АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА СЪСТЕЗАНИЕ

Наивна идея за решаване на задачата може да бъде да се пресметне във всяка милисекунда от състезанието броят на състезателите, пресичащи стартовата линия (този брой може да бъде и 0) и да се намери максималната от тези стойности.

Тази идея изисква масив, в който елемент с индекс *i* съответства на милисекунда с номер *i* от началото на състезанието. Първоначално стойностите на масива трябва да се занулят, а след това, пресмятайки за всеки състезател и за всяка негова обиколка в коя милисекунда ще завърши, в елемента на масива с индекс, равен на тази милисекунда, да се добавя 1. След като тези сметки се направят за всички състезатели и всички обиколки, в този масив трябва да се намери максималната стойност на негов елемент. Тази стойност дава отговора на задачата. Сложността на такова решение е O(max(K*N, Tmax)), където Tmax е броя милисекунди, които е продължило състезанието. Ясно е, че масивът ще трябва да съдържа Tmax на брой елементи. Лошото на това решение е, че Tmax може да достигне стойност 1 000 000 000 (поради ограниченията $\mathbf{1} \leq N \leq \mathbf{1\,000}$ и $\mathbf{1} \leq ms_i \leq \mathbf{1\,000\,000}$), което го прави приложимо само за онези 20% от тестовете, в които $\mathbf{1} \leq ms_i \leq \mathbf{100}$. Такова решение ще получи 20 точки.

Решение, което да удовлетворява ограниченията по памет и по време може да се получи по следния начин.

Да си представим, че сме изчислили моментите на пресичане на стартовата линия от всеки състезател след всяка обиколка и сме получили K*N числа, записани в масив, който е сортиран в ненамаляващ ред. Очевидно в този масив не може да има равни елементи, които да отразяват моментите на пресичане на стартовата линия от един и същи състезател след различни обиколки. Тогава, няколко равни елемента в сортирания масив (те ще бъдат съседни) ще съответстват на едновременното пресичане на стартовата линия от различни състезатели. Най-големият брой на последователни равни елементи в този „въображаем" масив ще ни даде търсения максимален брой състезатели.

Разбира се, при практическата реализация няма нужда да се създава масив, а само да се симулира неговото получаване, като се броят получаваните последователни равни елементи и текущо се определя най-големият брой такива. За целта може да се използва пирамида, в която, на всяка стъпка, се пазят моментите (в милисекунди), в които всеки състезател следващ път ще пресече стартовата линия (пирамидалната структура е организирана именно по тези моменти, като във върха на пирамидата стои следващият най-близък момент, в който някой състезател ще пресече стартовата линия). Заедно с числото, даващо момента на пресичане на стартовата линия, във всеки елемент на пирамидата трябва да се пази номерът на състезателя, който ще я пресече и номерът на обиколката, която ще бъде завършена при това пресичане. Първоначално в пирамидата се създава по един елемент за всеки състезател, съдържащ момента, в който той ще пресече стартовата линия след като завърши първата си обиколка. На всяка следваща стъпка се изпълнява следното:

- милисекундите (момент на пресичане на стартовата линия) от елемента във върха на пирамидата се сравняват с променлива, в която се пазят милисекундите – момент на предното пресичане на стартовата линия от някой състезател.
- ако е налице равенство, то текущият брой на едновременно пресекли стартовата линия състезатели се увеличава с едно, а иначе се прави равен на едно; сравнява се този текущ брой с намерения до момента максимален брой едновременно пресекли стартовата линия

състезатели и, ако текущият е по-голям, неговата стойност се записва в максималния.

- маха се елементът от върха на пирамидата, а в нея се добавя нов елемент, в който се записва изчисленият момент, когато състезателят от махнатия „връх” на пирамидата ще пресече стартовата линия след следващата си обиколка.

Работата продължава докато се изпразни пирамидата или докато се получат K последователни равни елемента (това ще означава, че е получен момент, в който K състезатели са пресекли едновременно стартовата линия, а те повече не могат и да бъдат).

Сложността на това решение се определя от това, че всеки момент на пресичане на стартовата линия от състезател след поредната му обиколка трябва да бъде добавен, а след това и премахнат от пирамидата. Тези моменти са точно K*N, пирамидата съдържа най-много K елемента, а добавянето и премахването на елемент в нея е със сложност logM, където M е броят на елементите и. Общата сложност на решението е O(K*N*logK). Необходимата памет се определя от масива, в който се реализира пирамидата и е много по-малка от ограничението от 1 MB.

```
#include <cstdio>

const int maxrunners=10000;   // maximum number of runners
const int maxlaps=1000;       // maximum number of laps

int s[maxrunners+1];
int p[maxrunners+1];

const int heapsize=1<<14;

struct he                     // heap element structure
{
  int r;                      // number of runner
  int l;                      // number of lap
  int crossingtime;           // time (in ms) from the beginning when
// runner #r will cross starting line after lap #l
};

he h[heapsize];
int uh=0;

int k,n;


void addtoheap(he ne)
{
  int child, parent;
  uh++;
  h[uh]=ne;
  child=uh;
  parent=child/2;
  while (parent>=1)
    if (h[parent].crossingtime > ne.crossingtime)
    {
      h[child]=h[parent];
      child=parent;
      parent=child/2;
    }
    else
      break;
  h[child]=ne;
}
```

```
void removefromheap()
{
  int parent, child;
  he ne;
  h[1]=h[uh];
  uh--;
  parent=1;
  ne=h[1];
  child=2*parent;
  while (child <= uh)
  {
    if (child<uh)
      if (h[child+1].crossingtime < h[child].crossingtime)
        child++;
    if (ne.crossingtime > h[child].crossingtime)
    {
      h[parent]=h[child];
      parent=child;
      child=2*parent;
    }
    else
      break;
  }
  h[parent]=ne;
}

void input()
{
  int i;
  he wne;

  scanf("%d%d",&k,&n);
  for (i=1;i<=k;i++)
  {
    scanf("%d%d",&s[i],&p[i]);
    wne.r=i;wne.l=1;wne.crossingtime=s[i];
    addtoheap(wne);
  }
}

int main()
{
  int i,j,ct;
  int cv,cvc,maxvc;
  he wne;

  input();
  cvc=0;
  maxvc=0;
  cv=0;

  while((uh>0) && (maxvc<k))
  {
    if (cv != h[1].crossingtime)
    {
      cvc=1;
      cv=h[1].crossingtime;
    }
    else
      cvc++;
    if (cvc>maxvc)
       maxvc=cvc;
    i=h[1].r;j=h[1].l;ct=h[1].crossingtime;
    removefromheap();
    if (j<n)
    {
```

```
      wne.r = i;
      wne.l = j+1;
      if (((j+1)%p[i]) > 0)
        wne.crossingtime=ct+s[i]+((j+1)%p[i])-1;
      else
        wne.crossingtime=ct+s[i]+p[i]-1;
      addtoheap(wne);
    }
  }

  printf("%d\n",maxvc);

  return 0;
}
```

Автор: Руско Шиков