Problem Solving Strategies – set of advanced problems

Maria Keet
University of Cape Town, South Africa

ICPC Training Session, April 12, 2015

Task: categorise the following set of more and less advanced problems into the right strategies.

Each problem fits into one or more category (mostly more than one): complete search, divide & conquer, greedy, dynamic programming, *ad hoc* (i.e., none in particular), some specific data structure, approach not listed here namely...

They are in one of the competitive programming systems online, so do try to code them *after* you categorised them. Where they are to be found is on the solution page, not here, because you are expected to have solved it to at least to the point of direction of the solution before 'testing' that in the online system.

Contents

1 Nuts for Nuts

Liam and Linda are two squirrels normally foraging nuts a bit up the hill on Upper Campus, but they decided they got bored with the taste of them. However, they've realized that there are some nuts located at certain places of the built-up area of the campus... and they love them! Since they're lazy, but greedy, they want to know the shortest tour that they can use to gather every single nut! Can you help them?

Input You'll be given x and y (both less than 20), followed by x lines of y characters each as a map of the area, consisting sorely of ".", "#", and "L". Liam and Linda are currently located in "L", and the nuts are represented by "#". They can travel in all 8 adjacent directions in one step. See below for an example. There will be at most 15 places where there are nuts, and "L" will only appear once.

Output On each line, output the minimum amount of steps starting from "L", gather all the nuts, and back to "L".

Sample input

```
5 5
L....
#....
#....
.....
#....
5 5
L....
#....
#....
.....
#....
```

Sample output

```
8
8
```

Liam and Linda will go south for a nut, then south again for another nut, then south twice for another nut, and then back where they are.

2 The Fortified Forest

Once upon a time, in a nearby land, there lived a king. This king owned a small collection of rare and valuable trees, which had been gathered by his ancestors on their travels. To protect his trees from thieves, the king decided he wanted to have high fence built around them and he put his wife, the queen, in charge of realising this.

Alas, the queen quickly noticed that the only suitable material available to build the fence was the wood from the trees themselves. In other words, it was necessary to cut down some trees in order to build a fence around the remaining trees. Of course, to prevent her head from being chopped off, the queen wanted to minimise the value of the trees that had to be cut. The queen went to her tower and stayed there until she had found the best possible solution to the problem. The fence was then built and everyone lived happily ever after.

You are to write a program that solves the problem the queen faced.

Input The input contains several test cases, each of which describes a hypothetical forest. Each test case begins with a line containing a single integer n, with $2 \leq n \leq 15$, the number of trees in the forest. The trees are identified by consecutive integers 1 to n. Each of the subsequent lines contains 4 integers x_i, y_i, v_i, l_i that describe a single tree. (x_i, y_i) is the position of the tree in the plane, v_i is its value, and l_i is the length of fence that can be built using the wood of the tree. v_i and l_i are between 0 and 10000. The input ends with an empty test case ($n = 0$).

Output For each test case, compute a subset of the trees such that, using the wood from that subset, the remaining trees can be enclosed in a single fence. Find the subset with a minimum value. If more than one such minimum-value subset exists, choose one with the smallest number of trees. For simplicity, regard the trees as having zero diameter.

Display, as shown below, the test case numbers (1, 2, ...), the identity of each tree to be cut, and the length of the excess fencing (accurate to two fractional digits).

Display a blank line between test cases.

Sample input

```
6
 0 0 8 3
 1 4 3 2
 2 1 7 1
 4 1 2 3
 3 5 4 6
 2 3 9 8
3
 3 0 10 2
 5 5 20 25
 7 -3 30 32
0
```

Sample output

```
Forest 1
Cut these trees: 2 4 5
Extra wood: 3.16

Forest 2
Cut these trees: 2
Extra wood: 15.00
```

3 Colliding Catamarans

For a catamaran on a small, constrained body of water, other traffic can be a major hazard. The more traffic there is in the same area, the higher the risk of a collision. Your job is to monitor traffic and help detect likely collisions before they occur. You have sensors to detect the position, direction, and speed of each catamaran. Assuming the direction and speed remain constant, your task is to determine whether any of the catamarans will collide. Two catamarans are considered to collide if they come within a given distance of each other.

Input The first line of input contains a single integer c, the number of test cases to follow. Each test case starts with a line containing two numbers, n, the number of catamarans, and r, the collision distance. Two catamarans are considered to collide if they come within r metres of each other. There will be no more than 1000 catamarans. Each catamaran is identified by a line containing four numbers x, y, d, and s. The numbers x and y give the current position of the catamaran as a distance east and north, respectively, from a common origin, and will be between -1000 and 1000, inclusive. The lake is small enough that we can model it as a flat surface. The number d gives the direction in which the catamaran is heading in degrees clockwise from north (so east is 90 degrees). The number s gives the speed of the catamaran in metres per second, and will be between 0.001 and 1000. Note that r, x, y, d, and s are not necessarily integers.

The input data will be such that the answer will not change if any of the numbers x, y, d and s are changed by 10^{-6} or less. If any two catamarans are within r metres of each other in their initial position, they are already collided, so you should output '`0`' for that case.

Output For each test case, output a line containing a single integer, the number of seconds, rounded to the nearest second, before any of the catamarans come within r metres of each other. If none of the catamarans ever collide, output the line: '`No collision.`'

Sample input

```
2
2 5
0 0 90 1
10 10 180 1
2 10
0 0 0 0
8 8 270 1
```

Sample output

```
6
2
```

4 Streaming System

A data stream is a real-time, continuous, ordered sequence of items. Some examples include sensor data, Internet traffic, financial tickers, on-line auctions, and transaction logs such as Web usage logs and telephone call records. Likewise, queries over streams run continuously over a period of time and incrementally return new results as new data arrives. For example, a temperature detection system of a factory warehouse may run queries like the following.

> *Query 1*: Every five minutes, retrieve the maximum temperature over the past five minutes.
> *Query 2*: Return the average temperature measured on each floor over the past 10 minutes.

At the Amazingly Current Management systems company, they have developed a Data Stream Management System called Argus, which processes the queries over the data streams. Users can register queries to the Argus. Argus will keep the queries running over the changing data and return the results to the corresponding user with the desired frequency. For the Argus, we use the following instruction to register a query:

> Register Q_num Period

Q_num ($0 < \text{Q\_num} \leq 3000$) is query ID-number, and Period ($0 < \text{Period} \leq 3000$) is the interval between two consecutive returns of the result. After Period seconds of register, the result will be returned for the first time, and after that, the result will be returned every Period seconds.

Here we have several different queries registered in Argus at once. It is confirmed that all the queries have different Q_num. Your task is to tell the first K queries to return the results. If two or more queries are to return the results at the same time, they will return the results one by one in the ascending order of Q_num.

Input The first part of the input are the register instructions to Argus, one instruction per line. You can assume the number of the instructions will not exceed 1000, and all these instructions are executed at the same time. This part is ended with a line of `#`.

The second part is your task. This part contains only one line, which is one positive integer K (≤ 10000).

Output You should output the Q_num of the first K queries to return the results, one number per line.

Sample input

```
Register 2004 200
Register 2005 300
#
5
```

Sample output

```
2004
2005
```

2004 2004 2005

5 Fewest Flops

A common way to uniquely encode a string is by replacing its consecutive repeating characters (or "chunks") by the number of times the character occurs followed by the character itself. For example, the string "aabbbaabaaaa" may be encoded as "2a3b2a1b4a". (Note for this problem even a single character "b" is replaced by "1b".)

Suppose we have a string S and a number k such that k divides the length of S. Let S_1 be the substring of S from 1 to k, S_2 be the substring of S from $k+1$ to $2k$, and so on. We wish to rearrange the characters of each block S_i independently so that the concatenation of those permutations S' has as few chunks of the same character as possible. Output the fewest number of chunks.

For example, let S be "uuvuwwuv" and k be 4. Then S_1 is "uuvu" and has three chunks, but may be rearranged to "uuuv" which has two chunks. Similarly, S_2 may be rearranged to "vuww". Then S', or S_1S_2, is "uuuvvuww" which is 4 chunks, indeed the minimum number of chunks.

Input The input begins with a line containing t (s.t. $1 \leq t \leq 100$), the number of test cases. The following t lines contain an integer k and a string S made of no more than 1000 lowercase English alphabet letters. It is guaranteed that k will divide the length of S.

Output For each test case, output a single line containing the minimum number of chunks after we rearrange S as described above.

Sample input

```
2
5 helloworld
7 thefewestflops
```

Sample output

```
8
10
```

6 Cult Caps

The SA Alternative Brewing Company has commissioned a new series of designer pub tables, with legs that are constructed from stacks of bottle caps. Each table has four legs, each of which uses a different type of cap. For example, one leg might be a stack of beer bottle screw caps, another wine bottle screw caps, and yet another with the longer caps from imported tequila bottles. Each leg must be exactly the same length.

Many (types of) caps are available for these tables. Given an inventory of available caps and a desired table height, compute the lengths nearest to the desired height for which four legs of equal length may be constructed using a different type of bottle cap for each leg.

Input Input consists of several test cases. Each case begins with two integers: $4 \leq n \leq 50$ giving the number of types of caps available, and $1 \leq t \leq 10$ giving the number of tables to be designed. n lines follow; each gives the thickness of a cap. t lines follow; each gives the height of a table to be designed. A line containing `0 0` follows the last test case.

Output For each table, output a line with two integers: the greatest leg length not exceeding the desired length, and the smallest leg length not less than the desired length.

Sample input

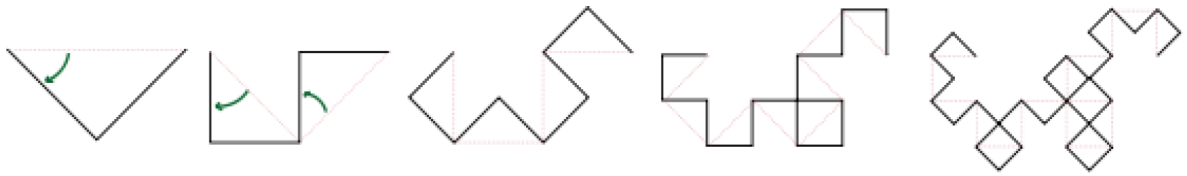
```
4 2
50
100
200
400
1000
2000
0 0
```

Sample output

```
800 1200
2000 2000
```

7 Hallway Help

Kody is now 21 years old. He has to appear in an exam to renew his "State Alchemist" title. This year the exam is arranged a bit differently. There will be a long hallway. Each alchemist will enter the hallway from the left side and come out from the right side (hopefully!) and s/he has to do this n times. During this tour they have to bend the hallway segments right-left alternatively. Let's describe the process in some pictures:



- First time (First picture): Initially, the hallway is a straight line (soft line in the first picture). So, the alchemist will bend this segment to right side (he is going from left to right) like the hard line in the first picture above.
- Second time (Second picture): Now he will find two segments in hallway (like the soft line in the picture). So he will bend the first hallway to right, second one to left (like the hard lines).
- Third time (Third picture): Now he will find four segments in the hallway (like the soft lines) and he will bend them to Right, Left, Right and Left respectively.
- And this goes on for the fourth and fifth times in the pictures on the right.

Since Alchemist Cum Mathematician Kody is so good, he did it perfectly. Now it is the turn of the judges to check the bending if it is correct or not. The judge enters at the left end and comes out from the right end. The judge notes down the turning during his travel, as R for Right and L for Left. So if $n = 1$, then the judge would have noted down L. If $n = 2$, it would have been LLR. For $n = 4$, it would have been: LLRLLRRLLLRRLRR.

Since this string will grow exponentially with n, it will be tough to check whether the bending is correct or not. So the judges have some pre-generated strings and they know whether this string will appear as substring in the final string or not. Unfortunately the, somewhat sloppy, judges have lost the answer sheet. Can you help them to recover it?

Input The first line of the test file contains a positive integer T denoting number of test cases ($T \leq 105$). Hence follow T lines, each containing an integer and a string: n S. n is the number of times Kody has passed through the hallway, and S is the string the judge is going to check with. You may assume that S consists of only the letters L and R. ($n \leq 1000$, length of $S \leq 100$). Also you may assume that length of S will not be greater than the length of the string for n.

Output For each test case output the case number and `Yes` or `No` denoting whether the string is in the final string as substring.

Sample input

```
2
1 R
4 LRRLL
```

Sample output

```
Case 1: No
Case 2: Yes
```

Solution

This page doesn't give you the solution to the actual problem, only the category and which UVa problem it is. You still have to solve and code it. You can test your solution at `http://uva.onlinejudge.org/`.

Problem 1: Nuts for nuts. This is UVa problem 10944, "Nuts for nuts". Solvable using Graph Preprocessing and Dynamic Programming.

Problem 2: The Fortified Forest. This is UVa problem 811, "The Fortified Forest". Solvable using three components. Which ones?

Problem 3: Colliding Catamarans. This is UVa problem 11574, "Colliding Traffic". Solvable using Complete Search and Geometry.

Problem 4: Streaming System. This is UVa problem 1203, "Argus", (and also used in the Asia - Beijing - 2004/2005 ACM ICPC Local Contest). Solvable using C++ STL priority_queue (Java PriorityQueue).

Problem 5: Fewest Flops. This is UVa problem 11552, "Fewest Flops". Solvable using String Processing with Dynamic Programming.

Problem 6: Cult Caps. This is essentially UVa problem 10717, "Mint"; I changed the storyline, but all the variables, input etc. are exactly the same. Solvable using two components, involving mathematics.

Problem 7: Hallway Help. This is the "Decoding the Hallway" (Problem D) of SWERC 2013. This requires and *ad hoc* solution.