Државно такмичење из програмирања, Београд – 26. април 2014 . II категорија

1. Напишите програм, конзолну апликацију, PROSTI, који пребројава колико има простих бројева који нису мањи од датог броја *a* и нису већи од датог броја *b*. Програм треба да испише резултат за датих *k* интервала, тј. парова бројева *a* и *b*.

Улаз: Стандардни улаз садржи садржи неколико линија. У првој линији стандардног улаза задат је цео број *k.* У наредних *k* линија улаза дата су два броја *a* и *b (*раздвојени бланко карактером) који представљају границе датих *k* интервала ($1 \le a \le b \le 1000000$, $1 \le k \le 1000$).

Излаз: Стандардни излаз треба да садржи k линија, тако i-ти ред садржи број простих бројева i-тог интервала са улаза.

Пример:улаз	излаз
3	8
2 20	17
20 100	25
1 100	

Објашњење: прости бројеви у интервалу [2,20] су {2,3,5,7,11,13,17,19}

Решење:

С обзиром на релативно велике вредности улазних параметара $1 \le a \le b \le 1000000$, $1 \le k \le 1000$, није било ефикасно радити тест прималности за сваки могући број из сегмента [а,б]. То нарочито важи за тест примере у којима неки од датих к сегмената [а,б] нису дисјунктни. Задатак се могао ефикасније решити употребом Ератостеновог сита, познатог поступка који налази све просте бројеве до неког броја.

```
#include <iostream>
using namespace std;
#define M 1000

const int N=M*M+1;


int sito[N] = {1,1,0,0};

int prostint[N];


int main()
{
 for(int i=4; i<N; i=i+2) sito[i]=1;

 int p=3;
 while(p<M)
 {
  for(int k=2*p; k<N; k=k+p) sito[k] = 1;


  p=p+2; while(sito[p]==1) p=p+2;
 }

 for(int i=2; i<N; i++)
  if(sito[i]==0) prostint[i]=prostint[i-1]+1;
  else prostint[i]=prostint[i-1];

 int k,a,b;
 cin >> k;
 for(int i=0; i<k; i++)
 { cin >> a >> b;
  cout << prostint[b]-prostint[a-1] << endl;
 }

 return 0;
}
```

2. Пастир Васа жели да спасе своју јагњад пред надолазећи талас поплава у Србији. Након дугог дана хода, Васа мора да смести своје стадо у штале како би сви мирно провели ноћ. Штале су ограниченог капацитета, јер у сваку шталу може стати највише *K* јагњади. Не мора свака штала бити потпуно пуна, чак неке штале могу остати и потпуно празне. Важно је да се свако јагње смести у шталу. Ради једноставности, представимо поље на ком стадо пасе као праву линију, јагњад као N

тачака на правој са целобројним координатама, штале као M тачака на правој са целобројним координатама. Могуће је да неколико јагњади, неколико штала, или неколико јагњади и штала дели исту координату. Свако Васино јагње прелази јединицу растојања за једну секунду (на пример, јагњету на координати 42 потребно је 12 секунди да дође до штале на координати 30 и 10 секунди да стигне до штале на координати 52). Напишите програм, конзолну апликацију, PASTIR који израчунава најмање време за које се стадо може сместити у штале. Наравно, јагњад се могу померати истовремено.

Улаз: Стандардни улаз садржи садржи једну линију са три цела броја раздвојених бланко карактером:
број јагњади *N*, број штала *M*, максимални број јагњади у једној штали $K(1 \le N, M, K \le 100000)$. У следећој линији дато је *N* целих бројева $J_1, J_2, \ldots, J_N$ – координате јагњади на правој. У трећој линији дато је *M* целих бројева $S_1, S_2, \ldots, S_M$ – координате штала на правој($1 \le S_i, J_i \le 1000000$).

Излаз:. Стандардни излаз треба да садржи један цео број – минимално време потребно да се смести стадо у штале тако да нити једна штала не садржи више од *K* јагњади. Ако није могуће направити такав распоред стада, исписати -1.

Пример: Улаз	Излаз
7 3 3 4 9 8 2 4 6 5 2 7 2	3

Појашњење: Јагњад с координатама 4, 2, 4 и 5 ће се расподелити у две штале с координатама 2, а остала јагњад у шталу с координатом 7. Највише времена потребно је јагњету с координатом 5 да стигне до штале са координатом 2.

Решење:
Неки од такмичара су за решење овог проблеме користили похлепну стратегију: смести свако јагње у текућу најближу шталу. Таква стратегија се показала као добра у неким тест случајевима, али та стратегија не доводи увек до тога да укупно време потребно да се смести стадо у штале буде минимално.

Задатак је могао да се реши тако што ћемо бинарном претрагом тражити одговор на питање: Да ли је неко растојање L такво да је потребно минимално време да се смести стадо у штале на путу растојања L тако да нити једна штала не садржи више од *K* јагњади?

1. Сортирамо координатите јагњади и штала
2. Покренемо претрагу Binary Search да нађемо одговор L прилагођен условима задатка
3. Проверавамо да ли је L ваљано решење (користећи похлепни алгоритам)

Тако релативно брзо налазими и жељено минимално растојање (и време).

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

#define Nmax 131075
#define BESKONACNO 1000001


int n, m, k;
int a[Nmax], b[Nmax];

bool provera(int duz)
{
	int idx = 0, ostalo = k;
	for (int i = 0; i < n; i++)
	{
		if (ostalo <= 0)
			idx++, ostalo = k;
		while (idx < m && b[idx] < a[i] - duz)
			idx++, ostalo = k;
		if (idx >= m) return false;
		if (a[i] < b[idx] - duz) return false;
		ostalo--;
	}
	return true;
}

int main(void)
{

  scanf("%d %d %d", &n, &m, &k);
	for (int i = 0; i < n; i++) scanf("%d", &a[i]);
	for (int i = 0; i < m; i++) scanf("%d", &b[i]);
```

```
	std::sort(a, a + n);
	std::sort(b, b + m);

	int dole = 0, gore = BESKONACNO;
	while (dole <= gore)
	{
		int sredina = (dole + gore) / 2;
		if (provera(sredina)) gore = sredina - 1;
		else dole = sredina + 1;
	}
	printf("%d\n", gore >= BESKONACNO ? -1 : gore + 1);
	return 0;
}
```

3. Током београдског маратона, одвија се и трка по кружној писти војног аеродрома Батајница у којој учествује K (2 <= K <= 10000) војника који морају да истрче N (1 <= N <= 1000) кругова трке. Сви тркачи почињу трку у исто време и са исте стартне линије. Познато је да уобичајена такмичарска форма пада током трке и да тркачи након сваког пуног круга трче 1 милисекунду спорије. Током уобичајене форме, i –ти такмичар истрчи један круг за ms_i милисекунди (1 <= ms_i <= 1000000). Пре почетка трке сваком такмичару се најави цео број t_i (1 <= t_i <= N) који указује да ће такмичар након сваког потпуно окончаног t_i круга добити енергетски стимуланс чим пређе стартну линију трке. Енергетски стимуланс омогућује такмичару да поврати потпуно своју снагу, али након тога такмичарска форма и издржљивост настављају да падају на већ описани начин. Узимање стимуланса захтева 0 времена. Напишите конзолну апликацију, VOJNICI који ће исписати максимални број тркача који ће пређи стартну линију истовремено у неком периоду трке (истовремено=после једнаког броја милисекунди које су протекле након почетка трке). Јасно је да ће сваки такмичар током истрчавања N кругова пређи стартну линију N пута, јер се броји прелазак преко линије након сваког круга, а не броји се прелазак преко линије на почетку трке.

Улаз: Стандардни улаз садржи садржи неколико линија. У првој линији су дата два цела броја K, N раздвојена бланко карактером. У свакој од наредних K линија задати су два цела броја ms_i, t_i раздвојена бланко карактером.

Излаз: Стандардни излаз треба да садржи једну линију. Ваш програм треба да испише максималан број тркача који ће пређи стартну линију истовремено у неком периоду трке

Пример: Улаз	Излаз
4 3 26 2 39 3 45 1 56 2	2

Појашњење: Војници ће редом трчати 3 круга за следеће време: војник1 (26, 27,26 ms); војник2 (39, 40 , 41 ms); војник3 (45, 45, 45 ms); војник4 (56, 57, 56 ms) Војници ће пређи стартну линију за следеће време: војник1 након 26, 53, 79 ms; војник2 након 39, 79, 120 ms; војник3 након 45, 90, 135 ms; војник4 након 56, 113, 169 ms. Дакле, војници 1 и 2 ће након 79 милисекунди пређи стартну линију заједно.

Решење:

```
#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;
}
```