Rendezett Halmaz adattípus

Horváth Gyula
horvath@inf.elte.hu

6.1. Halmaz (set)

Értékhalmaz: $Halmaz = \{ H : H \subseteq E \}$, E-n értelmezett a $<$ lineáris rendezési reláció.
Jelölés: $Adat(it)$ az it iterátor által hivatkozott eleme a halmaznak (csak olvasásra).
Műveletek argumantumai: $H : Halmaz, x : E, it : Iterator$

Előfeltétel	Művelet	Utófeltétel
$x \notin H$	Bővít(H,x)	$H = Pre(H) \cup \{x\}$
$x \in H$	Bővít(H,x)	$= it \wedge x = Adat(it) \wedge H = Pre(H)$
$H = H$	Töröl(H,x)	$H = Pre(H) - \{x\}$
$x \in H$	Keres(H,x)	$= it : Adat(it) = x) \wedge (x \notin H \Rightarrow it = H.end)$
$x \notin H$	Keres(H,x)	$= it : it = H.end$
$H = H$	ÜresE(H)	$= (H = \emptyset) \wedge H = Pre(H)$
$H = \{a_1, \ldots, a_n\}$	Elemszám(H)	$= n \wedge H = Pre(H)$
$H \neq \emptyset$	Első(H)	$= it : Adat(it) = \min(H) \wedge Pre(H) = H$
$H \neq \emptyset$	Utolsó(H)	$= it : Adat(it) = \max(H), \wedge Pre(H) = H$
$H \neq \emptyset$	Előző(H,x)	$= it : Adat(it) = \max\{y \in H : y < x\} \wedge Pre(H) = H$
$H \neq \emptyset$	Követő(H,x)	$= it : Adat(it) = \min\{y \in H : x < y\} \wedge Pre(H) = H$
$H = H$	Üresít(H)	$H = \emptyset$

6.2. A Halmaz C++ és C# műveletei

Művelet	C++	C#
Típusdef	$set<E>$	$SottedSet<E>, SortedSet<E>(rendrel)$
Bővít(H,x)	$H.insert(x)$	$H.Add(x)$
Bővít(H,x)	$H.insert(x)$	$H.Add(x)$
Töröl(H,x)	$H.erase(x)$	$H.Remove(x)$
Töröl(H,it)	$H.erase(it)$	$-$
Keres(H,x)	$H.find(x)$	$H.Contains(x)$
ÜresE(H)	$H.empty()$	$H.Count == 0$
Elemszám(H)	$H.size()$	$H.Count$
Első(H)	$H.begin()$	$H.Min$
Utolsó(H)	$--H.end()$	$H.Max$
Követő(H,x)	$H.upper\_bound(x)$	$H.GetViewBetween(x,H.Max).Min$
Üresít(H)	$H.clear()$	$H.Clear()$

6.2.1. Megjegyzés

C++ esetén a $H.insert(x)$ művelet olyan it iterátort ad, amelynek típusa

$$pair < set < E >:: iterator, bool >$$

tehát egy párt, amelynek második tagja, az $it.second$ akkor és csak akkor igaz érték, ha x a művelet előtt nem volt eleme H-nak. it első tagja, az $it.first$ olyan (csak olvasható) iterátor, amely a H halmaz azon elemére mutat, amelynek értéke x, akár eleme volt a művelet előtt H-nak x, akár nem.
C++ nem ad Előző műveletet, helyette a $lower\_boud$ művelet van, ami olyan it iterátort ad eredményül, amely az x-nél nagyobb vagy egyenlő elemek minimumára mutat:

$$H.lower\_boud(x) = it : Adat(it) = \min\{y \in H : x \leq y\}$$

Tehát a Előző(x) az alábbi módon valósítható meg:

```
1       it=H.lower_boud(x);
2       if(it==H.begin()) //nincs x-nél kisebb
3           it=H.end();
4       else                    //x<=*it
5           --it;
```

$$H.upper\_boud(x) = it : Adat(it) = \min\{y \in H : x < y\}$$

Tehát a Koveto(H,x) megvalósítható:

```
1       it=H.upper_boud(x);
2       y=*it
```

6.3. Rendezési reláció megadása rendezett halmazhoz

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4  struct Par{
5     int adat, poz;
6      bool operator < (const Par& jobb) const{
7         return adat<jobb.adat;
8      }
9  };
10 bool rendez(const Par& x, const Par& y){return x.adat > y.adat;}
11
12 int main(){
13    set<Par> H1;
14    set<Par, decltype(&rendez)> H2(&rendez);
15 }
```

6.4. Feladat: Mérési adatok

Egy kísérletben nagyszámú mérést végeztek, minden mérés eredménye egy természetes szám. A kutatók a mérési sorozatot elemzik. Szeretnék tudni, hogy hány különböző érték keletkezett és ezeknek mi a mediánja.

Bemenet

A standard bemenet első sora a mérések n ($1 \leq n \leq 100\,000$) számát tartalmazza. A második sor tartalmazza az n mérési eredményt, ei ($1 \leq ei \leq 1\,000\,000\,000$).

Kimenet

A standard kimenet első sorába kell kiírni a mérési sorban szereplő különböző értékek számát és ezek mediánját.

Példa bemenet és kimenet

bemenet	**kimenet**
10 12 2 13 12 20 6 10 4 13 2	7 10

Korlátok

Időlimit: 0.2

Memória limit: 32 MiB

6.5. Megoldás

Legyen $(e_1, \ldots, e_i, \ldots e_n)$ a bemeneti mérési eredmények sorozata. Rakjuk be a sorozat elemeit egy M halmazba. Minden adat csak egyszer szerepel majd M-ben mert M nem multihalmaz. Tehát M elemszáma adja az első kérdésre a választ. A mediánt úgy határozhatjuk meg, hogy sorbavesszük M elsemeit (növekvően) és íg a középső lesz a medián.
Az algoritmus futási ideje $O(n * \lg m)$ lesz, ahol m a különböző mérési adatok száma, mivel minden bővítés legfeljebb $O(\lg m)$ időt igényel.

6.5.1. C++ megvalósítás

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  // mérési adat
4  int main (){
5     set<int> M;
6     int n,x;
7     cin>>n;
8     for(int i=1;i<=n;i++){
9        cin>>x;
10       M.insert(x);
11    }
12    int medindex=M.size()/2;
13    int im=0;
14    int median=0;
15    for(int m : M){
16       im++;
17       if(im>medindex){
18         median=m;   break;
19       }
20    }
21    cout<<M.size()<<" "<<median<<endl;
22   return 0;
23 }
```

6.5.2. C# megvalósítás

```
1  using System; using System.Collections.Generic;
2  class Program{
3     public static void Main(){
4        int n,x;
5        n=Convert.ToInt32(Console.ReadLine());
6        var M = new SortedSet<int>();
7        string[] sor=Console.ReadLine().Split();
8        for(int i=0;i<n;i++){          //adatok be
9           x=Convert.ToInt32(sor[i]);
10          M.Add(x);
11       }
12       int medindex=M.Count/2;
13       int median=0;
14       int im=0;
15       foreach (var m in M){
16          im++;
17          if(im>medindex){
18             median=m; break;
19          }
20       }
21       Console.WriteLine(M.Count+" "+median);
22    }
23 }
```

6.6. Feladat: Raktár rendezés

Egy raktárban érkezési sorrendben egyetlen sorban tárolják a konténereket. Mindegyik konténer egy konténerhelyet foglal el a méretétől függetlenül. Konténereket egymásra lehet rakni, de egyetlen konténer sem rakható nálánál kisebbre. Minden konténer beérkezéseker el kell dönteni, hogy a sorban az utoló után rakjuk, vagy rérakjuk egy konténer oszlop tetejére.

Írjunk olyan programot, amely kiszámítja, hogy legkevesebb hány oszlopba rakható le egy konténers

Bemenet

A standard bemenet első sora a konténerek n ($1 \leq n \leq 100\,000$) számát tartalmazza. A második sor tartalmazza az n konténer m_i méretét ($1 \leq m_i \leq 10\,000$).

Kimenet

A standard kimenet első sorába azt az m számot kell írni, ami a leglevesebb oszlopok száma, amibe lerakhatók a konténerek!

Példa bemenet és kimenet

bemenet	**kimenet**
10 12 2 13 12 20 6 10 4 5 2	3

Korlátok

Időlimit: 0.2

Memória limit: 32 MiB

6.7. Megoldás

Balról jobra haladva, az aktuális m_i méretű konténerről el kell dönteni, hogy rárakható-e valamely tőle balra lévő konténer oszlopra. Legyenek az i-edik konténertől balra lévő konténer oszlopok legfelső konténer-méreteinek halmaza $T = \{t_1 < \ldots < t_k\}$. Ha $t_k < m_i$, akkor egyikre sem helyezhető, tehát új tornyot nyitunk, azaz m_i-t hozzávesszük a T halmazhoz. Egyébként, legyen t_j az a legkisebb eleme T-nek, amelyre $t_j \leq m_i$ teljesül. Tegyük T-ben t_j helyébe m_i-t. Tehát az i-edik konténert a arra a toronyra rakjuk, amelynek teteje t_j

A megvalósításhoz a Halmaz adattípus lower_bound illetve $FindGreaterEqual$ műveletét használjuk

6.7.1. C++ megvalósítás

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  int main(){
6    int n,x;
7    set<int> T;
8    set<int>::iterator it;
9    cin>>n;
10   for(int i=0;i<n;i++){
11     cin>>x;
12     it=T.lower_bound(x);
13     if(it!=T.end()) T.erase(it);
14     T.insert(x);
15   }
16   cout<<T.size()<<endl;
17 }
```

6.7.2. C# megvalósítás

```
1  using System; using System.Collections.Generic;
2  class Program{
3     public static void Main(){
4        int n,x;
5        n=Convert.ToInt32(Console.ReadLine());
6        var T = new SortedSet<int>();
7        string[] sor=Console.ReadLine().Split();
8        T.Add(Convert.ToInt32(sor[0]));
9        for(int i=1;i<n;i++){          //adatok be
10          x=Convert.ToInt32(sor[i]);
11          if(x>T.Max)
12             T.Add(x);
13          else{
14             int kovet=T.GetViewBetween(x, T.Max).Min;
15             if(x!=kovet){
16                T.Remove(kovet);
17                T.Add(x);
18             }
19          }
20       }
21       Console.WriteLine(T.Count);
22    }
23 }
```

6.8. Feladat: Parti

Egy partin sok vendég vett részt. mindegyikről feljegyezték, hogy mikor érkezett és mikor távozott. Ezen adatok alapján megy kell adni egy olyan időpontot, amikor a legtöbben voltak jelen, és azt is hogy kik.

Bemenet

A standard bemenet első sora a vendégek n ($1 \leq n \leq 100\,000$) számát tartalmazza. A további n sor mindegyike egy vendég nevét, érkezési és távozási idejét tartalmazza.

Kimenet

A standard kimenet első sorába két egész számot kell írni. A első szám egy olyan idópont legyen, amikor a legtöbben voltak jelen, a második pedig a jelenlévők száma m legyen. A következő m sor egy-egy jelenlévő nevét tartalmazza!

Példa bemenet és kimenet

bemenet

```
6
Anna 1 8
Ildi 2 6
Botond 3 8
Vera 2 7
Gyula 9 12
Gabi 2 8
```

kimenet

```
3 5
Anna
Botond
Gabi
Ildi
Vera
```

6.9. Megoldás

Növekvő sorrendben vegyük az időpontokat, amikor valaki érkezett, vagy távozott. Az azonos időpontok esetén előbb vegyük az érkezőt. Egy $Jelen$ halmazban tároljuk az adott időpontban jelenlévők nevét. Érkezés esetén rakjuk a $Jelen$ halmazba az érkező nevét, távozás esetén vegyük ki a halmazból a nevét. Ha a $Jelen$ halmaz elemszáma nagyobb, mint ami eddig volt, akkor jegyezzük fel, mint az eddigi legjobb megoldás.

6.10. C++ megvalósítás

```
#include <bits/stdc++.h>
using namespace std;
struct Adat{
  string ki;
  int mikor;
  bool be;
  Adat(){};
  Adat(string x, int y, bool z){ki=x; mikor=y; be=z;}
};
bool rend_rel (const Adat &a , const Adat &b) {
      return a.mikor<b.mikor || a.mikor==b.mikor && a.be;
   }
int main (){
  int n,tol,ig;
  string nev;
  cin>>n;
  Adat A[2*n];
  for(int i=0;i<n;i++){
    cin>>nev>>tol>>ig;
    A[i]=Adat(nev, tol, true);
    A[i+n]=Adat(nev, ig, false);
  }
```

```
23    sort(A, A+2*n, rend_rel);
24    set<string> Jelen;          //a jelenlévők halmaza
25    set<string> Kik;            //a legtöbb jelentlévő
26    int hanyan=0;               //a legtöbb jelentlévő száma
27    int mikor=0;                //ekkor voltak jelen legtöbben
28    for(int i=0;i<2*n;i++){
29      if(A[i].be) Jelen.insert(A[i].ki);// érkező
30      if(Jelen.size()>hanyan){          //jobb megoldást találtunk
31        hanyan=Jelen.size();
32        mikor=A[i].mikor;
33        Kik=Jelen;
34      }
35      if(!A[i].be)                    //távozó
36        Jelen.erase(A[i].ki);
37    }
38    cout<<mikor<<" "<<hanyan<<endl;
39    for(auto x : Kik)
40      cout<<x<<endl;
41    return 0;
42  }
```

6.11. C# megvalósítás

```
1  using System;
2  using System.Collections.Generic;
3
4  class Program{
5     public class Adat{
6        public string ki;
7        public int mikor;
8        public bool be;
9        public Adat(string x, int y, bool z){ki=x; mikor=y; be=z;}
10    }
11    public static void Main(){
12       int n;
13       n=Convert.ToInt32(Console.ReadLine());
14       string[] sor;
15       var A =new Adat[2*n];
16       for(int i=0;i<n;i++){// adatok beolvasásáa
17          sor=Console.ReadLine().Split();
18          A[i]=new Adat(sor[0], Convert.ToInt32(sor[1]), true);
19          A[i+n]=new Adat(sor[0], Convert.ToInt32(sor[2]), false);
20       }
```

```
21        //rendezés időpont érkezés/távozás szerint
22        Array.Sort(A,delegate (Adat x, Adat y){ if (x.mikor<y.mikor ||
23                   x.mikor==y.mikor && x.be) return -1; else return 1;});
24        var Jelen = new SortedSet<string>();//a jelenlévők halmaza
25        string[] Kik=new string[0];          //a legtöbb jelentlévő
26        int hanyan=0;                        //a legtöbb jelentlévő száma
27        int mikor=0;                         //ekkor voltak jelen legtöbben
28        for(int i=0;i<2*n;i++){
29           if(A[i].be) Jelen.Add(A[i].ki);   // érkező
30           if(Jelen.Count>hanyan){           //jobb megoldást találtunk
31              hanyan=Jelen.Count;
32              mikor=A[i].mikor;
33              Kik=new string[hanyan];
34              Jelen.CopyTo(Kik);
35           }
36           if(!A[i].be)                      //távozó
37              Jelen.Remove(A[i].ki);
38        }
39        Console.WriteLine(mikor+" "+hanyan);
40        foreach(var x in Kik)
41           Console.WriteLine(x);
42     }
43  }
```

6.12. Multihalmaz (multiset)

A C++ STL könyvtára tartalmaz multihalmaz megvalósítást is. A Multihalmaz (multiset) rendelkezik mindazon műveletekkel, amelyekkel a Halmaz (set), de a Bővit$(insert)$ művelettel hozzávehetünk tetszőleges számszor olyan elemet, ami már korábban eleme lett a halmaznak. A Töröl$(x)(erase)$ művelet pedig törli x összes példányát a halmazból. A $Keres(x)(find)$ művelet eredménye két iterátor, az x elem első és utolsó előfordulására mutató iterátorok.
Van egy új művelet is a Megszámol$(x)(count)$, ami az x elem halmazbeli multiplicitását adja meg.
Értékhalmaz: $Halmaz = \{ H : H \subseteq E \}$, E-n értelmezett a $<$ lineáris rendezési reláció.
Műveletek: $H : Halmaz, x : E, it : Iterator$

Előfeltétel	Művelet	Utófeltétel
$H = H$	Töröl(H,x)	$H = Pre(H) - \{x\}$
$H = H$	Keres(H,x)	$= (it1, it2) \wedge (\forall it)(Adat(it) = x) \Rightarrow it1 \leq it \leq it2)$
$H = H$	Megszámol(H,x)	x multiplicitása $H - ban$

6.13. A Multihalmaz C++ műveletei

Művelet	C++	Időigény
Típusdef	$multiset < E >$	
Töröl(H,x)	$H.erase(x)$	$=$ (x multiplicitása) $* O(\lg n)$
Keres(H,x)	$H.find(x)$	$O(\lg n)$
Megszámol(H)	$H.count(x)$	$O(\lg n + m)$