## 9.6 Problems

### *9.6.1 Bicoloring*

**PC/UVa IDs:** 110901/10004, **Popularity:** A, **Success rate:** high **Level:** 1

The *four-color theorem* states that every planar map can be colored using only four colors in such a way that no region is colored using the same color as a neighbor. After being open for over 100 years, the theorem was proven in 1976 with the assistance of a computer.

Here you are asked to solve a simpler problem. Decide whether a given connected graph can be bicolored, i.e., can the vertices be painted red and black such that no two adjacent vertices have the same color.

To simplify the problem, you can assume the graph will be connected, undirected, and not contain self-loops (i.e., edges from a vertex to itself).

#### *Input*

The input consists of several test cases. Each test case starts with a line containing the number of vertices $n$, where $1 < n < 200$. Each vertex is labeled by a number from 0 to $n-1$. The second line contains the number of edges $l$. After this, $l$ lines follow, each containing two vertex numbers specifying an edge.

An input with $n = 0$ marks the end of the input and is not to be processed.

#### *Output*

Decide whether the input graph can be bicolored, and print the result as shown below.

#### *Sample Input*

```
3
3
0 1
1 2
2 0
9
8
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0
```

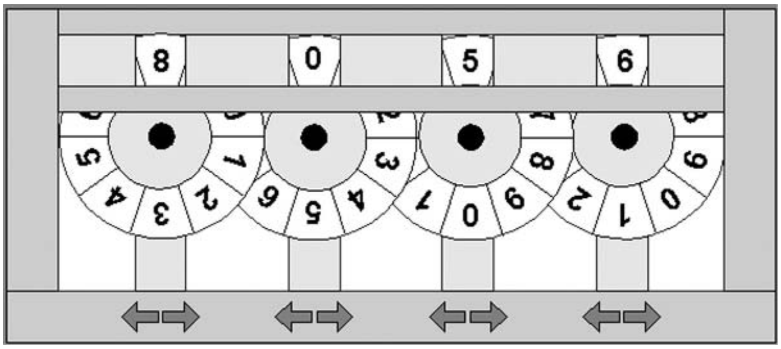
#### *Sample Output*

```
NOT BICOLORABLE.
BICOLORABLE.
```

## *9.6.2 Playing With Wheels*

**PC/UVa IDs:** 110902/10067, **Popularity:** C, **Success rate:** average **Level:** 2

Consider the following mathematical machine. Digits ranging from 0 to 9 are printed consecutively (clockwise) on the periphery of each wheel. The topmost digits of the wheels form a four-digit integer. For example, in the following figure the wheels form the integer 8,056. Each wheel has two buttons associated with it. Pressing the button marked with a *left arrow* rotates the wheel one digit in the clockwise direction and pressing the one marked with the *right arrow* rotates it by one digit in the opposite direction.



We start with an initial configuration of the wheels, with the topmost digits forming the integer $S_1S_2S_3S_4$. You will be given a set of $n$ forbidden configurations $F_{i_1}F_{i_2}F_{i_3}F_{i_4}$ $(1 \leq i \leq n)$ and a target configuration $T_1T_2T_3T_4$. Your job is to write a program to calculate the minimum number of button presses required to transform the initial configuration to the target configuration without passing through a forbidden one.

### *Input*

The first line of the input contains an integer $N$ giving the number of test cases. A blank line then follows.

The first line of each test case contains the initial configuration of the wheels, specified by four digits. Two consecutive digits are separated by a space. The next line contains the target configuration. The third line contains an integer $n$ giving the number of forbidden configurations. Each of the following $n$ lines contains a forbidden configuration. There is a blank line between two consecutive input sets.

### *Output*

For each test case in the input print a line containing the minimum number of button presses required. If the target configuration is not reachable print “`-1`”.

*Sample Input*

```
2

8 0 5 6
6 5 0 8
5
8 0 5 7
8 0 4 7
5 5 0 8
7 5 0 8
6 4 0 8

0 0 0 0
5 3 1 7
8
0 0 0 1
0 0 0 9
0 0 1 0
0 0 9 0
0 1 0 0
0 9 0 0
1 0 0 0
9 0 0 0
```

*Sample Output*

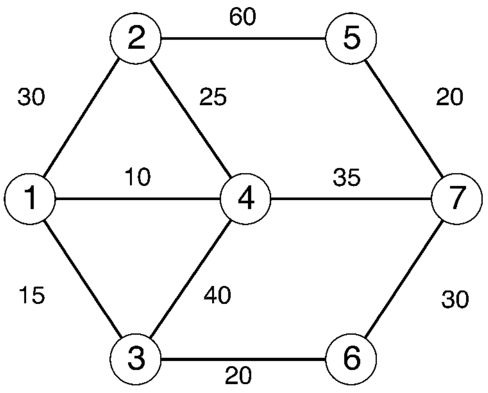
```
14
-1
```

## *9.6.3 The Tourist Guide*

**PC/UVa IDs:** 110903/10099, **Popularity:** B, **Success rate:** average **Level:** 3

Mr. G. works as a tourist guide in Bangladesh. His current assignment is to show a group of tourists a distant city. As in all countries, certain pairs of cities are connected by two-way roads. Each pair of neighboring cities has a bus service that runs only between those two cities and uses the road that directly connects them. Each bus service has a particular limit on the maximum number of passengers it can carry. Mr. G. has a map showing the cities and the roads connecting them, as well as the service limit for each each bus service.

It is not always possible for him to take all the tourists to the destination city in a single trip. For example, consider the following road map of seven cities, where the edges represent roads and the number written on each edge indicates the passenger limit of the associated bus service.



It will take at least five trips for Mr. G. to take 99 tourists from city 1 to city 7, since he has to ride the bus with each group. The best route to take is 1 - 2 - 4 - 7.

Help Mr. G. find the route to take all his tourists to the destination city in the minimum number of trips.

### *Input*

The input will contain one or more test cases. The first line of each test case will contain two integers: $N$ ($N \leq 100$) and $R$, representing the number of cities and the number of road segments, respectively. Each of the next $R$ lines will contain three integers ($C_1$, $C_2$, and $P$) where $C_1$ and $C_2$ are the city numbers and $P$ ($P > 1$) is the maximum number of passengers that can be carried by the bus service between the two cities. City numbers are positive integers ranging from 1 to $N$. The $(R+1)$th line will contain three integers ($S$, $D$, and $T$) representing, respectively, the starting city, the destination city, and the number of tourists to be guided.

The input will end with two zeros for $N$ and $R$.

### *Output*

For each test case in the input, first output the scenario number and then the minimum number of trips required for this case on a separate line. Print a blank line after the output for each test case.

### *Sample Input*

```
7 10
1 2 30
1 3 15
1 4 10
2 4 25
2 5 60
3 4 40
3 6 20
4 7 35
5 7 20
6 7 30
1 7 99
0 0
```

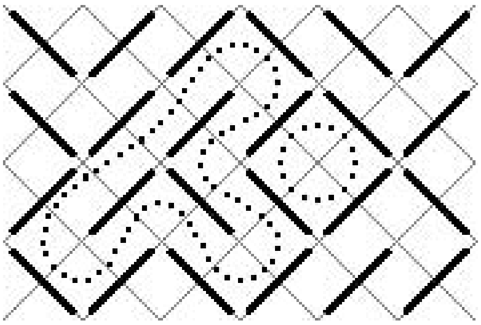
### *Sample Output*

```
Scenario #1
Minimum Number of Trips = 5
```

## *9.6.4 Slash Maze*

**PC/UVa IDs:** 110904/705, **Popularity:** B, **Success rate:** average **Level:** 2

By filling a rectangle with slashes (/) and backslashes (\), you can generate nice little mazes. Here is an example:



As you can see, paths in the maze cannot branch, so the whole maze contains only (1) cyclic paths and (2) paths entering somewhere and leaving somewhere else. We are only interested in the cycles. There are exactly two of them in our example.

Your task is to write a program that counts the cycles and finds the length of the longest one. The length is defined as the number of small squares the cycle consists of (the ones bordered by gray lines in the picture). In this example, the long cycle has length 16 and the short one length 4.

### *Input*

The input contains several maze descriptions. Each description begins with one line containing two integers $w$ and $h$ ($1 \leq w, h \leq 75$), representing the width and the height of the maze. The next $h$ lines describe the maze itself and contain $w$ characters each; all of these characters will be either "/" or "\".

The input is terminated by a test case beginning with $w = h = 0$. This case should not be processed.

### *Output*

For each maze, first output the line "`Maze #`$n$`:`", where $n$ is the number of the maze. Then, output the line "$k$ `Cycles; the longest has length` $l$`.`", where $k$ is the number of cycles in the maze and $l$ the length of the longest of the cycles. If the maze is acyclic, output "`There are no cycles.`"

Output a blank line after each test case.

*Sample Input*

```
6 4
\//\\/
\///\/
//\\/\
\/\///
3 3
///
\//
\\\
0 0
```

*Sample Output*

```
Maze #1:
2 Cycles; the longest has length 16.

Maze #2:
There are no cycles.
```

### *9.6.5 Edit Step Ladders*

**PC/UVa IDs:** 110905/10029, **Popularity:** B, **Success rate:** low **Level:** 3

An *edit step* is a transformation from one word $x$ to another word $y$ such that $x$ and $y$ are words in the dictionary, and $x$ can be transformed to $y$ by adding, deleting, or changing one letter. The transformations from *dig* to *dog* and from *dog* to *do* are both edit steps. An *edit step ladder* is a lexicographically ordered sequence of words $w_1, w_2, \ldots, w_n$ such that the transformation from $w_i$ to $w_{i+1}$ is an edit step for all $i$ from 1 to $n-1$.

For a given dictionary, you are to compute the length of the longest edit step ladder.

#### *Input*

The input to your program consists of the dictionary: a set of lowercase words in lexicographic order at one word per line. No word exceeds 16 letters and there are at most 25,000 words in the dictionary.

#### *Output*

The output consists of a single integer, the number of words in the longest edit step ladder.

#### *Sample Input*

```
cat
dig
dog
fig
fin
fine
fog
log
wine
```

#### *Sample Output*

```
5
```

## *9.6.6 Tower of Cubes*

**PC/UVa IDs:** 110906/10051, **Popularity:** C, **Success rate:** high **Level:** 3

You are given $N$ colorful cubes, each having a distinct weight. Cubes are not monochromatic – indeed, every face of a cube is colored with a different color. Your job is to build the tallest possible tower of cubes subject to the restrictions that (1) we never put a heavier cube on a lighter one, and (2) the bottom face of every cube (except the bottom one) must have the same color as the top face of the cube below it.

### *Input*

The input may contain several test cases. The first line of each test case contains an integer $N$ ($1 \leq N \leq 500$) indicating the number of cubes you are given. The $i$th of the next $N$ lines contains the description of the $i$th cube. A cube is described by giving the colors of its faces in the following order: front, back, left, right, top, and bottom face. For your convenience colors are identified by integers in the range 1 to 100. You may assume that cubes are given in increasing order of their weights; that is, cube 1 is the lightest and cube $N$ is the heaviest.

The input terminates with a value `0` for $N$.

### *Output*

For each case, start by printing the test case number on its own line as shown in the sample output. On the next line, print the number of cubes in the tallest possible tower. The next line describes the cubes in your tower from top to bottom with one description per line. Each description gives the serial number of this cube in the input, followed by a single whitespace character and then the identification string (`front`, `back`, `left`, `right`, `top`, or `bottom` of the top face of the cube in the tower. There may be multiple solutions, but any one of them is acceptable.

Print a blank line between two successive test cases.

### *Sample Input*

```
3
1 2 2 2 1 2
3 3 3 3 3 3
3 2 1 1 1 1
10
1 5 10 3 6 5
2 6 7 3 6 9
5 7 3 2 1 9
1 3 3 5 8 10
6 6 2 2 4 4
1 2 3 4 5 6
```

```
10 9 8 7 6 5
6 1 2 3 4 7
1 2 3 3 2 1
3 2 1 1 2 3
0
```

*Sample Output*

```
Case #1
2
2 front
3 front

Case #2
8
1 bottom
2 back
3 right
4 left
6 top
8 front
9 front
10 top
```

## *9.6.7 From Dusk Till Dawn*

**PC/UVa IDs:** 110907/10187, **Popularity:** B, **Success rate:** average **Level:** 3

Vladimir has white skin, very long teeth and is 600 years old, but this is no problem because Vladimir is a vampire. Vladimir has never had any problems with being a vampire. In fact, he is a successful doctor who always takes the night shift and so has made many friends among his colleagues. He has an impressive trick which he loves to show at dinner parties: he can tell blood group by taste. Vladimir loves to travel, but being a vampire he must overcome three problems.

1. He can only travel by train, because he must take his coffin with him. Fortunately he can always travel first class because he has made a lot of money through long term investments.
2. He can only travel from dusk till dawn, namely, from 6 P.M. to 6 A.M. During the day he has must stay inside a train station.
3. He has to take something to eat with him. He needs one litre of blood per day, which he drinks at noon (12:00) inside his coffin.

Help Vladimir to find the shortest route between two given cities, so that he can travel with the minimum amount of blood. If he takes too much with him, people ask him funny questions like, "What are you doing with all that blood?"

### *Input*

The first line of the input will contain a single number telling you the number of test cases.

Each test case specification begins with a single number telling you how many route specifications follow. Each route specification consists of the names of two cities, the departure time from city one, and the total traveling time, with all times in hours. Remember, Vladimir cannot use routes departing earlier than 18:00 or arriving later than 6:00.

There will be at most 100 cities and less than 1,000 connections. No route takes less than 1 hour or more than 24 hours, but Vladimir can use only routes within the 12 hours travel time from dusk till dawn.

All city names are at most 32 characters long. The last line contains two city names. The first is Vladimir's start city; the second is Vladimir's destination.

### *Output*

For each test case you should output the number of the test case followed by "`Vladimir needs # litre(s) of blood.`" or "`There is no route Vladimir can take.`"

### *Sample Input*

```
2
3
Ulm Muenchen 17 2
Ulm Muenchen 19 12
Ulm Muenchen 5 2
Ulm Muenchen
10
Lugoj Sibiu 12 6
Lugoj Sibiu 18 6
Lugoj Sibiu 24 5
Lugoj Medias 22 8
Lugoj Medias 18 8
Lugoj Reghin 17 4
Sibiu Reghin 19 9
Sibiu Medias 20 3
Reghin Medias 20 4
Reghin Bacau 24 6
Lugoj Bacau
```

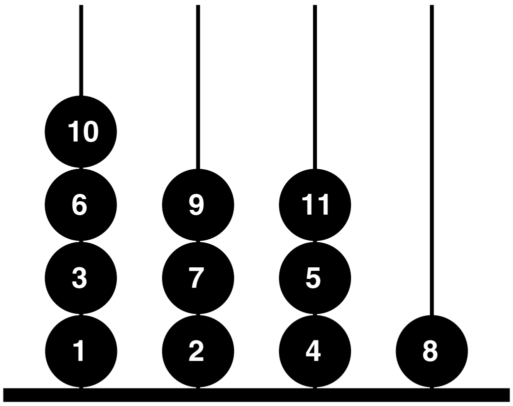
### *Sample Output*

```
Test Case 1.
There is no route Vladimir can take.
Test Case 2.
Vladimir needs 2 litre(s) of blood.
```

## *9.6.8 Hanoi Tower Troubles Again!*

**PC/UVa IDs:** 110908/10276, **Popularity:** B, **Success rate:** high **Level:** 3

There are many interesting variations on the Tower of Hanoi problem. This version consists of $N$ pegs and one ball containing each number from $1, 2, 3, \ldots, \infty$. Whenever the sum of the numbers on two balls is *not* a perfect square (i.e., $c^2$ for some integer $c$), they will repel each other with such force that they can never touch each other.



The player must place balls on the pegs one by one, in order of increasing ball number (i.e., first ball 1, then ball 2, then ball 3...). The game ends where there is no non-repelling move.

The goal is to place as many balls on the pegs as possible. The figure above gives a best possible result for 4 pegs.

### *Input*

The first line of the input contains a single integer $T$ indicating the number of test cases ($1 \leq T \leq 50$). Each test case contains a single integer $N$ ($1 \leq N \leq 50$) indicating the number of pegs available.

### *Output*

For each test case, print a line containing an integer indicating the maximum number of balls that can be placed. Print "`-1`" if an infinite number of balls can be placed.

*Sample Input*

```
2
4
25
```

*Sample Output*

```
11
337
```

## 9.7 Hints

9.1 Can we color the graph during a single traversal?

9.2 What is the graph underlying this problem?

9.3 Can we reduce this problem to connectivity testing?

9.4 Does it pay to represent the graph explicitly, or just work on the matrix of slashes?

9.5 What is the graph underlying this problem?

9.6 Can we define a directed graph on the cubes such that the desired tower is a path in the graph?

9.7 Can this be represented as an *unweighted* graph problem for BFS?

9.8 Can the constraints be usefully modeled using a DAG?