**Cost of balloons**

You are conducting a contest at your college. This contest consists of two problems and n participants. You know the problem that a candidate will solve during the contest.

You provide a balloon to a participant after he or she solves a problem. There are only green and purple-colored balloons available in a market. Each problem must have a balloon associated with it as a prize for solving that specific problem. You can distribute balloons to each participant by performing the following operation:

1. Use green-colored balloons for the first problem and purple-colored balloons for the second problem
2. Use purple-colored balloons for the first problem and green-colored balloons for the second problem

You are given the cost of each balloon and problems that each participant solve. Your task is to print the minimum price that you have to pay while purchasing balloons.

**Input format**

* First line: T that denotes the number of test cases (1≤T≤10)
* For each test case:
  + First line: Cost of green and purple-colored balloons
  + Second line: n that denotes the number of participants (1≤n≤10)
* Next n lines: Contain the status of users. For example, if the value of the jth integer in the ith row is 0, then it depicts that the ithparticipant has not solved the jth problem. Similarly, if the value of the jth integer in the ith row is 1, then it depicts that the ithparticipant has solved the jth problem.

**Output format**  
For each test case, print the minimum cost that you have to pay to purchase balloons.

**Sample Input**

2

9 6

10

1 1

1 1

0 1

0 0

0 1

0 0

0 1

0 1

1 1

0 0

1 9

10

0 1

0 0

0 0

0 1

1 0

0 1

0 1

0 0

0 1

0 0

**Question**

**2**

Max. Marks 100.00

**Median Game**

You are given an array A of N integers. You perform this operation N−2 times: For each contiguous subarray of **odd size** greater than 2, you find the median of each subarray(Say medians obtained in a move are M1,M2,M3,…,Mk). In each move, you remove the first occurrence of value min(M1,M2,M3,…,Mk) from the original array. After removing the element the array size reduces by 1 and no void spaces are left. For example, if we remove element 2 from the array {1,2,3,4}, the new array will be {1,3,4}.

Print a single integer denoting the sum of numbers that are left in the array after performing the operations. You need to do this for T test cases.

**Input Format**

The first line contains T denoting the number of test cases(1≤T≤10).

The first line of each test case contains N denoting the number of integers in the array initially(4≤N≤105).

The next line contains N space seperated integers denoting A1,A2,A3,…,AN(1≤Ai≤109 for all valid i).

**Output Format**

Output a single integer denoting the sum of numbers left in the array after performing the operations for each test case on a new line.

**Sample Input**

2

4

2 5 3 2

4

1 1 1 1

**Sample Output**

7

2

**Explanation**

For the first test case:  Initially, array is {2,5,3,2}. The medians obtained are {3,3} for subarrays [1,3] and [2,4] respectively. Hence, we remove min(3,3)=3 from the initial array. The array now becomes {2,5,2}. The median of the whole array is 2. Hence we remove the first occurrence of 2 from the array. So, we are left with {5,2} in our array.

 For the second test case, it is obvious that the minimum medium will be 1 every time. Hence finally, we will be left with {1,1} as the array.

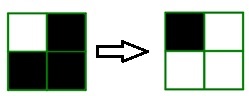
**Question**

**3**

Max. Marks 100.00

**Mosaics and holes**

John wants to cover his yard floor with mosaics. The yard floor is a n×m matrix and each cell is either a mosaic or a hole. John has invented a flipper machine. This machine has a size of k and can select a k×k square of the yard and flip the cells in it. By this action, every hole in the square becomes a mosaic and every mosaic in the square becomes a hole. It is illustrated by the following example:



The 2×2 square on the left has been flipped to the right one by the flipper with the size 2 and blacks are holes and whites are mosaics.

Help John to cover the floor of the yard completely by mosaics by using this machine. In each step, John selects a k×k square of the yard floor and sets the machine on it. He wants to compute the minimum steps needed to repair the yard floor.

**Input Format**

* First line: Three integers n,m,k (1≤n,m≤1000,1≤k≤min(n,m)), where n is the number of matrix rows, m as the number of matrix columns, and k as the size of the flipper machine.
* Next n lines: Each line will contain m integer 0 or 1 with space in between the consecutive one.
* This n lines are the data of the yard floor matrix, 1 indicates that the cell is mosaic and 0 indicates that the cell is a hole.

**Output format**

Print an integer denoting the minimum number of steps needed to repair the yard floor or -1 if it is impossible to repair the yard floor with this machine.

**Sample Input**

2 3 2

0 1 0

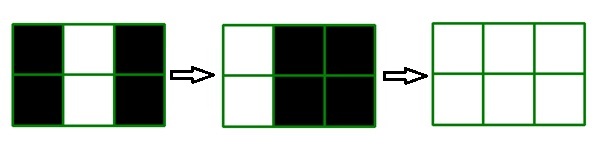
0 1 0

**Sample Output**

2

**Explanation**

we can repair the yard by 2 steps as shown below.



you can see more sample tests here!

|  |  |
| --- | --- |
| Sample Input | Sample output |
| 3 3 2  0 0 0  0 1 0  0 0 0 | -1 |
| 2 2 1  0 0  0 0 | 4 |

**Question**

**4**

Max. Marks 100.00

**Minimum cost**

You are standing at position 1. From position x, you can walk to x+1 or x−1 with cost 1. From position i, you can travel to without any cost (p is a permutation of numbers 1…n). You have to reach position n. Determine the minimum possible cost.

**Input format**

* First line: T denotes the number of test cases (1≤T≤10)
* For each test case:
  + First line: n 1≤T≤10
  + Second line: n integers where the ith integer is pi  
    **Note**: It is guaranteed that p is a permutation.

**Output format**  
Print the minimum possible cost.

**Sample Input**

6

6

1 4 3 2 5 6

6

3 2 6 5 4 1

6

3 6 2 4 1 5

6

5 4 6 3 2 1

6

1 6 5 4 2 3

6

5 1 3 6 2 4

**Sample Output**

3

0

0

0

1

1

**Question**

**5**

Max. Marks 100.00

**Flying square**

You are given a square. Let's say that the point of intersection of diagonals of the square is a square centre. Square flies to the Y-axis in the direction perpendicular to Y-axis with the speed S meters per second. Every point of the square rotates by alpha degrees per second clockwise around the centre. Your goal is to find the first moment of time when the square touches the line.

It is guaranteed that tests are numerically stable, i.e. if we change some of the input parameters by a very small value eps <10−9, then the |newAnswer−answer|<10−6.

**Input format**

Each line i of the 4 subsequent lines (where 1≤i≤4) contains two integers --- (xi,yi), (1≤xi,yi≤103) describing i-th vertex of the square in the clockwise or counter-clockwise order.   
Next line contains one integer S (1≤S≤103) --- speed of the square.  
Next line contains one integer alpha (1≤alpha<360) --- rotation speed of the square.

**Output format**

Print one decimal number --- time needed for the square to touch the Y-axis with exactly 5 digits after the decimal point

**Sample Input**

2 2

2 1

1 1

1 2

3 3

3 1

1

1

**Sample Output**

0.995703

**Question**

**6**

Max. Marks 100.00

**Car fuel**

You are given a weighted tree of N nodes that is rooted at node 1. The nodes represent the cities in your country. Also, the edges between the nodes u and v represent the fuel that will be utilized while traveling from node u to v or vice versa. Any cars consume the same fuel while moving from node u to v (that is equal to the edge weight between u and v). Cars can only move towards the root.

Your task is to answer Q queries of the following type:

In each query, you are provided with two values u and f, that denotes that you are currently at node u and your car has f amount of fuel in it initially. You will move from city p to city q only if q is the parent of p and the edge weight between p and q is **not more than the fuel left in the car**. While moving from node p to node q, the fuel in the car decrease by the amount equal to the edge weight between p and q. You'll stop whenever you either reach node 1 or cannot move any further up due to the restriction described above. Also,**you cannot pass through a node that has a car parked in it**. After you reach your destination, your car will be parked at that node and will affect the subsequent queries. For each query, you are required to print the node where the car is parked.

**Note**: Assume that no car is parked initially (before query 1).

**Input format**

* First line: Two space-separated integers N and Q denoting the number of nodes in the tree and the number of queries (1≤N,Q≤105)
* Next N−1 lines: Three space-separated integers u,v, and w representing a road from city u to city v with a weight of w (1≤u,v≤N, and 1≤w≤109)
* Each of the next Q lines: Two space-separated integers u and f denoting the node where you are standing currently and the amount of fuel that your car contain initially (1≤u≤N and 1≤f≤1015)

**Output format**

For each query, print the node where the car is parked on a new line.

**Sample Input**

9 6

1 2 3

1 3 4

1 4 5

2 5 10

2 6 8

5 7 12

5 8 1

7 9 5

2 2

5 50

4 4

6 10

7 13

8 500

**Sample Output**

2

2

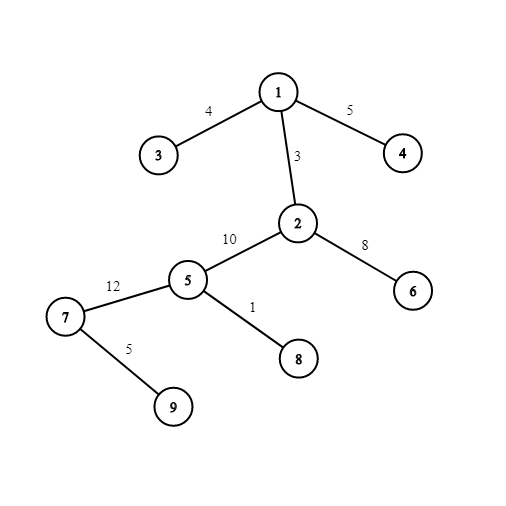
4

2

5

5

**Explanation**



The tree is shown in the figure. Initially, no cars are parked.

* In the first query, we are at node 2 and fuel in the car is 2 units initially. We cannot move to node 1 as the edge weight between node 1 and node 2 is 3(>2). Hence, the car gets parked at node 2.
* In the second query, we are at node 5 and fuel in the car is 50 units initially. We can move upto node 2 by utilizing 10 units of fuel. There is already a car parked at node 2 and thus, we cannot move any further up. Hence, this car also gets parked at node 2.
* In the third query, we start at node 4 with initial fuel of 4 units. As the edge weight is 5 units between node 1 and node 4, we cannot move up and hence, the car gets parked at node 4.
* In the fourth query, we can move up to node 2. A car is already parked at node 2 and hence, we cannot move any further up.
* In the fifth query, we can only move up to node 5 because we do not have enough fuel to move from node 5 to node 2. Thus, the car gets parked at node 5.
* In the final query, even if we have 500 units of fuel, we can only move up to node 5 because it has a car parked already!