Fényképezkedés

Egy rendezvényre sok 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.

Feladat

Írjunk olyan programot, amely kiszámítja, hogy legkevesebb hány fényképet kell készíteni, és megadja azokat az időpontokat is amikor csoportképet kell készíteni!

Bemenet

A standard bemenet első sorában a vendégek n száma van ($1 \leq n \leq 3000000$). A következő n sor mindegyike két egész számot tartalmaz egy szóközzel elválasztva, egy vendég e érkezési és t távozási időpontját ($1 \leq e < t \leq 100000$). 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 standard 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	Kimenet
6 2 4 1 4 2 7 7 13 5 10 3 9	2 3 9

Korlátok

Időlimit: 0.1 mp.
Memórilimit: 32 MiB
Pontozás: a tesztesetek 40%-ában $n < 1000$

Megoldás

Tehát a bemenet balról zárt és jobbról nyitott intervallumok egy

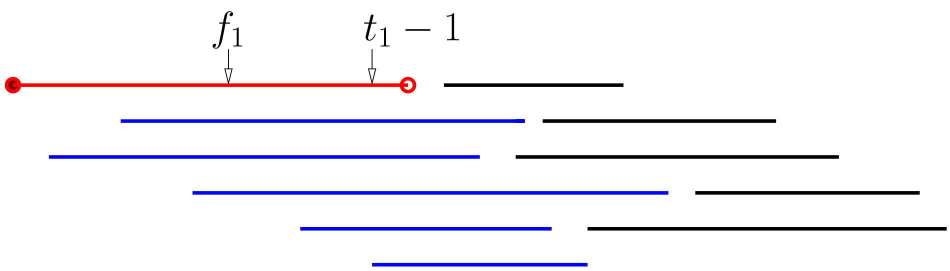
$$I = \{[e_1, t_1), \ldots, [e, t_n)\}$$

halmaza, mivel ha egy vendég távozási idejében készítenek fényképet, azon a vendég nem lesz rajta. 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 $e_i \leq f < t_i$.

A megoldás elemzése.
Tegyük fel, hogy az intervallumok jobb-végpontjuk szerint növekvően rendezettek, tehát $t_i < t_{i+1}$, $i = 1, \dots, n-1$ és az M megoldáshalmazra $f_1 < \dots < f_k$.
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. A redukált részprobléma a fekete intervallumokat tartalmazza.

Tehát a redukált részproblémát úgy kapjuk, hogy 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 $M' = \{f_2, \dots, f_k\}$ 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 k.
Tehát a helyes algoritmus:

```
rendezzük a bemeneti intervallumokat a jobb végpontjuk szerint növekvöen;
Utolso:=0;
ciklus minden [e,t) intervallumra
    ha Utolso<e akkor
        tegyük be t-1 -et a megoldáshalmazba;
        Utolso:=t-1;
    elágazás vége
ciklus vége
```

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 $f \in [e_i, t_i)$. Tehát a rendezés megspórolható úgy, hogy a bemeneti intervallumokat ábrázoljuk egy $Int[]$ tömbbel, úgy, hogy ha $[e, t)$ eleme a bemenetnek, akkor $Int[t] = e$ egyébként, ha nincs olyan bemenet, amelynek jobb végpontja t, akkor $Int[t] = 0$.

Megvalósítás C++ nyelven

```cpp
#include <iostream>
#define maxN 3000000
#define maxT 100000
using namespace std;

int Int[maxT];
int M[maxT];

int main(){
   int n, e,t;
   cin>>n;
   for(int x=1;x<=maxT;x++) Int[x]=0;
   for(int i=0;i<n;i++){
       cin>>e>>t;
       if(e>Int[t])
         Int[t]=e;
   }
   int Utolso=0, k=0;
   for(int x=1; x<=maxT; x++)
      if(Int[x]>0 && Utolso<Int[x]){
         Utolso=x-1;
         M[k++]=Utolso;
      }
   cout<<k<<endl;
   for(int i=0;i<k;i++)
      cout<<M[i]<<" ";
   cout<<endl;
 return 0;
}
```