MIT Programming Contest Problems

Feb. 26, 2005

This page was intentionally left blank.

1 Requirements

An undergraduate student, realizing that he needs to do research to improve his chances of being accepted to graduate school, decided that it is now time to do some independent research. Of course, he has decided to do research in the most important domain: the requirements he must fulfill to graduate from his undergraduate university. First, he discovered (to his surprise) that he has to fulfill 5 distinct requirements: the general institute requirement, the writing requirement, the science requirement, the foreign-language requirement, and the field-of-specialization requirement. Formally, a requirement is a fixed number of classes that he has to take during his undergraduate years. Thus, for example, the foreign language requirement specifies that the student has to take 4 classes to fulfill this requirement: French I, French II, French III, and French IV. Having analyzed the immense multitude of the classes that need to be taken to fulfill the different requirements, our student became a little depressed about his undergraduate university: there are so many classes to take...

Dejected, the student began studying the requirements of other universities that he might have chosen after high school. He found that, in fact, other universities had exactly the same 5 requirements as his own university. The only difference was that different universities had different number of classes to be satisfied in each of the five requirement.

Still, it appeared that universities have pretty similar requirements (all of them require a lot of classes), so he hypothesized that no two universities are very dissimilar in their requirements. He defined the dissimilarity of two universities X and Y as $|x_1 - y_1| + |x_2 - y_2| + |x_3 - y_3| + |x_4 - y_4| + |x_5 - y_5|$, where an x_i (y_i) is the number of classes in the requirement i of university X (Y) multiplied by an appropriate factor that measures hardness of the corresponding requirement at the corresponding university.

Your task is to verify the hypothesis by finding the two universities that have the highest dissimilarity.

Input

The first line of the input file contains an integer n ($1 \leq N \leq 100000$), the number of considered universities. The following N lines each describe the requirements of a university. A university X is described by the five non-negative real numbers x_1 x_2 x_3 x_4 x_5.

Output

On a single line, print the dissimilarity value of the two most dissimilar universities. Your answer should be rounded to exactly two decimal places.

Example

Sample Input

```
3
2 5 6 2 1.5
1.2 3 2 5 4
7 5 3 2 5
```

Sample Output

```
12.80
```

2 Constructing Roads

In the days of yore, Han was a prosperous kingdom. But at the turn of the century, hard times fell upon the kingdom. A plague swept through the land, and barbarians galloped in from the north, burning farms, destroying roads, and pillaging villages. All that remained were a few isolated strongholds scattered throughout the land. It has now been nearly a decade since the last wave of barbarians stormed through, and the land breathes a sigh of relief. People are becoming revitalized with the hope that they can once again transform the kingdom back into its former glory.

Since all the roads were destroyed, the strongholds were left in isolation, so the first order of business was to build a network of roads connecting all the strongholds. Each stronghold thought it would be a reasonable plan to start by building a road to the closest stronghold near it. If there were two strongholds of equal distance, the stronghold whose name comes before the other in the dictionary would be chosen. Each stronghold builds at its own rate, measured in feet/hour. Because they wished to finish as soon as possible, construction happened 24 hours a day, continuously advancing the construction site (the end of the road) towards the destination. Of course, construction on a road would stop if the road ran into another road or a city. At the beginning of the New Year, there was a big celebration, and all the strongholds began construction at the same time.

Little did the people of Han know that the barbarians had again infiltrated their kingdom and were carefully observing the progress of the roads. The barbarians were curious about the progress of the roads. In particular, they wanted answers to two type of questions.

1. After exactly t hours since construction began on the New Year, what is the absolute minimum length of additional roads that still need to be built in order to connect all cities? These additional roads are allowed to join two cities, two construction sites, or a city and a construction site.

2. What is the fewest number of hours that must elapse since the New Year before the minimum length of additional roads that still need to be built is at most l?

Write a program to answer these questions given Han's construction plan.

Input

There will be several test cases, each representing a possible scenario for Han. The first line of each test case will contain a positive integer number N, the number of strongholds ($1 \leq N \leq 2000$). Each of the subsequent N lines will contain a description of a stronghold: a name consisting of letters 'a'...'z', the x and y coordinates of the position of the stronghold (in feet), and the construction rate (in feet/hour). The next lines will contain questions. The first integer on the line is either 1 or 2, representing the type of question. For type 1 questions, the next number is t, the time (in hours) in question. For type 2 questions, the next number is l, the length (in feet) in question. The questions will be terminated by a line with 0.

The input data is terminated by a line that contains one zero, and should not be processed.

Output

For each test case, output the answers to each question, formatted as in the sample output. If for question 2, at no point in time will there be only l feet of construction left, then print `NEVER`. All numeric answers should be printed to as many decimal places as you feel necessary. To get credit for the problem, however, your answer must be within 0.01 of our answer.

Example

Sample Input

```
4
portland 0 0 3
seattle 0 10 2
newyork 20 6 1
boston 20 0 1
1 0
1 2.0
1 3.0
2 29
2 1.0
0
2
bree -10 -10 1
buckland 10 10 2
1 5
0
0
```

Sample Output

```
Kingdom 1
36.000 feet left at time 0.000
22.000 feet left at time 2.000
20.000 feet left at time 3.000
1.000 hours before 29.000 feet left
NEVER

Kingdom 2
13.284 feet left at time 5.000

End
```

3 A City of Skyscrapers

Modern cities often contain densely packed skyscrapers arranged neatly on a rectangular grid of streets and avenues. Skytown is no exception. The city has grown tremendously in the past few years. New skyscrapers, ever taller than previous skyscrapers, are constantly being erected with great haste. The skyscrapers have all been constructed identically, of course with the exception that some skyscrapers are taller than others. According to city regulations, each floor of a skyscraper has some minimum and maximum capacity, c and C, respectively. At least c people are required to live on a floor to ensure that the floor is utilized to its full potential. At most C people are permitted to live on a floor to prevent overcrowding.

Because Skytown has grown so fast, the mayor wanted to boast about the city's soaring population. The only problem is that he hasn't the faintest clue how many people live in Skytown. He has put you in charge of estimating the city's population. Of course, being a programmer, you seek a programming solution and do not want to go around the entire city asking how many people live on each floor. You come up with the following simple strategy: you will record the *skyline* as viewed from from both the south and the west. The skyline from the south is computed as follows: for each line of skyscrapers running north-south, the highest one in that line is recorded.

Given this data, compute the minimum and maximum number of people that could be living Skytown.

Input

The first line contains four integers, M ($1 \leq M \leq 100,000$), N ($1 \leq N \leq 100,000$), c ($0 \leq c \leq 500$), and C ($c \leq C \leq 500$), where M is the dimension of the grid in the north-south direction, N is the dimension of the grid in the east-west direction, c and C are the minimum and maximum number of people allowed per floor.

Each of the next M lines contains exactly one integer in $[0, 20000]$. Together, they specify the western skyline. After this, the next N lines specify the southern skyline in the same way.

Output

The output contains the minimum and maximum number of people that could be living in Skytown. Both numbers are guaranteed to fit in a 32-bit signed integer. If the two skylines specified in the input are consistent, that is, cannot possibly describe a possible configuration of skyscrapers, print "Impossible" on a single line.

Example

Sample Input

```
5 10 10 20
2
4
6
8
10
1
2
3
4
5
6
7
8
9
10
```

Sample Output

```
Minimum: 550, maximum: 4100
```

4 The Traveling Salesman

For thousands of years, salesmen have faced the problem of touring sites for potential customers, visiting each site exactly once, and minimizing the total cost of this tour. In the old days, these sites were towns spread out throughout the country, and the salesman had to travel the rickety, crooked roads that connected the towns. But now, in 2030, cities and have modernized and grown, so a salesman typically stays within the same city. Furthermore, modern cities are laid out in a grid-like pattern of tall skyscrapers, with sky bridges joining some pairs of adjacent skyscrapers on all floors.

The salesman has determined which floor of each skyscraper contains the most potential customers and will only visit the best floor of each skyscraper. Now the salesman must plan his course. Starting on the ground floor (floor 0) of the skyscraper in the northwest corner of the grid, the salesman must visit the specified best floor of each skyscraper before returning to the ground floor of either of the skyscrapers at the southwest, southeast, or northeast corners. In order to travel between floor 10 of one skyscraper and floor 6 of an adjacent skyscraper, the salesman first takes the sky bridge across to floor 10 of the adjacent skyscraper and then takes an elevator down 4 floors. Of course, because the salesman's time is precious, he wants to minimize the total number of floors he must travel by elevator. For example, if his tour involves traveling from floor 3 to 8 to 5 to 10, the number of floors traveled by elevator is $(8-3)+(8-5)+(10-5)=13$.

To make the salesman's life (and yours) more difficult, some sky bridges do not exist between two adjacent skyscrapers, and in that case, obviously the salesman cannot fly across the gap.

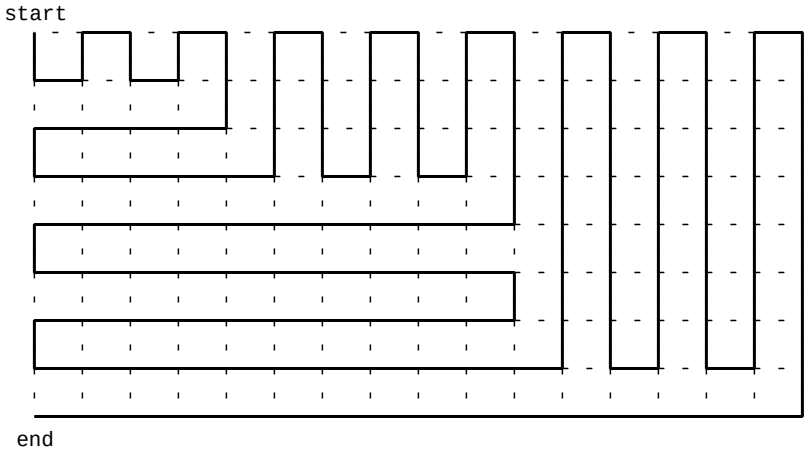
Because modern cities are much larger than the small towns that salesmen dealt with generations ago, our salesman needs a plan so that he does not lose his bearings and visit a skyscraper more than once (in which case he will get beaten by the people for pestering them so much) or less than once (in which case he might have lost profit). To that end, the salesman has decided to only consider tours similar to the one in the figure. The salesman starts at the northwest corner, and either goes south or east for any distance (as long as all sky bridges exist along that path). Then, he turns towards away from the edge of the city and travels to the adjacent skyscraper, and turns again, heading back opposite the initial direction. After any amount of zig-zagging, the salesman can choose to switch over to zig-zagging in the other orientation.

Given the floors of each skyscraper in the city that the salesman wishes to visit and which sky bridges do not exist, compute the minimum number of floors that the salesman must travel on a tour that meets the above criteria. Also, print out the number of possible tours achieving that minimum.

Input

The first line contains two numbers M ($1 \leq M \leq 1000$) and N ($1 \leq N \leq 1000$), the dimensions of the city. Each of the next M lines contains N integers in the range $[0, 100]$. Each integer is the floor that the salesman must visit. Optionally following each integer

might be a token specifying that some sky bridges do not exist. Suppose you just read the floor for location (x, y) (note that $(0, 0)$ is the northwest corner, $(N-1, 0)$ is the northeast corner, $(0, M-1)$ is the southwest corner, and $(N-1, M-1)$ is the southeast). If the token is `x`, no sky bridges exist between (x, y) and $(x+1, y)$. If the token is `y`, no sky bridges exist between (x, y) and $(x, y+1)$. All tokens and integers are separated by a single space.



Output

If there is no tour, print `No solution`. Otherwise, print out the number of tours and the minimum number of floors travelled.

Example

Sample Input	Sample Output

`2 4`
`0 y 10 20 30`
`5 8 25 28`
	`1 tours, traveling a minimum of 60 total floors`

5 Dialing Dice

In a certain gambling town, dice have become so popular that they are even used to dial phone numbers. Each face of a six-faced die has a single digit printed on it. The dialing process works as follows. Given a phone number, which is simply a string of digits, one dials the first digit of the number by placing the die on the dialing board. The digit on the bottom face of the die is automatically dialed. To dial the next digit, the die is turned over so that one of the adjacent sides is now on the bottom. Again the digit on the new bottom face is dialed. The procedure continues until all the digits of the target phone number are dialed.

Unfortunately, as you might imagine, there are quite a few problems with this dialing method. First, the standard dice (with faces labeled 1 through 6) are not capable of dialing certain crucial numbers such as 911. To remedy this situation, people were allowed to "program" their dice by choosing any digits to place on the faces (two faces may contain the same digit).

As it turns out, this remedy still does not fully solve the problem, since no die can dial 1234567 (there are 7 digits, but only 6 sides on a die). When this was discovered, the people threw up their hands along with their dice and went back to their gambling. A few days of mulling over the problem lead to a new solution: people would be only required to dial a number that has small discrepancy with respect to the number that they really want to dial. The *discrepancy* between the two numbers is the minimum number of additions, deletions, or substitutions of digits required to transform one number into the other. For example, the discrepancy between 91 and 911 is 1, between 12399 and 1499 is 2, etc.

People often wondered about the optimal way to program their dice. Your task is to write a program to help them out. Given a target number N, program a die so that the die can be used to dial some number N', where the discrepancy between N and N' is minimized. Note that the die can be programed with any digits, regardless of the target phone number.

(You might notice other difficulties with this dialing system, but those are to be solved in a future task.)

Input

Each line of the input contains exactly one phone number of length $1 \leq N \leq 100$. While the number may contain any digits from 0 to 9, a given number contains at most 7 distinct digits.

Output

For each number, output the minimum discrepancy that can be obtained by any die and the 6 digits on the die that achieves that discrepancy. Sort the digits in ascending order. If there are ties, print out the sequence of digits that is lexicographically smallest.

Example

Sample Input	Sample Output
000	Dice 1: discrepancy is 0, digits used: 0 0 0 0 0 0
000112222	Dice 2: discrepancy is 0, digits used: 0 0 1 1 2 2
1233456	Dice 3: discrepancy is 1, digits used: 1 2 3 3 4 5

6 Procrastination

Once upon a time, there were two graduate students that were best friends. During their short breaks from research (usually, not longer than several hours), the two students liked to play the game of Procrastination.

The game of Procrastination is for two players (black and white), who take turns moving. The game involves removing cubes from towers. Each cube is either black or white. At the start of the game, these cubes are arranged into 4 towers: each tower is a stack of several cubes. On a player's turn, he can remove any cube that matches his color (the white player removes only white cubes, and the black player only removes black cubes). All the cubes above the chosen cube are also removed from the tower, irrespective of color. For example, suppose a tower is composed of the following cubes (from bottom to top): black, white, black, white. Then, if black removes the bottom-most black cube, he removes the entire tower; black can also take the 3rd cube, removing the 4th cube with it. If white removes the 2nd cube, then only one black cube will remain; white can also take the 4th cube. If a player cannot remove any cubes, he loses.

Having already been trained in the intricacies of the game during their undergraduate years, the two students learned to play the game perfectly, i.e., if a player had a winning strategy, then that player would win the game. However, at some time, they discovered that, for most starting configurations, one of the players has the winning strategy irrespective of which player moves first. They called a configuration a *W-configuration* if white has a winning strategy irrespective of who moves first, and a *B-configuration* if black has a winning strategy irrespective of who moves first.

Moreover, the friends noted that some partial configurations are at least as favorable for one player as other configurations. A partial configuration C is defined as a set of 3 towers; note that a partial configuration C together with a 4th tower T forms a complete game configuration, which we denote as (C, T). A formal definition of the notion "at least as favorable" is as follows. A partial configuration C_1 (composed of 3 towers) is *at least as favorable for white* as another partial configuration C_2 (also composed of 3 towers) if and only if for any 4th tower T, if (C_2, T) is a W-configuration then (C_1, T) is also a W-configuration. In other words, there does not exist a 4th tower T such that (C_1, T) is not a W-configuration and (C_2, T) is a W-configuration.

Given two partial configurations C_1 and C_2, you are to check whether C_1 is at least as favorable for white as the partial configuration C_2.

Input

The first line of the input contains an integer, the number of test cases. A test case includes one line with `Test N`, where `N` is the current test case number followed by eight lines, specifying the two partial configurations C_1 and C_2 in this order. Each configuration is specified by four lines.

The first line of the partial configuration contains three numbers: n_1, n_2, n_3 denoting the heights of the three towers of the partial configuration ($0 \leq n_1, n_2, n_3 \leq 50$). The second

line contains n_1 letters (`B` or `W`) separated by spaces describing the first tower. The third line contains n_2 letters separated by spaces describing the second tower. The fourth line contains n_3 letters separated by spaces describing the third tower. A letter `W` denotes a white cube and the letter `B` denotes a black cube. Each tower is described in the bottom-to-top order.

Output

For each test case, print on a separate line the test case number and `Yes` if C_1 is at least as favorable for white as the partial configuration C_2, and `No` otherwise.

Example

Sample Input	Sample Output
2 Test 1 3 3 1 W B B W B W B 3 3 3 B W W B W W W B B Test 2 3 3 2 W B B W B W B B 3 3 3 B W W B W W W B B	Test 1: Yes Test 2: No

7 Coneology

A student named Round Square loved to play with cones. He would arrange cones with different base radii arbitrarily on the floor and would admire the intrinsic beauty of the arrangement. The student even began theorizing about how some cones *dominate* other cones: a cone A dominates another cone B when cone B is completely within the cone A. Furthermore, he noted that there are some cones that not only dominate others, but are themselves dominated, thus creating complex domination relations. After studying the intricate relations of the cones in more depth, the student reached an important conclusion: there exist some cones, *all-powerful cones*, that have unique properties: an all-powerful cone is not dominated by any other cone. The student became so impressed by the mightiness of the all-powerful cones that he decided to worship these all-powerful cones.

Unfortunately, after having arranged a huge number of cones and having worked hard on developing this grandiose cone theory, the student become quite confused with all these cones, and he now fears that he might worship the wrong cones (what if there is an evil cone that tries to trick the student into worshiping it?). You need to help this student by finding the cones he should worship.

Input

The input file specifies an arrangement of the cones. There are in total N cones ($1 \leq N \leq 40000$). Cone i has radius and height equal to R_i, $i = 1 \ldots n$. Each cone is hollow on the inside and has no base, so it can be placed over another cone with smaller radius. No two cones touch.

The first line of the input contains the integer N. The next N lines each contain three real numbers R_i, x_i, y_i separated by spaces, where (x_i, y_i) are the coordinates of the center of the base of cone i.

Output

The first line of the output file should contain the number of cones that the student should worship. The second line contains the indices of the cones that the student should worship in increasing order. Two consecutive numbers should be separated by a single space.

Example

Sample Input	Sample Output
5 1 0 -2 3 0 3 10 0 0 1 0 1.5 10 50 50	2 3 5

8 Explaining the Stock Market

Mark Stockle has taken interest in analyzing the stock market. Recently, he has focused on a particular company's stock. He observes the price of this stock over a period of N days. Then, he stares at these numbers for a long time and tries to identify the patterns and trends in the price. For example, he might notice that near the beginning of the N days, the price was decreasing, but then it steadily rising later. Or he might notice that generally the price was going up with the occasional daily dip. However, he was unable to come up with any rigorous or general approach for analyzing the stock.

One day, his friend suggested the following strategy: he ought to break up the problem of explaining N days of prices into smaller manageable subproblems. In particular, he would divide up the N days into K sets, and explain each set using a single hypothesis. Note that the days in a set do not have to be contiguous.

In his machine learning class, Mark learned the power of Occam's Razor, which was that simple hypotheses that explain the data tend to generalize better on future data. It was so important for him to have simple hypotheses that he only allowed two types of hypotheses: that the prices of the stock on the days in a set would not increase or not decrease over time. Of course, there must be at least two days per set, since it does not make sense to explain the stock price of one day by itself. Lastly, the most important method to impose simplicity was to have as few hypotheses as possible, i.e. minimize K.

Input

The first line contains N ($1 \leq N \leq 30$), the number of days for which Mark has collected stock prices. The next N lines contain N integers in the range $[0, 100]$, one on each line. These integers represent the stock prices over N days.

Output

On a single line, output K, the minimum number of hypotheses needed to explain the stock data. If it is impossible to explain the data using K hypotheses as described above, simply output 0.

Example

Sample Input	Sample Output
8 1 6 3 5 4 2 7 0	2