Dinamukus programozás

Horváth Gyula
horvath@inf.elte.hu

3. Mohó stratégiával megoldható feladatok

Optimalizálási probléma megoldására szolgáló algoritmus gyakran olyan lépések sorozatából áll, ahol minden lépésben adott halmazból választhatunk. Sok optimalizálási probléma esetén a dinamikus programozási megoldás túl sok esetet vizsgál annak érdekében, hogy az optimális választást meghatározza. Ennél egyszerűbb, hatékonyabb algoritmus is létezik. A **mohó algoritmus** mindig az adott lépésben optimálisnak látszó választást teszi. Vagyis, a lokális optimumot választja abban a reményben, hogy ez globális optimumhoz fog majd vezetni. Olyan optimalizálási problémákkal foglalkozunk, amelyek megoldhatók mohó algoritmussal.

Mohó algoritmus nem mindig ad optimális megoldást, azonban sok probléma megoldható mohó algoritmussal. Láttuk, hogy a pénzváltás probléma nem oldható meg mohó algoritmussal.

3.1. Esemény-kiválasztási probléma

Az első probléma, amit vizsgálunk közös erőforrást igénylő, egymással versengő események ütemezése, azzal a céllal, hogy kiválasszunk egy maximális elemszámú, kölcsönösen kompatibilis eseményekből álló eseményhalmazt. Tegyük fel, hogy adott **események** egy $S = \{a_1, a_2, \ldots, a_n\}$ n elemű halmaza, amelyek egy közös erőforrást, például egy előadótermet kívánnak használni, amit egy időben csak egyik használhat. Minden a_i eseményhez adott az k_i **kezdő időpont** és az b_i **befejező időpont** , ahol $k_i < b_i$. Ha az a_i eseményt kiválasztjuk, akkor ez az esemény az $[k_i, b_i)$ félig nyitott időintervallumot foglalja le. Az a_i és a_j események **kompatibilisek**, ha az $[k_i, b_i)$ és $[k_j, b_j)$ intervallumok nem fedik egymást (azaz a_i és a_j kompatibilisek, ha $k_i \geq b_j$ vagy $k_j \geq b_i$). Az esemény-kiválasztási probléma azt jelenti, hogy kiválasztandó kölcsönösen kompatibilis eseményeknek egy legnagyobb elemszámú halmaza.

Bemenet

A bemeneti állomány első sora az események n ($0 < n \leq 10000$) tartalmazza. A következő n sor mindegyike két egész számot tartalmaz, egy esemény k kezdő és b befejező időpontját ($0 < k < b \leq 1000000$).

Kimenet

A kimeneti állomány első sora a kiválasztott események m számát tartalmazza. A második sor pedig a kiválasztott események sorszámait, egy-egy szóközzel elválasztva. Több megoldás esetén bármelyik megadható.

Példa bemenet és kimenet

Bemenet	**Kimenet**
5 1 9 3 6 5 11 7 12 9 15	2 2 4

Megoldás

Az optimális megoldás szerkezetének elemzése

Tegyük fel, hogy az

$$M = \{i_1, ..., i_u\}$$

halmaz egy otimális megoldása az E esemény-kiválasztási problémának. Feltehtjük, hogy az esemény indexek felsorolására teljesül, hogy $b_{i_1} < b_{i_1} \cdots < b_{i_u}$.

Mohó választás
Legyen $1*$ az az esemény, amelynek a legkisebb a befejezési ideje. Ekkor az

$$M* = \{1*, ..., i_u\}$$

eseményhalmaz is megoldása a kiikdulási E problámának, mivel $b_{1*} \leq b_{i_1}$ ezért az 1* esemény kompatibilis az i_2 , ... , i_u események mindegyikével. Továbbá, $M*$ elemszáma megegyezik M elemszámával, tehát $M*$ is optimális.
Redukált részprobléma
Legyen $\overline{E}$ azon események halmaza, amelyek kompatibilisek az $1*$ eseménnyel, azaz

$$\overline{E} = \{i : b_{1*} \leq k_i, (i \neq i*, 1 \leq i \leq n)\}$$

Legyen $\overline{M}$ egy optimális megoldása az $\overline{E}$ problémának. Megmutatjuk, hogy az $\overline{M} \cup \{1*\}$ eseményhalmaz optimális megoldása lesz a kiindulási E problémának. $\overline{M} \cup \{1*\}$ páronként kompatibilis eseményeket tartalmaz, ezért legfeljebb annyi eleme lehet, mint az M optimális megoldásnak. Az $i_2, ..., i_u$ halmaz részhalmaza $\overline{E}$-nek és páronként kompatibilis eseményekből áll, ezért $\overline{M}$-nek legalább $u-1$ eleme van. Tehát $\overline{M}$ -nek is $u-1$ eleme van, azaz $\overline{M} \cup \{1*\}$ is optimális megoldása a kiindulási E eseményhalmaznak.
A fenti összefüggésekből következik, hogy az esemény-kiválasztási probléma az alábbi lépések végrehajtásával megoldható.
1. **Mohó választás**. Válassuk a fennmaradt eseményekből azt, amelyiknek legkisebb a befejezési ideje.
2. **Redukált részprobléma megoldása**: Oldjuk meg azt a redukált részproblémát, amely azokat az eseményeket tartalmazza, amelyek kezdési ideje nem kisebb, mint a mohó választás (a legkisebb befejezési idejű) eseményének a befejezési ideje.
3. A **redukált probléma megoldáshoz hozzávéve a mohó választást**, a kiindulási probléma optimális megoldását kapjuk.
Megvalósítás

```
1  #include <iostream>
2  #include <stdlib.h>
3  #define maxN 100000
4  using namespace std;
5
6  struct Esemeny{
7     int k,b,az;
8  };
9  Esemeny E[maxN];
10
11 int rend_rel(const void& a, const void& b) {
12   int ab =((Esemeny&)a)->b;
13   int bb =((Esemeny&) b)->b;
14   if (ab<bb) return -1;
15   else if (ab > bb) return 1;
16   else return 0;
17 }
```

```
18 int main(){
19     int n;
20     cin>>n;
21     for (int i=0;i<n; i++){
22         int k,b;
23         cin>>k>>b;;
24         E[i].k=k; E[i].b=b; E[i].az=i+1;
25     }
26 //rendezés
27     qsort((char *)E,n,sizeof(Esemeny), rend_rel);
28     int u=0;      //a megoldáshalmaz elemszáma
29     int M[maxN];//a megoldáshalmaz
30     int szabad=0;//az első szabad időpont
31     for (int i=0;i<n; i++)
32         if (szabad<=E[i].k){
33            M[u++]=E[i].az;
34            szabad=E[i].b;
35         }
36 //kiíratás
37     cout<<u<<endl;
38     for (int i=0;i<u;i++)
39         cout<<M[i]<<" ";
40     cout<<endl;
41 }
```

A mohó stratégia elemi

1. Fogalmazzuk meg az optimalizációs feladatot úgy, hogy választások sorozatával építjük fel a megoldást.
2. Mutassuk meg, hogy mindig van olyan megoldása az eredeti feladatnak, amely a mohó választással kezdődik. Ezt mohó választási tulajdonságnak nevezzük.
3. Bizonyítsuk be, hogy a mohó választással olyan redukált problémát kapunk, amelynek optimális megoldásához hozzávéve a mohó választást, az eredeti probléma megoldását kapjuk. Ezt optimális részprobléma tulajdonságnak nevezzük.

Általában sokféle mohó választás kínálkozik, de nem mindegyik mohó választás eredményez optimális megoldást, ezért fontos, hogy bebizonyítsuk, hogy az adott mohó választás tényleg optimumot eredményez.

3.2. Fényképezkedés

Egy rendezvényre n vendéget hívtak meg. Minden vendég előre jelezte, hogy mettől meddig lesz jelen. A szervezők fényképeken akarják megörökíteni a rendezvényen résztvevőket. Azt tervezik, hogy kiválasztanak k időpontot és minden kiválasztott időpontban az akkor éppen jelenlevőkről csoportképet készítenek. Az a céljuk, hogy a lehető legkevesebb képet kelljen készíteni, de mindenki rajta legyen legalább egy képen. Írjunk olyan programot, amely kiszámítja, hogy legkevesebb hány fényképet kell készíteni, és megadja azokat az időpontokat is amikor csoportképet kell készíteni!

Bemenet

A bemenet első sorában a vendégek n száma van $(1 \leq n \leq 3000)$. A következő n sor mindegyike két egész számot tartalmaz egy szóközzel elválasztva, egy vendég e érkezési és t távozási időpontját $(1 \leq e < t \leq 1000)$. Ha egy fényképet az x időpontban készítik és $e \leq x < t$, akkor azon a fényképen rajta lesz az e időben érkező és t időben távozó vendég.

Kimenet

A kimenet első sorába a készítendő fényképek k számát kell írni! A második sor pontosan k egész számot tartalmazzon egy-egy szóközzel elválasztva, azon időpontokat (tetszőleges sorrendben), amikor a csoportképeket készíteni kell.

Példa bemenet és kimenet

bemenet	kimenet
6 2 4 1 4 2 7 7 13 5 10 3 9	2 3 9

Megoldás

Tehát a bemenet intervallumok egy

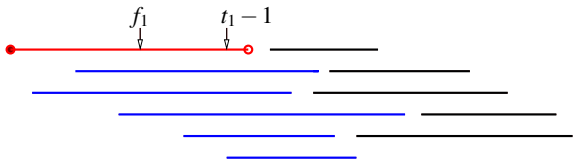
$$I = \{[e_1,t_1),\ldots,[e_n,t_n)\}$$

halmaza, a kimenet pedig olyan minimális elemszámú

$$M = \{f_1,\ldots,f_k\}$$

számhalmaz, hogy minden i-re $i = 1,\ldots,n$ van olyan $f \in M$, hogy $e_i \leq f < t_i$.
Vegyük észre, hogy ha két intervallum jobb-végpontja megegyezik, $t_i = t_j$ akkor amelyik bal-végpontja kisebb,$e_i < e_j$ az elhagyható, hiszen ha $f \in [e_j,t_j)$, akkor $f \in [e_i,t_i)$.
A megoldás elemzése.
Tegyük fel, hogy az intervallumok jobb-végpontjuk szerint növekvően rendezettek, tehát $t_i < t_{i+1}$, $i = 1,\ldots,n-1$ és az M megoldáshalmazra $f_1 < \ldots < f_k$.

Mohó választás.
Válasszuk a megoldáshalmaz első elemének $t_1 - 1$-et.
Megmutatjuk, hogy az optimális megoldásban f_1 helyett állhat a mohó választás, tehát $t_1 - 1$. Először is $f_1 < t_1$, mert különben az 1. intervallumba nem esne egy pontja sem az optimális megoldásnak. Továbbá, minden olyan intervallum, amelyben benne van f_1, benne van $t_1 - 1$ is, hiszen ha $e_i \leq f_1 < t_i$.
Redukált részprobléma.



1. ábra. Mohó választás és probléma redukálás.

Töröljünk I-ből mindan olyan intervallumot, amelyben benne van a $t_1 - 1$ mohó választás: $I' = I - \{[e_i,t_i) : e_i < t_1\}$. Az $M' = \{f_2,\ldots,f_k\}$ ponthalmaz megoldása lesz az I' problémának. I' optimális is, mert ha lenne kevesebb pontot tartalmazó megoldása I'-nek, akkor hozzávéve $t_1 - 1$-et, vagy f_1-et, a kiindulási I probléma kisebb elemszámú megoldását kapnánk, mint $|M|$.
Megvalósítás

```
1  #include <iostream>
2  #include <stdlib.h>
3  #define maxN 100000
4  using namespace std;
5
6  struct Vendeg{
7     int e,t;
8  };
9  Vendeg V[maxN];
10
```

```
11 int rend_rel(const void* a, const void* b) {
12   int at =((Vendeg*)a)->t;
13   int bt =((Vendeg*) b)->t;
14   if (at<bt) return -1;
15   else if (at > bt) return 1;
16   else return 0;
17 }

18 int main(){
19    int n;
20    cin>>n;
21    for (int i=0;i<n; i++){
22       int e,t;
23       cin>>e>>t;;
24       V[i].e=e; V[i].t=t;
25    }
26 //rendezés
27    qsort((char *)V,n,sizeof(Vendeg), rend_rel);
28    int k=0;     //a megoldáshalmaz elemszáma
29    int M[maxN];//a megoldáshalmaz
30    int utolso=0;//az utpolsó fényképezési időpont
31    for (int i=0;i<n; i++)
32       if (utolso<V[i].e){
33          utolso=V[i].t-1;
34          M[k++]=utolso;
35       }
36 //kiíratás
37    cout<<k<<endl;
38    for (int i=0;i<k;i++)
39       cout<<M[i]<<"␣";
40    cout<<endl;
41 }
```

Megoldás rendezés nélkül

Vegyük észre, hogy ha két intervallum jobb végpontja megegyezik, akkor a kisebb bal végpontú elhagyható a bemenetből. Tehát a bemeneti intervallumk halmaza ábrázolható egy E tömbbel úgy, hogy $E[b] = a$ ha az a érkezési, b pegig távozási időpont. $E[b] = 0$, ha nincs olyan vendég, aki a b időpontban távozna.

```
1  #include <iostream>
2  #include <stdlib.h>
3  #define maxN 100000
4  #define maxM 10001
5  using namespace std;

6  int E[maxN];
7  int main(){
8     int n;
9     cin>>n;
10    for (int x=1;x<=maxM;x++) E[x]=0;
11    for (int i=0;i<n; i++){
12       int e,t;
13       cin>>e>>t;
14       if (E[t]<e)
15          E[t]=e;
16    }
17    int k=0;      //a megoldáshalmaz elemszáma
18    int M[maxN]; //a megoldáshalmaz
19    int utolso=0;//az utpolsó fényképezési időpont
```

```
20      for (int x=1; x<maxM; x++)
21          if (utolso<E[x]){
22              utolso=x-1;
23              M[k++]=utolso;
24          }
25  //kiíratás
26      cout<<k<<endl;
27      for (int i=0;i<k;i++)
28          cout<<M[i]<<"␣";
29      cout<<endl;
30  }
```

3.3. Jegykiosztás

Egy rendezvényt olyan teremben tartanak, amelyben m ülőhely van, 1-től m-ig sorszámozva. A szervezők megrendeléseket fogatnak, minden megrendelés egy $a\,b$ számpárral adható meg, ami azt jelenti, hogy a megrendelő olyan s ülőhelyet szeretne kapni, amelyre teljesül, hogy $a \leq s \leq b$.
Adjunk olyan programot, amely kiszámítja, hogy a szervező a megrendelések alapján a legjobb esetben hány megrendelést tud kielégíteni és meg is ad egy olyan jegykiosztást, amely kielégíti a megrendeléseket!

Bemenet

A bemeneti állomány első sora két egész számot tartalmaz, az ülőhelyek m ($1 < m \leq 10000$) számát és a megrendelések n ($1 < n \leq 1000000$) számát. A további n sor mindegyike egy megrendelést tartalmaz, két egész számot $a\,b$ (egy szóközzel elválasztva), $1 \leq a \leq b \leq m$.

Kimenet

A kimeneti állomány első sorába a legtöbb kielégítheto megrendelés k számát kell írni! A további k sor tartalmazza a jegykiosztást a kiválasztott k megrendelés számára. Minden sor két egész számot tartalmazzon egy szóközzel elválasztva. Az első szám egy megrendelés sorszáma, a második pedig azon ülőhely sorszáma, amelyet a megrendelő kap. A kiosztás kiírása tetszőleges sorrendben lehet. Ha több megoldás van, akkor egytetszőlegeset ki lehet írni.

Példa bemenet és kimenet

bemenet	**kimenet**
10 9	1 1
1 3	2 2
2 4	5 3
5 7	4 4
2 6	6 5
1 5	3 6
3 7	7 7
4 8	9 8
7 9	8 9
3 8	

A problémának két összetevője van: az igények mint intervallumok halmaza: I; és az ölőhelyek H halmaza:

$$I = \{[a_1,b_1]\cdots[a_n,b_n]\}, H = \{1,\ldots,m\}$$

ahol $1 \leq a_i \leq b_i \leq m$.
Az (I,H) probléma minden megoldása megadható egy $M : \{1,\ldots,n\} \to H$ függvénnyel, ahol teljesül, hogy $M(x) = 0$ ha az x ülőhelyet nem adjuk oda egyetlen igénylőnek sem, egyébként ha $M(x) \neq 0$, akkor az x ülőhelyet az $M(x)$ sorszámú igénylő kapja. Teljesül, hogy $a_{M(x)} \leq x \leq b_{M(x)}$. Nyilvánvalóan teljesülni kell, hogy ha $x \neq y$ és $M(x) = M(y)$ akor $M(x) = 0 = M(y)$, azaz egy széket legfeljebb egy igénylőnek adunk.
Az M megoldás optimális, ha a $\{x : M(x) \neq 0\}$ elemszáma a lehető legnagyobb.

Első megoldás

Mohó választás.
Kettős problémával állunk szemben: intervallum- és ülőhely-választás. Válasszuk azt az intervallumot, amelynek jobb végpontja a legkisebb, ha több ilyen van, akkor ezek közül azt amelyik bal végpontja a legkisebb. Jelölje ennek sorszámát $i*$
Ülőhelnek válasszuk az $[a_{i*}, b_{i*}]$ intervallumba eső első szabad ülőhelyet, tehát az első választásnál ez a_{i*}.
Megmutatjuk, hogy van olyan optimális megoldás, amely a mohó választást tartalmazza.
Tegyük fel, hogy az M megoldás nem tartalmazza a mohó választást, pontosabban, hogy $M(a_{i*}) \neq i*$.
Két eset lehetséges: a) $M(a_{i*}) = 0$. Ekkor változtassuk az M megoldást úgy, hogy $M(a_{i*}) = i*$ és arra az x-re, amelyre $M(x) = i*$ volt $M(x) = 0$ legyen.
A második eset b) $M(a_{i*}) = j* \neq 0$. Ha az $i*$ igénylő is kap ülőhelyet az M optimális megoldásban, azaz van olyan u, hogy $M(u) = i*$, akkor az alábbi egyenlőtlenségek teljesülnek a mohó választás miatt.

$$a_{j*} \leq a_{i*} \leq u \leq b_{i*} \leq b_{j*}$$

Tehát módosíthatjuk az M optimális megoldást úgy, hogy

$$M(a_{i*}) = a_{i*} \text{ és } M(j*) = u$$

legyen. Ha az M optimálsi megoldás nem tartalmazza $i*$ igéynt, azaz nincs olyan x, hogy $M(x) = i*$, akkor módisítsuk M-et úgy, hogy $M(a_{i*} = i*)$ legyen.
Tehát beláttuk, hogy van olyan optimális megodás, amely tartalmazza a mohó választást, azaz $M(a_{i*}) = i*$.
Redukált részprobléma.
Legyen M egy optimális megoldása az (I,H) problémának és $i*$ a mohó választás, és $M(a_{i*}) = i*$.
Ekkor a redukált részprobléma:

$$\overline{I} = I - \{[a_{i*}, b_{i*}]\}\ \overline{H} = H - \{a_{i*}\}$$

Tekintsük azt az $\overline{M}$ megoldást, amelyre teljesül, hogy $\overline{M}(x) = M(x)$, ha $x \neq a_{i*}$. $\overline{M}$ nyilvánvalóan megoldása az $(\overline{I},\overline{H})$ redukált problémának. Továbbá optimális is, mert ha lenne nagyobb elemszámú megoldása, akkor ahhoz hozzávéve a mohó váalsztást, a kiindulási probléma jobb megoldását kapnánk, mint M.
Megvalósítás.

```
1  #include <iostream>
2  #include <stdlib.h>
3  #define maxN 1000000
4  #define maxM 10000
5
6  using namespace std;
7
8  struct Igeny{
9     int a,b,az;
10 };
11 Igeny E[maxN];
12
13 int rend_rel(const void* x, const void* y) {
14   int xa =((Igeny*)x)->a;
15   int xb =((Igeny*)x)->b;
16   int ya =((Igeny*)y)->a;
17   int yb =((Igeny*)y)->b;
18   if (xb<yb || xb==yb && xa<ya) return -1;
19   else if (xa==ya && xb==yb) return 0;
20   else return 1;
21 }

22 int main(){
23    int m,n;
24    cin>>m>>n;
25    for (int i=0;i<n; i++){
26       int a,b;
```

```
27          cin>>a>>b;;
28          E[i].a=a; E[i].b=b; E[i].az=i+1;
29      }
30      qsort((char *)E,n,sizeof(Igeny), rend_rel); //rendezés
31      int u=0;      //a megoldáshalmaz elemszáma
32      int Ki[maxM]; //jegykiosztás:x. helyre Ki[x] ül
33      for (int i=0;i<=m; i++)  Ki[i]=0;
34      for (int i=0;i<n;i++){
35          int x=E[i].a;
36          while (x<=E[i].b && Ki[x]>0) x++;
37          if (x<=E[i].b){
38              u++;
39              Ki[x]=E[i].az;
40          }
41      }
42      cout<<u<<endl;
43      for (int i=1;i<=m;i++) if (Ki[i]>0)
44              cout<<Ki[i]<<" "<<i<<endl;
45      cout<<endl;
46  }
```

Az algoritmus futási ideje legrosszabb esetben $O(n*m)$. Ez bekövetkezik, ha minden igény az $[1,m]$ intervallum.
Az algoritmus lényegesen gyorsabbá tehető, ha olyan adatszerkezetet használunk, amely lehetővé teszi, hogy gyorsan meg tudjuk határozni adott a-ra a legkisebb, olyan u helyet, amely még szabad és $a \leq u$. Az Unio-Holvan adatszerkezet egy ilyen megoldást kínál. Olyan $[u,v]$ diszjunkt intervallumokat ábrázolunk Unio-Holvan fával, ahol v szabad hely, és minden $u \leq x < v$ esetén x nem szabad hely. Ekkor az algoritmus futási ideje $O(n\log(n)+m)$.

Második megoldás

Adott x ülőhelyre jelölje $H(x)$ azokat az igényeket, amelyeknek megfelelő az x sorszámú ülőhely:

$$H(x) = \{i : a_i \leq x \leq b_i\}$$

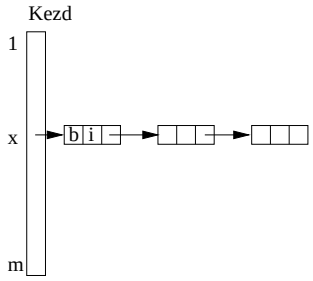
Moho választás
Az ülőhelyek szerint növekvően haladva az x ülőhelyre válasszuk azt a megfelelő igénylőt, akihez tartozó intervallum jobb végpontja a legkisebb.

```
1  H=Ures;
2  for (x=1; x<=m; x++){
3     H bővítése azon i-kkel, amelyekre a[i]=x;
4     if (H!=Ures){
5       i=H minimális jobb-végpontú eleme;
6       Beoszt[x]=i;
7     }
8     H azon elemeinek törlése, amelyekre x=bi;
9  }
```

A H halmazokon végzendő műveleteket megvalósíthatjuk priosítási sorral.
Ekkor az algoritmus futási ideje $\mathbb{O}(m \lg n)$ Megmutatjuk, hogy van olyan optimális megoldás, ami a mohó választással kezdődik. Legyen M egy optimális megoldása az I,H problémának, ahol az M megoldásra az első megoldásban megfogalmazott feltétel teljesül. Legyen $i*$ a mohó választás, azaz a_{i*}, b_{i*} a legisebb bal végpontú intervallum, ha több ilyen van, akkor ezek közül a legkisebb jobb végpontú. Az első megoldásban alkalmazott módon belatható, hogy az M megoldás módosítható úgy, hogy $M(a_{i*}) = i*$ legyen.
Redukált részprobléma
Ugyanaz, mint az első megoldásnál:

$$\overline{I} = I - \{[a_{i*}, b_{i*}]\}\ \overline{H} = H - \{a_{i*}\}$$

Tekintsük azt az $\overline{M}$ megoldást, amelyre teljesül, hogy $\overline{M}(x) = M(x)$, ha $x \neq a_{i*}$. $\overline{M}$ nyilvánvalóan megoldása az $(\overline{I},\overline{H})$ redukált problémának. Továbbá optimális is, mert ha lenne nagyobb elemszámú megoldása, akkor ahhoz hozzávéve a mohó váalsztást, a kiindulási probléma jobb megoldását kapnánk, mint M.



2. ábra. $Kezd[x]$ azon (a_i, b_i) intervallumokat tartalmazza, amelyekre $x = a_i$

```
#include <iostream>
#include <queue>
#define  maxM 10000
using namespace std;

class Elem{
  public:
    Elem() {};
    Elem(int x, int y) { b = x; az = y; }
    bool operator<(const Elem&) const;
    int b; int az;
};
bool Elem::operator < (const Elem& jobb) const{
  return b > jobb.b ;
}
struct Cella{
    int b; int az;
    Cella Λcsat;
};

int main() {
    int Beoszt[maxM+1];
    CellaΛ Kezd[maxM+1];
    int m,n,a,b;
    CellaΛ p;
    Elem e;
    priority_queue<Elem> H;
    cin >> m; cin >> n;
    for (int i=1; i<=n; i++) {
        cin >> a; cin >> b;
        p = new Cella;
        p-> b=b; p->az=i;
        p->csat=Kezd[a];
        Kezd[a]=p;
    }

    for (int x=1; x<=m; x++) {
       p=Kezd[x];
       while (p!=NULL){
         H.push(Elem(p->b,p->az));
         p=p->csat;
       }
       if (H.size()>0){
         e=H.top(); H.pop();
         Beoszt[x]=e.az;
```

```
44          }
45          while(H.size()>0 && H.top().b==x){
46            H.pop();
47          }
48      }//for x
49      int hany=0;
50      for (int x=1; x<=m; x++)
51        if (Beoszt[x]>0) hany++;
52      cout << hany << endl;
53      for (int x=1; x<=m; x++)
54        if (Beoszt[x]>0) cout << Beoszt[x] << "␣" << x << endl ;
55
56      return 0;
57  }//end main
```

3.4. Darabolás

Adott egy fémrúd, amelyet megadott számú és hosszúságú darabokra kell felvágni. A darabok hosszát milliméterben kifejezett értékek adják meg. Olyan vágógéppel kell a feladatot megoldani, amely egyszerre csak egy vágást tud végezni. A vágások tetszőleges sorrendben elvégezhetőek. Egy vágás költsége megegyezik annak a darabnak a hosszával, amit éppen (két darabra) vágunk. A célunk optimalizálni a műveletsor teljes költséget.
Készítsünk programot, amely kiszámítja a vágási műveletsor optimális összköltségét és megad egy olyan vágási sorrendet, amely optimális költséget eredményez.

Bemenet

A bemenet első sora egy egész számot tartalmaz, a darabok n számát $(0 < n \leq 1000)$. A második sor n darab pozitív egész számot tartalmaz egy-egy szóközzel elválasztva, a darabok hosszát. A második sorban szereplő számok nem nagyobbak, mint 1000.

Kimenet

A kimenet első sorába egyetlen számot, a vágási műveletsor optimális összköltségét kell írni! A további $n-1$ sor mindegyikébe két egész számot kell írni, egy szóközzel elválasztva. Az első szám legyen az adott lépésben kettévágott rúd hossza, a második szám pedig az egyik keletkező darab hossza. Minden sor csak olyan hosszúságú darab kettévágását tartalmazhatja, amelyből a korábbi lépések során több keletkezett, mint az azóta elvégzett lépések által felhasználtak száma.

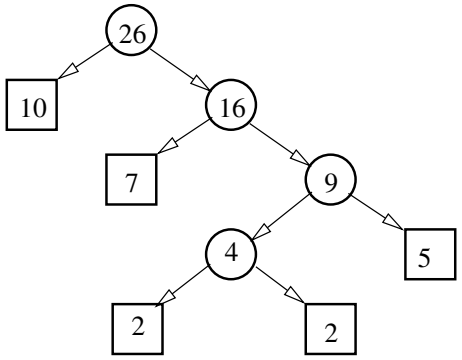
Példa bemenet és kimenet

bemenet

```
5
2 5 2 7 10
```

kimenet

```
55
26 10
16 7
9 4
4 2
```



3. ábra. A példa megoldásának ábrázolása bináris fával.

Elemezzük az optimális megoldás szerkezetét. Vegyük észre, hogy minden darabolás, így az optimális is leírható egy bináris fával. A fa levelei tartalmazzák a bemenetként kapott darabok hosszait, és minden belső pontja annak a darabnak a hosszát, amelyből vágással a két fiú-pontban lévő darab keletkezett, azaz a két fiának az összegét. Példánk esetén a fa a következőképpen néz ki.

A darabolás összköltsége is kifejezhető a fával, nevezetesen, az összköltség éppen a fa belső (nem levél) pontjaiban található számok összege. Fordítva is igaz, minden ilyen fa egy darabolást ír le. A fa költségén a fa belső pontjaiban lévő számok összegét értjük. Tehát keressük az optimális megoldást, mint egy darabolási fát, tehát azt, amelynek a költsége minimális. A darabolási fa költsége kifejezhető a következőképpen. Legyenek $d_1, \ldots, d_n$ a vágandó darabok hosszai és legyen m_i a d_i darabot tartalmazó levélpont mélysége (a fa gyökerétől vett távolsága) a fában. Ezekkel a jelölésekkel a fa költsége:

$$\sum_{i=1}^{n} m_i * d_i$$

Az optimális fára a következő két állítás teljesül.

3.1. lemma. *A két legkisebb értéket tartalmazó levélpont mélysége a legnagyobb, és testvérek.*
Bizonyítás. *Ha az állítás nem teljesülne, akkor a két legmélyebb testvér levélpontot felcserélve a két legkisebb értéket tartalmazó levéllel, kisebb költségű fát kapnánk.* ■

3.2. lemma. *Legyen d_u és d_v a két legkisebb darab. Ha az optimális fában töröljük a d_u-t és d_v-t tartalmazó levélpontot, akkor olyan fát kapunk, amely optimális arra a bemenetre, amely d_u és d_v helyett a $d_u + d_v$ darabot tartalmazza.*
Bizonyítás. *A két levél törlésével kapott fa nyilván darabolási fa lesz a módosított bemenetre, amelynek költsége*

$$\sum_{i=1}^{n} m_i * d_i - (d_u + d_v)$$

Legyen F egy optimális darabolási fa a módosított bemenetre és legyen a költsége K. Ha a $d_u + d_v$ darabot tartalmazó levélponthoz hozzávesszük bal fiúként a d_u értéket tartalmazó, jobb fiúként pedig a d_v értéket tartalmazó új levelet, akkor egy olyan fát kapunk, amely darabolási fa lesz az eredeti bemenetre, költsége pedig $K + d_u + d_v$. Ez azonban nem lehet kisebb, mint az optimális darabolási fa költsége az eredeti bemenetre, tehát

$$\sum_{i=1}^{n} m_i * d_i \leq K + (d_u + d_v)$$

$$\sum_{i=1}^{n} m_i * d_i - (d_u + d_v) \leq K \leq \sum_{i=1}^{n} m_i * d_i - (d_u + d_v)$$

Ami az állítás bizonyítását jelenti. ■

Most már megfogalmazhatjuk a mohó stratégiánkat. Építsük fel a darabolási fát úgy, hogy lépésenként a két legkisebb értéket tartalmazó pontot egy új pont két fiává tesszük, és az új pontba a két fiúban lévő érték összegét írjuk. Az 1. Állítás igazolja a mohó választási tulajdonságot, a 2. Állítás pedig az optimális részproblémák tulajdonságot, tehát korrekt algoritmust kapunk.
Megvalósítás.
A mohó választás megvalósítására prioritási sort alkalmazunk.
A darabolási fának n levele és $n-1$ belső pontja van. A leveleket az $1,\ldots,n$ számokkal, a belső pontokat az $n+1,\ldots,2*n-1$ számokkal azonosítjuk, a gyökér azonosítója $2*n-1$. A prioritási sorba (f,h) párokat teszünk, ahol f egy fapont azonosítója, h pedig annak a darabnak hossza, amit az f fapont reprezentál.

```cpp
#include <iostream>
#include <queue>
#define maxN 10000
using namespace std;
class Elem{
  public:
   int hossz; int azon;
    Elem() {};
    Elem(int x, int y) { hossz = x; azon = y; }
    bool operator<(const Elem&) const;
};
bool Elem::operator < (const Elem& jobb) const{
  return hossz > jobb.hossz ;
}

struct Fapont{
   int bal,jobb, hossz;
};
Fapont Fa[maxN];

void KiIr(int f){
   cout<<Fa[f].hossz<<" "<<Fa[Fa[f].bal].hossz<<endl;
   if (Fa[f].bal!=0) KiIr(Fa[f].bal);
   if (Fa[f].jobb!=0) KiIr(Fa[f].jobb);
}

int main(){
   int n,dh, OsszKolts=0;
   Elem e,x,y;
   priority_queue<Elem> S;

   cin>>n;
   for (int i=1;i<=n; i++){
      cin>>dh;
      e.azon=i; e.hossz=dh;
      S.push(e);
      Fa[i].hossz=dh;
      Fa[i].bal=0; Fa[i].jobb=0;
   }
   for (int i=n+1;i<2*n;i++){
      x=S.top(); S.pop();
      y=S.top(); S.pop();
      e.azon=i; e.hossz=x.hossz+y.hossz;
      S.push(e);
      Fa[i].hossz=e.hossz;
      Fa[i].bal=x.azon; Fa[i].jobb=y.azon;
      OsszKolts+=e.hossz;
   }
   cout<<OsszKolts<<endl;
   KiIr(2*n-1);
}
```