

Ordine pubblico (corteo)

On September 15, at the Olympics of Informatics, two programmers' marches will take to the streets and squares of Trento. The first group is made of *Spaces* proponents, while the other group is made of *Tabs* enthusiasts.

The city has N squares, linked by M two-way roads. The two groups depart from the squares P_1 and P_2 respectively, and must arrive at squares D_1 and D_2 respectively. Everytime the bell on Trento's Vanga tower will ring, one of the two groups will move to a neighboring square, crossing a street from one end to the other. During the move, the other group stays in its current square. The festive event ends when both parades have reached their destination.

Everyone knows it's not a good idea to mix *Spaces* and *Tabs*: for this reason, the organizers want to prevent the two groups from getting too close to each other. Help the organizers with this task! Write a program that calculates the distance you can keep between the two groups throughout the event, allowing each of them to reach their destination.

Clarifications

In each turn, only one of the two groups goes from one square to another. A movement indicates which of the two parades moves, and to which square. It is valid only if there is a road connecting *directly* the square where the procession is currently located with the square where to take it.

Movements are performed regularly, one at each bell ring, but **it isn't required** that the movements of the two groups alternate: it is possible that the same group moves for two or more consecutive turns.

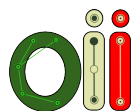
A group *can* go the same way more than once, as well as visiting a square more than once, if the members wish to.

After the last move, each group must be in its destination square: this is the only way for the event to end. The squares of departure and destination of a group may, however, be the same square.

The sequence of movements carried out one after the other during the event is called a *schedule*, and the organizers can choose any schedule they want.


We say that two squares are at *distance* d if you need to cross at least d roads to move from one to the other. The *margin* associated with a schedule is defined as the minimum distance d between two squares **simultaneously** occupied by the two parades. You are asked to find out what is the maximum margin that the organizers can obtain, choosing from every possible schedule.

If your program does not correctly calculate the maximum margin, you can still get a **partial score** by explicitly listing a schedule that gets a high margin, even if not optimal (see the score assignment section).



Implementation

You have to submit only one file, whose extension is `.cpp` or `.c`.

 You can find a template in the attachment section `corteo.cpp` and `corteo.c`, with an implementation example.

You need to implement the following function:

```
C/C++ int pianifica(int N, int M, int P1, int D1, int P2, int D2, int A[], int B[]);
```

The function `pianifica` is called only once, using the following values of the parameters.

- The integer number N represents the number of squares of the city (the squares are indexed from 0 to $N - 1$),
- The integer M represents the number of streets of the city.
- The integer P_1 represents the starting square of the *Spaces* group and D_1 is their last square. The integer number P_2 represents the starting square of the *Tabs* group and D_2 is their last square (P_1 , D_1 , P_2 and D_2 are integer numbers in the range from 0 to $N - 1$).
- The arrays A and B , indexed from 0 to $M - 1$, represent the streets in the city. The street indexed by i , for $0 \leq i \leq M - 1$, connect the squares $A[i]$ with $B[i]$. Keep in mind that the streets are bidirectional: walking along the street i it's possible to move from square $A[i]$ to square $B[i]$ and also from square $B[i]$ to square $A[i]$.
- The function must return an integer number that represents the maximum margin possible in a schedule.

If you want to explicitly describe your schedule with the aim to obtain a partial score (in case the margin your computer isn't optimal) you can print the movements one by one in the sequence by using the following function:

```
C/C++ void sposta(int chi, int dove);
```

- The integer number `chi` may be 1 or 2 and it tells which group we want to move.
- The integer number `dove` represents the square towards which the group is going to move.

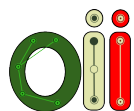
The function `pianifica` is going to be called only once at the beginning of the execution. In your implementation of the function `pianifica` you have the chance to call the function `sposta` how many times you wish. Every call to `sposta` will be stored, as well as the return value of `pianifica`.

Sample Grader

You can find a simplified version of the grader used during the competition in this task's directory. You can use it to test your solutions. The sample grader reads data from `stdin`, calls the functions that you have to implement and writes on `stdout`, according to the following format.

The input file consists of $M + 2$ rows, containing the following numbers, separated from spaces:

- Line 1: the integer numbers N and M .
- Line 2: the integer numbers P_1 , D_1 , P_2 and D_2 .
- Line $3 + i$ for $i = 0, \dots, M - 1$: the integer numbers $A[i]$ and $B[i]$.



The output file consists of $K + 1$ rows, where K represents the number of calls to the function `sposta`, stored in the following manner:

- Lines $1, \dots, K$: the parameters used when calling the function `sposta`.
- Line $K + 1$: the return value of `pianifica`.

Constraints

- $2 \leq N \leq 1000$.
- $1 \leq M \leq 5000$.
- $0 \leq P_1, D_1, P_2, D_2 \leq N - 1$.
- $0 \leq A[i], B[i] \leq N - 1$ for each $i = 0, \dots, M - 1$.
- There are no streets that connect a square to itself or that connect the same pair of squares twice.
- From every square it's possible to reach every other square.

Scoring

Your program is going to be tested on several test cases, grouped in subtasks. Each test case is assigned:

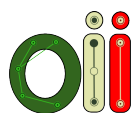
- 1 if the function `pianifica` return the optimum value for that specific case, or if it shows a correct plan that realizes the optimum border: otherwise,
- 0 if it doesn't show a correct schedule; otherwise,
- the score obtained from the formula

$$\frac{1}{2} \cdot \frac{\text{value of your schedule}}{\text{value of the optimal schedule}}$$

thus giving a score between 0 and 0.5.

The score assigned to a subtask is the *worst* of the scores obtained from its test cases, multiplied by the value of the subtask. The score assigned to this problem is given by the sum of scores in the various subtasks.

- **Subtask 1 [0 points]**: Examples.
- **Subtask 2 [9 points]**: The two groups are already at their destination.
- **Subtask 3 [10 points]**: The graph is a cycle.
- **Subtask 4 [13 points]**: $N, M \leq 10$.
- **Subtask 5 [17 points]**: One group does not need to move (in an optimal solution).
- **Subtask 6 [22 points]**: $N, M \leq 100$.
- **Subtask 7 [29 points]**: No specific limitations.



Examples

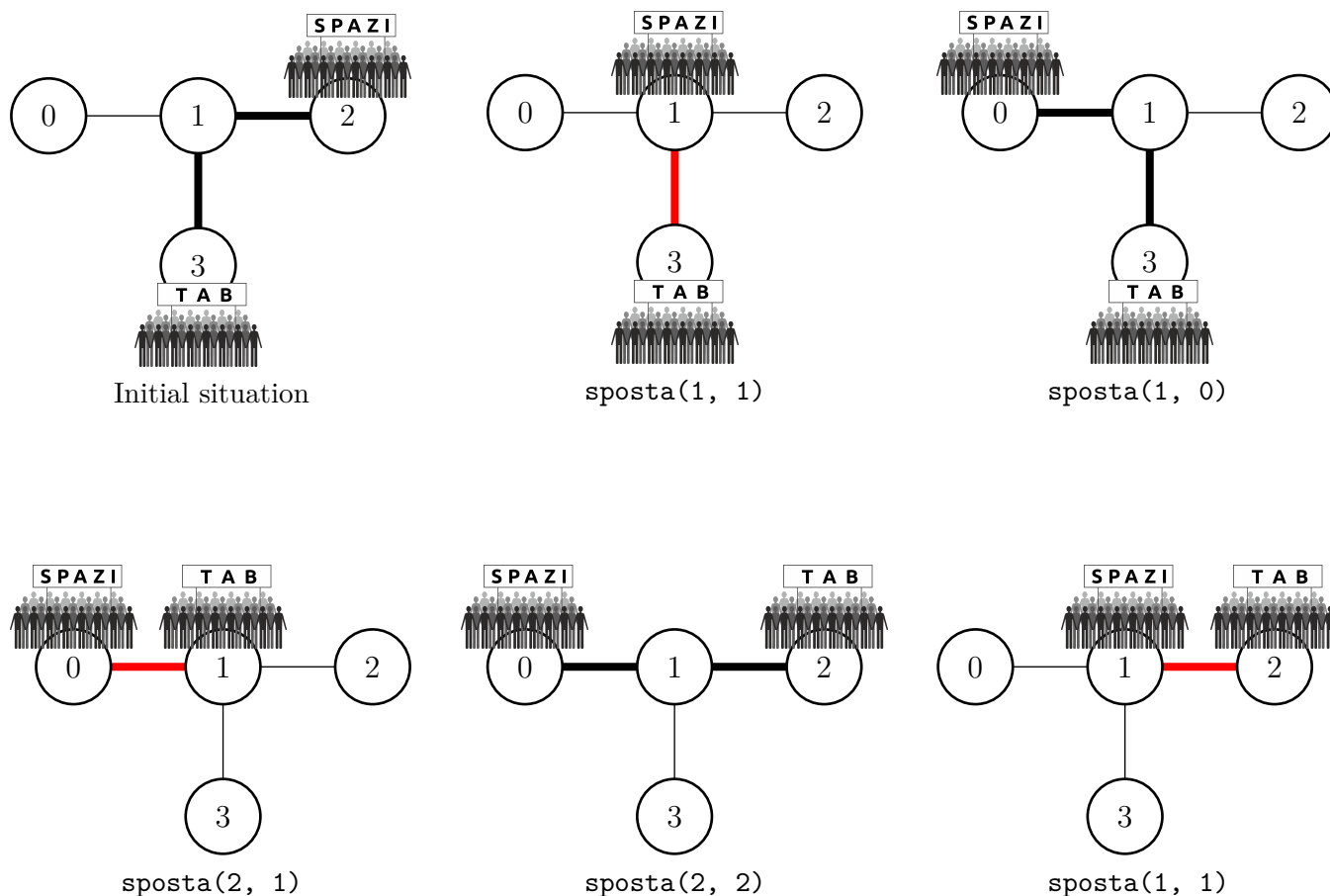
stdin	stdout
10 11 2 5 3 7 0 1 1 2 2 6 4 6 4 5 3 5 3 0 6 7 5 8 7 9 9 8	3
4 3 2 1 3 2 0 1 1 2 1 3	1 1 1 0 2 1 2 2 1 1 99999

Explanation

In the **first example case** the solution computes the optimal margin 3. Indeed, the two groups always keep at least at a distance 3, but you can not do better. One of the possible schedules is as follows:

- at the beginning the distance is 3;
- *Tabs* moves from 3 to 5 (distance 3), then from 5 to 8 (distance 4), then from 8 to 9 (distance 3);
- *Spaces* moves from 2 to 1 (distance 4), then from 1 to 0 (distance 4), then from 0 to 3 (distance 3);
- *Tabs* moves from 9 to 7 (distance 4);
- *Spaces* moves from 3 to 5 (distance 3).

In the **second case example** the solution explicitly computes a schedule that achieves the optimal margin 1. The schedule is shown in the following pictures. You can not get a better margin of 1.



Note that both sample solutions get the maximum score: the first because it calculates the optimal margin, even if it does not show a schedule, the second because it performs an optimal scheduling, even if it returns an incorrect value as margin.