Hip (eng. heap) – uvod i zadaci

1. Uvod (preskociti ako Vam je poznat)

Heap je uređeno potpuno binarno stablo.

Heap se može realizovati kao kompletno stablo u kome svaki čvor i njegov naslednik su u relaciji poretka R, gde R može biti relacija <, >,...
Max-heap – potpuno binarno stablo u kom je roditelj uvek veći od svoje dece.
Min-heap – potpuno binarno stablo u kom je roditelj uvek manji od svoje dece.

Heap je struktura podataka koja može u složenosti O(1) reći koji je najveći (ili najmanji-zavisno da li implementiramo max-heap ili min-heap) element u heap-u.
Heap u složenosti O(log n) može izbaciti najveći ili ubaciti novi element u heap-u.

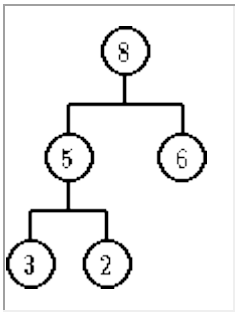
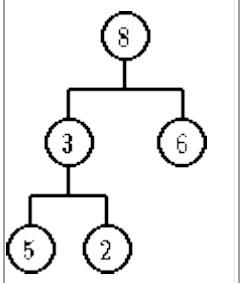
Na primer, binarno stablo je **kompletno popunjeno** ako ima visinu h i 2^{h+1}-1 cvorova.
Binarno stablo visine h je **kompletno** *ako i samo ako* je

a. prazno *ili* je
b. levo podstablo kompletno visine h-1 i desno podstablo je kompletno popunjeno visine h-2 *ili* je
c. levo podstablo kompletno popunjeno visine h-1 i desno podstablo je kompletno visine h-1.

Kompletno stablo se popunjava sa leve strane:

- svi listovi su
  - na istom nivou *ili*
  - na dva susedna nivoa *i*
- svi čvorovi na najnizem nivou se nalaze što je moguće više levo.

Primer: Neka R je relacija '>' i drvo ima stepen 2. Dakle, heap nije BSP (binarno stablo pretrage), jer svako dete mora u heap-u biti manje od oca. Ne može kao u BSP, da npr. desno dete bude veće.

Hip	Nije hip
	

Max-hip je binarno stablo koje zadovoljava uslov hipa: ključ svakog čvora veći je ili jednak od ključeva njegovih sinova.
Min-hip je binarno stablo koje zadovoljava uslov hipa: ključ svakog čvora manji je ili jednak od ključeva njegovih sinova.

Max-Heap je struktura podataka koja može za vremensku složenost O(1) dati odgovor koji je najveći element u hipu, dok u složenosti O(log n) može izbaciti najveći element u hipu ili ubaciti novi element.
Hip se može realizovati implicitno i eksplicitno.

Dakle, ako hip ima n elemenata, onda se za smeštanje elemenata koriste lokacije u nizu A sa indeksima A[1..n] (ako se niz indeksira počev od 1, tj. A[1] je 1. član), tako da ako element indeksa i predstavlja čvor stabla, tada

- element indeksa 2*i predstavlja levo dete čvora,
- element indeksa 2*i+1 predstavlja desno dete čvora.

Na primer, deca čvora A[1] su elementi A[2], A[3]

1. Primene

Lista sa prioritetom: dinamički skup čiji elementi se uklanjaju po redosledu veličina, počev od najvećeg. Na primer, objekat sa najvećim prioritetom se nalazi u korenu heap-a i trivijalno se uzima iz strukture. Ali ako se ukloni koren, tada ostaju dva podstabla i moraju se *efikasno* spojiti u jedno stablo koje će ponovo imati ***heap*** svojstvo.

Korist od upotrebe heap strukture zasniva se na svojstvu da se moze izvaditi objekat najviseg prioriteta ili ubaciti za **O(log*n*)** vremena.

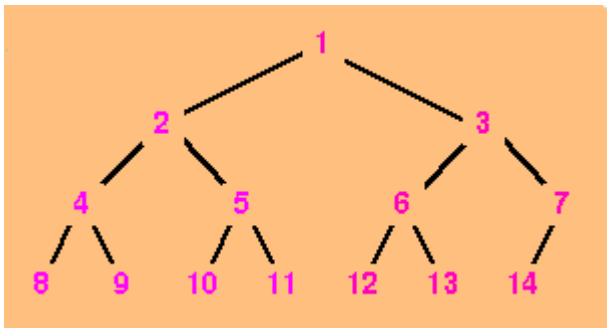
Heap Sort: od kolekcije elemenata kreira se heap za (O(n)) vremena i potom se uklanjaju elementi iz heap-a, a svako uklanjanje ima složenost proporcionalnu visini heap-a, tj. (O(logn)), te ukupna složenost je (O(n log n)) (donja granica).

REALIZACIJA

1. eksplicitno zadatim stablom
2. vektor,

tj. implicitno zadato stablo sa pozicijama 1..n. Koren je na poziciji 1. Na poziciji i > 1 je dete roditelja sa pozicije $\lfloor i/2 \rfloor$.Na primer, ako niz A je implicitno zadato stablo, onda element**A[16] odgovara čvoru čiji predak je čvor pridružen elementu A[8].**

Svojstva kompletnog stabla vode do veoma efikasnog mehanizma za memorijsko predstavljanje koristeci *n* uzastopnih pozicija u nizu, tj. vektoru..

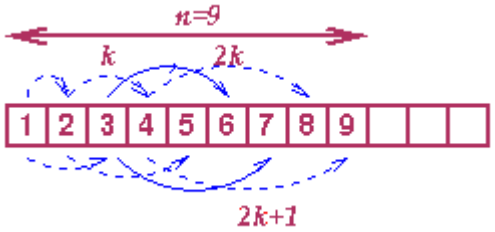


Ako su čvorovi numerisani počev od 1 u korenu i ako je:

- levi potomak čvora ***k*** na poziciji **2*k***
- desni potomak cvora ***k*** na poziciji **2*k*+1**

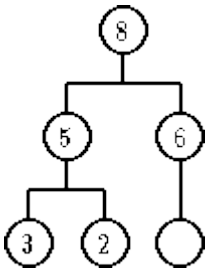
tada osobina 'popunjenosti s leve strane' kompletnog stabla obezbeđuje da heap može da se smesti u uzastopnim pozicijama u vektoru.

Posmatran kao niz, vidi se da se *n*-ti čvor nalazi na *n*-tom mestu u nizu.

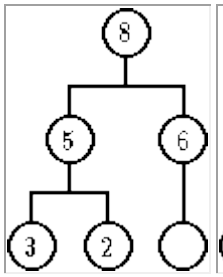
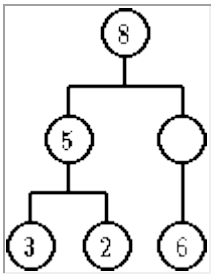
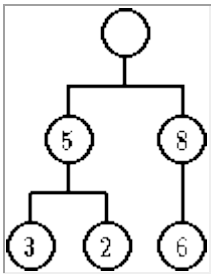


2. Umetanje elementa u hip

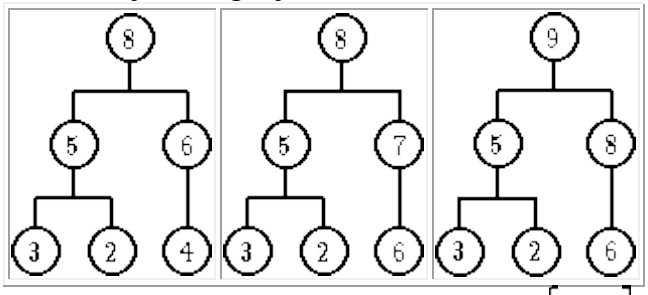
1. dodati čvor u stablo (postaviti ga na mesto sledeceg lista sa desne strane)



2. Pomeranje elemenata od korena do novog čvora sve dok se ne uspostavi svojstvo heap-a.

novi element	4	7	9
ažurirano stablo			

3. Ubacivanje novog ključa u slobodan čvor

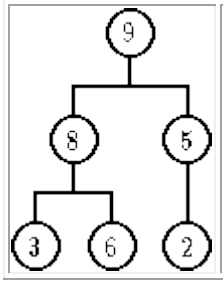
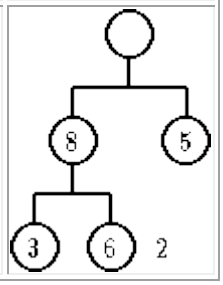


4. Potpuno stablo sa n čvorova ima dubinu $\lceil \log n \rceil$, te je vremenska kompletnost umetanja elementa u hip O(log n), jer u gore opisanom procesu je potrebno da izvrsimo najvise *h* izmena korena podstabla sa jednim od njegovih potomaka kako bismo u potpunosti vratili svojstvo heap-a. Zato je za brisanje korena iz heap-a potrebno **O(*h*)**, odnosno **O(log *n*)**vreme.

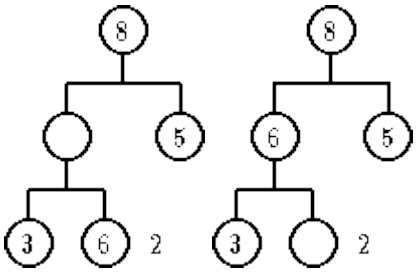
Ponovo, za ovu proceduru je potrebno **O(*h*)**, odnosno **O(log*n*)** zamena.

2. Uklanjanje elementa iz hipa

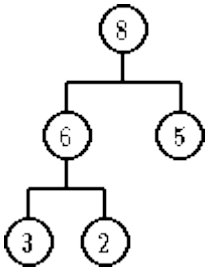
1. Ukloniti ključ najvećeg elementa, tj. koren, potom sačuvati ključ najmanjeg elementa, ali ukloniti poslednji čvor, tj. čvor koji je sadržao taj element (npr. vrednost 2)

pre	posle
	

2. Dok god je sačuvana vrednost manja od deteta upražnjenog čvora, pomeriti najveću vrednost deteta nagore do upražnjenog čvora. Konkretno, ključ deteta 8 se prekopira u koren, ključ 6 se prekopira u dete korena.



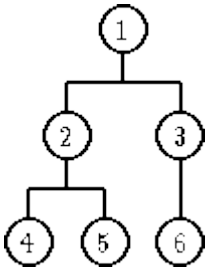
3. Zapamćena vrednost se kopira u ključ upražnjenog čvora.



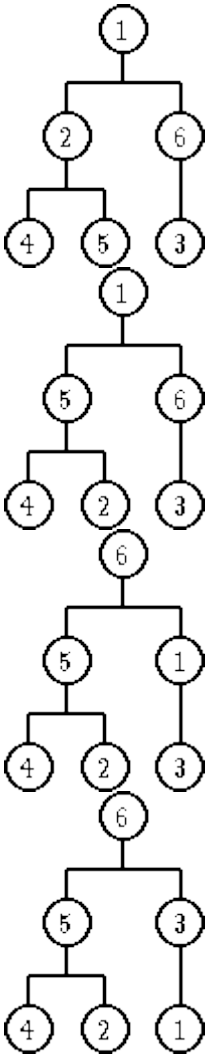
4. Složenost O(log n), jer u ovom procesu je potrebno da se izvrši najviše *h* izmena korena podstabla sa jednim od njegovih potomaka kako bismo u potpunosti važilo heap svojstvo . Zato je za brisanje korena iz heap-a potrebno **O(*h*)**, odnosno **O(log*n*)** vreme.

3. Formiranje hipa (korisno za heap sort) - bottom up pristup

- Ubaciti n elemenata e_1, ..., e_n u potpuno stablo



- Za svaki čvor, počev od poslednjih (najnižih) ka korenu, vrši se reorganizacija podstabla u heap tako što se izvrši zamena korena podstabala sa većim detetom sve dok koreni ne budu veći od dece.



- Vremenska složenost formiranja heap-a je jednaka sumi složenosti preuređivanja podstabala redom veličine n/2, n/4, n/8, ...

$O(0 * (n/2) + 1 * (n/4) + 2 * (n/8) + \ldots + (\log n) * 1) = O(n(0 * 2^{-1} + 1 * 2^{-2} + 2 * 2^{-3} + \ldots + (\log n) * 2^{-\log n})) = O(n)$

znajući da izraz u zagradi u stvari ima vrednost

$\sum_{k=1}^{\infty}(k - 1)2^{-k} = 2[\sum_{k=1}^{\infty}(k - 1)2^{-k}] - [\sum_{k=1}^{\infty}(k - 1)2^{-k}] = [\sum_{k=1}^{\infty}k2^{-k}] - [\sum_{k=1}^{\infty}(k - 1)2^{-k}] =$

$= \sum_{k=1}^{\infty}[k - (k - 1)]2^{-k} = \sum_{k=1}^{\infty}2^{-k} = 1$

2. Zadaci

1. U pekari se prodaju đevreci. Na početku dana spremljeno je n đevreka i svaki je kvaliteta x. Kako vreme prolazi, peku se novi đevreci različitog kvaliteta, dolaze kupci i kupuju đevrek. Kada kupac dođe, zahteva od prodavca u pekari da mu proda najbolji đevrek koji ima. Prodavac tada odabere najbolji đevrek i proda ga. Pomozite pekari da što bolje posluje tako što će prodavac uvek dati kupcima najbolji đevrek. Dato je n (1<=n<=100000) i nakon toga n brojeva x. Broj x opisuje kvalitet đevreka. Nakon toga sledi broj m, zatim m (1<=m<=100000) brojeva y. Sa y su opisani događaji tokom dana. Ako je y nula znači da je došao kupac i kupio najbolji đevrek, te treba ispisati kvalitet prodanog đevreka ili string “Nema!” ako nema đevreka. Ako y nije nula, znači da je ispečen novi đevrek kvaliteta y koji se od tog trenutka može kupiti.

Pokušajte da zadatak rešite bez korištenja struktura podataka: set, map, priority_queue iz STL-a.

Primer unosa
5
15 32 4 18 29
12
0 0 50 0 24 97 0 0 0 0 0 0

Izlaz
32 29 50 97 24 18 15 4 Nema!

Rešenje:

1. ucitavamo n clanova i ubacimo n elemenata x u max-heap
 Jednostavnosti radi, heap indeksiramo pocev od 1 (ne od 0).
2. ucitavamo m elemenata i ubacujemo/izbacujemo u/iz max-heap elemenat y

I način

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> heap;

void ubaci(int x);
void izvadi(void);

int main()
{ heap.push_back(0);
  //indeks 0 se ne koristi
  int n,x;
   cin >> n;
  for(int i=0;i<n;i++)
  {   cin >> x; ubaci(x); //umetanje x u heap
  }

  int m,y;
  cin >>m;
  for (int i=0;i<m;i++)
   {
     cin >> y;
     if (y==0)  //dolazi kupac
     {
       if(heap.size()>1)
       {cout<<heap[1]<<' ';
        izvadi();
```

```
        }
       else cout <<"Nema!";
    }else ubaci(y);
 }
   return 0;
}

void ubaci(int x)
{
  heap.push_back(x);
  int t=heap.size()-1;
  while (t/2 && heap[t]>heap[t/2])  //krecemo od vrha i dokle god je otac veci od dece spustamo se nanize
 {heap[t]=heap[t/2]; heap[t/2]=x;
   t=t/2;}
}

 void izvadi()
{
 heap[1]=heap.back();
 heap.pop_back();
 int t=1,r; //t=index koji //spustamo niz heap
//r=indeks s kojim trampimo heap[t]
 while (1)
 {if(t*2+1<heap.size()) //ako heap[t] ima 2

//deteta
  {if (heap[t*2]>heap[t*2+1]) r=t*2;
   else r=t*2+1; }

  //ako je samo levo dete u hipu, onda
 //r cuva njegov indeks
   else if (t*2<heap.size()) r=t*2;

 else break; //kraj ako heap[t] nema dece

 /*sledi provera da li je dete vece od

roditelja i ako jeste trampiti ih */
 if (heap[r]>heap[t])
  {heap[r]^=heap[t]; heap[t]^=heap[r];
   heap[r]^=heap[t]; t=r;}
  else break;
  }
}
```

II način

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
vector<int> heap;

int main()
{ int n,x;
   cin >> n;
  for(int i=0;i<n;i++)
```

```
{   cin >> x;
  heap.push_back(x); //umetanje x u vektor
 }
 make_heap (heap.begin(),heap.end());//kreiranje (by default) MAX hipa od vektora

 int m,y;
 cin >>m;
 for (int i=0;i<m;i++)
  {
    cin >> y;
    if (y==0)  //dolazi kupac
    {
      if(heap.size()>0)
      {   //skidanje max sa hipa
         cout << heap.front() << ' ';
         pop_heap (heap.begin(),heap.end());
         heap.pop_back();
      }
     else cout <<"Nema!";
   }else { //umetanje kvaliteta y u hip
      heap.push_back(y);
      push_heap (heap.begin(),heap.end());
      }
 }
  return 0;
}
```

2.

Tokom beogradskog maratona, odvija se i trka po kružnoj pisti vojnog aerodroma Batajnica u kojoj učestvuje K (2 <= K <= 10000) vojnika koji moraju da istrče N (1 <= N <= 1000) krugova trke. Svi trkači počinju trku u isto vreme i sa iste startne linije. Poznato je da uobičajena takmičarska forma pada tokom trke i da trkači nakon svakog punog kruga trče 1 milisekundu sporije. Tokom uobičajene forme, i –ti takmičar istrči jedan krug za ms_i milisekundi (1 <= ms_i <= 1000000). Pre početka trke svakom takmičaru se najavi ceo broj t_i (1 <= t_i <= N) koji ukazuje da će takmičar nakon svakog potpuno okončanog t_i kruga dobiti energetski stimulans čim pređe startnu liniju trke. Energetski stimulans omogućuje takmičaru da povrati potpuno svoju snagu, ali nakon toga takmičarska forma i izdržljivost nastavljaju da padaju na već opisani način. Uzimanje stimulansa zahteva 0 vremena. Napišite konzolnu aplikaciju, VOJNICI koja će ispisati maksimalni broj trkača koji će preći startnu liniju istovremeno u nekom periodu trke (istovremeno=posle jednakog broja milisekundi koje su protekle nakon početka trke). Jasno je da će svaki takmičar tokom istrčavanja N krugova preći startnu liniju N puta, jer se broji prelazak preko linije nakon svakog kruga, a ne broji se prelazak preko linije na početku trke.

Ulaz: Standardni ulaz sadrži sadrži nekoliko linija. U prvoj liniji su data dva cela broja K, N razdvojena blanko karakterom. U svakoj od narednih K linija zadati su dva cela broja ms_i, t_i razdvojena blanko karakterom.

Izlaz: Standardni izlaz treba da sadrži jednu liniju. Vaš program treba da ispiše maksimalan broj trkača koji će preći startnu liniju istovremeno u nekom periodu trke

***Primer*:** Ulaz	Izlaz
4 3 26 2 39 3 45 1 56 2	2

Pojašnjenje: Vojnici će redom trčati 3 kruga za sledeće vreme: vojnik1 (26, 27,26 ms); vojnik2 (39, 40 , 41 ms); vojnik3 (45, 45, 45 ms); vojnik4 (56, 57, 56 ms) Vojnici će preći startnu liniju za sledeće vreme: vojnik1 nakon 26, 53, 79 ms; vojnik2 nakon 39, 79, 120 ms; vojnik3 nakon 45, 90, 135 ms; vojnik4 nakon 56, 113, 169 ms. Dakle, vojnici 1 i 2 će nakon 79 milisekundi preći startnu liniju zajedno.

Resenje:

```
#include <cstdio>
#include <queue>
using namespace std;
#define TrkacMax 10000
#define KrugovaMax 1000
```

```
int s[TrkacMax+1], p[TrkacMax+1];
int k,n;


struct krugTrke
{
  int r;                    // broj trkaca
  int l;                    // broj kruga
  int vreme;                // vreme(milisekunde) mereno od pocetka trke kada trkac r
                            //prodje stratnu liniju nakon kruga
  bool operator <(const krugTrke &a) const
  {
    if (vreme>a.vreme)
      return true;
    else
      return false;
  }
};
priority_queue<krugTrke> pq;
//koristimo strukturu koja nam omogucuje da na vrhu strukture pq uvek bude max kandidat

void ucitaj()
{
  int i;
  krugTrke tekKrug;

  scanf("%d%d",&k,&n);
  for (i=1;i<=k;i++)
  {
    scanf("%d%d",&s[i],&p[i]);
    tekKrug.r=i;tekKrug.l=1;tekKrug.vreme=s[i];
    pq.push(tekKrug);
  }
}

int main()
{
  int cv=0,cvc=0,resenje=0; //cv, cvc pomocna vremena vojnika
  krugTrke tekKrug;
  ucitaj();

  while((!pq.empty()) && (resenje<k))
  {
    tekKrug=pq.top();
    if (cv != tekKrug.vreme)
      {
        cvc=1;
        cv=tekKrug.vreme;
      }
    else
      cvc++;
    if (cvc>resenje)
      resenje=cvc;
    pq.pop();
    if (tekKrug.l<n)
    {
      tekKrug.l++;
      if ((tekKrug.l%p[tekKrug.r]) > 0)
        tekKrug.vreme=tekKrug.vreme+s[tekKrug.r]+(tekKrug.l%p[tekKrug.r])-1;
      else
        tekKrug.vreme=tekKrug.vreme+s[tekKrug.r]+p[tekKrug.r]-1;
      pq.push(tekKrug);
    }
  }
```

```
 printf("%d\n",resenje);

 return 0;

}
```

3. Nedavno je Petko smislio novi izazov za Stanka– učestvovanje u takozvanom *cikličnom maratonu*. Ciklični maraton se organizuje pod sledećim pravilima. Maratonska staza je kružna staza dužine L. Jedna tačka staze se bira za Start/Cilj tačku. Svaki od N trkača je pozicioniran na stazi na rastojanju D u odnosu na Start/Cilj tačku mereno u smeru kretanja kazaljke na satu i svaki trkač dobija startni broj od 1 do N, počev od najbližeg u odnosu na Start/Cilj tačku u smeru kretanja kazaljke na satu. Nakon pucnja za početak trke, svaki trkač počinje da trči sopstvenom brzinom S u smeru kretanja kazaljke na satu. Ako trkač stigne drugog trkača ispred sebe, onda taj drugi trkač napušta maraton i kažemo da je eliminisan.

Takmičenje se završava kada nije moguće eliminisati trkače. Svi trkači koji su još uvek na stazi se proglašavaju za pobednike. Stanko je prihvatio izazov, ali želi da poveća svoje šanse za pobedu, birajući bolju poziciju na stazi. Napisati program koji će odrediti redosled u kom će trkači biti eliminisani.

Ulaz

Prva linija standardnog ulaza sadrži pozitivne cele brojeve N i L ($N \leq 500\,000$, $L < 5\,000\,000$). U svakoj i-toj liniji (od ukupno N linija koje slede) dat je parametar i-tog trkača – celobrojno rastojanje D_i, $0 \leq D_1 < D_2 < \ldots < D_N < L$, i brzina S_i, realni broj sa dve decimale, $0 < S_i \leq 5$. Sva rastojanja su data u metrima i brzine su date u metrima po sekundi.

Izlaz

Za svakog eliminisanog trkača, program treba da odštampa zasebnu liniju na standardnom izlazu sa startnim brojem trkača. Poslednja linija izlaza mora da sadrži string `"Winner(s):"`, jedan blanko i listu pobednika u rastućem redosledu i razdvojenih sa po jednim blanko karakterom.

Primer

Ulaz	**Izlaz**

`6 150`
`0 1.75`
`30 0.8`
`60 0.5`
`70 1`
`120 0.1`
`140 0.9`
	`2`
	`3`
	`5`
	`4`
	`6`
	`Winner(s): 1`

`Resenje:`

Zamislimo da je kretanje trkača pravolinijsko, na primer po pozitivnom delu realne prave i neka je Start/Cilj tačka nulta tačka realne prave. Zamislimo da vreme trčanja se takođe prati na realnoj vremenskoj pravoj i neka je početak trke u nultoj tački vremenske ose. U formulaciji zadatka je rečeno da se trkači kreću ravnomerno pravolinijski, brzinom s, započevši trku u tački sa koordinatom d.

Dakle, u određenom trenutku vremena $t \geq 0$, takmičar će dospeti u tačku s koordinatom $d + st$. Možemo izračunali kada će trkač sa rednim brojem i, koji je započeo trku u tački sa koordinatom d_i prestići trkača s rednim brojem $i + 1$, koji je započeo trku u tački s koordinatom d_{i+1}, $d_i < d_{i+1}$, uz uslov da je brzina s_i i-tog takmičara veća od brzine s_{i+1} takmičara s rednim brojem $i + 1$. To će se desiti kada takmičar sa rednim brojem i se nalazi na istoj lokaciji (tački) kao i trkač s rednim brojem $i + 1$, tj.

$$d_i + s_i t = d_{i+1} + s_{i+1} t$$

Odnosno, ekvivalentan izraz nakon transformacije

$$(s_i - s_{i+1})t = d_{i+1} - d_i ,$$

tj. vreme sustizanje je $t = (d_{i+1} - d_i)/(s_i - s_{i+1})$.

Dakle, prvi korak u algoritmu je pronaći sve parove trkača (r_i, r_{i+1}) sa uzastopnim rednim brojevima za koje je $s_i > s_{i+1}$ i da se izračuna vremenski trenutak u kom takmičar s rednim brojem i će prestići trkača s rednim brojem $i + 1$.

Pošto je trkačka staza kružna staza, potrebno je proveriti da li postoji još jedno sustizanje – trkača sa rednim brojem N i trkača sa rednim brojem 1, imajući na umu situaciju da trkač sa rednim brojem 1 započinje trku ne u tački sa koordinatom d_1, već u tački sa koordinatom $L + d_1$, gde je L data dužina kružne staze.
Sledeći koraci algoritma se odnose na obradu same trke, ali prethodno moramo da obezbedimo da u svakom vremenskom trenutku znamo koga sustiže trkač sa rednim brojem i. U tom cilju, na početku kreirajmo kružnu povezanu listu, tako da sledbenik i –tog element p_i je sledeći trkač koji će ga sustići i dajemo mu poziciju u spisku, $i + 1$. Kako je u pitanju kružna povezana lista (i trčanje po krugu), onda važi $p_N = 1$.

Sem toga, uvedimo jedan niz za svakog trkaca koji je prisutan u trci.
U algoritmu je kljucni sledeci korak, pri kom uredjujemo izracunata vremena za moguce preticanje, zajedno s brojevima dva trkača a i b na koje se odnosi prestizanje u binarnom hipu sa neredukovanim vrednostima vremena. Dakle, prvo preticanje će na biti vrhu hipa.
Poslednja faza algoritma sastoji se u simulaciji samog eliminisanja trkaca. Pogledajmo elemente koji se nalaze na vrhu hipa:

- Ako takmicar a u tački (k,a,b), koji je na vrhu hipa jeste već eliminisan - brišemo vrh hipa na uobičajeni način – zameni ga poslednji element i vratimo svojstvo heap-a, jer ovaj par ne može da uzrokuje preticanje u trci. Očigledno, nije moguće da trkac a bude još uvek u trci, a trkac b je eliminisan;
- Ako su oba igrača iz para (a,b) još uvek u trci, trkac b koji sustize će biti eliminisan. Zbog toga:

=Ako takmicar p_b još uvek trči i ako brzinu za a je veća od brzine za p_b, i ako potom trkač a postaje p_b, izračunati vreme k, kada će a uhvatiti novi p_a, zamena na vrhu heap-a trojke (k,a,p_a) i preuredjivanje heap-a;

= Ako gore navedeni uslovi nisu zadovoljeni, onda samo izbrisati zapis sa vrha steka.

Izvršite opisane korake eliminacije sa heap-a dok se ne isprazni heap. Tada svi trkaci koji nisu markirani kao eliminisani su pobednici.

```
#include <stdio.h>
#define MAXN 500001
// clan heap-a
typedef struct
{ double k; int ch,zn; //vremenski momenat kad a stigne b
  int a,b; } heap_spisak;

typedef struct // cvor koji opisuje trkaca r
{ double s; // brzina trkaca
  int s1,s2;
  int d,    // rastojanje od Start-Final tacke
      p;    // trkac ka kom se trci
} trkac;

int N,L;
heap_spisak h[MAXN];int hsz; // heap
trkac r[MAXN];        // spisak trkaca

int nzd(int a,int b)  //najveci zajednicki delilac
{  int r;
   if(a<b) {r=a;a=b;b=r;}
   while((r=a%b)!=0) {a=b;b=r;}
   return b;
}
void trampa(int i, int j)
//trampa i-tog i j-tog clana strukture
{  int t;
     t=h[i].ch; h[i].ch=h[j].ch;h[j].ch=t;
     t=h[i].zn; h[i].zn=h[j].zn;h[j].zn=t;
     t=h[i].a; h[i].a=h[j].a;h[j].a=t;
     t=h[i].b; h[i].b=h[j].b;h[j].b=t;
}
```

```
void popravi_hip(int i) //popravi implicitni heap, tj. niz
{  int j = i;

   if(2*i<=hsz && h[j].ch*h[2*i].zn>h[j].zn*h[2*i].ch) j=2*i;
   if(2*i+1<=hsz && h[j].ch*h[2*i+1].zn>h[j].zn*h[2*i+1].ch) j=2*i+1;
   if(j!=i) { trampa(i,j); popravi_hip(j); }
}

void kreirajHip() //kreiranje heap-a
{  int i;
   for(i=hsz/2;i>=1;i--) popravi_hip(i);
}

void uzmiTrkaca() //uzimanje clana hipa
{  int a,b,x,y,z,w;
   a=h[1].a;b=h[1].b;
   if(r[a].p!=0)
   {  //trkac b ce biti eliminisan
      printf("%d\n",b);
      r[a].p=r[b].p;r[b].p=0;
      if(a!=r[a].p && r[a].s1>r[r[a].p].s1)
      {  b=h[1].b=r[a].p; w=r[b].d;
         if(a>b) w+=L;
         x=100*(w-r[a].d);
         y=r[a].s1-r[b].s1;
         z=nzd(x,y);
         h[1].ch=x/z;
         h[1].zn=y/z;
      }
      else {
            h[1].ch=h[hsz].ch;h[1].zn=h[hsz].zn;
            h[1].a=h[hsz].a;h[1].b=h[hsz--].b;
            }
   }
   else
   {
      h[1].ch=h[hsz].ch;h[1].zn=h[hsz].zn;
      h[1].a=h[hsz].a;h[1].b=h[hsz--].b; }
   popravi_hip(1);
}

int main()
{  int i,x,y,z;
   //ucitavanje ulaza
   scanf("%d %d", &N,&L);
   for(i=1;i<=N;i++)
   {  scanf("%d %d.%d",&r[i].d,&r[i].s1,&r[i].s2);
      r[i].s1=100*r[i].s1+r[i].s2;
      r[i].p=i+1;
   }
   r[N].p=1;r[N+1].d=L+r[1].d;r[N+1].s1=r[1].s1;

   //inicijalizacija heap-a
   hsz=0;
   for(i=1;i<=N;i++)
   {  if(r[i].s1>r[i+1].s1)
```

```
    {
      x=100*(r[i+1].d-r[i].d);
      y=r[i].s1-r[i+1].s1;
      z=nzd(x,y);
      h[++hsz].ch=x/z;
      h[hsz].zn=y/z;
      h[hsz].a=i; h[hsz].b=i+1;
      if(i==N) h[hsz].b=1;
    }
  }

  kreirajHip();

  //eliminacija trkaca
  while(hsz>=1) {uzmiTrkaca();

}
  printf("Winner(s):");
  for(i=1;i<=N;i++) if(r[i].p!=0) printf(" %d",i);
  printf("\n");
}
```

4. U datoteci standardnog ulaza u prvoj liniji nalazi se broj N, ceo broj stranaka koje su učestvovale na izborima, zatim za svaku stranku u posebnim linijama njeno ime i broj osvojenih glasova. Kreirati program kojom se na osnovu sadržaja datoteke i ukupnog broja mandata određuje broj mandata koji se dodeljuje svakoj stranci po D'Ontovom sistemu.
Broj dobijenih glasova svake od tako odabranih lista deli se sa svim brojevima od 1 do K (K je broj mandata po izbornoj jedinici). Na taj je način svakoj listi pridruženo M (ne nužno celih) brojeva. Prvi od K mandata u toj izbornoj jedinici dobija lista koja ima najveći od svih tako dobivenih količnika, drugi mandat ona sa sledećim najvećim količnikom (to može biti i ista lista koja je dobila prvi mandat) i tako dalje dok se ne podele svi mandati.

U D'Ontovom sistemu za svaku stranku računamo količnik ukupnog broja glasova koje je ta stranka dobila i broj mandata koje je ta stranka dobila uvećan za 1. Na početku stranka dobija 0 mandata. Ona stranka koja ima najveći količnik dobija sledeći mandat i ponovo računamo količnik uzimajući u obzir mandate koji su dodeljeni. Postupak se nastavlja sve dok se ne raspodele svi mandati.

Objašnjenje:

Ilustrujmo D'Ontov sistem na primeru podele K=5 mandata za N=4 stranke A, B, C, D, koje su osvojile redom 200, 124, 100, 96 glasova. Formira se tabela tako što je u prvom redu broj glasova svake stranke, u sledećem redu polovine glasova svake stranke, u sledećem trećine itd. Tabela ima onoliko redova koliko se mandata deli, u našem primeru 5. U tabeli se zaokruže najveći količnici (brojevi), i to onoliko kolicnika koliko mandata delimo, a stranke čije smo količnike zaokružili dobijaju mandate. U našem primeru u tabeli smo zaokružili 5 najvećih količnika (200, 124,100, 100, 96) i stranke u čijim su kolonama ti količnici dobijaju mandate. Ovde stranka A dobija dva mandata, a ostale stranke po jedan.

A	B	C	D
(200)	(124)	(100)	(96)
(100)	62	50	48
66.6	41.3	33.3	32
50	31	25	24
40	24.8	20	19.2

Za svaku stranku korišćenjem strukture stranka pamtimo njeno ime, broj glasova i broj mandata koje smo joj za sada dodelili (na početku je to 0). Količnik i-te stranke je a[i].brglas/(a[i].brmandata+1). Stranka koja ima najveći količnik dobija još jedan mandat i u sledećem koraku za tu stranku posmatramo izmenjen količnik. Zadatak možemo rešiti korišćenjem hipa koji uređujemo prema

količnicima. Izdvajamo najveće količnike iz niza i to onoliko količnika koliko mandata tražimo. Prilikom izdvajanja najvećeg količnika broj mandata odgovarajuće stranke uvećamo za 1 i uredimo hip.

Rešenje:

```
#include<iostream>
using namespace std;

typedef struct stranka
{
char ime[20];
int brglas, brmandata;
}STRANKA;

STRANKA a[100];
int n;

int vecaStranka(STRANKA x, STRANKA y)
{
 if(x.brglas*(y.brmandata+1)>y.brglas*(x.brmandata+1))
 return 1;
else  return 0;
}


void ubaci(STRANKA x) //ubaci x u heap
{ n++;
  a[n]=x;
  int t=n;

    while (t/2 && vecaStranka(a[t],a[t/2]))
   { a[t]=a[t/2];
     a[t/2]=x;
      t=t/2;
   }
}

void pisi()
{
 for (int i = 1; i <=n; i++)
    cout<<a[i].ime <<" " <<a[i].brglas <<" "<<a[i].brmandata<<endl;
}

void urediHeap()
{
  int d, r=1;
  while (1)
 {
    if(2*r+1<=n)
     {if (vecaStranka(a[2*r],a[2*r+1]))
       d = 2 * r;
      else  d=2*r+1;
     }
  else
       if(2*r<=n)
         d=2*r;
       else break;

    if (vecaStranka(a[d],a[r]))
    {
       STRANKA pom=a[d];
       a[d]=a[r];
       a[r]=pom;
       r=d;
    }
   else break;
   }
}
```

```
int main ()
{
  STRANKA x;
  int br,k;
  cout<<" Broj stranki n";
  cin >> br;
  cout<<"ime_stranka broj_glasova\n";

 n=0;
for (int i = 1; i <= br; i++)
{
  cin>>x.ime>>x.brglas;
  x.brmandata = 0;
  ubaci(x);
}

cout<<" Broj mandata";
cin>>k;

for(int i=0;i<k;i++)
{
  a[1].brmandata++;
  urediHeap();
}
 pisi();
 cin>>k;
 return 0;
}
```

5. Dato je k sortiranih nizova takvih da svaki ima n elemenata. Konstruisati sto efikasniji algoritam koji spaja nizove i ispisuje sortirani izlaz.

Ulaz:

k = 3, n = 4

arr[][] = { {1, 3, 5, 7},
{2, 4, 6, 8},
{0, 9, 10, 11}} ;

Output: 0 1 2 3 4 5 6 7 8 9 10 11

Hint: http://poincare.matf.bg.ac.rs/~jelenagr/ASP/4cas.pdf
(zadatak 3, algoritam za spajanje 2 hipa veličine n, m)

JEDNOSTAVNO resenje

Kreirati izlazni niz velicine n*k kopiranjem svakog od k sortiranih nizova redom (bez smanjenja opstosti, pretpostavimo da su svi nizovi sortirani u rastucem poretku).

Sortirati izlazni niz koriscenjem nekog od O(n log n) algoritama za sortiranje. Dakle, ukupna vremenska slozenost je O(nkLognk).

EFIKASNIJE resenje

Spojimo k sortiranih nizova za O(nk*Logk) vremena koristeci Min Heap sledecim algoritmom.

1. Kreirati izlazni niz velicine n*k.

2. Kreirati min heap velicine k i ubaciti 1. element iz svakog niza u heap

3. Ponoviti sledece korake n*k puta.

a) Uzeti minimalni element iz heap-a (minimum je koren min-hipa) i smestiti ga u izlazni niz.

b) Zameniti koren heap-a sledecim elementom iz onog niza kom je pripadao koren hipa. Ako taj niz nema vise elemenata, zameniti vrednost u korenu heap-a sa „beskonacno“. Nakon zamene korena, izvrsiti popravku heap-a.

6. Osoba X je odabrala da opljačka zlataru.
U zlatari se nalazi **N** komada nakita i svaki komad nakita ima **težinu T_i** i **vrednost V_i**. X je sa sobom poneo **K** vreća i svaka može podneti **najveći teret M_i**. Sav svoj plen, X će odneti u tim vrećama. Međutim, ne želi u istu vreću staviti **više od jednog komada** nakita kako se ne bi oštetili u transportu.
Koja je najveća vrednost tereta koju X može odneti?

ULAZNI PODACI

U prvom redu nalaze se prirodni brojevi **N** i **K** ($1 \leq N, K \leq 300\,000$).
U sledećih **N** redova nalaze se parovi brojeva T_i i V_i ($1 \leq T_i, V_i \leq 1\,000\,000$).
U sledećih **K** redova nalaze se brojevi M_i ($1 \leq M_i \leq 100\,000\,000$).

IZLAZNI PODACI

U jedini red ispišite najveću moguću zaradu.

PRIMERI TEST PODATAKA

ulaz
2 1
5 10
100 100
11
izlaz
10

ulaz
3 2
1 65
5 23
2 99
10
2
izlaz
164

Pojašnjenje drugog primera: X stavlja prvi komad nakita u drugu vreću i treći komad nakita u prvu vreću.

Resenje:
Zadatak može da se reši jednostavnim pohlepnim algoritmom. Sortiramo komade nakita po ceni. Zatim krećemo od najskupljeg komada prema najjeftinijem i radimo sledeće:
-ako postoji, uzmi najmanju vreću čija je nosivost veća od težine trenutnog komada nakita. Izbaci tu vreću iz skupa vreća i pridodaj cenu trenutnog komada u rešenje
-ako ne postoji takva vreća, preskoči ovaj komad nakita i nastavi dalje
Za implementaciju ovog algoritma potrebna je struktura podataka koja podržava tri operacije: ubaci broj, nađi prvi broj veći od nekog broja x ili javi da ne postoji, izbaci neki broj. Oni koji programiraju u C++ mogu iskoristiti gotovu strukturu - *multiset*.

Ili algoritam može da koristi i potrebno sortiranje i binarni heap što je donekle lakše implementirati.
Binarni heap je struktura koja podržava tri operacije: ubaci broj, reci koji je najveći broj, izbaci najveći broj.
Algoritam je sledeći:
-sortiraj nakit i vreće po težini zajedno u jedan niz
-kreni od predmeta najmanje težine prema predmetu najveće težine
-kada naiđeš na komad nakita, ubaci njegovu vrednost u heap
-kada naiđeš na vreću, ako heap nije prazan, uzmi najskuplji komad nakita na koji si naišao do sad (on će sigurno imati manju težinu zbog načina na koji smo sortirali) i izbaci ga iz heapa.
Njegovu vrednost pribroji u trenutno rešenje.

```
#include <cstdio>
#include <algorithm>
#include <set>

using namespace std;
```

```
typedef long long llint;

const int MAXN = 300100;

int N, K;
pair<int, int> nakit[MAXN];
multiset<int> M;

bool cmp(const pair<int, int> &A, const pair<int, int> &B) {
	if(A.second != B.second) return A.second > B.second;
	return A.first < B.first;
}

int main(void) {

	scanf("%d%d", &N, &K);

	for(int i = 0; i < N; ++i)
		scanf("%d%d", &nakit[i].first, &nakit[i].second);

	sort(nakit, nakit + N, cmp);

	for(int i = 0; i < K; ++i) {
		int x; scanf("%d", &x);
		M.insert(x);
	}

	llint ans = 0;

	for(int i = 0; i < N; ++i) {
		if(M.empty()) break;
		if(M.lower_bound(nakit[i].first) != M.end()) {
			ans += nakit[i].second;
			M.erase(M.lower_bound(nakit[i].first));
		}
	}

	printf("%lld\n", ans);

	return 0;

}
```