ICPC ASIA DHAKA REGIONAL CONTEST 2022

Hosted by

March 11, 2023

You get 15 Pages, 10 Problems & 300 Minutes

Partnered with

DIGITAL BANGLADESH

ICT DIVISION

BANGLADESH COMPUTER COUNCIL
COMPUTER FOR EVERYTHING

Sponsored by

US-Bangla Group

Therap (BD) Ltd.
Person-Centered. Data-Driven.

In Association with

প্রথম আলো

Media Partners

DHAKA POST

আজকের পত্রিকা

BANGLA VISION

Problem A

Tree Flip

Alu will be given a rooted tree where each node contains an integer value of 0 or 1. He can perform flip operations at some nodes. This operation will flip the value of the selected node and the values of all the children of the selected node. Flipping a value means changing 0 to 1 or 1 to 0. Alu is intelligent so he performs this operation the minimum number of times to make the values of all nodes zero.

However, it's not fun if the tree is static. So here comes Begun into the scene. He has a tree. He can perform following updates on his tree:

- Update 1: Begun flips the value of a particular node (note, the value of the children of the selected node are not flipped).
- Update 2: Begun makes a particular node the root of the tree.

After each update, you need to output the minimum number of operations Alu requires to perform to make the tree zero. Note, Alu does not change the Begun's tree. You may imagine that Alu performs his operations on a copy of Begun's tree.

Input

Input begins with the number of test cases, **T**. Each case begins with two positive integers, the number of nodes **n** and the number of updates **q**. Next line contains **n** integers, the **i**'th integer is the value of the **i**'th node. Each of the next **n - 1** lines consists of two integers: **u** and **v**. These describe the edges of the tree. Each of the next **q** lines contains two integers: **id x**. id = 1 denotes "Update 1" and id = 2 denotes "Update 2". x denotes the node on which the update is performed. You may assume that the input tree is valid. Initially **1** is the root of the tree.

Note,

- **1 ≤ T ≤ 10,000**
- Sum of **n** overall test cases ≤ 100,000
- Sum of **q** overall test cases ≤ 100,000
- **1 ≤ id ≤ 2**
- **1 ≤ x ≤ n**
- **1 ≤ u, v ≤ n**

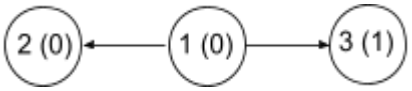
Output

For each case print the case number. Then for each update, print the answer.

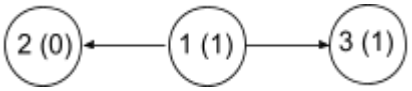
Sample Input	Output for Sample Input
1 3 3 0 0 1 1 2 3 1 1 1 2 2 1 1	Case 1: 2 1 1

Explanation

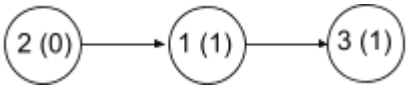
Initial tree. Inside the brace, you will find the value of that node. The arrows of the edges go from parent to child.



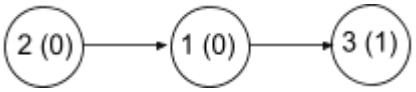
First update is **"1 1"**. It flips the value of the node **1**. Alu needs two operations. Operation on node **2**, followed by operation on node 1.



Second update is **"2 2"**. It makes the node **2** as the root of the tree. Alu needs only one operation on node **1** to make the entire tree zero.



Final update is **"1 1"**. It flips the value of the node **1**. Alu needs only one operation on node **3** to make the entire tree zero.



Terminology

Tree is a connected graph of **n** nodes and **n - 1** edges. A **rooted tree** is a tree where there is a node designated as **root** and all the edges are oriented away from the root. The **parent** of a node v is the node connected to v on the path from the root; every node has a unique parent except the root which has no parent. A **child** of a node v is a node of which v is the parent.

Problem B

Binary Sort

You are given a sequence of **n** positive integers $a_1, a_2, \ldots, a_n$ where $1 \le n \le 1000$ and $1 \le a_i \le 2^{30}-1$. You have to make the sequence strictly increasing by performing following operation zero or more times:

- Choose an integer a_i from the sequence where $1 \le i \le n$.
- And perform $a_i = a_i \mid (1 << p)$ where $0 \le p \le 59$.
  Here '|' denotes 'bitwise or' operation and '1 << p' means 2^p.

Now you need to find the minimum number of operations needed to convert the given sequence to a strictly increasing sequence. If it is not possible to make the sequence strictly increasing then output **-1**.

Input

The first line contains the number of test cases **T** where $1 \le T \le 10$. Then **T** test cases follow. The first line of each test case contains **n** $(1 \le n \le 1000)$ and the second line contains the sequence of **n** positive integers. The sum of **n** over all test cases is not larger than **5000**.

Output

Print the case number in a single line followed by the minimum number of operations needed or **-1** if there is no solution.

Sample Input	Output for Sample Input
3 5 5 2 1 3 4 3 1 3 9 2 2 2	Case 1: 4 Case 2: 0 Case 3: 1

Problem C

Network

Byteland is a country that can be modeled as a one dimensional axis of length **L**. You have **N** network towers that you want to set up in byteland, you want to place each tower at some position between **1 and L** (inclusive). You can only place towers at integer positions, you can't place more than one tower at the same place. Tower **i** has a range $\mathbf{R_i}$. Two towers **i** and **j** can only communicate if the distance between these towers doesn't exceed $\mathbf{min(R_i, R_j)}$. You want to place towers in byteland so that any two adjacent towers should be able to communicate with each other. Note, two adjacent towers need not be in the adjacent positions, there may be empty positions between two adjacent towers. You want to know how many possible ways there are. Two ways are considered different, if there is at least one tower i such that its position is not same in these two ways.

Input

The first line contains two integers **N ($1 \le N \le 100$)** and **L ($1 \le L \le 10^5$)**. The second line contains **N** integers $\mathbf{R_1, R_2 \ldots R_n}$ **($1 \le R_i \le 100$)**.

Output

Print a single integer with the answer. As the answer can be large, output it modulo 998244353.

Sample Input	Output for Sample Input
3 5 1 2 2	26

Explanation

Some possible placements for the sample case (0 means, no tower placed at this position)

12300	12030	21300
01230	01203	23100
02130	00123	20310
02310	00213	02031
00231		

Some incorrect placements

20301	The tower at position 3 and tower at position 5 can't communicate.
20031	The tower at position 1 and tower at position 4 can't communicate.

Problem D

Aspect Ratio

You had two identical rectangular monitors, each of which measured **d** inches diagonally. When you placed them side by side you could create a larger monitor that measured **k.d (k multiplied by d)** inches diagonally. You should assume that monitors had zero bezels (frame around the screen has width zero) and had no gaps in between when placed side by side. As many years have passed so now you cannot remember the size of the monitors but only remember the value of **k**. From this value of **k**, you will have to find the aspect ratio (Ratio of width and height of a monitor) of any one of the monitors (Both will obviously have the same aspect ratio).

Input

First line of the input file contains a positive integer **T (T ≤ 100000)** which denotes the number of test cases to follow. Each of the next **T** lines contains a floating point number which denotes a possible value of **k**. This floating-point number has **10** digits after the decimal point. The value of **k** will be within the range: **1.015** to **1.993**

Output

For each test case output the possible aspect ratio of the monitor. If the aspect ratio of the monitor is **16.5:9.5**, you should output $\frac{16.5}{9.5} = 1.7368$. So the output should be rounded to **4** digits after the decimal point. You can assume that for the given inputs there will always be an unique aspect ratio. You can also assume that the inputs will be such that for small precision errors the printed output will not differ.

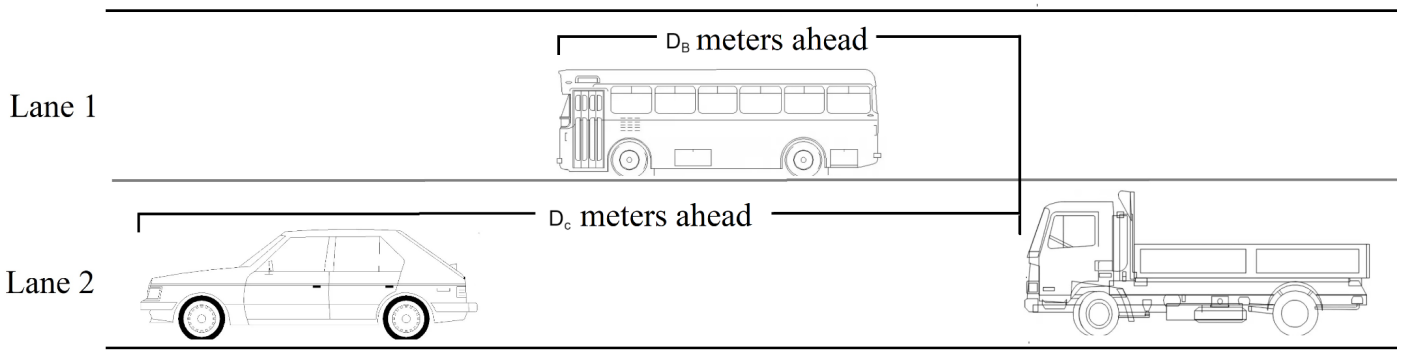
Sample Input	Output for Sample Input
1 1.2500000000	0.4804

Warning: Try to avoid endl with cout.

Problem E

Overtake

There is a car, a bus, and a truck moving in the same direction on an infinite road. The road has two lanes labeled **Lane 1** and **Lane 2**. The bus will keep moving forward staying on **Lane 1** with a constant speed $\mathbf{S_B}$ meters per second and the car will keep moving forward staying on **Lane 2** with a constant speed $\mathbf{S_C}$ meters per second. They will not change their lanes or speeds. The length of the bus is $\mathbf{L_B}$ meters and it is $\mathbf{D_B}$ meters ahead of the truck. The length of the car is $\mathbf{L_C}$ meters and it is $\mathbf{D_C}$ meters ahead of the truck. The length of the truck is $\mathbf{L_T}$ meters and the maximum speed of the truck is $\mathbf{S_T}$ meters per second.



The truck wants to overtake both of the vehicles without any accidents. The truck will overtake the car, if it is at least L_T meters ahead of the car. Similarly, the truck will overtake the bus, if it is at least L_T meters ahead of the bus.

If the truck is driven on lane 1, the truck must stay at least L_B meters behind the bus or at least L_T meters ahead of the bus to avoid an accident with the bus. Similarly, if the truck is driven on lane 2, the truck must stay at least L_C meters behind the car or at least L_T meters ahead of the car to avoid an accident with the car.

You may safely assume:

1) No accident has occurred till now.
2) The truck is either on lane 1 or lane 2, we do not know. The truck driver can instantly change between lane 1 and lane 2.
3) The truck driver can instantly change its current speed to any value between 0 to $\mathbf{S_T}$ meters per second.
4) Each lane contains sufficient space for only one vehicle and at any particular moment the truck will be driven either on lane 1 or lane 2.

You are given all the information, you have to find out the minimum amount of time the truck will take to overtake both of the vehicles.

Input

First line of the input file contains a positive integer **T ($\mathbf{T \le 100000}$)** which denotes the number of test cases to follow. Each of the next **T** test cases contains three lines of inputs. The first line of the test case contains three space-separated integers $\mathbf{L_B}$ **($\mathbf{1 \le L_B \le 100}$)**, $\mathbf{S_B}$ **($\mathbf{1 \le S_B \le 100}$)**, and $\mathbf{D_B}$ **($\mathbf{1 \le D_B \le 1000}$)**. The next line contains three space-separated integers $\mathbf{L_C}$ **($\mathbf{1 \le L_C \le 100}$)**, $\mathbf{S_C}$ **($\mathbf{1 \le S_C \le 100}$)**, and $\mathbf{D_C}$ **($\mathbf{1 \le D_C \le 1000}$)**. The last line of the case contains two space-separated integers $\mathbf{L_T}$ **($\mathbf{1 \le L_T \le 100}$)** and $\mathbf{S_T}$ **($\mathbf{1 \le S_T \le 100}$).**

You may safely assume that $L_C \le D_C$ or $L_B \le D_B$ or both are valid.

Output

For each case, print "**Case t: x**" (without quotes), where **t** is the test case number and **x** is the minimum time to overtake both of the vehicles rounded up to four digits after decimal points. If it is impossible to overtake both vehicles, then print "**-1**" instead.

Sample Input	Output for Sample Input
2 5 10 10 5 10 10 50 20 5 10 100 5 5 10 10 100	Case 1: -1 Case 2: 1.2222

Warning: Large I/O, Avoid endl with cout.

Problem F

2x2 Flip

When Alice is given a binary “m x n” matrix (a matrix with 0s and 1s with m rows and n columns), she tries to make it all zero using 2x2 flip operation. In this operation, she selects a 2x2 sub-matrix and flips the digits in it from 0 to 1 or 1 to 0.

Now it’s none other than Bob who gives Alice the matrix to make it zero. However, Bob is not as smart as Alice, so the matrix Bob hands over to Alice might not be one that can be converted to all zeros using the 2x2 flip operations. So Alice is also permitted to perform at most K additional 1x1 flip operations before she performs the 2x2 operations. She can flip a certain 1x1 cell as many times as she wishes (as long as the total number of 1x1 flip is at most K).

For a given matrix, find the number of ways Alice can perform 1x1 flip operation at most K times so that after these operations she can make the entire matrix zero using 2x2 flip operations. Two sequences of flipping are different if they are of different length or the i’th cell in one sequence is different from the other sequence.

For example, for following matrix:

```
100
000
101
```

Alice can flip the upper right hand corner to get

```
101
000
101
```

Although Alice could not convert the initial matrix to zero using only 2x2 flips, she can convert the second matrix above to all zero in following way:

```
101      011      000      000      000
000  ->  110  ->  101  ->  011  ->  000
101      101      101      011      000
```

Input

Input begins with the number of test cases T. Each case begins with 4 integers: m, n, K, b. Here, b is the number of 1s in the matrix. Then follow b pairs of integers: ri, ci, denoting the places of 1s.

NB: Please check the clarification for the constraints. Constraints are available in CodeMarshal.

Output

Output case number followed by the answer. Since the number can be big, output the answer in modulo 998244353.

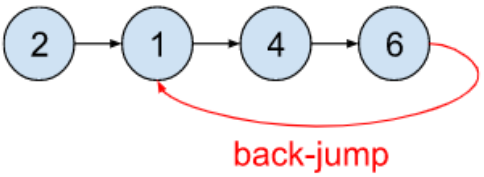
Sample Input	Output for Sample Input
2 3 3 1 3 1 1 3 1 3 3 5 5 100 0	Case 1: 1 Case 2: 753608861

Problem G

Jump If You Can

You are given an **undirected** graph with **n** nodes and **m** **weighted** edges. The weight of the edge **w** connecting two nodes **u** and **v** denotes the cost of going from **u** to **v** or vice versa. You need to go from starting node **s** to destination node **t** within total cost **c** with a **positive** energy level left.

At the beginning of the journey, you have an initial energy level. While moving from one node to an adjacent node, the energy level decreases by **1**. If your energy level becomes **0**, you must back-jump to any previously visited node (not the current node) with cost **d**. Note that, even if you reach the destination node with energy level **0**, you have to back-jump. You only finish the journey if you reach the destination node **t** with a **positive** energy level. Moreover, you can also back-jump from your current node even if you still have some energy. After any back-jump, you regain all the lost energies and your energy becomes the **initial value again**.



See the given figure for clarity (directed edges are used only to demonstrate the path). You started with initial energy level 3 from node 2. After reaching node 1, your energy level became 2. After reaching node 4, your energy level became 1, and after reaching node 6, your energy level became 0. So, you have to jump back. Now for a back-jump, you had three options: node 2, node 1 and node 4. So, if you jump back to any of these three nodes (in the figure, it is node 1), your energy level will become 3 again.

You need to find the minimum initial energy level required to go from starting node **s** to destination node **t** with total cost not exceeding **c** or determine if it's impossible to do so.

Input

The first line contains an integer **T** (**1 ≤ T ≤ 10**) denoting the number of test cases. The first line of each test case contains six integers **n**, **m**, **s**, **t**, **c**, **d** (**2 ≤ n ≤ 500**, **1 ≤ m ≤ 500**, **1 ≤ s**, **t ≤ n**, **s ≠ t**, **1 ≤ c**, **d ≤ 10^9**). Each of the next **m** lines contains three integers **u**, **v**, **w** (**1 ≤ u**, **v ≤ n**, **u ≠ v**, **1 ≤ w ≤ 10^9**) denoting an edge between node **u** and **v** with cost **w**.

Output

For each case, print "**Case t: x**" (without quotes), where **t** is the test case number and **x** is an integer denoting the minimum energy or a string "**Impossible**" (without quotes) if not possible.

Sample Input	Output for Sample Input
2 5 4 1 5 10 1 1 2 1 2 3 2 3 4 3 4 5 4 3 2 1 3 2 1 1 2 1 2 3 2 2 1 1 2 12 1 1 2 10	Case 1: 5 Case 2: Impossible Case 3: 1

Problem H

Product Manager

A factory has **N** Product Quality Analysts (PQA) working in a single line. The first PQA is standing at the start of the product line and the N^{th} PQA is standing at the end of the product line. The first PQA starts the checking of a product and takes t_1 minutes and then hands it over to the 2^{nd} PQA. More formally, the **i^{th}** PQA takes **t_i** minutes time to check a product and then hand it over to the $(i+1)^{th}$ PQA. When the N^{th} PQA completes his inspection for a product, we call the product a **"Verified Product"**.

For example, Let there be 4 PQAs working in the factory and they take 2, 3, 1, and 4 minutes time respectively to check a product. Product checking timeline these PQA is as follows:

Product		By PQA_1	By PQA_2	By PQA_3	By PQA_4
1^{st} Product	Start of checking	1	3	6	7
	End of checking	2	5	6	10
2^{nd} Product	Start of checking	3	6	9	11
	End of checking	4	8	9	14

So, the first product will be verified at the end of the 10th minute, and the second product will be verified at the end of the 14th minute.

The manager of the company wants to arrange these N product quality analysts in such a way that minimizes the production time for the first K verified products. You have to calculate how many ways he can do so.

Input

Input starts with an integer **T ($1 \le T \le 1001$)** denoting the number of test cases. Each of the test cases starts with two integers **N ($1 \le N \le 20$)** and **K ($1 \le K \le 10^{10}$)**. The next line of the test case will contain **N** space-separated integers **$t_1, t_2, \ldots t_n$ ($1 \le t_i \le 10^8$)**.

Output

The output should be in the format "`Case X: Y`" where X is the case number and Y is the number of ways. Please see the output in the sample case for more clarity.

Sample Input	Output for Sample Input
2 1 1 5 2 10 2 2	Case 1: 1 Case 2: 2

Problem I

Creating Triangles

Ayan is a small kid loves to play with stars "*" and dots "." and builds different triangles. Today he builds a triangular shape as follows and named it $\text{Triangle}_{\text{Level0}}$.

```
****
*.*
**
*
```

Figure 0: $\text{Triangle}_{\text{Level0}}$

By placing two $\text{Triangle}_{\text{Level0}}$(s) side by side and one $\text{Triangle}_{\text{Level0}}$ below them, he builds a new structure (in figure 1.1). Then gaps are filled with dots "." Then he names the structure as $\text{Triangle}_{\text{Level1}}$ (in figure 1.2).

```
********
*.* *.*
**  **
*   *
****
*.*
**
*
```

Figure 1.1:New Structure (Without dots)

```
********
*.*.*.*
**..**
*...*
****
*.*
**
*
```

Figure 1.2: $\text{Triangle}_{\text{Level1}}$

Similarly, the figure below shows the $\text{Triangle}_{\text{level2}}$.

```
****************
*.*.*.*.*.*.*.*
**..**..**..**
*...*...*...*
****....****
*.*.....*.*
**.......**
*.........*
********
*.*.*.*
**..**
*...*
****
*.*
**
*
```

Figure 2: $\text{Triangle}_{\text{Level2}}$

He repeated the same procedure until he built the $Triangle_{LevelN}$. Now he wants to know how many stars are in the K^{th} row after building $Triangle_{LevelN}$.

For example, if N = 2, then after building $Triangle_{Level2}$, 1^{st} row (counting of rows starts from the bottom) has 1 star, 2^{nd} and 3^{rd} row has 2 stars each, 4^{th} row has 4 stars, 5^{th} row has 2 stars, and so on.

Input

Input starts with an integer **T (1 ≤ T ≤ 100000)** denoting the number of test cases. Each of the test cases will contain two numbers **K (1 ≤ K ≤ 10^{16})** and **N (0 ≤ N ≤ 50)**.

Output

The output should be in the format "**`Case t: x`**" where t is the case number and x is the number of stars. If there is no such row in the triangle of level N, then print -1 instead.

Sample Input	Output for Sample Input
4 1 1 2 3 5 3 20 2	Case 1: 1 Case 2: 2 Case 3: 2 Case 4: -1

Problem J

Sort It

Alice has a permutation $p_1, p_2, \ldots, p_n$ of the integers **1, 2, …, n**. She performs **q** operations on it sequentially. Each operation is described by two integers **x** and **y**. If $p_x > p_y$, Alice swaps p_x and p_y.

Unfortunately Alice forgot what her initial permutation was, but she remembers that her permutation became sorted after performing all operations sequentially. Alice wants to recover her original permutation, but there could be many such permutations. Find the number of initial permutations such that after performing all operations it becomes sorted.

Input

The first line will contain a single integer **T**, the number of test cases.
Each test case starts with a single line containing two integers **n** and **q**. Each of the next **q** lines contain two integers **x** and **y**.

Output

For each case print a single integer in a new line - the number of permutations which become sorted after all operations.

Constraints

- **1 ≤ T ≤ 10**
- **2 ≤ n ≤ 20**
- **1 ≤ q ≤ 200**
- **Sum of q over all cases does not exceed 400**
- **1 ≤ x, y ≤ n and x ≠ y**

Sample Input	Output for Sample Input
3 3 2 1 2 2 3 2 1 2 1 4 4 3 2 1 3 2 4 3 4	4 0 10

For the first case, [1, 2, 3], [1, 3, 2], [2, 1, 3] and [2, 3, 1] are valid initial permutations.
For the second case, no initial permutation is valid.
For the third case, one valid permutation is [1, 3, 4, 2]. After the first operation it becomes [1, 4, 3, 2]. The second operation does not change it, After the third operation, it becomes [1, 2, 3, 4]. The last operation does not change it.