Préparation au concours ACM – TP 2

Christoph Dürr Jill-Jênn Vie

September 25, 2014

Quelques conseils

- Entraînez-vous à identifier les problèmes les plus faciles.
- Lisez bien les contraintes d'affichage : faut-il un retour de ligne entre deux instances consécutives ou même à la fin ? Y a-t-il des deux-points, ou des points à la fin des phrases ?
- Pensez aux cas extrêmes : que se passe-t-il lorsque le graphe est vide ? Lorsqu'il n'y a aucune instance ? Lorsque les entiers donnés en entrée sont négatifs ? Il peut être utile qu'un membre de votre équipe rédige des tests unitaires.
- Aussi, pensez aux bornes sur les entiers : faut-il un `unsigned long long int` (`%lld`) ?
- Évaluez la complexité de votre idée d'algorithme avant de l'implémenter.
- Attention à la mémoire : évitez d'utiliser `malloc` mais plutôt des instructions de type `vector<bool> dejaVu(nbNodes)`, ou n'hésitez pas à faire un tableau de taille suffisamment grande pour accueillir les instances lorsque les contraintes le permettent, quitte à ce que ce soit une variable globale.
- S'il y a plusieurs instances, pensez à réinitialiser toutes les données à chaque nouvelle lecture, notamment si vous utilisez des variables globales.
- Évitez d'utiliser des classes, éventuellement recodez vos propres structures si vous ne voulez pas utiliser `pair<int, int>` ou `tuple`. Un exemple d'implémentation minimale pour accueillir un graphe pondéré :

```
struct Node {
    int id;
    vector<pair<int, int> > neighbours;
};
vector<Node> graph(nb_nodes);
```

- Lorsque vous effectuez des comparaisons entre flottants, songez à conserver une tolérance de `1e-6`.

UVa 100

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

Consider the following algorithm:

1. input n
2. print n
3. while $n \neq 1$:
   (a) if n is odd, set $n := 3n + 1$ else $n := n/2$
   (b) print n

Given the input 22, the following sequence of numbers will be printed 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 \leq n \leq 1000000$ (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed (including the 1). For a given n this is called the *cycle-length* of n. In the example above, the cycle length of 22 is 16.

For any two numbers i and j you are to determine the maximum cycle length over all numbers between i and j.

The Input

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j.

You can assume that no operation overflows a 32-bit integer.

The Output

For each pair of input integers i and j you should output i, j, and the maximum cycle length for integers between and including i and j. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers i and j must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

```
1 10
100 200
201 210
900 1000
```

Sample Output

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

UVa 10116

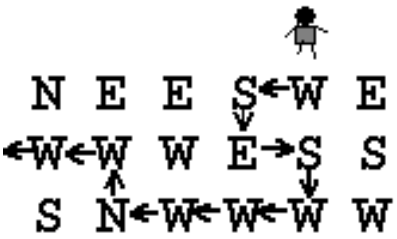
A robot has been programmed to follow the instructions in its path. Instructions for the next direction the robot is to move are laid down in a grid. The possible instructions are

N north (up the page)

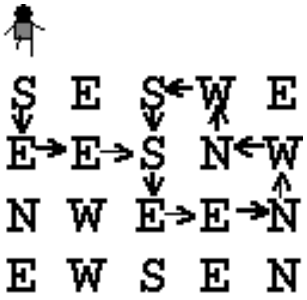
S south (down the page)

E east (to the right on the page)

W west (to the left on the page)



Grid 1



Grid 2

For example, suppose the robot starts on the north (top) side of Grid 1 and starts south (down). The path the robot follows is shown. The robot goes through 10 instructions in the grid before leaving the grid.

Compare what happens in Grid 2: the robot goes through 3 instructions only once, and then starts a loop through 8 instructions, and never exits.

You are to write a program that determines how long it takes a robot to get out of the grid or how the robot loops around.

The Input

There will be one or more grids for robots to navigate. The data for each is in the following form. On the first line are three integers separated by blanks: the number of rows in the grid, the number of columns in the grid, and the number of the column in which the robot enters from the north. The possible entry columns are numbered starting with one at the left. Then come the rows of the direction instructions. Each grid will have at least one and at most 10 rows and columns of instructions. The lines of instructions contain only the characters N,S,E or W with no blanks.

The end of input is indicated by a row containing 0 0 0.

The Output

For each grid in the input there is one line of output. Either the robot follows a certain number of instructions and exits the grid on any one the four sides or else the robot follows the instructions on a certain number of locations once, and then the instructions on some number of locations repeatedly. The sample input below corresponds to the two grids above and illustrates the two forms of output. The word "step" is always immediately followed by "(s)" whether or not the number before it is 1.

Sample Input

```
3 6 5
NEESWE
WWWESS
SNWWWW
4 5 1
SESWE
EESNW
NWEEN
EWSEN
0 0 0
```

Sample Output

```
10 step(s) to exit
3 step(s) before a loop of 8 step(s)
```

UVa 195

You are to write a program that has to generate all possible words from a given set of letters.

Example: Given the word "abc", your program should - by exploring all different combination of the three letters - output the words "abc", "acb", "bac", "bca", "cab" and "cba".

In the word taken from the input file, some letters may appear more than once. For a given word, your program should not produce the same word more than once, and the words should be output in alphabetically ascending order.

The Input

The input file consists of several words. The first line contains a number giving the number of words to follow. Each following line contains one word. A word consists of uppercase or lowercase letters from A to Z. Uppercase and lowercase letters are to be considered different.

The Output

For each word in the input file, the output file should contain all different words that can be generated with the letters of the given word. The words generated from the same input word should be output in alphabetically ascending order. An upper case letter goes before the corresponding lower case letter.

Sample Input

```
aAb
abc
acba
```

Sample Output

```
Aab
Aba
aAb
abA
bAa
baA
abc
acb
bac
bca
cab
cba
aabc
aacb
abac
abca
acab
acba
baac
baca
bcaa
caab
caba
cbaa
```

UVa 10305

John has n tasks to do. Unfortunately, the tasks are not independent and the execution of one task is only possible if other tasks have already been executed.

The Input

The input will consist of several instances of the problem. Each instance begins with a line containing two integers, $1 \leq n \leq 100$ and m. n is the number of tasks (numbered from 1 to n) and m is the number of direct precedence relations between tasks. After this, there will be m lines with two integers i and j, representing the fact that task i must be executed before task j. An instance with $n = m = 0$ will finish the input.

The Output

For each instance, print a line with n integers representing the tasks in a possible order of execution.

Sample Input

```
5 4
1 2
2 3
1 3
1 5
0 0
```

Sample Output

```
1 4 2 5 3
```

UVa 12544 (SWERC 2012)

Bees are one of the most industrious insects. Since they collect nectar and pollen from flowers, they have to rely on the trees in the forest. For simplicity they numbered the n trees from 0 to $n-1$. Instead of roaming around all over the forest, they use a particular list of paths. A path is based on two trees, and they can move either way i.e. from one tree to another in straight line. They don't use paths that are not in their list.

As technology has been improved a lot, they also changed their working strategy. Instead of hovering over all the trees in the forest, they are targeting particular trees, mainly trees with lots of flowers. So, they planned that they will build some new hives in some targeted trees. After that they will only collect their foods from these trees. They will also remove some paths from their list so that they don't have to go to a tree with no hive in it.

Now, they want to build the hives such that if one of the paths in their new list go down (some birds or animals disturbs them in that path) it's still possible to go from any hive to another using the existing paths.

They don't want to choose less than two trees and as hive-building requires a lot of work, they need to keep the number of hives as low as possible. Now you are given the trees with the paths they use, your task is to propose a new bee hive colony for them.

Input

Input starts with an integer T ($T \leqslant 50$), denoting the number of test cases.

Each case starts with a blank line. Next line contains two integers n ($2 \leqslant n \leqslant 500$) and m ($0 \leqslant m \leqslant 20000$), where n denotes the number of trees and m denotes the number of paths. Each of the next m lines contains two integers u v ($0 \leqslant u, v < n$, $u \neq v$) meaning that there is a path between tree u and v. Assume that there can be at most one path between tree u to v, and needless to say that a path will not be given more than once in the input.

Output

For each case, print the case number and the number of beehives in the proposed colony or 'impossible' if its impossible to find such a colony.

NOTE: Dataset is huge. Use faster I/O methods.

Sample Input 1

```
3

3 3
0 1
1 2
2 0

2 1
0 1

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

Sample Output

```
Case 1: 3
Case 2: impossible
Case 3: 3
```