Чернігівський обласний інститут післядипломної
педагогічної освіти імені К.Д. Ушинського

Чернігівська обласна Інтернет-школа
«Юний програміст»

ЗБІРНИК ЗАДАЧ ТА РОЗВ'ЯЗКІВ ІЗ ПРОГРАМУВАННЯ

ЧАСТИНА 3

choippo.cn.sch.in.ua

Чернігів – 2017

Автори:

Хрол Н.П., Бондаренко С.М., Гах С.О., Хорошок С.В., Голодяєва П.В., Пупов Н.А., Чорний А.В.

Рецензенти:

Горошко Ю. В., завідувач кафедри інформатики та обчислювальної техніки Чернігівського національного педагогічного університету імені Т. Г. Шевченка, доктор педагогічних наук, професор

Покришень Д. А., завідувач кафедри інформатики та інформаційно-комунікаційних технологій в освіті Чернігівського обласного інституту післядипломної педагогічної освіти імені К. Д. Ушинського, кандидат педагогічних наук, доцент

Загальна редакція:

Літош Ю. М., старший викладач кафедри інформатики та інформаційно-комунікаційних технологій в освіті Чернігівського обласного інституту післядипломної педагогічної освіти імені К. Д. Ушинського

Баранова О. Є., методист відділу природничо-математичних дисциплін Чернігівського обласного інституту післядипломної педагогічної освіти імені К. Д. Ушинського

Смірнова О.М., методист відділу природничо-математичних дисциплін Чернігівського обласного інституту післядипломної педагогічної освіти імені К. Д. Ушинського

Збірник задач та розв'язків із програмування / Н. П. Хрол, С. М. Бондаренко, С. О. Гах та ін.; за заг. ред. Ю. М. Літоша, О. Є. Баранової, О. М. Смірнової. – Чернігів: ЧОІППО імені К. Д. Ушинського, 2017. – Ч.3. – 83 с.

Рекомендовано до друку
вченою радою Чернігівського обласного інституту
післядипломної педагогічної освіти імені К. Д. Ушинського
(протокол № 4 від21.12.2017 р..)

ЗМІСТ

Задачі з використанням рядкових величин

Ідеї розв'язань задач

Гаха С.О., учителя інформатики та математики Ніжинської гімназії № 3 Ніжинської міської ради

ГОЛОСНІ

До голосних літер в латинському алфавіті відносяться літери **A**, **E**, **I**, **O**, **U** і **Y**. Інші літери вважаються приголосними. Напишіть програму, яка підраховує кількість голосних літер в тексті.

Вхідні дані

У вхідному файлі міститься один рядок тексту, який складається лише із заглавних латинських літер та проміжків. Довжина рядка не перевищує **100** символів.

Вихідні дані

У вихідний файл вивести одне ціле число – кількість голосних у вхідному тексті.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
COBRA	2

Розв'язання

Якщо виконати перебір масиву, порівнюючи кожний елемент відразу зі всіма «голосними» через АБО, то середовище тестування через перевищення по часу зарахує лише 37% відповідей. Тому доцільніше зробити наступним чином:

```
#include <iostream>
#include <stdio.h> // для gets
#include <cstring>
using namespace std;
int main (){
int length,i,j,count=0;
char vowel []="AEIOUY";
char text [100];
gets (text); //зчитую масив разом з пробілами поки не буде натиснено Enter
length=strlen (text);// довжина введеного рядка
for (j=0; j<=5; j++){
        for (i=0; i<=length; i++)
                if (text[i]==vowel[j])//порівняння з масивом «Голосних»
                        count++;
}
cout << count << endl;
return 0;
```

ШИФР ЮЛІЯ

Юлій Цезар використовував свій спосіб шифрування тексту. Кожна літера мінялась на наступну за алфавітом через **k** позицій по колу. Необхідно за заданою шифровкою встановити початковий текст.

Вхідні дані

У першому рядку задано шифровку, яка складається з не більш ніж **255** великих латинських літер. У другому рядку число **k (1 ≤ k ≤ 10).**

Вихідні дані

Вивести результат розшифровки.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
XPSE 1	WORD
ZABC3	WXYZ

Розв'язання

Щоб розшифрувати рядок cipher, необхідно замінити їхні позиції зі зміщенням на (find—k) позицій рядка alp, де alp — рядок з латинським алфавітом, а find — позиція букви, яку потрібно змінити. У випадку, якщо різниця find і k менша за нуль, буква рядка cipher заміняє на букву, яка знаходиться на (26—(k—find)) позицій від рядка alp, тобто не враховуючи ту кількість позицій, яка вже була пройдена від початкового символа до першого в рядку alp.

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

int main() {
string cipher;
getline(cin,cipher);
int k;
const int ABC_SIZE = 26; // Розмір латинської абетки
cin >> k;
string alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; // Простіше ніж використовувати таблицю кодів ASCII
for(int i = 0; i < cipher.length(); i++){
   int find = cipher[i] - 'A';
   if(find - k < 0) cipher.at(i) = alp.at(ABC_SIZE - (k - find          else
cipher.at(i) = alp.at(find-k); )); // індекси позицій як аргументи

}
cout << cipher;
return 0;
}
```

ЗАЙВІ ПРОПУСКИ

Задано рядок. Напишіть програму, яка видалить з цього рядка усі зайві пропуски. Пропуск будемо вважати зайвим, якщо:

- він знаходиться на самому початку рядка, до самого першого слова;
- він знаходиться у кінці рядка, після самого останнього слова;
- декілька пропусків розміщені між двома словами (простіше кажучи, якщо слова розділені більш ніж одним пропуском, тоді усі пропуски крім одного — зайві).

Вхідні дані

Задано рядок **S** (**0** $\leq$ |**S**| $\leq$ **255**). Рядок містить лише латинські літери і пропуски.

Вихідні дані

Потрібно вивести рядок без зайвих пропусків.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
Alexandr Sergeevich Pushkin	Alexandr Sergeevich Pushkin

Розв'язання

Знайшовши два пропуски, які розміщені в сусідніх позиціях рядка, потрібно видалити той, - який знаходиться «праворуч».

```
#include <iostream>
#include <string>

using namespace std;

int main(){
   string s;
   getline(cin, s);
   for (int i = 0; i < s.length(); i++)
      if (s[i] == ' ' && s[i + 1] == ' ') {s.erase(i, 1); i = (i - 1);}
   cout << s << endl;
   return 0;
}
```

Ідеї розв’язань задач

Хорошка С.В., учителя інформатики Коропської загальноосвітньої школи І-ІІІ ступенів ім. Т.Г.Шевченка Коропської селищної ради

КІЛЬКІСТЬ СЛІВ

Є деяке речення на невідомій мові. Порахувати кількість слів у ньому. Літерами алфавіту у невідомій мові є літери латинського алфавіту та арабські цифри. Гарантується, що інших символів, крім пропусків та розділових знаків у реченні, нема.

Вхідні дані

У єдиному рядку дано речення на невідомій мові.

Вихідні дані

Єдине число - кількість слів у ньому.

Ліміт часу **0.1** секунд

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
Hello, world!	2

Аналіз розв'язку: зчитати рядок даних. Перевірити, якщо символ не дорівнює знаку оклику, знаку питання, пробілу, комі, тире, а наступний дорівнює переліченим символам, то лічильник слів збільшуємо.

Після перебору всіх символів будемо знати кількість слів.

Розв'язання

```
var str:string;
i,n,k:integer;
begin
readln(str);
k:=0;
n:=length(str);
for i:=1 to n-1 do
begin
if (str[i]<>'!') and (str[i]<>' ') and (str[i]<>'?') and (str[i]<>',') and (str[i]<>'-') and (str[i+1]='!') then inc(k);
if (str[i]<>'!') and (str[i]<>' ') and (str[i]<>'?') and (str[i]<>',') and (str[i]<>'-') and(str[i+1]=' ') then inc(k);
if (str[i]<>'!') and (str[i]<>' ') and (str[i]<>'?') and (str[i]<>',') and (str[i]<>'-') and(str[i+1]='?') then inc(k);
if (str[i]<>'!') and (str[i]<>' ') and (str[i]<>'?') and (str[i]<>',') and (str[i]<>'-') and(str[i+1]=',') then inc(k) ;
if (str[i]<>'!') and (str[i]<>' ') and (str[i]<>'?') and (str[i]<>',') and (str[i]<>'-') and (str[i+1]='.') then inc(k) ;
end;
writeln(k);
end.
```

АНАГРАМИ

Слово називається анаграмою іншого слова, якщо воно може бути отримано перестановкою його літер.

Вхідні дані

Два слова задані в окремих рядках. Слова складаються з маленьких латинських літер і цифр. Довжини слів не перевищують **255.**

Вихідні дані

Вивести "**YES**", якщо задані слова є анаграмами один одного, та "**NO**" в інакшому випадку.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
sharm marsh	YES
ananas nnaass	NO

Аналіз розв'язку: Створити числовий масив з нумерацією елементів від '0' до 'z'. Зчитати посимвольно перше слово і збільшити у масиві число на одиницю на місці зчитаного символу (індексу). При зчитуванні другого слова зменшити дані числа. Потім перебрати всі елементи масиву на рівність нулю. Якщо знайдеться хоча б один елемент, який не дорівнює нулю, то рядки не будуть анаграмою.

Розв'язання

```
var mas:array['0'..'z'] of longint;
d,i:char;
j:longint;
begin
while not eoln do
begin
read(d);
inc(mas[d]);
end;
readln;
While not eoln do
begin
read(d);
dec(mas[d]);
end;
For i:='0' to 'z' do
if mas[i]<>0 then begin
write('NO');
exit;
```

```
end;
write('YES');
end.
```

ЮВІЛЕЙ ВІННІ-ПУХА

Ось і настав довгоочікуваний Ювілей Вінні-Пуха. У чарівний ліс на свято зібралось багато гостей. У тому числі Вінні-Пух запросив до себе друзів з інших галактик. Нажаль, коли він відсилав запрошення, він зовсім забув, що на планеті, де живуть його друзі інопланетяни, усі читають не зліва направо, а справа наліво. Вінні-Пух розуміє, що до Ювілею вони вже не прилетять, але ведмежа не сумує. Він хоче перевірити, чи правда, що дата його Ювілею, прочитана справа наліво, також існує, і інопланетяни прилетять в інший день. Допоможіть Вінни-Пуху визначити, чи чекати йому в гості інопланетних друзів.

Вхідні дані

Вхідний файл містить дату Ювілею Вінні-Пуха у форматі **dd.mm.gggg**. Гарантується, що дата коректна.

Вихідні дані

У вихідний файл потрібно вивести **YES**, якщо дата, яка читається справа наліво коректна, і **NO** у протилежному випадку.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** МіВ

Вхідні дані	**Вихідні дані**
Sample 1 23.02.2002 Sample 2 20.02.2023	Sample 1 YES Sample 2 NO

Аналіз розв'язку: задачу можна розділити на два етапи.

1. Перевернути дату.
   10 символ рядка буде першою цифрою правильного дня.
   9 символ — другою цифрою.
   8 символ — першою цифрою місяця.
   7 символ — другою цифрою місяця.
   5 символ — першою цифрою року.
   4 символ — другою цифрою року.
   2 символ — третьою цифрою року.
   1символ — четвертою цифрою року.

Для переводу символу у число можна скористатися функцією **val.** Для знаходження правильного дня першу цифру(показує на десяти), числа помножити на 10 та додати другу цифру(показує одиниці). Для знаходження місяця та року дії аналогічні.

2. Перевірити отриману дату на коректність.

Якщо рік високосний, то у лютому 29 днів, якщо ні то 28 днів. У місяцях: січень, березень, травень, липень, серпень, жовтень, грудень 31 день, в інші місяці по 30 днів. При невиконанні хоча б однієї з умов введена дата буде некоректна.

Розв'язання

```
var s,rez:string;
    den,mis,rik,n:integer;
    d1,d2,m1,m2,r1,r2,r3,r4:byte;
begin
  readln(s);
  val(s[10],d1);
  val(s[9],d2);
  val(s[8],m1);
  val(s[7],m2);
  val(s[5],r1);
  val(s[4],r2);
  val(s[2],r3);
  val(s[1],r4);
  den:=d1*10+d2;
  mis:=m1*10 +m2;
  rik:=r1*1000+r2*100+r3*10+r4;
  rez:='YES';
  if (rik mod 4=0) and (mis=2) and (den>29) then rez:='NO';
  if (rik mod 4<>0) and (mis=2) and (den>28) then rez:='NO';
  if ((mis=1) or (mis=3) or (mis=5) or (mis=7) or (mis=8) or (mis=10) or (mis=12))
and (den>31) then rez:='NO';
  if ((mis=4) or (mis=6) or (mis=9) or (mis=11)) and (den>30) then rez:='NO';
  if mis>12 then rez:='NO';
  writeln(rez);
  readln
end.
```

Задачі з використанням основних понять теорії графів. Способи представлення графів

Ідеї розв’язання задач

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи I-III ступенів фізико-математичного профілю № 12 м. Чернігова

ПЕРЕВІРКА НА НЕОРІЄНТОВНІСТЬ

За заданою квадратною матрицею **n** × **n** з нулів та одиниць визначити, чи може вона бути матрицею суміжності простого неорієнтовного графа.

Вхідні дані

У першому рядку задано число **n** (**1** ≤ **n** ≤ **100**). Потім йдуть **n** рядків по **n** елементів у кожному - опис матриці суміжності.

Вихідні дані

Вивести **YES**, якщо граф неорієтований, та **NO** у протилежному випадку.

Ліміт часу **1** секунда

Ліміт використання пам'яті **256** MiB

Вхідні дані	**Вихідні дані**
3 0 1 1 1 0 1 1 1 0	YES
3 0 1 1 1 0 1 0 1 0	NO

Розв'язання

Квадратна матриця не може бути матрицею суміжності неорієнтованого графа, якщо елементи ***A[i,j]<>A[j,i]*** або при ***i=j A[i,j]<>0***.

Лістінг програми:

```
var A:array[1..100,1..100] of integer;
     i,j,n:integer;
begin
readln(n);
for i:=1 to n do
  for j:=1 to n do
   read(A[i,j]);
for i:=1 to n do
  for j:=1 to n do
    begin
    if (A[i,j]<>A[j,i]) then
                          begin
                          write('NO');
                          exit;
                          end;
    if (i=j) and (A[i,j]<>0) then
                             begin
                             write('NO');
                             exit;
                             end;
 end;
```

```
write('YES');
end.
```

ВІД МАТРИЦІ СУМІЖНОСТІ ДО СПИСКІВ СУМІЖНОСТІ

Простий орієнтовний граф задано матрицею суміжності. Виведіть його подання у вигляді списків суміжності.

Вхідні дані

У першому рядку знаходиться кількість вершин графа **n** (**1** ≤ **n**≤ **100**). У другому рядку і далі - матриця суміжності. Гарантується, що граф не містить петель.

Вихідні дані

Виведіть n рядків - списки суміжності графа. В і-му рядку спочатку виведіть кількість ребер, які виходять з і-ої вершини, а потім - номери вершин, у які ці ребра входять, впорядковані за зростанням.

Ліміт часу 1 секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
5 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0	1 3 2 1 3 1 5 2 1 2 2 1 2

Розв'язання

Наявність ребра між вершинами ***i*** і ***j*** в орієнтованому графі визначається ***A[i,j]=1*** в матриці суміжності. Для кожного рядка матриці суміжності знаходимо кількість елементів рівних 1 та виводимо номери стовпців цих елементів.

Лістінг програми:

```
var A:array[1..100,1..100] of integer;
   n,i,j,s:integer;
begin
readln(n);
for i:=1 to n do
  for j:=1 to n do
  read(A[i,j]);
for i:=1 to n do
  begin
  s:=0;
  for j:=1 to n do
    if a[i,j]=1 then s:=s+1;
  write(s,' ');
  for j:=1 to n do
```

```
   if A[i,j]=1 then write(j,' ');
  writeln;
  end;
end.
```

ПІВСТЕПЕНІ ВЕРШИН

Орієнтовний граф задано матрицею суміжності. Знайдіть півстепені заходу та півстепені виходу усіх вершин графа (тобто кількості ребер, які входять у неї, та виходять з неї відповідно для кожної вершини).

Вхідні дані

N - число вершин у графі ($1 \le N \le 100$), потім матриця суміжності: N рядків по N чисел, кожне з яких дорівнює 0 або 1.

Вихідні дані

Виведіть N пар чисел: для кожної вершини спочатку півстепінь заходу і потім півстепінь виходу.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
4 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1	2 2 3 3 2 1 3 4

Розв'язання

В матриці суміжності знаходимо в кожному рядку кількість елементів ***s1*** рівних 1 – півстепінь виходу та в кожному стовпцю кількість елементів ***s2*** рівних 1 – півстепінь заходу.

Лістінг програми:

```
var A:array[1..100,1..100] of integer;
   i,j,n,s1,s2:integer;
begin
readln(n);
for i:=1 to n do
  for j:=1 to n do
  read(A[i,j]);
for i:=1 to n do
  begin
  s1:=0;
  s2:=0;
  for j:=1 to n do
    begin
    if A[i,j]=1 then s1:=s1+1;
```

```
    if A[j,i]=1 then s2:=s2+1;
    end;
  writeln(s2,'  ',s1);
  end;
end.
```

ВИТОКИ ТА СТОКИ

Вершина орієнтовного графа називається *витоком*, якщо в неї не входить жодне ребро, і *стоком*, якщо з неї не виходить жодного ребра.

Орієнтовний граф задано матрицею суміжності. Знайдіть усі його вершини-витоки та усі вершини-стоки.

Вхідні дані

Перший рядок містить кількість вершин у графі **n** (**1** ≤ **n** ≤ **100**), далі йде матриця суміжності – **n** рядків по **n** чисел, кожне з яких дорівнює **0** або **1.**

Вихідні дані

У першому рядку виведіть кількість витоків у графі, потім номери вершин, які є витоками у порядку зростання. У другому рядку виведіть інформацію про стоки у такому ж форматі.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	Вихідні дані
5 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	2 3 4 3 1 4 5

Розв'язання

В матриці суміжності знаходимо кількість стовпців, всі значення в яких дорівнюють 0 та номери цих стовпців. Цим ми знайдемо кількість витоків у графі та номери вершин, які є витоками.

Аналогічно знаходимо кількість рядків, всі значення в яких дорівнюють 0 та номери цих рядків – інформація про стоки.

Лістінг програми:

```
var A:array[1..100,1..100] of integer;
   V,S:array[1..100] of integer;
   i,j,n,s1,s2,k1,k2:integer;
begin
readln(n);
for i:=1 to n do
  for j:=1 to n do
  read(A[i,j]);
k1:=0;
```

```
k2:=0;
for i:=1 to n do
  begin
  s1:=0;
  s2:=0;
  for j:=1 to n do
    begin
    if A[i,j]=1 then s1:=s1+1;
    if A[j,i]=1 then s2:=s2+1;
    end;
  if s1=0 then begin
               k1:=k1+1;
               S[k1]:=i;
               end;
  if s2=0 then begin
               k2:=k2+1;
               V[k2]:=i;
               end;
  end;
write(k2,'  ');
for i:=1 to k2 do
  write(V[i],'  ');
writeln;
write(k1,'  ');
for i:=1 to k1 do
  write(S[i],'  ');
end.
```

ПОВНИЙ ГРАФ

Неорієнтовний граф називається *повним*, якщо довільна пара його різних вершин з'єднана хоча б одним ребром. Для заданого списком ребер графа перевірте, чи є він повним.

Вхідні дані

На вхід подаються кількість вершин **n** (**1** ≤ **n** ≤ **100**) та кількість ребер **m** (**1** ≤ **m** ≤ **10000**). Далі йдуть **m** пар чисел - ребра графа.

Вихідні дані

Виведіть **YES**, якщо граф є повним, і **NO** у протилежному випадку.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
3 3 1 2 1 3 2 3	YES

Розв'язання

Перетворюємо список ребер на матрицю суміжності.

В матриці суміжності визначаємо для кожного рядка кількість елементів ***s*** рівних 1. Якщо ***s<n-1***, то граф неповний.

Лістінг програми:

```
var A:array[1..100,1..100] of integer;
   i,j,n,s,m,k:longint;
begin
readln(n,m);
for k:=1 to m do
 begin
 read(i,j);
 A[i,j]:=1;
 A[j,i]:=1;
 end;
for i:=1 to n do
 begin
 s:=0;
 for j:=1 to n do
  if (A[i,j]=1) and (i<>j) then s:=s+1;
 if s<n-1 then begin
            write('NO');
            exit;
            end;
 end;
write('YES');
end.
```

КІЛЬКІСТЬ РЕБЕР У НЕОРІЄНТОВАНОМУ ГРАФІ

Простий неорієнтований граф задано матрицею суміжності.

Знайти кількість ребер у графі.

Вхідні дані

У першому рядку вхідного файлу задано число **N (1 ≤ N≤ 100)**. Потім йде **N** рядків по **N** елементів у кожному - опис матриці суміжності.

Вихідні дані

У вихідний файл виведіть єдине число - кількість ребер у графі.

Вхідні дані	**Вихідні дані**
3 0 1 0 1 0 1 0 1 0	2

Розв'язання

Матриця суміжності неорієнтованого графа є симетричною відносно головної діагоналі, тобто кожне ребро задається два рази. Підраховуємо в матриці суміжності кількість елементів ***k***, які дорівнюють 1 та виводимо половину значення змінної ***k***.

Лістінг програми:

```
var a,i,j,k,n:integer;
begin
readln(n);
k:=0;
for i:=1 to n do
  for j:=1 to n do
  begin
  read(a);
  if a=1 then k:=k+1;
  end;
write(k div 2);
end.
```

ПЕТЛІ

За заданою матрицею суміжності неорієнтовного графа визначте, чи містить він петлі.

Вхідні дані

У першому рядку вхідного файлу задано число **N** (**1** ≤ **N**≤ **100**). Потім йде **N** рядків по **N** елементів у кожному - опис матриці суміжності.

Вихідні дані

У вихідний файл вивести "**YES**", якщо граф містить петлі, і "**NO**" у протилежному випадку.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** МіВ

Вхідні дані	**Вихідні дані**
3 0 1 1 1 0 1 1 1 0	NO

Розв'язання

Якщо на головній діагоналі матриці суміжності є елемент, який дорівнює 1, то граф має петлю.

Лістінг програми:

```
var a, i,j,n:integer;
begin
readln(n);
for i:=1 to n do
  for j:=1 to n do
```

```
 begin
 read(a);
 if (i=j) and (a=1) then
                    begin
                    write('YES');
                    exit;
                    end;
 end;
write('NO');
end.
```

ВІД МАТРИЦІ СУМІЖНОСТІ ДО СПИСКУ РЕБЕР

Простий неорієнтовний граф задано матрицею суміжності.

Виведіть його подання у вигляді списку ребер.

Вхідні дані

У першому рядку вхідного файлу задано число **N** (**1** ≤ **N**≤ **100**). Потім йде **N** рядків по **N** елементів у кожному - опис матриці суміжності.

Вихідні дані

У вихідний файл виведіть список ребер, упорядкований спочатку по першій вершині і парі вершин, яка описує ребро, а потім по другій.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
3 0 1 1 1 0 1 1 1 0	1 2 1 3 2 3

Розв'язання

В матриці суміжності простого неорієнтованого графа знаходимо елементи над головною діагоналлю значення яких дорівнює 1 і виводимо індекси цих елементів, що відповідають парам вершин, які описують ребра цього графа.

Лістінг програми:

```
var A:array[1..100,1..100] of  integer;
      i,j,n:integer;
begin
readln(n);
for i:=1 to n do
   for j:=1 to n do
   read(A[i,j]);
for i:=1 to n do
  for j:=1 to n do
    if (j>i) and (A[i,j]=1) then  writeln(i,'  ',j);
```

end.

ВІД СПИСКУ РЕБЕР ДО МАТРИЦІ СУМІЖНОСТІ

Простий неорієнтовний граф задано списком ребер. Виведіть його подання у вигляді матриці суміжності.

Вхідні дані

У першому рядку задано два цілих числа **n** (**1** ≤ **n** ≤ **100**) - число вершин та **m** (**1** ≤ **m** ≤ **n** · (**n** - **1**) / **2**) - число ребер. Далі у **m** рядках міститься **m** пар чисел, кожна з яких описує одне ребро графа.

Вихідні дані

Виведіть матрицю суміжності графа.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** МіВ

Вхідні дані	**Вихідні дані**
3 3 1 2 2 3 1 3	0 1 1 1 0 1 1 1 0

Розв'язання

Зчитуємо пару чисел ***i*** та ***j***. В матриці суміжності задаємо значення елементам ***A[i,j]***= ***A[j,i]***=1.

Лістінг програми:

```
var A:array [1..100,1..100] of integer;
   i,j,k,n,m:longint;
begin
readln(n,m);
for k:=1 to m do
  begin
  readln(i,j);
  A[i,j]:=1;
  A[j,i]:=1;
  end;
for i:=1 to n do
  begin
  for j:=1 to n do
  write(A[i,j],' ');
  writeln;
  end;
end.
```

СТЕПЕНІ ВЕРШИН

Простий неорієнтовний граф задано матрицею суміжності. Знайдіть степені усіх вершин графа.

Вхідні дані

У першому рядку задано число **n** (**1** ≤ **n** ≤ **100**). Потім йдуть **n** рядків по **n** елементів у кожному - опис матриці суміжності.

Вихідні дані

Виведіть **n** чисел - степені усіх вершин.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** МіВ

Вхідні дані	**Вихідні дані**
3 0 1 0 1 0 1 0 1 0	1 2 1

Розв'язання

В матриці суміжності знаходимо в кожному рядку кількість елементів, значення яких дорівнює 1.

Лістінг програми:

```
var a,i,j,k,n:integer;
begin
readln(n);
for i:=1 to n do
  begin
  k:=0;
  for j:=1 to n do
    begin
    read(a);
    if a=1 then k:=k+1;
    end;
  writeln(k);
  end;
end.
```

КІЛЬКІСТЬ РЕБЕР У ОРІЄНТОВНОМУ ГРАФІ

Орієнтовний граф задано матрицею суміжності.

Знайдіть кількість ребер у графі.

Вхідні дані

У першому рядку вхідного файлу задано число **N** (**1** ≤ **N**≤ **100**). Потім йдуть **N** рядків по **N** елементів у кожному - опис матриці суміжності.

Вихідні дані

У вихідній файл виведіть єдине число - кількість ребер у графі.

Вхідні дані	Вихідні дані
3 0 1 1 1 0 1 0 1 1	6

Розв'язання

В матриці суміжності знаходимо кількість елементів, значення яких дорівнює 1.

Лістінг програми:

```
var a,i,j,k,n:longint;
begin
readln(n);
k:=0;
for i:=1 to n do
  for j:=1 to n do
  begin
  read(a);
  if a=1 then k:=k+1;
  end;
write(k);
end.
```

ВІД МАТРИЦІ СУМІЖНОСТІ ДО СПИСКУ РЕБЕР - 2

Орієнтовний граф задано матрицею суміжності.

Виведіть його подання у вигляді списку ребер.

Вхідні дані

У першому рядку вхідного файлу задано число **N** (**1** ≤ **N**≤ **100**). Потім йдуть **N** рядків по **N** елементів у кожному - опис матриці суміжності.

Вихідні дані

У вихідний файл виведіть список ребер, впорядкований спочатку за першою вершиною у парі вершин, яка описує ребро, а потім по другій.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	Вихідні дані
3 0 1 0 0 0 1 1 1 0	1 2 2 3 3 1 3 2

Розв'язання

Знаходимо елемент матриці суміжності, значення якого дорівнює 1. Виводимо номер рядка та номер стовпчика, в якому він знаходиться.

Лістінг програми:

```
var a,i,j,n:integer;
begin
readln(n);
for i:=1 to n do
  for j:=1 to n do
  begin
  read(a);
  if  a=1 then
        begin
        writeln(i,'  ',j);
        end;
  end;
end.
```

ВІД СПИСКУ РЕБЕР ДО МАТРИЦІ СУМІЖНОСТІ - 2

Простий орієнтовний граф задано списком ребер.

Виведіть його подання у вигляді матриці суміжності.

Вхідні дані

У першому рядку вхідного файлу задано два цілих числа **N** (**1** ≤ **N**≤ **100**) - число вершин та **M** (**1**≤ **M**≤ **N**·**(N-1)/2**) - число ребер. Далі у **M** рядках містяться **M** пар чисел, кожна з яких описує одне ребро графа.

Вихідні дані

У вихідний файл виведіть матрицю суміжності графа.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
3 4 1 2 2 3 3 1 3 2	0 1 0 0 0 1 1 1 0

Розв'язання

Зчитуємо пару чисел ***i*** та ***j***. В матриці суміжності задаємо значення елементу ***A[i,j]***=1.

Лістінг програми:

```
var A:array [1..100,1..100] of integer;
   i,j,k,n,m:longint;
begin
readln(n,m);
for k:=1 to m do
  begin
  readln(i,j);
  A[i,j]:=1;
```

```
  end;
for i:=1 to n do
  begin
  for j:=1 to n do
  write(A[i,j],'  ');
  writeln;
  end;
end.
```

Задачі з використанням теорії графів. Пошук у глибину і ширину

ОБХІД У ГЛИБИНУ

Задано неорієнтовний незважений граф, у якому виділено вершину. Вам потрібно знайти кількість вершин, які лежать з нею у одній компоненті зв'язності (включаючи саму вершину).

Вхідні дані

У першому рядку містяться два цілих числа **n** та **s** (**1** ≤ **s** ≤ **n** ≤ **100**), де **n** - кількість вершин графа, а **s** - виділена вершина. У наступних **n** рядках записано по **n** чисел - матриця суміжності графа, у якій цифра "**0**" позначає відсутність ребра між вершинами, а цифра "**1**" - його наявність. Гарантується, що на головній діагоналі матриці завжди стоять нулі.

Вихідні дані

Виведіть шукану кількість вершин.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	Вихідні дані
5 1 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0	3

Ідеї розв’язання задачі

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи I-III ступенів фізико-математичного профілю № 12 м. Чернігова

Розв'язання

Нехай масив ***A*** містить матрицю суміжності неорієнтованого незваженого графа, масив ***Ch*** – стек для роботи з вершинами графа, масив ***B*** – інформацію про відвідані вершини графа.

Реалізуємо алгоритм обходу графу з вершини **s** в глибину. Розв'язком буде кількість відвіданих вершин.

Лістінг програми:

```
var A:array[1..100,1..100] of byte;
    Ch:array[1..100] of byte;
    B:array [1..100] of boolean;
    i,j,tail,s,n,k:byte;
    flag:boolean;
begin
readln(n,s);
for i:=1 to n do
  for j:=1 to n do
  read(A[i,j]);
tail:=1;
Ch[tail]:=s;
B[s]:=true;
while tail>0 do
  begin
  flag:=false;
  for j:=1 to n do
    if (A[Ch[tail],j]=1) and (B[j]=false) then
                                  begin
                                  tail:=tail+1;
                                  Ch[tail]:=j;
                                  B[j]:=true;
                                  flag:=true;
                                  break;
                                  end;
  if flag=false then tail:=tail-1;
  end;
k:=0;
for i:=1 to n do
  if B[i]=true then k:=k+1;
write(k);
end.
```

Ідеї розв'язання задачі

Бондаренко С. М., учителя математики та інформатики Прилуцької загальноосвітньої школи І-ІІІ ступенів № 7 Прилуцької міської ради

Розв'язання

```
var head,n,m,teil,k,i,j,pos,st,sc,v:longint;visit:array [1..100] of boolean;
    a:array [1..100,1..100]of byte;
    queue:array [1..100]of longint;
```

```
begin
{Читаємо граф з таблиці}
 read(n,m);
 for i:=1 to n do
  for j:=1 to n do
   read(a[i,j]);
{Пошуком у глибину складаємо  таблицю відвіданих вершин, починаючи з даної вершини}
 st:=0;
 sc:=0;
 head:=1;
 teil:=2;
 fillchar(visit,sizeof(visit),0);
 st:=m;
 visit[st]:=true;
 queue[head]:=st;
 while head<teil do
 begin
  k:=queue[head];
  inc(head);
  for i:=1 to n do
   if (a[k,i]<>0)and(not visit[i])
   then
    begin
     queue[teil]:=i;
     inc(teil);
     visit[i]:=true;
    end;
  end;
{Підраховуємо кількість відвіданих вершин}
 for i:=1 to n do
  if visit[i] then inc(v);
 write(v);

end.
```

ПЛОЩА КІМНАТИ

Обчислити площу кімнати у квадратному лабіринті.

Вхідні дані

У першому рядку знаходиться число **n** - розмір лабіринту (**3** ≤ **n** ≤ **10**). У наступних **n** рядках, кожен з яких містить по **n** символів, задано лабіринт (символ "." позначає порожню клітинку, а "*" - стінку). Останній рядок містить два числа - номер рядка та стовбця клітинки, яка знаходиться у кімнаті, площу якої необхідно обчислити. Гарантується, що ця клітинка порожня і що лабіринт

оточено стінками з усіх сторін. Рядки та стовбці лабіринту нумеруються з одиниці.

Вихідні дані

Вивести кількість порожніх клітинок у даній кімнаті.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** МіВ

Вхідні дані	**Вихідні дані**
5 ***** **..* *.*.* *..** ***** 2 4	3

Ідеї розв'язання задачі

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи І-ІІІ ступенів фізико-математичного профілю № 12 м. Чернігова

Розв'язання

Нехай масив ***A*** містить інформацію про клітинки лабіринту (0 відповідає символу «*», 1 – символу «.»), масив ***cher*** – черга для збереження координат клітинок лабіринту, масив ***B*** – інформацію про відвідані клітинки лабіринту.

Реалізуємо обхід по лабіринту з заданої клітинки в ширину. Площею кімнати буде кількість відвіданих клітинок.

Лістінг програми:

```
var A:array[1..10,1..10] of byte;
    c:char;
    i,j,n,t1,t2,head,tail:integer;
    B:array[0..11,0..11] of boolean;
    cher:array[1..2,1..100] of integer;
begin
readln(n);
for i:=1 to n do
  begin
  for j:=1 to n do
    begin
    read(c);
    if c='*' then A[i,j]:=0 else A[i,j]:=1;
    end;
  readln;
  end;
read(t1,t2);
head:=1;
```

```
tail:=2;
cher[1,head]:=t1;
cher[2,head]:=t2;
B[t1,t2]:=true;
while head<tail do
 begin
  if (A[cher[1,head]-1,cher[2,head]]=1)
  and (B[cher[1,head]-1,cher[2,head]]=false) then begin
                                cher[1,tail]:=cher[1,head]-1;
                                cher[2,tail]:=cher[2,head];
                                B[cher[1,tail],cher[2,tail]]:=true;
                                tail:=tail+1;
                                end;
  if (A[cher[1,head]+1,cher[2,head]]=1)
  and (B[cher[1,head]+1,cher[2,head]]=false) then begin
                                cher[1,tail]:=cher[1,head]+1;
                                cher[2,tail]:=cher[2,head];
                                B[cher[1,tail],cher[2,tail]]:=true;
                                tail:=tail+1;
                                end;
  if (A[cher[1,head],cher[2,head]-1]=1)
    and (B[cher[1,head],cher[2,head]-1]=false) then begin
                                cher[1,tail]:=cher[1,head];
                                cher[2,tail]:=cher[2,head]-1;
                                B[cher[1,tail],cher[2,tail]]:=true;
                                tail:=tail+1;
                                end;
  if (A[cher[1,head],cher[2,head]+1]=1)
    and (B[cher[1,head],cher[2,head]+1]=false) then begin
                                cher[1,tail]:=cher[1,head];
                                cher[2,tail]:=cher[2,head]+1;
                                B[cher[1,tail],cher[2,tail]]:=true;
                                tail:=tail+1;
                                end;
 head:=head+1;
 end;
write(head-1);
end.
```

Ідеї розв’язання задачі

Бондаренко С. М., учителя математики та інформатики Прилуцької загальноосвітньої школи І-ІІІ ступенів № 7 Прилуцької міської ради

Розв'язання

Можна розглянути клітини даної таблиці лабіринту як вершини графа. Тоді задача зводиться до попередньої (обхід у ширину)

```
var head,n,m,teil,k,i,j,l,pos,st,sc,v:longint;visit:array [1..100] of boolean;
    a:array [1..100,1..100]of byte;
    b:array[1..10,1..10]of char;
    queue:array [1..100]of longint;
begin
{Читаємо лабіринт}
 readln(n);
 for i:=1 to n do
  begin
   for j:=1 to n do
    read(b[i,j]);
   readln;
  end;
 read(k,l);
 m:=(k-1)*n+l;
 fillchar(a,sizeof(a),0);
```

{За даним лабіринтом будуємо таблицю зв'язності нового графа. Для кожної ячейки таблиці розглядаємо лише можливість руху вправо та вниз}

	1	2	…	n
1	1	2		n
2	1*n+1	1*n+2		2n
…				
n	(n-1)*n+1	(n-1)*n+2		n*n

```
 for i:=2 to n-1 do
  for j:=2 to n-1 do
   if b[i,j]='.' then
    begin
     if b[i,j+1]='.' then
      begin a[(i-1)*n+j,(i-1)*n+j+1]:=1;a[(i-1)*n+j+1,(i-1)*n+j]:=1;end;
     if b[i+1,j]='.' then
      begin a[(i-1)*n+j,i*n+j]:=1;a[i*n+j,(i-1)*n+j]:=1;end;
    end;
 n:=n*n;
 st:=0;
 sc:=0;
 head:=1;
 teil:=2;
 fillchar(visit,sizeof(visit),0);
 st:=m;
 visit[st]:=true;
 queue[head]:=st;
 while head<teil do
 begin
  k:=queue[head];
  inc(head);
  for i:=1 to n do
```

```
      if (a[k,i]<>0)and(not visit[i])
      then
        begin
          queue[teil]:=i;
          inc(teil);
          visit[i]:=true;
        end;
    end;
  for i:=1 to n do
    if visit[i] then inc(v);
  write(v);
end.
```

ОБХІД В ШИРИНУ

Задано неорієновний граф. Знайти відстань від однієї заданої вершини до іншої.

Вхідні дані

У першому рядку міститься три натуральних числа **n**, **s** та **f** (**1 ≤ s, f ≤ n ≤ 100**) - кількість вершин у графі і номери початкової та кінцевої вершин відповідно. Далі у **n** рядках задано матрицю суміжності графа. Якщо значення у **j**-му елементі **i**-го рядка дорівнює **1**, то у графі є направлене ребро з вершини **i** до вершини **j**.

Вихідні дані

Вивести мінімальну відстань від початкової вершини до кінцевої. Якщо шляху не існує, виведіть **0**.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	Вихідні дані
4 4 3 0 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0	2

Ідеї розв'язання задачі

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи I-III ступенів фізико-математичного профілю № 12 м. Чернігова

Розв'язання

Нехай масив ***A*** містить матрицю суміжності неорієнтованого графа, масив ***cher*** – черга для роботи з вершинами графа, масив ***B*** – інформацію про відвідані вершини графа.

Реалізуємо алгоритм обходу графу з вершини **s** в вершину ***f*** в ширину. В масиві ***dist*** будемо зберігати номер кроку пошуку в ширину для кожної вершини, яка була знайдена.

Якщо вершина знайдена, то відстанню буде значення ***dist[tail]***, де ***tail*** – хвіст черги.

Лістінг програми:

```
var A:array[1..100,1..100] of byte;
   i,j,n,s,f,head,tail:integer;
   B:array[1..100] of boolean;
   cher,dist:array[1..100] of integer;
   flag:boolean;
begin
read(n,s,f);
for i:=1 to n do
  for j:=1 to n do
  read(A[i,j]);
head:=1;
tail:=2;
cher[head]:=s;
B[s]:=true;
flag:=false;
dist[head]:=0;
while (head<tail) and (flag=false) do
  begin
  for j:=1 to n do
    begin
    if (a[cher[head],j]=1) and (B[j]=false) then
                                  begin
                                  cher[tail]:=j;
                                  B[j]:=true;
                                  dist[tail]:=dist[head]+1;
                                  if j=f then
                                        begin
                                        flag:=true;
                                        break;
                                        end;
                                  tail:=tail+1;
                                  end;
    end;
  head:=head+1;
  end;
if  flag=false then write(0) else write(dist[tail]);
end.
```

Ідеї розв’язання задачі

Бондаренко С. М., учителя математики та інформатики Прилуцької загальноосвітньої школи І-ІІІ ступенів № 7 Прилуцької міської ради

Розв'язання

Використано алгоритм пошуку в глибину з добавлянням лічильнику мінімального шляху.

```
 var head,n,m,teil,k,i,j,pos,st,sc,v,con:longint;
    a:array [1..100,1..100]of longint;
    queue:array [1..100]of longint;
    hod:array [1..100] of longint;
    visit:array [1..100] of boolean;
begin
 read(n,st,con);
 for i:=1 to n do
  for j:=1 to n do
   read(a[i,j]);
 head:=1;
 teil:=2;
 queue[head]:=st;
 visit[st]:=true;
 while head<teil do
 begin
  k:=queue[head];
  inc(head);
  visit[k]:=true;
  for i:=1 to n do
  begin

   if (a[k,i]<>0)and(not visit[i])
   then
    begin
      if hod[k]+1<hod[i] then hod[i]:=hod[k]+1;
     if hod[i]=0 then hod[i]:=hod[k]+1;
      visit[i]:=true;
      queue[teil]:=i;
      inc(teil);
    end;
  end;
 end;
  write(hod[con]);
end.
```

Ідеї розв'язання задач

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи І-ІІІ ступенів фізико-математичного профілю № 12 м. Чернігова

ВИДАЛЕННЯ КЛІТИНОК

З прямокутного аркуша у клітинку (**m** рядків, **n** стовбчиків) видалили деякі клітинки. На скільки шматків розпадеться частина аркуша, що залишилась? Дві клітинки не розпадаються, якщо вони мають спільну сторону.

Вхідні дані

У першому рядку знаходяться числа **m** і **n** (**1** ≤ **m**, **n** ≤ **100**). У наступних **m** рядках міститься по **n** символів. Якщо клітинку не було вирізано, цьому відповідає знак #, якщо вирізано - крапка.

Вихідні дані

Вивести кількість частинок, на яку розпадеться аркуш.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** МіВ

Вхідні дані	**Вихідні дані**
4 8 #.##.#.# .....## #.###.## ##.##.##	6

Розв'язання

Для кожної клітинки аркуша, яка не є відвідуваною та позначена знаком #, запустимо алгоритм пошуку в ширину клітинок зі знаком #. Кількість таких виконаних алгоритмів є відповіддю задачі.

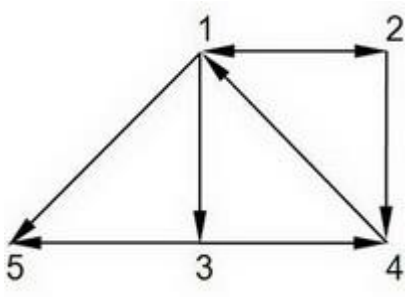
Лістінг програми:

```
var A:array[0..101,0..101] of byte;
    c:char;
    i,j,n,m,s,head,tail:integer;
    B:array[0..101,0..101] of boolean;
    cher:array[1..2,1..10000] of integer;
begin
readln(n,m);
for i:=1 to n do
  begin
  for j:=1 to m do
    begin
    read(c);
    if c='.' then A[i,j]:=0 else A[i,j]:=1;
    end;
  readln;
  end;
```

```
for i:=1 to n do
  for j:=1 to m do
    begin
    if (A[i,j]=1) and (B[i,j]=false) then begin
    s:=s+1;
fillchar(cher,sizeof(cher),0);
head:=1;
tail:=2;
cher[1,head]:=i;
cher[2,head]:=j;
B[i,j]:=true;
while head<tail do
  begin
    if (A[cher[1,head]-1,cher[2,head]]=1)
      and (B[cher[1,head]-1,cher[2,head]]=false) then begin
                                  cher[1,tail]:=cher[1,head]-1;
                                  cher[2,tail]:=cher[2,head];
                                  B[cher[1,tail],cher[2,tail]]:=true;
                                  tail:=tail+1;
                                  end;
    if (A[cher[1,head]+1,cher[2,head]]=1)
      and (B[cher[1,head]+1,cher[2,head]]=false) then begin
                                  cher[1,tail]:=cher[1,head]+1;
                                  cher[2,tail]:=cher[2,head];
                                  B[cher[1,tail],cher[2,tail]]:=true;
                                  tail:=tail+1;
                                  end;
    if (A[cher[1,head],cher[2,head]-1]=1)
      and (B[cher[1,head],cher[2,head]-1]=false) then begin
                                  cher[1,tail]:=cher[1,head];
                                  cher[2,tail]:=cher[2,head]-1;
                                  B[cher[1,tail],cher[2,tail]]:=true;
                                  tail:=tail+1;
                                  end;
    if (A[cher[1,head],cher[2,head]+1]=1)
      and (B[cher[1,head],cher[2,head]+1]=false) then begin
                                  cher[1,tail]:=cher[1,head];
                                  cher[2,tail]:=cher[2,head]+1;
                                  B[cher[1,tail],cher[2,tail]]:=true;
                                  tail:=tail+1;
                                  end;
  head:=head+1;
  end;
  end;
```

```
  end;
write(s);
end.
```

МАРШРУТИ В ГОРАХ



Гірський туристичний комплекс складається з n турбаз, з'єднаних між собою k гірськими переходами (інші маршрути в горах небезпечні). Кожен перехід між двома базами займає 1 день. Туристична група знаходиться на базі a і збирається потрапити на базу b не більш ніж за d днів. Скільки існує різних таких маршрутів (без циклів) між a і b?

Вхідні дані

В першому рядку через проміжок записані числа n, k, a, b, **d (**n ≤ **50, d** ≤ **10)**. Кожен з наступних k рядків містить пару чисел, яка описує можливий гірський перехід. Усі числові значення натуральні.

Вихідні дані

Вивести одне число – кількість маршрутів.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** МіВ

Вхідні дані	**Вихідні дані**
5 8 2 5 3 1 2 1 3 1 5 2 1 2 4 3 4 3 5 4 1	3

Розв'язання

Нехай масив ***T*** містить матрицю суміжності орієнтованого незваженого графа, масив ***cher*** – стек для роботи з вершинами графа, масив ***S*** – інформацію про відвідані вершини графа.

Реалізуємо модифікований алгоритм обходу графу в глибину з вершини ***a*** в вершину ***b***:

1. При досягненні вершини ***b***, повертаємося в стеку до попередньої вершини і продовжуємо пошук з вершини, що має номер більший на 1(наприклад, якщо повернулися до вершини з номером 3, то продовжимо пошук з вершини 4).

2. Глибина обходу графа повинна бути ***tail-1<d***, де ***tail*** – індекс елемента стеку, що відповідає поточній вершині, ***d*** – кількість днів.

Лістінг програми:

```
var T:array[1..51,1..51] of byte;
   i,j,n,k,a,b,d,tail,x1,x2,x,km:integer;
   cher:array[1..51] of integer;
   S:array[1..51] of boolean;
   flag:boolean;
begin
assign(input,'input.txt');
assign(output,'output.txt');
reset(input);
rewrite(output);
readln(n,k,a,b,d);
for i:=1 to k do
  begin
  readln(x1,x2);
  T[x1,x2]:=1;
  end;
tail:=1;
cher[tail]:=a;
S[a]:=true;
x:=1;
km:=0;
while tail>0 do
  begin
  flag:=false;
  if tail-1<d then begin
                for j:=x to n do
                  if (T[cher[tail],j]=1) and (S[j]=false) then
                                          begin
                                          tail:=tail+1;
                                          S[j]:=true;
                                          cher[tail]:=j;
                                          flag:=true;
                                          x:=1;
                                          break;
                                          end;

                if (cher[tail]=b) then
                                begin
                                km:=km+1;
                                S[b]:=false;
                                x:=cher[tail]+1 ;
```

```
                    tail:=tail-1;
                    end;
              if (flag=false)  then
                    begin
                    S[cher[tail]]:=false;
                    x:=cher[tail]+1 ;
                    tail:=tail-1;
                    end;
              end
            else begin
              S[cher[tail]]:=false;
              x:=cher[tail]+1 ;
              tail:=tail-1;
              end;
  end;
write(km);
close(input);
close(output);
end.
```

ЗВ’ЯЗНІСТЬ

Перевірити, чи є заданий неорієнтований граф зв'язним, тобто, що з довільної вершини можна по ребрам цього графа потрапити у довільну іншу.

Вхідні дані

У першому рядку задано кількість вершин **n** та кількість ребер **m** у графі відповідно (**1** ≤ **n** ≤ **100**, **1** ≤ **m** ≤ **10000**). Наступні **m** рядків містять по два числа u_i і v_i (**1** ≤ u_i, v_i ≤ **n**); кожен такий рядок означає, що у графі існує ребро між вершинами u_i і v_i.

Вихідні дані

Виведіть "**YES**", якщо граф є зв'язним, і "**NO**" у протилежному випадку.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
3 2 1 2 3 2	YES
3 1 1 3	NO

Розв'язання

Нехай масив ***A*** містить матрицю суміжності неорієнтованого графа, масив ***Ch*** – стек для роботи з вершинами графа, масив ***B*** – інформацію про відвідані вершини графа.

Реалізуємо алгоритм обходу графу з першої вершини в глибину. Якщо всі вершини було відвідані, то граф зв'язний, інакше – ні.

Лістінг програми:

```
var A:array[1..100,1..100] of byte;
    Ch:array[1..100] of byte;
    B:array [1..100] of boolean;
    i,j,tail,s,n,k:integer;
    m:integer;
    flag:boolean;
begin
readln(n,m);
for s:=1 to m do
  begin
  read(i,j);
  A[i,j]:=1;
  A[j,i]:=1;
  end;
tail:=1;
Ch[tail]:=1;
B[1]:=true;
while tail>0 do
  begin
  flag:=false;
  for j:=1 to n do
    if (A[Ch[tail],j]=1) and (B[j]=false) then
                                  begin
                                  tail:=tail+1;
                                  Ch[tail]:=j;
                                  B[j]:=true;
                                  flag:=true;
                                  break;
                                  end;
  if flag=false then tail:=tail-1;
  end;
k:=0;
for i:=1 to n do
  if B[i]=true then k:=k+1;
if k=n then write('YES') else write('NO');
```

ЛІНІЇ

У таблиці з **n** рядків та **n** стовпчиків деякі клітинки зайняті кульками, інші вільні. Обрано кульку, яку потрібно перемістити, і місце, куди її слід перемістити. Обрана кулька за один крок переміщується у сусідню по горизонталі або вертикалі вільну клітинку. Потрібно вияснити, чи можливо

перемістити кульку з початкової клітинки у задану, і якщо можливо, то знайти шлях з найменшою кількістю кроків.

Вхідні дані

У першому рядку знаходиться число **n** (**2 ≤ n ≤ 40**), в кожному з наступних **n** рядків – по **n** символів. Символом точки позначено вільну клітинку, латинською великою **O** - кульку, **@** - початкове положення кульки, яка повинна переміститися, латинською великою **X** - кінцеве положення кульки.

Вихідні дані

У першому рядку виводиться **Y**, якщо переміщення можливе, або **N**, якщо ні. Якщо переміщення можливе далі слід вивести **n** рядків по **n** символів - як і на вході, але **X**, а також всі точки на шляху замінюються плюсами.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** МіВ

Вхідні дані	**Вихідні дані**
5 ....X .OOOO ..... OOOO. @....	Y +++++ +OOOO +++++ OOOO+ @++++
5 ..X.. ..... OOOOO ....@..	N
5 ...X. ..... O.OOO ....@	Y ..++. .++.. O+OOO .++++ ....@

Розв'язання

Нехай масив ***A*** містить інформацію про клітинки (0 відповідає символу «O», 1 – символу «.»), масив ***cher*** – черга для збереження координат клітинок , масив ***B*** – інформацію про відвідані клітинки.

Реалізуємо обхід в ширину з клітинки позначеної символом «**@**» до клітинки позначеної символом «***X»***. Масив ***st*** буде зберігати координати клітинки, з якої дана клітинка була відвідана.

Використовуючи інформацію масиву ***st***, змінюємо в початковому масиві ***C*** значення в клітинках на «+».

Лістінг програми:

```
var A:array[0..41,0..41] of byte;
    c:array[0..41,0..41] of char;
    i,j,n,t1,t2,s1,s2,head,tail:integer;
    B:array[0..41,0..41] of boolean;
    cher,st:array[1..2,1..10000] of integer;
    flag:boolean;
begin
readln(n);
for i:=1 to n do
  begin
  for j:=1 to n do
    begin
    read(c[i,j]);
    if c[i,j]='.' then A[i,j]:=1;
    if c[i,j]='O' then A[i,j]:=0;
    if c[i,j]='X' then begin
                A[i,j]:=0;
                t1:=i;
                t2:=j;
                end;
    if c[i,j]='@' then begin
                A[i,j]:=1;
                s1:=i;
                s2:=j;
                end;
    end;
  readln;
  end;
head:=1;
tail:=2;
cher[1,head]:=t1;
cher[2,head]:=t2;
B[t1,t2]:=true;
st[1,head]:=0;
st[2,head]:=0;
flag:=false;
while head<tail do
  begin
    if (A[cher[1,head]-1,cher[2,head]]=1)
      and (B[cher[1,head]-1,cher[2,head]]=false) then begin
                                cher[1,tail]:=cher[1,head]-1;
```

```
                                cher[2,tail]:=cher[2,head];
                                B[cher[1,tail],cher[2,tail]]:=true;
                                st[1,tail]:=cher[1,head];
                                st[2,tail]:=cher[2,head];
                                if (cher[1,tail]=s1) and (cher[2,tail]=s2)  then
                                   begin flag:=true; break;end;
                                tail:=tail+1;
                                end;
    if (A[cher[1,head]+1,cher[2,head]]=1)
       and (B[cher[1,head]+1,cher[2,head]]=false) then begin
                                cher[1,tail]:=cher[1,head]+1;
                                cher[2,tail]:=cher[2,head];
                                B[cher[1,tail],cher[2,tail]]:=true;
                                st[1,tail]:=cher[1,head];
                                st[2,tail]:=cher[2,head];
                                if (cher[1,tail]=s1) and (cher[2,tail]=s2) then
                                   begin flag:=true; break;end;
                                tail:=tail+1;
                                end;
    if (A[cher[1,head],cher[2,head]-1]=1)
        and (B[cher[1,head],cher[2,head]-1]=false) then begin
                                cher[1,tail]:=cher[1,head];
                                cher[2,tail]:=cher[2,head]-1;
                                B[cher[1,tail],cher[2,tail]]:=true;
                                st[1,tail]:=cher[1,head];
                                st[2,tail]:=cher[2,head];
                                if (cher[1,tail]=s1) and (cher[2,tail]=s2) then
                                   begin flag:=true; break;end;
                                tail:=tail+1;
                                end;
    if (A[cher[1,head],cher[2,head]+1]=1)
         and (B[cher[1,head],cher[2,head]+1]=false) then begin
                                cher[1,tail]:=cher[1,head];
                                cher[2,tail]:=cher[2,head]+1;
                                B[cher[1,tail],cher[2,tail]]:=true;
                                st[1,tail]:=cher[1,head];
                                st[2,tail]:=cher[2,head];
                                if (cher[1,tail]=s1) and (cher[2,tail]=s2) then
                                   begin flag:=true; break;end;
                                tail:=tail+1;
                                end;
  head:=head+1;
  end;
if flag=false then begin
```

```
                write('N');
                exit;
                end;
    writeln('Y');
    t1:=st[1,tail];
    t2:=st[2,tail];
    c[st[1,tail],st[2,tail]]:='+';
    for j:=tail downto 1 do
      if (cher[1,j]=t1) and (cher[2,j]=t2) then begin
                                    c[st[1,j],st[2,j]]:='+';
                                    t1:=st[1,j];
                                    t2:=st[2,j];
                                    end;
    for i:=1 to n do
      begin
      for j:=1 to n do
        write(c[i,j]);
      writeln;
      end;
    end.
```

НАЙКОРОТШИЙ ШЛЯХ

Задано неорієнтовний граф. Знайдіть найкоротший шлях від вершини **a** до вершини **b**.

Вхідні дані

У першому рядку знаходиться два цілих числа **n** та **m** (**1** ≤ **n** ≤ **50000**, **1** ≤ **m** ≤ **100000**) - кількість вершин та ребер відповідно. У другому рядку йдуть цілі числа **a** та **b** - стартова та кінцева вершина відповідно. Далі йде **m** рядків, які описують ребра.

Вихідні дані

Якщо шляху між **a** та **b** немає, то виведіть **- 1**. Інакше виведіть у першому рядку довжину **l** найкоротшого шляху між цими двома вершинами у ребрах, а у другому рядку виведіть **l** + **1** число - вершини цього шляху.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** МіВ

Вхідні дані	Вихідні дані
4 5 1 4 1 3 3 2 2 4 2 1 2 3	2 1 2 4

4 4 2 3 2 1 2 4 4 3 1 3	2 2 1 3

Розв'язання

Особливістю задачі є те, що інформацію про ребра графа потрібно зберігати не в матриці суміжності.

Нехай масив ***T*** – містить інформацію про ребра між вершинами ***i*** та ***j*** графа, де номер рядка відповідає ***i***-тій вершині, ***T[i,0]*** – кількість ребер з ***i*** вершиною, ***T[i,j]*** – номер вершини, яка має ребро з вершиною ***i***.

Реалізуємо алгоритм обходу графу з вершини **a** в вершину ***b*** в ширину. В масиві ***dist*** будемо зберігати номер кроку пошуку в ширину для кожної вершини, яка була знайдена.

Якщо вершина знайдена, то відстанню буде значення ***dist[tail]***, де ***tail*** – хвіст черги.

Лістінг програми:

```
var T:array[0..50000,0..500] of longint;
    i,j,n,m,a,b,k1,k2,head,tail:longint;
    cher, mar, dist, marshrut:array[1..50000] of longint;
    S:array[1..50000] of boolean;
    flag:boolean;
begin
readln(n,m);
readln(a,b);
for i:=1 to m do
  begin
  readln(k1,k2);
    T[k1,0]:=T[k1,0]+1;
    T[k1,T[k1,0]]:=k2;
    T[k2,0]:=T[k2,0]+1;
    T[k2,T[k2,0]]:=k1;
  end;
head:=1;
cher[head]:=a;
mar[head]:=0;
S[a]:=true;
flag:=false;
tail:=2;
dist[head]:=0;
while (head<tail) and (flag=false) do
  begin
  for j:=1 to T[cher[head],0] do
```

```
    begin
    if S[T[cher[head],j]]=false then
                        begin
                        cher[tail]:=T[cher[head],j];
                        S[T[cher[head],j]]:=true;
                        mar[tail]:=cher[head];
                        dist[tail]:=dist[head]+1;
                        if cher[tail]=b then begin
                                        flag:=true;
                                        break;
                                        end;
                        tail:=tail+1;
                        end;
    end;
  head:=head+1;
  end;
if flag=false then writeln(-1)
            else begin
            writeln(dist[tail]);
            j:=1;
            marshrut[j]:=b;
            for i:=tail downto 1 do
              if cher[i]=marshrut[j] then begin
                                    j:=j+1;
                                    marshrut[j]:=mar[i];
                                    end;
            for i:=j-1 downto 1 do
              write(marshrut[i],' ');
            end;
end.
```

Задачі на визначення найкоротшого шляху в графі. Алгоритм Дейкстри. Алгоритм Флойда.

Ідеї розв'язання задач

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи I-III ступенів фізико-математичного профілю № 12 м. Чернігова

ДЕЙКСТРА

Задано орієнтовний зважений граф. Знайдіть найкоротшу відстань від однієї заданої вершини до іншої.

Вхідні дані

У першому рядку міститься три числа **n**, **s** та **f** (**1** ≤ **n** ≤ **2000**; **1** ≤ **s**, **f** ≤ **n**), де **n** - кількість вершин графа, **s** - початкова вершина, а **f** - кінцева. У наступних **n** рядках по **n** чисел - матриця суміжності графа, де **-1** означає відсутність ребра між вершинами, а довільне невід'ємне число - присутність ребра заданої ваги. На головній діагонали матриці завжди записані нулі.

Вихідні дані

Вивести шукану відстань або **- 1**, якщо шляху не існує.

Ліміт часу **0.5** секунд

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
3 1 2 0 -1 2 3 0 -1 -1 4 0	6

Розв'язання

Застосувати алгоритм Дейкстри.

Лістінг програми:

```
var D:array[1..2000,1..2000] of integer;
    dist:array[1..2000] of int64;
    flag:array[1..2000] of boolean;
    i,j,k,n,s,f,min:longint;
begin
readln(n,s,f);
for i:=1 to n do
  for j:=1 to n do
  read(D[i,j]);
flag[s]:=true;
for i:=1 to n do
  begin
   if D[s,i]<=0 then dist[i]:=65535 else dist[i]:=D[s,i];
  end;
dist[s]:=0;
j:=1;
while j<n do
  begin
  min:=65535;
  for i:=1 to n do
    if (flag[i]=false) and (dist[i]<min) and (dist[i]>0) then
                                          begin
                                          min:=dist[i];
                                          k:=i;
                                          end;
  for i:=1 to n do
```

```
  if (D[k,i]>0) and (dist[i]>dist[k]+D[k,i]) then dist[i]:=dist[k]+D[k,i];
 flag[k]:=true;
 j:=j+1;
 end;
if dist[f]=65535 then write(-1) else write(dist[f]);
end.
```

ЗАПРАВКИ

У країні **n** міст, деякі з яких з'єднані між собою дорогами. Для того, щоб проїхати по одній дорозі потрібно один бак бензину. У кожному місті бак бензину має різну вартість. Вам потрібно дістатись з першого міста у **n**-те, витративши якомога меншу кількість грошей.

Вхідні дані

Спочатку йде кількість міст **n** (**1** ≤ **n** ≤ **100**), потім йде **n** чисел, **i**-те з яких задає вартість бензину у **i**-ому місті (всі числа цілі з діапазону від **0** до **100**). Потім йде кількість доріг **m** в країні, далі йде опис самих доріг. Кожна дорога задається двома числами - номерами міст, які вона з'єднує. Всі дороги двосторонні (тобто по ним можна їздити як в одну, так і в іншу сторону); між двома містами завжди існує не більше однієї дороги; не існує доріг, які ведуть з міста в себе.

Вихідні дані

Виведіть одно число - сумарну вартість маршруту або **- 1**, якщо дістатись неможливо.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
4 1 10 2 15 4 1 2 1 3 4 2 4 3	**3**
4 1 10 2 15 0	**-1**

Пояснення: У першому прикладі оптимальний розвязок - з 1-го міста поїхати в 3-тє, а потім у 4-те. Паливо прийдеться купувати у 1-му і 3-му містах

Розв'язання

Застосувати алгоритм Дейкстри.

Лістінг програми:

```
var i,j,n,k,m,min:longint;
     D:array[1..100,1..100] of integer;
   st, dist:array[1..100] of longint;
   B:array[1..100] of boolean;
```

```
begin
  readln(n);
  for i:=1 to n do
    read(st[i]);
  readln(m);
  for k:=1 to m do
    begin
    read(i,j);
    D[i,j]:=st[i];
    D[j,i]:=st[j];
    end;
  B[1]:=true;
  for i:=1 to n do
    begin
    if D[1,i]=0 then dist[i]:=65535
              else dist[i]:=D[1,i];
    end;
  dist[1]:=0;
  j:=1;
  while j<n do
    begin
    min:=65535;
    for i:=1 to n do
      if (B[i]=false) and (dist[i]<min) and (dist[i]>0) then
                                          begin
                                          min:=dist[i];
                                          k:=i;
                                          end;
    for i:=1 to n do
      if (D[k,i]>0) and (dist[i]>dist[k]+D[k,i]) then dist[i]:=dist[k]+D[k,i];
    B[k]:=true;
    j:=j+1;
    end;

    if dist[n]<65535 then write(dist[n]) else write(-1);
end.
```

ФЛОЙД

Задано орієнтований зважений граф. Знайти пару вершин, найкоротша відстань від однієї з яких до іншої максимальна серед усіх пар вершин.

Вхідні дані

У першому рядку міститься кількість вершин графа **n** (**1** ≤ **n** ≤ **100**). У наступних **n** рядках задано по **n** чисел, що описують матрицю суміжності графа. В ній **- 1** означає відсутність ребра між вершинами, а довільне невід'ємне

число - наявність ребра заданої ваги. На головній діагоналі матриці завжди розташовані нулі.

Вихідні дані

Вивести шукану максимальну найкоротшу відстань.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** МіВ

Вхідні дані	**Вихідні дані**
4 0 5 9 -1 -1 0 2 8 -1 -1 0 7 4 -1 -1 0	**16**

Розв'язання

Застосувати алгоритм Флойда.

Лістінг програми:

```
procedure start;
begin
assign(input,'input.txt');
assign(output,'output.txt');
reset(input);
rewrite(output);
end;
procedure edd;
begin
close(input);
close(output);
end;
const Q=100000000;
var F:array[1..100,1..100] of longint;
i,j,k,z,n,m,a,b,max,s,f1:longint;
visitor:array[1..100] of boolean;
dist:array[1..100] of longint;

procedure chk(n:longint);
var i,j:longint;
begin
for i:=1 to n do
  begin
  for j:=1 to n do
     write(F[i,j],' ');
  writeln;
  end;
writeln;
```

```
end;

begin
start;
read(n);
for i:=1 to n do
  for j:=1 to n do
   begin
   read(a);
   if a>=0 then F[i,j]:=a
   else F[i,j]:=q;
   end;

for k:=1 to n do
   for i:=1 to n do
     for j:=1 to n do
        If (F[i,j]>0) and (F[i,j]>F[i,k]+F[k,j]) then
          F[i,j]:=F[i,k]+F[k,j];
max:=-1;
for i:=1 to n do
 for j:=1 to n do
   if (F[i,j]>max) and (F[i,j]<>q) then max:=F[i,j];
   write(max);
edd;
end.
```

ЧИ Є ЦИКЛ?

Задано орієнтовний граф. Визначити, чи містить він цикл.

Вхідні дані

Перший рядок містить кількість вершин **n** (**n** ≤ **50**). Далі у **n** рядках йде по **n** чисел, кожне з яких дорівнює "**0**" або "**1**". **j**-те число в **i**-му рядку дорівнює "**1**" тоді і лише тоді, коли існує ребро, яке йде з **i**-ї вершини у **j**-ту. Гарантується, що на діагоналі матриці будуть стояти нулі.

Вихідні дані

Виведіть "**0**", якщо у заданому графі циклу немає, і "**1**", якщо він є.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
3 0 1 0 0 0 1 0 0 0	**0**

Розв'язання

Заповнимо матрицю суміжності орієнтованого графа **A** таким чином: якщо A[i,j]=1, то A[i,j] надамо значення -10, якщо A[i,j]=0, то 1000000.

Застосуємо алгоритм Флойда.

Перевіряємо елементи A[i,i], якщо серед них є менші нуля, то в графі є цикл.

Лістінг програми:

```
var i,j,n,k:longint;
    A:array[1..50,1..50] of int64;
begin
readln(n);
for i:=1 to n do
  for j:=1 to n do
    begin
    read(k);
    if k=1 then A[i,j]:=-10 else A[i,j]:=1000000;
    end;
for k := 1 to n do
     for i := 1 to n do
       for j := 1 to n do
         if a[i,j] > a[i,k] + a[k,j] then
           a[i,j] := a[i,k] + a[k,j];
for i:=1 to n do
  if A[i,i]<0  then begin write(1); exit; end; write(0);
end.
```

Задачі на побудову остовного дерева мінімальної довжини. Алгоритми Прима і Краскала. Потоки в мережах. Алгоритм Форда-Фалкерсона побудови максимального потоку в мережі

Ідеї розв'язання задач

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи I-III ступенів фізико-математичного профілю № 12 м. Чернігова

ДЕРЕВО?

Неорієнтовний граф без петель та кратних ребер задано матрицею суміжності. Визначити, чи є цей граф деревом.

Вхідні дані

Перший рядок містить кількість вершин графа **n** (**1** ≤ **n** ≤ **100**). Далі записана матриця суміжності розміром **n** × **n**, у якій 1 позначає наявність ребра, **0** - його відсутність. Матриця симетрична відносно головної діагоналі.

Вихідні дані

Виведіть повідомлення **YES**, якщо граф є деревом, і **NO** у протилежному випадку.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
3 0 1 0 1 0 1 0 1 0	YES

Розв'язання

Якщо кількість ребер не дорівнює **n-1**(де **n** – кількість вершин графа), то граф не є деревом.

Пошуком у глибину визначити, якщо граф не має ізольованих вершин, то він є деревом.

Лістінг програми:

```
var i,j,n,k,top:integer;
    A:array[1..100,1..100] of integer;
    stek:array[1..100] of integer;
    B:array[1..100] of boolean;
    flag:boolean;
begin
  read(n);
  for i:=1 to n do
    for j:=1 to n do
    begin
    read(A[i,j]);
    if A[i,j]=1 then k:=k+1;
    end;
  k:=k div 2;
  if k<>n-1 then begin
                 write('NO');
                 exit;
                 end;
  top:=1;
  stek[top]:=1;
  B[top]:=true;
  while top>0 do
    begin
    flag:=false;
    for j:=1 to n do
      begin
      if (A[stek[top],j]=1) and (B[j]=false) then
              begin
              top:=top+1;
              stek[top]:=j;
              B[j]:=true;
```

```
            flag:=true;
            break;
            end;
      end;
    if flag=false then top:=top-1;
    end;
    k:=0;
    for i :=1 to n do
     if B[i]=false then k:=k+1;
    if k>0 then write('NO') else write('YES');
  end.
```

ОТРИМАЙ ДЕРЕВО

Задано зв'язний неорієнтовний граф без петель і кратних ребер. Дозволяється видаляти з нього ребра. Потрібно отримати дерево.

Вхідні дані

Перший рядок містить кількість вершин **n** (**1** ≤ **n** ≤ **100**) та кількість ребер **m** графу відповідно. Наступні **m** пар чисел задають ребра графу. Гарантується, що граф зв'язний.

Вихідні дані

Виведіть **n** – **1** пару чисел - ребра, які увійдуть у дерево. Ребра можна виводити у довільному порядку.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
4 4 1 2 2 3 3 4 4 1	1 2 2 3 3 4

Розв'язання

Скористаємося пошуком у глибину.

1. Визначити будь-яку вершину **i**, як поточну і запам’ятати її першою у послідовності відвіданих вершин.
2. Якщо із поточної вершини існує ребро у нову ще не відвідану вершину **j**, то вивести пару чисел **i** та **j** – ребра, які входять в дерево, і перейти до п.3, у протилежному випадку перейти до п.4.
3. Запам’ятати вершину **j** як нову відвідану у послідовності і визначити її як поточну, перейти до п.2.
4. У послідовності, де зберігаються номери переглянутих вершин, перейти до передостанньої вершини і визначити її як поточну.
5. Якщо у послідовності ще є вершини, до яких можна повернутися, то перейти до п.2.
6. Завершити алгоритм.

Лістінг програми:

```
var i,j,n,m,x,y,top:integer;
    A:array[1..100,1..100] of integer;
    stek:array[1..100] of integer;
    B:array[1..100] of boolean;
    flag:boolean;
begin
  read(n,m);
  for i:=1 to m do
    begin
    read(x,y);
    A[x,y]:=1;
    A[y,x]:=1;
    end;
  top:=1;
  stek[top]:=1;
  B[top]:=true;
  while top>0 do
    begin
    flag:=false;
    for j:=1 to n do
      begin
      if (A[stek[top],j]=1) and (B[j]=false) then
            begin
            writeln(stek[top],' ',j);
            top:=top+1;
            stek[top]:=j;
            B[j]:=true;
            flag:=true;
            break;
            end;
      end;
    if flag=false then top:=top-1;
    end;
end.
```

КАРКАС – РОЗМИНКА 1

Замініть елементи масиву згідно завдання.

Вхідні дані

У вхідному файлі записано спочатку число **N** (**1** ≤ **N** ≤ **100**), а потім **N** чисел від **1** до **100** - елементи масиву **A**[**i**]. Далі записано два числа **q** і **w** (від **1** до **N**, не обов'язково різні).

Потрібно всі елементи, які рівні **A**[**q**], зробити рівними **A**[**w**]. Постарайтесь спочатку зчитати дані, а потім виконати те, що вимагається, і

лише потім вивести результат (а не робити перетворення на етапі виведення). Постарайтесь не користуватись додатковими масивами.

Вихідні дані

У вихідний файл виведіть **N** чисел - елементи масиву **A**[**i**] після перетворення.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
5 1 4 2 2 5 3 2	1 4 4 4 5

Розв'язання

Потрібно запам’ятати в змінну **k** значення елемента масиву **A[q]**. Замінити всі елементи масиву, які мають значення **k,** на **A[w]**.

Лістінг програми:

```
var q,w,i,n,k:longint;
    A:array[1..10000] of int64;
begin
readln(n);
for i:=1 to n do
  read(A[i]);
read(q,w);
k:=A[q];
for i:=1 to n do
  if A[i]=k then A[i]:= A[w];
for i:=1 to n do
  write(A[i],' ');
end.
```

МІНІМІЛЬНИЙ КАРКАС

Визначити вагу мінімального остовного дерева для неорієнтовного зваженого зв'язного графа.

Вхідні дані

У першому рядку знаходиться кількість вершин **n** та ребер **m** (**1** ≤ **n** ≤ **100**, **1** ≤ **m** ≤ **6000**) у графі. У кожному з наступних **m** рядків записано по трійці чисел **a**, **b**, **c**, де **a** та **b** - номери вершин, з'єднаних ребром, а **c** - вага ребра (натуральне число, яке не перевищує **30000**).

Вихідні дані

Вивести вагу мінімального остовного дерева.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	Вихідні дані
3 3 1 2 1 2 3 2 3 1 3	3

Розв'язання

Використати алгоритм Прима, який передбачає перегляд вершин заданого графа, починаючи з будь-якої з них, і наступне приєднання іншої вершини, до якої веде ребро з меншою «вагою».

Алгоритм можна сформулювати так.

1. Визначити вершину *i*, з якої починається побудова остовного дерева, як відвідану.

2. Якщо є не відвідані вершини, які ще не приєднані до остовного дерева, то визначити ту з них *j*, до якої із множини відвіданих вершин *i* веде найкоротше ребро. У протилежному випадку перейти до п.4.

3. Додати ребро (*i*,*j*) до остовного дерева, а вершину *j* – до множини відвіданих вершин. Перейти до п.3

4. Завершити алгоритм.

Лістінг програми:

```
var i,j,n,min,et,m,k,c:longint;
    sum:int64;
    D:array[1..1000,1..1000] of longint;
    visit:array[1..1000] of boolean;
begin
readln(n,m);
for k:=1 to m do
  begin
  readln(i,j,c);
  D[i,j]:=c;
  D[j,i]:=c;
  end;
visit[1]:=true;
k:=2;
while  k<=n do
  begin
  min:=32000;
  for i:=1 to n do
    for j:=1 to n do
    if (visit[i]=true) and (visit[j]=false) and (D[i,j]<min) and (D[i,j]>0)
      then
      begin
      min:=D[i,j];
```

```
      et:=j;
      end;
   visit[et]:=true;
   sum:=sum+min;
   k:=k+1;
   end;
write(sum);
end.
```

МІНІМІЛЬНИЙ КАРКАС

Потрібно знайти у зв'язному графі остовне дерево мінімальної ваги.

Вхідні дані

Перший рядок містить два натуральних числа **n** та **m** (**1** ≤ **n** ≤ **20000**, **0** ≤ **m** ≤ **100000**) - кількість вершин та ребер графа відповідно. Наступні **m** рядків містять описи ребер по одному у рядку. Ребро номер **i** описується трьома натуральними числами b_i, e_i та w_i (**1** ≤ b_i, e_i ≤ **n**, **0** ≤ w_i ≤ **100000**) - номери кінців ребра та його вага відповідно.

Граф є зв'язним.

Вихідні дані

Виведіть єдине ціле число - вагу мінімального остовного дерева.

Ліміт часу **1** секунди

Ліміт використання пам'яті **64** МіВ

Вхідні дані	**Вихідні дані**
4 4 1 2 1 2 3 2 3 4 5 4 1 4	7

Розв'язання

Використати алгоритм Краскала. Ідеєю метода є поступова побудова остовного дерева за рахунок об'єднання окремих піддерев у єдине. Спочатку до порожнього остовного дерева включається ребро з найменшою «вагою». На наступному і решті кроках додаються ребра з «найменшою вагою» серед тих, що залишились, які ще не включені до остовного дерева.

Алгоритм можна сформулювати так.

1. Визначити початковий стан остовного дерева як порожній і вважати, що всі вершини утворюють N піддерев, які не мають жодного ребра. Позначити ці піддерева порядковими номерами відповідних вершин.
2. Якщо кількість ребер в остовному дереві менша за (N-1), то серед вільних ребер заданого графа, які ще не задіяні в остовному дереві, визначити ребро з найменшою «вагою». Таким може бути лише ребро, що належить різним піддеревам. У протилежному випадку перейти до п.4.

3. Додати нове ребро до остовного дерева, а вершинам, які належать двом піддеревам, що об’єднуються, і до яких належать вершини поточного ребра, надати значення порядкового номера одного з цих піддерев.

4. Завершити алгоритм.

Лістінг програми:

```
var i,j,n,m,et,count:longint;
   sum:int64;
   A,B,D:array[1..6000] of longint;
   color:array[1..100] of longint;
   visit:array[1..100] of boolean;
procedure sort(l,r:longint);
var i,j:longint;
   c,x:longint;
begin
i:=l;
j:=r;
x:=D[(l+r) div 2];
repeat
  while D[i]<x do i:=i+1;
  while x<D[j] do j:=j-1;
  if i<=j then
        begin
        c:=D[i];
        D[i]:=D[j];
        D[j]:=c;
        c:=A[i];
        A[i]:=A[j];
        A[j]:=c;
        c:=B[i];
        B[i]:=B[j];
        B[j]:=c;
        i:=i+1;
        j:=j-1;
        end;
until i>j;
if j>l then sort(l,j);
if i<r then sort(i,r);
end;
begin
readln(n,m);
for i:=1 to m do
  begin
  readln(A[i],B[i],D[i]);
  if A[i]>B[i] then begin
```

```
                et:=A[i];
                A[i]:=B[i];
                B[i]:=et;
                end;
  end;
for i:=1 to n do
  color[i]:=i;
sort(1,m);
i:=1;
count:=0;
while i<=m do
  begin
  if (color[A[i]]<>color[B[i]]) and (visit[A[i]]=false)
                        and (visit[B[i]]=false) then
                        begin
                        sum:=sum+D[i];
                        color[B[i]]:=color[A[i]];
                        count:=count+2;
                        visit[A[i]]:=true;
                        visit[B[i]]:=true;
                        end
                        else
                          if (color[A[i]]<>color[B[i]]) and (visit[A[i]]=true)
                                              and (visit[B[i]]=false) then
                          begin
                          sum:=sum+D[i];
                          color[B[i]]:=color[A[i]];
                          count:=count+1;
                          visit[B[i]]:=true;
                          end
                          else
                            if (color[A[i]]<>color[B[i]]) and (visit[A[i]]=false)
                            and (visit[B[i]]=true) then
                            begin
                            sum:=sum+D[i];
                            color[A[i]]:=color[B[i]];
                            count:=count+1;
                            visit[A[i]]:=true;
                            end
                            else
                              if (color[A[i]]<>color[B[i]]) and (visit[A[i]]=true)
                                 and (visit[B[i]]=true) then
                              begin
                              sum:=sum+D[i];
```

```
                    et:=color[B[i]];
                    for j:=1 to n do
                      if color[j]=et then color[j]:=color[A[i]];

                  end;
  i:=i+1;
  end;
write(sum);
end.
```

Задачі з використанням основ динамічного програмування. Одновимірна динаміка.

Ідеї розв'язання задачі

Голодяєвої П.В., учениці 9-го класу загальноосвітньої спеціалізованої школи I-III ступенів фізико-математичного профілю № 12 м. Чернігова

КОНИК

У одного з викладачів у кімнаті живе коник, який дуже полюбляє стрибати по одномірній клітчатій дошці. Довжина дошки **n** клітинок. На жаль, він вміє стрибати лише на **1**, **2**, ..., **k** клітинок уперед.

Одного разу викладачам стало цікаво, скількома способами коник може дострибати з першої клітинки до останньої. Допоможіть їм відповісти на це питання.

Вхідні дані

Два цілих числа **n** та **k** (**1** ≤ **n** ≤ **30**, **1** ≤ **k** ≤ **10**).

Вихідні дані

Виведіть кількість способів, якими коник зможе дострибати з першої клітинки до останньої.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
8 2	21

Розв'язання

Кількість шуканих способів – це **n**-ий член ряду Фібоначчі, де $\mathbf{F_1=1}$, $\mathbf{F_i}$ – сума **k-1** попередніх елементів цього ряду.

Лістінг програми:

```
var n,k,i,j:longint;
A:array[-9..40] of longint;
begin
```

```
read(n,k);
A[1]:=1;
for i:=2 to n do
  for j:=i-k to i-1 do
    A[i]:=A[i]+A[j];
write(A[n]);
end.
```

Ідеї розв'язання задач

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи І-ІІІ ступенів фізико-математичного профілю № 12 м. Чернігова

ТРИ ОДИНИЦІ ПІДРЯД

За заданим числом **N** визначте кількість послідовностей з нулів та одиниць довжни **N**, у яких ніякі три одиниці не стоять поряд.

Вхідні дані

У вхідному файлі написано натуральне число **N**, яке не перевищує **35**.

Вихідні дані

Виведіть кількість шуканих послідовностей. Гарантується, що відповідь не перевищує $\mathbf{2^{31}-1}$.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** МіВ

Вхідні дані	**Вихідні дані**
4	13

Розв'язання

Кількість шуканих послідовностей – це **n**-ий член ряду Фібоначчі, де $\mathbf{F_1 = 2}$, $\mathbf{F_2 = 4}$, $\mathbf{F_3 = 7}$, $\mathbf{F_i = F_{i-1} + F_{i-2} + F_{i-3}}$.

Лістінг програми:

```
var a1,a2,a3,n:int64;
    i:longint;
begin
readln(n);
a1:=2;
a2:=4;
a3:=7;
if n=1 then
      begin
      write(2);
      exit;
      end;
if n=2 then
```

```
        begin
        write(4);
        exit;
        end;
    for i:=4 to n do
      begin
      a3:=a1+a2+a3;
      a2:=a3-a1-a2;
      a1:=a3-a1-a2;
      end;
    write(a3);
    end.
```

КАЛЬКУЛЯТОР

Є калькулятор, який виконує наступні операції:

- помножити число **X** на **2**;
- помножить число **X** на **3**;
- додати до числа **X** одиницю.

Визначте, яку найменшу кількість операцій потрібно, щоб отримати з числа **1** число **N**.

Вхідні дані

У вхідному файлі записано натуральне число **N**, яке не перевищує $\mathbf{10^6}$.

Вихідні дані

У першому рядку вихідного файлу виведіть мінімальну кількість операцій. У другому рядку виведіть числа, які послідовно отримуються при виконанні операцій. Перше з них повинно бути рівним **1**, а останнє **N**. Якщо розв'язків декілька, виведіть довільний.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
Sample 1 1 Sample 2 5 Sample 3 962340	Sample 1 0 1 Sample 2 3 1 3 4 5 Sample 3 17 1 3 9 27 54 55 165 495 1485 4455 8910 17820 17821 53463 160389 160390 481170 962340

Розв'язання

Використаємо алгоритм, який число **N** перетворює в **1** шляхом ділення його на 3 або на 2 або відніманням від нього 1.

Лістінг програми:

```
var k,i,n:longint;
    A:array[1..10000] of int64;
begin
read(n);
i:=1;
A[i]:=n;
while n<>1 do
  begin
  if n mod 3 =0 then
                begin
                i:=i+1;
                n:=n div 3;
                A[i]:=n;
                end
               else
                if n mod 2 =0 then
                            begin
                            i:=i+1;
                            n:=n div 2;
                            A[i]:=n;
                            end
                           else
                            begin
                            i:=i+1;
                            n:=n-1;
                            A[i]:=n;
                            end;
  end;
writeln(i-1);
for k:=i downto 1 do
  write(A[k],' ');
end.
```

ВИБУХОНЕБЕЗПЕЧНІСТЬ

На одному з секретних заводів здійснюється обробка радіоактивних матеріалів, в результаті якої утворюються радіоактивні відходи двох типів: типу А (особливо небезпечні) і типу В (безпечні). Усі відходи упаковуються в спеціальні прямокутні контейнери однакових розмірів, після чого ці контейнери складаються у стопку один над одним для збереження. Стопка є вибухонебезпечною, якщо в ній є сусідами два ящики з відходами типу **А.** Потрібно написати програму, яка підраховує кількість можливих варіантів

формування вибухобезпечної стопки із заданого загального числа контейнерів N.

Вхідні дані

У вхідному файлі міститься єдине число N ($1 \le N \le 100$).

Вихідні дані

У вихідний файл необхідно вивести шукане число варіантів.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
4	8

Розв'язання

Шукане число варіантів формування вибухобезпечної стопки – це **n**-ий член ряду Фібоначчі, де $\mathbf{F_1 = 2}$, $\mathbf{F_2 = 3}$, $\mathbf{F_i = F_{i-1} + F_{i-2}}$.

Лістінг програми:

```
var A:array [1..100] of int64;
   i,n:longint;
begin
readln(n);
a[1]:=2;
a[2]:=3;
for i:=3 to n do
  A[i]:=A[i-1]+A[i-2];
write(A[n]);
end.
```

ЧИСЛА ФІБОНАЧЧІ

Числа Фібоначчі задаються формулами $\mathbf{F_1 = 1}$, $\mathbf{F_2 = 1}$, $\mathbf{F_i = F_{i-1} + F_{i-2}}$. Потрібно порахувати останні **k** цифр **n**-го числа Фібоначчі.

Вхідні дані

У першому рядку вхідного файлу міститься натуральне число **n**. $\mathbf{n \le 10^{18}}$, $\mathbf{k = 3}$.

Вихідні дані

Перший рядок вихідного файлу повинен містити єдине число - відповідь до задачі.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
Sample 1 2 Sample 2 10	Sample 1 1 Sample 2 55

Розв'язання

Число одиниць в числах Фібоначчі повторюється періодично через кожні 60 чисел. Останні дві цифри повторюються з періодичністю 300, а останні три цифри – з періодичністю 1500.

Лістінг програми:

```
var i,n:int64;
    A:array[0..1500] of int64;
begin
readln(n);
A[0]:=0;
A[1]:=1;
A[2]:=1;
for i:=3 to 1500 do
  A[i]:=(A[i-1]+A[i-2]) mod 1000;
write(A[n mod 1500]);
end.
```

Ідеї розв’язання задачі

Пупова Н. А., учня 10 класу загальноосвітньої спеціалізованої школи І-ІІІ ступенів фізико-математичного профілю № 12 м. Чернігова

СХОДИНКИ

На кожній з **n** + **2** сходинок сходів написане ціле число, причому на першій та на останній сходинках записано число **0**. На першій сходинці стоїть людина, якій потрібно піднятися на останню сходинку. За один крок вона може підніматись на довільну кількість сходинок, не більшу за **k.**

Підрахуємо суму всіх чисел, написаних на сходинках, на які наступила людина. Знайдіть найбільше можливе значення цієї суми.

Вхідні дані

У першому рядку міститься число **n** (**0** ≤ **n** ≤ **1000**). У другому рядку записано **n** цілих чисел, які не перевищують за модулем **1000**, відокремлених пропусками - числа, написані на сходинках (за виключенням першої та останньої сходинки, на яких написані нулі). У третьому рядку записано максимальну величину кроку людини **k** (**1** ≤ **k** ≤ **n**).

Вихідні дані

Вивести максимально можливу суму чисел, написаних на сходинках, на які наступила людина.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
3 1 -1 1 2	2

Розв'язання

Нехай масив **A** містить числа записані на сходах.

Створимо масив «максимальних сум» **B**, в який будемо зберігати найбільшу суму для кожної клітинки, де **B[i]=max+A[i]**, **max** – максимум в масиві **B** від **i-1** до **i-k**. Виведемо значення B[n-1].

Лістінг програми:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cstring>
#include <string>
#include <math.h>
#include <iomanip>
#include <memory.h>
#include <algorithm>
#include <cmath>
#include <vector>
#include <sstream>
#include <queue>

using namespace std;

int main()
{
	ifstream fin("input.txt");
	ofstream fout("output.txt");
	int n, k;
	fin >> n;
	long long [2000];
	long long [2000];
	A[0] = 0;
	A[n+1] = 0;
	for (int i = 1; i <= n; i++)
		fin >> A[i];

	n += 2;
	fin >> k;
	B[0] = 0;
	for (int i = 1; i < n; i++)
	{
		int k1;
		k1 = k;
		long long max = -1000000000000000;
		for (int j = i-1; j >= 0 && k1 > 0; j--, k1--)
		{
```

```
                    if (B[j]>max)
                        max = B[j];
                }
                B[i] = max + A[i];
            }
            fout << B[n-1];
            fin.close();
            fout.close();
        }
```

Задачі з використанням двовимірна динаміка на таблицях

Ідеї розв'язання задач

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи І-ІІІ ступенів фізико-математичного профілю № 12 м. Чернігова

ТРИКУТНИК ПАСКАЛЯ

Трикутник Паскаля - це числовий трикутник, по краям якого стоять одиниці, а кожне число всередині дорівнює сумі двох чисел вгору-праворуч і вгору-ліворуч.

Із-за помилки набірника трикутник Паскаля виявився записаним у рядок і утворилась послідовність виду **1**, **1**, **1**, **1**, **2**, **1**, **1**, **3**, **3**, **1**, **1**, **4**, **6**, **4**, **1**, ...

Вхідні дані

Задано один рядок, який містить натуральне число **N** (**N** ≤ **600**).

Вихідні дані

Потрібно вивести один рядок, який містить **N**-ий член утвореної послідовності.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
18	10

Розв'язання

Заповнимо масив **A** числами, що утворюють трикутник Паскаля, **A[1,1]=1**, а **A[i,j]** визначається, як сума елементів, що знаходяться в попередньому рядку над ним зверху та зліва.

1

1 1

1 2 1
1 3 3 1
1 4 6 4 1
Виведемо **A[i,j]**, який є **N**-ий член утвореної послідовності.

Лістінг програми:

```
var i,j,n,s:longint;
   A:array[0..50,0..50] of int64;
begin
readln(n);
A[1,1]:=1;
s:=1;
if s=n then begin
            write(1);
            exit;
            end;
for i:=2 to 50 do
  for j:=1 to i do
    begin
    A[i,j]:=A[i-1,j-1]+A[i-1,j];
    s:=s+1;
    if s=n then begin
            write(A[i,j]);
            exit;
            end;
    end;
end.
```

МАРШРУТ

У таблиці **N** рядків і **N** стовбців клітинки заповнені цифрами від **0** до **9**. Потрібно знайти такий шлях з клітинки (**1**, **1**) у клітинку (**N**, **N**), щоб сума цифр у клітинках, через які він проходить, був мінімальним; з довільної клітинки ходити можна лише вниз або праворуч.

Вхідні дані

У першому рядку знаходиться число **N** (**2** ≤ **N** ≤ **250**). У наступних **N** рядках міститься по **N** цифр без пропусків.

Вихідні дані

Виводиться **N** рядків по **N** символів. Символ решітка показує, що маршрут проходить через цю клітинку, а точка - що не проходить. Якщо шляхів з мінімальною сумою цифр декілька, вивести довільний.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64**МіВ

Вхідні дані	Вихідні дані
3 943 216 091	#.. ### ..#

Розв'язання

Ця задача є класичною задачею з використанням методу динамічного програмування. Розглянути всі можливі маршрути майже нереально.

Будемо шукати найкращі шляхи з верхньої лівої клітинки у всі інші. Для цього створимо таблицю «мінімальних сум»(у програмі – масив **B**).

Оскільки у клітинку **(i,j)** ми можемо потрапити тільки з сусідніх клітинок: зліва **(i,j-1)** і зверху **(i-1,j)**. Пріоритет віддамо клітинці з найменшим значенням. Отже, рухаючись по рядках (або стовпчиках), будемо заповнювати таблицю **B**, використовуючи формулу:

B[i,j]:=A[i,j]+min(B[i-1,j], B[i,j-1]).

Для виведення шляху застосуємо проходження таблиці **B** у зворотньому порядку, використавши початкову таблицю **A**. Якщо різниця **B[i,j]** і **A[i,j]** дорівнює **B[i-1,j]**, то рухаємось угору, а інакше ліворуч.

Лістінг програми:

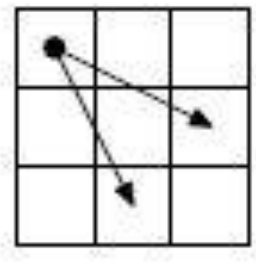
```
var i,j,n,k:longint;
    s:string;
    A,B:array[0..250,0..250] of int64;
    C:array[0..250,0..250] of char;
begin
readln(n);
for i:=1 to n do
  begin
  readln(s);
  for j:=1 to n do
    val(s[j],A[i,j],k);
  end;
for i:=1 to n do
  begin
  B[0,i]:=2000000;
  B[i,0]:=2000000;
  end;
B[1,1]:=A[1,1];
for i:=1 to n do
  for j:=1 to n do
    if (i=1) and (j=1) then
                  else
                  if A[i,j]+B[i-1,j]<A[i,j]+B[i,j-1] then
                                        B[i,j]:=A[i,j]+B[i-1,j]
                                        else
```

```
                                B[i,j]:=A[i,j]+B[i,j-1];
C[n,n]:='#';
i:=n;
j:=n;
while (i<>1) or (j<>1) do
  if B[i-1,j]<B[i,j-1] then
                begin
                C[i-1,j]:='#';
                i:=i-1;
                end
                else
                begin
                C[i,j-1]:='#';
                j:=j-1;
                end;
C[1,1]:='#';
for i:=1 to n do
 begin
 for j:=1 to n do
   if C[i,j]='#' then write(C[i,j]) else write('.');
 writeln;
 end;
end.
```

ХІД КОНЕМ

Задано прямокутну дошку **n** × **m** (**n** рядків та **m** стовпчиків). У лівому верхньому куті знаходиться шаховий кінь, якого необхідно перемістити у правий нижній кут дошки. У даній задачі кінь може переміщуватись на дві клітинки униз та на одну клітинку праворуч або на одну клітинку униз та на дві клітинки праворуч.

Необхідно визначити, скільки існує різних маршрутів, які ведуть з лівого верхнього у правий нижній кут.



Вхідні дані

Два натуральних числа **n** та **m** (**1** ≤ **n**, **m** ≤ **50**).

Вихідні дані

Виведіть кількість способів дістатись конем з лівого верхнього до правого нижнього кута дошки.

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** МіВ

Вхідні дані	Вихідні дані
3 2	1
31 34	293930

Розв'язання

В масив **A** будемо записувати кількість способів потрапити в клітинку **(i,j)**.

Кількість маршрутів, які ведуть в клітинку **(i,j)** дорівнює:

A[i,j]:=A[i-1,j-2]+A[i-2,j-1];

Виводимо значення елемента **A[n,m]**.

Лістінг програми:

```
var i,j,n,m:longint;
   A:array[-1..50,-1..50] of int64;
begin
  readln(n,m);
  A[1,1]:=1;
  for i:=1 to n do
    for j:=1 to m do
      if (i=1) and (j=1) then else
        A[i,j]:=A[i-1,j-2]+A[i-2,j-1];
  write(A[n,m]);
end.
```

ЧЕРЕПАШКА

У лівому верхньому куті прямокутної таблиці розміром **n** × **m** знаходиться черепашка. На кожній клітинці таблиці розлито деяку кількість кислоти. Черепашка може переміщуватись праворуч або вниз, при цьому маршрут черепашки завершується у правому нижньому куті таблиці.

Кожен мілілітр кислоти причиняє черепашці деяку кількість шкоди. Знайдіть найменше можливе значення шкоди, яку отримає черепашка після прогулянки по таблиці.

Вхідні дані

У першому рядку вхідних даних записано два натуральних числа **n** та **m**, які не перевищують **1000** - розміри таблиці. Далі йде **n** рядків, кожен з яких містить **m** чисел, відокремлених пропусками - опис таблиці з вказуванням для кожної клітинки вмісту кислоти на ній (у мілілітрах).

Вихідні дані

Вивести мінімальну можливу вартість маршруту черепашки.

Ліміт часу **1** секунд

Ліміт використання пам'яті **122.44** MiB

Вхідні дані	Вихідні дані
3 4 5 9 4 3 3 1 6 9 8 6 8 12	35
1 1 1	1

Розв'язання

Ця задача є класичною задачею з використанням методу динамічного програмування. Розглянути всі можливі маршрути майже нереально.

Будемо шукати найкращі шляхи з верхньої лівої клітинки у всі інші. Для цього створимо таблицю «мінімальних сум»(у програмі – масив **B**).

Оскільки у клітинку **(i,j)** ми можемо потрапити тільки з сусідніх клітинок: зліва **(i,j-1)** і зверху **(i-1,j)**. Пріоритет віддамо клітинці з найменшим значенням. Отже, рухаючись по рядках (або стовпчиках), будемо заповнювати таблицю **B**, використовуючи формулу:

B[i,j]:=A[i,j]+min(B[i-1,j], B[i,j-1]).

Відповідь – значення елемента **B[n,m]**.

Лістінг програми:

```
var i,j,n,m:longint;
    A:array[1..1000,1..1000] of longint;
    B:array[0..1001,0..1001] of int64;
begin
  readln(n,m);
  for i:=1 to n do
    for j:=1 to m do
    read(A[i,j]);
  B[1,1]:=A[1,1];
  for i:=1 to n do
    B[i,0]:=20000000;
  for j:=1 to m do
    B[0,j]:=20000000;
  for i:=1 to n do
    for j:=1 to m do
    begin
    if (i=1) and (j=1) then else
      if B[i,j-1]<B[i-1,j] then  B[i,j]:=A[i,j]+B[i,j-1];
                           else B[i,j]:=A[i,j]+B[i-1,j];
    end;
write(B[n,m]);
end
```

ВАРТІСТЬ МАРШРУТУ

На кожній клітинці шахової дошки розміром **8×8** записано ціле невід'ємне число. Король може переміщуватись по шаховій дошці з лівого нижнього кута у правий верхній, переміщуючись лише праворуч, вгору, або праворуч-вгору. При цьому вартість проходу через дану клітинку дорівнює числу, записаному на цій клітинці.

Перемістіть короля з лівого нижнього кута у правий верхній з найменшою вартістю проходу.

Вхідні дані

На вхід програмі подається вісім рядків, кожен рядок містить вісім цілих невід'ємних чисел, які не перевищують **1000**. У лівому нижньому куті завжди записано число **0**.

Вихідні дані

У першому рядку виведіть єдине число - мінімальну вартість проходу з лівого нижнього кута у правому верхньому. У другому рядку виведіть маршрут короля заданої вартості, відокремлюючи клітинки одним пропуском. Маршрут повинен починатись клітинкою **a1** і завершуватись клітинкою **h8**.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	Вихідні дані
9 9 9 9 9 9 1 9 9 9 9 9 9 1 9 2 9 9 9 9 9 9 1 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9	56 a1 a2 b3 c4 d5 e6 f7 g8 h8

Розв'язання

Будемо шукати найкращі шляхи з нижньої лівої клітинки у всі інші. Для цього створимо таблицю «мінімальних сум»(у програмі – масив **B**).

Оскільки у клітинку **(i,j)** ми можемо потрапити тільки з сусідніх клітинок: зліва **(i,j-1)**, знизу **(i+1,j)** та знизу-по діагоналі **(i+1,j-1)**. Пріоритет віддамо клітинці з найменшим значенням. Отже, рухаючись по рядках (або стовпчиках), будемо заповнювати таблицю **B**, використовуючи формулу:

B[i,j]:=A[i,j]+min(B[i+1,j],B[i,j-1],B[i+1,j-1]).

Лістінг програми:

```
var i,j,s:longint;
   A,B:array[0..9,0..9] of longint;
   st1,st2:array[1..8] of char;
   t:array[0..9,0..9] of string;
```

```
function min(x,y,z:longint):longint;
begin
  if (x<=y) and (x<=z) then min:=x
                       else
                       if y<=z then min:=y
                              else min:=z;
end;
begin
  for i:=1 to 8 do
    for j:=1 to 8 do
    read(A[i,j]);
   for i:=0 to 8 do
     begin
     B[i,0]:=10000;
     B[9,i]:=10000;
     end;
   st1[8]:='a';
   st1[7]:='b';
   st1[6]:='c';
   st1[5]:='d';
   st1[4]:='e';
   st1[3]:='f';
   st1[2]:='g';
   st1[1]:='h';
   st2[1]:='1';
   st2[2]:='2';
   st2[3]:='3';
   st2[4]:='4';
   st2[5]:='5';
   st2[6]:='6';
   st2[7]:='7';
   st2[8]:='8';
   for i:=8 downto 1 do
     for j:=1 to 8 do
     begin
     if (i=8) and (j=1) then else
     begin
       s:=min(B[i+1,j],B[i,j-1],B[i+1,j-1]);
       B[i,j]:=A[i,j]+s;
       if s=B[i+1,j] then t[i,j]:=t[i+1,j]+st1[i+1]+st2[j]+' ';
       if s=B[i+1,j-1] then t[i,j]:=t[i+1,j-1]+st1[i+1]+st2[j-1]+' ';
       if s=B[i,j-1] then t[i,j]:=t[i,j-1]+st1[i]+st2[j-1]+' ';
     end;
     end;
```

```
  writeln(B[1,8]);
  writeln(t[1,8]+'h8');
end.
```

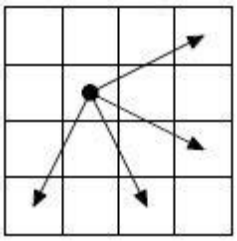
Ідеї розв'язаня задачі

Чорного А.В., учня 11-го класу Чернігівськкого колегіуму № 11 Чернігівської міської ради

ХІД КОНЕМ - 2

Задано прямокутну дошку **N**×**M** (**N** рядків та **M** стовбців). У лівому верхньому куті знаходиться шаховий кінь, якого необхідно перемістити у правий нижній кут дошки.

ри цьому кінь може ходити наступним чином:



Необхідно визначити, скільки існує різних маршрутів, які ведуть з лівого верхнього у правий нижній кут.

Вхідні дані

Вхідний файл містить два натуральних числа **N** та **M** (**1 ≤ N, M≤ 50**).

Вихідні дані

У вихідний файл виведіть єдине число - кількість способів дістатись конем до правого нижнього кута дошки.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
Sample 1 4 4 Sample 2 2 3 Sample 3 15 14	Sample 1 2 Sample 2 1 7884330

Розв'язання

У двомірному масиві будемо зберігати кількість варіантів, як можна дістатись до даної клітинки. На початку заповнимо її нулями, крім першої клітинки, де буде 1.

У комірках матриці зберігаємо «довгі» числа.

Проходимо матрицею паралельно побічній діагоналі матриці і значення кожної комірки, у яку може перейти кінь, збільшуємо на значення поточної комірки.

```
#include <iostream>
#include <cmath>
#include <vector>
```

```
using namespace std;
class larfm{
    public:
    int m[100], len;
    void sum(larfm x){
        if (x.len > len)
            len = x.len + 1;
        else
            len = len + 1;
        for (int ix = 0; ix < len; ix++)
        {
            m[99-ix] += x.m[99-ix];
            m[98-ix] += (m[99-ix] / 10);
            m[99-ix] %= 10;
        }
        if (m[100-len] == 0)
            len--;
        if (m[100-len] == 0)
            len--;
        if (len==0) len=1;
    }
};
int main()
{
    int A,B;
    cin>>A>>B;
    vector<vector<larfm> > m;
    m.resize(A);
    for (int i=0; i<A; i++){
        m[i].resize(B);
        for (int j=0; j<B; j++){
            m[i][j].m[99]=0;
            m[i][j].len=1;
        }
    }
    m[0][0].m[99]=1;

    for (int t=0; t<A; t++){
        int i=t,j=0;
        while (i>=0 && j<B){
            if (m[i][j].m[99]>0 || m[i][j].len>1){
                //cout<<'!'<<endl;
                if (i+2<A && j-1>=0) m[i+2][j-1].sum(m[i][j]);
                if (i+2<A && j+1<B) m[i+2][j+1].sum(m[i][j]);
                if (i+1<A && j+2<B) m[i+1][j+2].sum(m[i][j]);
```

```
            if (i-1>=0 && j+2<B) m[i-1][j+2].sum(m[i][j]);
        }
        i--;
        j++;
    }
}
//cout<<'!'<<endl;
for (int t=1; t<B; t++){
    int i=A-1,j=t;
    while (i>=0 && j<B){
          //cout<<'!'<<endl;
         // cout<<i<<' '<<j<<endl;
        if (m[i][j].m[99]>0 || m[i][j].len>1){
            if (i+2<A && j-1>=0) m[i+2][j-1].sum(m[i][j]);
            if (i+2<A && j+1<B) m[i+2][j+1].sum(m[i][j]);
            if (i+1<A && j+2<B) m[i+1][j+2].sum(m[i][j]);
            if (i-1>=0 && j+2<B) m[i-1][j+2].sum(m[i][j]);
        }
        i--;
        j++;
    }
}
/*for (int i=0; i<A; i++){
    for (int j=0; j<B; j++){
        for (int z=100-m[i][j].len; z<100; z++){
            cout<<m[i][j].m[z];
        }
        cout<<' ';
    }
    cout<<endl;
}
cout<<endl;*/
for (int i=100-m[A-1][B-1].len; i<100; i++){
      //cout<<'!'<<i<<'!'<<endl;
    cout<<m[A-1][B-1].m[i];
}
cout<<endl;
//cout<<m[A-1][B-1]<<endl;
return 0;
}
```

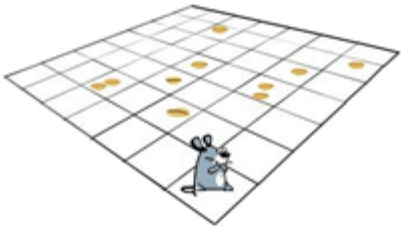
Ідеї розв'язання задачі

Пупова Н. А., учня 10 класу загальноосвітньої спеціалізованої школи І-ІІІ ступенів фізико-математичного профілю № 12 м. Чернігова

МИШКА І ЗЕРНИНКА

В індійському храмі підлогу прямокутної форми вимощено однаковими квадратними плитками **1** х **1**, на кожну з яких насипано від **0** до **k** зернинок (**k** ≤ **30000**). Розміри підлоги **m** х **n**. Мишка вибігає з лівого нижнього кута підлоги храму і рухається до входу у іншу нірку, розміщену у протилежному кутку. Мишка може рухатись лише праворуч або вперед, збираючи всі зернинки з плитки, на якій вона знаходиться. Знайти маршрут, рухаючись по якому мишка збере найбільшу кількість зернин.

Вхідні дані



Перший рядок містить числа **m** та **n** - розміри підлоги (**1** ≤ **m**, **n** ≤ **100**). Далі йде **m** рядків, починаючи з верхнього, у кожному з яких розміщено **n** чисел – кількість зернинок на відповідній плитці.

Вихідні дані

Вивести маршрут руху мишки у форматі: **RRFFFRF** (**F** – крок вперед, **R** – крок праворуч).

Ліміт часу **1** секунда

Ліміт використання пам'яті **122.17** MiB

Вхідні дані	**Вихідні дані**
2 3 3 2 4 1 5 1	RFR

Розв'язання

Будемо шукати найкращі шляхи з нижньої лівої клітинки у всі інші. Для цього створимо таблицю «максимальних сум»(у програмі – масив **B**).

Оскільки у клітинку **(i,j)** ми можемо потрапити тільки з сусідніх клітинок: зліва **(i,j-1)** і знизу **(i+1,j)**. Пріоритет віддамо клітинці з найбільшим значенням.

Лістінг програми:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <cstring>
#include <string>
#include <math.h>
#include <iomanip>
#include <memory.h>
#include <algorithm>
```

```
#include <vector>
#include <sstream>
#include <queue>

using namespace std;

template <typename T>
string to_string(T a)
{
  stringstream n;
  n << a;
  return n.str();
}
struct mas{ long long fromx, fromy, cost; };
int main()
{
  ifstream fin("input.txt");
  ofstream fout("output.txt");
  int A[200][200];
  mas dp[200][200];
  int n, m;
  fin >> n >> m;
  for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
              fin >> A[i][j];

  for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
              dp[i][j].cost = -100;

  dp[n-1][0].cost = A[0][0];
  dp[n-1][0].fromx = -1;
  dp[n-1][0].fromy = -1;

  for (int i = n-1; i >=0; i--)
        for (int j = 0; j < m; j++)
        {
              if (i <n-1)
              {
                    if (dp[i][j].cost <= dp[i + 1][j].cost + A[i][j])
                    {
                          dp[i][j].cost = dp[i + 1][j].cost + A[i][j];
                          dp[i][j].fromx = i + 1;
                          dp[i][j].fromy = j;
                    }
```

```
                }
                if (j > 0)
                {
                        if (dp[i][j].cost <= dp[i][j-1].cost + A[i][j])
                        {
                                dp[i][j].cost = dp[i][j-1].cost + A[i][j];
                                dp[i][j].fromx = i;
                                dp[i][j].fromy = j-1;
                        }
                }
        }
    string ansv = "";
    int fromx = 0;
    int fromy = m - 1;

    for(;;)
    {
        int fromx1 = fromx;
        int fromy1 = fromy;
        fromx = dp[fromx1][fromy1].fromx;
        fromy = dp[fromx1][fromy1].fromy;
        if (fromx == -1)
                break;
        if (fromx != fromx1)
                ansv += "F";
        else
                ansv += "R";
    }
    for (int i = ansv.size() - 1; i >= 0; i--)
        fout << ansv[i];
    fin.close();
    fout.close();
}
```

Задачі з використанням елементів комбінаторики

Ідеї розв’язаня задачі

Чорного А.В., учня 11-го класу Чернігівськкого колегіуму № 11 Чернігівської міської ради

ЛИСТ ПОШТАРЯ ПЄЧКІНА

Дорогі дітки!

Спостерігаючи за тим, як Шарик розпилював нестандартну шахову дошку, я також вирішив задати для вас задачку: "А скільки різних квадратних і прямокутних (не рахуючи квадратних) дощок міг би отримати при розпилюванні Шарик зі знайденої ним нестандартної прямокутної шахової дошки розміром **M** x **N***?"*

Вхідні дані

У першому рядку кількість завдань Пєчкіна **K**, у наступних **K** рядках по два цілих числа **M** та **N** (**1** ≤ **K**, **M**, **N** ≤ **100**), відокремлених пропуском.

Вихідні дані

Для кожного прикладу, заданого Пєчкіним, виведіть в окремому рядку через пропуск шукані кількості спочатку квадратних, а потім прямокутних дощок.

Ліміт часу **0.5** секунд

Ліміт використання пам'яті **64** MiB

Вхідні дані	**Вихідні дані**
1 3 2	8 10

Пояснення: Шарик міг би отримати квадратних дошок: 6 зі стороною 1 і дві зі стороною 2 – всього 8, прямокутних: 4 дошки 2x1, 3 дошки 1x2, 2 дошки 3x1 і одну початкову дошку 3x2 – всього 10.

Розв'язання

Задачу розв'язано перебором.

Перебираємо значення по одній і по другій стороні у вкладених циклах. Якщо це квадрат, то додаємо f до лічильника ківадратів - ansk, інакше до лічильника прямокутників - ansp

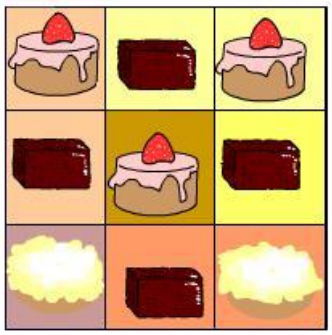
```
#include <iostream>
using namespace std;
long long fact(long long a, long long b){
   long long q=1;
   for (long long i=a; i<=b; i++){
      q*=i;
   }
   return q;
}
long long Cmn(long long m, long long n){
   if (m>=n) return 0;
   return fact(m, n)/fact(2,n-m);
}
int main()
```

```
{
   long long t;
   cin>>t;
   for (long long ti=0; ti<t; ti++){
      long long x,y,ansk=0,ansp=0;
      cin>>x>>y;
      for (long long i=1; i<=x; i++){
         for (long long j=1; j<=y; j++){
            long long f=(x+1-i)*(y+1-j);
            //cout<<i<<' '<<j<<' '<<f<<' '<<Cmn(i,x)<<' '<<Cmn(j,y)<<endl;
            if (i!=j) ansp+=f;
               else ansk+=f;
         }
      }
      cout<<ansk<<' '<<ansp<<endl;
   }
   return 0;
}
```

Ідеї розв'язання задач

Хрол Н. П., учителя інформатики загальноосвітньої спеціалізованої школи I-III ступенів фізико-математичного профілю № 12 м. Чернігова

ШКІЛЬНИЙ БУФЕТ



У шкільному буфеті до завершення уроків залишилось декілька тістечок: **A** ванільних, **B** шоколадних і **C** фруктових. Дмитро збирається придбати тістечка перед закриттям буфету. Скільки тістечок може вибрати Дмитро?

Вхідні дані

У єдиному рядку через пропуск задано три цілих невід'ємних числа - відповідні кількості тістечок, кожне з яких не перевищує **20000**.

Вихідні дані

Єдине число - відповідь до задачі.

Ліміт часу **0.1** секунд

Ліміт використання пам'яті **64** MiB

Вхідні дані	Вихідні дані
4 2 3	9

Пояснення: У даній задачі потрібно знайти максимально можливу кількість тістечок, вважати, що у Дмитра вистачить грошей на їх придбання.

Розв'язання

Дмитро може вибрати всі тістечка, що залишились в буфеті.

Лістінг програми:

```
var a,b,c:int64;
begin
read(a,b,c);
write(a+b+c);
end.
```

ЗМАГАННЯ З ТЕНІСУ

Необхідно сформувати команду, яка буде представляти навчальний заклад у змаганнях з тенісу. У секції тенісу займається **A** дівчат і **B** хлопців. Скільки різних змішаних пар можна вибрати для участі у змаганнях?

Вхідні дані

У єдиному рядку через пропуск знаходиться **2** цілих невід'ємних числа **A** та **B**, які не перевищують 10^6.

Вихідні дані

Єдине число - відповідь до задачі.

Ліміт часу **0.1** секунд

Ліміт використання пам'яті **64** MiB

Вхідні дані	Вихідні дані
6 9	54

Розв'язання

Кожна дівчина може зіграти разом в теніс з кожним хлопцем.

Лістінг програми:

```
var a,b:int64;
begin
read(a,b);
write(a*b);
end.
```

ЛАНЧ

Влад хоче взяти з собою для ланчу пару фруктів. У нього є **A** бананів, **B** яблук і **C** груш. Скількома способами він може вибрати **2** фрукти різного виду з наявних у нього?

Вхідні дані

У єдиному рядку через пропуск задано три невід'ємних числа: **A**, **B** і **C**. Всі числа не перевищують 10^6.

Вихідні дані

Єдине число - відповідь до задачі.

Ліміт часу **1** секунд

Ліміт використання пам'яті **64** MiB

Вхідні дані	Вихідні дані
3 4 2	26

Розв'язання

Влад може взяти з собою для ланчу до кожного з **A** бананів будь-яке з **B** яблук або до кожного з **B** яблук будь-яку з **C** груш або до кожної з **C** груш будь-який з **A** бананів.

Лістінг програми:

```
var a,b,c:int64;
begin
read(a,b,c);
write(a*b+b*c+a*c);
end.
```

КІЛЬКІСТЬ ПЕРЕСТАНОВОК

За заданим натуральним числом **n** знайти кількість різних перестановок чисел від **1** до **n**.

Вхідні дані

Одне число **n** (**1** ≤ **n** ≤ **12**).

Вихідні дані

Вивести кількість різних перестановок чисел від **1** до **n**.

Ліміт часу **1** секунда

Ліміт використання пам'яті **64** MiB

Вхідні дані	Вихідні дані
3	6

Розв'язання

Кількість перестановок з **n** елементів дорівнює **n!**.

Лістінг програми:

```
var i,n:longint;
    p:int64;
begin
read(n);
p:=1;
for i:=1 to n do
  p:=p*i;
write(p);
end.
```