Veri Modelleri

Ağaç Veri Modeli

Ağaç Veri Modeli

- Verilerin birbirine sanki bir ağaç yapısı oluşturuyormuş gibi sanal olarak bağlanmasıyla elde edilen bir veri modelidir.
- Ağaç veri modeli daha fazla bellek alanına gereksinim duyar.
- Ağaç veri modelinde, bir kök işaretçisi, sonlu sayıda düğümleri ve onları birbirine bağlayan dalları vardır.

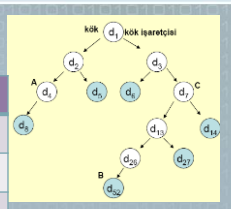
Ağaç Veri Modeli

- Veri ağacın düğümlerinde tutulur. Dallarda ise geçiş koşulları vardır.
- Her ağacın bir kök işaretçisi vardır. Ağaca henüz bir düğüm eklenmemiş ise ağaç boştur ve kök işaretçisi NULL değerini gösterir. Ağaç bu kök etrafında dallanır ve genişler.

Veri	Çocuk	Kardeş

Ağaç Veri Modeli

	Düğümler			
Tanım	**kök**	**A**	**B**	**C**
Derece/çocuk	2	1	0	2
Kardeş düğüm	1	2	1	2
Aile	yok	d2	d26	d3
Ata	yok	Kök(d1)	d13,d7,d3,kök	Kök(d1)
Yol (kök düğümünden)	d1	d1,d2,d4	d1,d3,d7,d13,d26,d52	d1,d3,d7
Derinlik/seviye	1	3	6	3
Yükseklik	6	2	1	4



Ağaç türleri

- İkili arama ağaç: Bir düğüm en fazla iki tane çocuğa sahip olabilir ve alt çocuk bağlantıları belirli bir sırada yapılır.
- Kodlama ağacı: Bir alfabedeki veya daha genel olarak bir kümedeki karakterlere kod atanması için kurulan ağaç şeklidir.

Ağaç türleri

- Sözlük ağacı: Bir sözlükte bulunan sözcüklerin tutulması için kurulan ağaç şeklidir. Arama işleminin performanslı bir şekilde yapılmasını ve belleğin optimum şekilde kullanılmasını sağlar. Sözlük ağaçları bir araya getirilerek sözlük ormanı oluşturur. Bu ormanda sözlükteki harf sayısı kadar ağaç vardır.
- Kümeleme ağacı: Kümeleme bir çeşit sıralama ağacıdır.

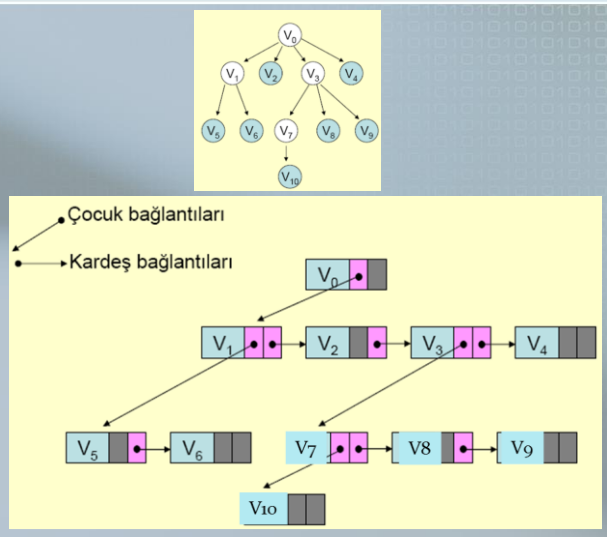
Ağaç üzerinde yapılan işlemler

- *Ağacı ilk oluşturma*
- *Ağacı dolaşma*
- *Düğüm ekleme*
- *Düğüm arama*
- *Düğüm silme*
- *Düğümleri listeleme*
- *Düğümleri saklama/yükleme*

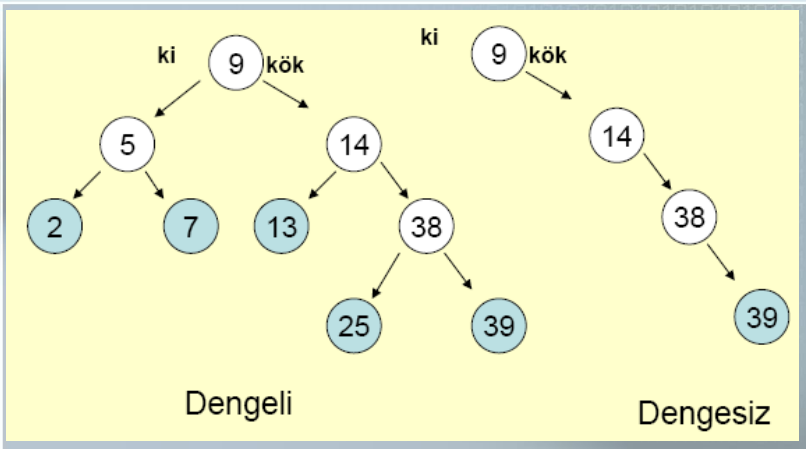
Ağaç kurulması ve veri yapısı

```
struct isaretcili{
        int veri;
        struct isaretcili*cocuk;
        struct isaretcili*kardes;
};
```

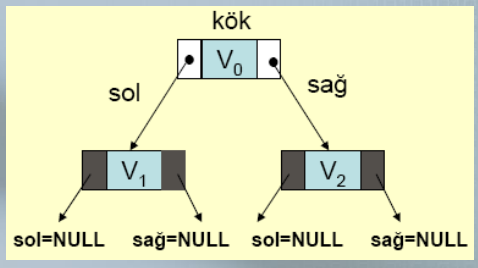
Ağaç kurulması ve veri yapısı



İkili Ağaç

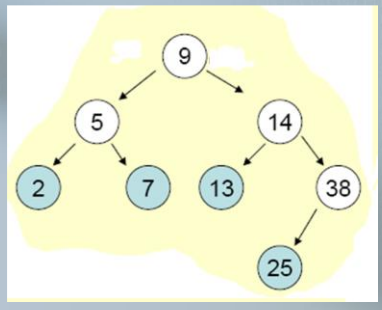


İkili ağaç veri yapısı



İkili ağaç veri yapısı

Kökten büyük değerler kökün sağına küçük değerler soluna yerleştirilir.



İkili ağaç oluşturma

C dili

```
struct agac{
    Dugumptr kok;
}
typedef struct agac Agac;
typedef Agac* Agacptr;
Agacptr yeni_agac(){
    Agacptr agac;
    agac=malloc(sizeof(Agac));
    agac>kok = NULL;
    return agac;
}
```

Java dili

```
public class Agac{
    Dugum kok;
public Agac(){
    kok = NULL;
  }
}
```


İkili ağaçta düğüm oluşturma

C dili

```
struct dugum {
    int icerik;
    struct dugum* sol;
    struct dugum* sag;
}
typedef struct dugum Dugum;
typedef Dugum* Dugumptr;
Dugumptr yeni_dugum(int icerik){
    Dugumptr dugum;
    dugum=malloc(sizeof(Dugum));
    dugum->icerik = icerik;
    dugum->sol = NULL;
    dugum->sag = NULL;
    return dugum;
}
```

Java dili

```
public class Dugum {
    int icerik;
    dugum sol;
    dugum sag;
public Dugum(int icerik){
    this.icerik = icerik;
    sol = NULL;
    sag = NULL;
    }
}
```

İkili ağaçta arama

C dili (özyinelemeli);

```
Dugumptr agac_ara(Dugumptr d, int eleman) {
   if (!d)
     return NULL;
   if (d->icerik==eleman)
     return d;
   else
     if (d->icerik>eleman)
        return agaç_ara(d->sol, eleman);
     else
        return agac_ara(d->sag, eleman);
}
```

Maliyet O(logn)

İkili ağaçta arama

Java dili (özyinelemeli);

```
Dugum agacAra(int eleman) {
   if (icerik==eleman)
     return this;
   else
     if (icerik>eleman)
        if (sol!= null);
           return sol.agacAra(eleman);
     else
        if (sag!=null)
           return sag.agacAra(eleman);
        else
           return null;
}
```

Maliyet O(logn)

İkili ağaçta arama

C dili (özyinelemesiz);

```
Dugumptr agac_ara(Agacptr a, int eleman) {
Dugumptr d;
d=a->kok;
    while (d!=NULL){
    if (d->icerik==eleman)
      return d;
    else
      if (d->icerik>eleman)
          d = d->sol;
      else
          d = d->sag;
          }
      return NULL;
}
```

Maliyet O(logn)

İkili ağaçta arama

Java dili (özyinelemesiz);

```
Dugum agacAra(int eleman) {
   Dugum d;
   d = kok;
   while (d! = null) {
    if (icerik==eleman)
      return d;
    else
      if (d.icerik>eleman)
          d = d.sol;
      else
          d= d.sag;
   }
   return null;
}
```

Maliyet O(logn)

En küçük elemanı bulma

C dili (özyinelemesiz);

```
Dugumptr asgari_ara(Dugumptr d) {
    Dugumptr sonuc = d;
    while (sonuc->sol)
      sonuc = sonuc->sol;
    return sonuc;
}
```

Java dili (özyinelemesiz);

```
Dugum asgariAra() {
    Dugum sonuc = this;
    while (sonuc.sol! = null)
      sonuc = sonuc.sol;
    return sonuc;
}
```

Maliyet O(logn)

En küçük elemanı bulma

C dili (özyinelemeli);

```
Dugumptr asgari_ara(Dugumptr d) {
  if (d->sol==NULL)
      return d;
  else
      return asgari_ara(d->sol);
}
```

Java dili (özyinelemeli);

```
Dugum asgariAra() {
   if (d->sol==NULL)
       return this;
    else
       return sol.asgariAra();
}
```

Maliyet O(logn)

En büyük elemanı bulma

C dili (özyinelemesiz);

```
Dugumptr azami_ara(Dugumptr d) {
   Dugumptr sonuc = d;
   while (sonuc->sag)
     sonuc = sonuc->sag;
   return sonuc;
}
```

Java dili (özyinelemesiz);

```
Dugum azamiAra() {
   Dugum sonuc = this;
   while (sonuc.sag! = null)
     sonuc = sonuc.sag;
   return sonuc;
}
```

Maliyet O(logn)

En büyük elemanı bulma

C dili (özyinelemeli);

```
Dugumptr azami_ara(Dugumptr d) {
   if (d->sag==NULL)
      return d;
   else
      return azami_ara(d->sag);
}
```

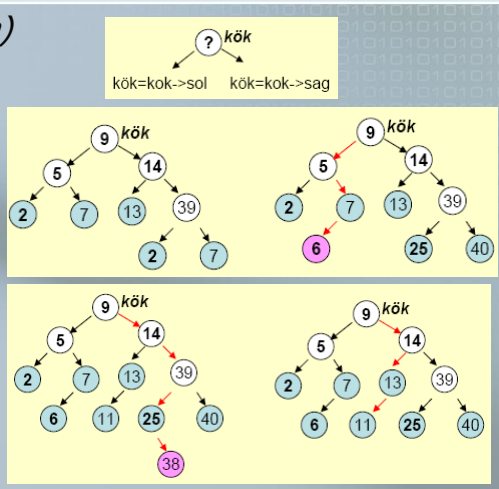
Java dili (özyinelemeli);

```
Dugum azamiAra() {
   if (d->sag==NULL)
      return this;
   else
      return sag.azamiAra();
}
```

Maliyet O(logn)

İkili ağaca düğüm ekleme

if(kök boş ağacı gösteriyorsa)
düğümü köke ekle;
else{
if(eklenen kökten küçükse)
sol alt ağaca dallan;
basa dön;
else sağ altağacadallan;
basa dön;
}



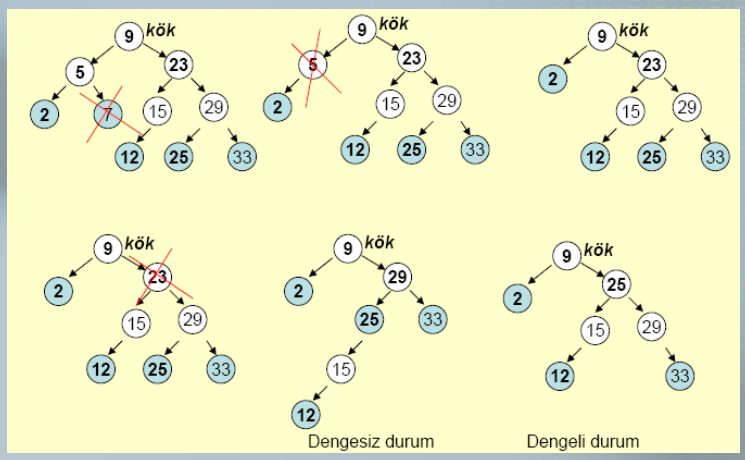
Ağaca düğüm ekleme (C dili)

```
void agaca_ekle(Agacptr a, Dugumptr yeni){
        Dugumptr y = NULL;
        Dugumptr a = a ->kok;
        while (x! = NULL){
              y = x;
              if (yeni->icerik<x->icerik)
                 x = x->sol;
              else
                 x = x->sag;
              }
              if (y == NULL)
                a->kok=yeni;
             else
                 if (yeni->icerik<y->icerik)
                       y = y->sol=yeni;
                 else
                       x = x->sag=yeni;
              }
```

Ağaca düğüm ekleme (Java dili)

```
void agacaEkle(Dugum yeni){
        Dugum y = NULL;
        Dugum x = kok;
        while (x! = NULL){
              y = x;
              if (yeni.icerik<x.icerik)
                 x = x.sol;
              else
                 x = x.sag;
              }
              if (y == NULL)
                 kok==yeni;
              else
                 if (yeni.icerik<y.icerik)
                      y.sol=yeni;
                 else
                      x.sag=yeni;
              }
```


Ağaçtan düğüm silme



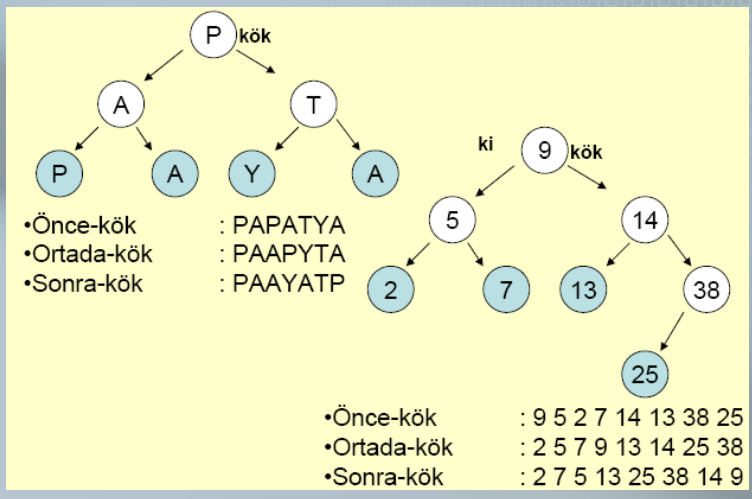
Ağaçtan düğüm silme (C dili)

```
void agac_sil(Agacptr a, int icerik){
  Dugumptr y, x=a->kok;
  while (x->icerik!=icerik){
    if (x->icerik!=icerik)
      x=x->sol;
    else
      x=x->sag;
  }
  while (1){
     y=azami_ara(x->sol);
     if (y==NULL)
       y=asgari_ara(x->sag);
     if (y==NULL)
       break;
     x->icerik=y->icerik;
     x=y;
     }
  }
```


Ağaçtan düğüm silme (Java dili)

```
void agacSil(int icerik){
  Dugum y=NULL, x=kok;
  while (x.icerik!=icerik){
    if (x.icerik!=icerik)
      x=x.sol;
    else
      x=x.sag;
  }
  while (true){
     if (x.sol !=NULL)
       y=x.sol.azamiAra();
     if (y==NULL&&x.sag!=NULL)
       y=x.sag.asgariAra();
     if (y==NULL)
       break;
     x.icerik=y.içerik;
     x=y;
     }
  }
```

İkili ağaç üzerinde dolaşma



İkili ağaç üzerinde dolaşma

Önce-kök:

C dili:

```
void once_gezinti(Dugumptr d){
  printf ("%d", d->icerik);
  if (d->sol)
    once_gezinti(d->sol);
  if (d->sag)
    once_gezinti(d->sag);
}
```

Java dili:

```
void onceGezinti(){
  System.out.print(icerik);
  if (sol!=null)
    sol.onceGezinti();
  if (sag!=null)
    sag.onceGezinti();
}
```

İkili ağaç üzerinde dolaşma

Ortada kök:

C dili:

```
void ara_gezinti(Dugumptr d){
  if (d->sol)
    ara_gezinti(d->sol);
  printf ("%d", d->icerik);
  if (d->sag)
    ara_gezinti(d->sag);
}
```

Java dili:

```
void araGezinti(){
  if (sol!=null)
    sol.araGezinti();
  System.out.print(icerik);
  if (sag!=null)
    sag.araGezinti();
}
```


İkili ağaç üzerinde dolaşma

Sonra kök:

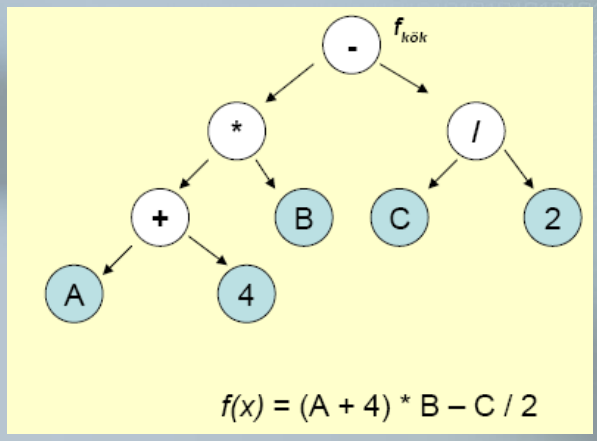
C dili:

```
void sonra_gezinti(Dugumptr d){
  if (d->sol)
    sonra_gezinti(d->sol);
  if (d->sag)
    sonra_gezinti(d->sag);
  printf ("%d", d->icerik);
}
```

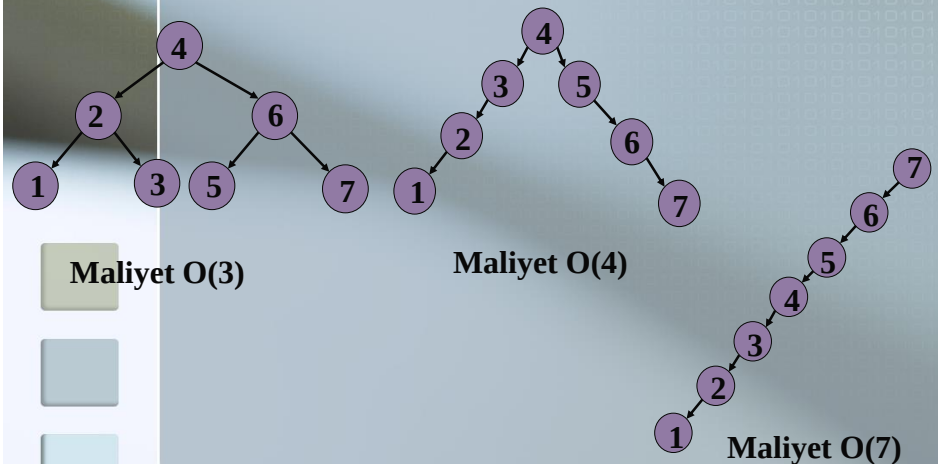
Java dili:

```
void sonraGezinti(){
  if (sol!=null)
    sol.sonraGezinti();
  if (sag!=null)
    sag.sonraGezinti();
  System.out.print(icerik);
}
```

Bağıntı ağaç yapısı

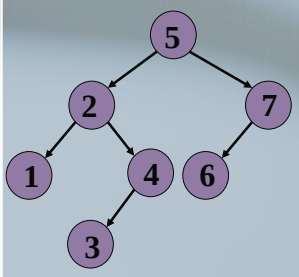


Adelson Velskii ve Landis (AVL) ağacı

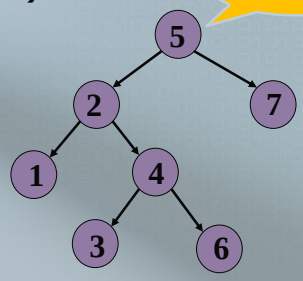


AVL ağacı

Ağaçtaki her düğüm için sağ ve sol düğüm boyları farkı en fazla bir (1) olmalıdır. (sağ-sol=<1)



Sol 3, sağ 1



AVL düğüm tanımı

C dili

```
struct avldugum{
  int icerik;
  int boy;
  struct avldugum* sol;
  struct avldugum* sag;
}
typedef struct avldugum Avldugum;
typedef Avldugum* Avldugumptr;
Avldugumptr yeni_avldugum(int icerik){
  Avldugumptr dugum;
  dugum = malloc(sizeof(Avldugum));
  dugum-> icerik = icerik;
  dugum-> sol = NULL;
  dugum-> sag = NULL;
  dugum-> boy = 1;
  return dugum;
}
```

Java dili

```
public class Avldugum{
  int icerik;
  int boy;
  Avldugum sol;
  Avldugum sag;
}
public Avldugum (int icerik){
  this.icerik = icerik;
  sol = NULL;
  sag = NULL;
  boy = 1;
  }
}
```

AVL ağaç tanımı

C dili

```
struct avlagac{
  Avldugumptr kok;
};
typedef struct avlagac Avlagac;
typedef Avlagac* Avlagacptr;
Avlagacptr yeni_avlagac(){
  Avlagacptr agac;
  agac= malloc(sizeof(Avlagac));
  agac-> kok= NULL;
  return agac;
}
int boy(Avldugumptr d){
  if (d = NULL)
     return 0;
  else
    return d->boy;
}
```

Java dili

```
Public class AvlAgac{
  Avldugum kok;
  public AvlAgac(){
    kok = null;
  }
}
int boy(AvlDugum d);
  if (d = null)
     return 0;
  else
    return d.boy;
}
```


B+ ağacı

Dengeli ağaç oluşturmanın bir diğer yolu da düğüm içinde iki adet veri yerleştirilmesidir. Bu tip veri tabanı na d-li ağaç yapısı denir. d-li ağaç yapılarına örnek olarak da B veya B+ verilebilir.

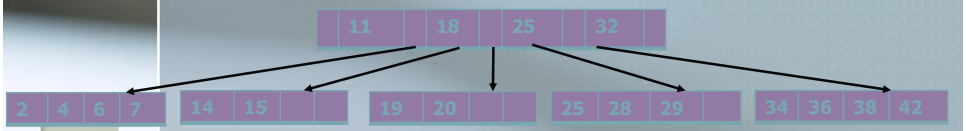
B+ ağacı

Dinamik bir arama ağacıdır. İndeks kısmı ve verilerin saklandığı kısımdan oluşmaktadır.

İndeks kısmı d-li ağaç yapısında olup her düğüm $d \leq m \leq 2d$ değer içermektedir. d değeri B+ ağacının parametresi olup, B+ ağacının kapasitesini göstermektedir ve ağacın derecesi olarak adlandırılır. Kök bu durumda istisnadır ve $1 \leq m \leq 2d$ dir.

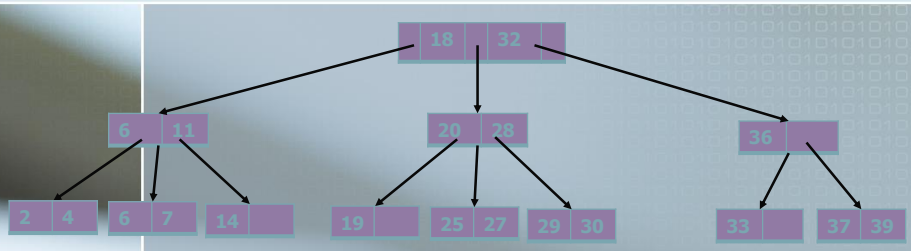
B+ ağacı

Her düğüm kendisine ait m+1 çocuk düğümü gösteren m+1 tane işaretçiye sahiptir.



15 veri içeren d=2 dereceli B+ ağacı
Düğüm sayısı $d \leq m \leq 2d$ den en az 2 en fazla 4 dür.
Kök için $1 \leq m \leq 2d$ için en fazla 4 değer ve 5 işaretçi vardır.
Alt kısımda ise K<11 için 4 veri; $11 \leq K<18$ için 2 veri;
$18 \leq K < 25$ için 2 veri; $25 \leq K < 32$ için 3 veri
Son olarak $K \geq 32$ için 4 veri olabilir.

B+ ağacı



13 veri içeren d=1 dereceli B+ ağacı
Düğüm sayısı $d \leq m \leq 2d$ den en az 1 en fazla 2 dür.
Kök için $1 \leq m \leq 2d$ için en fazla 2 değer ve 3 işaretçi vardır.

B+ ağacında düğüm tanımı

C dili

```
#define SBD sizeof(Bdugum)
  struct bdugum{
  int *K;
  int m;
  int d;
  int yaprak;
  struct bdugum* nesil;
}
typedef struct bdugum Bdugum;
typedef Bdugum* Bdugumptr;
Bdugumptr yeni_bdugum(int d){
  Bdugumptr dugum;
  dugum = malloc(SBD);
  dugum->yaprak=1;
  dugum->d=d;
  dugum->m=0;
  dugum->K=malloc((2*d+1)*sizeof(int));
  dugum->nesil=malloc((2*d+1)*SBD);
  Return dugum;
}
```

Java dili

```
public class Bdugum{
  int [] K;
  int m;
  int d;
  boolen yaprak;
  Bdugum[] nesil;
  public Bdugum(int d){
    m = 0;
    this.d=d;
    yaprak = true;
    K = new int [2*d+1];
    nesil = new Bdugum[2*d+1];
  }
}
```

B+ ağacı tanımlama

C dili

```
struct bagac{
  Bdugumptr kok;
}
typedef struct bagac Bagac;
typedef Bagac* Bagacptr;
  Bagacptr agac;
  agac = malloc (sizeof(Bagac));
  agac ->kok = NULL;
  return agac;
}
```

Java dili

```
structpublic class BAgac{
  BDugum kok;
  public Bagac(){
     kok = NULL;
  }
}
```


Ödev

AVL ve B+ ağaçlarında dolaşma, ekleme ve sıralama gibi işlemleri gerçekleşecek algoritmalar tasarlayınız.