University Interscholastic League

Computer Science Competition

2015 Invitational B Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Table of Contents

1. Age

Program Name: Age.java Input File: age.dat

We have some great news. Today is the day you shall write a program to solve one of the world's most pressing questions: "At what point was this person twice my age?" And since we know you are boundary-pushers, we'll take it one step further. You will write a program to determine whether one person's age ever has, or will be, *m* times the age of another.

Input
The first line of input consists of a single integer, *t*, indicating the number of test cases that follow.

For each test case, there will be three space-separated integers, *a*, *b*, and *m*, on a single line. *a* represents the younger person's age, *b* represents the older person's age, and *m* represents the age multiple.

Output
For each test case, print the ages at which the older person was (or will be) *m* times the age of the younger person. If this has never happened, and never will, simply print `NEVER`. Printed ages will be two positive integers on a single line separated by a space.

Constraints
```
1 <= t <= 10
1 <= a < b <= 1000
2 <= m <= 1000
```

Example Input File
```
3
2 14 4
10 20 2
30 31 3
```

Example Output to Screen
```
4 16
10 20
NEVER
```

2. Bitmap

Program Name: Bitmap.java Input File: bitmap.dat

A bitmap is a rectangular grid of bits. A bit may have one of two values, most commonly represented as `0` and `1`. In this problem, you will be given a bitmap of `0`s and `1`s, and you will have to determine the area of the largest rectangle that is filled with `1`s.

Input
The first line of input consists of two space-separated integers, *m* and *n*, which represent the dimensions of the bitmap, where *m* is the number of rows and *n* is the number of columns. This will be followed by *m* lines of *n* bits each.

Output
The output will be a single line giving the area of the largest rectangle that is filled with only 1s. You may think of the area as the total number of 1s in that rectangle.

Constraints
```
1 <= m <= 1000
1 <= n <= 1000
```

Example Input File
```
5 24
110010010000011111110110010001l0
001010001001101111101000110l0111
100111101011100001000011001011OO
101101011101010101111101000001l0
001101101101110000100011001101O1
```

Example Output to Screen
```
10
```

3. Blocks

Program Name: Blocks.java Input File: blocks.dat

Little Kitu is excited by all the blocks that he sees in his day care center. They are colorful and they come in various sizes. He wants to build a wall with those blocks so that they go along the length of his table. You are to determine if he can build that wall from one end of his table to the other, or not.

Input
The first line in the input file consists of a single integer *n*, the number of datasets that you have to consider. Each dataset has two lines. The first line consists of a single integer, *l*, which represents the length of the table. On the second line are *b* integers separated by one or more spaces, which represent the sizes of blocks that are available to him. He only has one block of any given size.

Output
You will print YES if Kitu can build a wall with his blocks or NO if he cannot. The wall must be the exact length of the table.

Constraints
`1 <= n <= 10`

Example Input File
```
2
50
15 9 30 21 19 3 12 6 25 27
100
12 34 8 42 22 5
```

Example Output to Screen
YES
NO

4. Birthday Plans

Program Name: Birthday.java **Input File: birthday.dat**

It's your best friend's birthday, and you are throwing her a surprise party! Your friend is a big fan of old, text-based video games from the 80s (the 1980s), and you have decided to theme the party accordingly. You've never thrown a themed party before, so you and some other creative compatriots get together and put together some ideas. One feature you all settle on is a banner with a simple ASCII-art design, as shown below.

You decide it would be much more interesting (and in keeping with the theme!) to write a computer program to generate this output.

Input
There is no input for this problem.

Output
You are to output the design seen below. Each character is of equal width (including spaces). Your output may look wrong and still be correct (it requires a fixed-width font to look the same as below), so make sure you are careful to follow the design exactly. You may assume that there are no trailing spaces on any given line, and no trailing newline.

```
     i i i i i i
    +-----------+
    |:::::::::::|
    |:H:a:p:p:y:|
    |:::::::::::|
  __%%%%%%%%%%%__
 |^^^^^^^^^^^^^^^^^|
 |:B:i:r:t:h:d:a:y:|
 |:::::::::::::::::|
 %%%%%%%%%%%%%%%%%
```

5. Candy

Program Name: Candy.java Input File: candy.dat

Halloween is just around the corner, and you are only now getting around to going shopping for candy!

Not a complete slacker, you've been thorough in your research and have calculated how many neighborhood children will come to your door. Fortunately (or unfortunately), they travel in a single pack of candy-craving goblins and ghouls.

Each child should get an equal amount of candy, otherwise they'll begin to squabble, throw tantrums, and wreak general havoc. According to your deductions, there are *n* possible group sizes of children that can come to your door.

You want to make sure that no matter how many children come to your door, you will have enough to split the candy evenly among them all. Since you waited so late to go candy shopping, all the good stuff was sold out at every grocery store in the city (how unlucky!), so you want to make sure that you have NO candy left after distributing it on Halloween, otherwise you'd have to eat subpar candy, and nobody wants that.

What is the smallest amount of candy you have to buy in order to split up the candy evenly to any possible group of children without having any left to spare?

Input
The first line of input consists of a single integer, *t*, indicating the number of test cases that follow.

For each test case, the first line will consist of a single integer, *n*, indicating how many potential group sizes there may be on Halloween. On the next line, there will be *n* integers, which represent all of the potential group sizes.

Output
For each test case, print on its own line the minimum number of candy pieces you should buy.

Constraints
```
1 <= t <= 8
1 <= n <= 10

All group sizes will be under 100. The answer will always fit within a 32-bit
signed integer.
```

Example Input File
```
2
2
3 4
3
5 10 12
```

Example Output to Screen
```
12
60
```

6. Coordinator

Program Name: Coordinator.java Input File: coordinator.dat

The annual Convention for People Who Like to Do Stuff and Things is rolling around. As head event coordinator, you've been tasked with finding a single room big enough to fit all the events that need to take place.

You've been given a list of the events, their start and end times (in minutes since the start of the convention), and their estimated number of guests. You've also been furnished with a list of rooms, and their respective capacities, that are within the allotted budget.

You are hoping to have some of the budget roll over into next year's convention budget, and smaller rooms are always cheaper than larger rooms. Find the smallest room that is still large enough to fit all the events that need to take place. Specifically, if two or more events overlap, the room must be big enough to hold all the overlapping events at the same time.

Input

The first line of input consists of a single integer, *t*, indicating the number of test cases to follow.

Test cases are arranged thusly:

- The first line of each test case consists of two space-separated integers, *n* and *m*, where *n* is the number of rooms and *m* is the number of events.
- The next *n* lines that follow represent the *n* rooms, and each consists of a single integer *c*, where *c* is the max capacity of the room. The rooms are named alphabetically, so the first room is room A, the second is room B, and so on.
- Following that are *m* lines, representing the events. Each event is on its own line and contains three integers, *s*, *e*, and *g*, where *s* is the inclusive start time in minutes since convention start, *e* is exclusive end time in minutes since convention start, and *g* is the anticipated number of guests for the event.

Output

For each test case, output the smallest room that is big enough to hold all the events or "Not Possible" if none of the rooms can fit the requested events. If there are multiple smallest rooms, print the room that comes first alphabetically.

Constraint

```
1 <= t <= 10
1 <= n <= 26
1 <= m <= 200
1 <= c, g <= 10^9
0 <= s < e <= 10^9
```

Example Input File

```
2
2 2
20
25
0 60 20
60 120 20
2 3
10
20
0 30 10
15 45 10
25 30 5
```

6. Coordinator (cont.)

Example Output to Screen

```
A
Not Possible
```

Explanation of sample cases

In the first case, the two events do not overlap, so the room only needs to be as big as the biggest event. Thus, we output `A`.

In the second case, from time 25 to time 30, all three events overlap and so the room must be big enough to hold 25 people. The biggest available room's max capacity is only 20 people, so we output `Not Possible`.

7. HTML

Program Name: HTML.java Input File: html.dat

Bobby is the UT ACM webmaster, and he has made an amazing website for this year's students. Joey, the Texas A&M ACM webmaster, is lazy and decides to just copy Bobby's website, and change the text in certain places.

Bobby wants to prove that Joey did this but has little coding experience outside of HTML. As the UT ACM President, your job is to write a program for Bobby to determine whether two given HTML pages have the same structure. Two pages have the same structure if the structure of their tags are the same. Plain text is ignored when making this determination.

You can assume there are no escaped characters, and any given HTML element will not have any extraneous attributes. A tag is simply "<" or "</", followed by one or more lowercase letters, followed by ">" (e.g., `<div>` and `</div>`). You may also assume the HTML will not have any broken or incomplete tags.

Input
The first line of input consists of a single integer, *n*, indicating the number of test cases.

Each test case consists of two lines, which are the two pages requiring comparison. The first line is Bobby's code, and the second line is the page he thinks is copied from his code.

Output
For each pair of pages, output `same` if the pages have the same structure, or `different` if the pages have a different structure.

Constraints
```
1 <= n <= 10
1 <= length of any page <= 100 characters
```

Example Input File
```
4
<div>hi</div>
<div>hi</div>
<div>hi</div>
<div>hello</div>
<div>hi</div>
<p>hi</p>
<p><b>hello</b> world <b>foobar</b></p>
<p><b>hello</b> world <i>foobar</i></p>
```

Example Output to Screen
```
same
same
different
different
```

Explanation of sample cases
In the first case, both pages are exactly the same, so the output is `same`.
In the second case, both pages just have a single div element. The plain text inside the div is ignored. Thus the output is `same`.
In the third case, the elements are different. The first page has a div, while the second has a paragraph. Thus, the output is `different` (Note: we still ignore the plain text).
In the fourth case, both HTML pages consist of a paragraph with two child elements. The first child of both is a bold tag, which maintains our match. However, the second child differs; one is a bold tag, and the other is an italics tag. Thus, the output is `different`.

8. Nested Palindromes

Program Name: Palindromes.java Input File: palindromes.dat

A palindrome is a sequence of letters of the English alphabet that read the same going left to right or going right to left. In determining a palindrome, ignore case, digits, punctuation marks and spaces. The string "Able was I ere I saw Elba" is a palindrome. For this problem, we will consider strings of a single letter, like "Z", to be a palindrome.

Given a string, you will determine the length of the longest substring that is a palindrome. For example, the string "Mississippi" is not a palindrome. But the substrings that are palindromes are: I, P, M, S, PP, SS, IPPI, ISSI, ISSISSI. The length of the longest substring that is a palindrome is 7.

Input

The first line in the input file will consist of a single integer *n*, indicating the number of strings to follow.

Following this will be *n* lines of strings, each between 1 and 128 characters, inclusive. The strings will be a mix of uppercase and lowercase letters, digits, punctuation marks, and spaces.

Output

For each string, you will print on its own line the length of the longest substring that is a palindrome, ignoring case, digits, punctuation marks, and spaces.

Constraints

```
1 <= n <= 100
```

Example Input File

```
3
Madam, I'm Adam.
1234.4321
Mississippi Burning
```

Example Output to Screen

```
11
0
7
```

9. Platform

Program Name: Platform.java Input File: platform.dat

You've been placed in a maze. You really dislike mazes. Thankfully, you've been given a layout of the maze to make your life a bit easier, but looking at it, you're not sure if it's actually possible to reach an exit.

The layout is given as an *n* x *m* grid. You are suddenly very glad you have it, as it marks the locations of the floor tiles that will begin a headfirst slide across what is turning out to be an incredibly slippery maze, which are represented as follows:

- `^` makes you slide upwards
- `>` makes you slide to the right
- `<` makes you slide to the left
- `v` makes you slide downwards

Slippery tiles, which will only help you along your runaway slide, are represented by a '`.`'. Thankfully, there are some platforms that prevent you from sliding further. Those platforms are denoted by an '`x`'. You may move from one platform to any adjacent sliding floor piece or platform within the maze, but not a slippery tile. You may not change your sliding direction while sliding, unless you slide over another floor piece that makes you slide another direction.

You always start in the top left corner of the grid. In order to escape, you must land on the bottom right corner. Both locations are always guaranteed to be platforms. You can never slide off the grid, and there won't be a situation where you slide forever.

Input
The first line of input consists of a single integer, *t*, indicating the number of test cases that follow.
For each test case, there will be two integers, *n* and *m*. The following *n* lines have *m* characters each, NOT separated by a space. These *n* lines represent the maze layout.

Output
For each test case, print YES if it's possible to reach the bottom right platform, and NO if it's not possible.

Constraints
```
1 <= t <= 10
1 <= n, m <= 100
```

Example Input File
```
3
1 5
x>..x
3 3
x>v
xv<
^<x
7 7
x>....v
xv.>.v.
...^...
...x..<
...>..v
...^.<.
.>..x.x
```

9. Platform (cont.)

Example Output to Screen

```
YES
NO
YES
```

10. Resources

Program Name: Resources.java Input File: resources.dat

The date is 1965. You've been put in charge of manually managing the resource allocation for the local college's supercomputer. At the start of each day you're given a list of the jobs that must be run that day. You must figure out how much peak memory that day's jobs will use and whether or not the supercomputer has sufficient memory.

Two overlapping jobs cannot reuse the same resources, but two disjoint jobs can use the same resources. This means that if job A uses 7KiB of memory but finishes before job B starts, which requires 6KiB of memory, then the max amount of memory you need is 7KiB.

On the other hand, if job B starts before job A ends, then the max amount of memory you will require is 13KiB (enough for the amount of time that job A and job B overlap).

Input

The first line of input consists of a single integer *t*, the number of test cases.

Each test case begins with a line consisting of two integers, *g* and *n*, where *g* is the amount of memory available to the machine (given in KB) and *n* is the integer number of jobs for that day. *n* lines follow, each line contains three integers, *m*, *s*, and *d*, where *m* is the memory requirement of the job (given in KB), *s* is the start time of the job given in minutes since 00:00, and *d* is the duration of the job in minutes. A job that starts at minute 4 for a duration of 3 minutes and a job that starts at minute 7 are disjoint.

Output

For each test case, if the supercomputer has sufficient memory to execute the schedule, then print `SUFFICIENT`. If not, print the peak amount of memory that the schedule required in the following form:

```
OUT OF MEMORY: xKB REQUESTED
```

where x is the peak memory required.

Constraint

```
1 <= t <= 10
1 <= g <= 20480
1 <= n <= 1000
0 <= m <= 20480
0 <= s < 86400
0 <= s + d <= 86400
```

Example Input File

```
2
32 3
22 10 5
9 13 1
1 12 2
640 4
630 0 60
630 60 60
630 120 60
11 59 2
```

Example Output to Screen

```
SUFFICIENT
OUT OF MEMORY: 641KB REQUESTED
```

11. Rook

Program Name: Rook.java Input File: rook.dat

You're playing chess on a nice *n* x *m* board. Your opponent has decided to place *k* rooks around the board. How many positions do you have to safely place your pawn? A rook can move only horizontally or vertically and capture horizontally or vertically.

Input
The first line of input consists of a single integer, *t*, indicating the number of test cases that follow.

For each test case, there will be three space-separated integers, *n*, *m*, *k* on a single line, where *n* represents the number of rows, *m* represents the number of columns, and *k* represents the number of rook positions. The next *k* lines have two space-separated integers each, representing the row and column placement of a rook. All positions are 0-indexed.

Output
For each test case, print on its own line number of locations you can place your pawn on the board.

Constraints
```
1 <= t <= 15
1 <= n, m <= 1000
0 <= k <= n * m
```

Example Input File
```
3
2 2 1
0 0
2 2 2
0 0
1 0
3 3 1
1 1
```

Example Output to Screen
```
1
0
4
```

12. Round Robin

Program Name: RoundRobin.java Input File: roundrobin.dat

Kevin has an Ultimate Frisbee tournament coming up. The tournament is supposed to be a round robin between *n* teams. In a round robin, every team plays every other team exactly once. Unfortunately, the tournament organizers did the scheduling by hand, and there may be some errors. They've asked Kevin to validate that every team plays every other team once. Can you help Kevin check the schedule?

Input

The first line of input consists of a single integer, *t*, indicating the number of schedules.

Each schedule is comprised of the following:

- The first line consists of two space-separated integers, *n* and *m*, where *n* represents the number of teams in the round robin and *m* represents the number of games.
- The next *n* lines contain team names, which are strings of one or more words.
- The next *m* lines represent the *m* games. Each game is in the format X v Y, where X and Y are the indexes of two teams that are playing each other. Indices begin at 1 (team numbers are not zero-indexed).

Output

For each schedule, if the schedule is a round robin, print YES. Otherwise, print NO. Follow this with a newline.

If there are invalid games (a team playing itself or playing a team that doesn't exist, print INVALID GAMES).

If there are teams that aren't playing each other in the schedule, print MISSING GAMES followed by a newline, followed by the missing games. Print the games in the form A v B, where A is the team whose name comes first alphabetically. Order the listing of missing games alphabetically. Separate the output for each game with a newline.

If there are teams which play each other more than once, print DUPLICATE GAMES followed by a newline, followed by the duplicate match. Even if two teams play each other more than twice, only print the duplicate game once. Print the games in the same format as in the missing games. Separate the output for each game with a newline.

Constraints

```
1 <= t <= 10
1 <= n < 30
1 <= m < 400
```

12. Round Robin (cont.)

Example Input File

```
3
3 3
University of Texas
Texas A&M
Rice University
1 v 3
3 v 2
2 v 1
4 5
Harvard
Princeton
Cornell
Dartmouth
1 v 2
3 v 1
4 v 3
2 v 1
1 v 4
1 1
Lonely Guy
1 v 1
```

Example Output to Screen

YES

NO
MISSING GAMES
Cornell v Princeton
Dartmouth v Princeton
DUPLICATE GAMES
Harvard v Princeton

NO
INVALID GAMES