



Czech ACM Student Chapter  
Charles University in Prague  
Slovak University of Technology  
University of Žilina  
Matej Bel University in Banská Bystrica

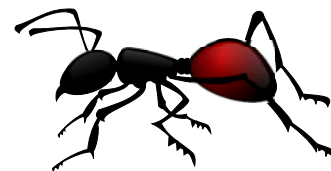
Czech Technical University in Prague  
Technical University of Ostrava  
Pavol Jozef Šafárik University in Košice  
Masaryk University  
University of West Bohemia



## CTU Open Contest 2012

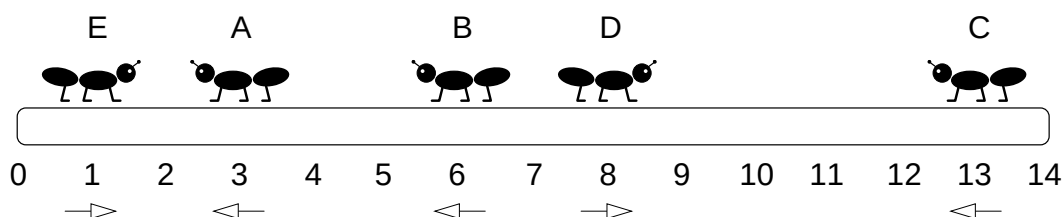
### Andrew the Ant

ants.c, ants.cpp, Ants.java



Andrew the Ant is fascinated by the behavior of his friends. Thousands of them are marching their paths on and on. They can build highly organized ant-hills. Sometimes, however, they act a little bit stupidly.

Recently, Andrew watched his fellow ants marching on top of a long piece of wood. He noticed their behavioral pattern is very simple: Each ant walks slowly forward with a constant speed of 1 cm per second. Whenever it meets another ant, both of them only touch with their antennae and immediately turn around and walk the opposite direction. If an ant comes to the end of the wood, it falls down and does not affect other ants anymore.



The picture above shows an example of moving ants in time 0 s. In one second, the ants E and A meet at position 2 and change their directions. The ant A then meets B in the next 1.5 seconds. At the same time (2.5 seconds after the start), the ants C and D will meet too. All four of them change their directions. In the next 0.5 second (time 3 s), the first ant (E) falls down off the left end, etc.

Your task is to simulate the movement of ants. For simplicity, suppose that the ants have zero size (although the picture could suggest something else).

### Input Specification

The input consists of several scenarios. Each scenario starts with a line containing two integer numbers  $L$  and  $A$ , separated by a space.  $L$  is the length of the wood in cms ( $1 \leq L \leq 99\,999$ ), and  $A$  is the number of ants at the beginning of the simulation ( $1 \leq A \leq L + 1$ ).

Then there are  $A$  lines, each containing a positive integer  $X_i$ , one space, and an uppercase letter. The number ( $0 \leq X_i \leq L$ ) specifies the position of the  $i$ -th ant and the letter its initial direction: either “L” for left (towards zero) or “R” for right. No two ants will start at the same position.

## Output Specification

For each scenario, you should print a single line containing the text “The last ant will fall down in  $T$  seconds - started at  $P$ .”, where  $T$  is the exact time when the last ant (or two) reaches the end of the wood, and  $P$  is the position where that particular ant has originally started in time 0. If two last ants fall down at the same time, print “started at  $P$  and  $Q$ ”, indicating both of their positions,  $P < Q$ .

## Sample Input

```
90000 1
0 R
10 1
0 L
14 5
3 L
6 L
13 L
8 R
1 R
```

## Output for Sample Input

```
The last ant will fall down in 90000 seconds - started at 0.
The last ant will fall down in 0 seconds - started at 0.
The last ant will fall down in 13 seconds - started at 6 and 8.
```

*(The last sample input scenario corresponds to the picture.)*



## Czech ACM Student Chapter

Charles University in Prague  
Slovak University of Technology  
University of Žilina  
Matej Bel University in Banská Bystrica

## Czech Technical University in Prague

Technical University of Ostrava  
Pavol Jozef Šafárik University in Košice  
Masaryk University  
University of West Bohemia



# CTU Open Contest 2012

---

## Software Bugs

bugs.c, bugs.cpp, Bugs.java



Not all bugs are insects. There are also certain kinds of bugs that may become a real nightmare for software developers. You definitely know the situation when a user calls to say “Hey, I’ve found a bug in your program.” Once you have located and removed the bug, another one appears immediately. It is a hard and never-ending process.

Recently, there appeared a promising open-source initiative called the *Bug Preprocessor*. The preprocessor is a program able to find all bugs in your source code and mark them, so they are relatively easy to remove. Your task is to write a program that will remove all marked bugs from the preprocessed source code.

### Input Specification

The input contains several test cases. Each test case starts with a line containing one integer number  $T$  ( $0 \leq T \leq 100\,000$ ), one space and a string  $B$  used by the preprocessor to mark all bugs. The next  $T$  lines then contain the preprocessed source code. All bugs are represented by a case-sensitive string  $B$ .

Each line of the input will be between 0 and 2000000 characters long. The bug marker  $B$  consists of at least 1 and at most 1000 uppercase letters (“A” through “Z”).

### Output Specification

Your program must remove all of the bugs and produce the same result as if the first occurrence of  $B$  on each line was removed repeatedly, until there is no other existing occurrence of  $B$ . Nothing else may be removed, not even spaces.

### Sample Input

```
7 BUG
print "No bugs here..."

void hello() {
BUGBUG
    printfBUG("Hello, world!\n");
}

1 ERR
wriERRERRtelERRn("Hello E-R-R");
```

### Output for Sample Input

```
print "No bugs here..."

void hello() {

    printf("Hello, world!\n");
}

writeln("Hello E-R-R");
```

(This problem has been adapted from SWERC archives.)



## Czech ACM Student Chapter

Charles University in Prague  
Slovak University of Technology  
University of Žilina  
Matej Bel University in Banská Bystrica

## Czech Technical University in Prague

Technical University of Ostrava  
Pavol Jozef Šafárik University in Košice  
Masaryk University  
University of West Bohemia

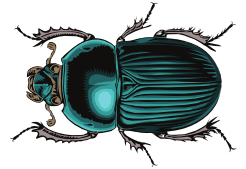


## CTU Open Contest 2012

---

### Charlie the Cockchafer

`cockchaf.c`, `cockchaf.cpp`, `Cockchaf.java`



Charlie knows how to fly. Despite this, whenever Charlie wants to move from one point to another, it becomes a tedious task for him. The main trouble is that Charlie is a cockchafer. And it is a well-known fact that all cockchafers (do not confuse them with cockroaches) are clumsy and slow. Not only they need some time to fly along a straight line, they also spend more time making turns. Knowing their limitations, will you help Charlie to find the quickest route?

#### Input Specification

The input consists of several instances. The first line of each instance contains integers  $N$ ,  $S$  and  $T$  ( $1 \leq N \leq 1000$ ,  $1 \leq S, T \leq 1000$ ), where  $N$  is the number of straight cockchafer flight trajectories (also known as “cockridors”),  $S$  is Charlie’s speed in meters per second, and  $T$  is the speed of him turning in degrees per second. The second line contains six integers ( $0 \leq X_f, Y_f, Z_f, X_t, Y_t, Z_t \leq 10000$ ) indicating the starting point  $(X_f, Y_f, Z_f)$  and the destination  $(X_t, Y_t, Z_t)$ .

Each of the following  $N$  lines contains six integers  $0 \leq X_1, Y_1, Z_1, X_2, Y_2, Z_2 \leq 10000$  stating there is a line segment (cockridor) joining points  $(X_1, Y_1, Z_1)$  with  $(X_2, Y_2, Z_2)$ . You are guaranteed that no internal point of the segment is an endpoint of another segment, and that both the initial and the final positions are endpoints of at least one of the segments. All coordinates are given in meters.

#### Output Specification

For each input instance, print a single line containing one real number  $R$ , giving the shortest time Charlie needs to get from the initial to the final point. Charlie can only move along the whole straight segments, all of them can be used in both directions. The time for any such path is  $R = L/S + D/T$  seconds, where  $L$  is the sum of the lengths of all segments traversed (in meters) and  $D$  is the sum of the angles needed to turn between consecutive segments (in degrees). You can choose the initial and final direction that Charlie is facing, and assume that there always exists at least one path from the initial to the final point.

The answer will be accepted as correct if the difference between  $R$  and the answer computed by the judges is at most 0.001.

### Sample Input

```
1 3 1
0 0 0 10 10 10
0 0 0 10 10 10
2 3 1
0 0 0 10 10 10
0 0 0 0 10 10
10 10 10 0 10 10
```

### Output for Sample Input

```
5.7735
98.0474
```



Czech ACM Student Chapter

Charles University in Prague  
Slovak University of Technology  
University of Žilina  
Matej Bel University in Banská Bystrica

Czech Technical University in Prague

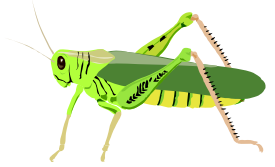
Technical University of Ostrava  
Pavol Jozef Šafárik University in Košice  
Masaryk University  
University of West Bohemia



## CTU Open Contest 2012

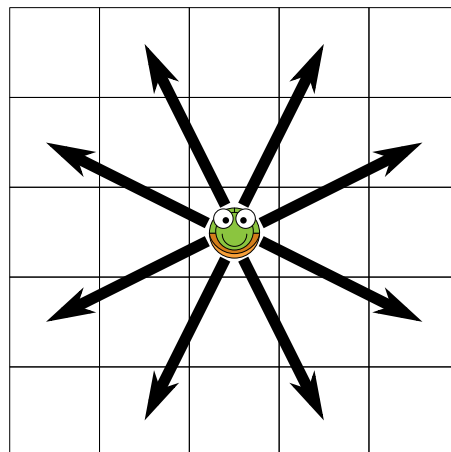
### Gregory the Grasshopper

grasshop.c, grasshop.cpp, Grasshop.java



Gregory is a grasshopper. His favourite food are clover leafs — he can simply never have enough of them. Whenever he spots such a leaf, he wants to eat it as quickly as possible. Gregory is also lazy, so he wants to move to the leaf with minimal effort. Your task is to help him to find the shortest way to a clover leaf.

For simplicity, we will assume that Gregory lives on a rectangular grid consisting of unit squares. As a grasshopper, he prefers to move by jumping (or, more exactly, *hopping*) from one square to the other. Each hop takes him to a square that is in the adjacent row or column in one direction, and two columns or rows away in the other direction. So, his hops resemble the moves of a knight on a chessboard.



### Input Specification

The input consists of several test cases, each of them specified by six integer numbers on one line:  $R$ ,  $C$ ,  $G_R$ ,  $G_C$ ,  $L_R$ , and  $L_C$ .  $R$  and  $C$  specify the size of the grid in unit squares,  $1 \leq R, C \leq 100$ . Gregory cannot hop outside a rectangle of this size, because it would be too dangerous. The values of  $G_R, G_C$  are the coordinates of the square that Gregory is standing on, and  $L_R, L_C$  are the coordinates of the square with the delicious clover leaf. ( $1 \leq G_R, L_R \leq R$ ;  $1 \leq G_C, L_C \leq C$ )

### Output Specification

For each test case, print one integer number — the minimal number of hops that Gregory needs to reach the square with his beloved delicacy. If it is not possible to reach that square at all, print the word “impossible” instead.

### Sample Input

```
10 10 10 10 1 1
2 2 1 1 1 2
8 8 1 1 1 2
```

### Output for Sample Input

```
6
impossible
3
```



## Czech ACM Student Chapter

Charles University in Prague  
Slovak University of Technology  
University of Žilina  
Matej Bel University in Banská Bystrica

## Czech Technical University in Prague

Technical University of Ostrava  
Pavol Jozef Šafárik University in Košice  
Masaryk University  
University of West Bohemia



## CTU Open Contest 2012

### Lisa the Ladybug

ladybug.c, ladybug.cpp, Ladybug.java

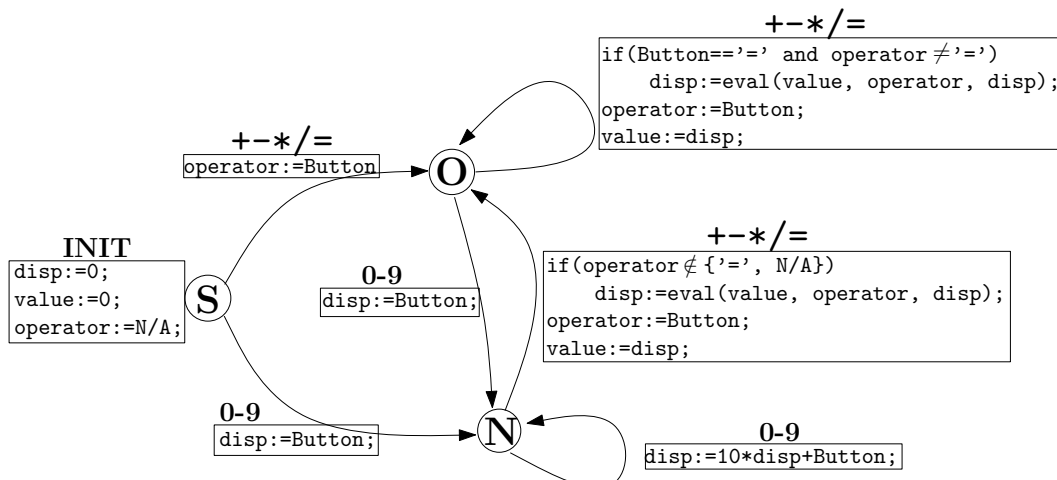


Lisa the Ladybug likes mathematics. She counts her seven dots several times every day. To further improve her skills, she studies at the Arthropoda College in the Meadow. Today, she is preparing for an exam by her professor, Calculon the Centipede. At the exam, the student is expected to show the number of Calculon's legs on the display of a calculator. The calculator was found many years ago among other waste in a landfill. Some of its keys are not working and the display has only three digits. The working keys form a subset of:

0 1 2 3 4 5 6 7 8 9 + - \* / =

Buttons "0", "1", ..., "9" are called the *numeric buttons*, "=" is the *equals button*, and the other buttons are the *operator buttons*. The calculator also has the "C" key to reset it, but only Calculon is allowed to press it, so we will not consider it in this problem. The result of the exam depends on the number of keys pressed to obtain the prescribed value. Lisa wants a perfect score and so she needs to find the shortest possible sequence of key presses.

The work of the calculator is described by the diagram below. The calculator has three registers: *disp* holds the value displayed, *operator* is the last operator or equals button pressed and *value* is a stored value. At the beginning of the exam, the calculator is in the state S and the registers have the values set by the INIT block. Pressing a button *B* changes the state in the direction of the arrow labeled by *B* and the code in the block next to the arrow is executed. The result of the function  $eval(val_1, op, val_2)$  is the result of the operator applied on  $val_1$  and  $val_2$ , that is, for example,  $val_1 + val_2$ .



The description may look complicated, but the calculator works in quite a standard way. There are just a few specifics and details that you should take note of:

- It displays only non-negative integers in the range  $[0, 999]$ . Obtaining a value outside this range anytime during the exam causes an error and failure at the exam.
- Result of a division is an integer; the exact result is always rounded *down*.
- Pressing the equals button two or more times always has the same effect as pressing it once.
- Pressing the equals button right after an operator button results in the operator being evaluated with two equal operands.
- A sequence of operator buttons behaves in the same way as if only the last button of the sequence was pressed.

For example, in the first sample input,  $2\ 2\ /\ 3\ +$  and  $2\ 2\ /\ 3\ /\$  are among the optimal solutions, while  $3\ +\ 2\ +\ 2\ +$  is also a solution, but longer. In the second sample input,  $3\ +\ 2\ +\ 2\ +$  and  $2\ +\ =\ +\ 3\ +$  are among the optimal solutions.

## Input Specification

Each line of the input describes one test case. A line starts with a non-empty list of available buttons (from the allowed set) followed by a single space and the number  $N$  ( $0 \leq N \leq 999$ ) denoting the number of Calculon's legs, that means the number to be displayed.

## Output Specification

For each test case, output a single line with the length of the shortest sequence of buttons, the pressing of which will result in the number  $N$  being displayed by the calculator. If the task cannot be solved, write a line with the word "impossible".

### Sample Input

23+ / 7  
32=+ 7  
3+= 6  
3+= 9  
2 0  
1+ 11  
2+\* 1  
1+ 17  
1+= 17

### Output for Sample Input

5  
6  
3  
6  
0  
2  
impossible  
15  
11



## Czech ACM Student Chapter

Charles University in Prague  
Slovak University of Technology  
University of Žilina  
Matej Bel University in Banská Bystrica

## Czech Technical University in Prague

Technical University of Ostrava  
Pavol Jozef Šafárik University in Košice  
Masaryk University  
University of West Bohemia

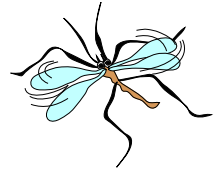


## CTU Open Contest 2012

---

### Mosquito Multiplication

`mosquito.c`, `mosquito.cpp`, `Mosquito.java`



Have you ever wondered why there are so many mosquitos in wet environments? One of the reasons is that adult female mosquitos can lay as many as hundreds of eggs. Even if most of them do not (fortunately!) survive, it is said that a single pair of mosquitos can generate a population of thousands in just a couple of weeks. Let us have a closer look at these numbers.

The mosquito life cycle includes four stages: egg, larva, pupa, and adult. For simplicity, we will make several assumptions that are not quite true in the nature: The egg stage lasts less than one day, and all the other stages are one week long. Each mosquito lives as a larva for the first week, the second week “hibernates” in the form of a pupa, and finally, the third week lives as an adult mosquito. At the end of its three-week life, each mosquito lays eggs and dies.

To simplify things even further, we will assume that the transformation from one life stage into another always happens on Sunday. Each Sunday, the following things happen:

- An adult mosquito lays  $E$  eggs and dies. Within a day, one larva hatches from each egg.
- Some of the larvae hatched from egg the last Sunday were not strong enough and died or got eaten. Only every  $R$ -th larva will transform into a pupa on Sunday.
- An adult mosquito emerges from every  $S$ -th pupa, all other pupae dry.

For example, if there are 5 pupae and every 3-rd of them survives, there will be 1 mosquito left. Out of 6 pupae, two adult mosquitos emerge.

In the first week, there are  $M$  mosquitos,  $P$  pupae and  $L$  larvae. Calculate how many mosquitos will there be after  $N$  Sunday transformations. Of course, we are counting only living *adult* mosquitos, not the dry dead bodies.

### Input Specification

The input consists of several instances, each instance per one line. Each line contains seven integers  $M$ ,  $P$ ,  $L$ ,  $E$ ,  $R$ ,  $S$ ,  $N$  separated by space.  $M$ ,  $P$  and  $L$  are the numbers of mosquitos, pupae, and larvae, respectively, in the first week. You may assume that  $0 \leq M, P, L \leq 100\,000$ ,  $0 \leq E \leq 100$ ,  $1 \leq R, S \leq 10$ , and  $1 \leq N \leq 1000$ .  $E$  is the number of eggs laid by one mosquito,  $R$  and  $S$  specify the survival rates of larvae and pupae, and  $N$  is the number of weeks.

## Output Specification

For each input instance, output a single line containing an integer number  $C$ , giving the count of mosquitos after the  $N$ -th Sunday.

You may assume that the number of mosquitos during each of the first  $N$  weeks will not exceed 1 000 000.

## Sample Input

```
10 20 40 4 2 2 10
144 55 8 0 1 9 4
10 10 10 2 3 2 6
10 20 40 86 9 9 999
```

## Output for Sample Input

```
10
0
1
10
```



## Czech ACM Student Chapter

Charles University in Prague  
Slovak University of Technology  
University of Žilina  
Matej Bel University in Banská Bystrica

## Czech Technical University in Prague

Technical University of Ostrava  
Pavol Jozef Šafárik University in Košice  
Masaryk University  
University of West Bohemia

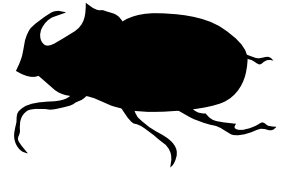


# CTU Open Contest 2012

---

## Rhinoceros Beetle

`rhino.c`, `rhino.cpp`, `Rhino.java`



Given a big strength, Rhinoceros Beetles have a natural tendency to fight. Because of that, they are even used for gambling fights in some countries, especially in Asia. Two male beetles fight against each other and the people watch them and bet which one would win. Recently, animal protection groups protested against such gambling events. Therefore, many gamblers started to play different games instead, like Texas Hold'em Poker. In this game, it may be hard to pick the winner among many players. Since a big tournament called the Arthropoda Card Marathon will be organized soon, please write a program to decide which hand is the strongest.

### Input Specification

The input consists of several instances. The first line of each instance contains one integer  $N$  ( $1 \leq N \leq 10$ ), the number of players. The second line contains exactly five cards separated by a space — these are the Texas Hold'em "community" cards shared by all the players. Each of the following  $N$  lines contains two cards separated by a space — the cards held by individual players, the first such line being cards of Player 1, second line the cards of Player 2, etc.

Each card is described by exactly 2 characters, first character being the card *rank*, one of {"2", "3", "4", "5", "6", "7", "8", "9", "X", "J", "Q", "K", "A"}, where "X" stands for the 10, second character being the card *suit* (sometimes called *color*), one of {"c", "d", "h", "s"} for Clubs, Diamonds, Hearts, or Spades. For example, "Xh" is the Ten of Hearts and "As" is the Ace of Spades.

### Output Specification

For each input instance, output one single line containing numbers of all winners in ascending order, separated by a space. More specifically, output an ordered ascending sequence of indexes of players that hold the strongest hand for that given configuration of cards.

### Rules of Texas Hold'em Poker

Five cards (the community cards) are shared by all the players and each player holds 2 other cards that may be combined with the community cards, meaning that each player has 7 cards available, out of which 5 cards are chosen to make the best possible hand.

Note that no more than 5 cards can be counted as a hand under any circumstances. For instance, if all players make the same hand using all 5 community cards, the two cards they are holding do not count.

The hands in Texas Hold'em Poker are the usual poker hands ("figures"). Listed from strongest to the weakest figure, these are:

**Royal Flush**

a Flush that is also a Straight

**Poker**

also known as Four of a kind — four cards of the same rank

**Full House**

A Three and a Pair together

**Flush**

five cards of the same suit (color)

**Straight**

5 consecutive cards (such as 7,8,9,X,J), where Ace may be the highest or lowest card, but not both — for example, A,2,3,4,5 or X,J,Q,K,A are Straights, but Q,K,A,2,3 is not

**Three**

3 cards of the same rank (value)

**Two pairs**

two pairs of cards of the same rank

**Pair**

two cards of the same rank

**One card**

anything else (no two cards of the same rank, and no Straight)

If two or more players have the same figure, the following tie-breaker rules are applied, in order:

1. With Full House, the highest card of the Three wins (3,3,Q,Q,Q > 9,9,9,K,K).
2. If Ace plays low in a Straight, it is weaker than other Straights (5,6,7,8,9 > A,2,3,4,5).
3. The highest card of the figure wins (2,2,4,K,K > X,X,Q,Q,A), or the other cards of the figure if necessary (4,4,5,K,K > 3,3,K,K,A).
4. Otherwise, the highest card that is not used in the figure decides (2,2,6,X,A > 2,2,X,Q,K; 2,2,6,X,A > 2,2,3,4,A; and 3,4,7,8,A > 8,X,J,Q,K).

If no tie-breaker can be used, both hands have the same strength. Specifically, all four suits have the same strength.

**Sample Input**

3  
2s 9c Ad 4h Xs  
As Ac  
9h 7h  
Xh 6h  
2  
3s 4s 5s As Ad  
6h 7h  
7d 6c

**Output for Sample Input**

1  
1 2



## Czech ACM Student Chapter

Charles University in Prague  
Slovak University of Technology  
University of Žilina  
Matej Bel University in Banská Bystrica

## Czech Technical University in Prague

Technical University of Ostrava  
Pavol Jozef Šafárik University in Košice  
Masaryk University  
University of West Bohemia



## CTU Open Contest 2012

---

### Simon the Spider

`spider.c`, `spider.cpp`, `Spider.java`



In the *Mosquito* problem, you have seen that the potential reproductive capabilities of insects are very high. Fortunately for us humans, the insects also have natural enemies that help to reduce their population. Spiders belong among the best known insect predators, so there is no wonder we included them in this problem set.

Simon the Spider ate many insects this summer and got fat. The threads of his web will soon be too thin to carry him and so he needs to reinforce them. Since fat spiders are also lazy, Simon wants to use as little material as possible — well, you know that spiders have to produce the material for building webs by themselves. Finally, he decided to reinforce only some of the threads, but it is important that every node of his web must be reachable over the reinforced links.

Additionally, Simon plans to spend his free time on one of the reinforced links and he wants this one link to be long. Therefore, when calculating the total length of all reinforced links, the length of the longest reinforced link will be *subtracted* instead of added. Help Simon to decide which links should be reinforced to have the lowest possible total length under these circumstances.

### Input Specification

The input consists of descriptions of several webs. The first line of each description contains two numbers: the number  $N$  ( $2 \leq N \leq 2000$ ) of nodes in the web and the number  $M$  ( $0 \leq M \leq 1\,000\,000$ ) of the links between pairs of nodes. Each of the following  $M$  lines describes one link. The description of each link contains three positive integers  $u_i, v_i, \ell_i$ , where  $u_i$  and  $v_i$  ( $1 \leq u_i, v_i \leq N$  and  $u_i \neq v_i$ ) are the two nodes connected by the link and  $\ell_i$  is its length ( $1 \leq \ell_i \leq 100\,000$ ).

### Output Specification

For each web, print a single line with the minimum possible total length of reinforced links under all given conditions. Remember that the total length is the sum of the lengths of all reinforced links *minus* twice the length of the longest reinforced link.

If it is not possible to reach every node from every other node through a sequence of links, then print “**disconnected**” instead of the cost.

### Sample Input

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

### Output for Sample Input

```
disconnected
-1
```