29th ACM International Collegiate Programming Contest, 2004-2005
Asia Region, Tehran Site
Sharif University of Technology
1–3 Dec. 2004

Sponsored by

Problem A (Program filename: A.cpp, A.dpr, or A.java)

IOI Photos

Shaborz, Hoidin, Alssein, and Ayan, members of the Olandican IOI team attended the Fall semester classes the same day they returned from IOI, Athens 2004. During their stay in Athens, they took several pictures in different places and occasions like Hydra island, opening ceremony, closing award ceremony, and city of Athens. But, being excited with their first university experience, they forgot about the pictures until the midterm recess, which has coincided with the ACM Regional Contest days. They now want to make prints of the pictures and each of them makes his own IOI album.

There are several negative rolls, and each contains photos of just a single place or occasion. There may be more than one roll, containing pictures from the same place or occasion. Each roll may have 36 negatives, numbered from 1 to 36. The team members and their friends want to order photo prints. Shaborz is to collect all orders and collects a fixed amount of money per each photo print. He makes a deal with a photo printing shop as follows and saves a good sum of money for himself. Shaborz pays S Rials for each single print, but printing all photos of a single role costs him R Rials, and printing all photos from all rolls in one order costs A Rials. Shaborz is provided with a list of orders, and you are to minimize the overall printing cost. Note that to have the minimum overall cost, Shaborz is allowed to print more photos than required.

Input (filename: A.in)

The first line of the input contains a single integer t ($1 \le t \le 20$) which is the number of test cases in the input. Each test case starts with one line containing four integers: N ($1 \le N \le 100$), the number of orders, S, R, and A, the costs of a single print, all prints from one roll, and all prints of all rolls respectively. Then follows N lines, each representing an order from one of the clients (team members and their friends). An order line contains a number of items separated by blank characters. Each item is of the form *PlaceName* : *RollNo* : *FromPhoto..ToPhoto*. *PlaceName* is the name of a place which is a string of at most 100 characters (case sensitive). *RollNo* specifies the desired roll among several rolls for the *PlaceName* and is between 1 and 10 inclusive. *FromPhoto* and *ToPhoto* are two numbers specifying the range of photos to be printed from the specified roll ($1 \le$ *FromPhoto* $\le$ *ToPhoto* ≤ 36). You may assume there are at most 20 places. If there is only a single photo required from a roll, the format may be simplified as *PlaceName* : *RollNo* : *PhotoNo*. All costs are non-negative integers.

Output (filename: A.out)

For each test case, there should be one line containing one integer indicating the minimum cost for printing all photos of the original order set.

Sample Input

```
1
2 15 100 400
Hydra:2:1..3 Athens:1:12
Delphi:1:4..5 Athens:3:20
```

Sample Output

```
105
```

Problem B (Program filename: B.cpp, B.dpr, or B.java)

ACM: Vice City

"Tommy, there will be a programming contest here in Vice City. One of the coaches has stolen a copy of the problem set. The chief judge wants it back. Take out the coach guy at his hotel and return the problems back. The address is taped under the phone. Do it now!"

Not a tough job for you, Tommy Vercetti! Getting the mission at the pay phone, you must head off the coach at WK Charriot Hotel before he leaves. You have to get there fast! Get there very fast indeed! Unfortunately, the vehicle you start with may not run fast enough. But there are some fixed locations in Vice City at which you can find certain vehicles, like Diaz's Mansion where you can find an Infernus. This way, you may change your vehicle on your way to hotel several times. For example, in the first sample input, you ride from `PayPhone` to `CarShowRoom` on a `PCJ600` and drive the rest of the path in a `HotRingRacer`. Don't forget that it takes one minute each time you change your vehicle.

You are given the names of these locations in the city and the distances between each pair. At each location you can find a certain vehicle anytime you get there. Knowing the top speed of each vehicle, you want to find out the minimum time in which you can reach the hotel. For the sake of simplicity, assume that you always drive at top speed of your vehicle.

Input (filename: B.in)

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. Each test case has three parts. The first part consists of m lines ($1 \leq m \leq 100$) of the form *vehicle speed* where *vehicle* is the unique name of a vehicle and *speed* is a positive integer giving the top speed of the vehicle measured in Km/h.

The next part of the test case identifies the locations in the city and is separated from the first part by exactly one blank line. It consists of n lines ($2 \leq n \leq 500$) of the form *location vehicle* where *location* is the unique name of a location in the city and *vehicle* is the name of the vehicle available in that location. The list of locations always includes the starting location `PayPhone` and the destination `WKCharriot`.

The third part of the test case identifies the roads between locations and is separated from the previous part by exactly one blank line. It consists of several lines of the form loc_1 loc_2 *distance* indicating there is a (two-way) road of length *distance* between the locations loc_1 and loc_2. Distances are expressed in kilometers and are positive integers. The test case is terminated by a line containing a single asterisk character (`*`).

All names (for vehicles and locations) are strings of at most 100 letters and digits with no space characters and are considered case sensitive. Items in an input line are separated by one or more space characters. Also, there may be arbitrary leading or trailing blanks except in empty lines used as separators.

Output (filename: B.out)

For each test case, there is one line in the output containing the minimum time (in minutes) you need to travel from `PayPhone` to `WKCharriot`, or the word `UNREACHABLE` if the destination is unreachable from the starting point. Print the results as numbers with exactly three decimal digits after decimal point. That is, the possible decimal digits after the third one should be ignored, and if there are less than three digits after decimal point, zero digits should be printed for missing digits.

Sample Input

```
2
Infernus     280
Cheetah      285
PCJ600       250
Stallion     180
HotRingRacer 300

Mansion         Infernus
CarShowRoom     HotRingRacer
VicePort        Cheetah
NorthPointMall  Infernus
PayPhone        PCJ600
WKCharriot      Stallion

PayPhone       CarShowRoom    10
PayPhone       VicePort       15
VicePort       WKCharriot     20
CarShowRoom    Mansion        15
Mansion        WKCharriot     15
Mansion        NorthPointMall 5
NorthPointMall WKCharriot     5
*
Caddy        80
MrWhoopie    60
Stretch      120
CubanHermes  160
Voodoo       170

CherryPoppy  MrWhoopie
Mansion      Stretch
PayPhone     CubanHermes
LittleHaiti  Voodoo
WKCharriot   Caddy

PayPhone      CherryPoppy    10
CherryPoppy   LittleHaiti    15
Mansion       WKCharriot     20
*
```

Sample Output

```
8.400
UNREACHABLE
```

Problem C (Program filename: C.cpp, C.dpr, or C.java)

Fixing Codes

A binary string is a string of characters from the set $\{0, 1\}$. A code is a multiset of binary strings (i.e., a string can be repeated arbitrary number of times). A fixed code is a code that none of its strings is a prefix of another string. We say that a code $A = \{a_1, a_2, \ldots, a_n\}$ is extended to code $B = \{b_1, b_2, \ldots, b_n\}$ if and only if for $1 \leq i \leq n$, a_i be a prefix of b_i. The cost of this extension is $\Sigma_{i=1}^{n} |b_i| - |a_i|$ where $|a_i|$ is the number of characters in a_i.

For this problem you are given a fixed code C, and a new binary string s. You have to find the minimum needed cost to extend the code $C \cup \{s\}$ into a fixed code. In other words, you are to append the minimum number of bits to zero or more codes in $C \cup \{s\}$ to make it a fixed code.

Input (filename: C.in)

The first line of the input contains a single integer t $(1 \leq t \leq 20)$ which is the number of test cases in the input. For $1 \leq i \leq t$ the line $i + 1$ consists a nonzero number of binary strings. The number of binary strings in each line is at most 41, and the length of each binary string is no more than 40 characters. The last string in each line stands for the new incoming string s and the other strings in that line make the fixed code of the relevant test case.

Output (filename: C.out)

The output consists m lines. The solution to ith test case should be written in the line i of the output.

Sample Input

```
2
001 01 00
000 001 010 011 100 101 110 1
```

Sample Output

```
1
2
```

Problem D (Program filename: D.cpp, D.dpr, or D.java)

Wiping Words

In this problem, you are given a paragraph of text in terms of a sequence of lines. Each lines contains a number of words which are sequences of lowercase and uppercase letters and are separated by either blank characters or asterisks. A word is wiped out if for each character in that word, there is no letter or asterisk character in the same position in the next line, or the word appears in the last line of the input. If such a case happens, all occurrences of that word in the text is converted to blanks independent of the corresponding characters in the next line. Note that the asterisks and black characters never disappear. Also, note that the words are considered case-sensitive. Write a program to read a sequence of lines described above and wipe out as many word as it can iteratively.

Input (filename: D.in)

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. Each test case contains a sequence of lines containing characters A..Z, a..z, blank and asterisk (`*`). After each test case, there is a line containing single hash character (`#`) which is not a part of the lines you must consider in your algorithm.

Output (filename: D.out)

For each test case, write the input lines in the output with the wiped out words converted to blanks. The whitespace at the end of each line is ignored when evaluating your output. Separate outputs for consecutive test cases with lines containing a single hash character.

Sample Input

```
2
ACM is
**
#
in this world
you are in*side
the world
*
#
```

Sample Output

```
ACM
**
#

you       *
the
*
#
```

Problem E (Program filename: E.cpp, E.dpr, or E.java)

Food Cubes

The spacemen in the space shuttle are waiting for the next escape window to return to the mother land Earth, where they are expected to fall somewhere in the deep blue waters of the Persian Gulf. Bored of waiting with nothing to do, they decide to play a game with their unit size food cubes. In the zero gravity environment of their spaceship, anything can stay motionless where it is placed. One spaceman places several food cubes in space such that there may be holes between cubes. Others, given the coordinates of the food cubes, should find the number of holes. A hole is a continuous empty space surrounded by food cubes in all six directions. You are to write a program to read the coordinates of each food cube and compute the number of holes.

Input (filename: E.in)

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. Each test case starts with an integer M, the number of food cubes. Each line i ($1 \leq i \leq M$) of the M following lines contains integers x_i, y_i and z_i, all between 1 and 100 inclusive, indicating the three coordinates of a food cube in the 3D space.

Output (filename: E.out)

For each test case, there is one line containing the number of holes.

Sample Input

```
2
26
1 1 1
1 2 1
1 3 1
2 1 1
2 2 1
2 3 1
3 1 1
3 2 1
3 3 1
1 1 2
1 2 2
1 3 2
2 1 2
2 3 2
3 1 2
3 2 2
3 3 2
1 1 3
1 2 3
1 3 3
2 1 3
2 2 3
```

```
2 3 3
3 1 3
3 2 3
3 3 3
7
1 1 1
1 1 2
1 2 1
1 2 2
2 1 1
2 1 2
2 2 1
```

Sample Output

```
1
0
```

Problem F (Program filename: F.cpp, F.dpr, or F.java)

Points

Let $p_1, p_2, \ldots, p_n$ be n points on the plane. We have m rules of form p_i *rel* p_j, each inform us that the relation *rel* holds among the locations of points p_i and p_j on the plane. For example, "p_i *NE* p_j" indicates that point p_j is located NorthEast of point p_i. There are eight different relations {`N, E, S, W, NE, NW, SE, SW`}, corresponding to the eight directions on the plane. Let (x_i, y_i) and (x_j, y_j) be the coordinates of p_i, and p_j respectively. Then p_i *rel* p_j exactly means one of the following, depending on the value of *rel*:

1. `N` stands for North. This means that $x_j = x_i$ and $y_j > y_i$,
2. `E` stands for East. This means that $x_j > x_i$ and $y_j = y_i$,
3. `S` stands for South. This means that $x_j = x_i$ and $y_j < y_i$,
4. `W` stands for West. This means that $x_j < x_i$ and $y_j = y_i$,
5. `NE` stands for NorthEast. This means that $x_j > x_i$ and $y_j > y_i$,
6. `NW` stands for NorthWest. This means that $x_j < x_i$ and $y_j > y_i$,
7. `SE` stands for SouthEast. This means that $x_j > x_i$ and $y_j < y_i$, and
8. `SW` stands for SouthWest. This means that $x_j < x_i$ and $y_j < y_i$.

The problem is to determine whether it possible to locate $p_1, p_2, \ldots, p_n$ on the plane so that all given rules are satisfied.

Input (filename: F.in)

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. The first line of each test case contains two integers n ($2 \leq n \leq 500$) which is the number of points and m ($1 \leq m \leq 10^4$) which is the number of rules. In each of the following m lines, there is one rule of the form i *rel* j which means that p_i has relation *rel* with p_j.

Output (filename: F.out)

The output contains one line per each test case containing one of the words `POSSIBLE` or `IMPOSSIBLE` indicating if the set of points in the test case can be located on the plane according to the given rules.

Sample Input

```
2
3 2
1 N 2
2 N 1
6 6
1 E 2
1 E 3
2 N 4
3 NW 5
```

```
4 SW 6
6 NE 5
```

Sample Output

```
IMPOSSIBLE
POSSIBLE
```

Problem G (Program filename: G.cpp, G.dpr, or G.java)

ACM-Telecom

ACM-Telecom provides communication services for international telephone calls. Cost of calling a country (call rates), varies from one to another. The company maintains the rates in a cost table mapping the country codes to call rates. Upon receiving a call, an automatic system determines the country code by looking at the leftmost digits in the 8-digit dialed number and charges the client according to the call rate of that country. More precisely, the automatic system maintains a list of country codes sorted in decreasing order. Upon receiving a call, the system starts from the top of the list and checks if the country code is a prefix of the dialed number. The first country code satisfying this property is considered as the destination of the call and the client is charged according to the call rate of that country. Note that the cost table covers every possible 8-digit number dialed, i.e., every dialed number matches with some country code in the table. Given the relatively high number of rows in the cost table and the increasing number of calls, the computation of call rates has become a quite lengthy process. Your task is to find a **new** cost table with the minimum number of rows such that the computed call costs for every possible (8-digit) dialed number are the same as before.

Input (filename: G.in)

The first line of the input contains a single integer t ($1 \le t \le 20$) which is the number of test cases in the input. Each test case starts with a line containing a single integer $N (1 \le N \le 1000)$ which is the number of rows in the cost table. Following the first line, there are N lines of the form *code cost* where *code* is an integer between 1 and 9999 inclusive, and *cost* ($1 \le cost \le 100$) is a positive integer which is the cost rate of the calls to the country *code*. There are no two lines with the same country code in a test case.

Output (filename: G.out)

For each test case, there is one line in the output containing the minimum number of rows of the table required to compute the costs correctly.

Sample Input

```
1
12
331 4
33 4
335 4
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
```

Sample Output

```
10
```

Problem H (Program filename: H.cpp, H.dpr, or H.java)

Prince of Persia

A new job for the well known Prince of Persia! Not surprisingly, Jafar, the treacherous vizier of the king has put the daughter of the king in a cellar while he is away to visit a neighboring country. The prince fights bravely with the guards on his way down to the cellar. Just one step to the cellar, the prince enters a dark room containing the information to unlock the door of the cellar. The information is carved on several stone plates installed in the walls of the room. Luckily, there is a ray of light emanating through a tiny hole in one of the walls. There are a number of mirrors in the room that the prince may use to direct the ray of light to illuminate a stone plate. Each mirror may be placed in certain locations in the room without any change in its directions (i.e., angles with the walls). The question is, if the prince can illuminate every plate on the wall using the mirrors.

To simplify the problem, we consider the room when viewed from above as a grid with n rows and m columns. The bounding cells are walls of the room and the mirrors can be put in the interior cells. No two mirrors can be placed in one cell. We have a number of mirrors available, each allowed to be put in certain positions inside the room. The direction of each mirror is known in advance and cannot be changed. To illuminate one plate, the prince must place a subset of mirrors in their allowed positions such that the ray of light reaches the plate after reflections made by the mirrors. The mirrors are opaque, meaning that the back-side of the mirrors absorbs the light. The problem is to find whether he can illuminate all and each plate on the wall. Obviously, the plates must be illuminated in turn since there is no way to fork the ray of light into multiple rays. When illuminating a plate, he may use a different subset of mirrors in possibly different positions.

Input (filename: H.in)

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. Each test case starts with zero or more lines each describing one mirror. A mirror is described by a single digit which is its identifier, followed by exactly one blank character, and one of the four character pairs `-/`, −\, `/-`, and \− indicating the mirror's direction. The hyphen character (`-`) determines the back-side of the mirror. For example, `-/` indicates a mirror which reflects the ray from the right to bottom and vice versa, but blocks the rays from the left or top.

After the mirror description lines, there are n lines each of length m characters which describe the grid representing the room when viewed from the above ($(2 \leq m, n \leq 40)$). The walls of the room (which are always in the boundary of the grid) are denoted by the hash signs (`#`). Those cells of the wall which contain plates are represented by `-` and `|` for horizontal and vertical walls respectively. The only cell in the boundary containing a blank character represents the hole from which the light is emanated into the room. The light ray enters the room perpendicular to the wall holding the hole. You may assume that the four cells in the corner always contain hash signs. The interior cells of the grid are all filled by the dot character (`.`) except those that a mirror can be placed into which are specified by digits. A mirror can be placed in a cell only if the digit of the cell is the same as the mirror identifier.

Output (filename: H.out)

For each test case, there is one line in the output containing the word `YES` or `NO` indicating if all plates in the input can be illuminated according to the problem conditions or not.

Sample Input

```
2
0 -/
1 /-
3 -/
5 \-
######-#####-#########
#....................#
|....................#
#.0..............3...#
#..........0.........#
#....................#
#....................#
#.....1....1.........#
#..........1..5......#
|....................#
|....................#
#....................#
#........1......1....#
#....................#
#....0......0........#
#..............3.....#
#...3................#
#...........3........#
## #####--############
0 -/
1 /-
3 -\
## ####-##
#........#
#.3..0.1.#
|.1......#
##########
```

Sample Output

```
NO
YES
```