ЕСЕНЕН ТУРНИР ПО ИНФОРМАТИКА
Шумен, 10–12 ноември 2006 г.
Тема за група С (8-9 клас)

Задача за С1. ПРОСТОТА ОТДЯСНО

Десетичният запис на естествените числа крие много любопитни свойства. Да наречем „просто отдясно" такова просто число, в десетичния запис на което:

- ако има повече от една цифра и изтрием цифрата на единиците – пак се получава просто число;
- ако полученото има повече от една цифра и направим същото – пак остава просто;
- и все така, докато остане само една цифра (и тя пак е просто число)!

Има ли „прости отдясно" числа? Да, вижте, например, числото 7193. Самото то е просто. Но прости са и всички числа, които се получават от него чрез описания по-горе процес – това са 719, 71 и 7. За да няма недоразумения, ще припомним, че числото 1 не е просто. За числото 5, например, също ще считаме, че е „просто отдясно" – то е просто и едноцифрено и за него описаният процес не започва (има нула стъпки).

Задачата Ви, която ще реализирате с програмата **`rprimes`**, се състои в пребройване на „простите отдясно" числа в зададен интервал [***a***, ***b***].

От стандартния вход се въвежда един ред с естествените числа ***a*** и ***b***, разделени с интервал, като $1 \le a \le b \le 2000000000$. Изведете на стандартния изход един ред с едно неотрицателно цяло число, равно на броя на простите числа с търсеното свойство, не по-малки от ***a*** и не по-големи от ***b***.

ПРИМЕР

Вход
```
7 300
```

Изход
```
13
```

Обяснение на изхода: В интервала [7, 300] „простите отдясно" числа са: 7, 23, 29, 31, 37, 53, 59, 71, 73, 79, 233, 239 и 293.

Решение:

Оказва се, че простите числа с описаното свойство са краен (и даже – малък) брой – всичко на всичко 83, като най-малкото от тях е 2, а най-голямото е 73939133. Генерирането им става лесно и бързо, което прави ненужно предварителното им изчисляване (но и това е допустимо на състезание): започвайки от списъка {2, 3, 5, 7} посочваме първото от тях и му долепваме отдясно всяка една от цифрите 1, 3, 7 и 9. Проверяваме всяко получено число за простота. Ако полученото е просто – добавяме го в края на списъка. Посочваме следващото в списъка и повтаряме процеса. Той приключва, когато показалецът надхвърли големината на текущия списък. Понеже числата се генерират наредени в нарастващ ред, броенето е лесно: прескачаме тези от списъка, които са по-малки от ***a*** и броим по-нататък в списъка тези, които са не по-големи от ***b***.

Примерна реализация на C:

```
#include <stdio.h>
#include <math.h>

unsigned long a,b,rprimes[1024]={2,3,5,7};
int count=4;

int isPrime(unsigned long a)
{unsigned long p,d;
 if (a==1) return 0;
 if (a==2) return 1;
 if (!(a&1)) return 0;
 d=ceil(sqrt(a));
 p=3;
 while (p<=d)
 {if (!(a%p)) return 0;
  p+=2;
 }
 return 1;
}

void makeRPrimes(void)
{int p=0;
 unsigned long d;
 do
 {d=10*rprimes[p]+1;
  if (isPrime(d)) rprimes[count++]=d;
  d+=2;
  if (isPrime(d)) rprimes[count++]=d;
  d+=4;
  if (isPrime(d)) rprimes[count++]=d;
  d+=2;
  if (isPrime(d)) rprimes[count++]=d;
  p++;
 } while (p<count);
}

int main (void)
{int c=0,i;
 scanf("%lu %lu",&a,&b);
 makeRPrimes();
 for (i=0;i<count&&rprimes[i]<a;i++);
 for (;i<count&&rprimes[i]<=b;i++,c++);
 printf("%d\n",c);
 return 0;
}
```

Примерна реализация на Pascal:

```
Var
 a,b:LongInt;
```

```
 rprimes: Array [0..1023] of LongInt;
 count,c,i:Integer;

Function isPrime(a:LongInt):Boolean;
Var p,d:LongInt;
Begin
 if a=1 then Begin isPrime:=FALSE; Exit; End;
 if a=2 then Begin isPrime:=TRUE; Exit; End;
 if Not odd(a) then Begin isPrime:=FALSE; Exit; End;
 d:=round(sqrt(a));
 p:=3;
 while p<=d do
 begin
  if a mod p=0 then Begin isPrime:=FALSE; Exit; End;
  p:=p+2;
 end;
 isPrime:=TRUE;
End;

procedure makeRPrimes;
Var p:Integer;
    d:LongInt;
Begin
 p:=0;
 Repeat
  d:=10*rprimes[p]+1;
  if isPrime(d) then Begin rprimes[count]:=d; Inc(count); End;
  d:=d+2;
  if isPrime(d) then Begin rprimes[count]:=d; Inc(count); End;
  d:=d+4;
  if isPrime(d) then Begin rprimes[count]:=d; Inc(count); End;
  d:=d+2;
  if isPrime(d) then Begin rprimes[count]:=d; Inc(count); End;
  Inc(p);
 Until p=count;
End;

BEGIN
 rprimes[0]:=2;rprimes[1]:=3;
 rprimes[2]:=5;rprimes[3]:=7;
 count:=4;
 ReadLn(a,b);
 makeRPrimes;
 c:=0;
 for i:=0 to count-1 do if rprimes[i]>=a Then break;
 for i:=i to count-1 do if rprimes[i]>b then break else Inc(c);
 WriteLn(c);
END.
```

Павлин Пеев

Задача С2. ПЛАТКА

Пешо, нашият прочут програмист, отишъл да учи в технически университет. Една от курсовите му задачи била да поправи една счупена платка. Платката представлявала точки, свързани с пътечки от алуминий, направени от поялник. За нещастие тези пътечки били разкъсани на някои места и за да я поправи трябвало да мине повторно с поялника върху тях. Платката работи само, ако всички точки са свързани с една непрекъсната пътечка от алуминий и заради това Пешо трябва да си избере една от точките, да сложи поялника на нея и без да го вдига да обходи всички останали точки. Проблемът бил, че ако мине два пъти по една и съща връзка между две точки – тя се повреждa. Помогнете на Пешо да се справи с тази трудна задача, като напишете програма **`platka`**, която по зададени връзките между точките, извежда реда, в който Пешо трябва да обходи точките.

Програмата приема входните данни от стандартния си вход. На първия ред са записани две числа N ($1 \leq N \leq 10000$) – броя на точките по платката и M ($1 \leq M \leq 100000000$) – броя на съществуващите пътечки между точките. Следващите M реда съдържат също по две числа – номерата на точките, свързани с пътечка.

Програмата трябва да изведе на стандартния изход на един ред начина на обхождане на точките, така, че след това платката да работи. Ако не съществува начин, при който да се обходят всички точки наведнъж, да се изведе съобщението “Sorry, Pesho”, следвано от броя на точките, който няма да бъдат достигнати, тръгвайки от избраната точка.

Пример:

Вход:
```
6 10
1 2
1 3
1 4
2 3
2 5
3 4
3 5
4 5
4 6
5 6
```

Изход:
```
1 3 5 6 4 1 2 3 4 5 2
```

Вход:
```
6 6
1 2
1 3
1 4
2 3
3 4
5 6
```

Изход:
```
Sorry, Pesho 2
```

Решение:

Решението на задачата прилага известния алгоритъм за намиране на Ойлеров път. (описанието на този алгоритъм може да се намери в различни книги за алгоритми). Единственото “усложнение” е изиксването за проверка дали графът има повече от една компонента:

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

int a[10000][10000];
```

```
int p[10000], vis[10000];
int n, m, u, v;

void read_data()
{
     cin >> n >> m;
     for (int i = 1; i <= m; i++)
     {
         cin >> u >> v;
         a[u][v] = 1;
         a[v][u] = 1;
         a[u][0]  =  !a[u][0];  //контролира  се  четността  на
върховете
         a[v][0] = !a[v][0]; //за четен връх – 0, а за нечетен –
1
     }
}

void dfs(int u) // за намиране броя на свързаните компоненти
{
     vis[u] = 1;
     for (int i = 1; i <= n; i++)
         if (a[u][i] && !vis[i])
            dfs(i);
}

void Euler()
{
     p[0] = u; //слага се първия връх в началото на пътя
     int l = 0, k = m; //l – края на началото, k – началото на
края
     while (l < k) //докато не се срещнат
     {
           u  =  p[l];  //взема  се  последния  връх  от  началото  на
пътя
           v = 1;
           while (v <= n && !a[u][v])
             //търси се има ли на къде да се ходи
                 v++; //цикъла спира или когато сме намерили
           if (v <= n) //съсед (v не е минало n)
           { //тогава
                 l++;  //добавяме  намерения  съсед  в  началото  на
пътя
                 p[l] = v;
                 a[u][v] = 0; //премахваме връзките
                 a[v][u] = 0; //между двата върха
                 }
           else // ако сме обходили всички върхове
                 // и не сме намерили съсед
           {
               p[k] = u; //тогава този връх се маха от началото
               l--; k--; //и се слага в края на пътя
```

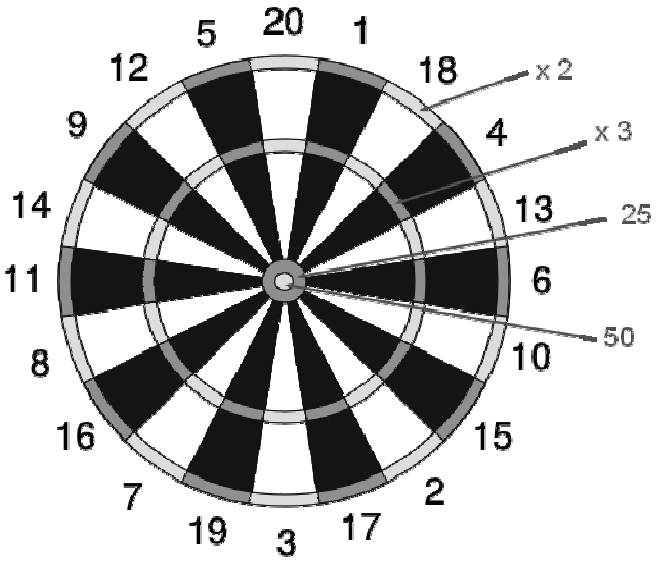
```
            }
       }
}

void print_data() // отпечатва пътя
{
     for (int i = 0; i <= m; i++)
         cout << p[i] << " ";
 }

int main()
{
    read_data();
    int i, br = 0;
    for (i = 1; i <= n; i++)
      if (a[i][0]) { u = i; break;} // намира се първия нечетен
връх
   dfs(u);
        for (i = 1; i <= n; i++)
      if (!vis[i]) br++;
     //преброяват се всички необходени от DFS върхове
     if (!br) //ако няма такива
     {
           Euler(); //се намира ойлеровия път
           print_data(); // и се отпечатва
     }
     else
           cout << "Sorry, Pesho " << br << endl;
           //иначе се отпечатва броя
    return 0;
}
```

Антония Йовчева

Задача С3. DARTS

Дартс е игра, в която играчите хвърлят къси стрели в кръгла мишена, окачена на стената. Играта се е зародила преди няколко столетия на Британските острови. И досега дартс е традиционна игра във Великобритания, Нидерландия, Скандинавските страни, Съединените Щати и др.

Стандартната дъска за дартс е квадрат със страна 48 cm, центърът на който съвпада с центъра на мишената – окръжност с радиус 20 cm. Ми-



шената е разделена на двадесет номерирани и еднакви по размер сектора, като сектор 20 е симетричен спрямо ординатната ос и секторите са номерирани по посока на часовниковата стрелка както следва: 20, 1, 18, 4, 13, 6, 10, 15, 2, 17, 3, 19, 7, 16, 8, 11, 14, 9, 12, 5.

В центъра на дъската са изобразени два концентрични кръга - с радиуси 0,5 cm и 1 cm, попадението в които се оценява съответно с 50 точки и 25 точки. Съществуват и два други концентрични обръча – вътрешен и външен. Вътрешният се определя от две окръжности с радиус 9,5 и 10,0 cm, а външният – от окръжности с радиуси 19,5 и 20,0 cm. Ако дадено попадение е между външния и вътрешния обръч или между вътрешния обръч и кръговете в центъра, се дават брой точки, равни на номера на сектора. Когато има попадение във външния обръч, играчът получава броят на точките за съответния сектор, умножени по 2. Ако попадението е във вътрешния обръч, точките от сектора се умножават по 3. Попадането на стреличка извън външния обръч не носи точки.

Когато дадено попадение е на границата на някой от гореописаните кръгове или обръчи, се счита, че попадението е в тази зона, която носи повече точки. Когато попадението е на границата между два сектора, се счита, че стреличката е попаднала в по-далечния сектор по посока, обратна на часовниковата стрелка (например, попадение между сектори 1 и 20 се счита в сектор 20, между 20 и 5 – в сектор 5, а между 17 и 2 – в сектор 2).

Във всеки рунд играчът хвърля 3 стрелички, като точките за рунда са сума от отделните попадения. Максимално възможният резултат е 180 точки (ако играчът попадне и с трите стрелички във вътрешния обръч на сектор 20).

Нека долният ляв ъгъл на дъската има координати (0,0), а горният десен – координати (48,48). Направете програма **darts**, която отчита броя точки за даден рунд при зададени координати на трите попадения.

Вход: От стандартния вход се въвеждат три двойки числа (x,y) – координатите на стреличките, с точност не повече от три знака след десетичната точка.

Изход: На стандартния изход се извежда единствено цяло число – броят точки за съответния рунд.

Примерен вход:	**Примерен изход:**
`2.212 1.423 24 24 24 34`	`110`

Решение:

Преди всичко, изчисляваме центъра на мишената с координати (24,24). За да се реши задачата, всъщност трябва да се открие отговорът на три подзадачи:

1. На какво разстояние от центъра на мишената е попадението? Това разстояние може да се сметне чрез Питагоровата теорема или разстоянието между точка A (x_1,y_1) и точка B (x_2,y_2) е $sqrt((x_1-x_2)^2+(y_1-y_2)^2)$.

Според разстоянието от центъра можем да направим следните изводи:

- При разстояние по-малко или равно на 0.5, се дават 50 точки.
- При разстояние по-малко или равно на 1.0, се дават 25 точки.
- При разстояние по-голямо от 20.0, се дават 0 точки.

В тези случаи попадението е отчетено и алгоритъмът приключва до тук.

В останалите случаи се определя коефициент за умножение M:

- При разстояние между 9.5 и 10.0, M=3.
- При разстояние между 19.5 и 20.0, M=2.
- Във всички останали случаи M=1.

2. Какъв е ъгълът между попадението и абсцисата на координатна система с център центъра на мишената? Ъгълът в градуси се пресмята по формулата:

$\angle\alpha$ = *atan(y/x)*(180/pi);*

3. В кой сектор е попадението? Тъй като сектор 20 е симетричен спрямо ординатата, можем да се досетим, че обхваща интервала [81˚, 99˚). Нормализираме получения в подзадача 2 ъгъл: $\angle\alpha = \angle\alpha + 279^\circ$. Ако $\angle\alpha > 360^\circ$, то го нормализираме чрез $\angle\alpha = \angle\alpha - 360^\circ$. По този начин сектор 20 се намира в ъгловия диапазон от 0˚ до 18˚, сектор 5 – от 18˚ до 36˚ и т. н.

Всъщност, тъй като секторите са през 18 градуса, можем да използваме цялата част от делението $\angle\alpha$ / 18˚. Секторът можем да получим чрез следната таблица:

Резултат от α/18	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Сектор	20	5	12	9	14	11	8	16	7	19	3	17	2	15	10	6	13	4	18	1

Накрая, получаваме брой точки, равен на M*Сектор.

Забележка: Откриването на ъгъла може да стане и без *atan*. Например, може да се използват отсечки, минаващи през центъра и чрез насочено лице да се сметне в кой от образуваните сектори попада дадена точка.

```
// Solution of "Darts" problem
#include <cstdlib>
#include <iostream>
#include <math.h>
#define pi 3.14159

using namespace std;

// Sector numbers

int Sectors[20]={
    20, 5, 12, 9, 14, 11, 8, 16, 7, 19,
    3, 17, 2, 15, 10, 6, 13, 4, 18, 1};

// Center of the target

double cx=24;
double cy=24;

double DistanceFromCenter(double x, double y)
// Returns the distance from the center
{
       return sqrt((x-cx)*(x-cy)+(y-cy)*(y-cy));
}

double Angle (double x, double y)
// Angle in degrees from the center of the target
{
       double ang=atan2(y,x)*(180/pi);
       if (ang<0) ang+=360;
       return ang;
```

```
       }

int GetSector (double x, double y)
// Returns the number of the sector
{
  double z=Angle(x-cx,y-cy)+279;
  while (z>360) z-=360;
  return (int) floor(z/18);
}

int Score(double x, double y)
// Calculates scores according to distance and sector
{
  double DfC=DistanceFromCenter(x,y);
  if (DfC<=0.5) return 50; else
     if (DfC<=1) return 25; else
          {
                 int MultiFactor=1;
                 if ((DfC>=19.5)&&(DfC<=20)) MultiFactor=2; else
                 if ((DfC>=9.5)&&(DfC<=10)) MultiFactor=3; else
                 if (DfC>20) MultiFactor=0;
                 int Sec=GetSector(x,y);
                 return Sectors[Sec]*MultiFactor;
          }
  cout<<'\n';
}
int main()
{   double x1,y1,x2,y2,x3,y3;
    cin>>x1>>y1>>x2>>y2>>x3>>y3;
    cout<<Score(x1,y1)+Score(x2,y2)+Score(x3,y3);
    cin>>x1;
}
```

Петър Събев