Fejlett adatszerkezetek

Diákolimpiai Felkészítő

horvath@inf.elte.hu

1. Permutáció dekódolás

Az $1,\ldots,n$ számok minden $A = (a_1,\ldots,a_n)$ permutációja kódolható azzal a $B = (b_1,\ldots,n_n)$ számsorozattal, ahol $b_i =$ azon a_j elemek számával, amelyekre $j < i$ és $a_j > a_i$.
A b_i érték a permutáció i-edik elem inverzióinak száma. A B sorozatot a permutáció inverziós vektorának is nevezik.

1.1. feladat

Bemenet: Az $1,\ldots,n$ számok egy permutációjához tartozó B inverziós vektor.
Kimenet: Az az A permutáció, amelynek inverziós vektora B.

Példa bemenet és kimenet

bemenet

```
7
0 0 1 0 2 0 4
```

kimenet

```
1 5 2 6 4 7 3
```

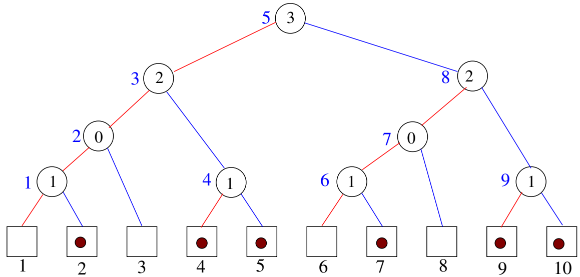
Naív ($\mathbb{O}(n^2)$ idejű) Megoldás

Jelölje *Maradt* azon elemek halmazát, amelyeknek még nem határoztuk meg a permutációbeli pozícióját.
A $Keres(k)$ művelet a *Maradt* halmaz k-adik legnagyobb elemét adja, és ki is törli a halmazból. Tehát $Keres(k)$ eredménye a *Maradt* halmaz azon leme, amelynél pontosan k-1 nálánál nagyobb van.

```
int n;
int B[maxN]; int A[maxN];
bool Maradt[maxN];
int Keres(int k){
  int e=n+1;
  while(k>0){
    e--;
    if (Maradt[e]) k--;
  }
  Maradt[e]=false;
  return e;
}
int main(){
  cin>>n;
  for (int i=1;i<=n;i++){
      cin>>B[i];
      Maradt[i]=true;
  }
  for (int i=n;i>0;i--)
    A[i]=Keres(B[i]+1);
  for (int i=1;i<n;i++)
    cout<<A[i]<<" ";
  cout<<A[n]<<endl;
   return 0;
}
```

Gyorsabb, ($\mathbb{O}(n \log n)$ idejű) Megoldás

A Keres művelet hatékony megvalósításához a $Maradt$ halmazt ábrázoljuk bináris fában az alábbi módon.



1. ábra. Az $\{2, 4, 5, 7, 9, 10\}$ maradék halmaz ábrázolása, ha $n = 10$.

1. A fának n levélpontja van.
2. A fának $n-1$ belső (nem levél) pontja van az $1, \ldots, n-1$ számokkal azonosítva.
3. Minden belső pontnak két fia van.
4. Az $a, \ldots, b$ elemeket (leveleket) tartalmazó részfa gyökere $k = \lfloor (a+b)/2 \rfloor$, bal részfája az $a..k$, jobb részfája pedig a $k+1..b$ elemeket (leveleket) tartalmazza.
5. A fa minden p belső pontja a $Maradt$ halmaz p jobbrészfájában lévő elemeinek számát tartalmazza.

```
int n;
int B[maxN]; int A[maxN];
bool Maradt[maxN];
int Hany[maxN];

int Keres(int k, int bal, int jobb){
//Global: Maradt, Hany
  if(bal==jobb){
      Maradt[bal]=false;
      return bal;
  }
  int s=(bal+jobb)/2;
  if (Hany[s]>=k){
    Hany[s]--;
    return Keres(k, s+1,jobb);
  }else{
    return Keres(k-Hany[s],bal,s);
  }
}
void InitHany(int bal, int jobb){
  if (bal<jobb){
    int s=(bal+jobb)/2;
    Hany[s]=jobb-s;
    InitHany(bal,s);
    InitHany(s+1,jobb);
  }
}
```

```
int main(int argc, char *argv[]){
  cin>>n;
  for (int i=1;i<=n;i++){
      cin>>B[i];
      Maradt[i]=true;
      Hany[i]=1;
  }
  InitHany(1,n);
  for (int i=n;i>0;i--){
    A[i]=Keres(B[i]+1, 1,n);
  }
  for (int i=1;i<n;i++)
    cout<<A[i]<<" ";
  cout<<A[n]<<endl;
   return 0;
}
```

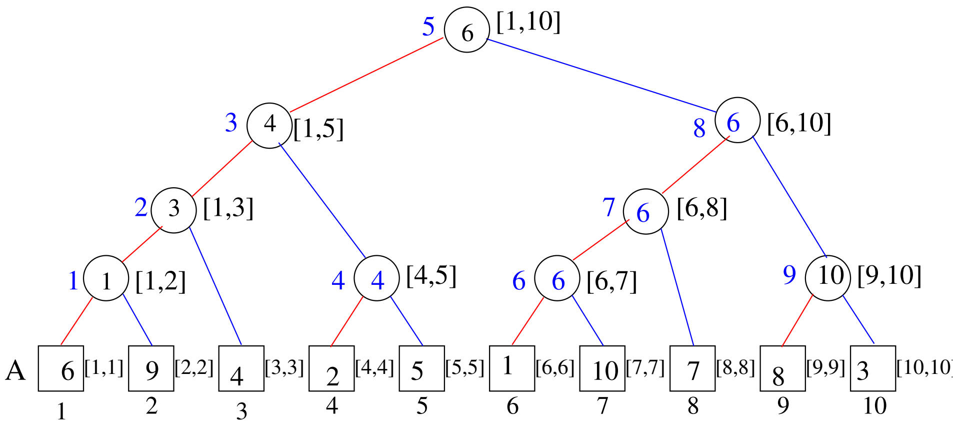
1.2. Tartományi minimum keresés (RMQ)

Specifikáció:

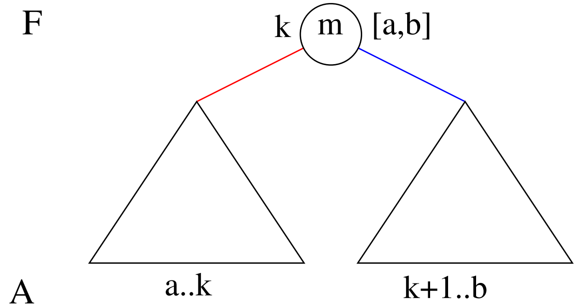
Bemenet: $A = (a_1, \ldots, a_n)$ rendezett elemtípusból vett elemek sorozata.

Kimenet: $IKeres(A, i, j)$ az $a_i, \ldots, a_j$ elemek legkisebb elemének indexe.

Megvalósítás szegmes-fával



2. ábra. Szegmens-fa az $A = (6, 9, 4, 2, 5, 1, 10, 7, 8, 3)$ sorozatra



3. ábra.

$$k = \lfloor (a+b)/2 \rfloor$$

$$F[k] = m = \min\{A[a], \ldots, A[b]\}$$

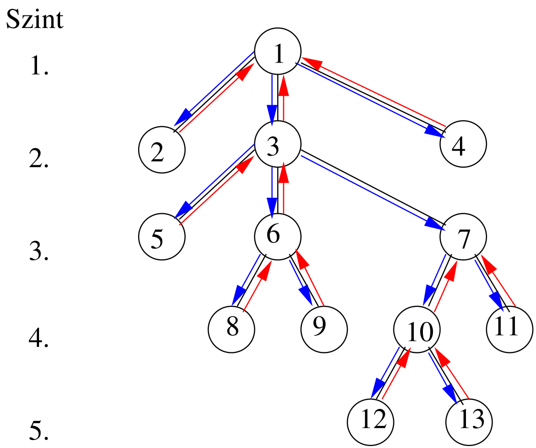
Bármely $i \leq k < j$ hármasra teljesül, hogy

$$\min(i,j) = \begin{cases} min(i,k) = m1 & \text{ha } A[m1] < A[m2] \\ \min(k+1,j) = m2 & \text{egyébként} \end{cases}$$

```
int IKeres(int A[], int F[], int a, int b, int i, int j){
//a<=i<=j<=b
   if(i==j)
      return i;
   int k=(a+b)/2;
   if(a==i && b==j)
      return F[k];
   if(j<=k) return IKeres(A,F, a,k,i,j);
   if(k<i) return IKeres(A, F, k+1,b,i,j);
   int m1=IKeres(A,F, a,k,i,k);
   int m2=IKeres(A,F, k+1,b,k+1,j);
   if (A[m1]<A[m2])
      return m1;
   else
      return m2;
}
int Elofeldolg(int A[], int F[], int bal, int jobb){
   if(bal==jobb)
      return bal;
   else{
      int k=(bal+jobb)/2;
      int m1=Elofeldolg(A,F, bal,k);
      int m2=Elofeldolg(A,F, k+1,jobb);
      if (A[m1]<A[m2])
         F[k]= m1;
      else
         F[k]= m2;
      return F[k];
   }
}
```

1.3. Legközelebbi közös ős probléma megoldása

Bemenet: $G = (V, E)$ irányítatlan gráf, amely gyökeres fa, és két pontja a, b.
Kimenet: Az r gyökértől legtávolabbi olyan p pont, amely rajta van mind az $r \rightsquigarrow a$, mind az $r \rightsquigarrow b$ úton.
Megoldás *IKeres* (szegmes-fa) alkalmazásával.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
E	1	2	1	3	5	3	6	8	6	9	6	3	7	10	12	10	13	10	7	11	7	3	1	4	1
Szint	1	2	1	2	3	2	3	4	3	4	3	2	3	4	5	4	5	4	3	4	3	2	1	2	1
Elso	1	2	4	24	5	7	13	8	10	14	20	15	17												

4. ábra.

Első lépésként mélységi bejárással állítsuk elő az $E[]$, $Szint[]$, és $Elso$ függvényeket (tömböket).

E : egy r gyökértől induló Euler-kör.

$Szint[i]$: az $E[i]$ pont gyökértől mért szintje.

$Elso[p]$: az első (legkisebb) i index, amelyre $E[i] = p$ teljesül.

Ekkor a és b legközelebbi közös őse $E[IKeres(Szint, F, 1, n, Elso[a], Elso[b])]$.

```
list<int> G[maxN];
int Elso[maxN], Szint[maxN];
int E[2*maxN], F[2*maxN];
int n,en;

void Beolvas();

void MelyBejar(int p){
//Global: G, Szint, E, Elso, en
    E[++en]=p;
    Elso[p]=en;
    Szint[en]=Szint[en-1]+1;
    for (list<int>::iterator pq=G[p].begin(); pq!=G[p].end(); ++pq){
        int q=*pq;
        if (Elso[q]==0){ //p->q faél
            MelyBejar(q);
            E[++en]=p;
            Szint[en]=Szint[Elso[p]];
        }
    }
}
int lKeres(int A[], int F[], int a, int b, int i, int j);
int Elofeldolg(int A[], int F[], int bal, int jobb);

int LKO(int a, int b){
    if ( Elso[a]<Elso[b])
        return E[lKeres(Szint,F, 1,en, Elso[a],Elso[b])];
    else
        return E[lKeres(Szint,F, 1,en, Elso[b],Elso[a])];
}
```

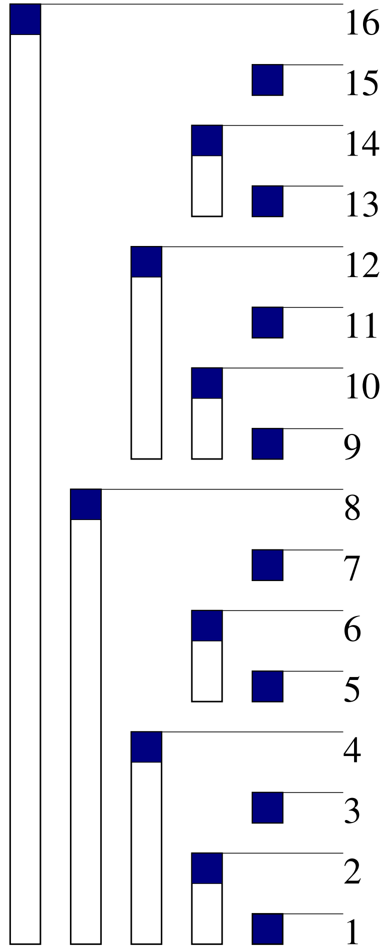
```
int main() {
    int a,b;
    Beolvas();
    for(int p=1;p<=n;p++)
        Elso[p]=0;
    en=0;
    Szint[0]=0;
    MelyBejar(1);
    Elofeldolg(Szint,F,1,en);
    int k;
    cin>>k;
    for(int i=1;i<=k;i++){
        cin>>a>>b;
        cout<<LKO(a,b)<<endl;
    }

   return 0;
}
```

2. Fenwick fa

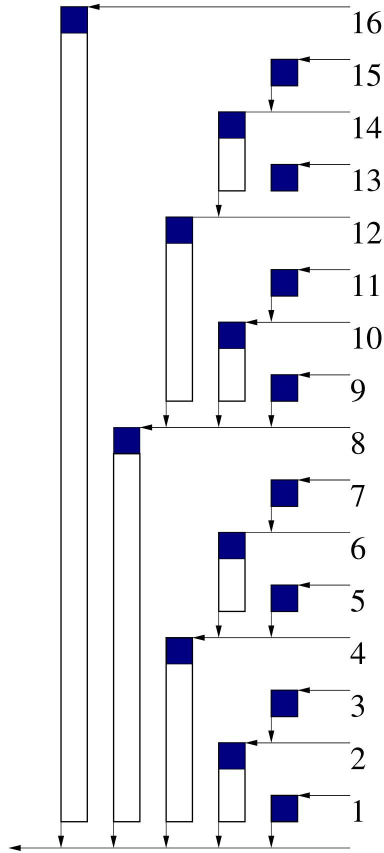
Adott $a_1, \ldots, a_n$ kezdeti sorozat.
$modosit(i,d)$ és $kerdes(i,j)$ műveletek sorozta.
$modosit(i,d)$ hatása: $a_i := a_i + d$
$kerdes(i,j)$ étkéke: $a_i + \cdots + a_j$
Hatékony adatszerkezetet keresünk.
Alapötlet: az $a_1 + \cdots + a_i$ összeget $log\, n$ diszjunkt intervallum kumulatív összegeként lehessen megadni.

$F[i]$ felelős az $i - 2^r + 1..i$ itervallum kumulatív összegéért, ahol r az i szám kettes számrendszerbeli alakjában a legisebb helyiértékő 1-es pozíciója.

```
int osszeg(int ix){
    int szum=0;
    while(ix > 0){
        szum=F[ix];
        ix-=(ix & -ix);
    }
}
void modosit(int i, int d){
    while(i <=n){
        F[i]+=d;
        i+=(i & -i);
    }
}
void init(int n){
   F[0]=0;
   for(int i=1;i<=n;i++)
      F[i]=F[i-1]+a[i];
   for(int i=n;i>0;i--){
      F[i]-=F[i-(i & -i)];
   }
}
```



5. ábra. Binárisan indexel fa



6. ábra. Binárisan indexel fa

3. Véletlenített bináris keresőfák (Fapac, Treap)

3.1. Bináris fa-pont típus

```
class BinFaPont{
    public:
    Elemtip adat; //Elemtip típuson értelmezett a < rend. rel.
    int pri; //prioritási érték
    BinFaPont *bal,*jobb;
    BinFaPont(){};
    BinFaPont(Elemtip x){
        adat=x;
        bal=NULL,jobb=NULL;
        pri=((rand()<<15)+rand())%INF; //véletlen
    }
};
```

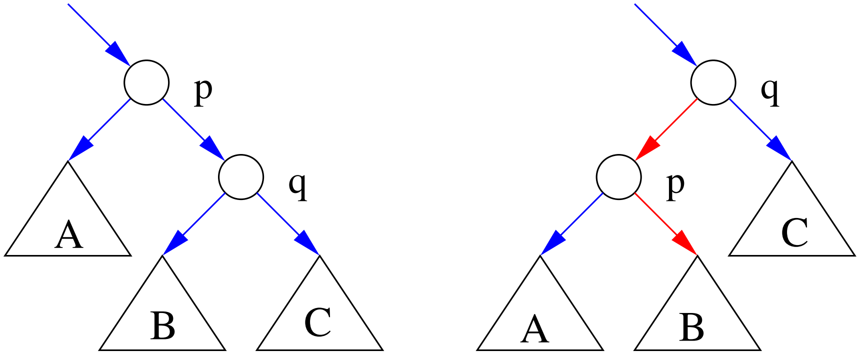
3.1. definíció. *Kupac tulajdonság:* `p->pri <= p->bal->pri` *és* `p->pri <= p->jobb->pri`

3.2. Keresés

```
BinFaPont* Keres(BinFaPont* p, Elemtip x){
    while(p!=NULL){
        if(x<p->adat)
            p=p->bal;
        else if(p->adat<x)
            p=p->jobb;
        else //x==p->adat
            return p;
    }
```

```
    return p;
}
```

3.3. Forgatások

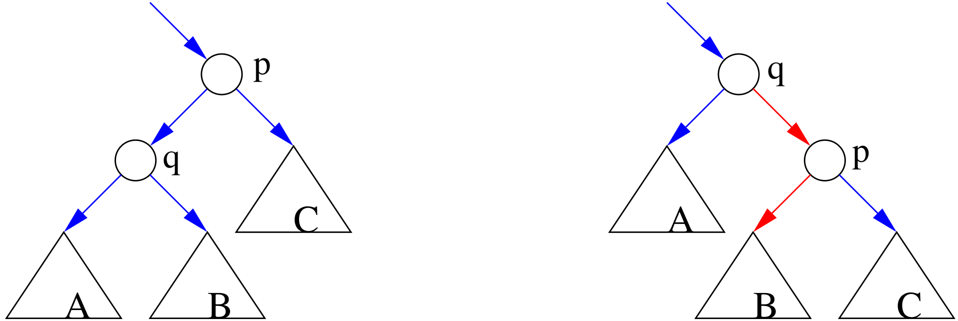


7. ábra.

```
void BForgat(BinFaPont *&p){
    BinFaPont *q=p->jobb;
    p->jobb=q->bal; q->bal=p;
//    Frissit(p); Frissit(q);
    p=q;
}
```

A balra forgatás megőrzi a keresőfa tulajdonságot.

Ha `p->jobb->pri < p->pri` akkor a balra forgatás megőrzi a kupac tulajdoságot.



8. ábra.

```
void JForgat(BinFaPont *&p){
    BinFaPont *q=p->bal;
    p->bal=q->jobb; q->jobb=p;
//    Frissit(p); Frissit(q);
    p=q;
}
```

A jobbra forgatás megőrzi a keresőfa tulajdonságot.

Ha `p->bal->pri < p->pri` akkor a balra forgatás megőrzi a kupac tulajdoságot.

3.4. Bővítés

```
void Bovit(BinFaPont *&p, Elemtip x){
    if (p==NULL){
        p=new BinFaPont(x);
        return;
    }
    if (x<p->adat){
        Bovit(p->bal,x);
        if(p->bal->pri < p->pri) JForgat(p);
    }else{
        Bovit(p->jobb,x);
        if(p->bal->pri < p->pri) BForgat(p);
    }
}
```

3.5. Törlés

```
const long INF = 200000000;
class BinFaPont; //előre deklarálás NIL miatt
BinFaPont* NIL;  //az üres fa repreuentánsa
BinFaPont* fa=NULL;

class BinFaPont{
    public:
    Elemtip adat; int pri;
    BinFaPont *bal,*jobb;
    BinFaPont(){};
    BinFaPont(Elemtip x){
        adat=x;
        bal=NIL,jobb=NIL;
        pri=((rand()<<15)+rand())%INF;
    }
};
//főprogramban:
NIL=new BinFaPont();
NIL->pri=INF;
fa=NIL;
```

```
void Torol(BinFaPont *&p){
    if(p==NIL) return;
    if (p->bal==NIL && p->jobb==NIL)
        {p=NIL; return; }
    if (p->bal->pri < p->jobb->pri){
        JForgat(p); Torol(p->jobb);
    }else{
        BForgat(p); Torol(p->bal);
    }
//      if(p!=NIL) Frissit(p);
}
void Torol(BinFaPont *&p, Elemtip x){
    if(p==NIL) return;
    if (x<(p->adat)){
        Torol(p->bal,x);
    }else if((p->adat)<x){
        Torol(p->jobb,x);
    }else{//p->adat==x
        Torol(p);
    }
 //     if(p!=NIL) Frissit(p);
}
```