NWERC 2007

The 2007 ACM Northwestern European Programming Contest
Utrecht University, The Netherlands

The Problem Set

Almost blank page

A Assemble

Recently your team noticed that the computer you use to practice for programming contests is not good enough anymore. Therefore, you decide to buy a new computer.

To make the ideal computer for your needs, you decide to buy separate components and assemble the computer yourself. You need to buy exactly one of each type of component.

The problem is which components to buy. As you all know, the quality of a computer is equal to the quality of its weakest component. Therefore, you want to maximize the quality of the component with the lowest quality, while not exceeding your budget.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with two integers: $1 \leq n \leq 1\,000$, the number of available components and $1 \leq b \leq 1\,000\,000\,000$, your budget.
- n lines in the following format: "`type name price quality`", where `type` is a string with the type of the component, `name` is a string with the unique name of the component, `price` is an integer ($0 \leq$ `price` $\leq 1\,000\,000$) which represents the price of the component and `quality` is an integer ($0 \leq$ `quality` $\leq 1\,000\,000\,000$) which represents the quality of the component (higher is better). The strings contain only letters, digits and underscores and have a maximal length of 20 characters.

It will always possible to construct a computer with your budget.

Output

Per testcase:

- One line with one integer: the maximal possible quality.

Sample in- and output

Input

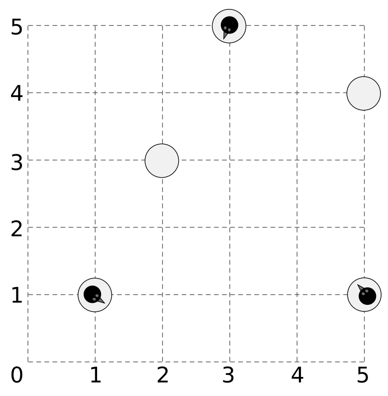
```
1
18 800
processor 3500_MHz 66 5
processor 4200_MHz 103 7
processor 5000_MHz 156 9
processor 6000_MHz 219 12
memory 1_GB 35 3
memory 2_GB 88 6
memory 4_GB 170 12
mainbord all_onboard 52 10
harddisk 250_GB 54 10
harddisk 500_FB 99 12
casing midi 36 10
monitor 17_inch 157 5
monitor 19_inch 175 7
monitor 20_inch 210 9
monitor 22_inch 293 12
mouse cordless_optical 18 12
mouse microsoft 30 9
keyboard office 4 10
```

Output

```
9
```

B March of the Penguins

Somewhere near the south pole, a number of penguins are standing on a number of ice floes. Being social animals, the penguins would like to get together, all on the same floe. The penguins do not want to get wet, so they have use their limited jump distance to get together by jumping from piece to piece. However, temperatures have been high lately, and the floes are showing cracks, and they get damaged further by the force needed to jump to another floe. Fortunately the penguins are real experts on cracking ice floes, and know exactly how many times a penguin can jump off each floe before it disintegrates and disappears. Landing on an ice floe does not damage it. You have to help the penguins find all floes where they can meet.



A sample layout of ice floes with 3 penguins on them.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with the integer N ($1 \leq N \leq 100$) and a floating-point number D ($0 \leq D \leq 100\,000$), denoting the number of ice pieces and the maximum distance a penguin can jump.
- N lines, each line containing x_i, y_i, n_i and m_i, denoting for each ice piece its X and Y coordinate, the number of penguins on it and the maximum number of times a penguin can jump off this piece before it disappears ($-10\,000 \leq x_i, y_i \leq 10\,000$, $0 \leq n_i \leq 10$, $1 \leq m_i \leq 200$).

Output

Per testcase:

- One line containing a space-separated list of 0-based indices of the pieces on which all penguins can meet. If no such piece exists, output a line with the single number -1.

Sample in- and output

Input	Output

`2`
`5 3.5`
`1 1 1 1`
`2 3 0 1`
`3 5 1 1`
`5 1 1 1`
`5 4 0 1`
`3 1.1`
`-1 0 5 10`
`0 0 3 9`
`2 0 1 1`
	`1 2 4`
	`-1`

C Containers

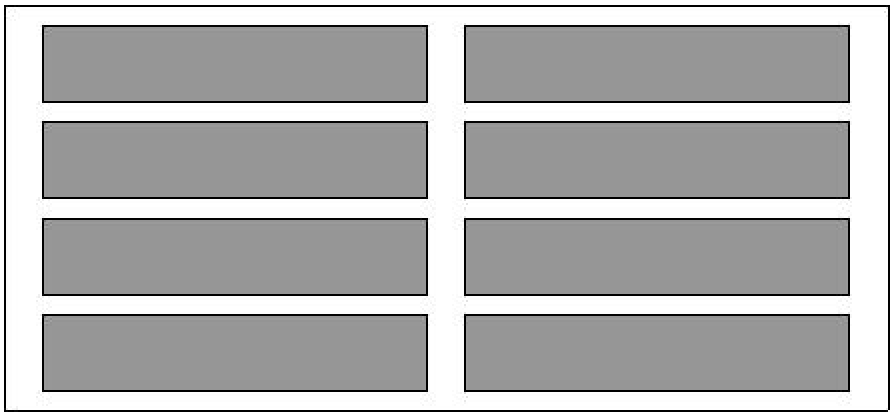
At a container terminal, containers arrive from the hinterland, one by one, by rail, by road, or by small ships. The containers are piled up as they arrive. Then the huge cargo ships arrive, each one capable of carrying thousands of containers. The containers are loaded into the ships that will bring them to far away shores. Or the other way round, containers are brought in over sea, piled up, and transported to the hinterland one by one. Anyway, a huge parking lot is needed, to store the containers waiting for further transportation.

Building the new container terminal at the mouth of the river was a good choice. But there are disadvantages as well. The ground is very muddy, and building on firm ground would have been substantially cheaper. It will be important to build the parking lot not larger than necessary.

A container is 40 feet long and 8 feet wide. Containers are stacked, but a stack will be at most five containers high. The stacks are organized in rows. Next to a container stack, and between two container stacks (along the long side of the containers) a space of 2 feet is needed for catching the containers. Next to a row of stacks, and between two stacks (along the short side of the containers) a space of 4 feet is needed for the crane that lifts the containers. All containers are placed in the same direction, as the cranes can not make turns on the parking lot.

The parking lot should be rectangular. Given the required capacity of the parking lot, what will be the best dimension for the parking lot? In the first place the area should be minimal. The second condition is that the parking lot should be as square as possible.

Below you see a plan for a parking lot with a capacity of 8 stacks. Two rows of four containers each turns out to be the best solution here, with a total area of $92 \times 42 = 3864$.



A parking lot with 8 container stacks.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- A single positive integer n ($n \leq 10^{12}$) on a single line: the required capacity (number of containers) for the parking lot.

Output

Per testcase:

- A single line, containing the length, width (length $\geq$ width) and area of the optimal solution. The optimal solution has the least possible area, and if there are multiple solutions having the same area, the difference length $-$ width should be minimal.

Use the sample format.

Sample in- and output

Input	Output

`6`
`1`
`15`
`22`
`29`
`36`
`43`
	`48 X 12 = 576`
	`48 X 32 = 1536`
	`52 X 48 = 2496`
	`92 X 32 = 2944`
	`92 X 42 = 3864`
	`136 X 32 = 4352`

D Youth Hostel Dorm

The Utrecht Youth Hostel has a giant dorm which usually accommodates all customers easily. With NWERC in town, however, lots of people would like to stay there and all the space available in the dorm should be used as efficiently as possible. You are assigned to provide the dorm layout.

The size of the dorm is given and the layout should consist of a map of that particular size. The map should display one '`E`', the point of entrance of the dorm, and furthermore '`B`'s and '`.`'s, indicating beds and empty spaces. The entrance should be located somewhere on the boundary of the dorm and every single bed should be reachable by starting at the entrance and walking through empty squares only. You can only walk in vertical and horizontal directions.

The provided layout should contain as many beds as possible.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with two integers l and w with $1 \leq l, w \leq 8$: the size of the dorm.

Output

Per testcase:

- l lines with w characters: the dorm layout. Any layout with the maximum number of beds is correct.

Sample in- and output

Input	Output

```
3
1 1
4 7
3 8
```
	```
	E
	B.B.BEB
	B.BBB.B
	B.....B
	B.BBB.B
	BBBBBBBB
	.......E
	BBBBBBBB
	```

Almost blank page

E Escape from Enemy Territory

A small group of commandos has infiltrated deep into enemy territory. They have just accomplished their mission and now have to return to their rendezvous point. Of course they don't want to get caught even if the mission is already over. Therefore they decide to take the route that will keep them as far away from any enemy base as possible.

Being well prepared for the mission, they have a detailed map of the area which marks all (known) enemy bases, their current position and the rendezvous point. For simplicity, we view the the map as a rectangular grid with integer coordinates (x, y) where $0 \leq x < X$, $0 \leq y < Y$. Furthermore, we approximate movements as horizontal and vertical steps on this grid, so we use Manhattan distance: $\text{dist}((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1|$. The commandos can only travel in vertical and horizontal directions at each step.

Can you help them find the best route? Of course, in case that there are multiple routes that keep the same minimum distance to enemy bases, the commandos want to take a shortest route that does so. Furthermore, they don't want to take a route off their map as it could take them in unknown, dangerous areas, but you don't have to worry about unknown enemy bases off the map.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with three positive numbers N, X, Y. $1 \leq N \leq 10\,000$ is the number of enemy bases and $1 \leq X, Y \leq 1\,000$ the size of the map: coordinates x, y are on the map if $0 \leq x < X, 0 \leq y < Y$.
- One line containing two pairs of coordinates x_i, y_i and x_r, y_r: the initial position of the commandos and the rendezvous point.
- N lines each containing one pair of coordinates x, y of an enemy base.

All pairs of coordinates are on the map and different from each other.

Output

Per testcase:

- One line with two numbers separated by one space: the minimum separation from an enemy base and the length of the route.

Sample in- and output

Input

```
2
1 2 2
0 0 1 1
0 1
2 5 6
0 0 4 0
2 1
2 3
```

Output

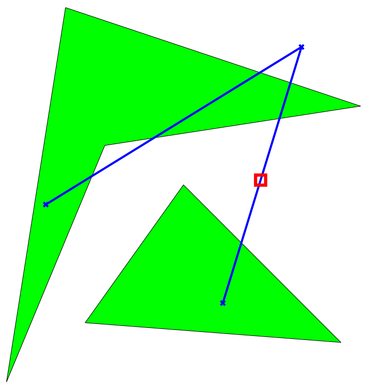
```
1 2
2 14
```

F Flight Safety

Safety is an important issue when planning flights. First and foremost, one should of course take every possible measure to make sure that the trip is uneventful and that no incidents occur. But even then, one should always be prepared for the worst and try to make sure that if an incident does happen, people's chances of surviving are as high as possible.

When making an emergency landing over water, the distance to the nearest land is a critical factor. In general, the further out on open waters, the worse are the odds of survival. Thus, one important safety parameter of a flight is how far away from the nearest land any part of the flight will take you. Your job is to write a program which, given a flight route, will determine this distance.

To simplify matters, we model the world as a 2-dimensional plane rather than a sphere. We model continents as polygons, and a flight route as a sequence of key points connected by straight line segments. Flight routes always start and end strictly inside a continent, but intermediate key points may be located over water. Continents do not intersect themselves or touch each other.



Second sample case (furthest point marked with a square).

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line containing two integers C ($1 \le C \le 20$) and N ($2 \le N \le 20$), where C is the number of continents and N is the number of key points in the flight route.
- N lines each containing two integers X, Y giving the coordinates of the key points, from first to last.
- The descriptions of the C continents. Each continent description starts with a line containing an integer M ($3 \le M \le 30$) giving the number of vertices of this continent. It is followed by M lines, each containing a pair of integers X, Y giving the coordinates of the M vertices, in either clockwise or counter-clockwise order.

Every coordinate in the input is between $-10\,000$ and $10\,000$.

Output

For each test case:

- One line with the furthest distance from land that the flight route will go. The answer should be given with an absolute or relative error of at most 10^{-3}.

Sample in- and output

Input	**Output**
`2`	`0.000000`
`1 2`	`2.942685`
`-9 -6`	
`5 1`	
`3`	
`0 16`	
`-16 -12`	
`17 -6`	
`2 3`	
`12 4`	
`16 17`	
`3 9`	
`4`	
`1 0`	
`4 19`	
`19 14`	
`6 12`	
`3`	
`10 10`	
`5 3`	
`18 2`	

G Summits

You recently started working for the largest map drawing company in the Netherlands. Part of your job is to determine what the summits in a particular landscape are. Unfortunately, it is not so easy to determine which points are summits and which are not, because we do not want to call a small hump a summit. For example look at the landscape given by the sample input.

We call the points of height 3 summits, since there are no higher points. But although the points of height 2, which are to the left of the summit of height 3, are all higher than or equal to their immediate neighbours, we do not want to call them summits, because we can reach a higher point from them without going to low (the summits of height 3). In contrast, we do want to call the area of height 2 on the right a summit, since if we would want to walk to the summit of height 3, we first have to descend to a point with height 0.

After the above example, we introduce the concept of a d-summit. A point, with height h, is a d-summit if and only if it is impossible to reach a higher point without going through an area with height smaller than or equal to $h - d$.

The problem is, given a rectangular grid of integer heights and an integer d, to find the number of d-summits.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with three integers $1 \leq h \leq 500$, $1 \leq w \leq 500$ and $1 \leq d \leq 1\,000\,000\,000$. h and w are the dimensions of the map. d is as defined in the text.
- h lines with w integers, where the xth integer on the yth line denotes the height $0 \leq h \leq 1\,000\,000\,000$ of the point (x, y).

Output

Per testcase:

- One line with the number of summits.

Sample in- and output

Input	Output

```
1
6 10 2
0 0 0 0 0 0 0 0 0 0
0 1 2 1 1 1 1 0 1 0
0 2 1 2 1 3 1 0 0 0
0 1 2 1 3 3 1 1 0 0
0 2 1 2 1 1 1 0 2 0
0 0 0 0 0 0 0 0 0 0
```
	`4`

Almost blank page

H Obfuscation

It is a well-known fact that if you mix up the letters of a word, while leaving the first and last letters in their places, words still remain readable. For example, the sentence "tihs snetncee mkaes prfecet sesne", makes perfect sense to most people.

If you remove all spaces from a sentence, it still remains perfectly readable, see for example: "thissentencemakesperfectsense", however if you combine these two things, first shuffling, then removing spaces, things get hard. The following sentence is harder to decipher: "tihssnetnceemkaesprfecetsesne".

You're given a sentence in the last form, together with a dictionary of valid words and are asked to decipher the text.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with a string s: the sentence to decipher. The sentence consists of lowercase letters and has a length of at least 1 and at most 1 000 characters.
- One line with an integer n with $1 \leq n \leq 10\,000$: the number of words in the dictionary.
- n lines with one word each. A word consists of lowercase letters and has a length of at least 1 and at most 100 characters. All the words are unique.

Output

Per testcase:

- One line with the deciphered sentence, if it is possible to uniquely decipher it. Otherwise "`impossible`" or "`ambiguous`", depending on which is the case.

Sample in- and output

Input

```
3
tihssnetnceemkaesprfecetsesne
5
makes
perfect
sense
sentence
this
hitehre
2
there
hello
hitehre
3
hi
there
three
```

Output

```
this sentence makes perfect sense
impossible
ambiguous
```

I Tower Parking

There is a new revolution in the parking lot business: the parking tower. The concept is simple: you drive your car into the elevator at the entrance of the tower, and the elevator and conveyor belts drag the car to an empty parking spot, where the car remains until you pick it up. When you return, the elevator and conveyor belts move your car back to the entrance and you're done.

The layout of the tower is simple. There is one central elevator that transports the cars between the different floors. On each floor there is one giant circular conveyor belt on which the cars stand. This belt can move in clockwise and counterclockwise direction. When the elevator arrives on a floor, it becomes part of the belt so that cars can move through it.

At the end of the day the tower is usually packed with cars and a lot of people come to pick them up. Customers are processed in a first come first serve order: the elevator is moved to the floor of the first car, the conveyor belt moves the car on the elevator, the elevator is moved down again, and so on. We like to know how long it takes before the last customer gets his car. Moving the elevator one floor up- or downwards takes 10 seconds and moving a conveyor belt one car in either direction takes 5 seconds.

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with two integers h and l with $1 \leq h \leq 50$ and $2 \leq l \leq 50$: the height of the parking tower and the length of the conveyor belts.
- h lines with l integers: the initial placement of the cars. The jth number on the ith line describes the jth position on the ith floor. This number is -1 if the position is empty, and r if the position is occupied by the rth car to pick up. The positive numbers form a consecutive sequence from 1 to the number of cars. The entrance is on the first floor and the elevator (which is initially empty) is in the first position. There is at least one car in the parking tower.

Output

Per testcase:

- One line with the number of seconds before the last customer is served.

Sample in- and output

Input	Output

```
2
1 5
-1 2 1 -1 3
3 6
-1 5 6 -1 -1 3
-1 -1 7 -1 2 9
-1 10 4 1 8 -1
```
	```
	25
	320
	```

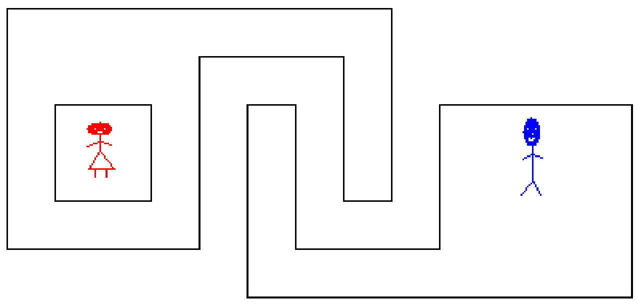
Almost blank page

J Walk

Alice would like to visit Bob. However, they live in a hilly landscape, and Alice doesn't like to walk in hills. She has a map of the area, showing the height curves. You have to calculate the total altitude climbed, and the total altitude descended, for the route which minimizes these numbers. It does not matter how far she has to walk to achieve this.

Since you don't know what the landscape looks like in between the height curves, you cannot know exactly how much climb and descent she will actually get in practice, but you should calculate the minimum possible under optimal conditions based on what you can deduce from the map.

The map is represented as an xy grid. Alice lives in $(0, 0)$, and Bob lives in $(100\,000, 0)$. The height curves are represented as polygons, where a polygon cannot intersect itself or another polygon. Furthermore, neither Alice nor Bob lives exactly on a height curve.



Second test case from sample input (compressed).

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line with $0 \leq N \leq 2\,500$, the number of height curves.
- One line for each height curve, with $1 \leq H_i \leq 1\,000$ being the height of the curve, $3 \leq P_i \leq 2\,000$ the number of vertices in the polygon, and the vertices $x_1, y_1, ..., x_{P_i}, y_{P_i}$ having integral values $-300\,000 \leq x_i, y_i \leq 300\,000$.

There will be no more than 200 000 polygon vertices in total in all test cases.

Output

Per testcase:

- One line with two numbers: the total altitude climbed and the total altitude descended.

Sample in- and output

Input

```
2
2
20 3 10 10 0 -10 -10 10
25 3 20 20 0 -20 -20 20
3
100 4 -1 1 1 1 1 -1 -1 -1
300 8 -2 2 2 2 2 -2 5 -2 5 1 6 1 6 -3 -2 -3
50 8 3 3 100001 3 100001 -1 7 -1 7 2 4 2 4 -1 3 -1
```

Output

```
5 0
200 250
```