2018 SBU Local Programming Contest (Sponsored by Google)

(Stony Brook University, Sep 26, 2018)

Welcome to the contest! We would like to remind you of several things:

- This contest is a local ACM-style contest. Each participant must compete alone (this is not a team contest).
- The contest lasts for 3 hours (7:30PM - 10:30PM).
- C / C++ / Java / Python would be supported. GCC 7.3.0, G++ 7.3.0, Java 8, PyPy 2.7 and PyPy 3.5 are used.
- All submissions read from stdin and output to stdout.
- You can write and test your code directly on the HackerRank platform.

If you have any question, don't hesitate to ask. Again, enjoy the contest!

Problem A: Going! Going! Going! Gone!

Problem Statement

Figure 1: The vanishing bowling ball!

I am pretty sure the bowling balls in our neighborhood "Vanishing Bowling Lanes" are made of onions. When you throw one of those balls the weird material it is made of come off layer by layer just like an onion. Sometimes all layers come off before the ball even hits a pin. The ball effectively vanishes and the frightened pins remain safe!

Now given the initial radius of a ball and the number of layers it has can you compute the distance it will go after being thrown and before vanishing (i.e., by the time all its layers completely come off)? You can assume that all layers have the same thickness. Figure 1 shows an example. Assuming that the ball has a diameter of 8 inches[1] and 4 layers of the same thickness, it goes 25.13 inches before its first layer comes off, 18.85 more inches before the second layer disappears, 12.57 more inches before the third layer goes, and finally an additional 6.28 inches before the last layer comes off. So, the ball goes almost 63 inches before it completely vanishes.

Input

A single line containing two integers: the diameter (in inches) of the ball d and the number of layers k (from surface to center) it has.

Constraints

- $1 \leq d \leq 20$
- $1 \leq k \leq 100$

Output

Output a floating point number giving the total distance in inches (rounded to three decimal places) the ball can travel before vanishing, on a single line.

Sample Input

```
8 4
```

Sample Output

```
62.832
```

[1] slightly lower than the standard 8.5 inches

Problem B: Romario and Julianne

Problem Statement

Chances are you haven't heard of "Romario and Julianne" – a romantic tragedy not written by William Shakespeare. The tragedy unfolds in a city of castles somewhere in medieval Europe, and guess what? At some point in the play the pair find themselves captive – unfortunately, in two separate rooms (which may or may not be in the same castle).

Unsurprisingly, Romario plans to flee to his Julianne. He has a map of all castles in the city and knows precisely where Julianne is. Unfortunately though the castles have a very complicated structure – rooms inside rooms inside rooms inside, well, again rooms. On top of that all doors are extremely heavy. There are some good news, too. The rooms are all of square shape with each wall going either from north to south or from east to west[2]. No two rooms intersect or even touch. Also, Romario can bribe the guards to collect the keys to all doors he needs to open.

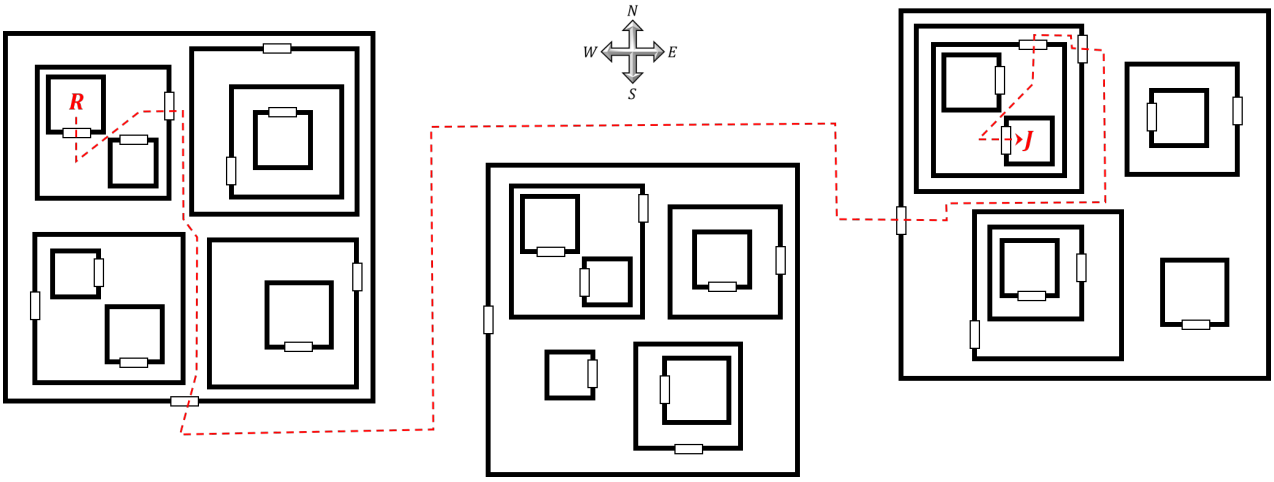

Figure 2: In this example, Romario (R) must open at least 7 doors to reach Julianne (J).

After such a long and exhausting captivity Romario does not have much strength left. So, he wants to find a path from his location to Julianne's which will allow him to open the fewest doors. Can you help him to figure that out? To start with he only wants to know the minimum number of doors he must open. The path is not needed at this point.

Input

The first line contains 4 integers in the following order: x_r, y_r, x_j, y_j giving the coordinates (x_r, y_r) and (x_j, y_j) of Romario and Julianne, respectively. The second line contains an integer N specifying the number of room coordinates to follow. The i-th of the next N lines contains 4 integers x_i, y_i, x_i', y_i' giving the coordinates (x_i, y_i) and (x_i', y_i') of the lower-left and upper-right corners of a room.

Constraints

- $1 \leq x_r, y_r, x_j, y_j \leq 10^6$
- $1 \leq x_i, y_i, x_i', y_i' \leq 10^6$
- $2 \leq N \leq 10^4$

[2] A castle can also be viewed as a square room with a door.

Output

Output the minimum number of doors Romario must open to reach Julianne on a single line.

Sample Input

```
35 94 66 38
12
8 5 110 107
10 20 50 60
20 40 30 50
40 30 45 35
30 80 50 100
31 91 36 96
40 85 44 89
55 70 85 100
60 80 78 98
65 85 70 90
60 30 75 45
65 35 70 40
```

Sample Output

```
4
```

Problem C: Recursive Storage on a Δ-NVM

Problem Statement

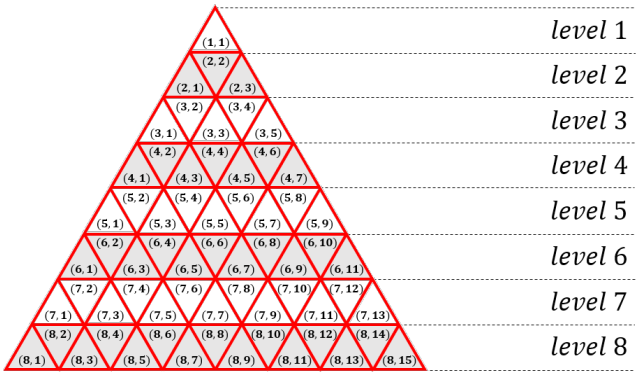

Figure 3: An equilateral triangular storage medium of side length 8. It contains $8^2 = 64$ storage cells.

This problem is about storing an array of numbers on an equilateral triangular storage medium which we will call a Δ-NVM (Δ non-volatile memory). A Δ-NVM of side length s has exactly s^2 storage cells. The cells are distributed among k levels with level $l \in [1, s]$ containing exactly $2l - 1$ cells. The cells in level l have coordinates $(l, 1), (l, 2), \ldots, (l, 2l - 1)$ from left to right. The side length (s) is always a power of 2, say, 2^k for some integer $k \geq 0$. Figure 3 shows a Δ-NVM of side length 8 and containing $8^2 = 64$ cells distributed among 8 levels.

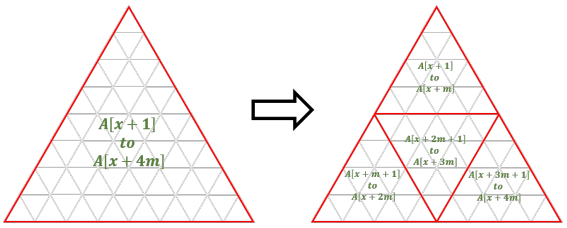

Figure 4: Recursive distribution of data inside an upright Δ-NVM.

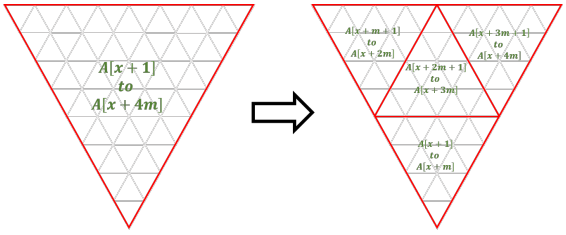

Figure 5: Recursive distribution of data inside an inverted Δ-NVM.

We will recursively store an array $A[1 \ldots s^2]$ of length $s^2 = \left(2^k\right)^2 = 4^k$ on a Δ-NVM of side length $s = 2^k$. We will view the whole Δ-NVM as a single upright triangle. But when we recursively subdivide it we will end up with both upright and inverted sub-triangles. Figure 4 shows how to divide a given upright equilateral triangle into four equal-size equilateral sub-triangles of which three are upright and one is inverted. If we are supposed to store $A[x + 1 \ldots 4m]$ on the original upright triangle we do that as follows: store $A[x + 1 \ldots m]$ on the top upright sub-triangle, $A[x + m + 1 \ldots 2m]$ on the bottom-left upright sub-triangle, $A[x + 2m + 1 \ldots 3m]$ on the bottom-center inverted sub-triangle, and $A[x + 3m + 1 \ldots 4m]$ on the bottom-right upright sub-triangle. Figure 5 shows how to do the decomposition when the given triangle is inverted. This distribution of data to sub-triangles happens recursively, that is, data is distributed among sub-sub-triangles inside the sub-triangles, then among sub-sub-sub-triangles, and so on and so forth.

Figure 6 shows how to recursively store $A[1 \ldots 64]$ on a Δ-NVM of side length 8 containing $8^2 = 64$ cells. For brevity the figure shows $A[i]$ only as i, and $A[i \ldots j]$ as i–j.

Now suppose you are given the side length 2^k of a Δ-NVM that stores $A[1 \ldots 4^k]$ recursively as described above and also the coordinates (l, c) of a cell in that NVM. Can you find out which entry of A is stored in that cell? You only need to report the index of the entry. For example, in Figure 6 cell $(4, 6)$ stores $A[15]$, and cell $(7, 5)$ stores $A[29]$.

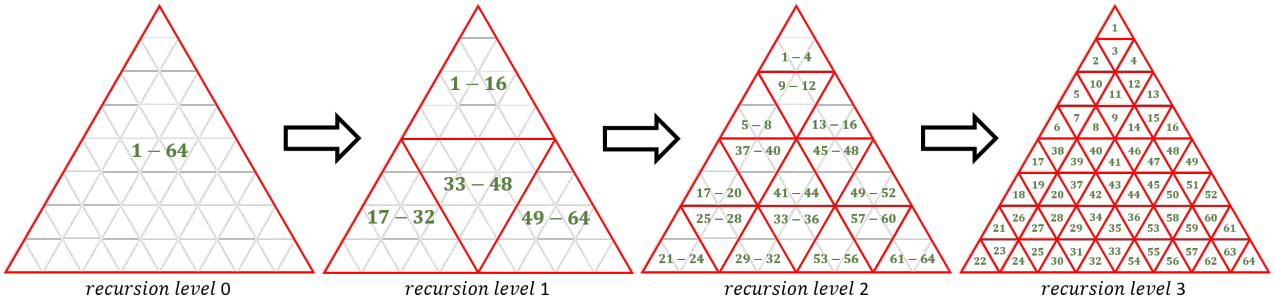

Figure 6: Recursively storing $A[1 \ldots 64]$ on a Δ-NVM of side length 8. The figure shows only indices to array A for brevity, e.g., $A[i]$ is shown only as i, and i–j means $A[i \ldots j]$.

Input

The first line contains a single integer T specifying the number of test cases to follow. Each test case appears on a separate line and contains three integers: the side length s of the Δ-NVM followed by a level number l and a column number c giving the (valid) coordinates of a cell in that NVM. The side length will be a power of 2.

Constraints

- $1 \leq T \leq 10^5$
- $1 \leq s < 2^{15}$
- $1 \leq l \leq s$
- $1 \leq c \leq 2s - 1$

Output

For each test case, output the index of the entry of A stores at the given cell on a single line.

Sample Input

```
3
1 1 1
4 3 4
8 7 6
```

Sample Output

```
1
12
34
```

Problem D: Major Muffin: A First-Person Eater (FPE) Game!

Problem Statement

To combat the popularity of "First-Person Shooter (FPS) Games" a completely new genre of non-violent computer games is emerging which we call the "First-Person Eater (FPE) Games". Following the way Wikipedia defines the FPS genre[3], the FPE genre is defined as follows: "**First-person eater (FPE)** is a video game genre centered around food and other eating-based combat in a first-person perspective; that is, the player experiences the action through the mouth of the protagonist."

"Major Muffin" is our first-ever FPE, and we need some major help from you in making its implementation efficient!

Unlike the players in FPS games, Major Muffin is equipped with a spoon and a dish instead of a weapon (e.g., sword/gun) and an armour (e.g., shield/vest), respectively. He loses health when he works and combats with other players for food and regains health when he eats. He has a health meter (HM) that keeps track of his health on a scale of 0 to k, where k is a positive integer specifying the maximum health he can reach. There are N different food items he can consume (in arbitrary order). There is only one piece of each food item. Each food f_i can boost his HM score by at most h_i. Indeed, if his current HM reading is $k' < k$ then f_i increases it to $\min\{k' + h_i, k\}$. But if $k' = k$, then Major Muffin can sell that food item to earn m_i dollars instead. That's how we discourage overeating in the game and instead encourage trading to get rich.

Figure 7: The Major Muffin FPE motto: Work hard to get healthier and richer!

Assuming that Major Muffin's initial HM reading is 0, what is the maximum dollar amount he can earn?

Input

The first line of the input contains an integer T that specifies the number of test cases to follow. Each test case starts with a line containing the two integers k and N. Then follows N lines, where the i-th line ($1 \leq i \leq N$) contains h_i and m_i.

Constraints

- $1 < T \leq 100$
- $1 \leq k \leq 1000$
- $1 \leq N \leq 1000$
- $0 \leq h_i, m_i \leq 1000$

[3] "**First-person shooter (FPS)** is a video game genre centered around gun and other weapon-based combat in a first-person perspective; that is, the player experiences the action through the eyes of the protagonist." – Wikipedia

Output

For each test case, output the maximum dollar amount Major Muffin can earn by selling his extra food items on a single line.

Sample Input

```
2
100 3
20 80
80 90
100 23
1000 3
0 23
900 290
99 555
```

Sample Output

```
170
0
```

Problem E: Editing Robot DNAs (Again!)

Problem Statement

In the SBU local contest last year, we solved a problem[4] on a DNA editing technique developed by an advanced robot civilization that existed "a long time ago in a galaxy far, far away." This year we have another one.

As you already know from last year's contest those robots' genomes consisted of DNA strands which were sequences of 0's and 1's[5]. The DNA editing approach we will consider for this problem is as follows. They started with a source strand (fragment) and repeatedly applied the following two simple operations until they could create the target strand (fragment): (OP_1) append a '0' to the current segment, and (OP_2) append a '1' and flip (i.e., reverse) the current fragment.

Figure 8 shows an example in which the source strand is transformable to the target strand.

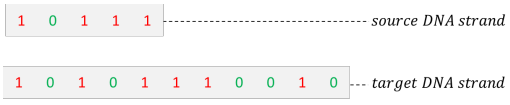

Figure 8: The source DNA given above can be transformed into the given target DNA (see Figure 9).

A sequence of operations that can make the transformation is shown in Figure 9 below.

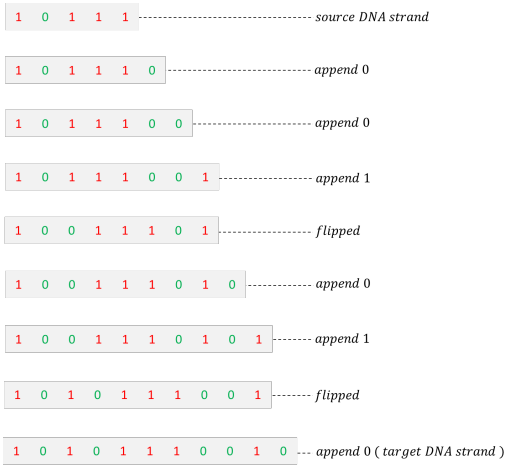

Figure 9: A solution (a series of operations) for the case shown in Figure 8.

Figure 10 shows an example in which the source strand cannot be transformed into the target strand using any sequence of OP_1 and OP_2.

Figure 10: In the case above, the given source DNA cannot be transformed into the given target DNA.

Now given a source strand and a target strand, this problem asks you to determine if the source can be transformed into the target using some sequence of OP_1 and OP_2.

[4] Problem G (Robot Gene Therapy), SBU Selection Contest 2017

[5] unlike, say, our DNA strands which are sequences of four nucleotide bases – adenine, guanine, cytosine, and thymine

Input

The first line of the input contains an integer $T(1 < T \leq 100)$ giving the number of test cases to follow. Each test case consists of two lines. The first line contains the source DNA fragment and the second line contains the target DNA fragment. Neither fragment will be longer than 50.

Constraints

- $1 < T \leq 100$
- The length of all source and target DNA fragments will not be longer than 50.

Output

For each test case, output "Possible" (without quotes) if the transformation is possible, and "Impossible" (again without quotes) otherwise. Output for each test case must be on a separate line.

Sample Input

```
2
10111
10101110010
10011
1010111
```

Sample Output

```
Possible
Impossible
```

Problem F: The Monkey and the Hot Air Balloon

Problem Statement

Figure 11: A monkey finds a hot air balloon floating above its backyard.

Once a monkey in Monkeyland found a hot air balloon floating above its backyard. It climbed up a tall tree and jumped into the gondola (i.e., balloon basket). It figured that if it loaded a banana stem (i.e., a bunch of bananas) on the gondola the balloon lost height but regained the lost height when the stem is thrown out. Being a clever monkey it immediately found a way to use the balloon for a publicity stunt for its banana distribution company.

The monkey decided to float over every city of the Monkeyland in some fixed sequence. It would start without any banana stem, and in every city it visited it would either (*i*) throw away a banana stem it already had and gain height, or (*ii*) load a new banana stem and lose height, or (*iii*) do nothing and keep the height unchanged. It wanted to end its journey with no banana stems on board.

The monkey wanted to make sure that more monkeys see its balloon go upward than downward. It knew the monkey population of each city it would visit.

Assume that whenever the balloon was above a city it either continuously went up or continuously went down or always remained at the same height, and every monkey in the city noticed what the balloon did. Let U and D be the total number of monkeys in all cities that saw the balloon go upward and downward, respectively. Now given the sequence of cities the monkey would visit along with the monkey population of each of them, and assuming that the monkey starts and ends the journey with zero banana stems, can you find the maximum value of $U - D$ the monkey could achieve?

Input

The first line of the input contains an integer T giving the number of test cases to follow. Each test case will start with a line containing a single integer N giving the number of cities the monkey would visit. The i-th of the next N lines gives the population p_i of the i-th city the monkey visited.

Constraints

- $1 < T \leq 1000$
- $2 \leq N \leq 3 \cdot 10^5$
- $1 \leq p_i \leq 10^6$

Output

For each test case, output a single line containing the maximum value of $U - D$.

Sample Input

```
2
9
10 5 4 7 9 12 6 2 10
20
3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4
```

Sample Output

```
20
41
```

Sample Output Explanation

In the first example, the monkey should load a banana stem in the second city, load another one in the third city, throw away a banana steam in the fifth city and another one in the sixth city. Finally, it should load a banana stem in the eighth city and throw it away in the last city. The value of $U - D$ is $(9 + 12 + 10) - (5 + 4 + 2) = 20$.

Problem G: Oh Gasoline! No Gasoline!

Problem Statement

Figure 12: Severe gasoline crisis in the island of Paradise.

A catastrophic hurricane has caused severe gasoline shortages in the island of Paradise recently by causing widespread damage to power lines and distribution terminals[6]. No gas station was operational for days.

Finally, one night the government transported gasoline from outside the island and delivered directly to every gas station on the island. The next day all gas stations opened at 6am and remained open until all customers were served. Only one gas pump per station was operational.

A very smart program was used to direct each customer to an appropriate station so that the last customer could be served as early as possible. The algorithm made use of the knowledge of the time (minutes elapsed since 6am) it would take for each customer to reach any given station. It assumed that serving any customer takes the same amount of time, say, 1 minute. Unfortunately, however, the program was lost somehow.

Could you please find and re-implement the algorithm so that it can be used to tackle similar gasoline crises in the future?

Input

The input will start with an integer T giving the number of test cases to follow. Each test case will start with a line containing two integers: the number of customers N and the number of gas stations M. The i-th line of the next N lines contains M integers, where the j-th integer is the time t_{ij} it would take for customer i to reach gas station j.

Constraints

- $1 < T \leq 100$
- $1 \leq N, M \leq 100$
- $1 \leq t_{ij} \leq 100$

[6] This problem is motivated by the severe gasoline shortages caused by Hurricane Sandy in 2012.

Output

For each test case, output a single line containing the minimum time (in minutes) it takes to serve all customers.

Sample Input

```
1
5 3
1 3 2
2 4 1
2 2 1
3 1 2
4 2 1
```

Sample Output

```
3
```