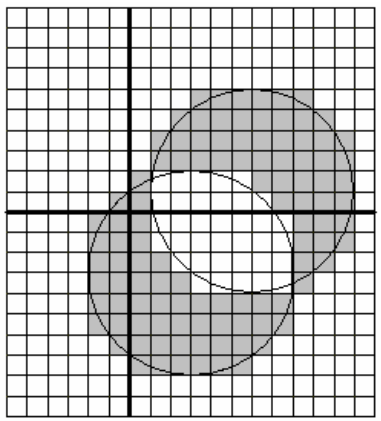
Група А

Задача А1. КРЪГОВЕ, автор Веселин Кулев



За наближаващия летен сезон, първи Европейски туристически сезон на нашето Черноморие, управата на Община Варна решила да оцвети плочките на един от площадите и възложила задачата на известен художник. Той направил идеен проект, предвиждащ да се оцветят плочките, които се съдържат в няколко кръга с еднакъв радиус. Кръговете могат да се припокриват. Една плочка се съдържа в кръг, ако има част от нея с ненулево лице, която е в кръга. За доставчик на боята художникът посочил фирмата ColXor. Оказало се, че боите на тази фирма имат много интересно свойство – когато оцветиш повторно една плочка с боята на ColXor, ефектът е, че вторият пласт боя унищожава първия и плочката става неоцветена (вж. на Фигурата). Следователно плочките, които имат общо лице с два, четири и т.н. кръга, не си струва да се оцветяват и така ще се спести много боя. Напишете програма **COLXOR**, която да определя броя на плочките, които трябва да бъдат оцветени, за да се реализира идеята на художника. Ще считаме, че плочките на площада съвпадат с квадратчетата със страна единица и целочислени координати на върховете в правоъгълна координатна система.

Вход

На първия ред на стандартния вход ще бъдат зададени две цели – броят N на кръговете и радиусът им R, $1 \le N \le 1000$, $1 \le R \le 1000$. На всеки от следващите N реда ще бъдат зададени, разделени с един интервал, координатите X и Y на центъра на една от кръговете – цели числа в интервала [–10000,10000].

Изход

На единствения ред на стандартния изход програмата трябва да изведе броя на квадратчетата, които трябва да бъдат оцветени.

ПРИМЕР

Вход	**Изход**
2 5 3 -3 6 1	104

Решение

Под „обхождане на кръг" ще разбираме обхождане на всички квадратчета със страна 1 от големия квадрат, описан около кръга. При това за всяко квадратче проверяваме дали се включва в кръга. Критерий за това може да бъде

наличието на поне един връх на квадратчето, който се намира в кръга (т.е. разстоянието от него до центъра на кръга да е по-малко от R). Тази процедура може да се направи за един условен кръг с радиус R, например този с център в началото на координатната система. Резултатът от обхождането може да бъде различен, в зависимост от това как ще продължи по-нататък работата на алгоритъма.

Ако знаехме, че покритите от кръговете области не се пресичат и при обхождането на кръг намерим броя C на покритите от него квадратчета, то търсеният отговор ще бъде $C.N$ и това ще бъде решение със сложност $O(R^2)$. Проверката за непресичане е доста неприятна и ще вдигне сложността до $O(N^2+R^2)$, без да имаме гаранция, че такова решение ще даде верен отговор дори за един тест.

Наивното решение, което би могло да се използва, ако нямаше ограничения по време и памет би изглеждало така. Нека A и B са страните на минималния правоъгълник, в който се съдържат всички кръгове. За всяко от квадратчетата на правоъгълника отделяме памет от един бит с начална стойност 0. Обхождаме всеки един от кръговете (но вече не условно, а според мястото му в равнината) и за всяко покрито от кръга квадратче инвертираме съответния му бит. Броят на битовете, стойността на които е 1 в края на обхождането е търсеният резултат. Сложността на този алгоритъм е $O(A.B+NR^2)$, а необходимата памет $O(A.B)$ – в случая около 400 мегабита. Ако вместо да отделяме по един бит за всяко квадратче използваме хеш-таблица само за квадратчетата, които се покриват от поне един кръг, тогава сложността ще падне до $O(NR^2)$, но сега пък необходимата памет може да се вдигне до $O(NR^2)$, което в случая е 1000 мегабита.

Идеята за по-добро решение изключва разглеждането на всяко квадратче. Нека при обхождането на условния кръг с център началото на координатната система разбием покриваните от кръга квадратчета на $2R$ вертикални ленти с ширина 1, като всяка лента се определя с два от върховете си, например (x,y) и $(x,y+z)$. За всеки от реалните кръгове разбиването се получава като към x и y добавим съответните координати на радиуса. Сортираме така полученото множество от $4.NR$ точки по x, а при равен x – по y. За решаването на задачата е достатъчно да обходим този масив, като за точките с еднаква x координата броим квадратчетата от първата до втората точка за оцветени, от втората до третата за неоцветени, от третата до четвъртата за оцветени и т.н. Сложността на получения алгоритъм, при използване на някой от алгоритмите за бързо сортиране ще бъде $O(NR.\log(NR))$.

Това решение може да бъде подобрено. Нека, вместо да събираме кращата на всички ленти в един масив, построим отделен списък на лентите с една и съща x-координата и да сортираме потделно всеки от тези масиви. Интуитивно, най-лошият случай ще бъде когато всички кръгове имат центрове с равни x-координати. Броят на списъците в такъв случай е $2R$ а във всеки от списъците ще има по N ленти. При сортиране на всеки от списъците с някой от бързите алторитми ще получим алгоритъм със сложност $O(N\text{R}.\log N)$.

```
#include <cstdio>
#include <vector>
#include <algorithm>
#include <math.h>
using namespace std;
#define MAXN      11100
#define SQ(a)     ((a)*(a))
#define EQ(a,b) ((fabs(a-b)<(1e-9))?(1):(0))
int N,X,Y,R,ans = 0; double RR;
vector<pair<int,pair<int,int>>> initv;
vector<int> v[2*MAXN];
inline double dist (int x,int y)
{return sqrt((double)SQ(X-x)+SQ(Y-y));}
inline int inside(int x,int y)
{
   double temp;
   temp = dist(x,y);
   if (temp<RR && !EQ(temp,RR)) return 1;
   temp = dist(x+1,y  );
   if ( temp < RR && !EQ(temp,RR)) return 1;
   temp = dist(x  ,y+1);
   if ( temp < RR && !EQ(temp,RR)) return 1;
   temp = dist(x+1,y+1);
   if ( temp < RR && !EQ(temp,RR)) return 1;
   return 0;
}
void init()
{
   for (int x=X-R,y= -1;x< 0;x++ )
   {
      while(inside(x,y-1)) y--;
       initv.push_back(make_pair(x,make_pair(y,-y)));
       initv.push_back(make_pair(-x-1,make_pair(y,-y)));
   }
}
int main()
{
   scanf("%d%d",&N,&R);
   RR = (double)R;
   init();
   for(int k=1;k<= N;k++)
   {
      scanf("%d%d",&X,&Y);
       for(unsigned i=0;i<initv.size();i++)
      {  int a= initv[i].second.first+Y;
         v[initv[i].first+X+MAXN].push_back(a);
         a= initv[i].second.second+Y;
         v[initv[i].first+X+MAXN].push_back();
      }
   }
```



```
   for(unsigned i=0;i<2*MAXN;i++)
   {
      if(v[i].size()==0)
      {
         if(i+R<2*MAXN && v[i+R].size()==0) i+=R;

       }
      else
      {
          sort(v[i].begin(),v[i].end());
          for(unsigned j=0;j<v[i].size();j+= 2)
            ans += (v[i][j+1]-v[i][j]);
      }
   }
   printf("%d\n",ans);
   return 0;
}
```

Задача А2. СКЛАД, автор Красимир Манев

За да станат някои продукти (сирена, вина и т.н.) добри за консумация, те трябва да отлежават в помещения с постоянна температура. Затова складът на фирмата TStore е разположен под земята и се състои от N зали, номерирани с числата от 1 до N, свързани чрез тунели с еднаква дължина. Единствен вход на склада е залата с номер 1 и от нея се извършва експедицията на стоките. От входната зала на склада, вървейки по тунелите, може да се стигне по единствен начин до всяка друга зала. Във всяка от залите е складирано някакво количество стока, подредена в еднакви кашони. Изнасянето на стоките до входната зала става с електрокар, който може да носи по M кашона, а във всяка от залите може да се складират по време на изнасянето толкова кашона, колкото се налага. Дошло е време, всичките стоки за бъдат пренесени до входната зала и Директорът на склада се пита какво е минималното необходимо време за това, ако преминаването по кой да е от коридорите на склада, свързващ две зали, става за единица време, а времето за товарене и разтоварване е пренебрежимо малко. Преди да започне изнасянето на стоките електрокарът се намира във входната зала. Напишете програма **STORE,** която да решава така поставената задача.

Вход

На първия ред на стандартния вход ще бъдат зададени числата N и M, $3 \le N \le 10000$, $1 \le M \le 100$. Всеки от следващите N реда описва една от залите, като описанията са подредени в нарастващ ред на номерата на залите. За всяка зала Z, редът започва с броя K_z на кашоните, намиращи се в тази зала, $0 < K_z \le 200$. Следва броят S_z на залите, свързани чрез коридор с Z, без тази, която е първа по единствения път от Z до входната зала. Ако S_z не е нула, в реда има още S_z числа – номерата на съседните на Z зали. Числата в реда са разделени с по един интервал.

Изход

На единствения ред на стандартния изход програмата трябва да изведе намереното минимално необходимо време.