

Група А

Задача А1. КРЪГОВЕ, автор Веселин Кулев

За приближавания 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 неочветена (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 , $1 \leq N \leq 1000$, $1 \leq R \leq 1000$. 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.

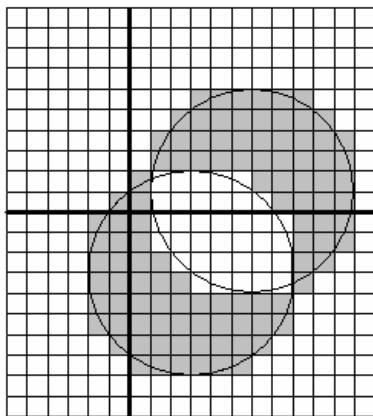
ПРИМЕР

Вход

2 5
3 -3
6 1

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



Изход

104

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 $C \cdot N$ 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 $O(A \cdot B + NR^2)$, а необходимата памет $O(A \cdot B)$ – 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 условия 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 $4 \cdot NR$ 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 неочветени, от третата до четвъртата за оцветени и т.н. Сложността на получения алгоритъм, при използване на някой от алгоритмите за бързо сортиране ще бъде $O(NR \cdot \log(NR))$.

Тава решение може да бъде подобро. 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 алгоритми ще получим алгоритъм със сложност $O(NR \cdot \log N)$.

```

#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(a);
        }
    }
}

```

```

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 A2. 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 складира 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 , $3 \leq N \leq 10000$, $1 \leq M \leq 100$. 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, $0 < K_z \leq 200$. 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.

ПРИМЕР

Вход

```
6 10
3 2 2 4
5 0
3 0
2 2 3 5
15 1 6
20 0
```

Изход

```
24
```

Решение

Складът за който става дума може да бъде представен като неориентиран граф – залите са върховете, а тунелите – ребрата. От условието е ясно, че този граф е дърво – съществуването на път от върха, съответен на входа, до всеки друг връх означава свързаност, а единствеността на този път – отсъствие на цикли. Обявявайки входния връх за корен, превръщаме дървото в кореново. Така, съседите на всеки връх, зададени във входа, са синовете му в кореновото дърво. Като използваме наредбата на върховете зададена във входа, можем да въведем наредба на синовете в кореновото дърво.

За решаването на задачата ще използваме едно от по-малко популярните, но много удобно за случая, представяне на кореновите дървета с наредба на синовете – „най-ляв син – десен брат“. Това представяне е толкова компактно, колкото и популярното представяне на двоични коренови дървета с наредба на синовете – „ляв син – десен син“, защото за всеки връх помним два други върха – най-левия от синовете му (ако изобщо има синове) и следващият в наредбата, т.е. братът отдясно (ако има такъв). При въведената номерация на върховете, за означаване на отсъствието на най-ляв син или десен брат използваме 0. Първата част на решението построява исканото представяне.

За намиране на минималното време (или на минималния път, което е същото, е достатъчно да обходим полученото кореново дърво в „постордер“, т.е. обхождането на всеки връх ще извършим тогава, когато са обходени всички върхове на поддървото, на което той е корен. Така кашоните от върховете на всяко поддървото ще се съберат в корена му и могат да бъдат изнесени от там при оптимално натоварване на електрокара. Обхождането започваме от корена с дължина на пътя 0. Под обхождане на текущия връх v ще разбираме следното:

- ако v е с необходим най-ляв син – отлагаме обхождането на v за по-късно, преместваме се в най-левия му син и добавяме единица в дължината на търсения път;
- ако v няма синове или най-левият му син е обходен – изнасяме всички кашони в бащата на v , като добавяме $2 \cdot \lceil K_v/M \rceil - 1$ към дължината на изминатия път и обявяваме v за обходен. Ако v има десен брат, той става текущ връх и добавяме единица в дължината на търсения път, а ако няма – нов текущ е бащата на v ;
- когато се окажем отново в корена на дървото, обхождането е завършено – т.е. всички кашони са пренесени до корена и с това задачата е решена.

Един недостатък на представянето „най-ляв син – десен брат“ е, че в него трудно се намира бащата на зададен връх. Това може да се преодолее с още едно поле за всеки възел, в което е записан неговият баща. В представената програма се използва стек, в който държим всички върхове по пътя от корена до текущия връх и с негова помощ намираме бащата елементарно. Тъй като за броя M на ребрата на всяко дърво е в сила $M = N - 1$, сложността на този алгоритъм е $O(N)$.

```
#include <stdio.h>
#define MAXN 10001
int main()
{
    int N,M,L=0;
    int lson[MAXN],rbro[MAXN],q[MAXN],st[MAXN],sp;
    int i,j,s,x,y,a,b;
    scanf("%d %d",&N,&M);
    for(i=1;i<=N;i++)
    {
        scanf("%d %d",&q[i],&s);
        if(s==0)
            lson[i]=0;
        else
        {
            if(s==1)
            {
                scanf("%d",&x);lson[i]=x;rbro[x]=0;
            }
            else
            {
                scanf("%d",&x);lson[i]=x;y=x;
                for(j=2;j<=s;j++)
                {
                    scanf("%d",&x);rbro[y]=x;y=x;
                }
                rbro[y]=0;
            }
        }
    }
    //postorder na "lson-rbro"
    st[0]=1;sp=0;
    while(1)
    {
        x=st[sp]; //x e varhat na steka
        y=lson[x]; // y e leviat mu sin
        if(y!=0) //ako ima lyav sin y
        {
            st[++sp]=y;L++;
        }
        else // x niama liav sin
        {
            //iznasiane do bashtata
            a=q[x]/M; b=q[x]%M;
            q[st[sp-1]]+=q[x];
            L+=2*a-1;if(b>0) L+=2;
            y=rbro[x];
            if(y!=0) //x ima desen brat
            {
                st[sp]=y;L++;
            }
        }
    }
}
```

```

else // x niama desen brat
{
    sp--; lson[st[sp]]=0;
    if(sp==0) break;
}
}
}
printf("%d\n",L);
return 0;
}

```

Задача А3. ЧЕТВОРКИ, автор Стоян Капралов

Да разгледаме следните 4-елементни множества с елементи цели числа от 1 до 8: {1,2,3,4}, {1,3,5,7}, {2,3,5,8}, {2,3,6,7}, {3,4,5,6}, {5,6,7,8}.

Тази съвкупност притежава свойството, че всеки две множества имат най-много два общи елемента. Може да се докаже, че от множество с 8 елемента, могат да бъдат избрани най-много 14 четворки, така, че всеки две от тях да имат най-много два общи елемента.

Намерете съвкупност от колкото се може повече на брой 4-елементни подмножества на множеството {1,2,3,4,5,6,7,8,9,10}, така, че всеки две от тях да имат най-много два общи елемента.

Резултата запишете във файл **A3.txt**. На първия ред запишете броя **m** на подмножествата в намерената съвкупност. На всеки от следващите **m** реда запишете по едно от намерените подмножества – четири различни цели числа от 1 до 10, разделени с интервали.

Ако записаната във файла съвкупност е коректна, ще получите **100m/M** точки, където **M** е броя на четворките в съвкупността, построена от журито.

Решение

Условието две четворки да имат най-много два общи елемента означава, че нямат обща тройка. Всяка четворка „покрива” 4 различни тройки. Следователно в решение с **m** четворки ще бъдат покрити общо **4m** различни тройки. Всички възможни тройки са $\binom{10}{3}=120$. Следователно $4m \leq 120$,

откъдето $m \leq 30$.

Ако съществува решение с 30 четворки, всяка тройка ще се среща точно веднъж в някоя от четворките. Такава конфигурация се нарича шайнерова система от четворки.

Ще опитаме да построим решение с 30 четворки. Всички четворки са $\binom{10}{4}=210$. Избирането на 30 от тях, без използването на никакви допълнителни съображения е практически неизпълнимо.

Нека **k** е броят на всички четворки в решението, съдържащи числото 1. Всяка такава четворка покрива по 3 двойки от множеството {2,3,4,5,6,7,8,9,10}, а две такива четворки не бива да имат обща двойка, защото вече имат общ елемент – числото 1. Общо двойките, които могат да се образуват от 9 числа са 36. Следователно $k \leq 12$.

Всички четворки, съдържащи числото 1 са $\binom{9}{3}=84$. От тях искаме да

изберем 12, така че да са съвместими помежду си. Това вече става за секунди. Едно възможно решение е:

{1,2,3,4} {1,2,5,6} {1,2,7,8} {1,2,9,10} {1,3,5,7} {1,3,6,9}
 {1,3,8,10} {1,4,5,10} {1,4,6,8} {1,4,7,9} {1,5,8,9} {1,6,7,10}.

Остава да продължим решението с още 18 четворки, несъдържащи числото 1. Общият брой на четворките, от които ще избираме е $210-84=126$.

Оказва се, че с праволинейно написана програма, без никакви допълнителни „хитрости” намираме решение с 30 четворки за не повече от 10 секунди.

```

#include <fstream>
using namespace std;
int a[210][4];
int c[30];
void init()
{
    int p=0;
    for(int i0=1; i0<=7; i0++)
        for(int i1=i0+1; i1<=8; i1++)
            for(int i2=i1+1; i2<=9; i2++)
                for(int i3=i2+1; i3<=10; i3++)
                    { a[p][0]=i0; a[p][1]=i1; a[p][2]=i2; a[p][3]=i3;
                      p++;}
}

bool ok2(int i1, int i2)
{
    int s=0;
    for(int j1=0; j1<4; j1++)
        for(int j2=0; j2<4; j2++)
            if(a[i1][j1]==a[i2][j2]) s++;
    if(s>2) return false;
    return true;
}

bool ok(int k)
{
    for(int i=0; i<k; i++)
        if(!ok2(c[i],c[k])) return false;
    return true;
}

void show()
{
    ofstream fout("A3.txt");
    fout << 30 << endl;
    for(int i=0; i<30; i++)
        fout << a[c[i]][0] << " " << a[c[i]][1] << " "
            << a[c[i]][2] << " " << a[c[i]][3] << endl;
    fout.close();
    exit(0);
}

```

```

void gen(int k)
{ if(k==30) show();
  int min,max;
  if(k==0) min=0; else min=c[k-1];
  if(k<12) max=84; else max=210;
  for(c[k]=min; c[k]<max; c[k]++)
    if(ok(k)) gen(k+1);
}
int main()
{ init();
  gen(0);
}

```

Група В

Задача В1. СУМИ, автор Младен Манев

Напишете програма **SUM**, която да намира броя на начините, по които зададено цяло положително число N може да се представи като сума на поне две последователни положителни цели числа. Числото 5, например, може да се представи като такава сума точно по един начин: $5=2+3$. Числото 8, пък, изобщо не може да се представи така (0 начини), а числото 21 има три такива представяния: $21=10+11=6+7+8=1+2+3+4+5+6$.

Вход

Програмата трябва да въвежда цяло число N ($2 < N < 10^8$) от стандартния вход.

Изход

Програмата трябва да изведе на стандартния изход броя на начините, по които N може да се запише като сума на поне две последователни положителни цели числа.

ПРИМЕР

Вход	Изход
9	2

Решение

Нека n е такова, че може да се представи като сума на k последователни положителни цели числа, най-малкото от които е m . Тогава

$$n = m + (m+1) + \dots + (m+k-1) = \frac{m + (m+k-1)}{2} \cdot k.$$

Следователно броят на търсените начини е равен на броя на наредените двойки от цели числа (m, k) , които са решения на диофантовото уравнение

$$2n = (2m+k-1)k \text{ и за които } m > 0, k > 1. \text{ Тъй като } 2m = \frac{2n}{k} - k + 1 \text{ и}$$

$k < 2m + k - 1$, то наредената двойка числа (m, k) ще е решение на задачата, ако:

- k дели $2n$;

- $\frac{2n}{k} - k + 1$ е четно число;

- $k^2 < 2n$.

```

#include<iostream>
using namespace std;
int main()
{ int n;
  cin >> n;
  int s=0;
  for(int k=2; k*k<2*n; k++)
    if (!(2*n%k) && !((2*n/k-k+1)%2)) s++;
  cout << s << endl;
  return 0;
}

```

Задача В2. СКОКОВЕ, автор Емил Келеведжиев

По дължината на една алея, на разстояние един метър една от друга, са разположени $N+1$ площадки, номерирани с целите числа от 0 до N , където N е цяло число, $1 < N < 200\,000$. На всяка площадка е поставена кошница с зададен брой ябълки. Скокълъ е известен със способността си да извършва M различни по дължина (измерена в метри) скока, $0 < M < 200$. Скокълъ обича да се разхожда по алеята, като скача от площадка на площадка и събира ябълките които намери в съответната кошница. Разходките му винаги започват от началото на алеята (площадката с номер 0). Когато е на някоя площадка, той може да скочи, с един от възможните M скока, на някои от следващите площадки (такава с по-голям номер). Разходката може да бъде прекратена на произволна площадка до която Скокълъ е достигнал.

Скокълъ може да взема ябълки само от кошниците на площадките, в които е попаднал, включително от тази, от която е тръгнал и тази, в която е прекратил разходката. Напишете програма **SKOK**, която да му помогне да събере максимален брой ябълки.

Вход

Програмата трябва да прочете входните данни от стандартния вход. На първия ред са зададени числата N и M , разделени с интервал. На втория ред са зададени M числа – позволените дължини на скоковете в метри. Всяка от тези дължини е цяло положително число, по-малко от 1000. На третия ред са зададени $N+1$ цели числа – бройките на ябълките, подредени в нарастващ ред на номерата на съответните им площадки. На една площадка може да има най-малко нула и най-много 1000 ябълки. Всички числа във втория и третия ред на входа са разделени с по един интервал.

Изход

Програмата трябва да изведе на един ред в стандартния изход две цели числа, разделени с един интервал. Първото от тези числа трябва да е равно на максималния брой ябълки, които Скокълъ може да събере по време на разходката, а второто число да е равно на номера на последната площадка, до която той е достигнал, получавайки този максимален брой. Ако са възможни

няколко начина за получаване на максимален брой ябълки, програмата трябва да изведе най-малкия възможен номер на последната достигната площадка.

ПРИМЕР

Вход	Изход
4 2	16 2
2 3	
7 8 9 9 0	

Решение

Разглеждаме множеството от подзадачи $\{T_i, i=0,1,\dots,N\}$ подобни на дадената, при които търсим максималния брой ябълки, които Скокълъ може да събере, тръгвайки от началото и завършвайки в площадката на i -тия метър. Подзадачата T_i може да се реши, ако вече имаме решенията на подзадачите за по-малки стойности на i . Тази идея на динамичното оптимизиране е реализирана в приложената програма. В $p[i]$ въвеждаме броя ябълки, намиращи се на i -тата площадка, а позволените дължини на скоковете са в елементите на масива $s[j]$. В $t[i]$ пресмятаме последователно решенията на описаните по-горе подзадачи. Очевидно $t[0]=p[0]$, а стойността на $t[i]$, за $i>0$ е равна на $p[i]$ плюс най-голяма от тези стойности на $t[k]$, които са такива, че от позиция k може да се стигне с един скок до позиция i . Решението на основната задача е най-голямата стойност на масива $t[i]$.

```
#include <iostream>
using namespace std;
const int N=200000, M=200;
int n,m; int s[M]; int p[N], t[N];
int main()
{
    cin >> n >> m;
    for(int j=1;j<=m;j++) cin >> s[j];
    for(int i=0;i<=n;i++) cin >> p[i];
    t[0]=p[0];
    for(int i=1;i<=n;i++)
    {
        int v=-1;
        for(int j=1;j<=m;j++)
        {
            int k=i-s[j];
            if(k>=0) if(t[k]>-1) if(v<t[k]) v=t[k];
        }
        if(v>-1) t[i]=v+p[i]; else t[i]=-1;
    }
    int max=t[0]; int pos=0;
    for(int i=1;i<=n;i++)
    if(t[i]>max) {max=t[i]; pos=i;}
    cout << max << " " << pos << endl;
}
```

Задача В3. MUSIC IDOL, автор Зорница Дженкова

Млади хора от цялата страна участвали в поредния кастинг на Music Idol. Някои кандидати имали музикален опит, други били завършили различни

школи, но всички са участвали в предварителни изпити и всеки има получен бал, който се изразява с дробно число от 0 до 100, с точност до 4 цифри след десетичната точка. Журито изслушвало участниците от град Варна в продължение на N дни, $1 < N < 200$. За да има ред, организаторите на кастинга направили така, че участниците да пристигат в града само през нощта преди съответния работен ден на журито (включително и в нощта преди първия работен ден) и се настанявали в един хотел. Всеки ден, журито избирало да прослушва определен брой от настанените в хотела участници, които имали най-висок бал, а ако имало такива с равен бал, от тях се избирали тези, които имали по-ранна регистрация в хотела. След като един участник бъде прослушван, той веднага трябвало да напусне хотела. Хотелът имал 10000 места и никога не се препълнил по време на целия кастинг. След N -тия ден в хотела имало останали неизслушани участници, а в нощта преди $(N+1)$ -вия работен нови кандидати не били настанявани.

Напишете програма **MUSIC**, която намира името на първия неизслушан участник, т.е. този, който би трябвало да бъде изслушан първи през $(N+1)$ -вия ден, ако журито би решило да работи един ден повече от предварително определените дни.

Вход

Данните се четат от стандартния вход. На първия ред е записано числото N . Следват N реда, всеки започващ с броя M на пристигналите участници през нощта преди съответния ден, $0 \leq M < 500$, следван от M двойки, съставени от името на поредния участник и неговия бал. Накрая на реда е записан броят K на действително изслушаните от журито участници за деня, $0 \leq K < 300$. Всички числа и имена са разделени с по един интервал. Имената са съставени от по най-много 10 малки и главни латински букви.

ПРИМЕР

Вход	Изход
3	Krum
3 Ana 90 Krum 90 Kalina 99 2	
0 0	
1 Roza 90.0001 1	

Решение

Данните за всеки участник – името, балът и поредният номер на регистрацията в хотела – са представени в структурата **data**. За решаване на задачата е използвана приоритетна опашка Q , реализирана в STL. След прочитане на данните, отнасящи се за поредния ден, те се внасят в Q , в съответствие с указаните приоритети – първо по намаляваща стойност на бала, а при равен бал – по нарастващ номер на пристигане в хотела. Приоритетът на данните в опашката се определя от класа **cmp**, където освен големината на бала, се взема предвид и поредността на регистрацията в хотела. След това се изваждат от Q тези участници, които са били прослушани за същия ден. Този процес се повтаря N пъти, колкото са дните на конкурса. Накрая се извежда името на този участник, които ще се окаже на върха на приоритетната опашка.

```

#include<iostream>
#include<queue>
using namespace std;
struct data
{ string s; double b; int r; };
struct cmp:
public binary_function<data, data, bool>
{ bool operator()(data x, data y)
  {if(x.b < y.b) return true;
   if(x.b==y.b) if(x.r > y.r) return true;
   return false;
  }
};
priority_queue<data, vector<data>, cmp> Q;
int main()
{int c=0;
 int n; cin >> n;
 for(int i=1;i<=n;i++)
 { int m; cin >> m;
  for(int j=1;j<=m;j++)
  { data d;
   cin >> d.s >> d.b;
   d.r=c; c++;
   Q.push(d);
  }
  int k; cin >> k;
  for(int j=1; j<=k; j++) Q.pop();
 }
 cout << (Q.top()).s << endl;
}

```

Група С

Задача С1. РЕДИЦИ, автор Емил Келеведжиев

Да разгледаме редицата (3,1,4,2,4). Тя може да бъде разбита на две подредици – (3,4) и (1,2,4), които са строго растящи, т.е. елементите им са сортирани в строго нарастващ ред. *Разбиване* означава, че всяка от двете подредици трябва има поне по един елемент, обединението на елементите на двете подредици трябва да се състои от елементите на дадената редица и никой елемент от дадената редица не трябва едновременно да е елемент и на двете подредици. Забележете, че за редицата (4,8,1,5,3) такова разбиване не може да се направи. Напишете програма **SEQ**, която “познава” дали дадена редица от цели положителни числа може да бъде разбита на две строго растящи подредици.

Вход

На първия ред на стандартния вход е зададен броят N на редиците, които програмата трябва да провери, $N \leq 10$. Всеки от следващите N реда дефинира по една редица. Редът започва с броя M на елементите в съответната редица, $2 \leq M \leq 100\,000$. В същия ред, след един интервал, са зададени елементите на редицата, разделени един от друг с интервали. Всички елементи са цели положителни числа, не надминаващи 1 милиард.

Изход

На стандартния изход програмата трябва да изведе низ с дължина N , съставен от нули и единици (без разделящи интервали), с резултатите от направените проверки. За поредната редица низът трябва да съдържа цифрата 1, ако е възможно тази редица да се разбие на две строго растящи подредици, или цифрата 0, ако такова разбиване не е възможно.

ПРИМЕР

Вход

```

2
5 3 1 4 2 4
5 4 8 1 5 3

```

Изход

```

10

```

Решение

Наивната идея да образуваме всички възможни разбивания на редицата в две подредици и да проверяваме, дали получените подредици са строго растящи, ще доведе до алгоритъм с експоненциално нарастване на броя на стъпките, при увеличаване размерите на задачата. Алгоритъм с линейна сложност се получава по следния начин. Да допуснем, че успешно сме разбили началото (съставено от първите I елемента) на редицата и a и b са последните (максималните) елементи на двете подредици. За началото, съставено от първия елемент, такова разбиване е възможно и единствено – в едната подредица е първият елемент, а втората е празна. Максималният измежду a и b да означим с max . Да разгледаме $(I+1)$ -вия елемент c . Ако $c > max$, трябва да го причислим към подредицата на max . Ако $c < max$ – опитваме да го причислим към другата подредица. Ако това е възможно – получили сме разбиване на първите $I+1$ елемента. Ако с добавянето му там строгата монотонност на подредицата се нарушава, ясно е, че задачата няма решение.

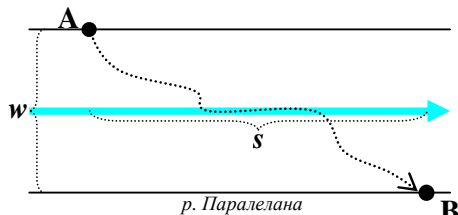
```
#include <iostream>
using namespace std;
void check(int k)
{
    int a; cin >> a; int b=0;
    int i=2; int c;
    while(i<=k)
    {
        cin >> c;
        if(c>a) a=c; else if(c>b) b=c;
        else
        { i++; while(i<=k){cin >> c; i++;}
          cout << 0; return;
        }
        i++;
    }
    cout << 1;
}
int main()
{
    int n; cin >> n;
    for(int i=1;i<=n;i++)
    { int k; cin >> k; check(k); }
    cout << endl;
}
```

Задача С2. ПРЕСИЧАНЕ, автор Павлин Пеев

Река Паралелана почти замръзна! Преди беше лесно да стигнете с кануто си от коя да е точка A на единия бряг до коя да е точка B , разположена на другия бряг и надолу по течението. Сега по средата е останало водно пространство точно колкото за едно кану. Това все пак е нещо, но нали, за да стигнете до водата, се налага да влачите кануто по леда, а като излезете от водата – да го влачите отново до крайната точка? А в този студ ви се иска да стигнете колкото може по-бързо! Както се вижда от картата на местността, реката в този участък е с прави, успоредни брегове. Широчината ѝ е w км, а хоризонталното преместване от A до B е s км. Вашата скорост по леда е a км/ч, а във водата – b км/ч, като, естествено, $a < b$. Напишете програма **CROSSING**, която да намира минималното необходимо време за придвижване от точка A до точка B .

Вход

На единствения ред на стандартния вход са зададени, разделени с интервали, числата w , s , a и b , в тази последователност. Ако някое от числата има дробна



част, тя е разделена от цялата част с точка и е с не повече от два десетични знака. Числата са положителни, като $w < s \leq 40$, $a < b \leq 20$.

Изход

Програмата трябва да изведе на стандартния изход един ред с минималното време, за което можете да стигнете от A до B , закръглено с точност до секунда, в стандартен времеви формат – часове, двоеточие, минути, двоеточие, секунди – и без водещи нули.

ПРИМЕР

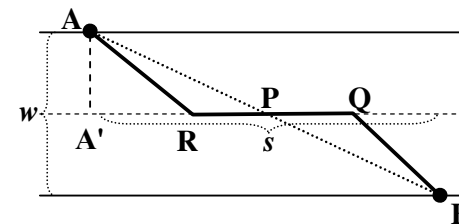
Вход
1.5 6 3.12 9.6

Изход
1:4:47

Решение

От съображения за симетрия и равнопоставеност, маршрутът трябва да има вид, подобен на начупената линия $ARQB$ на чертежа – симетричен относно P . Тогава е достатъчно да пресметнем времето за достигане до точка P и да го удвоим.

Остава да определим положението на точка R върху средната права. Ясно е, че тя не може да е по-наляво от A' , мястото ѝ е в затворената отсечка $A'P$. Също така е ясно, че с преместването на R по $A'P$ от



ляво на дясно, търсеното време или само ще нараства, или само ще намалява, или ще намалява донякъде и после ще започне да расте, т. е., ще има една най-малка стойност – или в A' , или в P (когато отсечката RQ ще се изроди в точка), или някъде между тях. На тези разсъждения се основава конкретната реализация на алгоритъма „клатене“ – търсене на интервал с ширина 0.1, съдържащ точката P , търсене в него на интервал с ширина 0.01, съдържащ P и т.н., до намиране на интервал с ширина 0.00001, съдържащ P , което е напълно достатъчно за точност до секундата в искания формат.

```
#include <stdio.h>
#include <math.h>
typedef struct {double x,y;} Point;
double w,s,a,b;
Point A,P;

//Разстояние между точки
double dist(Point p1, Point p2)
{ return sqrt((p1.x-p2.x)*(p1.x-p2.x)+
              (p1.y-p2.y)*(p1.y-p2.y));
}

//Време за изминаване на ARP
double getTime(Point R)
{ return dist(A,R)/a+dist(R,P)/b; }
```



```

//Закръгляне
double round(double x)
{ return floor(x+0.5); }

//Време в стандартен формат
char *format(double t,char *r)
{ double h,m,s;
  float p=modf(t,&h);
  p=round(p*3600);
  s=fmod(p,60);
  m=floor(p/60);
  sprintf(r,"%0f:%0f:%0f",h,m,s);
  return r;
}

//Намиране на минимално време
double minTime(void)
{ double m,min,x,st;
  Point R;
  R.x=0;R.y=w/2;
  min=getTime(R);
  st=x=0.1; //"едра" стъпка
  do
  { do
    { R.x=x;
      m=getTime(R);
      if (m<=min) min=m;
      else break;
      x+=st;
    }while (x<=s/2);
    x-=2*st;
    st/=10; // намаляване на стъпката 10 пъти
    R.x=x;
    m=getTime(R);
    if(fabs(m-min)<0.00001) break; //край
    min=m;
  }while(1);
  return 2*min;
}

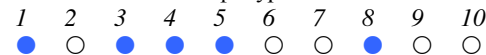
int main (void)
{ char buf[16];
  scanf("%lf %lf %lf %lf",&w,&s,&a,&b);
  A.x=0;A.y=w;
  P.x=s/2;P.y=w/2; //координати на точките А и Р.
  printf("%s\n",format(minTime(),buf));
  return 0;
}

```

Задача С3. ОГЪРЛИЦИ, автор Павлин Пеев

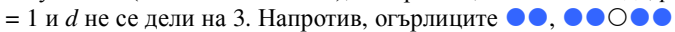
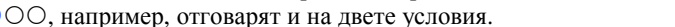
Фирма „Гери и Дани“ има утвърден оригинален патент за мънистени огърлици в класическите два цвята – синьо и бяло. Освен интересната закопчалка на винт, към патента е включен и уникален метод за подредба

на мънистата, който е и гаранция за автентичност. Нека при разкопчана огърлица с дължина n да номерираме мънистата, отляво надясно, с целите числа от 1 до n , както е показано на фигурата:



Патентованата подредба се подчинява на следните две правила:

1. Най-лявото мънисто (това с номер 1) е синьо.
2. Нека сините мъниста с нечетни номера са p на брой, а сините мъниста с четни номера – q на брой. Разглеждаме абсолютната стойност d на разликата на числата p и q , $d = |p - q|$. Тя е цяло неотрицателно число. Патентът гласи, че правилна е тази подредба, за която d се дели на 3 без остатък.

Показаната по-горе огърлица, например, не е правилно подредена: тя изпълнява първото условие (мънисто 1 е синьо), но при нея, както се вижда, $p = 3$, $q = 2$, т. е. $d = 1$ и d не се дели на 3. Напротив, огърлиците  и , например, отговарят и на двете условия.

От фирмата искат да знаят колко са правилните огърлици с определена дължина (брой мъниста). Помогнете им, като напишете програма **NECKLACE**, която решава задачата.

Вход

Програмата трябва да въведе от единствения ред на стандартния вход дължината n ($1 \leq n \leq 60$) на огърлицата.

Изход

Програмата трябва да изведе броя на различните правилни огърлици с дължина n на стандартния изход.

ПРИМЕР

Вход	Изход
6	11

Обяснение: Правилните огърлици с дължина 6 са: , , , , , , , , ,  и .

Решение

Очевидна е връзката с двоични числа. Ако кодираме синьото мънисто с 1, а бялото – с 0, то зададените правила описват n -цифрените двоични числа без водещи нули, които се делят на 3. Наистина, ако с a_i означим двоичните цифри на едно число A , то

$$A = a_1 a_2 a_3 \dots a_n = a_1 \cdot 2^{n-1} + a_2 \cdot 2^{n-2} + a_3 \cdot 2^{n-3} + \dots + a_n.$$

Като имаме предвид, че $2 \equiv -1 \pmod{3}$, то четните степени в това представяне могат да се заместят с 1, а нечетните – с -1 по модул 3. Тогава е ясно, че един признак за делимост на 3 в двоична бройна система е делимостта на 3 на алтернативната сума от двоичните цифри $a_1 - a_2 + a_3 - a_4 + \dots$. Това изказване всъщност е еквивалентно на формулировката в задачата. Ще припомним само, че, поради аналогични причини, подобно изглежда и известният признак за делимост на 11 в десетична бройна система. И естествено, защото $11_2 = 3$.

И така, в задачата се търси броят на n -цифрените двоични числа без водещи нули, които се делят на 3. Като извършим елементарни изчисления,

$$C(n) = \begin{cases} \frac{2^{n-1} - 1}{3}, & \text{при нечетно } n \\ \frac{2^{n-1} + 1}{3}, & \text{при четно } n \end{cases}$$

Тъй като ограничението за n е 60, то 64-битово цяло число ще е достатъчно да обхване този брой. По-долу е даден вариант на решението, написан на C/C++:

```
#include <iostream>
using namespace std;
int main(void)
{
    int n;
    long long c;
    cin >> n;
    c = (long long) 1 << (n-1);
    if (n & 1) c--;
    else c++;
    cout << c / 3 << endl;
}
```

Група D

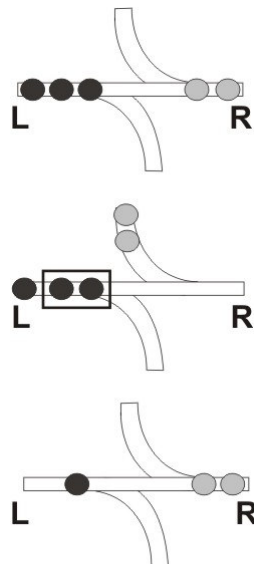
Задача D1. ЧАЙКА, Галина Момчева

На една от улиците в курортен комплекс „Чайка“ е паднал пътният знак, който показвал, че улицата е еднопосочна. Автомобили, идващи от двете посоки трябва да се разминат. Но как?

Случайно минаващият от там пешеходец (и начинаещ програмист) Станчо измислил алгоритъм за разминаване. За щастие има достатъчно дълги отбивки в дясно за всяка от колоните. Правилата, които Станчо измислил (според него справедливи) са следните:

1. Дава знак на колоната с по-малко коли да влезе в отбивката си. Ако колите от двете страни са по равен брой, тогава в отбивката влизат колите отляво.
2. Пуска да минат през кръстовището толкова коли от по-дългата колона, колкото е имало в по-късата.
3. Връща колите от отбивката обратно на улицата.

След изпълнението на стъпки 1, 2 и 3 следва нова оценка на ситуацията, която се регулира по същите правила. Не е изключено, обаче, по време на изпълнението на стъпки 1, 2 и 3 да се появят нови коли и от двете страни. Напишете програма **ЧАЙКА**, която да помогне на Станчо в регулирането на движението.



Вход

На първия ред на стандартния вход са зададени две цели числа: общият брой N на пристигналите отляво и отдясно коли и броят C на повторенията на стъпките 1, 2 и 3, $0 < N \leq 100$, $0 < C \leq 20$. Следва последователност от $N + C$ реда, на всеки от които има по една от латинските букви L, R и P, като броят на редовете с букви P е точно C . Буквата L означава, че е пристигнала кола отляво, а буквата R – отдясно. Буквата P означава, че е извършено регулиране. Входът започва с буква, различна от P и завършва с буква P. При всяко регулиране, във всяка от двете посоки ще има поне по една кола.

Изход

На първите C реда на стандартния изход, за всяко изпълняване на стъпките 1, 2 и 3, се извежда колко коли са минали през кръстовището и от коя страна на кръстовището са дошли. На последните два реда се извеждат броя на колите останали след последното регулиране в кръстовището, съответно от лявата и дясната част на пътя. Ако няма такива се извежда 0.

ПРИМЕР

Вход	Изход
7 2	2 L
L	1 R
R	1
L	3
R	
L	
P	
R	
R	
P	

Решение

Броят на колите, пристигнали в кръстовището отляво, ще съхраняваме в променливата l , а тези пристигнали отдясно – в променливата r , които в началото имат стойности нула. След това $n+c$ пъти се извършва четене на поредният ред и увеличаване на брояча за левите (ако буквата е L), увеличаване на брояча за десните (ако буквата е R) или извършване на регулиране (ако буквата е P). Регулирането е определяне на по-малкото от l и r . Ако r е по-малко от l , то колите от дясно влизат в отбивката и същият брой коли от ляво преминават кръстовището, т.е. трябва да се изведе броят на колите отляво и буквата L и да се отчете, че колите в ляво са намалели с r . Аналогично, ако l е по-малко или равно на r .

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int n, c;
    cin >> n >> c;
    n += c;
```

```

int l = 0, r = 0;
string inp;
for (int i = 0; i < n; ++i)
{
    cin >> inp;
    if (inp == "L") ++l;
    if (inp == "R") ++r;
    if (inp == "P")
    {
        cout << min(l, r) << " ";
        if (l > r)
        {
            cout << "L" << endl;
            l -= r;
        }
        else
        {
            cout << "R" << endl;
            r -= l;
        }
    }
}
cout << l << endl << r << endl;
return 0;
}

```

Задача D2. ЧИСЛО НА ЕДИНГТЪН, автор Красимир Манев

Всеки уважаващ себе си велосипедист трябва да знае собственото си *число на Едингтън*, а то е най-голямото цяло неотрицателно E такова, че поне E пъти в живота си е изминавал с велосипеда си поне E метра без почивка. Естествено, ако един човек никога не е карал велосипед, числото му на Едингтън е нула. Но младият програмист Станчо е и запален велосипедист. Научавайки за числото на Едингтън той решил да пресметне, колко е то в неговия случай. Записал на един лист броя N на случаите, в които е бил на седлото на любимия си „байк“ без почивка и по колко метра е изминал във всеки от тях. Започнал да пресмята, но числата били толкова много, че скоро се забъркал в сметките. Знанията му по програмиране, пък, не били достатъчни и затова оставя на Вас да се справите с проблема. Напишете програма **EDI**, която да пресмята числото на Едингтън.

Вход

На първия ред на стандартния вход ще бъде зададено числото N , $1 \leq N \leq 100000$. Всеки от следващите N реда ще съдържа по едно цяло положително число L , дължината в метри измината при поредното каране $1 \leq L \leq 10000$.

Изход

На стандартния изход програмата трябва да изведе намереното число на Едингтън.

ПРИМЕР

Вход	Изход
10	5
3	
4	
12	
5	
2	
6	
2	
8	
1	
7	

Решение

Това, че числото N е голямо, е без значение. Решението на задачата, което предлагаме, ще може да работи и с много по-големи стойности на N – милиони и даже милиарди. Важното е, че при всяко каране Станчо е изминавал не повече от 10000 метра. Затова дефинираме масив `int a[10001]` и даваме стойност нула на елементите му (първият цикъл `for`). В него ще натрупаме статистиката за каранията, които е осъществил Станчо – в `a[i]` ще намерим колко пъти е изминавал по i метра (вторият цикъл `for`). Числото на Едингтън намираме в променливата E с цикъла `while`. Първата стойност, която проверяваме е 10000. За да има Станчо число на Едингтън 10000 е необходимо `a[10000] ≥ 10000`. Ако това е така, изпълнението на цикъла ще се прекрати и програмата ще изведе 10000. Ако това не е така, за да има Станчо число на Едингтън 9999 е необходимо `a[9999] + a[10000] ≥ 9999`. Затова, в тялото на цикъла, към съдържанието на `a[E-1]` (в случая `a[9999]`) добавяме съдържанието на `a[E]` (в случая `a[10000]`) и намаляваме стойността на E (след `E--` в променливата E ще имаме 9999). Така ще накараме оператора `while` да сравни броя на каранията от поне 9999 метра с 9999 и т.н. На всяка стъпка на цикъла, при която E още не е търсеното число на Едингтън, ще добавяме общия брой на каранията от поне E метра към броя на тези от $E-1$ метра, за да намерим броя на каранията от поне $E-1$ метра и да го сравним с $E-1$. Тъй като дължините са положителни числа, натрупваната сума ще расте, а числото в E ще намалява и неравенството `a[E] < E`, рано или късно, ще бъде нарушено. Изпълнението на цикъла ще се прекрати и съдържанието на E ще бъде търсеното число на Едингтън.

```

#include <iostream>
using namespace std;
int main()
{
    int a[10001], N, i, L, E;
    cin >> N;
    for (i = 0; i <= 10000; i++) a[i] = 0;
}

```

```

for(i=1;i<=N;i++)
{  cin >> L; a[L]++; }
E=10000;
while(a[E]<E) a[E-1]+=a[E--];
cout << E << endl;
return 0;
}

```

Друг начин да се реши задачата, не толкова бърз и рационален при използването на памет, е да се съхранят данните в масив с максимум 100000 елемента и да се сортират елементите му в намаляващ ред. Отговорът е най-големият индекс, за който стойността на масива е по-голяма или равна на индекса.

```

#include <cstdio>
#include <vector>
#include <algorithm>
#include <functional>

using namespace std;

int main()
{
    int n;
    vector<int> dist;
    scanf("%d",&n);
    dist.resize(n);
    for (int i=0;i<n;++i)
        scanf("%d",&dist[i]);
    sort(dist.begin(),dist.end(),greater<int>());
    for (int i= 0;i<n && dist[i]>=i+1;++i);
    printf("%d\n", i);
    return 0;
}

```

Бързината на алгоритъма за сортиране е от съществено значение за това решение на задачата. Предложеното решение използва вградения QSORT. Всъщност, натрупването на статистика в първото решение също е вид сортиране – чрез броене. То е възможно, защото броят на различните възможни числа е ограничен до 10000.

Задача D3. БАНКОВИ СМЕТКИ, автор Венета Богданова

Напоследък в пресата се срещат съобщения, че личните данни на гражданите не са добре защитени. На първокурсниците от специалност “Приложна математика” в един университет поставили за домашно да измислят начин да кодират номерата на банкови сметки IBAN. Росица измислила едно правило, но то е доста трудно за прилагане. Идеята е следната:

- Номерът на сметка IBAN е последователност от 22 знака – главни английски букви и цифри. Например BGFINV1234KL0987BC1234.
- Всеки знак от номера IBAN се заменя с двойка цифри. Пред цифрите от 1 до 9 се добавя нула, а латинските букви се кодират по следния начин ‘A’

с 10, ‘B’ с 11, ‘C’ с 12, ‘D’ с 13 и т.н. до ‘Z’ с 35, с изключение на буквата ‘O’ – тя, както и цифрата ‘0’, се кодира с две нули.

A	B	C	D	E	F	G	H	I	J	K	L	M
10	11	12	13	14	15	16	17	18	19	20	21	22
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
23	00	25	26	27	28	29	30	31	32	33	34	35

Така номерът BGFINV1234KL0987BC1234 се преобразува до
11161518233101020304202100090807111201020304

- Полученото в стъпка 2 число се умножава с 11 и това е окончателно кодираната сметка. Т.е. за горния пример се получава:
122776700564111223346223100998878223211223344.

Помегнете на Росица в затруднението. Напишете програма **IBAN**, която да извършва описаното кодиране на номера на банкови сметки.

Вход

На единствения ред на стандартния вход ще бъде зададен номер на банкова сметка IBAN. Той ще се състои от 22 знака – главни английски букви и цифри.

Взход

На стандартния изход програмата трябва да изведе кодираният номер на сметката.

ПРИМЕР 1

Вход

BGFINV1234KL0987BC1234

Изход

122776700564111223346223100998878223211223344

ПРИМЕР 2

Вход

0TTTTTTTTTTTTTTTTTTTTTTT

Изход

0032219

Решение

И двете предложени решения извършват една и съща последователност от действия, но използват различни структури от данни. Прочита се дадената банкова сметка. Замества се всеки знак с два знака по описаното правило. Умножението по 11 всъщност е събиране на полученото число и същото число, с добавена една нула в края, т.е. десет пъти по-голямо от първото.

Вариант 1:

```

#include<iostream>
#include<fstream>
#include<string.h>
using namespace std;

```

```

int main()
{ char s[23],ns[45];
  char b[36][3]=
    {"00","01","02","03","04","05","06","07","08",
     "09","10","11","12","13","14","15","16","17",
     "18","19","20","21","22","23","00","25","26",
     "27","28","29","30","31","32","33","34","35"};

  int i,k,pr=0,sum;
  int v1[45],v2[45];
  // въвеждане на банковата сметка
  cin>>s;
  // удвояване на знаците в нея
  strcpy(ns,"");
  for(i=0;s[i];i++)
  { if ((s[i]>='0')&& (s[i]<='9'))
    strcat(ns,b[s[i]-'0']);
    if ((s[i]>='A')&& (s[i]<='Z'))
    strcat(ns,b[10+s[i]-'A']);
  }
  k=strlen(ns);
  // създаване на двата масива с цифрите за събиране
  for(i=0;i<k;i++) v2[i]=v1[i+1]=ns[i]-'0';
  v2[k]=0;
  v1[0]=0;
  // събиране
  for(i=k-1;i>=0;i--)
  { sum=pr+v1[i]+v2[i];
    v1[i]=sum%10;
    pr=sum/10;
  }
  // извеждане на резултата
  for(i=0;i<=k;i++) cout<<v1[i];
  cout<<endl;
  return 0;
}

```

Вариант 2:

```

#include<iostream>
#include<string>
#include<vector>
using namespace std;
string iban;
vector<int> a,b;
vector<int> ans;
int i,t;
int main ()
{
  // въвеждане на банковата сметка
  cin>>iban;
  // удвояване на символите в нея –
  // данните се съхраняват във вектора a

```

```

for (i=iban.length()-1;i>=0;i--)
  if (isdigit(iban[i]))
  { a.push_back(iban[i]-'0');
    a.push_back(0);
  }
  else if (iban[i]=='o')
  { a.push_back(0);
    a.push_back(0);
  }
  else
  { t=iban[i]-'A'+10;
    a.push_back(t%10);
    a.push_back(t/10);
  }
  // създаване на вторият вектор съдържащ 10 пъти
  // по-голямото число
  b.push_back(0);
  b.insert(b.end(),a.begin(),a.end());

  // събиране
  t=0;
  for (i=0;i<a.size();i++)
  {
    t+=a[i]+b[i];
    ans.push_back(t%10);
    t/=10;
  }
  ans.push_back(t+b.back());

  // извеждане на резултата
  for (i=ans.size()-1;i>=0;i--)
    cout<<ans[i];
  cout<<endl;
  return 0;
}

```

Група Е

Задача Е1. ТОЧНОСТ, автор Пламена Христова

Първият учебен час във всяко училище започва в точно определено време (час и минути). Учениците трябва да са в клас 5 минути преди това. Виктор е любопитен и иска да знае дали ще закъснее за училище, ако майка му го събуди в зададен час и минути. Той знае за колко минути прави сутрешния си тоалет, колко минути закусва, колко минути приготвя чантата си и, че отива до училище за четири пъти повече време от сумарното време за сутрешен тоалет и приготвяне на чантата. Помогнете му да бъде точен. Напишете програма **ЕХАСТ**, която да определи дали Виктор ще закъснее за училище.

Вход

Програмата трябва да въвежда данните от три реда. На първия ред ще бъдат зададени две числа – часът и минутите на започване на учебните занятия. На втория ред също ще бъдат зададени две числа – часът и минутите, когато

майката на Виктор го събужда. На третия ред числата са три – необходимия брой минути за сутрешния тоалет на Виктор, необходимия брой минути за закуска и необходимия брой минути за подготовка на чантата. Всички минути във входните данни са цели числа от 0 до 59.

Изход

Програмата трябва да изведе **Yes**, ако при така зададените времена Виктор ще закъснее за училище и **No**, ако няма да закъснее.

ПРИМЕР 1

Вход
8 0
6 30
7 10 2

Изход
No

Решение

От времето на започване на учебните занятия (час **h** и минути **m**) изваждаме 5 минути, за да получим в колко часа най-късно Виктор трябва да бъде в училище. Правим това внимателно, защото **m** може да е по-малко от 5. След това пресмятаме колко време общо е необходимо на Виктор за сутрешен тоалет, закуска, приготвяне на чантата и път до училище (**ob**). Към това време добавяме и минутите на ставане (**wm**). След това преобразуваме полученото време на пристигане в училище в нормален вид (час $lh = wh + ob / 60$ и минути $lm = ob \% 60$). Това време сравняваме с времето, в което най-късно Виктор трябва да е на училище, за да не закъснее и извеждаме съответно **Yes** или **No**. Първо сравняваме двата часа (**lh** и **h**) и само когато те са равни, сравняваме и минутите (**lm** и **m**).

Вариант на C/C++

```
#include <iostream>
using namespace std;
int main()
{ unsigned short int h,m,wh,wm,st,mz,tb,ob,p,lh,lm;
  cin>>h>>m; cin>>wh>>wm; cin>>st>>mz>>tb;
  if (m>=5) m=m-5; else {h=h-1; m=m+60-5;}
  p=4*(st+tb); ob=st+mz+tb+p+wm;
  lh=wh+ob/60; lm=ob%60;
  if (lh<h) cout<<"No"<<endl;
  else if ((lh==h) && (lm<=m)) cout<<"No"<<endl;
  else cout<<"Yes"<<endl;
  return 0;
}
```

Вариант на Pascal

```
program Exact;
var
  h,m,wh,wm,st,mz,tb,ob,p,lh,lm:integer;
begin
  readln(h,m); readln(wh,wm); readln(st,mz,tb);
```

```
if m>=5 then m:=m-5
else
begin
  h:=h-1; m:=m+60-5;
end;
p:=4*(st+tb);
ob:=st+mz+tb+p+wm;
lh:=wh+ob div 60;
lm:=ob mod 60;
if lh<h then writeln('No')
else
  if (lh=h) and (lm<=m) then writeln('No')
  else writeln('Yes');
```

end.

Задача E2. ТЕКСТ, автор Бисерка Йовчева

Виктор току що е научил, че една дума се нарича *палиндром* когато се чете по един и същ начин отляво надясно и отдясно наляво. Такива са думите “капак”, “невен”, “боб” и др. Съставете програма **ТЕКСТ**, която проверява дали зададена дума е палиндром.

Вход

Програмата трябва да прочете зададената дума от клавиатурата. Думата ще бъде съставена от седем букви на латинската азбука (както малки така и главни, като съответните малка и главна буква считаме за еднакви).

Изход

Ако въведената дума е палиндром, програмата трябва да я изведе с главни букви, а ако не е палиндром – с малки.

ПРИМЕР 1

Вход
abcdCba
Изход
ABCDcba

ПРИМЕР 2

Вход
Tarator
Изход
tarator

Решение

За да решим задачата първо трябва да прочетем думата, буква по буква, и да проверим дали е палиндром. Една седембуквена дума е палиндром, ако са еднакви (включително голяма и малка от един вид) следните двойки букви: първа и седма, втора и шеста, трета и пета. Ако думата е палиндром, ще преобразуваме всички малки букви в нея до съответните главни, а ако не е палиндром – всички главни в малки. Една малка буква се превръща в главна като към стойността и се добави 'A' – 'a', а за обратното преобразуване – като се добави 'a' – 'A'. Тъй като и в двата случая към буквата се прибавя една и съща стойност, добре е да я пресметнем еднократно в началото. Преобразуването на буквите се осъществява за всяка една буква поотделно. Предлагаме две различни решения, в зависимост от това дали учениците познават масивите от знаци или не. И в двата случая използваме цялата

променливата l като Булева, за да помним дали думата е палиндром, а в променливата d запомняме стойността на 'A'-'a' или 'a'-'A'.

Вариант 1 (без използване на масив): В този случай буквите на думата съхраняваме в седем отделни знакови променливи c1, c2, c3, c4, c5, c6, c7 – по една за всяка от буквите.

```
int main()
{
    char c1, c2, c3, c4, c5, c6, c7;
    int d, l;
    cin.get(c1); cin.get(c2);
    cin.get(c3); cin.get(c4);
    cin.get(c5); cin.get(c6);
    cin.get(c7);
    l = c1 == c7 || c1 == c7 + 'A' - 'a' || c7 == c1 + 'A' - 'a';
    l = l && (c2 == c6 || c2 == c6 + 'A' - 'a' || c6 == c2 + 'A' - 'a');
    l = l && (c3 == c5 || c3 == c5 + 'A' - 'a' || c5 == c3 + 'A' - 'a');
    d = l ? ('A' - 'a') : ('a' - 'A');
    if (l && c1 >= 'a' && c1 <= 'z' || !l && c1 >= 'A' && c1 <= 'Z') c1 += d;
    if (l && c2 >= 'a' && c2 <= 'z' || !l && c2 >= 'A' && c2 <= 'Z') c2 += d;
    if (l && c3 >= 'a' && c3 <= 'z' || !l && c3 >= 'A' && c3 <= 'Z') c3 += d;
    if (l && c4 >= 'a' && c4 <= 'z' || !l && c4 >= 'A' && c4 <= 'Z') c4 += d;
    if (l && c5 >= 'a' && c5 <= 'z' || !l && c5 >= 'A' && c5 <= 'Z') c5 += d;
    if (l && c6 >= 'a' && c6 <= 'z' || !l && c6 >= 'A' && c6 <= 'Z') c6 += d;
    if (l && c7 >= 'a' && c7 <= 'z' || !l && c7 >= 'A' && c7 <= 'Z') c7 += d;
    cout << c1 << c2 << c3 << c4 << c5 << c6 << c7 << endl;
    return 0;
}
```

Вариант 2 (с използване на масив): В този случай буквите на думата съхраняваме в елементите на масива c[7].

```
#include <iostream>
using namespace std;
int main()
{
    char c[7]; int d, l, i;
    cin >> c;
    l = c[0] == c[6] || c[0] == c[6] + 'A' - 'a' ||
        c[6] == c[0] + 'A' - 'a';
    l = l && (c[1] == c[5] || c[1] == c[5] + 'A' - 'a' ||
        c[5] == c[1] + 'A' - 'a');
    l = l && (c[2] == c[4] || c[2] == c[4] + 'A' - 'a' ||
        c[4] == c[2] + 'A' - 'a');
    d = l ? ('A' - 'a') : ('a' - 'A');
    for (i = 0; i < 7; i++)
        if (l && c[i] >= 'a' && c[i] <= 'z' || !l && c[i] >= 'A' &&
            c[i] <= 'Z') c[i] += d;
    cout << c << endl;
}
```

Задача Е3. ЛИНИЙКА, автор Бистра Танева

Виктор има N летвички ($2 \leq N \leq 100$) и се пита какви са дължините на най-дългата и най-късата летвичка. За целта решил да използва линейка с дължина 100 см, на която са нанесени деления през 1 см. Оказало се, че дължините на всички летвички са цели числа не надминаващи 100. Виктор обича да си създава трудности и вместо да измерва дължините, започвайки от началото на линейката, поставял единия край на летвичката на произволно деление на линейката и си записвал мястото на двата ѝ края. При това даже не подреждал двете числа по един и същ начин. Сега ще му трябва програма **RULER**, която да намери най-голямата и най-малката дължина.

Вход

Програмата трябва да въвежда данните от клавиатурата. На първия ред ще бъде зададено числото N , а на всеки от следващите N реда по една двойка цели числа, разделени с интервал – мястото на краищата на една от летвичките върху линейката.

Изход

Програмата трябва да изведе на един ред, разделени с интервал, дължините на най-голямата и най-малката летвичка.

ПРИМЕР

Вход	Изход
5	25 4
3 9	
8 12	
15 6	
10 19	
0 25	

Решение

Преди да започнем обработката, задаваме като стойност на променливата **max** най-малката възможна дължина 0, а на променливата **min** – най-голямата възможна дължина 100. В променливите **a1** и **a2** въвеждаме двойката числа, показващи местата на краищата на летвичката върху линейката. Тъй като не е указано кой край е първи (този намиращ се по-близо до началото на линейката или по-далечния), проверяваме кое от въведените числа по-голямо. От него изваждаме другото число, за да намерим дължината на летвичката, която запомняме в променливата **a**. За пресметнатата дължина проверяваме дали не е по-малка от намерения до момента минимум или по-голяма от намерения до момента максимум и, ако се налага, заменяме старата стойност с новата. Дължината може да намерим и като пресметнем абсолютната стойност на разликата **a1-a2** (с функцията **abs** или като умножим с -1 разликата, ако е отрицателно число).

Начин: Без да се използва вградената функция **abs**.

```
#include <iostream>
using namespace std;
int main()
{ unsigned short int a1, a2, max=0, min=100, n;
```

```

cin>>n;
for(int i=0;i<n;i++)
{  cin>>a1>>a2;
   if(a1>a2) a=a1-a2;
   else a=a2-a1;
   if(max<a1) max=a1;
   if(min>a1) min=a1;
}
cout<<max<<" "<<min<<endl;
return 0;
}

```

II начин: Като се използва вградената функция `abs`.

```

#include <iostream>
using namespace std;
int main()
{  unsigned short int a1,a2,max=0,min=100,n;
   cin>>n;
   for(int i=0;i<n;i++)
   {  cin>>a1>>a2;
      a=abs(a1-a2);
      if(max<a1) max=a1;
      if(min>a1) min=a1;
   }
   cout<<max<<" "<<min<<endl;
   return 0;
}

```