Veri Yapıları

Oğuzhan ÖZTAŞ
İstanbul Üniversitesi
Bilgisayar Mühendisliği Bölümü

İÇERİK

Veri nedir?
Veri Yapısı nedi?
Temel Veri Yapıları
Sıralama Algoritmaları
Çizge(Graph)
İndeks yapıları

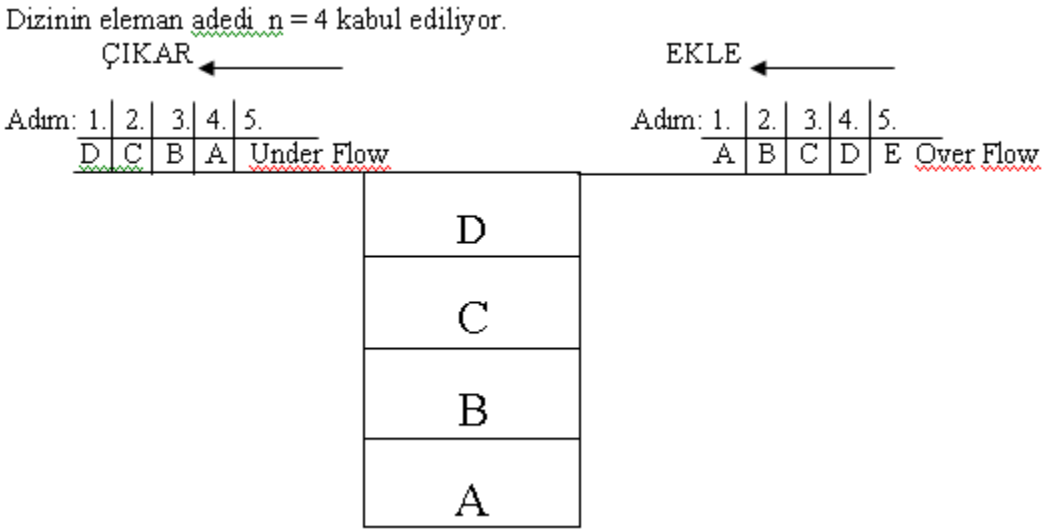
Veri : Bilgisayara dışarıdan çeşitli donanımsal araçlarla(klavye, mouse, parmakizi okuyucu, kamera, ses kaydedici, … vb.) girilen her türlü bilgi veri dir.

Veri Yapısı : Dışarıdan bilgisayara girilen verilerin hafızada tutuluş şekline(modeline) veri yapısı denir. Uygulamada çok farklı veri yapısı modelleri vardır. Bunlardan biri amaca yönelik olarak seçilir.

Temel Veri Yapıları

Yığıt(Stack)
Kuyruk (Queue)
Liste(List)
Ağaç(tree)

Yığıt(Stack)


Dizinin eleman adedi n = 4 kabul ediliyor.
ÇIKAR
EKLE
Adım: 1. 2. 3. 4. 5.
D C B A Under Flow
Adım: 1. 2. 3. 4. 5.
A B C D E Over Flow
D
C
B
A

```
#define STACKSIZE 100
#define INT     1
#define FLOAT   2
#define STRING  3
struct stackelement {
   int etype;  /* etype equals INT, FLOAT, or STRING   */
               /*  depending on the type of the        */
               /*     corresponding element.           */
   union {
     int   ival;
     float fval;
     char *pval;  /* pointer to a string */
   } element;
};
struct stack {
   int top;
   struct stackelement items[STACKSIZE];
};
```

```
struct stack s;

struct stackelement se;
        . . .
se = s.items[s.top];
switch (se.etype)  {
   case INTGR  : printf("% d\n", se.ival);
   case FLT    : printf("% f\n", se.fval);
   case STRING : printf("% s\n", se.pval);
}  /*end switch  */
```

```
int empty(struct stack *ps)
{
  if (ps->top == -1)
    return(TRUE);
  else
    return(FALSE);
}  /* end empty */
```

```
if (empty (&s))
   /* stack is empty */
else
   /* stack is not empty */
```

Yığıttan eleman çıkarma

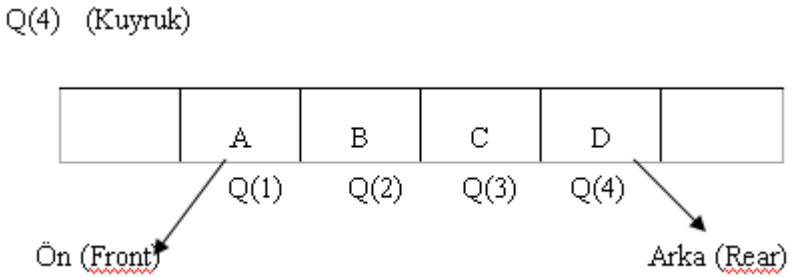
```
int pop(struct stack *ps)
{
  if (empty(ps)) {
      printf("%s", "stack underflow");
      exit(1);
  }  /* end if */
  return(ps->items[ps->top--]);
}  /* end pop */
```

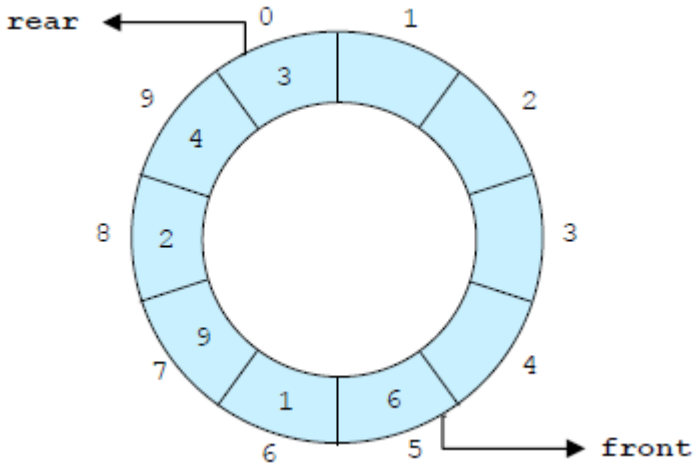
```
x = pop(s);
push (s,x);
```

Yığıta eleman ekleme

```
void push(struct stack *ps, int x)
{
   if (ps->top == STACKSIZE-1) {
      printf("%s", "stack overflow");
      exit(1);
   }
   else
      ps->items[++(ps->top)] = x;
   return;
} /* end push */
```

Kuyruk(Queue)





Kuyruğun hafıza modeli

```
#define  MAXQUEUE 100
struct queue {
   int items[MAXQUEUE];
   int front, rear;
};
struct queue q;
q.front = q.rear = MAXQUEUE-1;
```

```
int empty(struct queue *pq)
{
   return ((pq->front == pq->rear) ? TRUE : FALSE);
} /* end empty */
```

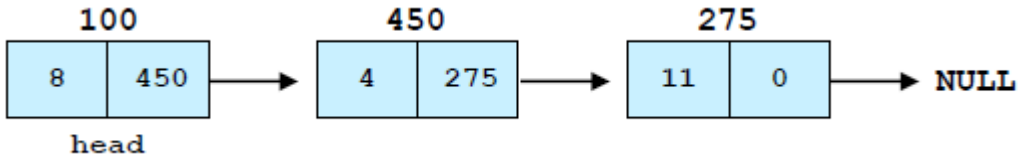
Kuyruktan eleman çıkarma

```
int remove(struct queue *pq)
{
   if (empty(pq)) {
       printf("queue underflow");
       exit(1);
   } /* end if */
   if (pq->front == MAXQUEUE-1)
       pq->front = 0;
   else
      (pq->front)++;
   return (pq->items[pq->front]);
} /* end remove */
```

Kuyruğa eleman ekleme

```
void insert(struct queue *pq, int x)
{
   /* make room for new element */
   if (pq->rear == MAXQUEUE-1)
      pq->rear = 0;
   else
      (pq->rear)++;
   /* check for overflow */
   if (pq->rear == pq->front) {
      printf("queue overflow");
      exit(1);
   } /* end if */
   pq->items[pq->rear] = x;
   return;
} /* end insert */
```

Liste(List)



Tek bağlı listenin hafıza modeli

```
#define NUMNODES 500
struct nodetype  {
   int info, next;
};
struct nodetype node[NUMNODES];
```

```
struct node  {
     int info;
     struct node *next;
};
typedef struct node *NODEPTR;
```

Hafızada bir düğüm oluşturma

```
NODEPTR getnode(void)
{
   NODEPTR p;
   p = (NODEPTR) malloc(sizeof(struct node));
   return(p);
}
```

```
void freenode(NODEPTR p)
{
   free(p);
}
```

Listeye eleman ekleme

```
void insafter(NODEPTR p, int x)
{
   NODEPTR q;
   if (p == NULL) {
      printf("void insertion\n");
      exit(1);
   }
   q = getnode();
   q -> info = x;
   q -> next = p -> next;
   p -> next = q;
} /*  end insafter */
```

Listeden eleman silme

```
void delafter(NODEPTR p, int *px)
{
   NODEPTR q;
   if ((p == NULL) || (p -> next == NULL))  {
      printf("void deletion\n");
      exit(1);
   }
   q = p -> next;
   *px = q -> info;
   p -> next = q -> next;
   freenode(q);
} /* end delafter */
```

Bir x elemanını bir liste içinde arama

```
NODEPTR search(NODEPTR list, int x)
{
   NODEPTR p;
   for (p = list; p != NULL; p = p->next)
      if (p->info == x)
         return (p);
   /* x is not in the list */
   return (NULL);
} /* end search */
```

Çift bağlı listeler:

Array Implementation

```
struct nodetype  {
   int info;
   int left, right;
 } ;
struct nodetype node[NUMNODES];
```

Dynamic Implementation

```
struct node {
   int info;
   struct node *left, *right;
 } ;
typedef struct node *NODEPTR;
```

Çift bağlı listeden eleman çıkarma

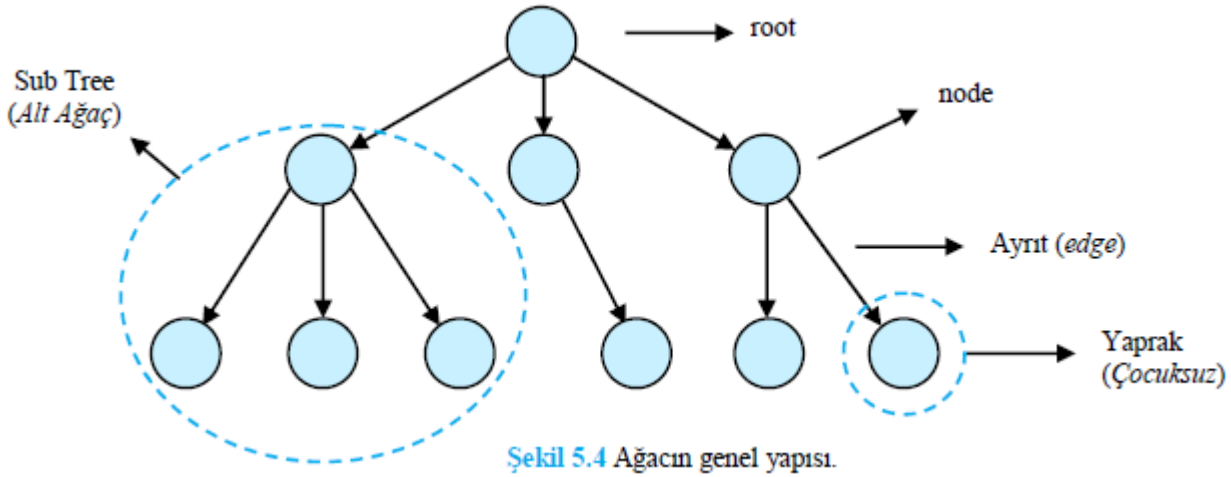
```
void delete(NODEPTR p, int *px)
{
   NODEPTR q, r;
   if (p == NULL)  {
      printf("void deletion\n");
      return;
   } /* end if */

   *px = p->info;
   q = p->left;
   r = p->right;
   q->right = r;
   r->left = q;
   freenode(p);
   return;
} /* end delete */
```

Çift bağlı listeye eleman ekleme

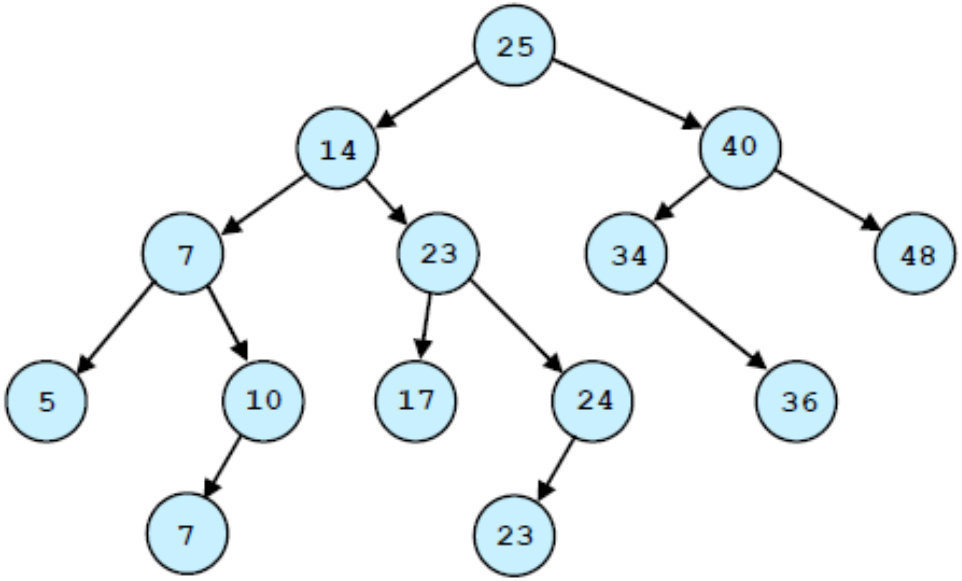
```
void insertright(NODEPTR p, int x)
{
   NODEPTR q, r;
   if (p == NULL)  {
      printf("void insertion\n");
      return;
   } /* end if */
   q = getnode();
   q->info = x;
   r = p->right;
   r->left = q;
   q->right = r;
   q->left = p;
   p->right = q;
   return;
} /* end insertright */
```

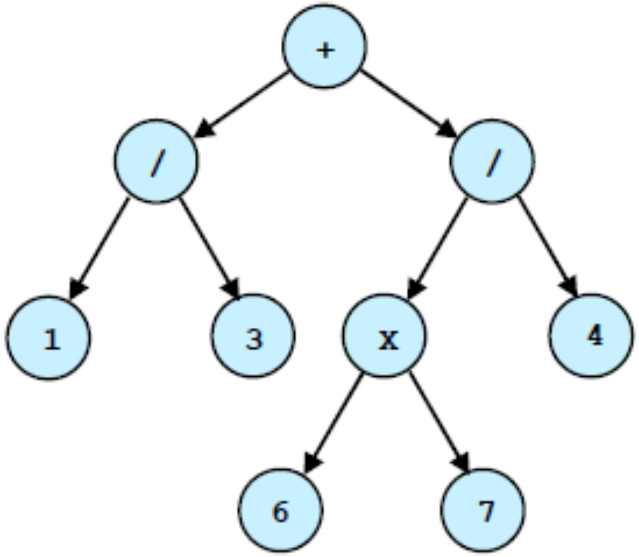
Ağaç(Tree)

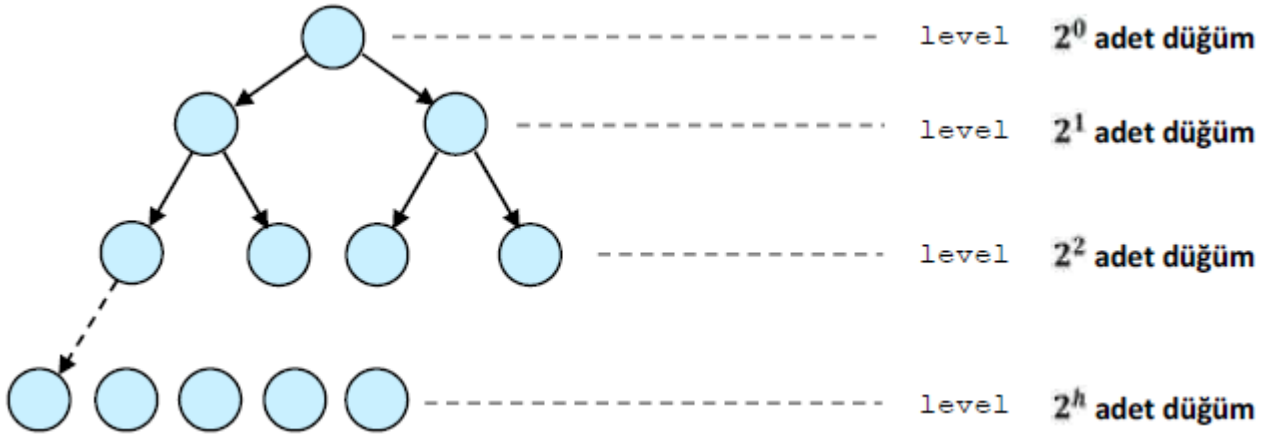


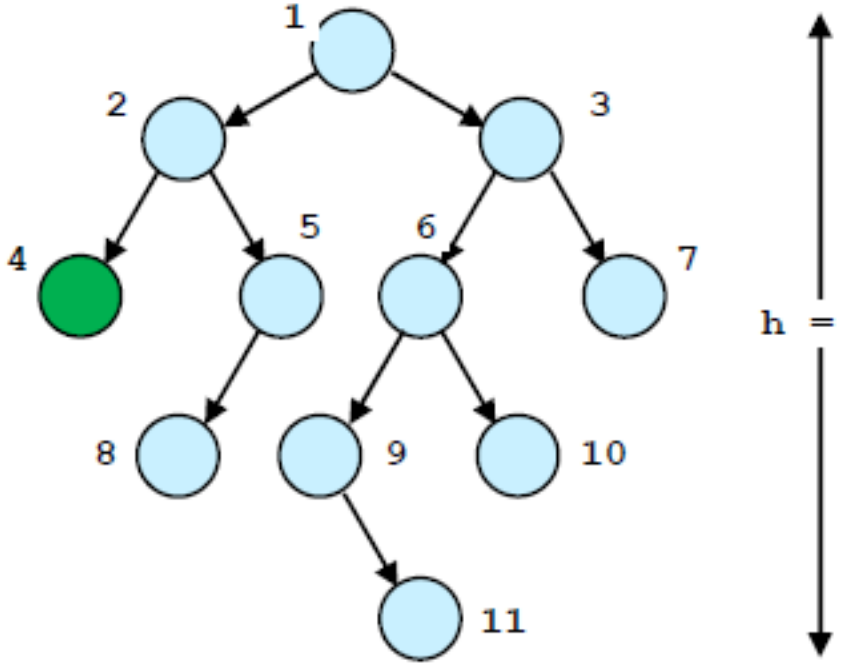
Şekil 5.4 Ağacın genel yapısı.

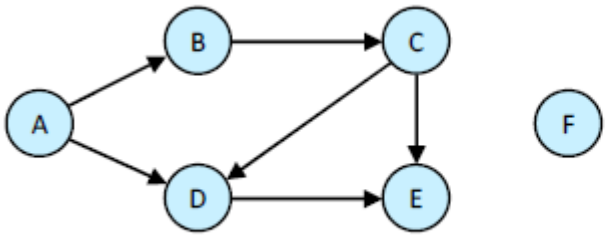
Ağaçlar





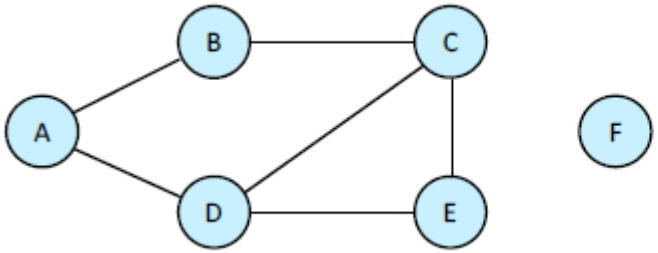

level 2^0 adet düğüm
level 2^1 adet düğüm
level 2^2 adet düğüm
level 2^h adet düğüm


1
2
3
4
5
6
7
8
9
10
11
h =

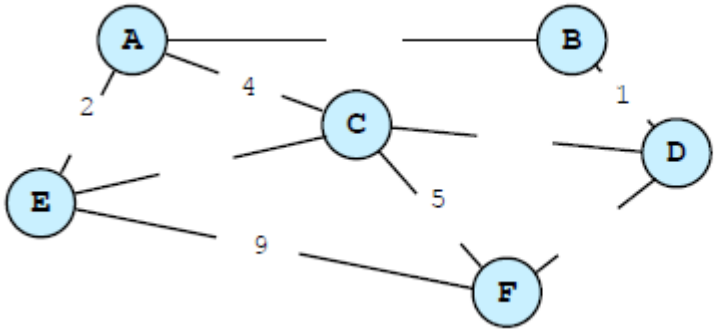


	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	1	1	0
D	0	0	0	0	1	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

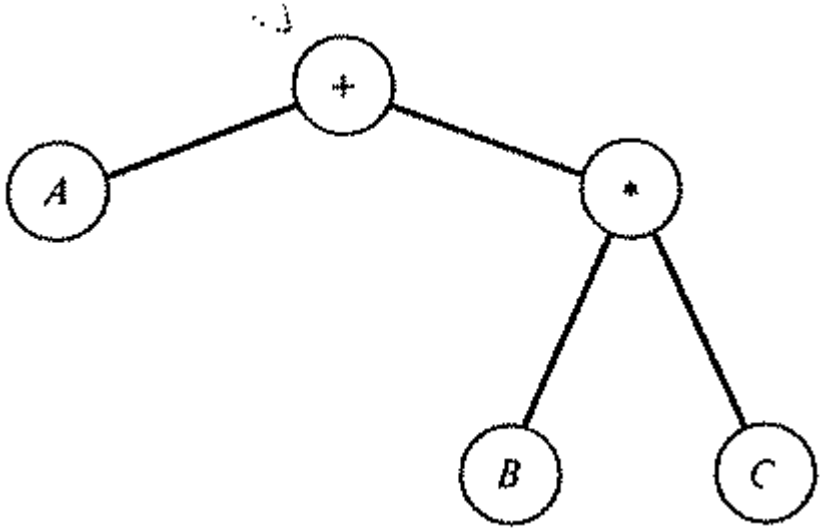
Orijin ↓ → Hedef



	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	1	1	0
D	1	0	1	0	1	0
E	0	0	1	1	0	0
F	0	0	0	0	0	0



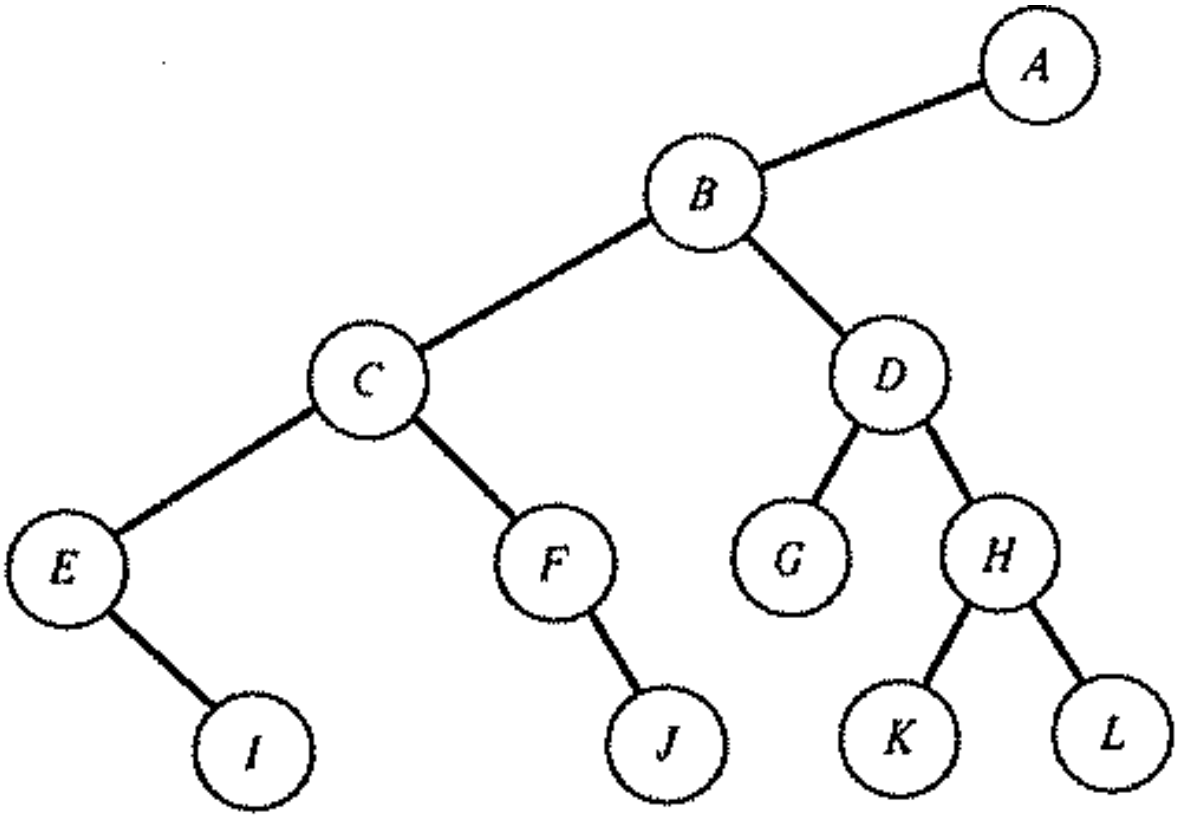
	A	B	C	D	E	F
A	∞	8	4	∞	2	∞
B	8	∞	∞	1	∞	∞
C	4	∞	∞	7	6	5
D	∞	1	7	∞	∞	3
E	2	∞	6	∞	∞	9
F	∞	∞	5	3	9	∞



DLR= $+A*BC$

LDR= $A+B*C$

LRD= $ABC*+$


A
B
C
D
E
F
G
H
I
J
K
L

Preorder: *ABCEIFJDGHKL*
Inorder: *EICFJBGDKHLA*
Postorder: *IEJFCGKLHDBA*

İkili ağacın hafıza modeli

```
#define NUMNODES 500
struct nodetype {
   int info;
   int left;
   int right;
   int father;
};
struct nodetype node[NUMNODES];
```

```
struct nodetype {
   int info;
   struct nodetype *left;
   struct nodetype *right;
   struct nodetype *father;
};
typedef struct nodetype *NODEPTR;
```

```
NODEPTR maketree(int x)
{
   NODEPTR p;

   p = getnode();
   p->info = x;
   p->left = NULL;
   p->right = NULL;
   return(p);
} /* end maketree */
```

```
void setleft(NODEPTR p, int x)
{
   if (p == NULL)
      printf("void insertion\n");
   else if (p->left != NULL)
      printf ("invalid insertion\n");
   else
      p->left = maketree(x);
} /* end setleft */
```

Ekrandan girilen sayıları bir ikili ağaca yerleştirme

```
struct nodetype {
   int info;
   struct nodetype *left;
   struct nodetype *right;
};

typedef struct nodetype *NODEPTR;

main()
{
NODEPTR ptree;
NODEPTR p, q;
int number;

scanf("%d", &number);
ptree = maketree(number);
while (scanf("%d", &number) != EOF) {
   p = q = ptree;
   while (number != p->info && q != NULL) {
      p = q;
      if (number < p->info)
         q = p->left;
      else
         q = p->right;
   }  /* end while */
   if (number == p->info)
      printf("%d is a duplicate\n", number);
   else if (number < p->info)
      setleft(p, number);
   else
      setright(p, number);
   }  /* end while */
}  /* end main */
```

```
void pretrav(NODEPTR tree)
{
   if (tree != NULL) {
      printf("%d\n", tree->info);      /* visit the root          */
      pretrav(tree->left);             /* traverse left subtree   */
      pretrav(tree->right);            /* traverse right subtree  */
   }  /* end if */
}  /* end pretrav */

void intrav(NODEPTR tree)
{
   if (tree != NULL) {
      intrav(tree->left);              /* traverse left subtree   */
      printf("%d\n", tree->info);      /* visit the root          */
      intrav(tree->right);             /* traverse right subtree  */
   }  /* end if */
}  /* end intrav */

void posttrav(NODEPTR tree)
{
   if (tree != NULL) {
      posttrav(tree->left);            /* traverse left subtree   */
      posttrav(tree->right);           /* traverse right subtree  */
      printf("%d\n", tree->info);      /* visit the root          */
   }  /* end if */
}  /* end posttrav */
```

DLR

LDR

LRD