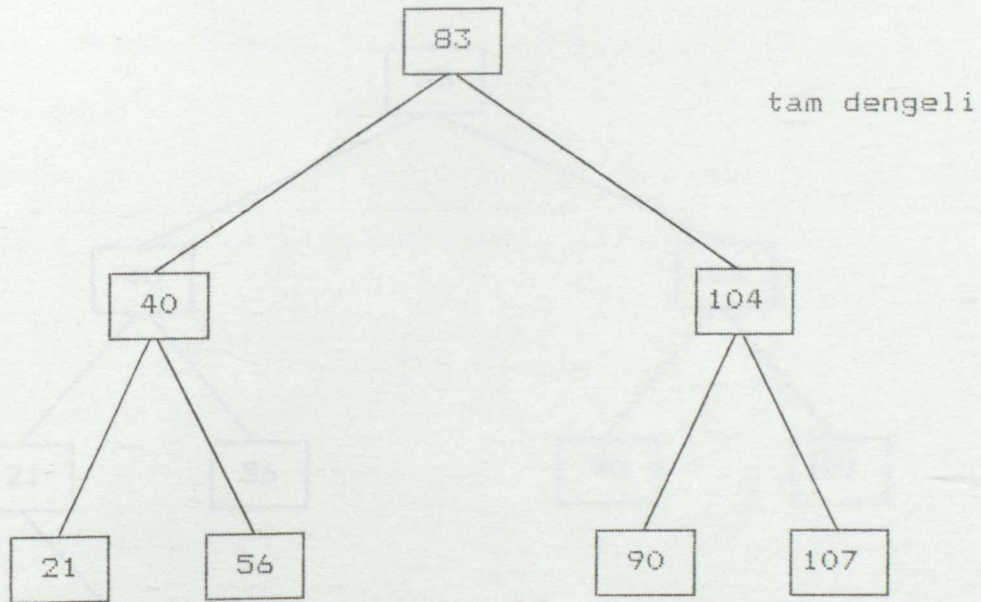
4.2.5 Sıralama Ağaçları ile Arama

Sıralama ağaçlarından daha önce bahsedilirken sıralama ağaçlarının sola çarpık veya sağa çarpık olabileceğini bu durumun arama zamanını arttıracaklarını belirtmiştik. Şimdi Bu çarpıklığı önleyecek ' dengelenmiş ikili arama ağacı ' kuruluşuna ilişkin bazı kavramlar verelim.

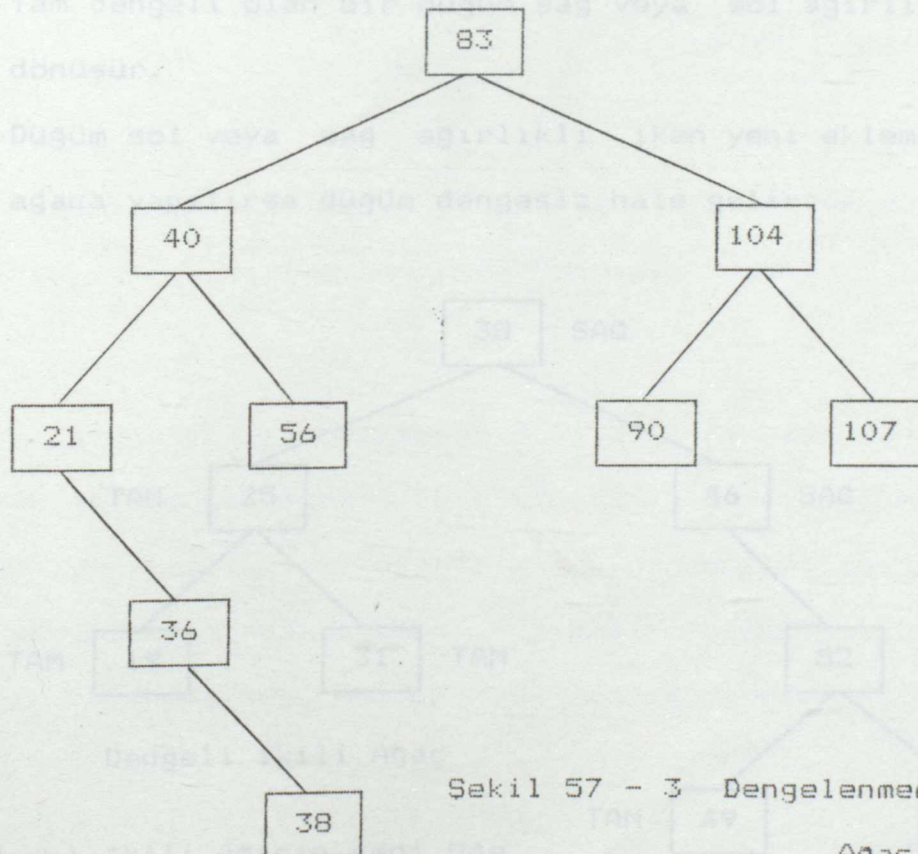
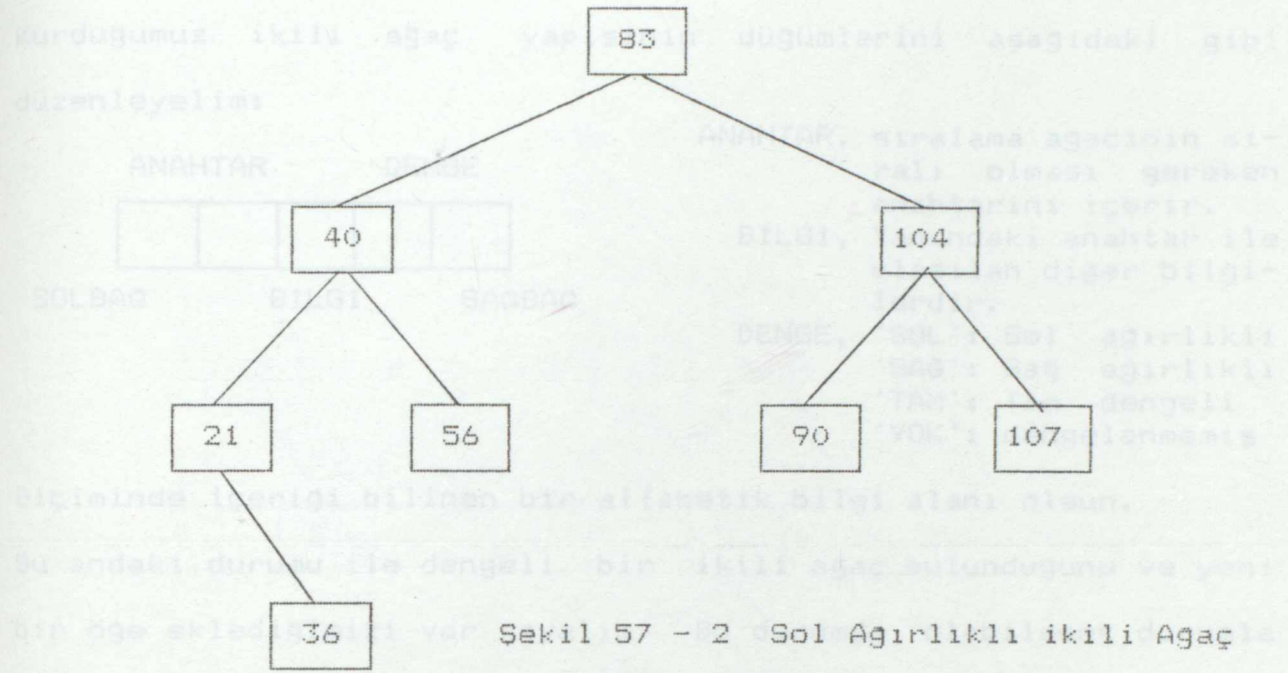
Bir ikili ağaç aşağıdaki üç durumdan birinde ise ' dengelenmiş ikili ağaç ' olarak belirlenir.

- i) Sol ağırlıklı : Sol alt ağaçtaki en uzun iz sağ alt ağaçtaki en uzun izden sadece bir birim fazladır.
- ii) Tam dengeli : Her iki alt ağaçtaki en uzun iz boyutları eşittir.
- iii) Sağ ağırlıklı : Sağ alt ağaçtaki en uzun iz sol alt ağaçtaki en uzun izden yalnızca bir birim fazladır.

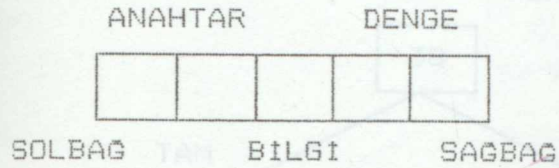
Bu durumlardan birine uymayan ikili ağaç dengelenmemiş ikili ağaçtır, aşağıda çeşitli örnekler görelim.



Şekil 57 - 1 Tam Dengeli İkili Ağaç Örneği



Kurdugumuz ikili ağaç yapısının düğümlerini aşağıdaki gibi düzenleyelim:



ANAHTAR, sıralama ağacının sıralı olması gereken anahtarını içerir.

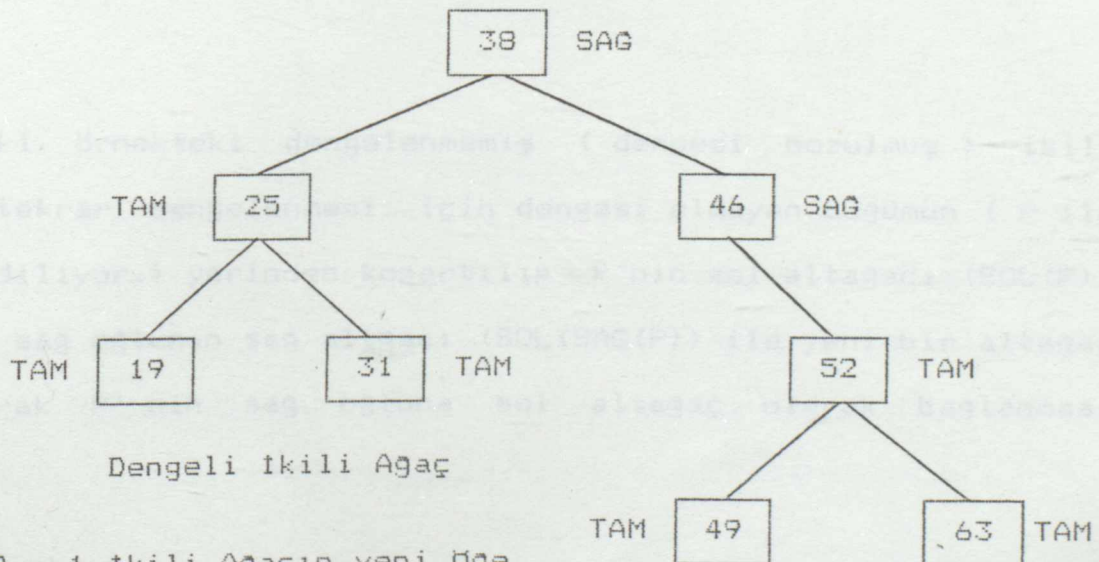
BİLGİ, Yanındaki anahtar ile ulaşılan diğer bilgilerdir.

DENGİ, 'SOL': Sol ağırlıklı
'SAĞ': Sağ ağırlıklı
'TAM': Tam dengeli
'YOK': dengelenmemiş

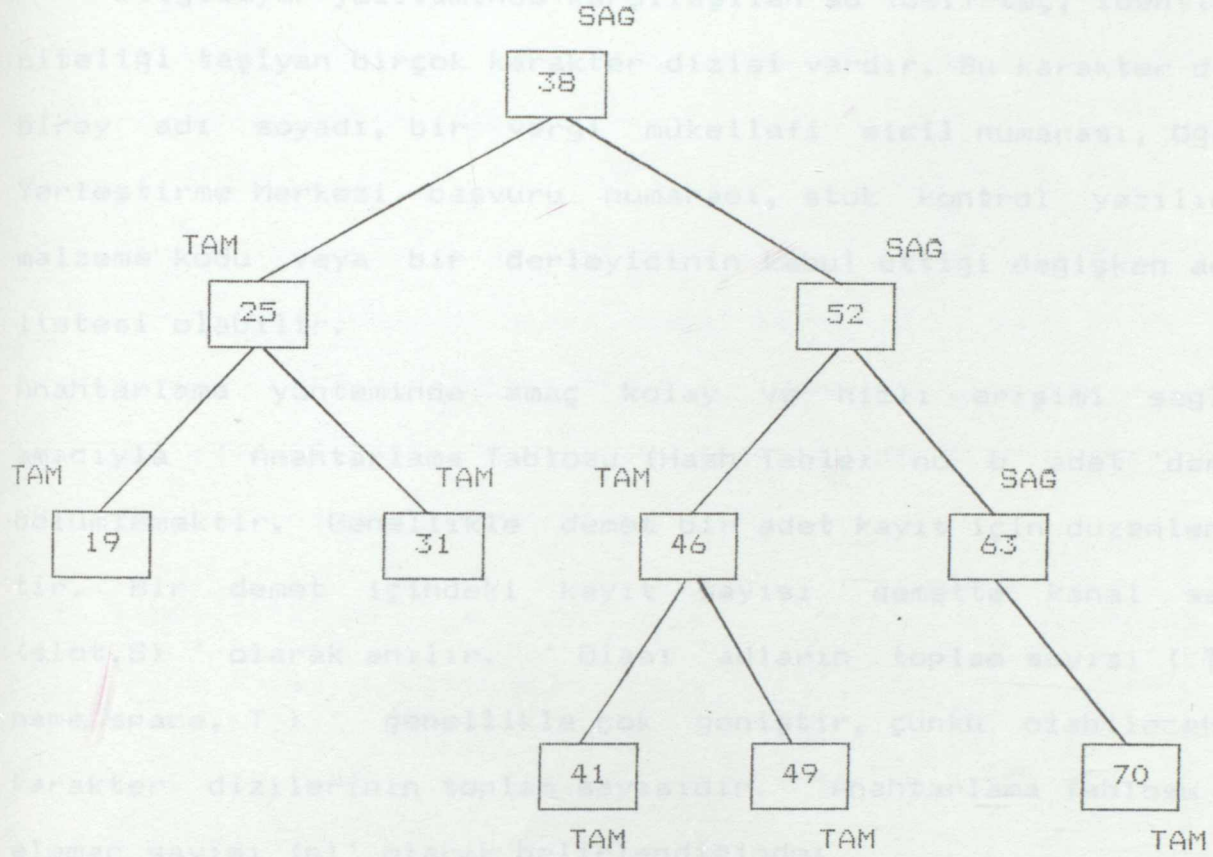
Biçiminde içeriği bilinen bir alfabetik bilgi alanı olsun.

Su andaki durumu ile dengeli bir ikili ağaç bulunduğunu ve yeni bir öge eklediğimizi var sayalım. Bu durumda olabilecek durumla şunlardır:

- Düğüm sol veya sağ ağırlıklı iken şimdi tam dengeli olur.
- Tam dengeli olan bir düğüm sağ veya sol ağırlıklı duruma dönüşür.
- Düğüm sol veya sağ ağırlıklı iken yeni ekleme ağırlıklı ağaca yapılırsa düğüm dengesiz hale gelir.



Sekil 58 - 1 İkili Ağacın yeni Öge Eklemede Yeniden Dengelenmesi - 1



Sekil 58 - 3 İkili Ağacın Üge Eklemede

Yeniden Düzenlenmesi

Simdi bu ağaçtan koparılan 46 sayısı ile yeri değişen 49 sayılarının özelliğine dikkat edelim. Bu sayılar sıralama ağacını kök ortada yöntemi ile dolaşma sırasında bu düğümün önceki düğüm'ü ve sonraki düğüm'üdür. Bu düğümlerin kolayca ve hızla bulunabilmesi dengelenmiş sıralama algoritmaları için önem kazanmaktadır.

4.3 ANAHTARLAMA YÖNTEMLERİ

Bilgisayar yazılımında karşılaşılan ad (belirteç, identifier) niteliği taşıyan birçok karakter dizisi vardır. Bu karakter dizisi birey adı soyadı, bir vergi mükellefi sicil numarası, Öğrenci Yerleştirme Merkezi başvuru numarası, stok kontrol yazılımında malzeme kodu veya bir derleyicinin kabul ettiği değişken adları listesi olabilir.

Anahtarlama yönteminde amaç kolay ve hızlı erişimi sağlamak amacıyla 'Anahtarlama Tablosu (Hash Table)' nu b adet 'demet'e bölümlenektir. Genellikle demet bir adet kayıt için düzenlenmiştir. Bir demet içindeki kayıt sayısı ' demette kanal sayısı (slot, S) ' olarak anılır. ' Olası adların toplam sayısı (Total name space, T) ' genellikle çok geniştir, çünkü olabilecek tüm karakter dizilerinin toplam sayısıdır. 'Anahtarlama Tablosu dolu eleman sayısı (n)' olarak belirlendiğinde;

$$\text{belirteç yoğunluğu (identifier density)} := \frac{n}{T}$$

$$\text{doldurma yoğunluğu (loading factor)} := \frac{n}{s.b}$$

anahtarlama fonksiyonu ise f olarak tanımlanır.

$T \gg b$ olduğundan birbirinden farklı iki ad (yada daha çok ad) aynı demette bulunacaktır. Öteki deyişle $f(11) = f(12)$, $11 \neq 12$ olacaktır. Bu durum oluştuğunda buna ' anahtarlama çarpışması ' adı verilir. Eger demet içindeki kanal sayısı (s) bir olarak seçilmişse ve boş ise bir kanal'a yerleştirilir ve anahtarlama çarpışması çözümlenmiş olur. Eger tüm kanallar dolu yada $s = 1$

seçilmişse ' anahtarlama demet taşması (bucket overflow) oluşur. $S = 1$ seçmek gelenek olduğundan her anahtarlama çarpışması anahtarlama demet taşmasına neden olur ve sözlü ifade de bu iki kavram birbirini yerine kullanılabilir.

Örnek:

Bir BASIC derleyicisi ilki harf olmak üzere harf-rakam dizilerini değişken adı olarak kabul etmektedir. En uzun değişken adı 4 karakter olacaktır. Bu anda değişken tablosunda 23 değişken adı vardır. Yapı 26 demet ve demetteki kanal sayısı $S = 2$ olarak belirlenmiştir.

$b := f(\text{MOD}(\underbrace{\text{ilk(AD)} - 'A'}_{\text{6 ikil ASCII kodu}}, 26)$

6 ikil ASCII kodu

$* T = 26 \cdot (26 + 10)^3$

$T = 1\ 203\ 056$

* Belirteç yoğunluğu

$= 23 / 1203056$

$= 1.8 \cdot 10^{-5}$

* Doldurma yoğunluğu

$x = n / (s \cdot b) = 23 / (2 \cdot 26) = 0.44$

* Verilen örnekte değişken adının ilk harfinin 6 ikil (byte) ASCII tablosundaki sayısal karşılığı bulunmakta bu değerden A harfinin değeri çıkarılmakta elde edilenin 26'ya bölümündeki kalan alınmaktadır.

f	
0	ATOP ALT
1	BTOP BX
2	CTOP CX
3	D1 D2
4	
5	
6	
7	
8	
9	
.	
.	

Örnekte ilk anahtarlama çarpışması 'DX' değişkeni geldiği durumda oluşacaktır. Ancak bu tür basit anahtarlama fonksiyonları ile birçok çarpışma olacak, bu tür taşmaların ele alınması bilgisayar zamanını arttıracaktır. Bir anahtarlama fonksiyonu'nun hem kolay hesaplanabilir olması hem de çarpışmaları azaltacak, eldeki adlar listesini demetlere oldukça dengeli biçimde dağıtacak özellikte olması istenir.

Anahtarlama fonksiyonları iki gruba ayrılır.

i) Düzgün anahtarlama fonksiyonları

ii) Uyarılmış anahtarlama fonksiyonları

Düzgün anahtarlama fonksiyonları rastgele üretilecek karakterlerin birleşiminden oluşacak 'ad'ların demetler arasında dengeli dağılmasını sağlar. Uyarılmış anahtarlama fonksiyonları ise eldeki verilerin özellikleride incelenerek özel olarak kurulurlar.

Şimdi bazı anahtarlama fonksiyonlarını inceleyelim:

4.3.1 DÜZGÜN ANAHTARLAMA FONKSİYONLARI

4.3.1.1 Karenin Ortası (Mid - square) Yöntemi

Bu yöntemde ad bir sayısal değer gibi bir bellek sözcüğü içine konur ve bellek sözcüğünün karesi alınır. Bu işlem sırasında bazı ikili sayılar üst taşma (overflow) ile kaybolur. Bazı kısa bellek sözcüklerinde ikili sayılar hesap dışı bırakılarak ortadaki r adet ikili sayı demet numarası olarak kullanılır. Tablo boyuda r olarak belirlenir. Birkaç aşamadan sonra oldukça düzenli dağılım sağlar. Ancak 32 ikillik makinalarda 4 adet EBCDIC karaktere bağlı olmak bazı düzensizlikler oluşturabilir.

r	2^r
6	64
7	128
8	256
9	512
10	1024

4.3.1.2 Bölme ve Kalan (Division) Yöntemi

Bir bellek sözcüğü içine yerleştirilmiş karakter dizisinin karesi alınacağına bir sayıya (M) bölünür. Kalan demet numarası olarak alınır. Bu yöntemde düzgün anahtarlama yöntemidir.

not1 : Bu yöntemde $M = 2^1$ sakıncalıdır.

Çünkü programcılar aynı birkaç karakterle başlayan değişken adlarını tercih ederler.

not2 : M sayısının 20 sayısından daha küçük asal sayı böleni bulunmalıdır.

not3 : $M = r^k \pm a$ r : Karakter kümesi tabanı (ASCII için $r = 64$ EBCDIC için $r = 256$)

eşitliği k ve a değerleri için sağlanmalıdır.

4.3.1.3 Katlama (Folding)

Bu yöntemde 'ad' eşit boyda (son kesim dışında) kesimlere bölünür ve bunların toplamı demet numarası olarak alınır.

4.3.2 UYARLAMIS ANAHTARLAMA FONKSİYONLARI

4.3.2.1 Basamak çözümlemesi (Digit Analysis)

Derleyicinin ayrılmış sözcükleri gibi oldukça durgun (Statik) bir veri kümesi üzerinde işlemler varsa bu yöntem kullanılabilir.

Yapılacak çözümlleme ile ayırıcılığı az olan basamaklar atılır. Geri kalan yeterli sayıda basamak birleştirilerek demet numarası bulunur.

4.3.2.2 Alfabetik Konum Çözümlemesi

Anahtarlama için kullanılan bilgi alanı bir alfabetik bilgi alanı ise (örneğin soyadı bilgisi) anahtarlama fonksiyonunu kurarken hangi basamakların en çok ayırıcı olduğu da bir veri altkütmesi üzerinde incelenmelidir. Örneğin Türkçe soyadlarında birinci ve üçüncü basamağa dayalı olarak kurulan anahtarlama fonksiyonları daha iyi bir demet ayırımı sağlarken ikinci basamakta harf dağılımı Türkçenin sekiz sesli harfi üzerinde daha yoğun bir dağılım göstermekte ve anahtarlama da tercih edilmeyen bir basamak olmaktadır.

4.3.3 ANAHTARLAMA TABLOSUNDA TAŞMA DENETİMİ

Bir anahtarlama tablosunun en önemli özelliği ilk demet taşmasında tıkanıp kalmamasıdır. Bunu sağlamak için anahtarlama tablosunun ve taşmaların her demet için bir tek bağlı liste olabileceği ilk akla gelen çözümdür. Ancak çok kolay ve doğal gibi gelen bu çözümün tüm öğeleri birer bağlaç alanı taşıması gibi ek bir bellek yükü ve kalabalıklaşan demetlerde artan arama zamanı gibi bazı sakıncaları vardır. Şimdi olası yöntemleri görelim:

4.3.3.1 Taşma Alansız Doğrusal Arama (Linear Probing)

Demet 12 taşarsa aşağıya doğru aranır ilk boş yere (demet 13) konur. Bundan sonra demet 11' e gelen bir eleman

ancak demet 15' de yer bulabilecektir. Bu yöntemde doluluk oranı arttıkça ardışık dolu demetlerin oranı artar, arama tekniği hızla kötüleşir. Doluluk oranının düşük değerleri ($X < 0.40$ gibi) için kolay ve pratik bir yöntemdir.

Yeni —		Demet 10
	*****	Demet 11
	*****	Demet 12
		Demet 13
	*****	Demet 14
		Demet 15

Sekil 59 Taşma Alansız Anahtarlama Tablosu

4.3.3.2 Taşma Alansız Yeniden Anahtarlama (Rehashing)

Bir çarpışma durumunda ardışık demette yer aramak yerine kalan $((f(X) + i), b)$ fonksiyonu yada

$$0 < i < b-1 \quad b : \text{demet sayısı}$$

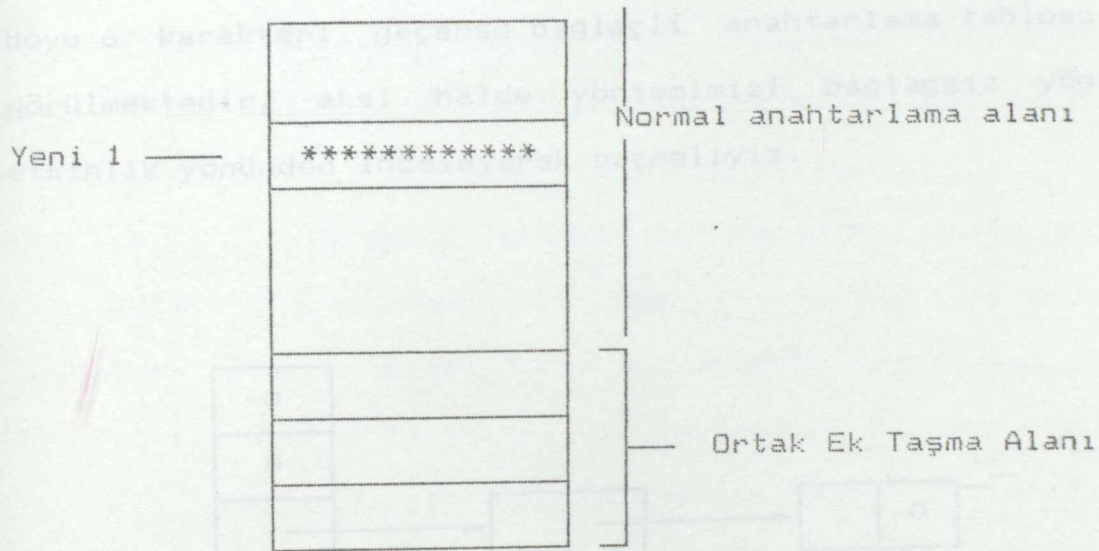
$$\text{kalan } ((f(X) + i^2), b)$$

$$0 < i < (b-1)/2$$

gibi fonksiyonlar ile bulunan demetlerde yer aranır. Yeniden anahtarlama ile hiç ilgisi olmayan demetlerde yapılacak gereksiz karşılaştırmalar azaltılmış olur. Yukarıdaki yöntemle göre doluluk oranının artışına bağlı etkinlik düşümü daha azdır. Ancak yeniden anahtarlama fonksiyonu tablonun dolu ögeleri arasında sonsuz döngüye girebilir, önlem alınmalıdır.

4.3.3.3 Ek Taşma Alanı ile Taşma Denetimi

Bağlaçsız yapıda ortak ek taşma alanı bizi felsefe olarak yukarıdaki iki yöntemin arasında bir performans düzeyine getirir. Çatışma olduğunda yeni gelen eleman ortak ek taşma alanına kaydedilir. Arama sırasında da denetim ilk demet kanalında bulunmayan bilgiyi ortak ek taşma alanında arar.

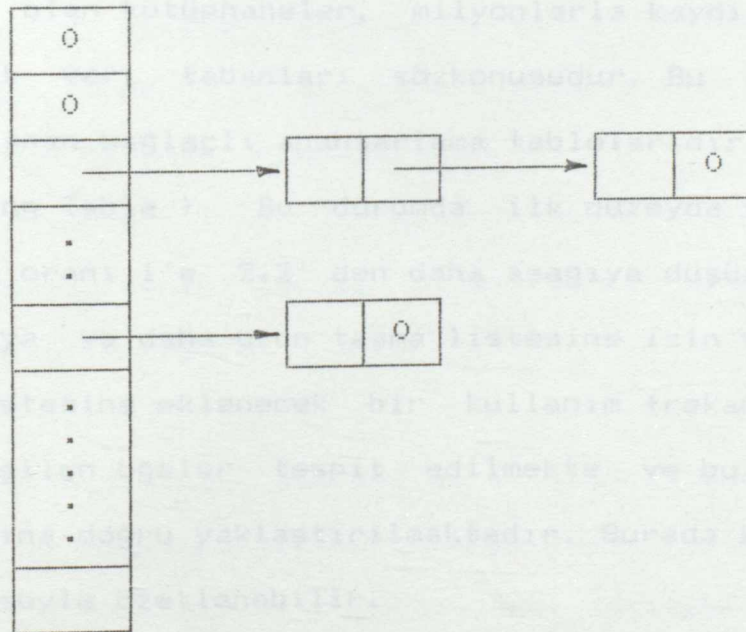


Sekil 60 Ortak Taşma Alanlı Anahtarlama Tablosu

4.3.3.4 Bağlaçlı Anahtarlama Tablosu

Taşma denetimi konusuna başlarken tek bağlı listenin doğal seçim gibi alınmaması gerektiğini belirtmiştik. Ancak bağlaçlı olmayan anahtarlama tablolarında 0.40 - 0.50 doluluk oranını aşmama tavsiyesi bir çok kaynakta (100 eleman için 220 dolayında tablo yeri önerisi) yerleştirme ve arama etkinliğinin kötüleşmesi açısından önerilmektedir. Bu durumda bağlaçsız anahtarlama tablolarında normal olarak var olan % 50 dolayındaki bellek ek

yükü bağlaçlı anahtarlama tablolarında 6 karakter veri, % 50 doluluk oranlı yapılarda ortaya çıkan bellek ek yükü oranıdır ve öğelerin içerdigi bilgi boyu 6 karakteri geçtikçe bağlaçlı anahtarlama tablolarında kaybedilen bellek oranı % 50' nin altına doğru adım adım azalmaktadır. Öte yünden taşma durumundaki yeniden anahtarlama işlemine göre bağlaçlı listede bir sonraki elemana ulaşma daha kolay ve etkindir. Özetle herbir öge içindeki bilgi boyu 6 karakteri geçerse bağlaçlı anahtarlama tablosu avantajlı görülmektedir, aksi halde yöntemimizi bağlaçsız yöntemleri de etkinlik yönünden inceleyerek seçmeliyiz.



Sekil 61 Bağlaçlı Anahtarlama Tablosu

4.3.3.5 Otomatik Uyarlanan Bağlaçlı Anahtarlama Tablosu

Yukarıda açıklanan taşma tablosu yöntemleri anahtarlama ile yerleştirilecek verilerin sayısı birkaç bini aşmadığı durumlarda pratik hayatta kullanılabilmektedir. Bilgisayar yazılımında sorunların pek çoğunda bu sınır aşılmaktadır. Ancak kütüphane sistemleri, nüfus kayıt sistemi, seçmen kütüğü gibi büyük kapasiteli verilerin işlenmesi söz konusu olduğunda bir veri için 2.2 demet alanı pratik kuralını taşma öncesi adımda uygulama güçleşmektedir. Örneğin 70000 kitabı olan bir kütüphanede 70000×2 karakterlik birinci düzey dizin (demetlerin ilk kanallarının yeri) yanında $70000 \times 1.2 \times 2$ byte yerin bellekte her zaman boş kalmasını önermek her zaman mümkün değildir. Öte yandan 400 bin, 800 bin ve milyon kitabı olan kütüphaneler, milyonlarla kaydı olan nüfus ve sosyal güvenlik veri tabanları söz konusudur. Bu durumda çözüm otomatik uyarlanan bağlaçlı anahtarlama tablolarıdır. (Self- Adaptive Hash Coding Table) Bu durumda ilk düzeyde veri sayısının demet sayısına oranı 1'e 2.2' den daha aşağıya düşürülmekte, daha fazla çatışmaya ve daha uzun taşma listesine izin verilmektedir. Ancak taşma listesine eklenecek bir kullanım frekansı sayacı ile daha sıkça erişilen öğeler tespit edilmekte ve bu öğeler taşma listesinin başına doğru yaklaştırılmaktadır. Burada kullanılabilecek algoritma şöyle özetlenebilir.

- i) Yeni bir elemanın eklenmesinde çatışma durumunda yeni elemanın kullanım sayacına 1 koyarak ait olduğu demetin taşma listesinin sonuna ekle.
- ii) Bir elemanın aranması durumunda kullanım frekansı sayacına

bir ekle.

- iii) Eger erişilen elemanın kullanım frekansı sayacı azalan sırayı bozan düzeyde ise liste içi düzenleme yap.
- iv) Kullanım frekansı sayacı taşma oluşturacak düzeye gelmiş ise o listedeki tüm kullanıcı sayaçlarını ikiye böl.

Bu tür önlemler özellikle üniversite kütüphaneleri için yapılmış yazılım sistemlerinde işletim performansını yüksek tutmak için gerekli olmaktadır, çünkü

- i) Bu gün kaydedilmiş ve çok kullanılan bir kaynak 3 - 4 yıl sonra az kullanılır hale gelmektedir.
- ii) İlgili dersi bahar yarıyılında olan bir kitap Bahar yarıyılında çok sık aranmakta, aynı özelliği güz yarıyılında göstermemekte ancak sonraki bahar yarıyılında sık kullanım özelliği tekrarlanmaktadır.
- iii) Verilen bir ödev, bir güncel konu nedeniyle bazı kaynakların kullanımı onbeş yirmi günlük dönemlerde olağanüstü artabilmektedir.

Özetle, veri sayısının bir kaç bini geçtiği anahtarlama sorunlarında hem anahtarlama yöntemi hem de taşma yöntemi eldeki sorunun özelliklerine göre dikkatle incelenerek seçilmelidir.

5 UYGULAMA

5.1 Çizge Yapısı ile Çözülen Mühendislik Problemleri

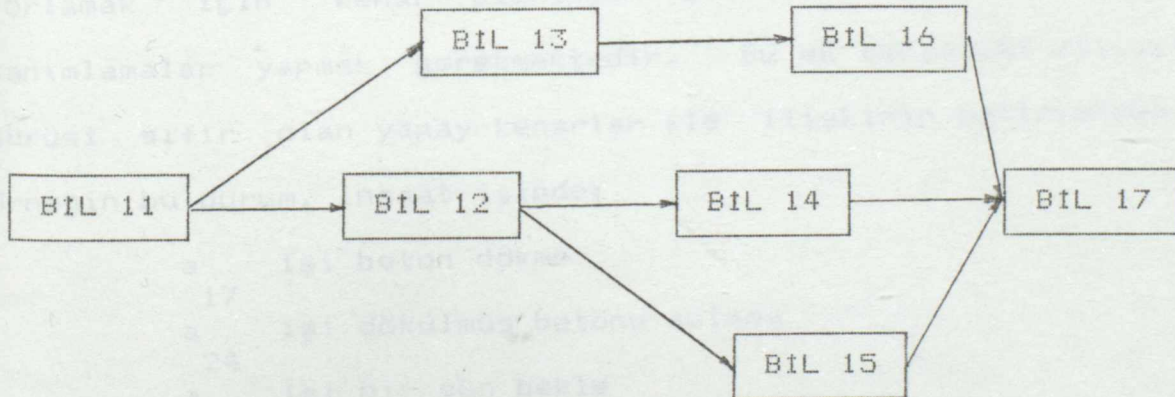
5.1.1 Proje Yönetim ve Değerlendirme Sistemi

Çizge uygulamalarının bilgisayar yazılımında en çok bilinenleri arasında CPM (Practical Path Method) ve PERT (Project Evaluation And Review Technique) yöntemleri vardır. Bu yöntemlerin kullanılabilmesi için bir projenin gerçekleştiriminin aşağıda verilen iki yönlü çizge biçiminden biri ile tanımlanması gerekir.

i) Düğüm üzerinde etkinlik gösterimi

ii) Kenar üzerinde etkinlik gösterimi

Örneğin bir bölümün ders önkoşulları listesi bir 'düğüm üzerinde etkinlik' gösterimi olabilir. Aşağıdaki çizgede düğümler dersleri kenarlar o dersi almadan önceki önkoşulları gösterir.



DÜĞÜM ÜZERİNDE ETKİNLİK GÖSTERİMİ

Düğüm üzerinde etkinlik gösterimi inşaat mühendisliği gösteriminde daha fazla benimsenen bir yöntemdir. Bu yöntem içinde yapılacak işler düğümler üzerinde gösterilmekte ve bu düğüme gelen ve giden kenarlar işlerin birbirlerine göre eş zamanlanması için kullanılmaktadır. Bu yöntem içinde bir iş tanımlanırken bu işin eş zamanlı ilişkisi tanımlanabilmektedir. Bu ilişki türleri şunlardır.

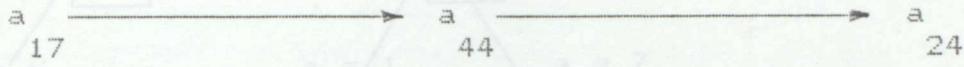
- i) Başla - Başla ilişkisi (START To START) (Belirlenen iki iş eş zamanlı başlar)
- ii) Bitiş - Başla ilişkisi (FINISH To START) (İkinci iş belirtilen birincisinin bitiminde başlar)
- iii) Başla - Bitiş ilişkisi (START To FINISH) (Birinci işin bitişi ikinci işin başlayışından önce olmalıdır)
- iv) Bitiş - Bitiş ilişkisi (FINISH To FINISH) (iki iş eş zamanlı olarak bitmelidir.

Kenar üzerinde etkinlik gösterimi yukarıdaki ' iii ' ilişkiyi doğal olarak sağlamakta diğer tür ilişkileri belirlemek ve zorlamak için kenar üzerinde etkinlik gösteriminden ek tanımlamalar yapmak gerekmektedir. Bu ek tanımlama maliyeti ve sürüsü sıfır olan yapay kenarlar ile ilişkinin belirlenmesidir. Örneğin bu durum, inşaat işinde;

- a işi beton dökme
- 17
- a işi dökülmüş betonu sulama
- 24
- a işi bir gün bekle
- 44

ise ortaya çıkmaktadır. Betonun susuz kalıp yanmaması için döküldükten bir gün sonra genellikle 4 - 5 gün süreyle sulanması

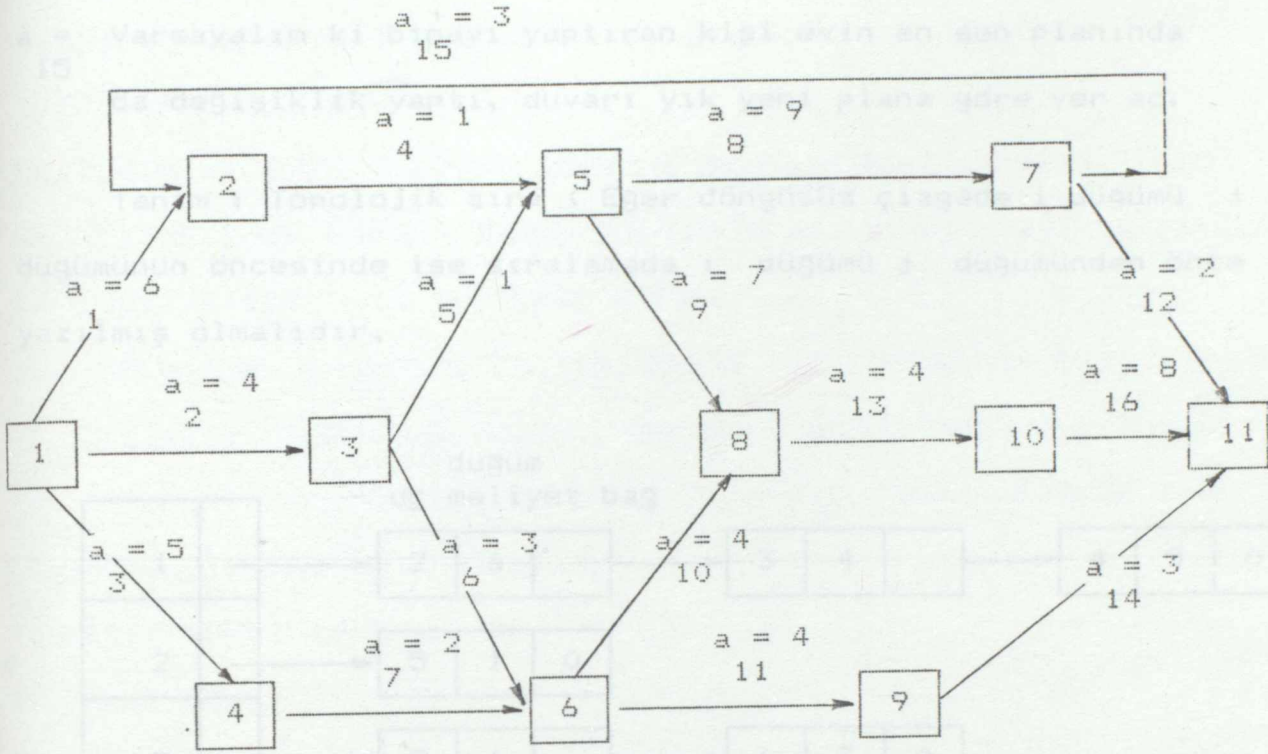
gerekmektedir. Eger kenar üzerinde etkinlik gösterimini kullanıyorsak ve yapay kenarlar ile



iş dizisini ara vermeden uygulamaya zorlanmıyorsak, bilgisayar destekli sistem kaynak atama sırasında a_{17} , a_{44} , a_{24} arasına boş zamanlar koyabilmekte, bu durum ise pratikteki gereğinden ayrılan bir durum oluşturmaktadır. Şimdi bir kritik yol bul algoritması verelim:

5.1.1.1 Kritik Yol Problemi

Aşağıda hayali bir projenin 'Kenar üzerinde etkinlik gösterimi' ile verilmiş yönlü çizgesini bulunmaktadır. Bu tür bir yönlü çizgenin 'yönlü döngü' içermemesi gerektiği kolayca anlaşılır, öteki deyişle döngüsüz çizge oluşmalıdır. (Basarıla bitirilmiş bir aşama geriye dönüyorsa projeyi bitirme olanakı yoktur.) Ancak birkaç yüz düğüm içeren bir çizgede bu özelliğin yazılım ile denetlenmesi ve garanti edilmesi gerekir. Aşağıdaki örnek 10 düğüm ve 14 kenardan oluşmuştur. Şimdi 7. düğümden 2. düğüme doğru bir 15. kenar ekleyerek döngüsüz çizge özelliğini bozalım ve yazacağımız algoritma ile bunu denetleyelim.

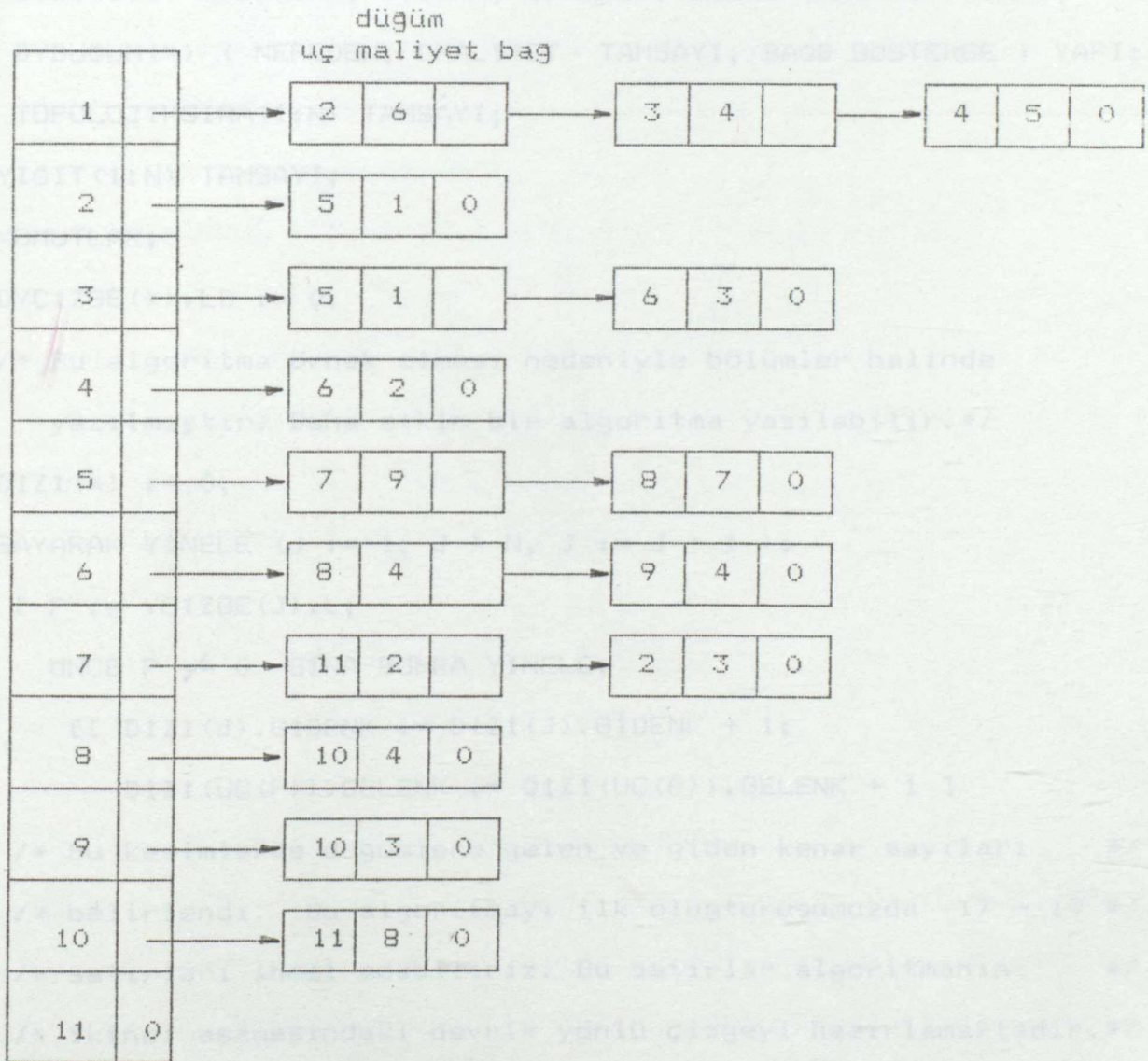


KENAR ÜZERİNDE ETKİNLİK YÖNTEMİ İLE PROJE GÖSTERİMİ

- a = Temel çukuru kaz ve zemin betonunu dök
 1
 a = Foseptik çukuru kaz
 2
 a = Bahçe duvarı ve site içi yolları yap
 3
 a = Foseptik çukuru taş duvarı örülecek
 4
 a = Kolon betonları ve tavan betonu
 5
 a = Duvarları ör
 6
 a = Su ve elektrik bina bağlantısı kaz
 7
 a = Sıhhi tesisat ve elektrik tesisatını döşe
 8
 a = Toprağı bahçeye yay, taş ve molozu bir köşeye topl
 9,10,11
 a = Bahçeye süs bitkileri dik
 12
 a = Su ve elektrigi bağla, kullanım denemesi yap
 13
 a = Yağlı boya yap, cam taktır, musluk, duş, priz tamamla
 14
 16

a = Varsayalım ki binayı yaptıran kişi evin en son planında
15
da değişiklik yaptı, duvarı yık yeni plana göre yer ac.

Tanım : Topolojik sıra (Eger döngüsüz çizgede i düğümü j
düğümünün öncesinde ise sıralamada i düğümü j düğümünden önce
yazılmış olmalıdır.




```

1 KRITIKYOLBUL(N, YÖNLÜÇİZGE, TABLO) MANTIKSAL YORDAMI;
2 DISYAPILAR;
3 DÜĞÜM(*) ( UC, MALİYET TAMSAYI, BAĞ GÖSTERGE ) YAPI;
4 YÇİZGE(1:N) ( BASLA TAMSAYI, L GÖSTERGE ) YAPI /*yönlü çizge*/
5 DYÇİZGE(1:N) ( LD GÖSTERGE ) YAPI;
6 TABLO(1:N) ( KBASLA, KBITİS, ERKEN, GEC, ESNEK TAMSAYI ) YAPI;
7 DİZİ(1:N) ( GELENK, GİDENK, ERKENOL, GECOL TAMSAYI ) YAPI;
8 DYDÜĞÜM(*) ( NEREDEN, DMALİYET TAMSAYI, BAĞD GÖSTERGE ) YAPI;
9 TOPOLOJİKSIRA(1:N) TAMSAYI;
10 YİĞİT(1:N) TAMSAYI;
11 KOMUTLAR;
12 DYÇİZGE(*).LD := 0;

/* Bu algoritma örnek olması nedeniyle bölümler halinde
   yazılmıştır. Daha etkin bir algoritma yazılabilir.*/

13 DİZİ(*) := 0;
14 SAYARAK YİNELE ( J := 1, J > N, J := J + 1 );
15 [ P := YÇİZGE(J).L;
16   ÖNCE P ≠ 0 SİNA SONRA YİNELE;
17   [[ DİZİ(J).GİDENK := DİZİ(J).GİDENK + 1;
18     DİZİ(UC(P)).GELENK := DİZİ(UC(P)).GELENK + 1 ]

/* Bu kesimlerde düğümlere gelen ve giden kenar sayıları
/* belirlendi. Bu algoritmayı ilk oluşturduğumuzda 17 - 19 */
/* satırları ihmal edebiliriz. Bu satırlar algoritmanın
/* ikinci aşamasındaki devrik yönlü çizgeyi hazırlamaktadır.*/

19 [ EĞER DÜĞÜMBUL( SERBEST, X )
20   İSE [ DYDÜĞÜM(X).NEREDEN := J;

```

```

21      DYDUGUM(X).DMALIYET := DUGUM(P).MALIYET;
22      DYDUGUM(X).BAGD := YCIZGE(UC(P)).LD;
23      DYCIZGE(UC(P)).BAG := X ]
24      DEGI LSE [ KRITIKYOLUBUL := YANLIS;
25                  'SERBEST YETERSIZ') YAZ; DON ]
26      P := YCIZGE(P).BAG ] /* Aynı liste üzerinde ilerle */
27 ] /* Bir başka liste ele al Simdi Topolojik sıra ve düğümler-
      deki olayın en erken baslama zamanları belirleniyor. */
28 DIZI(*).ERKENYOL := 0
29 S := 0; /* Yığıt göstergesi */
30 SAYARAK YINELE ( J := 1, J > N, J := J + 1 );
31      [ EGER DIZI(J).GELENK := 0
32          ISE [ S := S + 1; YIGIT(S) := J ] ]
      /* Başlangıçta gelen kenar sayısı sıfır olan düğümler yığıtı
      alındı */
33 SAYARAK YINELE ( J := 1, J > N, J := J + 1 )
34      [ EGER S := 0
35          ISE [ (' YÖNLÜ DÖNGÜ VAR ') YAZ ;
36                  (' EKSİK TOPOLOJİK SIRA ') YAZ;
37                  ( TOPOLOJİKSIRA ) YAZ;
38                  KRITIKYOLBUL := YANLIS; DON ]
      /* Simdi gelen kenar sayısı sıfır olan düğümden
      başlayarak topolojik sıraya alınacak düğümler belirleniyor, bu
      biçimde alınan düğümden çıkan kenarlar başka düğümleri de serbest
      bırakmak üzere siliniyor ( Daha dogrusu vardıkları yerdeki gelen
      kenar sayısı azaltılıyor ) */

```



```

39     DEĞİLSE [ I := YIGIT(S); S := S - 1;
40             TOPOLOJİKSIRA(J) := I;
41             P := YÇİZGE(I).L;
42             ÖNCE P ≠ 0 SİNA SONRA YİNELE;
43             [ K := DÜĞÜM(P).ÜÇ;
44             DİZİ(K).GELENK := DİZİ(K).GELENK - 1
45             SAYARAK YİNELE DİZİ(K).ERKENOL := ENBÜYÜK(DİZİ(K).ERKENOL,
46             DİZİ(I).ERKENOL + DÜĞÜM(P).MALİYET);
47             ÖNCE P ≠ 0 SİNA SONRA YİNELE;
48             EĞER DİZİ(K).GELENK := 0
49             [ TABLO (J) := DİZİ(K).GELENK; YIGIT (S) := K ]
50             P := DÜĞÜM(P).BAĞ ]
51             ]
52             ]
53             /* Program bu noktaya ulaşırsa yönlü döngü yoktur.
54             TOPOLOJİKSIRA oluşturulmuştur. Dğümler için en erken
55             başlama zamanları ( DİZİ(*).ERKENOL ) belirlenmiştir. */
56             BÜYÜK := BÜYÜKBUL(N, DİZİ(*).ERKENOL);
57             /* Bu yordam en büyük ERKENOLay zamanını bulur */
58             DİZİ(*).GECOL := BÜYÜK ;
59             /* Şimdi TOPOLOJİKSIRA dizisini ters yönde kullanarak
60             kenarların en geçbaşlama zamanlarını bulalım */
61             SAYARAK YİNELE ( K := N, K < 1, K := K - 1 );
62             [ J := TOPOLOJİKSIRA(K);
63             P := DYÇİZGE(J).LD;
64             ÖNCE P ≠ 0 SİNA SONRA YİNELE ;
65             [ I := DYDÜĞÜM(P).NEREDEN;

```

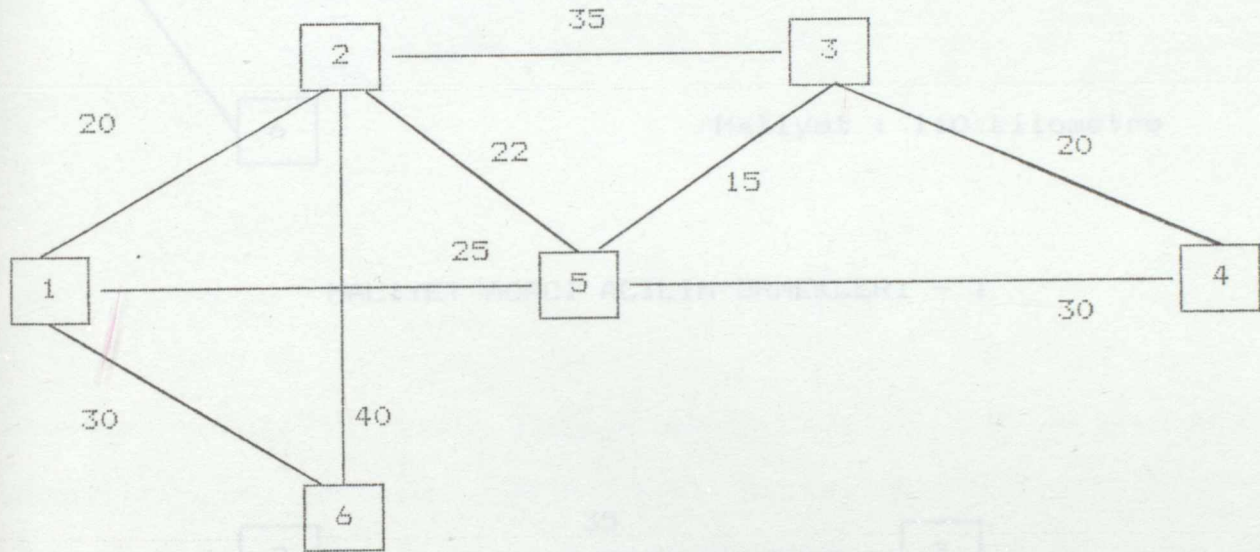
```

58     Dizi(I).GECOL := ENKUCUK(Dizi(I).GECOL,
        Dizi(J).GECOL - DYDUGUM(P).DMALIYET);
59     P := DYDUGUM(P).BAGD 1
60 /*     Simdi GECOLAY zamanları da belirlendi. Aşağıda hem
        yazma işlemi yapılmakta hemde TABLO oluşturulmaktadır */
61 ( 'KENAR ', 'ERKEN', ' GEC ', ' ESNEK ' ) YAZ;
62 SAYARAK YINELE ( J := 1, J > N, J := J + 1 );
63 [ P := YONLUCIZGE(J).L
64     ÖNCE P ≠ 0 SINA SONRA YINELE
65     [ TABLO (J).GEC := DYDUGUM(DUGUM(P).UC).
        DMALIYET -DYDUGUM(P).DMALIYET;
66     TABLO(J).ESNEK := TABLO(J).GEC - Dizi(J).ERKENOL ;
67     TABLO(J).ERKEN := Dizi(J).ERKENOL;
68     TABLO(J).KBASLA := J ;
69     TABLO(J).KBİTİS := DUGUM(P).UC;
70     ( KBASLA, KBİTİS, ERKEN, GEC, ESNEK ) YAZ;
71     P := DYDUGUM(P).DBAG 1;
72 KRITIKYOLBUL := DOGRU;
73 KRITIKYOLBUL BİTTİ;

```

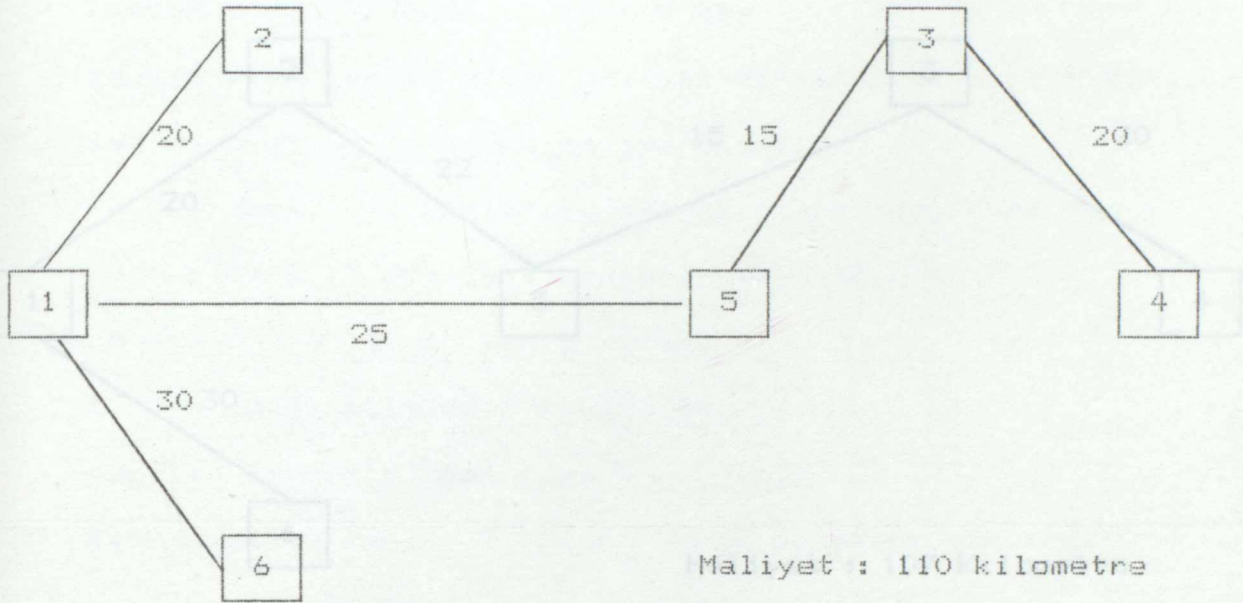

5.1.2 En Az Açılım Ağacının Mühendislik Tasarımında Kullanımı

Bir çizgenin 'açılım ağacı (Spanning tree)' bu çizgenin bütün düğümlerini içinde bulunduran, kenarları, çizgenin kenarlarının bir alt kümesi olan ağaçtır. Çizgede bulunmayan bir kenar açılım ağacında da bulunmamalıdır.

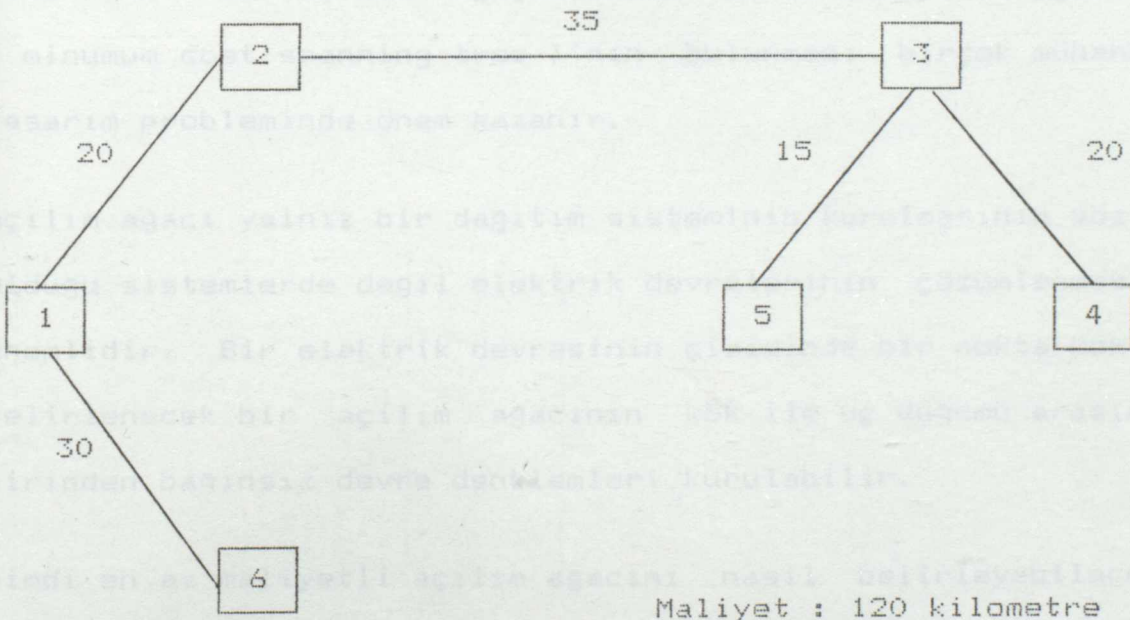


TASARIM İÇİN ÖRNEK YÖNSÜZ ÇİZGE

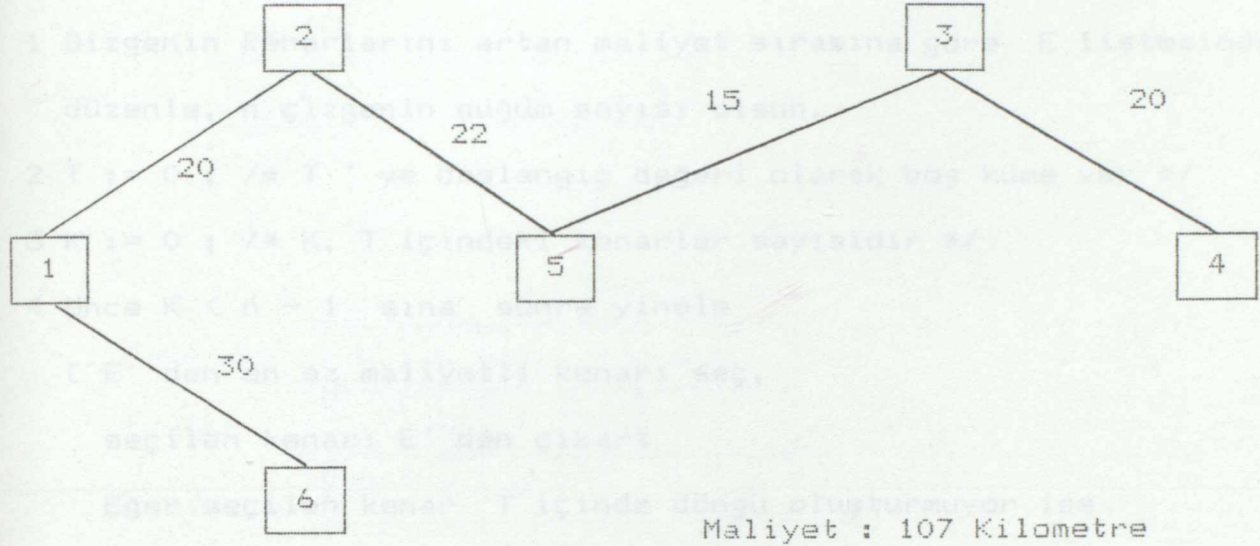
Bir çizge için genellikle birden fazla açılım ağacı vardır ve mühendislik tasarımında zaman zaman değişik seçenekleri inceleyip tartışmak gerekir. Varsayalım ki yukarıdaki çizge yeni elektrik götürülecek bir grup dağ köyü olsun. Burada çizgede belirtilen kenarlar yapımı düşünülebilecek elektrik hatları olsun. Örneğin dağlık bir bölge olan 4. ve 6. düğümler arasında bir hat düşünülmesin. Bu örnek için kurulabilecek açılım ağaçlarından üçü aşağıda çizilmiştir:



MALİYET AĞACI AÇILIM ÖRNEKLERİ - 1



MALİYET AÇILIM AĞACI ÖRNEKLERİ - 2



MALİYET AÇILIM AĞACI ÖRNEKLERİ - 3

Örnekte görüldüğü gibi bir çizgenin birden çok açılım ağacı olabilir. Bu açılım ağaçlarından 'en az maliyetli açılım ağacı' (minimum cost spanning tree)'nin bulunması birçok mühendislik tasarım probleminde önem kazanır.

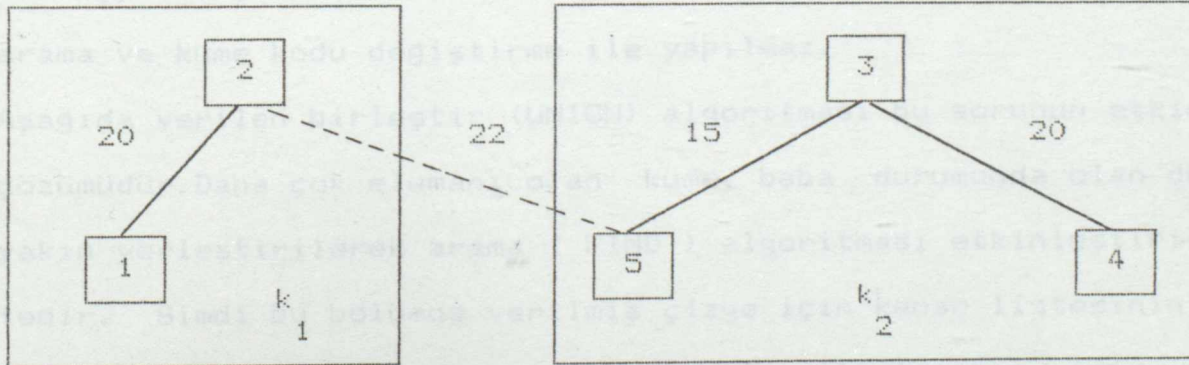
Açılım ağacı yalnız bir dağıtım sisteminin kurulmasının söz konusu olduğu sistemlerde değil elektrik devrelerinin çözülmesinde de önemlidir. Bir elektrik devresinin çiziminde bir nokta kök olarak belirlenecek bir açılım ağacının kök ile uç düğümü arasında bir birinden bağımsız devre denklemleri kurulabilir.

Şimdi en az maliyetli açılım ağacını nasıl belirleyebileceğimizi inceleyelim. Bu soruna ilk düzenli yaklaşımı oluşturan KRUSKAL'dır. Önce bu yaklaşımı üst düzeydeki algoritması ile inceleyelim.

KRUSKAL ALGORİTMASI (Üst düzey)

- 1 Dizgenin kenarlarını artan maliyet sırasına göre E listesinde düzenle, n çizgenin düğüm sayısı olsun.
- 2 $T := 0$; /* T ' ye başlangıç değeri olarak boş küme ver */
- 3 $K := 0$; /* K, T içindeki kenarlar sayısıdır */
- 4 Önce $K < n - 1$ sına sonra yinele
 [E' den en az maliyetli kenarı seç,
 seçilen kenarı E' den çıkart
 Eğer seçilen kenar T içinde döngü oluşturmuyor ise
 [$K := K + 1$; Bu kenarı T' ye ekle]

Bu algoritmanın kurulabilmesi için eklenecek kenarın o ana değin alınmış olan kenarlar ile döngü oluşturulup oluşturmadığı incelenmelidir. Bu kenarın içinde bulunduğu altagaç (Subtree) bir küme numarası olarak tutulur ve gerektiğinde birleştirilirse bu inceleme yapılabilir. Yukarıda verilen yönsüz çizge için kenarların nasıl seçildiğinin örneği aşağıdadır:



Not : Bu aşamada seçilen (2,5) kenarı k_1 ve k_2 kümelerinin birleştirilmesini gerektirir.

KUMENO						
	1	2	3	4	5	6
KUME	0	0	0	0	0	0
TABLO						
	1	2	3	4	5	6
BABAKUME	0	0	0	0	0	0
ISARET	1	1	1	1	1	1
SAYAC	0	0	0	0	0	0

ISARET = 1 ise BABAKUME alanı BABA göstergesi anlamında değilse
KUME numarası anlamındadır.

EN AZ AÇILIM AĞACI ALGORİTMASI BAŞLAYIŞI - 1

Bir kenar eklenirken kenarın iki ucundaki düğümlerin aynı küme içinde olmaması gerekir. Bir kenarın eklenmesi ayrı kümelerde bulunan düğümlerin birleştirilmesini gerektirir. Bu birleştirme birkaç yüz düğümden oluşabilecek çizgelerde her defasında sıradan arama ve küme kodu değiştirme ile yapılmaz.

Aşağıda verilen birleştir (UNION) algoritması bu sorunun etkin bir çözümüdür. Daha çok elemanı olan küme, baba durumunda olan düğüme yakın yerleştirilerek arama (FIND) algoritması etkinleştirilmektedir. Şimdi bu bölümde verilmiş çizge için kenar listesinin (E) ve birleştir (UNION) ve bul (FIND) algoritmalarının işleyişini görerelim.


```

7  TABLO(1:N) ( BABAKUME, ISARET, SAYAC TAMSAYI ) YAPI;
8  KUME(1:N), KUME1B, KUME2B TAMSAYI;
9  X, P GÖSTERGE;
10 KUMENO TAMSAYI;
11 KOMUTLAR;
12 T := 0 ;
13 K := 0 ; P := E ; KUMENO := 1;
14 ÖNCE K < N - 1 SINA SONRA YINELE ;
15 [ EGER P = 0 İSE [ ('KENAR LİSTESİ KURULMAMIS') YAZ ;
16 ENAZACILIMAGACI := YANLIŞ ; DÖN ]
17 DEĞİLSE [
18     KUME1B := BUL(TABLO, KENAR(P).NEREDEN);
19     KUME2B := BUL(TABLO, KENAR(P).NEREYE);
20     EGER KUME1B = 0 VE KUME2B = 0
21     İSE [ /* Kenarın her iki düğümünde ilk kez kümeye
22         girecek */
23         TABLO(KENAR(P).NEREYE).BABAKUME := KUMENO;
24         TABLO(KENAR(P).NEREYE).ISARET := 0;
25         /*Bu kümenin lideri KENAR(P).NEREYE ile işaret edilen oldu*/
26         KUME(KUMENO) := KENAR(P).NEREYE
27         /* Bu kümenin üye sayısının bulunaacağı yer KUME dizisinde */
28         /* KENAR(P).NEREYE'ye yönlendiriliyor. */
29         TABLO(KENAR(P).NEREYE).SAYAC := 2 ;
30         /* Bu kümede iki eleman vardır */
31         TABLO(KENAR(P).NEREDEN).ISARET := 1 ;
32         TABLO(KENAR(P).NEREDEN).BABAKUME := KENAR(P).NEREYE ;

```

```

/* Diğer düğüm küme liderine bağlanıyor */
27     KUMENO := KUMENO + 1 ;

/* Sonraki işlem için yeni küme açılıyor */
28     X := P ; P := KENAR(P).BAG ; K := K + 1 ;
29     KENAR(X).BAG := T ; T := X ;
30 ]
31 EGER KUME1B := 0 VE KUME2B ≠ 0
    ISE [ /* KENAR(P).NEREYE ile işaret edilen düğüm
        bir kümenin ögesi, öteki düğüm buna bağlanacaktır.*/
32     TABLO(KENAR(P).NEREDEN).BABAKUME := KUME2B ;
33     TABLO(KENAR(P).NEREDEN).ISARET := 1 ;
34     TABLO(KUME2B).SAYAC := TABLO(KUME2B).SAYAC + 1 ;
35     X := P ; P := KENAR(P).BAG ; K := K + 1 ;
36     KENAR(X).BAG := T ; T := X ;
37     EGER KUME1B ≠ 0 VE KUME2B = 0
38     ISE [ /* KENAR(P).NEREDEN ile işaret edilen küme
        ögesi öteki kümeye bağlanacaktır. */
39     TABLO(KENAR(P).NEREYE).BABAKUME := KUME1B ;
40     TABLO(KENAR(P).NEREYE).ISARET := 1 ;
41     TABLO(KUME1B).SAYAC := TABLO(KUME1B).SAYAC + 1 ;
42     X := P ; P := KENAR(P).BAG ; K := K + 1 ;
43     KENAR(X).BAG := T ; T := X ;
44 ]
45 EGER KUME1B ≠ 0 VE KUME2B ≠ 0
46     ISE [ /* Her ikiside kümeler içinde eger aynı küme
        ise döngü oluşturur, kenar alınmaz. */

```



```

47 5 KONTROLAR; EGER KUME1B = KUME2B
48 6 P := DUGUM; ISE [ /* Bu ögeyi E listesinden T listesine
49 7 ENCE TABLO(P).ISARET aktarma */
49 8 E /* Bu ögeyi T listesinden X := P; P := KENAR(P).BAG;
50 9 P := TABLO(P).DEGERIVER(X) CAGIR; ]
51 10 I DEGILSE [ BIRLESTIR (KUME1B, KUME2B) CAGIR;
52 11 BUL := P; X := P; P := KENAR(P).BAG;
53 12 BUL BITTI; K := K + 1;
54 KENAR(X).BAG := T; T := X; ]
55 ]
56 ] /* Burası K ile denetlenen döngü sonu */
57 ENAZACILIMAGACI := DOGRU
58 ENAZACILMAGACI BITTI;

```

Bu algoritmayı dizlediğinizde dikkatimizi çekecek noktalardan biri en az açılım ağacının içerdigi kenarları tutan T listesine eklemelerin daima başlangıca yapıldığıdır. E listesinin maliyet yönünden artan sırada olması T listesinin azalan maliyet sırasında oluşması yan etkisi yaratır. Ancak bu önemli değildir, önemli olan bu eklemeler için işlem zamanının sınırlı kullanılmasıdır. Şimdi BUL ve BIRLESTIR algoritmalarını yazalım.

```

1 BUL ( TABLO, DUGUM ) YORDAMI;
2 DISYAPILAR;
3 TABLO(*);
4 DUGUM GOSTERGE;

```

```

5 KOMUTLAR;   TABLO(DUGUM2).SAYAC := TABLO(DUGUM2).SAYAC +
6   P := DUGUM ;   TABLO(DUGUM1).SAYAC ;
7   ÖNCE TABLO(P).İŞARET := 0   SİNA   SONRA YİNELE ;
8   [ /*Küme liderini buluncaya kadar ara   */
9     P := TABLO(P).BABAKÜME
10 ]
11   BUL := P ;
12   BUL BİTTİ ;

```

```

1   BİRLEŞTİR ( DUGUM1, DUGUM2 ) YORDAMI;
2   DİSYAPILAR;
3   TABLO(*);
4   DUGUM1, DUGUM2   GÖSTERGE ;
5   KOMUTLAR ;
6   EĞER TABLO(DUGUM1).SAYAC > TABLO(DUGUM2).SAYAC
7     İSE [ TABLO(DUGUM2).İŞARET := 1 ;
8           TABLO(DUGUM2).BABAKÜME := DUGUM1 ;
           /* Daha az ögeli olan grup büyük gruba bir bağlaç fazla ile
              bağlanıyor, tersi yapılısaydı arama zamanı artacaktır   */
9           TABLO(DUGUM1).SAYAC := TABLO(DUGUM1).SAYAC +
              TABLO(DUGUM2).SAYAC ;
10       ]
11   DEĞİLSE [ TABLO(DUGUM1).İŞARET := 1 ;
12             TABLO(DUGUM1).BABAKÜME := DUGUM2 ;

```



```

13      6. SONUC      TABLO(DUGUM2).SAYAC := TABLO(DUGUM2).SAYAC +
                                TABLO(DUGUM1).SAYAC ;

```

```

14      ]
15 BİRLEŞTİR BİTTİ;

```

Bönmüde, bldmge hırlı hale gelen yagantımların sonuou olarak
Kısmi olarak bldmge hırlı çözümler, son derece önemli hale
gelmiştir. Bu hırlı problem çözümleri ise ancak bilgisayar
yardımı ile yapılabilir. Yagantımlar bu derece giren, ilgili
işlem konusunun gelişimiyle beraber daha etkin, daha ekonomik ve
hızlı bilgisayar kullanımı gerekmektedir.

Bu hırlı ve etkin kullanımı sağlamak ise bilgisayarı başarılı
çözümlerle en kısa zamanda ve en etkin şekilde kullanmak
sistemler ve sistem yapıları ile kullanılabilmektedir. Bu sistem
yapıları ise veri yapılarının analizi ve bu yapıların uygun
verilerde uygun şekilde kullanılması konusunda ortaya çıkmaktadır.

Bunun sonucu olarak da veri yapıları konusunda önemli ortaya
çıkmaktadır. Kendi başına çalışan bir amatör kullanıcı veya
amatör programcı için veri yapıları konusunda önemli bir işlet-
meler ve sorunların boyutu, boyutları, sorunları incelenerek
bu sorunlara uygun yapılar geliştirilerek haline gelmektedir.
Örneğin 30 x 30 boyutlarında bir matris yapısı kullanan bir
amatör programcı tarafından matris yapısı normal matris yapısı
olarak algılanırken, T. M. Y. Devlet Geniraylları gibi
500 x 500 veya 1000 x 1000 boyutlarında matris kullanan
işletmelerde işletme etkinliğini artırarak, büyük kazanç
sağlamak amacıyla özel matris türleri incelenerek uygun yapının
seçimine gidilir.

6. SONUÇ

Günümüzde oldukça hızlı hale gelen yaşantımızın sonucu olarak karşılaşılan problemlere hızlı çözüm aramak son derece önemli hale gelmiştir. Bu hızlı problem çözümleri ise ancak bilgisayar yardımı ile yapılabilir. Yaşantımıza bu derece giren, bilgi işlem konusunun gelişimiyle beraber daha etkin, daha ekonomik ve hızlı bilgisayar kullanımı gerekmektedir.

Bu hızlı ve etkin kullanımı sağlamak ise bilgisayarı karşılaşılan problemlere en kısa sürede ve en etkin biçimde uyarlayacak sistemler ve sistem yazılımları ile sağlanabilmektedir. Bu sistem yazılımları ise veri yapılarının analizi ve bu yapıların uygun yerlerde uygun şekillerde kullanımı konusunu ortaya çıkarmaktadır.

Bunun sonucu olarak da veri yapıları konusunun önemi ortaya çıkmaktadır. Kendi çapında çalışan bir amatör kullanıcı yada amatör programcı için veri yapıları konusu önem taşımasa da işletmelerin ve sorunların boyutları büyüdükçe sorunlar incelenerek bu sorunlara uygun yapılar geliştirmek zorunlu hale gelmektedir. Örneğin 30 x 30 boyutlarında bir matris yapısı kullanan bir amatör programcı tarafından matris yapısı normal matris yapısı olarak algılanırken, T. H. Y. , Devlet Demiryolları gibi 500 x 500 veya 1000 x 1000 boyutlarında matris kullanılan işletmelerde işletim etkinliğini arttırmak, bellek kazancı sağlamak amacıyla özel matris türleri incelenerek uygun yapının seçimine gidilir.

Bunun yanında bu yapılar mantıksal olarak oluşturulduktan sonra , kullanımını sağlamak için üst düzey bir dile dönüştürmek ihtiyacı ortaya çıktığı andaki geçiş zorluğunu ortadan kaldıracak bir algoritma diline ihtiyaç vardır. Yabancı ülkelerde bu olay daha önce oluşturulmasına rağmen Türkçe' ye uygun bir algoritma dili ilk olarak Hacettepe Üniversitesi'nde oluşturulan " ANLAT " algoritma dili ile doldurulmaya çalışılmaktadır. Anlat algoritma dilinin Türkçe oluşu anlaşılma kolaylığı getirdiği ve üst düzey programlama dillerine geçişte büyük kolaylık, sağladığı için bilgi işlem alanında oldukça önemlidir. Anlat algoritma dilinin ortaya çıkışı nedeni ile daha önceleri verimsizce kullandığımız veri yapılarının kullanım hızları, etkinlikleri ve bellek kazançları artmıştır. Bu gelişim mikro alanda önemli gözükmesede nüfus idareleri, Sosyal Sigortalar Kurumu, T.C.D.D ve T.H.Y. gibi olaylara makro düzeyde yaklaşan kurumlarda sağlanacak küçük bir kazanç makro düzeyde büyük önem taşıyacaktır. Ayrıca sorunların üzerine gidilerek geliştirilen veri yapılarına dayalı geliştirilen sistemler problem çözümünde çok daha etkin olmaktadır. Örneğin taşıma şirketleri için çizge veri yapısının uygulanması onların karşılaştıkları problemler karşısında daha çabuk ve sağlıklı karar vermelerini sağlayacaktır.

Bunların dışında günümüzde her alanda karşılaşılan sıralama problemleri için ise değişik problemlere uygun değişik özellikleri olan algoritmalar geliştirilmiştir. Örneğin gerçek yerlerinden çok uzak yerlere dağılmış değerleri bulunan bir sıralama probleminde SEÇSIRALA algoritması etkin olmazken ÇABUK SIRALA algoritma-

si son derece etkin olabilmektedir. Bu arada mukerrer verilerin sıkça kullanıldığı yerlerde ise yığıt veri yapısı önerilebilir.

Bu örneklerden başka bilgi işlem merkezlerinde işlem zamanlarının çoğunu arama ve sıralama işlemleri oluşturmaktadır. Bu atıl zamanların etkin hale getirilmesi ise etkin algoritmalar yanında etkin bir arama tekniği ile sağlanabilmektedir. Bunun yanında aramanın kolaylaştırılması için veriler yerlerine yerleştirilirken uygun anahtarlama fonksiyonları ile yerleştirilmelidirler. Bunun sonucu olarakta aranan veriye kısa sürede ulaşılabilecektir.

Günümüzde farkında olmadan dahi kullandığımız bir veri yapısında liste veri yapılarıdır. Bu veri yapısı mikro bilgisayarlar da uygulanabildiği gibi büyük işletim sistemlerinde kullanıcılara bellek atanması problemine çözüm sağlamak içinde kullanılmaktadır. Örneğin Çift bağlı döngülü liste veri yapısı büyük işlerim sistemlerinden " UNIX "de başarıyla kullanılmaktadır.

Sonuç olarak ise yaptığım bu çalışmada, işlediğim bütün konuların amacı bilgisayar işletimini hızlandırmak, etkin ve hatasız hale getirerek bellek kazancı sağlamaya yöneliktir. Bu çalışmamın sonucu olarak elde ettiğim tüm veri yapıları bilgileri ileride üzerinde çalışmayı çok arzu ettiğim veri tabanı (Data Base) konusuna temel teşkil etmektedir. Amacım oluşturmaya çalıştığım bu temel bilgilerin üzerine Doktora çalışması ile (Data Base) Veri Tabanı konusu üzerinde uzmanlaşabilmektir.

KAYNAKLAR:

KARAKAŞ M. ÜMİT

Bilgisayar Öğrenme Sisteminin Tasarımı: H. Ü. B.B.M. : Mayıs
1979 y. 1973

KARAKAŞ M. ÜMİT ve AKSAYAN M.

ANLAT Türkçe algoritma dili tanımı : H.Ü. B.B.M Yayınları
Mart 1983, ANKARA

KORU NACI

KARAKAŞ M. ÜMİT

Bilgisayar yazılımında Veri yapıları ve Algoritmalar : H. Ü .
B.B.M. Eylül 1987, ANKARA

Altprogramlar ve Deyişimlilik : H.Ü. B.B.M. : Eylül 1987, ANKARA

AYDIN EMİN.D

Veri Tabanı Data Base : Evrim Basım yayım Dağıtım : Eylül 1990
İstanbul

Basic ile programlama : İstanbul : 1980

TÖRECİ ERSİN

Bilgisayar programlama ve COBOL : 1983

KNUTH DONALD E.

The Art of Computer Programming : Fundamental Algorithms : Addison
Wesley, 1968

KNUTH DONALD E.

The art of Computer programming : Sorting Searching : Addison
Wesley 1973

BAŞLIGIL HÜSEYİN ve ERKOLLAR A.

Proje Yönetimi : Yıldız Üniversitesi: İstanbul : 1990

KORU NACI

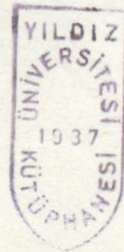
dBASE III Plus ile Programlama : Ankara : Ekim 1990

TANIŞLAR ERDİNÇ

Altprogramlar ve Depoorganizasyonu : Yıldız Üniversitesi Endüstri
Bölümü Yılıçi Projesi : İstanbul 1989

ÖZEL GALİP

Basic ile programlama : İstanbul 1989



Ö Z G E Ç M İ Ş

1965 yılında Sakarya' da doğdum. İlk ve ortaokul tahsilimi Adapazarı'nda bitirdim . Daha sonra Adapazarı Endüstri Meslek lisesi Elektrik bölümünden mezun oldum . 1984 Ö.S.Y. sınavında İ.T.Ü Sakarya Mühendislik Fakültesi Endüstri bölümüne yerleştirildim. 1988 yılında bu okuldan mezun oldum. 1989 yılında Yıldız Üniversitesi' nde Yüksek Lisans eğitimine başladım. Halen dalımda Akademik kariyer yapmak amacıyla Doktora sınavlarına hazırlanıyorum.

Y. T. Ö.

KÜTÜPHANE DOK. DAL. BAŞKANLIĞI

