**Ministerul Educaţiei al Republicii Moldova**
**Agenţia Naţională pentru Curriculum şi Evaluare**

# Olimpiada Republicană la Informatică 2017

**Ediţia XXX**

**Chişinău 2017**

# Cuprins

# Clasele 7 – 9

## Cifra de control

Este cunoscut faptul că datele de pe purtătorii de informaţie pot fi compromise. În scopul depistării eventualelor erori, s-a decis ca fiecare număr natural de pe oricare purtător de informaţie să fie însoţit şi de o cifră de control. Imediat cum numărul este citit de pe purtătorul de informaţie, se calculează cifra lui de control. Dacă ea nu coincide cu cifra de control stocată anterior pe purtător, se consideră că datele de pe purtătorul de informaţie sunt compromise.

Cifră de control *C* a numărului natural *A* se calculează după cum urmează:

1) se însumează toate cifrele numărului *A*, numărul obţinut fiind notat prin $S_1$;
2) se însumează toate cifrele numărului $S_1$, numărul obţinut fiind notat prin $S_2$;
3) calculul sumelor $S_1, S_2, S_3, S_4$, ... continuă până când ultima din ele va fi formată exact dintr-o singură cifră. Anume ea este cifra de control.

De exemplu, pentru numărul $A = 2884299379$, obţinem:

$S_1 = 2 + 8 + 8 + 4 + 2 + 9 + 9 + 3 + 7 + 9 = 61$;

$S_2 = 6 + 1 = 7$;

$C = 7$.

În anumite cazuri, datele eronate citite de pe purtătorul de informaţie ar putea fi reconstituite. Pentru a face acest lucru, inginerii au nevoie să ştie câte din numerele naturale formate din *n* cifre au cifra de control egală cu *k*. Vom nota acest număr prin *M*.

De exemplu, în cazul numerelor naturale formate din $n = 2$ cifre, pentru cifra de control $k = 3$, există $M = 10$ astfel de numere.

**Sarcină.** Elaboraţi un program, care, cunoscând numerele *n* şi *k*, calculează numărul *M*.

**Date de intrare.** Intrarea standard conţine pe o singură linie numerele întregi *n* şi *k* separate prin spaţiu.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie numărul întreg *M*.

**Restricţii.** $1 \le n \le 6$; $1 \le k \le 9$. Timpul de execuţie nu va depăşi 0,1 secunde. Programul va folosi cel mult 1 Megaoctet de memorie operativă. Fişierul sursă va avea denumirea `CIFRA.PAS`, `CIFRA.C` sau `CIFRA.CPP`.

**Exemplu.**

| *Intrare* | *Ieşire* |
|---|---|
| `2 3` | `10` |

## Контрольная цифра

Известно, что при хранении данных на носителях информации могут возникать ошибки. Для обнаружения таких ошибок было предложено записывать на носителях информации не только сами натуральные числа, но и дополнительные, проверочные данные, в виде контрольных цифр. Как только с носителя информации считывается натуральное число, немедленно вычисляется его контрольная цифра. Если она не совпадает с контрольной цифрой, ранее сохраненной на этом же на носителе информации, считается, что возникла ошибка.

Контрольная цифра *C* натурального числа *A* вычисляется следующим образом:

4) суммируются все цифры исходного числа *A*; полученную сумму обозначают через $S_1$;
5) суммируются все цифры числа $S_1$; полученную сумму обозначают через$S_2$;
6) вычисление сумм $S_1, S_2, S_3, S_4$, ... продолжается до тех пор, пока последняя из сумм будет состоять ровно из одной цифры; именно эта цифра и является контрольной.

Например, для числа $A = 2884299379$, получаем:

$S_1 = 2 + 8 + 8 + 4 + 2 + 9 + 9 + 3 + 7 + 9 = 61$;

$S_2 = 6 + 1 = 7$;

$C = 7$.

В некоторых случаях, ошибки, содержащиеся в считываемых с носителей информации данных, могут быть исправлены. Для этого инженеры должны знать, сколько натуральных чисел, состоящих ровно из *n* цифр, имеют контрольную цифру равную *k*. Обозначим число таких чисел через *M*.

Например, в случае натуральных чисел состоящих из двух цифр ($n = 2$), при контрольной цифре $k = 3$, существуют 10 таких чисел. Следовательно, $M = 10$.

**Задание.** Напишите программу, которая, зная числа *n* и *k*, вычисляет число *M*.

**Входные данные.** Стандартный ввод содержит в единственной строке целые числа *n* и *k*, разделенные пробелом.

**Выходные данные.** Стандартный вывод должен содержать в единственной строке целое число *M*.

**Ограничения.** $1 \le n \le 6$; $1 \le k \le 9$. Время выполнения программы не должно превышать 0,1 секунды. Объем используемой оперативной памяти не должен превышать 1 Мегабайт. Исходный файл должен иметь имя `cifra.pas`, `cifra.c` или `cifra.cpp`.

**Пример.**

| *Ввод* | *Вывод* |
|---|---|
| `2 3` | `10` |

## Rezolvare

Algoritmul în care se calculează cifra de control pentru fiecare număr de *n* cifre, numărând pe cele care au această cifră egală cu *k* poate fi considerat „falimentar" din punctul de vedere al timpului de execuţie. Astfel de algoritmi de obicei se numesc *"algoritmi de triere".*

O varianta a unui astfel de algoritm poate fi exemplificat astfel.

**Exemplu.** Fie *n=2* şi *k=5.* Atunci numere din *2* cifre sunt de la *10* la *99,* si in total avem *99-10+1=90* numere. Sa se determine cate dintre aceste 90 de numere au cifra de control *k=5.*

Printr-o singura iteraţie:

*14,41,23,32,50*

Prin doua iteraţii( numerele(de la 10 la 99) suma cifrelor cărora va fi egala cu numerele(nu toate) din prima iteraţie):

59,95,68,86,77 (la prima iteraţie suma cifrelor este 14). Nu mai putem construi numere din 2 cifre suma cărora sa fie una dintre numerele: 41,23,32,50.

Astfel obţinem 10 numere.

Fie *n* =2, *k* = 1.

Printr-o singura iteraţie:

10.

Prin doua iteraţii (se determina acele numere din 2 cifre, a căror suma cifrelor este 10)

19 ,91 ,28, 82, 37, 73, 46, 64, 55.

Astfel obţinem 10 numere.

Astfel se poate construi o matrice de tipul celei de mai jos (pentru n=2), in care pe coloane se indica numerele cu cifra de control *k* si numărul de linii va indica cate numere de *n* cifre au cifra de control *k*

| k=1 | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 | k=8 | k=9 |
|---|---|---|---|---|---|---|---|---|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 |
| 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |

Algoritmul de triere necesita un timp de calcul foarte mare deja pentru n=7.

Din analiza exemplelor de mai sus se observa o alta abordare a probleme. Şi anume, analiza problemei conduce la o soluţie matematică: toate cifrele de control pentru numere succesive parcurg circular mulţimea *{1,2,..,9}*. Numerele de *n* cifre sunt cele aflate între $10^{n-1}$ şi $10^{n}-1$, cifra de control a numărului $10^{n-1}$ este *1,* a lui $10^{n}-1$este 9 şi există $[(10^{n}-1)-(10^{n-1})]+1= 10^{n}-10^{n-1} =10^{n-1} * 9$ astfel de numere; dintre acestea, $(10^{n-1} * 9)/9$ vor avea cifra de control egală cu ***k*** (indiferent cât ar fi acest ***k)*.**

Deci răspunsul este $10^{n-1}$, adică se tipăreşte un 1 urmat de *n*-1 cifre de 0. Cazul *n*=1 se analizează separat. Algoritmul funcţionează pentru *n* oricât de mare.

```
Program Cifra;
uses Crt;
var k:integer;
n,i: longint;
begin

  read(n,k);
```

```
 if n=1 then writeln(1)
 else begin
        write(1);
        for i:=1 to n-1 do write(0);
      end;
end.
```

# Enigma

*Enigma* este numele unei familii de maşini electromecanice utilizate pentru criptarea şi decriptarea de mesaje secrete. Maşina a căpătat notorietate deoarece în timpul celui de al Doilea Război Mondial criptologii ţărilor aliate au reuşit să decripteze un mare număr de mesaje care fuseseră cifrate cu această maşină. În procesul descifrării, criptologii au fost nevoiţi să soluţioneze următoarea problemă.

Se consideră şirurile de caractere $A$ şi $B$. Prin definiţie, şirul $A$ se află într-o relaţie de corespondenţă $R(A, B')$ cu şirul $B$ dacă:

1) în şirul $B$ există cel puţin un subşir $B'$ de aceiaşi lungime ca şi şirul $A$;
2) în şirurile $A$ şi $B'$ există cel puţin o poziţie $i$ pentru care caracterele $A[i]$ şi $B'[i]$ coincid.

Lungimea $L(A, B')$ a relaţiei de corspondenţă $R(A, B')$ se defineşte prin numărul de poziţii $i$ pentru care caracterele din şirurile $A$ şi $B'$ coincid.

În general, pentru oricare două şiruri $A$ şi $B$ pot să nu existe, poate exista una sau chiar şi mai multe relaţii de corespondenţă de tipul $R(A, B')$.

De exemplu, pentru şirurile $A$ = `'abaab'` şi $B$ = `'aababacab'` există următoarele relaţii de corespondenţă (în scopuri didactice, în şirul $B$, subşirurile $B'$ sunt evidenţiate prin subliniere, iar caracterelt ce coincid – prin dimeniuni mai mari):

**a**ab**ab**acab, $L = 3$;

a**aba**bacab, $L = 3$;

aabab**a**cab, $L = 1$;

aab**aba**cab, $L = 3$;

aababac**ab**, $L = 2$.

Gradul de corespondenţă $G$ a şirului $A$ cu şirul $B$ se defineşte ca suma lungimilor tuturor relaţiilor de corespondenţă de tipul $R(A, B')$. Dacă astfel de relaţii nu există, prin definiţie, $G = 0$.

În condiţiile exemplului de mai sus, $G = 3 + 3 + 1 + 3 + 2 = 12$.

**Sarcină.** Elaboraţi un program, care, cunoscând şirurile $A$ şi $B$, calculează gradul de corespondenţă $G$.

**Date de intrare.** Prima linie a intrării standard conţine şirul de caractere $A$. Linia a doua a fişierului de intrare conţine şirul de caractere $B$.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie numărul întreg $G$.

**Restricţii.** Şirurile $A$ şi $B$ sunt formate din literele mici ale alfabetului latin. Fiecare din şiruri conţine cel puţin una, dar nu mai multe de 5000 de litere. Lungimea şirului $A$ nu depăşeşte lungimea şirului $B$. Timpul de execuţie nu va depăşi 0,1 secundă. Programul va folosi cel mult 5 Megaocteţi de memorie operativă. Fişierul sursă va avea denumirea `enigma.pas`, `enigma.c` sau `enigma.cpp`.

**Exemplu.**

| *Intrare* | *Ieşire* |
|---|---|
| abaab<br>aababacab | 12 |

## Энигма

*Энигма* – это название семейства электромеханических машин, используемых для шифрования и дешифрования секретных сообщений. Машина получила известность благодаря тому, что во время Второй мировой войны, криптологи союзников смогли расшифровать большое количество сообщений, которые были зашифрованы с ее помощью. В процессе дешифрирования, криптологам пришлось решить следующую задачу.

Рассматриваются символьные строки $A$ и $B$. По определению, строка $A$ находится в отношении корреспондентности $R(A, B')$ со строкой $B$ если:

3) в строке $B$ существует хотя бы одна подстрока $B'$, длина которой равна длине строки $A$;
4) в строках $A$ и $B'$ существует по крайней мере одна позиция $i$, для которой символы $A[i]$ и $B'[i]$ совпадают.

По определению, длина $L(A, B')$ отношения корреспондентности $R(A, B')$ равно числу позиций $i$, для которых соответствующие символы из строк $A$ и $B'$ совпадают.

В общем случае, для двух произвольных строк символов $A$ şi $B$, могут не существовать, может существовать одна или могут существовать несколько отношений корреспондентности вида $R(A, B')$.

Например, для строк $A$ = `'abaab'` и $B$ = `'aababacab'` существуют следующие отношения корреспондентности (в строке $B$, для наглядности, подстроки $B'$ выделены подчеркиванием, а совпадающие символы – бóльшим размером):

**a**ab**ab**acab, $L = 3$;

a**aba**bacab, $L = 3$;

aabab**a**cab, $L = 1$;

aab**aba**cab, $L = 3$;

aababac**ab**, $L = 2$.

По определению, степень корреспондентности $G$ строки $A$ со строкой $B$ равна сумме длин всевозможных отношений корреспондентности вида $R(A, B')$. Если таких отношений не существует, по определению, $G = 0$.

В частности, для приведенного выше примера, $G = 3 + 3 + 1 + 3 + 2 = 12$.

**Задание.** Напишите программу, которая, зная строки $A$ и $B$, вычисляет степень корреспондентности $G$.

**Входные данные.** Первая строка стандартного ввода содержит строку символов $A$. Вторая строка стандартного ввода содержит строку символов $B$.

**Выходные данные.** Стандартный выход должен содержать в единственной строке целое число $G$.

**Ограничения.** Символьные строки $A$ и $B$ состоят из строчных букв латинского алфавита. Каждая строка может содержать от 1 до 5000 букв. Длина строки $A$ не превышает длину строки $B$. Время выполнения программы не должно превышать 0,1 секунды. Объем используемой оперативной памяти не должен превышать 5 Мегабайт. Исходный файл должен иметь имя `enigma.pas`, `enigma.c` или `enigma.cpp`.

**Пример.**

| *Ввод* | *Вывод* |
|---|---|
| abaab<br>aababacab | 12 |

## Rezolvare

Fie $n$ lungimea şirului $A$, iar $m$ lungimea şirului $B$. Observăm că există $m-n+1$ corespondenţe posibile. Presupunem că prima literă din şirul $A$ corespunde literei $i$ din şirul $B$. Ceea ce trebuie sa facem este să calculăm lungimea fiecărei corespondenţe, adică pentru fiecare $j = 1, 2, 3, \dots, n$, cazurile în care $A[j] = B[i + j - 1]$. Gradul de corespondenţă va fi suma tuturor lungimilor corespondenţelor.

**Soluţia naivă**

Pentru fiecare literă $i$ din şirul $B$ se consideră toate literele $j$ din şirul $A$. Dacă literele coincid, atunci se incrementează gradul de corespondenţă.

Această soluţie are complexitatea $O(nm)$ şi funcţionează pentru cazurile când $n$ şi $m$ sunt mai mici sau egali cu 5000.

Problema are şi o soluţie mult mai eficientă, însă ea este neobligatorie pentru elevii din clasele a 7-a – a 9-a.

**Soluţia optimizată**

În loc să calculăm lungimile fiecărei corespondenţe, vom calcula pentru fiecare poziţie $i$ din şirul $B$, numărul corespondenţelor în care litera $i$ din şirul $B$ coincide cu litera corespunzătoare din şirul $A$. Dacă nu, în condiţia în care pentru fiecare corespondenţă şirul $A$ trebuie să se încadreze în şirul $B$, această valoare va fi numărul de apariţii a literei $B[i]$ în şirul $A$.

Pentru a ţine cont de această condiţie pentru fiecare literă $i$ din şirul $B$, $i = 1, 2, 3, \dots, n-1$, vom considera primele $i$ litere ale şirului $A$, respectiv pentru litera $m-i+1$ a şirului $B$, $i = 1, 2, 3, \dots, n-1$, vom considera ultimele $\boldsymbol{i}$ poziţii ale şirului $A$.

Pentru a implementa eficient această abordare, vom considera un tablou unidimensional de tip caracter în care vom stoca numărul de apariţii a unui caracter în şirul $A$.

Algoritmul este schiţat mai jos:

1. Considerăm un tabel: *t['a'... 'z'] = (0, 0,..., 0)*
2. Pentru fiecare *i de la 1 la m*
   a. Daca *i<=n*, atunci *t[A[i]]:=t[A[i]]+1*;
   b. Grad := Grad + t[B[i]];
   c. Daca *i>=m-n+1* atunci *t[A[n+i-m]]:=t[A[n+i-m]]-1*

Această soluţie are complexitatea $O(m)$ şi funcţionează pentru cazurile când $n$ şi $m$ sunt mai mici sau egali cu 2 000 000.

Un alt aspect de care ar trebui ţinut cont este considerarea unui tip de 64 bit pentru G (gradul de corespondenţă).

**Soluţia Pascal**

```
Program Enigma;
var
  A, B : AnsiString;
  n, m, i : LongInt;
  G : Int64;
```

```
  t : array['a' .. 'z'] of LongInt;
  ch : Char;
begin
  ReadLn(A);
  ReadLn(B);
  n := Length(A);
  m := Length(B);
  for ch := 'a' to 'z' do
    t[ch] := 0;
  G := 0;
  for i := 1 to m do
  begin
    if i <= n then
      t[A[i]] := A[p[i]] + 1;
    G := G + t[B[i]];
    if i >= m - n + 1 then
      t[A[n - m + i]] := t[A[n - m + i]] - 1;
  end;
  WriteLn(G);
end.
```

# Puşculiţa

Puşculiţa reprezintă un recipient sau un dispozitiv special, destinat depozitării şi acumulării de monede. Puşculiţele tradiţionale reprezintă figurine din ceramică, goale în interior, în formă de animale, fructe, legume ş.a.m.d. Cele mai populare sunt puşculiţele în formă de purceluş. Deasupra ele au o fantă, în care pot fi aruncate monede.

În Republica Moldova se folosesc monede de valori 1, 5, 10, 25 şi 50 de bani. Greutatea monedei depinde de valoarea ei. În scopuri didactice se consideră că monedele au următoarele greutăţi:

- 1 ban — 1 gram;
- 5 bani — 2 grame;
- 10 bani — 3 grame;
- 25 bani — 4 grame;
- 50 de bani — 5 grame.

Este cunoscut faptul că greutatea monedelor din puşculiţă este de $G$ grame.

În general, pentru una şi aceiaşi greutate $G$, în puşculiţă ar putea fi diferite sume de bani. De exemplu, pentru $G = 13$, în puşculiţă ar putea fi:

- 13 monede de un ban, în suma totală de 13 bani;
- 5 monede de 5 bani şi 3 monede de 1 ban, în suma totală de 28 bani;
- 4 monede de 10 bani şi o monedă de 1 ban, în sumă totală de 41 bani ş.a.m.d.

Prin $M$ vom nota suma maximă de bani care ar putea să fie în puşculiţă în condiţiile în care greutatea monedelor din ea este egală cu $G$.

**Sarcină.** Elaboraţi un program, care, cunoscând greutatea monedelor din puşculiţă $G$, calculează suma maximă de bani $M$ care ar putea să fie în ea.

**Date de intrare.** Intrarea standard conţine pe o singură linie numărul întreg $G$ — greutatea monedelor din puşculiţă (în grame).

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie numărul întreg $M$ — suma maximă de bani care ar putea să fie în puşculiţă (în bani).

**Restricţii.** $1 \leq G \leq 1000$. Timpul de execuţie nu va depăşi 0,1 secunde. Programul va folosi cel mult 1 Megaoctet de memorie operativă. Fişierul sursă va avea denumirea `pusculita.pas`, `pusculita.c` sau `pusculita.cpp`.

**Exemplu.**

| *Intrare* | *Ieşire* |
|---|---|
| `13` | `110` |

## Копилка

Копилка представляет собой контейнер или специальное устройство для хранения и накопления монет. Как правило, это полые керамические статуэтки в форме животных, фруктов, овощей и т.п. Наиболее популярными являются копилки в форме поросенка. Сверху у такой копилки есть щель, в которую можно бросать монеты.

В Республике Молдова находятся в обращении монеты достоинством 1, 5, 10, 25 и 50 баней. Вес монеты определяется ее достоинством. В дидактических целях считается, что монеты имеют следующие веса:

- 1 бан — 1 грамм;
- 5 баней — 2 грамма;
- 10 баней — 3 грамма;
- 25 баней — 4 грамма;
- 50 баней — 5 граммов.

Известно, что вес содержащихся в копилке монет равен $G$ граммам.

В общем случае, для одного и того же веса $G$ в копилки могут быть различные суммы денег. Например, при $G = 13$, в копилке могут быть:

- 13 монет по одному бану, в сумме 13 баней;
- 5 монет по 5 баней и 3 монеты по одному бану, в сумме 28 баней;
- 4 монет по 10 баней и одна монета в один бан, в сумме 41 бан и т.д.

Через $M$ обозначим максимальную сумму денег, которая, при заданном весе монет $G$, может быть в копилке.

**Задание.** Напишите программу, которая, зная вес монет $G$, содержащихся в копилке, вычисляет максимальную сумму денег $M$, что может быть в ней.

**Входные данные.** Стандартный ввод содержит в единственной строке целое число $G$ — вес монет из копилки (в граммах).

**Выходные данные.** Стандартный вывод должен содержать в единственной строке целое число $M$ — максимальная сумма денег, что может быть в копилке (в баннах).

**Ограничения.** $1 \leq G \leq 1000$. Время выполнения программы не должно превышать 0,1 секунды. Объем используемой оперативной памяти не должен превышать 1 Мегабайт. Исходный файл должен иметь имя `pusculita.pas`, `pusculita.c` или `pusculita.cpp`.

**Пример.**

| *Ввод* | *Вывод* |
|---|---|
| 13 | 110 |

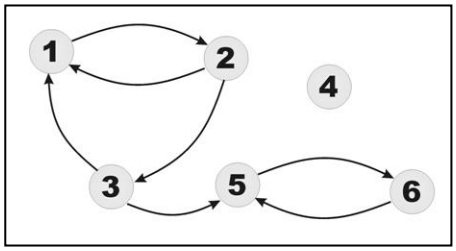
## Rezolvare

```
#include <iostream>

const size_t SIZE = 5;
typedef size_t monede_t[SIZE];
const size_t values[SIZE] = {1, 5, 10, 25, 50};


int main(int argc, char** argv) {
    size_t g, value = 0, count = SIZE;
```

```
    monede_t monede;
    std::cin >> g;

    while(count > 0){
        monede[count - 1] = g / count;
        g = g % count;
        --count;
    }

    for(count = 0; count != SIZE; ++count){
        value += monede[count] * values[count];
    }

    std::cout << value;
    return 0;
}
```

# Regiuni

Într-o ţară există $n$ localităţi, numerotate prin $1, 2, 3, \ldots, n$. Unele localităţi sunt legate între ele prin drumuri, notate prin $(i, j)$, însă circulaţia pe fiecare din aceste drumuri este permisă doar într-un singur sens, şi anume, de la localitatea $i$ la localitatea $j$ (vezi figura alăturată). Se consideră că orice drum $(i, j)$ nu trece prin alte localităţi.




Orice succesiune de drumuri $(i, k)$, $(k, m)$, ..., $(l, j)$, ce oferă posibilitatea de a ajunge regulamentar, conform sensurilor de circulaţie permise, din localitatea $i$ în localitatea $j$ se numeşte traseu şi se notează prin $(i, k, m, \ldots, l, j)$. Localităţile $i$, $j$ se numesc interconectate, dacă între ele există cel puţin câte un traseu, de la $i$ la $j$ şi de la $j$ la $i$. Prin definiţie, fiecare localitate este interconectată cu ea înseşi.

În exemplul din figura alăturată, localităţile 1 şi 2 sunt interconectate prin traseele (1, 2) şi (2, 1); localităţile 1 şi 3 – prin traseele (1, 2, 3) şi (3,1); localităţile 5 şi 6 – prin traseele (5, 6) şi (6, 5). Localitatea 4 nu este interconectată cu nici o altă localitate. Localitatea 3 nu este interconectată cu localitatea 5, chiar dacă există traseul (3, 5), deoarece nu există nici un traseu din localitatea 5 în localitatea 3.

Guvernul a decis să grupeze localităţile ţării în regiuni, fiecare regiune urmând să fie formată doar din localităţi ce sunt interconectate intre ele. Din raţiuni de eficienţă administrativă şi economică, numărul unor astfel de regiuni trebuie să fie cât mai mic.

**Sarcină.** Elaboraţi un program, care, cunoscând localităţile şi drumurile ce le leagă, calculează numărul minimal posibil de regiuni, fiecare regiune fiind formată doar din localităţi ce sunt interconectate.

**Date de intrare.** Intrarea standard conţine pe prima linie numărul întreg $n$. Fiecare din următoarele linii ale intrării standard conţine câte două numere întregi, separate prin spaţiu. Primul număr indică localitatea $i$, iar al doilea – localitatea $j$ a drumului $(i, j)$. Enumerarea drumurilor se termină cu ultima linie, care conţine două valori de 0, separate prin spaţiu.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie un număr întreg – numărul minimal posibil de regiuni.

**Restricţii.** $3 \leq n \leq 1000$. Timpul de execuţie nu va depăşi 0,5 secunde. Programul va folosi cel mult 16 Megaocteţi de memorie operativă. Fişierul sursă va avea denumirea `regiuni.pas`, `regiuni.c` sau `regiuni.cpp`.

**Exemplu 1** (vezi figura de mai sus)**.**

*Intrare*

```
6
1 2
2 1
2 3
3 5
3 1
5 6
6 5
0 0
```

*Ieşire*

```
3
```

**Exemplu 2.**

*Intrare*

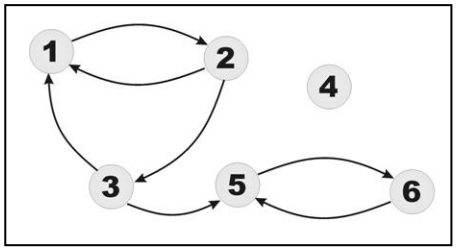
```
4
1 2
2 1
2 4
4 2
3 1
3 4
0 0
```

*Ieşire*

```
2
```

## Регионы

В некоторой стране есть $n$ населенных пунктов, пронумерованных через $1, 2, 3, \ldots, n$. Некоторые населенные пункты соединены дорогами, обозначаемые через $(i, j)$. Движение по каждой из этих дорог является односторонним, а именно, от населенного пункта $i$ к населенному пункту $j$. По определению, любая дорога $(i, j)$ не проходит ни через один другой населенный пункт.




Любая последовательность дорог $(i, k)$, $(k, m)$, ..., $(l, j)$, по которой, из населенного пункта $i$ можно доехать до населенного пункта $j$, соблюдая, естественно, правила дорожного движения, называется маршрутом и обозначается через $(i, k, m, \ldots, l, j)$. Населенные пункты $i, j$ называются взаимосвязанными, если существует хотя бы один маршрут от $i$ до $j$ и хотя бы один маршрут от $j$ до $i$. По определению, каждый населенный пункт взаимосвязан с самим собой.

В примере, показанном на рисунке, населенные пункты 1 и 2 взаимосвязаны, т.к. существуют маршруты (1, 2) и (2, 1). Населенные пункты 1 и 3 также взаимосвязаны, т.к. существуют маршруты (1, 2, 3) и (3,1). Населенные пункты 5 и 6 тоже взаимосвязаны, т.к. существуют маршруты (5, 6) и (6, 5). Населенный пункт 4 не взаимосвязан ни с одним другим населенным пунктом. Населенные пункты 3 и 5 не являются взаимосвязанными, поскольку, хотя и существует маршрут (3, 5), не существует ни одного маршрута от населенного пункта 5 к населенному пункту 3.

Правительство решило объединить населенные пункты страны в регионы. Каждый регион должен включать только населенные пункты, взаимосвязанные между собой. Из соображений административной эффективности, число регионов должно быть минимальным.

**Задание.** Напишите программу, которая, зная населенные пункты и дороги, связывающие их, вычисляет минимально возможное число регионов, состоящих из взаимосвязанных населенных пунктах.

**Входные данные.** Стандартный ввод содержит в первой строке целое число $n$. Каждая из следующих строк стандартного ввода содержит по два целых числа, разделенных пробелом. Первое число указывает населенный пункт $i$, а второе число ─ населенный пункт $j$ дороги $(i, j)$. Перечень дорог заканчивается строкой, содержащей два значения 0, разделенных пробелом.

**Выходные данные.** Стандартный вывод должен содержать в единственной строке одно целое число ─ минимально возможное число регионов.

**Ограничения.** $3 \leq n \leq 1000$. Время выполнения программы не должно превышать 0,5 секунды. Объем используемой оперативной памяти не должен превышать 16 Мегабайт. Исходный файл должен иметь имя `regiuni.pas`, `regiuni.c` или `regiuni.cpp`.

**Пример 1** (смотри рисунок).

*Ввод*

```
6
1 2
2 1
2 3
3 5
3 1
5 6
6 5
0 0
```

*Вывод*

```
3
```

**Пример 2.**

*Ввод*

```
4
1 2
2 1
2 4
4 2
3 1
3 4
0 0
```

*Вывод*

```
2
```

## Rezolvare

Rezolvarea se bazează pe câteva observaţii simple, şi anume:

1. Sistemul de drumuri şi localităţi poate fi considerat un graf orientat, reprezentat ca un tablou bidimensional, în care elementul a[i,j] indică prezenţa arcului (i,j) prin valoarea 1 sau absenţa acestuia prin valoarea 0.
2. Dacă se schimbă direcţiile tuturor arcelor în opus, grupurile de vârfuri interconectate rămân aceleaşi.
3. Pentru a determina localităţile în care se poate ajunge dintr-o localitate dată este suficient să se parcurgă recursiv toate drumurile posibile, pornind de la această localitate. (Parcurgere DFS)

Din cele spuse mai sus rezultă următorul algoritm:

Pas 1. Se construieşte graful cu arcele inversate $G^T$

Pas 2. Se lansează căutarea în adâncime (DFS) pornind de la fiecare vârf necercetat din $G^T$. Pentru fiecare parcurgere se memorează cumulativ ordinea de cercetare a vârfurilor în vectorul *black*.

Pas 3. Se lansează căutarea în adâncime (DFS) pe graful iniţial *G*, consecutiv, pornind de la ultimul vârf inclus în *black* către primul, după vârfurile necercetate.

Pas 4. La fiecare căutare în adâncime realizată în pasul 3, se marchează vârfurile cercetate – acestea formează un grup maximal de localităţi interconectate.

De remarcat, că la pasul 2 căutarea vârfurilor care pot fi atinse poate fi efectuată nu pe graful $G^T$ ci pe *G*. Atunci la pasul 3 căutarea va fi făcută pe $G^T$.

```
type    matrice_drumuri = array[0..1010,0..1010] of integer;
        stari_localitati = array[0..1010] of integer;

var
a, at : matrice_drumuri;
b, black : stari_localitati;
i, j, n, s, k : integer;
f, g : text;

procedure DFS_DIR (s: integer);
{ parcurgerea pe sistemul initial de drumuri}
var
   i : integer;
begin
   b[s] := 1;
   for i := 1 to n do
        if (a[s,i] <>0) AND (b[i] = 0) then DFS_DIR(i);
end;

procedure DFS_TRANS (s : integer);
{ parcurgerea pe sistemul inversat de drumuri}
var i : integer;
begin
   b[s] := 1;
   for i :=1 to n do
        if (at[s,i] <> 0) AND (b[i] =0) then DFS_TRANS(i);
   k := k + 1;
   {memorarea cumulativa in tabloul black a ordinii de cercetare}
   black[k] := s;
end;

begin
```

```
    assign (f, 'uni_00.in'); reset(f);
    assign (g, 'uni_00.out'); rewrite(g);
    readln(f, n);
    repeat
      readln(f, i, j);
      a[i,j] := 1;
      at[j,i] := 1;
    until(i = 0 );
    close(f);
    k :=0;
{ Cautarea in adancime at pe sistemul transpus }
    for i := 1 to n do
 if (b[i] = 0)  then DFS_TRANS(i);

{ resetarea starii varfurilor }
 for i := 1 to n do b[i] := 0;
    k := 0;

{ parcurgerea in adancime pe a / sistemul initial de conexiuni, pornind de
la ultimul nod cercetat ]n sistemul inversat }

    for i := n downto 1 do
 if  b[black[i]] = 0 then
        begin
            k :=k +1;
            DFS_DIR(black[i]);
        end;

    writeln(g, k);
    close(g);
end.
```

# Sistemul factorial de numeraţie

Este cunoscut faptul că numerele naturale pot fi reprezentate în diferite sisteme de numeraţie. De exemplu, în sistemul binar, numărul zecimal 37 se reprezintă în felul următor:

$$37_{10} = 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 100101_2.$$

Mai puţini însă cunosc faptul că numerele naturale pot fi reprezentate şi în sisitemul factorial, de exemplu:

$$37_{10} = 1 \cdot 4! + 2 \cdot 3! + 0 \cdot 2! + 1 \cdot 1! = 1201_!;$$

$$463_{10} = 3 \cdot 5! + 4 \cdot 4! + 1 \cdot 3! + 0 \cdot 2! + 1 \cdot 1! = 34101_!.$$

Forma generală a acestei reprezentări este:

$$a = d_n \cdot n! + d_{n-1} \cdot (n-1)! + \cdots + d_i \cdot i! + \cdots + d_2 \cdot 2! + d_1 \cdot 1!,$$

unde $d_n, d_{n-1}, \ldots, d_i, \ldots, d_2, d_1$ sunt numere naturale, $d_n > 0$ şi $d_i \leq i$.

**Sarcină.** Elaboraţi un program, care, cunoscând numărul natural $a$, calculează reprezentarea acestuia în sistemul factorial.

**Date de intrare.** Intrarea standard conţine pe o singură linie numărul întreg $a$.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie numerele întregi $d_n, d_{n-1}, \ldots, d_i, \ldots, d_2, d_1$, separate prin spaţiu.

**Restricţii.** $1 \leq a \leq 10^{18}$. Timpul de execuţie nu va depăşi 0,2 secunde. Programul va folosi cel mult 1 Megaoctet de memorie operativă. Fişierul sursă va avea denumirea `factor.pas`, `factor.c` sau `factor.cpp`.

**Exemplu.**

| *Intrare* | *Ieşire* |
|---|---|
| `463` | `3 4 1 0 1` |

## Факториальная система счисления

Известно, что натуральные числа могут быть представлены в различных системах нумерации. Например, в двоичной системе, десятичное число 37 представляется следующим образом:

$$37_{10} = 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 100101_2.$$

Однако не все знают, что натуральные числа могут быть представлены и в факториальной системе счисления, например:

$$37_{10} = 1 \cdot 4! + 2 \cdot 3! + 0 \cdot 2! + 1 \cdot 1! = 1201_!;$$

$$463_{10} = 3 \cdot 5! + 4 \cdot 4! + 1 \cdot 3! + 0 \cdot 2! + 1 \cdot 1! = 34101_!.$$

Общий вид этого представления:

$$a = d_n \cdot n! + d_{n-1} \cdot (n-1)! + \cdots + d_i \cdot i! + \cdots + d_2 \cdot 2! + d_1 \cdot 1!,$$

где $d_n, d_{n-1}, \ldots, d_i, \ldots, d_2, d_1$ натуральные числа, $d_n > 0$ и $d_i \leq i$.

**Задание.** Напишите программу, которая, зная натуральное число $a$, вычисляет его представление в факториальной системе счисления.

**Входные данные.** Стандартный ввод содержит в единственной строке целое число $a$.

**Выходные данные.** Стандартный вывод должен содержать в единственной строке целые числа $d_n, d_{n-1}, \ldots, d_i, \ldots, d_2, d_1$, разделенные пробелом.

**Ограничения.** $1 \leq a \leq 10^{18}$ Время выполнения программы не должно превышать 0,2 секунды. Объем используемой оперативной памяти не должен превышать 1 Мегабайт. Исходный файл должен иметь имя `factor.pas`, `factor.c` или `factor.cpp`.

**Пример.**

| *Ввод* | *Вывод* |
|---|---|
| `463` | `3 4 1 0 1` |

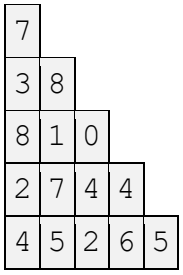
**Rezolvare**

```
#include <iostream>
#include <deque>

typedef unsigned long long ULL;

constexpr ULL factorial(ULL p){
    return  (p == 1) ? 1 : p * factorial(p - 1);
}

const ULL base[] = {
    factorial(1ull),
    factorial(2ull),
    factorial(3ull),
    factorial(4ull),
    factorial(5ull),
    factorial(6ull),
    factorial(7ull),
    factorial(8ull),
    factorial(9ull),
    factorial(10ull),
    factorial(11ull),
    factorial(12ull),
    factorial(13ull),
    factorial(14ull),
    factorial(15ull),
    factorial(16ull),
    factorial(17ull),
    factorial(18ull),
    factorial(19ull),
    factorial(20ull)
};

int main(int argc, char** argv) {
    ULL num;
    int top = sizeof(base) / sizeof(ULL) - 1;
    std::deque<ULL> result;

    std::cin >> num;

    while(num < base[top]){
       --top;
    }

    while(num > 0){
        result.push_back(num / base[top]);
        num = num % base[top];
        -- top;
    }
   if(top+1 > 0) { result.insert(result.end(), top+1, 0);}

    for(auto value : result){
        std::cout << value << " ";
    }
    return 0;
}
```

```
Program Factor;
var values: array [1..20] of integer;
   n: Int64;
   i: Integer;

begin
   for i := 1 to 20 do values[i] := 0;
   i := 2;
   readln(n);
   while n > 0 do
   begin
         values[i - 1] := n mod i;
           n := n div i;
         i := i + 1;
   end;
   while i > 2 do
   begin
         i := i - 1;
         write(values[i - 1], ' ');
   end;
end.
```

# Triunghiuri numerice

Se consideră un triunghi format din numere naturale, în care pe prima line apare un număr, pe a linia doua apar două numere, pe linia a treia apar trei numere ş.a.m.d. Pentru exemplificare, în figura alăturată este prezentat un triunghi format din cinci linii.




Numărul de linii ale triunghiului numeric se notează prin *n*.

Prin drum vom înţelege o secvenţă de numere $(a_1, a_2, \dots, a_i, \dots, a_n)$, formată după cum urmează:

- $a_1$ este numărul din vârful de sus al triunghiului;
- succesor al numărului $a_i$ poate fi doar unul din numerele care se află pe linia de mai jos al triunghiului, şi anume, fie cel imediat dedesubt, fie cel de pe diagonala din dreapta;
- numărul $a_n$ aparţine liniei de bază a triunghiului numeric.

Lungimea drumului se defineşte ca suma numerelor din secvenţa în cauză:

$$L = a_1 + a_2 + \cdots + a_i + \cdots + a_n.$$

Exemple de drumuri:

(7, 3, 1, 7, 5); $L = 23$;

(7, 8, 1, 4, 6); $L = 26$;

(7, 8, 0, 4, 5); $L = 24$.

Prin $L_{min}$ vom nota lungimea celui mai scurt, iar prin $L_{max}$ lungimea celui mai lung drum din triunghiul numeric.

**Sarcină.** Elaboraţi un program, care, cunoscând triunghiul numeric, calculează lungimile celui mai scurt şi celui mai lung din drumuri.

**Date de intrare.** Intrarea standard conţine pe prima linie numărul întreg *n*. Următoarele *n* linii ale intrării standard reprezintă liniile triunghiului numeric, de la vârf la bază. Linia $i + 1$ a intrării standard conţine *i* numere întregi separate prin spaţiu.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie numerele întregi $L_{min}$ şi $L_{max}$ separate prin spaţiu.

**Restricţii.** $1 \leq n \leq 50$. Numerele care apar în triunghiul numeric sunt naturale şi au valori mai mici ca $10^7$. Timpul de execuţie nu va depăşi 0,5 secunde. Programul va folosi cel mult 32 Megaocteţi de memorie operativă. Fişierul sursă va avea denumirea `triunghi.pas`, `triunghi.c` sau `triunghi.cpp`.

**Exemplu.**

*Intrare*

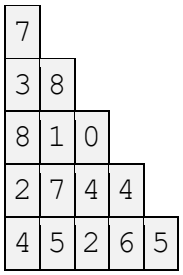
```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

*Ieşire*

```
17 30
```

## Triunghiuri numerice / Числовые треугольники

Рассматриваются числовые треугольники, состоящие из натуральных чисел. В таких треугольниках первая строка содержит одно число, вторая строка – два числа, третья строка – три числа и т.д. В качестве примера, на рисунке представлен числовой треугольник, состоящий из пяти строк.




Число строк треугольника обозначается через *n*.

По определению, путем называется последовательность чисел вида $(a_1, a_2, \dots, a_i, \dots, a_n)$, получаемая следующим образом:

- $a_1$ является числом из первой, верхней строки треугольника;
- последователем числа $a_i$ может быть одно из двух чисел, находящихся в строке снизу, а именно, либо под ним, либо вправо по диагонали;
- число $a_n$ находится в последней строке, являющейся основанием числового треугольника.

Длина пути определяется как сумма чисел рассматриваемой последовательности:

$$L = a_1 + a_2 + \cdots + a_i + \cdots + a_n.$$

Примеры путей:

(7, 3, 1, 7, 5); $L = 23$;
(7, 8, 1, 4, 6); $L = 26$;
(7, 8, 0, 4, 5); $L = 24$.

Через $L_{min}$ обозначим длину самого короткого, а через $L_{max}$ длину самого длинного пути в числовом треугольнике.

**Задание.** Напишите программу, которая, зная числовой треугольник, вычисляет длину самого короткого и длину самого длинного пути.

**Входные данные.** Стандартный ввод содержит в первой строке целое число *n*. Следующие *n* строк стандартного входа содержат строки числового треугольника, от вершины к основанию. Строка $i + 1$ стандартного ввода содержит $i$ целых чисел, разделенных пробелом.

**Выходные данные.** Стандартный вывод должен содержать в единственной строке целые числа $L_{min}$ и $L_{max}$, разделенные пробелом.

**Ограничения.** $1 \leq n \leq 50$. Числовой треугольник состоит из натуральных чисел, меньших, чем $10^7$. Время выполнения программы не должно превышать 0,5 секунды. Объем используемой оперативной памяти не должен превышать 32 Мегабайт. Исходный файл должен иметь имя `triunghi.pas`, `triunghi.c` или `triunghi.cpp`.

**Exemplu.**

*Intrare*

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

*Ieşire*

```
17 30
```

## Rezolvare

Problema se rezolvă prin metoda programării dinamice, folosind în acest scop o matrice triunghi în care reţinem datele iniţiale şi - pe baza ei – se construieşte o altă matrice triunghiulară *T* cu următoarea semnificaţie: *T[i,j]* = suma maxima (minimală) pentru cel mai scurt drum care pleacă de pe o poziţie de pe ultima linie şi ajunge pe poziţia *(i, j)* în matricea iniţială. Relaţiile de recurenţă sunt:

a) Pentru drumul de lungime maximală

```
T[i,j]=max{A[i,j]+T[i+1,j+1], A[i,j]+T[i+1,j]}.
```

a) Pentru drumul de lungime minimală

```
T[i,j]=min{A[i,j]+T[i+1,j+1], A[i,j]+T[i+1,j]}.
```

Metoda de programare dinamică folosită este varianta *înainte.*

**Exemplu**:

| Drum maxim | Drum minim |
|---|---|
| **7**<br>**3** 8<br>**8** 1 0<br>2 **7** 4 4<br>4 **5** 2 6 5 | **7**<br>**3** 8<br>8 1 0<br>2 7 4 4<br>4 5 2 6 5 |

Atunci răspunsul este: valoarea max=30, min=17.

Pentru exemplu de mai sus vom ob'ine:
a) Pentru drumul de lungime maximala
*2 7 4 4 rândul A[4,j=1,2,3,4]*
*4 5 2 6 5 rândul T[5,j=1,...,5]*
T[4,j=1]=max{2+4,2+5}=**7**;
T[4,j=2]=max{7+5,7+2}=**12**,
T[4,j=3]= max{4+2,4+6}=**10**,
T[4,j=4]= max{4+6,4+5}=**10.**
---
*8 1 0 rândul A[3,j=1,2,3]*
*7 12 10 10 rândul T[4,j=1,...,4]*
T[3,j=1]=max{8+7,8+12}=**20**;
T[3,j=2]=max{1+12,1+10}=**13**;
T[3,j=3]= max{0+10,0+10}=**10**.
---
*3 8 rândul A[2,j=1,2]*
*20 13 10 rândul T[3,j=1,2,3]*
*T[2,j=1]=max{ 3+20,3+13}=**23**,*
*T[2,j=2]=max{8+13,8+12}=**21***
---
*7 rândul A[1,1]*
*23 21 rândul T[2,j=1,2]*
*T[1,j=1]=max{7+23,7+21}=30*

Matricea *T* are forma: $\begin{pmatrix} 30 & & & & \\ 23 & 21 & & & \\ 20 & 13 & 10 & & \\ 7 & 12 & 10 & 10 & \\ 4 & 5 & 2 & 6 & 5 \end{pmatrix}$

Complexitate timp al algoritmului este $O(n^2)$ unde n este numărul de rânduri.

```
Program Triunghi;
{ Clasele 07-09 }
uses crt;
const nmax=100;
type tab=array[1..nmax, 1..nmax] of longint;
var tmax,tmin,a:tab;
i,j,n:integer;

function max(a,b:longint):longint;
begin
if a>b then max:=a else max:=b;
end;

function min(a,b:longint):longint;
begin
if a<b then min:=a else min:=b;
end;

procedure dinamic;
begin
for i:=1 to n do
begin
tmax[n,i]:=a[n,i];
tmin[n,i]:=a[n,i];
end;
for i:=n-1 downto 1 do
begin
for j:=1 to i do begin
tmax[i,j]:=max(a[i,j]+tmax[i+1,j],a[i,j]+tmax[i+1,j+1]);
tmin[i,j]:=min(a[i,j]+tmin[i+1,j],a[i,j]+tmin[i+1,j+1]);
                end;
end;
end;

begin
readln(n);{se citeste numarul de randuri}
{se citesc randurile matricei din fisier}
for i:=1 to n do
begin
for j:=1 to i do read(a[i,j]);
end;
dinamic;
{==
for i:=1 to n do
begin
for j:=1 to i do write(fout,' ',tmax[i,j],'');
writeln(fout);
end;
for i:=1 to n do
begin
for j:=1 to i do write(fout,' ',tmin[i,j],'');
```

```
writeln(fout);
end;
==}
writeln(tmin[1,1],' ',tmax[1,1]);
end.
```

# Clasele 10 – 12

## Cifra de control

Este cunoscut faptul că datele de pe purtătorii de informaţie pot fi compromise. În scopul depistării eventualelor erori, s-a decis ca fiecare număr natural de pe oricare purtător de informaţie să fie însoţit şi de o cifră de control. Imediat cum numărul este citit de pe purtătorul de informaţie, se calculează cifra lui de control. Dacă ea nu coincide cu cifra de control stocată anterior pe purtător, se consideră că datele de pe purtătorul de informaţie sunt compromise.

Cifră de control *C* a numărului natural *A* se calculează după cum urmează:

7) se însumează toate cifrele numărului *A*, numărul obţinut fiind notat prin $S_1$;
8) se însumează toate cifrele numărului $S_1$, numărul obţinut fiind notat prin $S_2$;
9) calculul sumelor $S_1, S_2, S_3, S_4$, ... continuă până când ultima din ele va fi formată exact dintr-o singură cifră. Anume ea este cifra de control.

De exemplu, pentru numărul $A = 2884299379$, obţinem:

$S_1 = 2 + 8 + 8 + 4 + 2 + 9 + 9 + 3 + 7 + 9 = 61$;

$S_2 = 6 + 1 = 7$;

$C = 7$.

În anumite cazuri, datele eronate citite de pe purtătorul de informaţie ar putea fi reconstituite. Pentru a face acest lucru, inginerii au nevoie să ştie câte din numerele naturale formate din *n* cifre au cifra de control egală cu *k*. Vom nota acest număr prin *M*.

De exemplu, în cazul numerelor naturale formate din $n = 2$ cifre, pentru cifra de control $k = 3$, există $M = 10$ astfel de numere.

**Sarcină.** Elaboraţi un program, care, cunoscând numerele *n* şi *k*, calculează numărul *M*.

**Date de intrare.** Intrarea standard conţine pe o singură linie numerele întregi *n* şi *k* separate prin spaţiu.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie numărul întreg *M*.

**Restricţii.** $1 \le n \le 10000$; $1 \le k \le 9$. Timpul de execuţie nu va depăşi 0,1 secunde. Programul va folosi cel mult 1 Megaoctet de memorie operativă. Fişierul sursă va avea denumirea `cifra.pas`, `cifra.c` sau `cifra.cpp`.

**Exemplu.**

| *Intrare* | *Ieşire* |
|---|---|
| `2 3` | `10` |

## Контрольная цифра

Известно, что при хранении данных на носителях информации могут возникать ошибки. Для обнаружения таких ошибок было предложено записывать на носителях информации не только сами натуральные числа, но и дополнительные, проверочные данные, в виде контрольных цифр. Как только с носителя информации считывается натуральное число, немедленно вычисляется его контрольная цифра. Если она не совпадает с контрольной цифрой, ранее сохраненной на этом же на носителе информации, считается, что возникла ошибка.

Контрольная цифра *C* натурального числа *A* вычисляется следующим образом:

10) суммируются все цифры исходного числа *A*; полученную сумму обозначают через $S_1$;
11) суммируются все цифры числа $S_1$; полученную сумму обозначают через$S_2$;
12) вычисление сумм $S_1, S_2, S_3, S_4$, ... продолжается до тех пор, пока последняя из сумм будет состоять ровно из одной цифры; именно эта цифра и является контрольной.

Например, для числа $A = 2884299379$, получаем:

$S_1 = 2 + 8 + 8 + 4 + 2 + 9 + 9 + 3 + 7 + 9 = 61$;

$S_2 = 6 + 1 = 7$;

$C = 7$.

В некоторых случаях, ошибки, содержащиеся в считываемых с носителей информации данных, могут быть исправлены. Для этого инженеры должны знать, сколько натуральных чисел, состоящих ровно из $n$ цифр, имеют контрольную цифру равную $k$. Обозначим число таких чисел через *M*.

Например, в случае натуральных чисел состоящих из двух цифр ($n = 2$), при контрольной цифре $k = 3$, существуют 10 таких чисел. Следовательно, $M = 3$.

**Задание.** Напишите программу, которая, зная числа *n* и *k*, вычисляет число *M*.

**Входные данные.** Стандартный ввод содержит в единственной строке целые числа *n* и *k*, разделенные пробелом.

**Выходные данные.** Стандартный вывод должен содержать в единственной строке целое число *M*.

**Ограничения.** $1 \le n \le 10000$; $1 \le k \le 9$. Время выполнения программы не должно превышать 0,1 секунды. Объем используемой оперативной памяти не должен превышать 1 Мегабайт. Исходный файл должен иметь имя `cifra.pas`, `cifra.c` или `cifra.cpp`.

**Пример.**

| *Ввод* | *Вывод* |
|---|---|
| 2 3 | 10 |

## Rezolvare

Algoritmul în care se calculează cifra de control pentru fiecare număr de *n* cifre, numărând pe cele care au această cifră egală cu *k* poate fi considerat „falimentar" din punctul de vedere al timpului de execuţie. Astfel de algoritmi de obicei se numesc *"algoritmi de triere".*
O varianta a unui astfel de algoritm poate fi exemplificat astfel.

**Exemplu.** Fie *n=2* şi *k=5.* Atunci numere din *2* cifre sunt de la *10* la *99,* si in total avem *99-10+1=90* numere. Sa se determine cate dintre aceste 90 de numere au cifra de control *k=5.*

Printr-o singura iteraţie:

*14,41,23,32,50*

Prin doua iteraţii( numerele(de la 10 la 99) suma cifrelor cărora va fi egala cu numerele(nu toate) din prima iteraţie):

59,95,68,86,77 (la prima iteraţie suma cifrelor este 14). Nu mai putem construi numere din 2 cifre suma cărora sa fie una dintre numerele: 41,23,32,50.

Astfel obţinem 10 numere.

Fie *n* =2, *k* = 1.

Printr-o singura iteraţie:

10.

Prin doua iteraţii (se determina acele numere din 2 cifre, a căror suma cifrelor este 10)

19 ,91 ,28, 82, 37, 73, 46, 64, 55.

Astfel obţinem 10 numere.

Astfel se poate construi o matrice de tipul celei de mai jos (pentru n=2), in care pe coloane se indica numerele cu cifra de control *k* si numărul de linii va indica cate numere de *n* cifre au cifra de control *k:*

| k=1 | k=2 | k=3 | k=4 | k=5 | k=6 | k=7 | k=8 | k=9 |
|---|---|---|---|---|---|---|---|---|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 |
| 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 |

Algoritmul de triere necesita un timp de calcul foarte mare deja pentru n=7.

Din analiza exemplelor de mai sus se observa o alta abordare a probleme. Şi anume, analiza problemei conduce la o soluţie matematică: toate cifrele de control pentru numere succesive parcurg circular mulţimea *{1,2,..,9}.* Numerele de *n* cifre sunt cele aflate între $10^{n-1}$ şi $10^{n}-1$, cifra de control a numărului $10^{n-1}$ este *1,* a lui $10^{n}-1$este 9 şi există $[(10^{n}-1)-(10^{n-1})]+1= 10^{n}-10^{n-1} =10^{n-1} * 9$ astfel de numere; dintre acestea, $(10^{n-1} * 9)/9$ vor avea cifra de control egală cu ***k*** (indiferent cât ar fi acest ***k)*.**

Deci răspunsul este $10^{n-1}$, adică se tipăreşte un 1 urmat de *n*-1 cifre de 0. Cazul *n*=1 se analizează separat. Algoritmul funcţionează pentru *n* oricât de mare.

```
Program Cifra;
uses Crt;
```

```
var k:integer;
n,i: longint;
begin

  read(n,k);
  if n=1 then writeln(1)
  else begin
         write(1);
         for i:=1 to n-1 do write(0);
       end;
end.
```

# Enigma

*Enigma* este numele unei familii de maşini electromecanice utilizate pentru criptarea şi decriptarea de mesaje secrete. Maşina a căpătat notorietate deoarece în timpul celui de al Doilea Război Mondial criptologii ţărilor aliate au reuşit să decripteze un mare număr de mesaje care fuseseră cifrate cu această maşină. În procesul descifrării, criptologii au fost nevoiţi să soluţioneze următoarea problemă.

Se consideră şirurile de caractere $A$ şi $B$. Prin definiţie, şirul $A$ se află într-o relaţie de corespondenţă $R(A, B')$ cu şirul $B$ dacă:

5) în şirul $B$ există cel puţin un subşir $B'$ de aceiaşi lungime ca şi şirul $A$;
6) în şirurile $A$ şi $B'$ există cel puţin o poziţie $i$ pentru care caracterele $A[i]$ şi $B'[i]$ coincid.

Lungimea $L(A, B')$ a relaţiei de corspondenţă $R(A, B')$ se defineşte prin numărul de poziţii $i$ pentru care caracterele din şirurile $A$ şi $B'$ coincid.

În general, pentru oricare două şiruri $A$ şi $B$ pot să nu existe, poate exista una sau chiar şi mai multe relaţii de corespondenţă de tipul $R(A, B')$.

De exemplu, pentru şirurile $A$ = `'abaab'` şi $B$ = `'aababacab'` există următoarele relaţii de corespondenţă (în scopuri didactice, în şirul $B$, subşirurile $B'$ sunt evidenţiate prin subliniere, iar caracterelt ce coincid – prin dimeniuni mai mari):

**a**ab**ab**acab, $L = 3$;

a**aba**bacab, $L = 3$;

aabab**a**cab, $L = 1$;

aab**aba**cab, $L = 3$;

aababac**ab**, $L = 2$.

Gradul de corespondenţă $G$ a şirului $A$ cu şirul $B$ se defineşte ca suma lungimilor tuturor relaţiilor de corespondenţă de tipul $R(A, B')$. Dacă astfel de relaţii nu există, prin definiţie, $G = 0$.

În condiţiile exemplului de mai sus, $G = 3 + 3 + 1 + 3 + 2 = 12$.

**Sarcină.** Elaboraţi un program, care, cunoscând şirurile $A$ şi $B$, calculează gradul de corespondenţă $G$.

**Date de intrare.** Prima linie a intrării standard conţine şirul de caractere $A$. Linia a doua a fişierului de intrare conţine şirul de caractere $B$.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie numărul întreg $G$.

**Restricţii.** Şirurile $A$ şi $B$ sunt formate din literele mici ale alfabetului latin. Fiecare din şiruri conţine cel puţin una, dar nu mai multe de 2 000 000 de litere. Lungimea şirului $A$ nu depăşeşte lungimea şirului $B$. Timpul de execuţie nu va depăşi 0,1 secundă. Programul va folosi cel mult 5 Megaocteţi de memorie operativă. Fişierul sursă va avea denumirea `enigma.pas`, `enigma.c` sau `enigma.cpp`.

**Exemplu.**

| *Intrare* | *Ieşire* |
|---|---|
| abaab<br>aababacab | 12 |

## Энигма

*Энигма* – это название семейства электромеханических машин, используемых для шифрования и дешифрования секретных сообщений. Машина получила известность благодаря тому, что во время Второй мировой войны, криптологи союзников смогли расшифровать большое количество сообщений, которые были зашифрованы с ее помощью. В процессе дешифрирования, криптологам пришлось решить следующую задачу.

Рассматриваются символьные строки $A$ и $B$. По определению, строка $A$ находится в отношении корреспондентности $R(A, B')$ со строкой $B$ если:

1) в строке $B$ существует хотя бы одна подстрока $B'$, длина которой равна длине строки $A$;
2) в строках $A$ и $B'$ существует по крайней мере одна позиция $i$, для которой символы $A[i]$ и $B'[i]$ совпадают.

По определению, длина $L(A, B')$ отношения корреспондентности $R(A, B')$ равно числу позиций $i$, для которых соответствующие символы из строк $A$ и $B'$ совпадают.

В общем случае, для двух произвольных строк символов $A$ şi $B$, могут не существовать, может существовать одна или могут существовать несколько отношений корреспондентности вида $R(A, B')$.

Например, для строк $A$ = `'abaab'` и $B$ = `'aababacab'` существуют следующие отношения корреспондентности (в строке $B$, для наглядности, подстроки $B'$ выделены подчеркиванием, а совпадающие символы – бóльшим размером):

**a**ab**ab**acab, $L = 3$;

a**aba**bacab, $L = 3$;

aabab**a**cab, $L = 1$;

aab**aba**cab, $L = 3$;

aababac**ab**, $L = 2$.

По определению, степень корреспондентности $G$ строки $A$ со строкой $B$ равна сумме длин всевозможных отношений корреспондентности вида $R(A, B')$. Если таких отношений не существует, по определению, $G = 0$.

В частности, для приведенного выше примера, $G = 3 + 3 + 1 + 3 + 2 = 12$.

**Задание.** Напишите программу, которая, зная строки $A$ и $B$, вычисляет степень корреспондентности $G$.

**Входные данные.** Первая строка стандартного ввода содержит строку символов $A$. Вторая строка стандартного ввода содержит строку символов $B$.

**Выходные данные.** Стандартный выход должен содержать в единственной строке целое число $G$.

**Ограничения.** Символьные строки $A$ и $B$ состоят из строчных букв латинского алфавита. Каждая строка может содержать от 1 до 2 000 000 букв. Длина строки $A$ не превышает длину строки $B$. Время выполнения программы не должно превышать 0,1 секунды. Объем используемой оперативной памяти не должен превышать 5 Мегабайт. Исходный файл должен иметь имя `enigma.pas`, `enigma.c` или `enigma.cpp`.

**Пример.**

| *Ввод* | *Вывод* |
|---|---|
| abaab<br>aababacab | 12 |

## Rezolvare

Fie $n$ lungimea şirului $A$, iar $m$ lungimea şirului $B$. Observăm că există $m-n+1$ corespondenţe posibile. Presupunem că prima literă din şirul $A$ corespunde literei $i$ din şirul $B$. Ceea ce trebuie sa facem este să calculăm lungimea fiecărei corespondenţe, adică pentru fiecare $j = 1, 2, 3, \dots, n$, cazurile în care $A[j] = B[i+j-1]$. Gradul de corespondenţă va fi suma tuturor lungimilor corespondenţelor.

**Soluţia naivă**

Pentru fiecare literă $i$ din şirul $B$ se consideră toate literele $\boldsymbol{j}$ din şirul $A$. Dacă literele coincid, atunci se incrementează gradul de corespondenţă.

Această soluţie are complexitatea $O(nm)$ şi funcţionează doar în cazul restricţiilor pentru clasele de gimnaziu, adică în cazurile în care $n$ şi $m$ nu depăşesc valoarea 5 000.

**Soluţia optimizată**

În loc să calculăm lungimile fiecărei corespondenţe, vom calcula pentru fiecare poziţie $i$ din şirul $B$, numărul corespondenţelor în care litera $i$ din şirul $B$ coincide cu litera corespunzătoare din şirul $A$. Dacă nu, în condiţia în care pentru fiecare corespondenţă şirul $A$ trebuie să se încadreze în şirul $B$, această valoare va fi numărul de apariţii a literei $B[i]$ în şirul $A$.

Pentru a ţine cont de această condiţie pentru fiecare literă $i$ din şirul $B$, $i = 1, 2, 3, \dots, n-1$, vom considera primele $i$ litere ale şirului $A$, respectiv pentru litera $m-i+1$ a şirului $B$, $i = 1, 2, 3, \dots, n-1$, vom considera ultimele $\boldsymbol{i}$ poziţii ale şirului $A$.

Pentru a implementa eficient această abordare, vom considera un tablou unidimensional de tip caracter în care vom stoca numărul de apariţii a unui caracter în şirul $A$.

Algoritmul este schiţat mai jos:

1. Considerăm un tabel: *t['a'... 'z'] = (0, 0,..., 0)*
2. Pentru fiecare *i de la 1 la m*
   a. Daca *i<=n*, atunci *t[A[i]]:=t[A[i]]+1*;
   b. Grad := Grad + t[B[i]];
   c. Daca *i>=m-n+1* atunci *t[A[n+i-m]]:=t[A[n+i-m]]-1*

Această soluţie are complexitatea $O(m)$ şi funcţionează pentru cazurile când $n$ şi $m$ sunt mai mici sau egali cu 2 000 000.

Un alt aspect de care ar trebui ţinut cont este considerarea unui tip de 64 bit pentru G (gradul de corespondenţă).

**Soluţia Pascal**

```
Program Enigma;
var
  A, B : AnsiString;
  n, m, i : LongInt;
  G : Int64;
```

```
  t : array['a' .. 'z'] of LongInt;
  ch : Char;
begin
  ReadLn(A);
  ReadLn(B);
  n := Length(A);
  m := Length(B);
  for ch := 'a' to 'z' do
    t[ch] := 0;
  G := 0;
  for i := 1 to m do
  begin
    if i <= n then
      t[A[i]] := A[p[i]] + 1;
    G := G + t[B[i]];
    if i >= m - n + 1 then
      t[A[n - m + i]] := t[A[n - m + i]] - 1;
  end;
  WriteLn(G);
end.
```

# Puşculiţa

Puşculiţa reprezintă un recipient sau un dispozitiv special, destinat depozitării şi acumulării de monede. Puşculiţele tradiţionale reprezintă figurine din ceramică, goale în interior, în formă de animale, fructe, legume ş.a.m.d. Cele mai populare sunt puşculiţele în formă de purceluş. Deasupra ele au o fantă, în care pot fi aruncate monede.

În Republica Moldova se folosesc monede de valori 1, 5, 10, 25 şi 50 de bani. Greutatea monedei depinde de valoarea ei. În scopuri didactice se consideră că monedele au următoarele greutăţi:

- 1 ban — 1 gram;
- 5 bani — 2 grame;
- 10 bani — 3 grame;
- 25 bani — 4 grame;
- 50 de bani — 5 grame.

Este cunoscut faptul că în puşculiţă se află *N* monede cu greutatea totală de *G* grame.

Prin *M* vom nota suma minimă de bani care ar putea să fie în puşculiţă, iar prin *K* — numărul tuturor combinaţiilor posibile de monede din ea, în condiţiile în care greutatea monedelor din puşculiţă este egală cu *G*.

De exemplu, pentru $N = 2$ şi $G = 4$ în puşculiţă se pot afla:

- 2 monede de 5 bani, în suma totală de 10 bani;
- o monedă de 1 ban şi o monedă de 10 bani, în suma totală de 11 bani.

Evident, în acest exemplu, suma minimă de bani este $M = 10$, iar numărul tuturor combinaţiilor posibile de monede este $K = 2$.

**Sarcină.** Elaboraţi un program, care, cunoscând numărul de monede *N* din puşculiţă şi greutatea acestora *G*, calculează suma minimă de bani *M* care ar putea să fie în ea şi numărul tuturor combinaţiilor posibile de monede *K*.

**Date de intrare.** Intrarea standard conţine pe o singură linie două numere întregi separate prin spaţiu: numărul de monede *N* şi greutatea monedelor *G* (în grame).

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie două numerele întregi separate prin spaţiu: suma minimă *M* (în bani) şi numărul tuturor combinaţiilor posibile *K*.

**Restricţii.** $1 \leq N \leq 100$; $3 \leq G \leq 500$. Timpul de execuţie nu va depăşi 0,1 secunde. Programul va folosi cel mult 1 Megaoctet de memorie operativă. Fişierul sursă va avea denumirea `pusculita.pas`, `pusculita.c` sau `pusculita.cpp`.

**Exemplu.**

| *Intrare* | *Ieşire* |
|---|---|
| `2 4` | `10 2` |

## Puşculiţa / Копилка

Копилка представляет собой контейнер или специальное устройство для хранения и накопления монет. Как правило, это полые керамические статуэтки в форме животных, фруктов, овощей и т.п. Наиболее популярными являются копилки в форме поросенка. Сверху у такой копилки есть щель, в которую можно бросать монеты.

В Республике Молдова находятся в обращении монеты достоинством 1, 5, 10, 25 и 50 баней. Вес монеты определяется ее достоинством. В дидактических целях считается, что монеты имеют следующие веса:

- 1 бан — 1 грамм;
- 5 баней — 2 грамма;
- 10 баней — 3 грамма;
- 25 баней — 4 грамма;
- 50 баней — 5 граммов.

Известно, что в копилке находятся *N* монет с общим весом *G* грамма.

Через *M* обозначим минимальную сумму денег, которая может быть в копилке, а через *K* — число всевозможных сочетаний монет, при условии, что их суммарный вес равен *G*.

Например, для $N = 2$ и $G = 4$, в копилке могут быть:

- 2 монеты по 5 баней, в общей сумме 10 баней;
- одна монета в один бан и одна монета в 10 баней, в общей сумме 11 баней.

Очевидно, в этом примере минимальная сумма денег $M = 10$, а число всевозможных сочетаний монет $K = 2$.

**Задание.** Напишите программу, которая, зная число монет *N*, содержащихся в копилке, и их общий вес *G*, вычисляет минимальную сумму денег *M* и число всевозможных сочетаний монет *K*, что могут быть в ней.

**Входные данные.** Стандартный ввод содержит в единственной строке два целых числа, разделенных пробелом: число монет *N* и их общий вес *G* (в граммах).

**Выходные данные.** Стандартный вывод должен содержать в единственной строке два целых числа, разделенных пробелом: минимальная сумма денег *M* (в банах) и число всевозможных сочетаний монет *K*.

**Ограничения.** $1 \le N \le 100$; $3 \le G \le 500$. Время выполнения программы не должно превышать 0,1 секунды. Объем используемой оперативной памяти не должен превышать 1 Мегабайт. Исходный файл должен иметь имя `pusculita.pas`, `pusculita.c` или `pusculita.cpp`.

**Пример.**

| *Ввод* | *Вывод* |
|---|---|
| `2 4` | `10 2` |

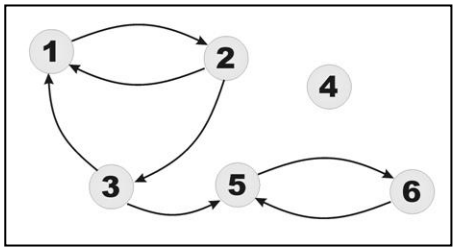
**Rezolvare**

```
#include <iostream>
#include <ctime>

const size_t SIZE = 5;
typedef size_t monede_t[SIZE];
const size_t values[5] = {1, 5, 10, 25, 50};

size_t greutatea(const monede_t& p) {
    size_t result = 0;
    size_t i;
    for (i = 0; i != SIZE; ++i) {
        result += (i + 1) * p[i];
    }
    return result;
}

size_t valoarea(const monede_t& p) {
    size_t result = 0;
    size_t i;
    for (i = 0; i != SIZE; ++i) {
        result += values[i] * p[i];
    }
    return result;
}

void init(monede_t& p, size_t numar) {
    size_t index;
    for (index = 0; index != SIZE; ++index) {
        p[index] = 0;
    }
    p[0] = numar;
}

bool is_last(monede_t& p, size_t numar) {
    size_t i;
    for (i = 0; i != SIZE - 1; ++i) {
        if (p[i] != 0) {
            return false;
        }
    }
    return p[i] == numar;
}

std::ostream& operator<<(std::ostream& out, const monede_t& p) {
    for (auto el : p) {
        out << el << " ";
    }
    return out;
}

void next(monede_t& p) {
    size_t i = SIZE - 2;
    size_t tmp;
    while (p[i] == 0) {
        --i;
        if (i == -1) {
            return;
        }
    }
    p[i] = p[i] - 1;
```

```
    tmp = p[SIZE - 1];
    p[SIZE - 1] = 0;
    p[i + 1] = 1 + tmp;
}

int main(int argc, char** argv) {
    clock_t start, end;

    monede_t monede;
    size_t min_ = 0;
    size_t count = 0;
    size_t n, g;
    std::cin >> n >> g;
//        start = clock();

    min_ = n * 50;
    if (n > g) {
        std::cout << "no solution" << std::endl;
        return 0;
    }
    init(monede, n);
    while (is_last(monede, n) == false) {
//                std::cout << monede
//                        << ", greutatea = " << greutatea(monede)
//                        << ", valoarea = " << valoarea(monede)
//                        << std::endl;
        if (greutatea(monede) == g) {
            ++count;
            if (valoarea(monede) < min_) {
                min_ = valoarea(monede);
            }
        }
        next(monede);
    }
    if (greutatea(monede) == g) {
        ++count;
        if (valoarea(monede) < min_) {
            min_ = valoarea(monede);
        }
    }
//        end = clock();
//        std::cout << "used: " << (double)(end - start) / CLOCKS_PER_SEC
<< std::endl;
    if (count == 0) {
        std::cout << "no solution";
        return 0;
    }
    std::cout << min_ << " " << count;
    return 0;
}
```

# Regiuni

Într-o ţară există $n$ localităţi, numerotate prin $1, 2, 3, \ldots, n$. Unele localităţi sunt legate între ele prin drumuri, notate prin $(i, j)$, însă circulaţia pe fiecare din aceste drumuri este permisă doar într-un singur sens, şi anume, de la localitatea $i$ la localitatea $j$ (vezi figura alăturată). Se consideră că orice drum $(i, j)$ nu trece prin alte localităţi.




Orice succesiune de drumuri $(i, k)$, $(k, m)$, ..., $(l, j)$, ce oferă posibilitatea de a ajunge regulamentar, conform sensurilor de circulaţie permise, din localitatea $i$ în localitatea $j$ se numeşte traseu şi se notează prin $(i, k, m, \ldots, l, j)$. Localităţile $i$, $j$ se numesc interconectate, dacă între ele există cel puţin câte un traseu, de la $i$ la $j$ şi de la $j$ la $i$. Prin definiţie, fiecare localitate este interconectată cu ea înseşi.

În exemplul din figura alăturată, localităţile 1 şi 2 sunt interconectate prin traseele (1, 2) şi (2, 1); localităţile 1 şi 3 – prin traseele (1, 2, 3) şi (3,1); localităţile 5 şi 6 – prin traseele (5, 6) şi (6, 5). Localitatea 4 nu este interconectată cu nici o altă localitate. Localitatea 3 nu este interconectată cu localitatea 5, chiar dacă există traseul (3, 5), deoarece nu există nici un traseu din localitatea 5 în localitatea 3.

Guvernul a decis să grupeze localităţile ţării în regiuni, fiecare regiune urmând să fie formată doar din localităţi ce sunt interconectate intre ele. Din raţiuni de eficienţă administrativă şi economică, numărul unor astfel de regiuni trebuie să fie cât mai mic.

**Sarcină.** Elaboraţi un program, care, cunoscând localităţile şi drumurile ce le leagă, calculează numărul minimal posibil de regiuni, fiecare regiune fiind formată doar din localităţi ce sunt interconectate.

**Date de intrare.** Intrarea standard conţine pe prima linie numărul întreg $n$. Fiecare din următoarele linii ale intrării standard conţine câte două numere întregi, separate prin spaţiu. Primul număr indică localitatea $i$, iar al doilea – localitatea $j$ a drumului $(i, j)$. Enumerarea drumurilor se termină cu ultima linie, care conţine două valori de 0, separate prin spaţiu.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie un număr întreg – numărul minimal posibil de regiuni.

**Restricţii.** $3 \leq n \leq 1000$. Timpul de execuţie nu va depăşi 0,2 secunde. Programul va folosi cel mult 16 Megaocteţi de memorie operativă. Fişierul sursă va avea denumirea `regiuni.pas`, `regiuni.c` sau `regiuni.cpp`.

**Exemplu 1** (vezi figura de mai sus)**.**

*Intrare*

```
6
1 2
2 1
2 3
3 5
3 1
5 6
6 5
0 0
```

*Ieşire*

```
3
```

**Exemplu 2.**

*Intrare*

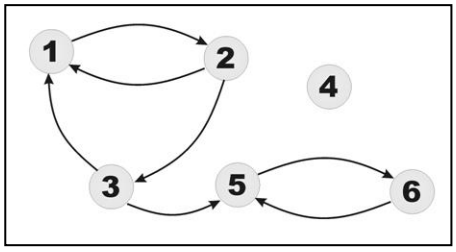
```
4
1 2
2 1
2 4
4 2
3 1
3 4
0 0
```

*Ieşire*

```
2
```

## Регионы

В некоторой стране есть $n$ населенных пунктов, пронумерованных через $1, 2, 3, \ldots, n$. Некоторые населенные пункты соединены дорогами, обозначаемые через $(i, j)$. Движение по каждой из этих дорог является односторонним, а именно, от населенного пункта $i$ к населенному пункту $j$. По определению, любая дорога $(i, j)$ не проходит ни через один другой населенный пункт.




Любая последовательность дорог $(i, k)$, $(k, m)$, ..., $(l, j)$, по которой, из населенного пункта $i$ можно доехать до населенного пункта $j$, соблюдая, естественно, правила дорожного движения, называется маршрутом и обозначается через $(i, k, m, \ldots, l, j)$. Населенные пункты $i$, $j$ называются взаимосвязанными, если существует хотя бы один маршрут от $i$ до $j$ и хотя бы один маршрут от $j$ до $i$. По определению, каждый населенный пункт взаимосвязан с самим собой.

В примере, показанном на рисунке, населенные пункты 1 и 2 взаимосвязаны, т.к. существуют маршруты (1, 2) и (2, 1). Населенные пункты 1 и 3 также взаимосвязаны, т.к. существуют маршруты (1, 2, 3) и (3,1). Населенные пункты 5 и 6 тоже взаимосвязаны, т.к. существуют маршруты (5, 6) и (6, 5). Населенный пункт 4 не взаимосвязан ни с одним другим населенным пунктом. Населенные пункты 3 и 5 не являются взаимосвязанными, поскольку, хотя и существует маршрут (3, 5), не существует ни одного маршрута от населенного пункта 5 к населенному пункту 3.

Правительство решило объединить населенные пункты страны в регионы. Каждый регион должен включать только населенные пункты, взаимосвязанные между собой. Из соображений административной эффективности, число регионов должно быть минимальным.

**Задание.** Напишите программу, которая, зная населенные пункты и дороги, связывающие их, вычисляет минимально возможное число регионов, состоящих из взаимосвязанных населенных пунктах.

**Входные данные.** Стандартный ввод содержит в первой строке целое число $n$. Каждая из следующих строк стандартного ввода содержит по два целых числа, разделенных пробелом. Первое число указывает населенный пункт $i$, а второе число ─ населенный пункт $j$ дороги $(i, j)$. Перечень дорог заканчивается строкой, содержащей два значения 0, разделенных пробелом.

**Выходные данные.** Стандартный вывод должен содержать в единственной строке одно целое число ─ минимально возможное число регионов.

**Ограничения.** $3 \leq n \leq 1000$. Время выполнения программы не должно превышать 0,2 секунды. Объем используемой оперативной памяти не должен превышать 16 Мегабайт. Исходный файл должен иметь имя `regiuni.pas`, `regiuni.c` или `regiuni.cpp`.

**Пример 1.** (смотри рисунок).

*Ввод*

```
6
1 2
2 1
2 3
3 5
3 1
5 6
6 5
0 0
```

*Вывод*

```
3
```

**Пример 2.**

*Ввод*

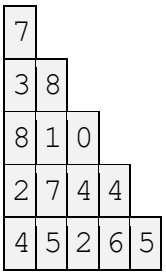
```
4
1 2
2 1
2 4
4 2
3 1
3 4
0 0
```

*Вывод*

```
2
```

# Triunghiuri numerice

Se consideră un triunghi format din numere naturale, în care pe prima line apare un număr, pe a linia doua apar două numere, pe linia a treia apar trei numere ș.a.m.d. Pentru exemplificare, în figura alăturată este prezentat un triunghi format din cinci linii.




Numărul de linii ale triunghiului numeric se notează prin *n*.

Prin drum vom înţelege o secvenţă de numere $(a_1, a_2, \dots, a_i, \dots, a_n)$, formată după cum urmează:

- $a_1$ este numărul din vârful de sus al triunghiului;
- succesor al numărului $a_i$ poate fi doar unul din numerele care se află pe linia de mai jos al triunghiului, şi anume, fie cel imediat dedesubt, fie cel de pe diagonala din dreapta, fie cel de pe diagonala din stânga;
- numărul $a_n$ aparţine liniei de bază a triunghiului numeric.

Lungimea drumului se defineşte ca suma numerelor din secvenţa în cauză:

$$L = a_1 + a_2 + \cdots + a_i + \cdots + a_n.$$

Exemple de drumuri:

$(7, 3, 1, 7, 5), L = 23;$

$(7, 8, 8, 7, 2), L = 32;$

$(7, 8, 0, 4, 5), L = 24.$

Prin $L_{min}$ vom nota lungimea celui mai scurt, iar prin $L_{max}$ lungimea celui mai lung drum din triunghiul numeric.

**Sarcină.** Elaboraţi un program, care, cunoscând triunghiul numeric, calculează lungimile celui mai scurt şi celui mai lung din drumuri.

**Date de intrare.** Intrarea standard conţine pe prima linie numărul întreg *n*. Următoarele linii ale intrării standard reprezintă liniile triunghiului numeric, de la vârf la bază. Linia $i + 1$ a intrării standard conţine *i* numere întregi separate prin spaţiu.

**Date de ieşire.** Ieşirea standard va conţine pe o singură linie numerele întregi $L_{min}$ şi $L_{max}$ separate prin spaţiu.

**Restricţii.** $1 \leq n \leq 100$. Numerele care apar în triunghiul numeric sunt naturale şi au valori mai mici ca $10^7$. Timpul de execuţie nu va depăşi 0,1 secunde. Programul va folosi cel mult 1 Megaoctet de memorie operativă. Fişierul sursă va avea denumirea `triunghi.pas`, `triunghi.c` sau `triunghi.cpp`.

**Exemplu.**

*Intrare*

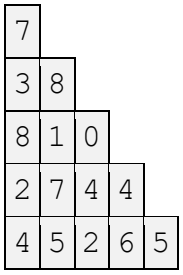
```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

*Ieşire*

```
17 35
```

**Числовые треугольники**

Рассматриваются числовые треугольники, состоящие из натуральных чисел. В таких треугольниках первая строка содержит одно число, вторая строка – два числа, третья строка – три числа и т.д. В качестве примера, на рисунке представлен числовой треугольник, состоящий из пяти строк.




Число строк треугольника обозначается через $n$.

По определению, путем называется последовательность чисел вида $(a_1, a_2, \dots, a_i, \dots, a_n)$, получаемая следующим образом:

- $a_1$ является числом из первой, верхней строки треугольника;
- последователем числа $a_i$ может быть одно из трех чисел, находящихся в строке снизу, а именно, либо под ним, либо влево, либо вправо по диагонали;
- число $a_n$ находится в последней строке, являющейся основанием числового треугольника.

Длина пути определяется как сумма чисел рассматриваемой последовательности:

$$L = a_1 + a_2 + \cdots + a_i + \cdots + a_n.$$

Примеры путей:

$$(7, 3, 1, 7, 5), L = 23;$$
$$(7, 8, 8, 7, 2), L = 32;$$
$$(7, 8, 0, 4, 5), L = 24.$$

Через $L_{min}$ обозначим длину самого короткого, а через $L_{max}$ длину самого длинного пути в числовом треугольнике.

**Задание.** Напишите программу, которая, зная числовой треугольник, вычисляет длину самого короткого и длину самого длинного пути.

**Входные данные.** Стандартный ввод содержит в первой строке целое число $n$. Следующие $n$ строк стандартного входа содержат строки числового треугольника, от вершины к основанию. Строка $i + 1$ стандартного ввода содержит $i$ целых чисел, разделенных пробелом.

**Выходные данные.** Стандартный вывод должен содержать в единственной строке целые числа $L_{min}$ и $L_{max}$, разделенные пробелом.

**Ограничения.** $1 \leq n \leq 100$. Числовой треугольник состоит из натуральных чисел, меньших, чем $10^7$. Время выполнения программы не должно превышать 0,1 секунды. Объем используемой оперативной памяти не должен превышать 1 Мегабайт. Исходный файл должен иметь имя `triunghi.pas`, `triunghi.c` или `triunghi.cpp`.

**Exemplu.**

*Intrare*

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

*Ieşire*

```
17 35
```

## Rezolvare

Problema se rezolvă prin metoda programării dinamice, folosind în acest scop o matrice triunghi în care reţinem datele iniţiale şi - pe baza ei – se construieşte o altă matrice triunghiulară *T* cu următoarea semnificaţie: *T[i,j]* = suma maxima pentru cel mai scurt drum care pleacă de pe o poziţie de pe ultima linie şi ajunge pe poziţia *(i, j)* în matricea iniţială. Relaţiile de recurenţă sunt:

b) Pentru drumul de lungime maximală

```
T[i,j]=max{A[i,j]+T[i+1,j+1], A[i,j]+T[i+1,j-1],
                    A[i,j]+T[i+1,j]}.
```

b) Pentru drumul de lungime minimală

```
T[i,j]=min{A[i,j]+T[i+1,j+1], A[i,j]+T[i+1,j-1],
                    A[i,j]+T[i+1,j]}.
```

Metoda de PD folosită este varianta *înainte.*

**Exemplu**:

| Drum maxim | Drum minim |
|---|---|
| **7**<br>**3** 8<br>**8** 1 0<br>2 **7** 4 4<br>4 **5** 2 6 5 | **7**<br>**3** 8<br>8 1 0<br>2 7 4 4<br>4 5 2 6 5 |

Atunci răspunsul este valoarea max=35, min=17.

Pentru exemplu de mai sus vom obţine:
a) Pentru drumul de lungime maximala
*2 7 4 4 rândul A[4,j=1,2,3,4]*
*4 5 2 6 5 rândul T[5,j=1,...,5]*
T[4,j=1]=max{2+4,2+5}=**7**;
T[4,j=2]=max{7+4,7+5,7+2}=**12**,
T[4,j=3]= max{4+5,4+2,4+6}=**10**,
T[4,j=4]= max{4+2,4+6,4+5}=**10.**
---
*8 1 0 rândul A[3,j=1,2,3]*
*7 12 10 10 rândul T[4,j=1,...,4]*
T[3,j=1]=max{8+7,8+12}=**20**;
T[3,j=2]=max{1+7,1+12,1+10}=**13**;
T[3,j=3]= max{0+12,0+10,0+10}=**12**.
---
*3 8 rândul A[2,j=1,2]*
*20 13 12 rândul T[3,j=1,2,3]*
*T[2,j=1]=max{ 3+20,3+13}=**23**,*
*T[2,j=2]=max{8+20,8+13,8+12}=**28***
---
*7 rândul A[1,1]*
*23 28 rândul T[2,j=1,2]*
*T[1,j=1]=max{7+23,7+28}=35*

Matricea $T$ are forma: $\begin{pmatrix} 35 & & & & \\ 23 & 28 & & & \\ 20 & 13 & 12 & & \\ 7 & 12 & 10 & 10 & \\ 4 & 5 & 2 & 6 & 5 \end{pmatrix}$

Complexitate timp al algoritmului este $O(n^2)$ unde n este numărul de rânduri.

```
Program Triunghi;
{ Clasele 10-12 }
uses crt;
const nmax=100;
type tab=array[1..nmax, 1..nmax] of longint;
var tmax,tmin,a:tab;
fin,fout:text;
i,j,n:integer;
function max(a,b:longint):longint;
begin
if a>b then max:=a else max:=b;
end;
function min(a,b:longint):longint;
begin
if a<b then min:=a else min:=b;
end;
procedure dinamic;
begin
for i:=1 to n do
begin
tmax[n,i]:=a[n,i];
tmin[n,i]:=a[n,i];
end;
for i:=n-1 downto 1 do
begin
for j:=1 to i do begin
  if j=1 then
  begin
  tmax[i,j]:=max(a[i,j]+tmax[i+1,j],a[i,j]+tmax[i+1,j+1]);
  tmin[i,j]:=min(a[i,j]+tmin[i+1,j],a[i,j]+tmin[i+1,j+1]);
  end
  else
  begin
max[i,j]:=max(max(a[i,j]+tmax[i+1,j],a[i,j]+tmax[i+1,j+1]),a[i,j]+tmax[i+1,j-
1]);
tmin[i,j]:=min(min(a[i,j]+tmin[i+1,j],a[i,j]+tmin[i+1,j+1]),a[i,j]+tmin[i+1,j-
1]);
 end;
               end;
end;
end;
begin
assign(fin,'TEST.in');
reset(fin);
assign(fout,'TEST.out');
rewrite(fout);
readln(fin,n);{se citeste numarul de randuri}
{se citesc randurile matricei din fisier}
for i:=1 to n do
begin
for j:=1 to i do read(fin,a[i,j]);
readln(fin);
end;
```

```
dinamic;
write(fout,tmin[1,1],' ',tmax[1,1]);
close(fin);
close(fout);
end.
```

# Ursul

Ursul Mecho a găsit o mică comoara – rezerva de miere a albinelor! Fericit, el a început să mănânce din comoara nou-găsită, însă, deodată, o albina l-a văzut şi a dat alarma. Mecho ştie că, din acest moment, roiuri întregi de albine vor ieşi din stupi şi vor începe să se răspândească în jur, încercând să-l prindă. El ştie că trebuie să părăsească mierea şi să plece acasă repede, dar mierea este atât de dulce încât Mecho nu vrea sa plece prea curând. Ajutaţi-l pe Mecho să găsească cel mai târziu moment posibil când trebuie să plece de lângă miere.

Pădurea în care se afla Mecho este reprezentata sub forma unui caroiaj compus din $N$ linii şi $N$ coloane, ale cărui laturi sunt paralele cu direcţiile nord-sud şi est-vest. Fiecare celulă a caroiajului este ocupata de un copac, de iarba, de un stup, sau de casa lui Mecho.

Mecho este un urs neîndemânatic, astfel ca de fiecare data când face un pas, acesta are lungimea de exact 1 unitate şi poate fi doar către nord, sud, est sau vest. Mecho se poate deplasa doar prin celulele cu iarba, nu poate trece prin celulele cu copaci şi poate face cel mult $S$ paşi pe minut. Evident, Mecho poate intra în celula în care se află casa lui.

Albinele pot trece prin celulele cu iarbă, însă nu pot trece prin celulele cu copaci. De asemenea, ele nu pot trece prin casa lui Mecho şi nu pot zbura peste aceasta. Pentru ele, casa lui Mecho este un obstacol, la fel ca şi copacul.

La momentul când albinele dau alarma, Mecho se afla în celula ce conţine mierea, iar albinele se află în fiecare din celulele ce conţin câte un stup (în pădure pot exista mai mulţi stupi). În timpul fiecărui minut (începând din momentul în care albinele au dat alarma), următoarele evenimente au loc după cum urmează:

- Dacă Mecho încă mănâncă miere, atunci el decide, să mănânce în continuare sau imediat să plece. Daca Mecho decide să mănânce în continuare, el nu iese din celula cu miere pe întreaga durată a minutului. În caz contrar, el pleacă imediat din celula cu miere şi face prin pădure cel mult $S$ paşi, evident, conform restricţiilor menţionate mai sus. Mecho nu poate lua cu el niciun pic de miere, astfel că o dată ce a plecat din locaţia în care se află mierea, el nu o mai poate mânca.
- Pe durata minutului curent, în timp ce Mecho mănâncă sau se deplasează, albinele se răspândesc mai departe prin caroiaj cu o unitate de distanţă. Mai exact, roiurile de albine se răspândesc din celulele în care se află în toate celulele cu iarba care le sunt vecine (către nord, sud, est sau vest). Mai mult ca atât, o parte din albine rămân pentru totdeauna în celulele în care se aflau deja, adică roiurile de albine cresc, ocupând tot mai multe şi mai multe celule cu iarbă.

Cu alte cuvinte, albinele se răspândesc după cum urmează: Când se dă alarma, ele ocupa doar celulele în care se afla stupi. La sfârşitul primului minut, ele ocupa toate celulele cu iarba, care sunt vecine cu cele ce conţin stupi. La sfârşitul celui de-al doilea minut, ele mai ocupa în plus şi toate celulele cu iarba, adiacente celor ocupate în minutul precedent ş.a.m.d. Daca au la dispoziţie suficient timp, albinele vor ocupa toate celulele cu iarba, la care vor putea ajunge.

Nici Mecho, nici albinele nu pot ieşi din pădure. De asemenea, conform regulilor de mai sus, Mecho manca mierea un număr întreg de minute.

**Sarcină**. Scrieţi un program care, cunoscând harta detaliată a pădurii, calculează numărul maximal posibil de minute pe durata cărora Mecho poate manca miere în celula sa iniţială, cu condiţia că, ulterior, el va putea să ajungă acasă înainte ca albinele sa-l prindă.

**Date de intrare.** Intrarea standard conţine următoarele date:

- Prima linie conţine numerele întregi $N$ şi $S$, separate prin spaţiu.

- Următoarele *N* linii descriu harta pădurii. Fiecare din aceste linii conţine *N* caractere. Fiecare din aceste caractere reprezentă o celula a caroiajului. Valorile posibile şi semnificaţiile asociate acestor caractere sunt:

  *T* – reprezintă o celulă ce conţine un copac;

  *G* – reprezintă o celulă cu iarbă;

  *M* – reprezintă locaţia iniţiala a lui Mecho, care este o celulă cu iarbă;

  *D* – reprezintă locaţia casei lui Mecho;

  *H* – reprezintă locaţia unui stup; celula în cauză întotdeauna conţine şi un copac.

**Nota.** Se garantează ca harta va conţine exact o litera *M*, exact o litera *D* şi cel puţin o litera *H*. Se garantează, de asemenea, că exista o secvenţa de celule adiacente, reprezentate doar prin litere de *G*, care leagă celula în care iniţial se afla Mecho, cu celula în care se află casa acestuia. Aceste secvenţe pot avea chiar şi de lungimea 0 (caz în care celula cu casa lui Mecho este adiacenta cu locaţia iniţială a lui Mecho).

**Date de ieşire**. Ieşirea standard va conţine pe o prima linie un singur număr întreg – numărul maximal posibil de minute în care Mecho poate manca miere în celula sa iniţiala, cu condiţia ca ulterior el să poată ajunge acasă în siguranţa. Daca Mecho nu poate ajunge acasă înainte ca albinele sa-l prindă, numărul care trebuie afişat la ieşirea standard este -1. De asemenea, numarul „-1” se va scrie si in cazurile in care albinele nu-l pot prinde pe Mecho in nici un fel.

**Restricţii.** $1 \leq N \leq 800$; $1 \leq S \leq 1000$. Timpul de execuţie nu va depăşi 0,3 secunde. Programul va folosi cel mult 32 Megaocteţi de memorie operativă. Fişierul sursă va avea denumirea `ursul.pas`, `ursul.c` sau `ursul.cpp`.

**Exemplul 1.**

*Intrare*

```
7 3
TTTTTTT
TGGGGGT
TGGGGGT
MGGGGGD
TGGGGGT
TGGGGGT
THHHHHT
```

*Ieşire*

```
1
```

După ce mănâncă miere timp de un minut, Mecho merge pe drumul cel mai scurt, spre dreapta, şi ajunge acasă după alte doua minute, fără ca albinele sa-l prindă.

**Exemplul 2.**

*Intrare*

```
7 3
TTTTTTT
TGGGGGT
TGGGGGT
MGGGGGD
TGGGGGT
TGGGGGT
TTHHTTT
```

*Ieşire*

```
2
```

După ce mănâncă miere timp de două minute, în timpul celui de-al treilea minut Mecho face paşii $\rightarrow \uparrow \rightarrow$; în timpul celui de-al patrulea minut – paşii $\rightarrow \rightarrow \rightarrow$; timpul celui de-al cincilea minut – paşii $\downarrow \rightarrow$.

## Медведь

Медведь по имени Мекко нашел небольшой клад – мед, который пчелы заготовили на зиму. Он начал есть найденный мед, однако сразу же одна из пчел увидела его и забила тревогу. Медведь знает, что, начиная с этого момента, целые рои пчел выходят из своих ульев и начинают распространяться во все стороны, намереваясь поймать его. Хотя Мекко понимает, что он должен оставить мед и вернуться домой, мед так сладок, что он не хочет уходить слишком рано. Помогите Мекко определить самый поздний момент, когда он должен оставить мед и направиться домой.

Карта леса, в котором находится Мекко, представлена в виде сетки, состоящей из $N$ строк и $N$ столбцов, которые, соответственно, параллельны направлениям север-юг и запад-восток. Каждая клетка сетки может быть занята деревом, травой, ульем, или домом Мекко.

Мекко неуклюж, поэтому он может перемещаться только отдельными шагами. Каждый шаг представляет собой переход из текущей клетки в одну из соседних клеток, на север, на юг, на восток или на запад. По определению, длина шага равна единице. За одну минуту Мекко может выполнить не более $S$ шагов. Естественно, Мекко может ходить только через клетки с травой, не может проходить через клетки с деревьями, и может заходить в клетку со своим домом.

Пчелы могут проходить через клетки с травой, но не могут пролетать через клетки с деревьями. Они также не могут пролетать через дом Mecho, а также не могут пролетать над ним. Для них дом Mecho является таким же препятствием, как и деревья.

В момент времени, когда пчела подает сигнал тревоги, Мекко находится в клетке с медом, а пчелы – в клетках с ульями (в общем случае, в лесу могут быть много ульев). В каждую из минут, начиная с момента подачи сигнала тревоги, происходят следующие события:

- Если Мекко ест мед, он решает, будет ли он есть дальше, или должен немедленно уйти. Если он принял решение есть мед дальше, он остается в клетке с медом на протяжении всей текущей минуты. В противном случае, он немедленно выходит из клетки с медом и проходит по лесу до $S$ шагов, естественно, в соответствии с ограничениями, упомянутыми выше. Мекко не может взять мед с собой, так что, после ухода из исходной клетки, больше он его не ест.
- В то время как Мекко ест мед либо двигается по лесу, пчелы распространяются на расстояние равной одной единице. Точнее, рои пчел распространяются из клеток, где они находились до начала текущей минуты, в клетки с травой, находящиеся рядом, на север, на юг, на восток или на запад. Более того, однажды заняв некоторую клетку, часть пчел остается в ней навсегда, т.е.рои пчел растут.

Другими словами, в момент подачи сигнала тревоги, пчелы находятся в клетках с ульями. В конце первой минуты после подачи сигнала тревоги, они захватывают клетки с травой, которые являются соседними с клетками с ульями. В конце второй минуты, пчелы захватывают клетки с травой, соседние с ранее занятыми клетками, и так далее. После определенного числа минут, все клетки, до которых смогли добраться пчелы, окажутся занятыми.

Ни Мекко, ни пчелы не могут выйти за пределы леса. В соответствии с вышеуказанными правилами, Мекко всегда ест мед целое число минут.

**Задание**. Напишите программу, которая, зная подробную карту леса, вычисляет максимальное количество минут, в течение которых Mecho может есть мед, при условии, что после этого он сможет добраться домой без того чтобы пчелы его поймали.

**Входные данные.** Стандартный ввод содержит следующие данные:

- Первая строка содержит целые числа $N$ и $S$, разделенные пробелом.

- Следующие $N$ строк описывают карту леса. Каждая из этих строк содержит $N$ символов. Каждый символ описывает клетку сетки. Символы имеют следующий смысл:
  $T$ – обозначает клетку с деревом;
  $G$ – обозначает клетку с травой;
  $M$ – обозначает клетку с Мекко, которая, одновременно, является и клеткой с травой;
  $D$ – обозначает клетку с домом Мекко;
  $H$ – обозначает клетку с ульем, которая, одновременно всегда содержит и дерево.

**Примечание.** Гарантируется, что карта содержит ровно одну букву *M*, ровно одну букву *D*, и хотя бы одну букву *H*. Также гарантируется, что существует последовательность соседних клеток, обозначенных буквой *G*, которая связывает клетку, в которой находится Мекко, с клеткой с его домом. Такие последовательности могут иметь и нулевую длину (случай, при котором клетка с домом Мекко является соседней с клеткой, в которой, в исходном состоянии, находится Мекко).

**Выходные данные**. Стандартный вывод должен содержать в единственной строке одно целое число: максимальное число минут, во время которых Мекко может есть мед, при условии, что после этого он сможет добраться домой до того, как пчелы его поймают. Если же Mecho не сможет дойти домой до того, как пчелы его поймают, стандартный вывод должен содержать целое число -1. Число „-1" пишется и в тех случаях, когда пчелы никак не смогут поймать Мекко.

**Ограничения.** $1 \leq N \leq 800$; $1 \leq S \leq 1000$. Время выполнения программы не должно превышать 0,3 секунды. Объем используемой оперативной памяти не должен превышать 32 Мегабайт. Исходный файл должен иметь имя `ursul.pas, ursul.c` или `ursul.cpp`.

**Пример 1.**

*Ввод*

```
7 3
TTTTTTT
TGGGGGT
TGGGGGT
MGGGGGD
TGGGGGT
TGGGGGT
THHHHHT
```

*Вывод*

```
1
```

В течение одной минуты Мекко ест мед; в течение следующих двух минут Мекко идет все время вправо.

**Пример 2.**

*Ввод*

```
7 3
TTTTTTT
TGGGGGT
TGGGGGT
MGGGGGD
TGGGGGT
TGGGGGT
TTHHTTT
```

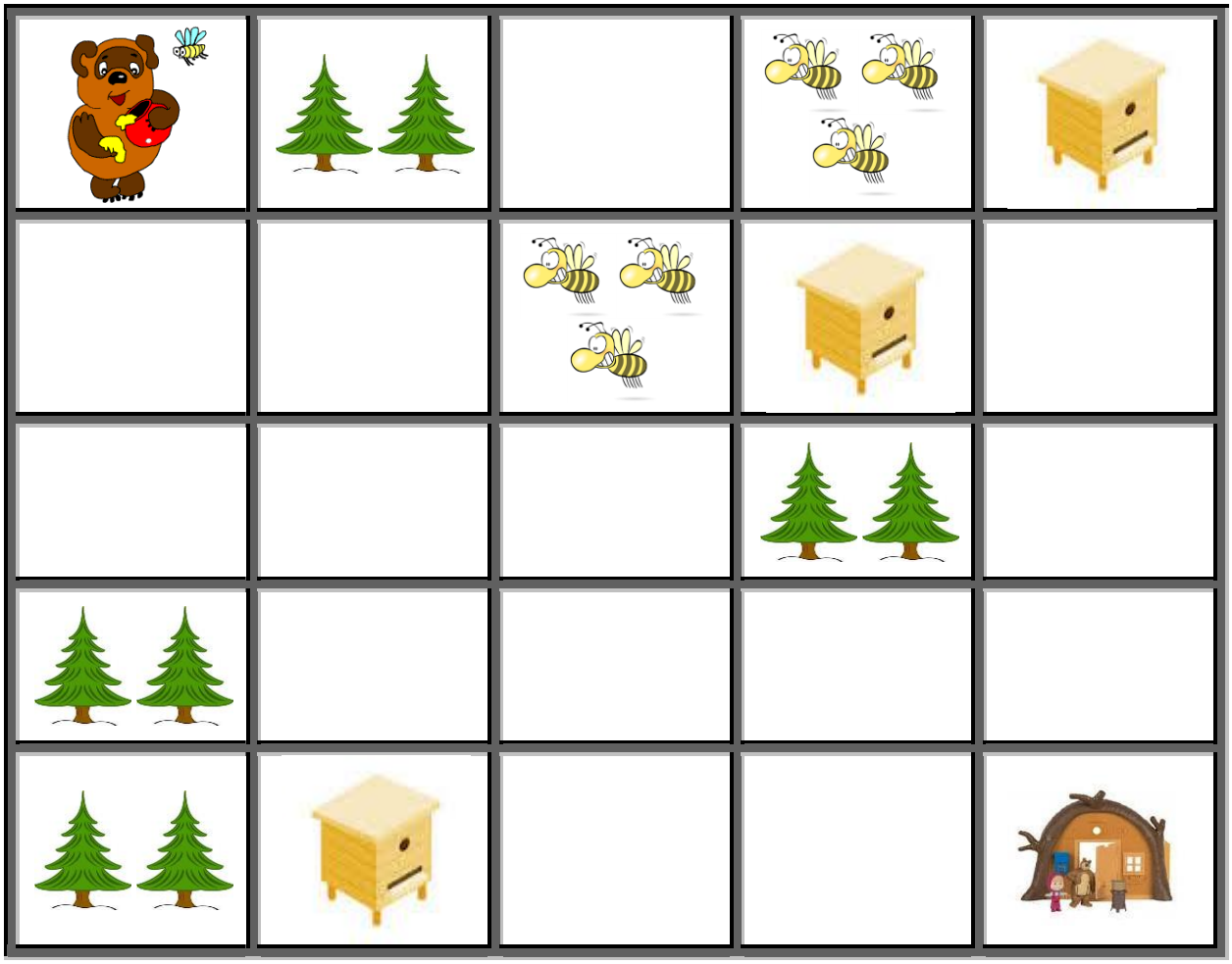
*Вывод*

```
2
```

В течение двух минут Мекко ест мед; в течение третей минуты Мекко выполняет последовательность шагов →, ↑, →; в течении четвертой минуты – последовательность шагов →, →, →; в течении пятой минуты – последовательность шагов ↓, →.

## Rezolvare

Ursul Mecho a găsit o mică comoara – rezerva de miere a albinelor! Fericit, el a început să mănânce din comoara nou-găsită, însă, deodată, o albina l-a văzut şi a dat alarma. Mecho ştie că, din acest moment, roiuri întregi de albine vor ieşi din stupi şi vor începe să se răspândească în jur, încercând să-l prindă. El ştie că trebuie să părăsească mierea şi să plece acasă repede, dar mierea este atât de dulce încât Mecho nu vrea sa plece prea curând. Ajutaţi-l pe Mecho să găsească cel mai târziu moment posibil când trebuie să plece de lângă miere.



***Alegerea tipului potrivit de date – Lucrăm cu numere întregi***

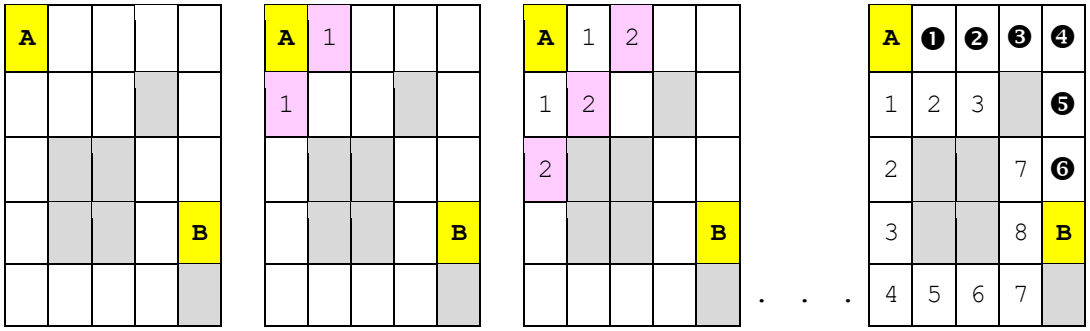
Durata unui pas al ursului este de $\frac{1}{S}$ secunde. Întrucât lucrul cu numere fracţionare poate duce la erori, vom măsura timpul nu în secunde, ci în unităţi speciale, denumite ticuri:

$$1\ tic = \frac{1}{S} \text{ secunde.}$$

Evident, în cazul noii unităţi de măsură a timpului, durata unui pas de urs este de 1 tic, iar a unui "pas" de albină – de *S* ticuri.

***Algoritmul lui Lee***

Acest algoritm este destinat găsirii celui mai scurt drum din celula *A* în celula *B* pe un teren cu obstacole (celulele haşurate). În acest scop propagăm o undă numerică, după cum se arată în figura de mai jos:




Complexitatea algoritmului Lee depinde de metoda de implementare:

- iteraţii: $O(N^3)$;
- recursie în adâncime: $O(N^2)$.

***Cătarea soluţiei prin metoda trierii***

Vom căuta timpul $t_{max}$ pe durată căruia ursul poate mânca miere în celula în care iniţial se află după cum urmează:

1) Facem o copie *A* a hărţii pădurii şi calculăm cu ajutorul algoritmului lui Lee cel mai mic timp $a_{i,j}$ (cel mai scurt drum), de care vor avea nevoie albinele pentru a ajunge în fiecare din celulele libere $(i, j)$.

2) Atribuim lui *t* valoarea 0.

3) Facem o copie $U^t$ a hărţii pădurii şi calculăm cu ajutorul algoritmului lui Lee cel mai mic timp $u_{i,j}^t$ (cel mai scurt drum), de care va avea nevoie ursul pentru a ajunge în fiecare din celulele libere $(i, j)$.

4) Alcătuim harta obstacolelor $H^t$, în care obstacole, pe lângă cele din harta iniţială, devin şi celule în care albinele ajung mai repede sau chiar odată cu ursul, adică, celulele pentru care $a_{i,j} \leq u_{i,j}^t$.

5) Căutăm pe harta obstacolelor $H^t$ un drum oarecare ce leagă celula în care se află mierea cu celula în care se află casa ursului. Evident, putem folosi în acest scop algoritmul lui Lee.

6) Repetăm punctele 3) - 5) atâta timp, cât există drumuri neprimejdioase. Ultima din valorile lui *t* şi va fi soluţia $t_{max}$.

Întrucât harta iniţială are dimensiunile $N \times N$, lungimea celui mai lung drum nu poate depăşi valoarea $N^2$. Prin urmare, numărul de iteraţii cerut de acest algoritm este mai mic ca $N^2$.

Complexitatea algoritmului de căutare a soluţiei prin metoda trierii depinde de modul de implementare a algoritmului lui Lee: $O(N^4)$ în cazul recursiei şi $O(N^5)$ în cazul iteraţiilor.

***Găsirea soluţiei prin căutarea binară***

Căutarea soluţiei se efectuează în acelaşi mod ca şi în cazul trierii, însă valorile ce i se atribuie lui *t* se calculează prin înjumătăţirea consecutivă a intervalului iniţial $[0, N^2]$; a unuia din intervalele $[0, \frac{N^2}{2}]$, $[\frac{N^2}{2} + 1, N^2]$ ş.a.m.d.

Numărul de iteraţii, cerut de algoritmul de căutare binară, este de $\log_2 N^2$. Evident, complexitatea algoritmului de găsire a soluţiei prin căutarea binară depinde de modul de implementare a algoritmului lui Lee şi va fi $O(N^2 \log_2 N)$ în cazul recursiei şi $O(N^3 \log_2 N)$ în cazul iteraţiilor.

***Găsirea soluţiei prin programarea dinamică***

Se porneşte de la căsuţa ursului, parcurgând celulele hărţii în direcţia celulei cu miere. În procesul parcurgerii hărţii se calculează momentul de timp în care ursul trebuie să plece din celula curentă, cu condiţia ca ulterior să ajungă în siguranţă acasă. Evident, în cazul celulei cu căsuţa ursului, acest timp este ∞.

Formula de recurenţă poate fi obţinută în baza următoarelor raţionamente. Presupunem că pentru a ajunge în siguranţă acasă, ursul trebuie să plece din celula curentă $(x, y)$ după cel mult $t_{x,y}$ ticuri de la declanşarea alarmei. Cât timp $t_{z,w}$ s-ar putea el afla într-o celulă $(z, w)$, vecină cu celula $(x, y)$ ?

Evident, limita de sus a acestui timp este $(t_{x,y} - 1)$, întrucât în caz contrar el ar ajunge prea târziu în celula curentă. Limita de jos a acestui timp este dată de sosirea albinelor în celula în cauză $a_{z,w}$, întrucât, în caz contrar, ele îl vor înţepa. Prin urmare, ursul trebuie să plece din celula $(x, y)$ după cel târziu în momentul de timp $t_{x,y} = \min(t_{x,y} - 1, a_{z,w})$.

Întrucât o celulă poate avea mai mulţi vecini liberi, calculele în cauză se vor face pentru fiecare din ei, memorând în continuare vecinul cu cel mai târziu moment de timp.

Complexitatea temporară a acestei soluţii depinde de lungimea cozii în care sunt ordonate celulele în funcţie de timpul de părăsire sa acestora. Un *heap* binar ar garanta o complexitate de $O(N^2 \log N)$, complexitate suficientă ca să treacă chiar şi cele mai "grele" teste.

Juriul a ales testele de verificare a soluţiilor propuse de competitori în aşa mod, încât să se facă o discriminare a soluţiilor de complexitate $O(N^2 \log_2 N)$, $O(N^3 \log_2 N)$, $O(N^4)$ şi $O(N^5)$, fapt ce permite alocarea punctelor în funcţie de ingeniozitatea soluţiei propuse.

```
/**
 * A binary search solution for IOI 2009 problem "mecho"
 *
 * This solution should score 100%
 *
 * Carl Hultquist, chultquist@gmail.com
 */

#include <iostream>
#include <string>
#include <cstdlib>
#include <fstream>
#include <vector>
#include <utility>
#include <deque>
#include <cstring>

using namespace std;
```

```
#define MAX_N 2000

int cx[4] = {1, -1, 0, 0};
int cy[4] = {0, 0, 1, -1};

char mainMap[MAX_N][MAX_N];
bool reachable[MAX_N][MAX_N];

// The time that it takes the bees to reach any cell in the map
int beeDistance[MAX_N][MAX_N];

int n, s;
int dx, dy;
int mx, my;

/**
 * Tests if Mecho is able to reach his home after staying with
 * the honey for the given delay time.
 */
bool test(int delay)
{
    // Check if the bees catch Mecho whilst he is still with
    // the honey.
    if (delay * s >= beeDistance[mx][my])
        return false;

    // Initialise data structures -- at the beginning of the search,
    // Mecho has only reached the cell with the honey. Note that it
    // is possible for the bees to catch Mecho at the honey -- but
    // we checked for this case above, and so if we reach this point
    // we know that Mecho is safe with the honey after the given
    // delay.
    memset(reachable, 0, sizeof(reachable));
    deque<pair<int, pair<int, int> > > q;
    q.push_back(make_pair(delay * s, make_pair(mx, my)));
    reachable[mx][my] = true;

    // Now do the main loop to see what other cells Mecho can reach.
    while (!q.empty())
    {
        int distance = q.front().first;
        int x = q.front().second.first;
        int y = q.front().second.second;

        q.pop_front();

        // If Mecho has reached his home, then we are done.
        if (mainMap[x][y] == 'D')
            return true;

        // Check neighbouring cells
        for (int c = 0; c < 4; c++)
        {
            int nx = x + cx[c];
            int ny = y + cy[c];

            // Check that the cell is valid, that it is not a tree, and
            // that Mecho can get here before the bees.
            if (nx < 0 || nx >= n || ny < 0 || ny >= n || mainMap[nx][ny]
== 'T' || (distance + 1) >= beeDistance[nx][ny] || reachable[nx][ny])
                continue;

            // All OK, so add it to the queue
```

```
            q.push_back(make_pair(distance + 1, make_pair(nx, ny)));
            reachable[nx][ny] = true;
        }
    }

    // If we reach here, then Mecho was unable to reach his home.
    return false;
}
int main(int argc, char **argv)
{
    // Read in the data
    cin >> n >> s;

    deque<pair<int, int> > bq;
    memset(beeDistance, -1, sizeof(beeDistance));

    for (int i = 0; i < n; i++)
    {
        cin >> ws;
        for (int j = 0; j < n; j++)
        {
            cin >> mainMap[i][j];
            if (mainMap[i][j] == 'H')
            {
                bq.push_back(make_pair(i, j));
                beeDistance[i][j] = 0;
            }
            else if (mainMap[i][j] == 'M')
            {
                mx = i;
                my = j;

                // Bees can travel through the location of the honey
                mainMap[i][j] = 'G';
            }
            else if (mainMap[i][j] == 'D')
            {
                dx = i;
                dy = j;
            }
        }
    }

    // Precompute the time that it takes the bees to reach any other
    // cell in the map.
    while (!bq.empty())
    {
        int x = bq.front().first;
        int y = bq.front().second;

        bq.pop_front();

        for (int c = 0; c < 4; c++)
        {
            int nx = x + cx[c];
            int ny = y + cy[c];

            if (nx < 0 || nx >= n || ny < 0 || ny >= n || mainMap[nx][ny]
!= 'G' || beeDistance[nx][ny] != -1)
                continue;

            beeDistance[nx][ny] = beeDistance[x][y] + s;
            bq.push_back(make_pair(nx, ny));
        }
```

```
    }

    // The bees can never enter Mecho's home, so set this to a large
    // sentinel value.
    beeDistance[dx][dy] = n * n * s;

    // Binary search to find the maximum delay time.
    int low = -1, high = 2 * n * n;
    while (high - low > 1)
    {
        int mid = (low + high) >> 1;
        if (test(mid))
            low = mid;
        else
            high = mid;
    }

    cout << low << endl;
    return 0;
}
```