ARRAYS

1. Rotate Array

Given an unsorted array **arr[ ]** of size **N**, rotate it by **D** elements (clockwise).

Input:

The first line of the input contains **T** denoting the number of testcases. First line of each test case contains two space separated elements, **N** denoting the size of the array and an integer **D** denoting the number size of the rotation. Subsequent line will be the N space separated array elements.

Output:

For each testcase, in a new line, output the rotated array.

Constraints:

1 <= T <= 200

1 <= N <= 10^7

1 <= D <= N

0 <= arr[i] <= 10^5

Example:

Input:

2

5 2

1 2 3 4 5

10 3

2 4 6 8 10 12 14 16 18 20

Output:

3 4 5 1 2

8 10 12 14 16 18 20 2 4 6

Explanation :

Testcase 1: 1 2 3 4 5 when rotated by 2 elements, it becomes 3 4 5 1 2.

2. Kth smallest element

Given an array arr [ ] and a number K where K is smaller than size of array, the task is to find the Kth smallest element in the given array. It is given that all array elements are distinct.

Expected Time Complexity: O(n)

Input:

The first line of input contains an integer T, denoting the number of testcases. Then T test cases follow. Each test case consists of three lines. First line of each testcase contains an integer N denoting size of the array. Second line contains N space separated integer denoting elements of the array. Third line of the test case contains an integer K.

Output:

Corresponding to each test case, print the kth smallest element in a new line.

Constraints:

1 <= T <= 100

1 <= N <= 105

1 <= arr[i] <= 105

1 <= K <= N

Example:

Input:

2

6

7 10 4 3 20 15

3

5
7 10 4 20 15
4
Output:
7
15
Explanation:
Testcase 1: 3rd smallest element in the given array is 7.

3. **Addition of submatrix**

Given a matrix C of size N x M. You are given position of submatrix as X1, Y1 and X2, Y2 inside the matrix. Find the sum of all elements inside that submatrix.
Input:
The first line of input contains an integer T denoting the number of test cases. The first line of each test case is n and m,n is the number of rows and m is the number of columns. The second line of each test case contains C[N][M]. The third line contains four value of X1, Y1, X2, Y2. X1, Y1 is the top left cell and X2, Y2 is the bottom right cell.
Output:
Print the sum of all elements inside that submatrix.
Constraints:
1 ≤ T ≤ 15
1 ≤ N, M ≤ 103
1 ≤ C[N][M] ≤ 106
1 <= X1, Y1, X2, Y2 <= M
Example:
Input:
2
5 6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
3 4 4 5
3 3
9 8 7 4 2 1 6 5 3
1 2 3 3
Output:
78
26
Explanation:
Testcase 2: Sum from cell starting at position (1, 2) (1-based indexing) and ending at (3, 3) is 26.

4. First negative integer in every window of size k

Given an array and a positive integer k, find the first negative integer for each and every window (contiguous subarray) of size k.
Input:
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains an integer n denoting the size of the array. The next line contains n space separated integers forming the array. The last line contains the window size k.

Output:
Print the space separated negative integer starting from the first till the end for every window size k. If a window does not contain a negative integer , then print 0 for that window.
Constraints:
1<=T<=10^5
1<=n<=10^5
1<=a[i]<=10^5
1<=k<=n
Example:
Input:
2
5
-8 2 3 -6 10
2
8
12 -1 -7 8 -15 30 16 28
3
Output:
-8 0 -6 -6
-1 -1 -7 -15 -15 0

5. Find Transition Point

You are given a sorted array containing only numbers 0 and 1. Find the transition point efficiently. Transition point is a point where "0" ends and "1" begins.
Input:
You have to complete the method which takes 2 argument: the array arr[] and size of array N. You should not read any input from stdin/console. There are multiple test cases. For each test cases, this method will be called individually.
Output:
Your function should return transition point.
Constraints:
1 ≤ T ≤ 100
1 ≤ N ≤ 500000
0 ≤ C[i] ≤ 1
Example:
Input
1
5
0 0 0 1 1
Output
3

6. Count Occurrences of Anagrams

Given a word S and a text C. Return the count of the occurrences of anagrams of the word in the text.

Input:

The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows. The first line of each test case contains a text S consisting of only lowercase characters.

The second line contains a word consisting of only lowercase characters.

Output:

Print the count of the occurrences of anagrams of the word C in the text S.

Constraints:

1 <= T <= 50

1 <= |S| <= |C| <= 50

Example:

Input:

2

forxxorfxdofr

for

aabaabaa

aaba

Output:

3

4

Explanation:

for, orf and ofr appears in the first test case and hence answer is 3.

LINKED LIST

7. Delete without head pointer

You are given a pointer/ reference to the node which is to be deleted from the linked list of **N** nodes. The task is to delete the node. Pointer/ reference to head node is not given.

Note: No head reference is given to you.

Input Format:

First line of input contains number of testcases T. For each testcase, first line of input contains length of linked list and next line contains the data of the linked list. The last line contains the node to be deleted.

Output Format:

For each testcase, in a newline, print the linked list after deleting the given node.

Your Task:

This is a **function** problem. You only need to complete the **function deleteNode** that takes **reference** to the node that needs to be **deleted**. The **printing** is done **automatically** by the **driver code**.

Constraints:

1 <= T <= 100

1 <= N <= 10^3

Example:

Input:

2

2

1 2
1
4
10 20 4 30
20
Output:
2
10 4 30
Explanation:
Testcase 1: After deleting 20 from the linked list, we have remaining nodes as 10, 4 and 30.

8. **Leaves to DLL**
Given a Binary Tree with **N** edges. The task is to extract all leaves of it in a **D**oubly **L**inked **L**ist (DLL). Note that the DLL need to be created in-place. Assume that the node structure of DLL and Binary Tree is same, only the meaning of left and right pointers are different. In DLL, left means previous pointer and right means next pointer. Head of DLL should point to the left most leaf and then in inorder traversal and so on.
Input:
First line of input contains number of testcases T. For each testcase, there will be two lines, first of which containing the number of edges (between two nodes) in the tree. Next line contains N pairs (considering **a** and **b**) with a '**L**' (means node b on left of a) or '**R**' (means node b on right of a) after a and b.
Output:
For each testcase, there will be two lines containing the nodes of DLL, first in reverse order and next in order of inorder traversal of tree.
User Task:
The task is to complete the function **convertToDLL**() which converts given binary tree to DLL.
Constraints:
1 <= T <= 100
1 <= N <= 10^3
Example:
Input:
2
2
1 2 L 1 3 R
3
1 2 L 1 3 R 2 4 L
Output:
2 3
3 2
4 3
3 4
Explanation:
Testcase 2: After extracting leaves, 3 and 4 from the tree, we have doubly linked list as 3 <-> 4.

9. **Count Pairs whose sum is equal to X**
Given two linked list of size **N1** and **N2** respectively of distinct elements, your task is to complete the function **countPairs()**, which returns the count of all pairs from both lists whose sum is equal to the given value **X**.
Input:
The function takes three arguments as input, reference pointer to the head of the two linked list

(**head1** and **head2**), and an variable **X**.
There will be **T** test cases and for each test case the function will be called separately.
Output:
For each test case the function should return the count of all the pairs from both lists whose sum is equal to the given value X.
Constraints:
1<=T<=100
1<=N1,N2<=1000
1<=X<=10000
Example:
Input:
2
6
1 2 3 4 5 6
3
11 12 13
15
4
7 5 1 3
4
3 5 2 8
10
Output:
3
2

10. **Binary Tree to CDLL**
Given a Binary Tree of **N** edges. The task is to convert this to a Circular Doubly Linked List(**CDLL**) in-place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted CDLL. The order of nodes in CDLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (left most node in BT) must be head node of the CDLL.
Input Format:
First line of input contains number of testcases T. For each testcase, there will be two lines, first of which containing the number of edges (between two nodes) in the tree. Next line contains N pairs (considering **a** and **b**) with a '**L**' (means node b on left of a) or '**R**' (means node b on right of a) after a and b
Output Format:
For each testcase, in a new line, print the traversals of CDLL.
Your Task:
You don't have to take input. Complete the function **bTreeToCList()** that takes root as parameter and **returns** the **head of the list**. The printing is done by the driver code.
Constraints:
1 <= T <= 100
1 <= N <= 10^3
1 <= Data of a node <= 10^4
Example:
Input:
2
2
1 2 R 1 3 L
4

10 20 L 10 30 R 20 40 L 20 60 R

Output:

3 1 2

2 1 3

40 20 60 10 30

30 10 60 20 40

Explanation:

Testcase 1: After converting tree to CDLL, when CDLL is traversed from head to tail and then tail to head, elements are displayed as in the output.

STACK

11. **Evaluation of Postfix Expression**

Given a postfix expression, the task is to evaluate the expression and print the final value. Operators will only include the basic arithmetic operators like ***,/,+ and -** .

Input:

The first line of input will contains an integer T denoting the no of test cases . Then T test cases follow. Each test case contains an postfix expression.

Output:

For each test case, in a new line, evaluate the postfix expression and print the value.

Constraints:

1 <= T <= 100

1 <= length of expression <= 100

Example:

Input:

2

231*+9-

123+*8-

Output:

-4

-3

12. **Sort a stack**

Given a stack, the task is to sort it such that the top of the stack has the greatest element.

Input:

The first line of input will contains an integer T denoting the no of test cases . Then T test cases follow. Each test case contains an integer N denoting the size of the stack. Then in the next line are N space separated values which are pushed to the the stack.

Output:

For each test case output will be the popped elements from the sorted stack.

Constraints:

1<=T<=100

1<=N<=100

Example(To be used only for expected output):

Input:

2

3

3 2 1

5

11 2 32 3 41

Output:

3 2 1
41 32 11 3 2
Explanation:
For first test case stack will be
1
2
3
After sorting
3
2
1
When elements popped : 3 2 1

13. **Special Stack**
Design a data-structure **SpecialStack** (using the STL of stack) that supports all the stack operations like **push()**, **pop()**, **isEmpty()**, **isFull()** and an additional operation **getMin()** which should return **minimum** element from the SpecialStack. Your task is to **complete all the functions**, using stack data-Structure.
Input Format:
The first line of input contains an integer **T** denoting the no of test cases. Then T test cases follow. Each test case contains two lines. The first line of input contains an integer **n** denoting the number of integers in a sequence. In the second line are **n** space separated integers of the stack.
Output Format:
For each testcase, in a new line, print the minimum integer from the stack.
Your Task:
Since this is a function problem, you don't need to take inputs. Just complete the provided functions.
Constraints:
1 <= T <= 100
1 <= N <= 100
Example:
Input:
1
5
18 19 29 15 16
Output:
15

14. **Preorder to Postorder**
Given an array arr[] of N nodes representing preorder traversal of BST. The task is to print its postorder traversal.
Input:
The first line of input contains an integer T denoting the number of test cases. The first line of each test case is N, the size of array. The second line of each test case contains N input as arr[i].
Output:
Postorder traversal of the given preorder traversal is printed.
Constraints:
1 <= T <= 100
1 <= N <= 10^3
1 <= arr[i] <= 10^4

Example:
Input:
3
5
40 30 35 80 100
8
40 30 32 35 80 90 100 120
Output:
35 30 100 80 40
35 32 30 120 100 90 80 40

QUEUE

15. **Min Heap implementation**

Given a **priority queue (max heap)** and **Q** queries to be performed on priority queue. The task is to perform operations based on queries.

1. **p :** query to **push** element (x, given with query) to priority_queue **and print size**.
2. **pp :** query to **pop** element from priority_queue **and print size**.
3. **t :** query to **return top** element of priority_queue, **if empty -1 will be printed**.

Input Format:
First line of input contains number of testcases **T**. For each testcase, first line of input contains Q queries. Next Q lines contains Q queries.
Output Format:
For each testcase, perform the required operation, and print if anything required.
Your Task:
Your task is to complete the functions **push_pq(), pp_pq()** and **pq_top(),** so that the queries are performed.
Constraints:
1 <= T <= 100
1 <= Q <= 100
Example:
Input:
1
5
p 5
p 3
p 1
t
pp
Output:
1
2
3
1
2
Explanation:
Testcase 1: Pushing 5, 3, 1 to queue. Now, fetching top which is 1 (according to min heap property smallest element at top). Popping back element from queue, size reduced to 2.

16. **Make Binary Tree From Linked List**

Given a Linked List Representation of Complete Binary Tree. The task is to construct the Binary tree.
Note : The complete binary tree is represented as a linked list in a way where if root node is stored at position i, its left, and right children are stored at position **2*i+1**, **2*i+2** respectively.

First negative integer in every window of size k

Given an array and a positive integer k, find the first negative integer for each and every window(contiguous subarray) of size k.

Input:

The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains an integer n denoting the size of the array. The next line contains n space separated integers forming the array. The last line contains the window size k.

Output:

Print the space separated negative integer starting from the first till the end for every window size k. If a window does not contain a negative integer , then print 0 for that window.

Constraints:

1<=T<=10^5
1<=n<=10^5
1<=a[i]<=10^5
1<=k<=n

Example:

Input:

2
5
-8 2 3 -6 10
2
8
12 -1 -7 8 -15 30 16 28
3

Output:

-8 0 -6 -6
-1 -1 -7 -15 -15 0

17. **Generate Binary Numbers**

Given a number **N**. The task is to generate and print all binary numbers with decimal values from **1 to N**.

Input:

The first line of input contains an integer T denoting the number of test cases. There will be a single line for each testcase which contains N.

Output:

Print all binary numbers with decimal values from 1 to N in a single line.

Constraints:

$1 \le T \le 10^6$
$1 \le N \le 10^6$

Example:

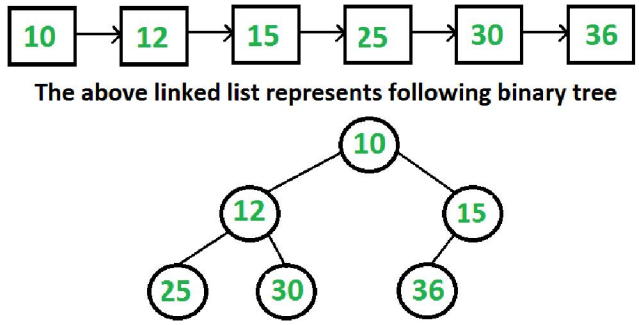
Input:

2
2
5

Output:

1 10
1 10 11 100 101

Explanation:
Testcase 1: Binary numbers from 1 to 2 are 1 and 10.



Input:
First line of input contains number of testcases T. For each testcase, first line of input contains length of linked list and next line contains linked list elements.
Output:
Output of each test case will be the level order traversal of the the constructed binary tree.
User Task:
The task is to complete the function **convert**() which takes head of linked list and **root** of the tree as the reference. The driver code prints the level order.
Constraints:
$1 <= T <= 100$
$1 <= N <= 10^3$
$1 <= K_i <= 10^3$
Example(To be used only for expected output) :
Input
2
5
1 2 3 4 5
5
5 4 3 2 1
Output:
1 2 3 4 5
5 4 3 2 1

18. **First negative integer in every window of size k**
Given an array and a positive integer k, find the first negative integer for each and every window (contiguous subarray) of size k.
Input:
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains an integer n denoting the size of the array. The next line contains n space separated integers forming the array. The last line contains the window size k.
Output:
Print the space separated negative integer starting from the first till the end for every window size k. If a window does not contain a negative integer , then print 0 for that window.
Constraints:
1<=T<=10^5
1<=n<=10^5
1<=a[i]<=10^5
1<=k<=n

Example:
Input:
2
5
-8 2 3 -6 10
2
8
12 -1 -7 8 -15 30 16 28
3
Output:
-8 0 -6 -6
-1 -1 -7 -15 -15 0

TREE

19. Clone a Binary Tree

Given a Binary Tree having random pointers clone the Binary Tree. The task is to complete the function **cloneTree** which take one argument the root of the tree to be copied and should return the root of the cloned tree.

Input:

The first line of input contains the no of test cases. Then T test cases follow. Each test case contains 2 lines the first line contains and integer N denoting the no of edges of the tree and then in the next line are N space separated queries .The query on tree are of three types

a) a b L (Represents that b is the left child of a)
b) a b R (Represents that b is the right child of a)
c) a b X (Represents a random pointer from node a to node b)

Output:

Your function should return the root of the cloned tree. The output will be 1 if the tree is successfully cloned.

Constraints:

1 <=T<= 30
1 <=Number of nodes<= 100
1 <=Data of a node<= 1000

Example(To be used only for expected output):

Input

1
6
6 3 L 6 8 R 3 1 L 3 5 R 1 3 X 5 6 X

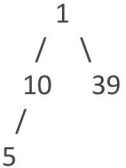
Output

1

Above test case represents the below tree with 6 edges .

20. Height of Binary Tree

Given a binary tree, find height of it.



The above tree has a height of 3.

Note: Height of **empty** tree is considered **0**.

Input Format:
The first line of input contains T denoting the number of testcases. T testcases follow. Each testcase contains two lines of input. The first line contains number of edges. The second line contains relation between nodes.
Output Format:
For each testcase, in a new line, print the height of tree.
Your Task:
You don't have to take input. Complete the function **height()** that takes **node** as parameter and **returns** the **height**. The **printing** is done by the **driver** code.
Constraints:
1 <= T <= 100
1 <= Number of nodes <= 100
1 <= Data of a node <= 1000
Example:
Input:
2
2
1 2 L 1 3 R
4
10 20 L 10 30 R 20 40 L 20 60 R
Output:
2
3
Explanation:
Testcase1: The tree is

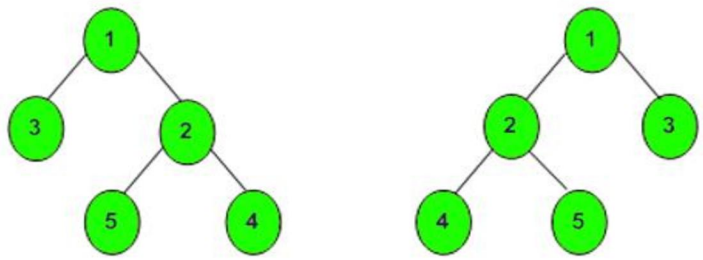
```
    1
  /   \
 3     2
```

So, the height would be 2.
Testcase2: The tree is

```
          10
        /    \
      20      30
    /   \
  40    60
```

So, height would be 3.

21. Mirror Tree
Given a Binary Tree, convert it into its mirror.



Mirror Trees

Input Format:
The first line of input contains T denoting the number of testcases. T testcases follow. Each testcase

contains two lines of input. The first line contains number of edges. The second line contains relation between nodes.

Output Format:

For each testcase, in a new line, print inorder traversal of mirror tree.

Your Task:

You don't have to take any input. Just complete the **function mirror()** that takes **node** as **paramter**. The printing is done by the driver code.

Constraints:

1 <= T <= 100

1 <= Number of nodes <= 100

1 <= Data of a node <= 1000

Example:

Input:

2

2

1 2 R 1 3 L

4

10 20 L 10 30 R 20 40 L 20 60 R

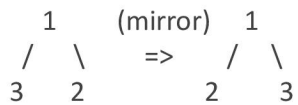
Output:

2 1 3

30 10 60 20 40

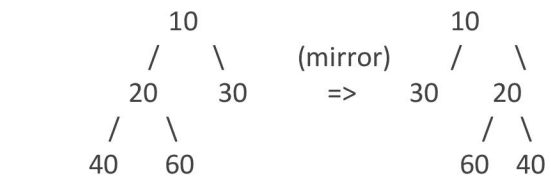
Explanation:

Testcase1: The tree is



The inorder of mirror is 2 1 3

Testcase2: The tree is



The inorder traversal of mirror is 30 10 60 20 40.

22. Construct Tree from Preorder Traversal

Given an array **pre[]** that represents Preorder traversal of a binary tree. One more array **preLN[]** is given which has only two possible values **'L'** and **'N'**. The value **'L'** in **preLN[]** indicates that the corresponding node in Binary Tree is a **leaf node** and value **'N'** indicates that the corresponding node is **non-leaf node**.

it is a special binary tree where every node has either 0 or 2 children

Input:

There will be **T**, test cases and for each test case the function will be called separately.

The function takes three arguments as input, first an integer **N**, denoting the size of both the array, second an array **pre[]** that represents Preorder traversal of the binary tree and the last argument a character array **preLN[]** which indicates that the corresponding node in Binary Tree is a leaf node or a normal node.

Output:

The output will be the inorder traversal of the resultant tree.

Constraints:

1 <=T <= 75

1 <= N <= 100
1 <= pre[i] <= 100
preLN[i]: {'N', 'L'}

User Task:

Your task is to complete the function **constructTree()**, that constructs the tree from the given two arrays and return the root pointer to new binary tree formed.

Example:

Input:

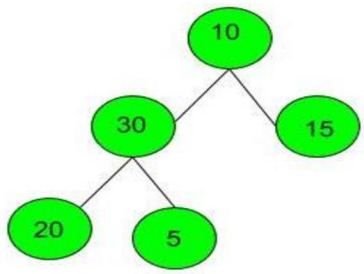
3
5
10 30 20 5 15
N N L L L
4
1 2 4 3
N N L L
6
1 2 4 6 5 3
N N N L L L

Output:

20 30 5 10 15
4 2 3 1
6 4 5 2 3 1

Explanation:

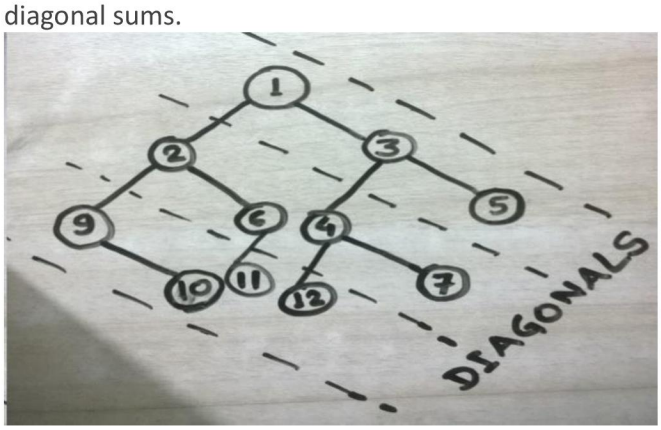
Testcase 1: Binary tree for the given pre array is:



The inorder traversal of given binary tree is: 20 30 5 10 15.

23. Diagonal Sum In Binary Tree

Consider lines of slope -1 passing between nodes (dotted lines in below diagram). The diagonal sum in a binary tree is the sum of all node's data lying between these lines. Given a Binary Tree, print all diagonal sums.



Note: The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

Input:

The first line consists of T test cases. The first line of every test case consists of N, denoting the number of edges in the tree. The second and third line of every test case consists of N, nodes of the binary tree.

Output:

Print space separated integers which are the diagonal sums for every diagonal present in the tree with slope -1.

Constraints:

1<=T<=100

1<=N<=100

Example:

Input:

2

3

4 1 L 4 3 R 3 3 L

5

10 8 L 10 2 R 8 3 L 8 5 R 2 2 L

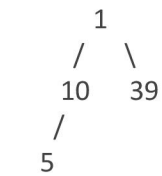
Output:

7 4

12 15 3

24. Inorder Traversal

Given a binary tree, the task is to print the nodes in inorder traversal of it. Inorder traversal of below tree is 5 10 1 39



Input Format:

First line of input contains number of testcases T. For each testcase, first line of input contains number of edges in the tree. For each edge there will be three consecutive inputs, first two integers contains parent node and child node, and the last character defines whether the child is in left or right.

Output Format:

For each testcase, in a new line, print the inorder traversal.

Your Task:

This is a function problem. You just need to complete the function **inorder()** that takes **node** as parameter. The newline is **automatically** appended by the **driver code**.

Constraints:

1 <= T <= 100

1 <= Number of nodes <= 100

1 <= Data of a node <= 1000

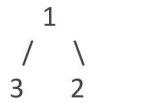
Example:

Input:

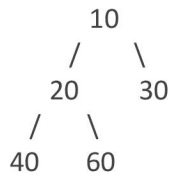
2

2

1 2 R 1 3 L
4
10 20 L 10 30 R 20 40 L 20 60 R
Output:
3 1 2
40 20 60 10 30
Explanation:
Testcase1: The tree is



So, the inorder would be 3 1 2
Testcase2: The tree is



So, the inorder would be 40 20 60 10 30

25. **Leaves to DLL**

Given a Binary Tree with **N** edges. The task is to extract all leaves of it in a **D**oubly **L**inked **L**ist (DLL). Note that the DLL need to be created in-place. Assume that the node structure of DLL and Binary Tree is same, only the meaning of left and right pointers are different. In DLL, left means previous pointer and right means next pointer. Head of DLL should point to the left most leaf and then in inorder traversal and so on.
Input:
First line of input contains number of testcases T. For each testcase, there will be two lines, first of which containing the number of edges (between two nodes) in the tree. Next line contains N pairs (considering **a** and **b**) with a '**L**' (means node b on left of a) or '**R**' (means node b on right of a) after a and b.
Output:
For each testcase, there will be two lines containing the nodes of DLL, first in reverse order and next in order of inorder traversal of tree.
User Task:
The task is to complete the function **convertToDLL**() which converts given binary tree to DLL.
Constraints:
1 <= T <= 100
1 <= N <= 10^3
Example:
Input:
2
2
1 2 L 1 3 R
3
1 2 L 1 3 R 2 4 L
Output:
2 3
3 2
4 3
3 4

Explanation:
Testcase 2: After extracting leaves, 3 and 4 from the tree, we have doubly linked list as 3 <-> 4.

26. **Binary Tree to CDLL**

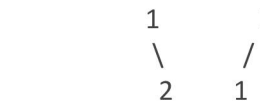
Given a Binary Tree of **N** edges. The task is to convert this to a Circular Doubly Linked List(**CDLL**) in-place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted CDLL. The order of nodes in CDLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (left most node in BT) must be head node of the CDLL.
Input Format:
First line of input contains number of testcases T. For each testcase, there will be two lines, first of which containing the number of edges (between two nodes) in the tree. Next line contains N pairs (considering **a** and **b**) with a '**L**' (means node b on left of a) or '**R**' (means node b on right of a) after a and b
Output Format:
For each testcase, in a new line, print the traversals of CDLL.
Your Task:
You don't have to take input. Complete the function **bTreeToCList()** that takes root as parameter and **returns** the **head of the list**. The printing is done by the driver code.
Constraints:
1 <= T <= 100
1 <= N <= 10^3
1 <= Data of a node <= 10^4
Example:
Input:
2
2
1 2 R 1 3 L
4
10 20 L 10 30 R 20 40 L 20 60 R
Output:
3 1 2
2 1 3
40 20 60 10 30
30 10 60 20 40
Explanation:
Testcase 1: After converting tree to CDLL, when CDLL is traversed from head to tail and then tail to head, elements are displayed as in the output.

BST

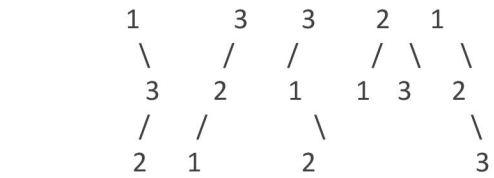
27. **Unique BST's**

Given an integer **N**, how many **structurally unique binary search trees** are there that store values 1...N?
Input:
First line of input contains **T** denoting the number of testcases. T testcases follow. Each testcase contains a single line of input containing N.
Ouput:
For each testcase, in a new line, print the answer.
Constraints:
1<=T<=15
1<=N<=11

Example:
Input:
2
2
3
Output:
2
5
Explanation:
Testcase1:
for N = 2, there are 2 unique BSTs



Testcase2:
for N = 3, there are 5 possible BSTs



28. **Add all greater values to every node in a BST**
Given a **B**inary **S**earch **T**ree (BST), modify it so that all greater values in the given BST are added to every node.
In this function problem, the task is to complete the function **modify** which takes one argument: Address of the root of the BST. The function should contain the logic to modify the BST so that in the modified BST, every node has a value equal to the sum of its value in the original BST and values of all the elements larger than it in the original BST.
The BST node structure has 3 fields, a data part which stores the data, a left pointer which points to the left element of the BST and a right pointer which points to the right node of the BST.
There are multiple test cases. For each test case, this function will be called individually.
Input:
The first line of the input contains an integer 'T' denoting the number of test cases. Then 'T' test cases follow. Each test case consists of two lines. First line of each test case contains an integer N denoting the no of nodes of the BST . Second line of each test case consists of 'N' space separated integers denoting the elements of the BST. These elements are inserted into BST in the given order.
Output:
Print the inorder traversal of the modified BST.
Expected Time Complexity: O(N)
Constraints:
1<=T<=100
1<=N<=200
Example:
Input:
1
7
50 30 20 40 70 60 80 (Elements are inserted into BST in this order only)
Output:
350 330 300 260 210 150 80 (Inorder Traversal of the modified BST)

Explanation:

```
        50
      /    \
     30      70
    /  \    / \
   20   40  60  80
```

The above tree should be modified to following

```
        260
      /    \
     330     150
    /  \    / \
   350  300  210  80
```

29. **Binary Tree to BST**

Given a binary tree, your task is to complete the function **binaryTreeToBST** which converts the binary tree to binary search tree(BST) and returns the root of the converted binary search tree.

Note : The conversion must be done in such a way that keeps the original structure of Binary Tree.

```
      1
    /   \
   2     3
  /
4
```

For the tree above the converted tree will be

```
     3
   /   \
  2     4
 /
1
```

Input:

The task is to complete the method which takes one argument, root of Binary Tree and returns the root of the new converted binary search tree.There are multiple test cases. For each test case, this method will be called individually.

Output:

The output will be the inorder traversal of the the converted binary search tree .

Constraints:

1 <=T<= 30

1 <= Number of nodes<= 20

Example(To be used only for expected output):

Input:

2

2

1 2 R 1 3 L

4

10 20 L 10 30 R 20 40 L 20 60 R

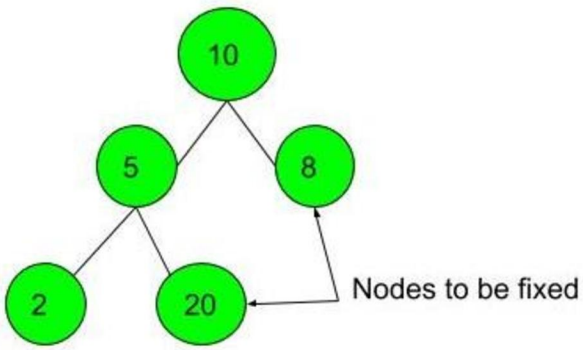
Output:

1 2 3

10 20 30 40 60

There are two test cases.
First case represents a tree with 3 nodes and 2 edges where root is 1, left child of 1 is 3 and right child of 1 is 2.
Second test case represents a tree with 4 edges and 5 nodes.

30. **Fixing Two nodes of a BST**
Two of the nodes of a Binary Search Tree (BST) are swapped. Fix (or correct) the BST.
Input Format:
First line consists of T test cases. First line of every test case consists of N, denoting number of elements in BST. Second line of every test case consists 3*N elements 2 integers and a character
Note: It is guaranteed than the given input will form BST ,except for 2 nodes that will be wrong.
Output Format:
For each testcase, in a new line, print either 0 or 1.
Your Task:
You don't need to take any input. Just complete the function **correctBst()** that takes node as **parameter**. The corrected BST will then be checked internally.
Constraints:
1 <= T <= 100
1 <= N <= 100
Example:
Input:
2
4
10 5 L 10 8 R 5 2 L 5 20 R
5
8 3 L 8 10 R 3 1 L 3 6 R 6 7 R
Output:
1
0
Explanation:
Testcase 1:



31. **Find the Closest Element in BST**
Given a **BST** with **N** nodes and a target node **K**. The task is to find an integer having minimum **absolute difference** with given target value K and return this difference.
Input Format:
The first line of the input contains an integer 'T' denoting the number of test cases. Then 'T' test cases follow. Each test case consists of three lines. First line of each test case contains an integer N denoting the number of nodes in the BST . Second line of each test case consists of 'N' space

separated integers denoting the elements of the BST. These elements are inserted into BST in the given order.The last line of each test case contains an integer k as specified in problem statement.

Output Format:

The output for each test case will be the minimum absolute difference of a node with given target value K.

Your Task:

This is a function problem. You don't have to take any input. Just complete the **function maxDiff()** that takes **node and K** as parameter and returns the minimum difference.

Constraints:

1 <= T <= 100

1 <= N <= 200

Example:

Input:

2

9

9 4 3 6 5 7 17 22 20

18

9

9 4 3 6 5 7 17 22 20

4

Output:

1

0

32. **Lowest Common Ancestor in a BST**

Given a Binary Search Tree and 2 nodes value **n1** and **n2**, your task is to find the **lowest common ancestor(LCA)** of the two nodes .

Note: Duplicates are not inserted in the BST.

Input Format:

The first line of input contains T denoting the number of testcases. T testcases follow. Each testcase contains three lines of input.

The first line contains **N**, the number of nodes of the BST.

The second line contains the **values of the nodes**, each separated by a space.

The third line contains the two values **n1 and n2** separated by a space.

Output Format:

For each testcase, in a new line, print the LCA of n1 and n2.

Your Task:

This is a function problem. You don't have to take any input. You are required to complete the function **LCA()** that takes node, n1, n2 as parameters and returns the node that is LCA of n1 and n2.

Constraints:

1 <= T <= 200

1 <= N <= 100

1 <= Node value <= 1000

Example:

Input

2

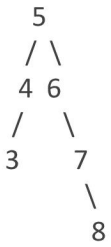
6

5 4 6 3 7 8

7 8

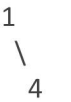
6

12 14 15 10 9 8
8 15
Output
7
12
Explanation
The BST in above test case will look like

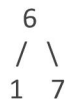


here the LCA of 7 and 8 is 7.

33. **Merge two BST 's**
Given two BST, you need to print the elements of both BSTs in **sorted** form.
Input Format:
The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. Each test case contains three lines. The first line of each test case contain's an integer N and M denoting the size of the two BST's. Then In the next two line are space separated values of the two BST's.
Output Format:
For each testcase, in a new line, print the elements of both BSTs in sorted form.
Your Task:
This is a function problem, you don't have to take any input. Just complete the function **merge() that** nodes of both **BSTs as parameters.**
Constraints:
1 <= T <= 100
1 <= N,M <= 100
Example:
Input:
2
3 3
1 33 4
6 7 1
2 2
1 6
9 2
Output:
1 1 4 6 7 33
1 2 6 9
Explanation:
Testcase1: The BSTs look like
BST1:



BST2:



The sorted elements of above BSTs are 1 1 4 6 7 33

34. Print BST elements in given range

Given a Binary Search Tree **(BST)** and a range, print all the numbers in the BST that lie in the given range **l-h(inclusive)** in **non-decreasing order.** If no such element exists**, an empty line will be printed**.

Note: Element greater than or equal to root go to the right side.

Input Format:

The first line of the input contains an integer **'T'** denoting the nummber of test cases. Then **'T'** test cases follow. Each test case consists of **three** lines. Description of test cases is as follows:

The First line of each test case contains an integer **'N'** which denotes the number of nodes in the BST. .

The Second line of each test case contains **'N' space separated values** of the nodes in the BST.

The Third line of each test case contains **two space separated integers 'l' and 'h'** denoting the **range**.

Output Format:

For each testcase, in a new line, print print all the numbers in the BST that lie in the given range in **non-decreasing order.**

Your Task:

This is a function problem. You don't have to take any input. Complete the function **printNearNodes()** that takes root, l, h as parameters and prints numbers in the BST that lie in the given range in **non-decreasing order.**

Constraints:

1 <= T <= 100

1 <= N <= 50

1 <= l < h < 1000

Example:

Input:

2

6

10 5 50 1 40 100

5 45

4

5 6 3 2

1 4

Output:

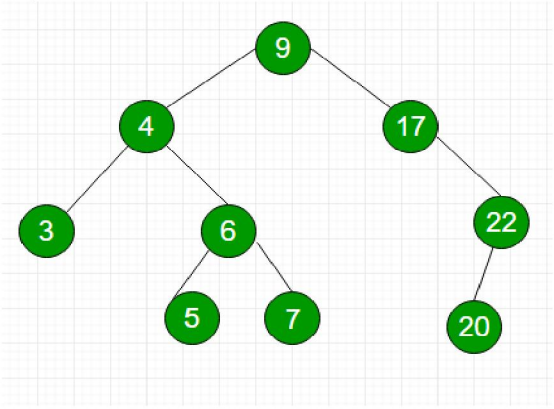
5 10 40

2 3

Explanation:

Testcase 1: Elements which are in the range of 5 and 45(inclusive) are 5 10 40.

Explanation:



Testcase1:
K=18. The node that has value nearest to K is 17 so it's difference with k is the answer.
Testcase2:
K=4. The node that has value nearest to K is 4 so it's difference with k is the answer.

HEAP

35. **Merge k Sorted Arrays**

Given **K** sorted arrays arranged in form of a matrix. The task is to merge them. You need to complete **mergeKArrays**() function which takes 2 arguments, an arr[k][k] 2D Matrix containing k sorted arrays and an integer k denoting the number of sorted arrays. The function should return a pointer to the merged sorted arrays.
Input:
The first line of input contains the number of test cases, then T test cases follows. Each test case will contain an integer N denoting the number of sorted arrays. Then in the next line contains all the elements of the array separated by space.
Output:
The output will be the sorted merged array.
User Task:
The task is to complete the function **mergeKArrays**() which takes two arguments, and returns pointer to the modified array.
Constraints:
1 <= T <= 50
1 <= N <= 10^3
1 <= K <= 10
Example:
Input:
1
3
1 2 3 4 5 6 7 8 9
Output:
1 2 3 4 5 6 7 8 9
Explanation:
Testcase 1:
Above test case has 3 sorted arrays of size 3, 3, 3
arr[][] = [[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]
The merged list will be [1, 2, 3, 4, 5, 6, 7, 8, 9].

36. **Rearrange characters**

Given a string S with repeated characters (only lowercase). The task is to rearrange characters in a string such that no two adjacent characters are same.

Note : It may be assumed that the string has only lowercase English alphabets.

Input:

The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains a single line containing a string of lowercase english alphabets.

Output:

For each test case in a new line print "1" (without quotes) if the generated string doesn't contains any same adjacent characters, else if no such string is possible to be made print "0" (without quotes).

Constraints:

1 <= T <= 100

1 <= length of string <= 10^4

Example:

Input:

3

geeksforgeeks

bbbabaaacd

bbbbb

Output:

1

1

0

Explanation:

Testcase 1: All the repeated characters of the given string can be rearranged so that no adjacent characters in the string is equal.

Testcase 3: Repeated characters in the string cannot be rearranged such that there should not be any adjacent repeated character.

37. **Heap Sort**

Given an array of size N. The task is to sort the array elements by completing functions **heapify**() and **buildHeap**() which are used to implement Heap Sort.

Input:

First line of the input denotes number of test cases T. First line of the test case is the size of array and second line consists of array elements.

Output:

Sorted array in **ascending** order.

Constraints:

1 <= T <= 100

1 <= N <= 10^6

1 <= arr[i] <= 10^6

Example:

Input:

2

5

4 1 3 9 7

10

10 9 8 7 6 5 4 3 2 1

Output:
1 3 4 7 9
1 2 3 4 5 6 7 8 9 10
Explanation:
Testcase 1: After sorting elements using heap sort, elements will be in order as 1, 3, 4, 7 and 9.

38. **Minimum Cost of ropes**
There are given **N** ropes of different lengths, we need to connect these ropes into one rope. The cost to connect two ropes is equal to sum of their lengths. The task is to connect the ropes with minimum cost.
Input:
The first line of input contains an integer T denoting the number of test cases. The first line of each test case is N where N is the number of ropes. The second line of each test case contains N input L[i],length of ropes.
Output:
For each testcase, print the minimum cost to connect all the ropes.
Constraints:
$1 \le T \le 100$
$1 \le N \le 10^6$
$1 \le L[i] \le 10^6$
Example:
Input:
2
4
4 3 2 6
5
4 2 7 6 9
Output:
29
62
Explanation:
For example if we are given 4 ropes of lengths 4, 3, 2 and 6. We can connect the ropes in following ways.
1) First connect ropes of lengths 2 and 3. Now we have three ropes of lengths 4, 6 and 5.
2) Now connect ropes of lengths 4 and 5. Now we have two ropes of lengths 6 and 9.
3) Finally connect the two ropes and all ropes have connected.
Total cost for connecting all ropes is 5 + 9 + 15 = 29. This is the optimized cost for connecting ropes. Other ways of connecting ropes would always have same or more cost. For example, if we connect 4 and 6 first (we get three strings of 3, 2 and 10), then connect 10 and 3 (we get two strings of 13 and 2). Finally we connect 13 and 2. Total cost in this way is 10 + 13 + 15 = 38.

GRAPH

39. **X Total Shapes**
Given **N** * **M** string array of **O's** and **X's**. The task is to find the number of '**X**' total shapes. '**X**' shape consists of one or more adjacent **X**'s (diagonals not included).
Input: The first line of input takes the number of test cases T. Then T test cases follow. Each of the T test cases takes 2 input lines. The first line of each test case have two integers N and M.The second line of N space separated strings follow which indicate the elements in the array.
Output:
For each testcase, print number of shapes.

Constraints:
1 <= T <= 100
1 <= N, M <= 50
Example:
Input:
2
4 7
OOOOXXO OXOXOOX XXXXOXO OXXXOOO
10 3
XXO OOX OXO OOO XOX XOX OXO XXO XXX OOO
Output:
4
6
Explanation:
Testcase 1: Given input is like:
OOOO**XX**O
O**X**O**X**OO**X**
XXXXO**X**O
O**XXX**OOO
So, X with same colour are adjacent to each other vertically for horizontally (diagonals not included). So, there are 4 different groups in the given matrix.
Testcase 2: Given input is like:
XXO
OOX
OXO
OOO
XOX
XOX
O**X**O
XXO
XXX
OOO
So, this matrix has 6 groups with is having adjacent Xs. Total number of groups is 6.

40. **Unit Area of largest region of 1's**
Consider a matrix with **N** rows and **M** columns, where each cell contains either a **'0'** or a **'1'** and any cell containing a 1 is called a filled cell. Two cells are said to be connected if they are adjacent to each other horizontally, vertically, or diagonally. If one or more filled cells are connected, they form a region. The task is to find the unit area of the largest region.
Input:
The first line of input will be the number of testcases **T**, then T testcases follow. The first line of each testcase contains 2 space separated integers n and m. Then in the next line are the **n x m** inputs of the matrix A separated by space.
Output:
The output in the expected output will be the length of the largest region formed.
Constraints:
1 <= T <= 100
1 <= N, M <= 50
0 <= A[][] <= 1
Example:
Input:

2
3 3
1 1 0 0 0 1 1 0 1
1 3
1 1 1
Output:
4
3
Explanation:
Testcase 1: Matrix can be shown as follows:
1 1 0
0 0 1
1 0 1
Largest region of 1s in the above matrix is with total 6 1s (colored in Red).

41. **DFS of Graph**
Given **N** nodes and **E** edges of a graph. The task is to do depth first traversal of the graph.
Note: Use recursive approach.
Input:
First line of input contains number of testcases T. For each testcase. First line of each testcase contains number of nodes and edges seperated by space and next line contains N pairs of integers (X and Y each) where X Y means an edge from X to Y.
Output:
For each testcase, print the nodes while doing DFS starting from node 0.
Your task:
The task is to complete the function **dfs**() which should do the depth first traversal of given graph and prints the node in DFS order.
Constraints:
1 <= T <= 100
1 <= N <= 200
Example:
Input:
2
5 4
0 1 0 2 0 3 2 4
4 3
0 1 1 2 0 3
Output:
0 1 2 4 3 // dfs from node 0
0 1 2 3
Explanation:
Testcase 1: There is one test cases. First line of each test case represent an integer N denoting number of edges and then in the next line N pairs of values **a** and **b** are fed which denotes there is an edge from **a** to **b**.

42. **Find whether path exist**
Given a **N** X **N** matrix (**M**) filled with 1, 0, 2, 3. The task is to find whether there is a path possible from source to destination, while traversing through blank cells only. You can traverse up, down, right and left.
    - A value of cell **1** means Source.
    - A value of cell **2** means Destination.

- A value of cell **3** means Blank cell.
- A value of cell **0** means Blank Wall.

Note: there is only single source and single destination.

Input:

The first line of input is an integer **T** denoting the no of testcases. Then T test cases follow. Each test case consists of 2 lines. The first line of each test case contains an integer N denoting the size of the square matrix. Then in the next line are N*N space separated values of the matrix (M).

Output:

For each test case in a new line print 1 if the path exist from source to destination else print 0.

Constraints:

1 <= T <= 20

1 <= N <= 20

Example:

Input:

2

4

3 0 0 0 0 3 3 0 0 1 0 3 0 2 3 3

3

0 3 2 3 0 0 1 0 0

Output:

1

0

Explanation:

Testcase 1: The matrix for the above given input is:

3 0 0 0

0 3 3 0

0 1 0 3

0 2 3 3

From the matrix we can see that there exists a path from to reach destination 2 from source 1.

Testcase 2: The matrix for the above given input is:

0 3 2

3 0 0

1 0 0

From the matrix we can see that there does not exists any path to reach destination 2 from source 1.

43. Unit Area of largest region of 1's

Consider a matrix with **N** rows and **M** columns, where each cell contains either a **'0'** or a **'1'** and any cell containing a 1 is called a filled cell. Two cells are said to be connected if they are adjacent to each other horizontally, vertically, or diagonally. If one or more filled cells are connected, they form a region. The task is to find the unit area of the largest region.

Input:

The first line of input will be the number of testcases **T**, then T testcases follow. The first line of each testcase contains 2 space separated integers n and m. Then in the next line are the **n x m** inputs of the matrix A separated by space.

Output:

The output in the expected output will be the length of the largest region formed.

Constraints:

1 <= T <= 100

1 <= N, M <= 50

0 <= A[][] <= 1

Example:
Input:
2
3 3
1 1 0 0 0 1 1 0 1
1 3
1 1 1
Output:
4
3
Explanation:
Testcase 1: Matrix can be shown as follows:
1 1 0
0 0 1
1 0 1
Largest region of 1s in the above matrix is with total 6 1s (colored in Red).

44. **Rotten Oranges**

Given a matrix of dimension **r*c** where each cell in the matrix can have values 0, 1 or 2 which has the following meaning:
0 : Empty cell
1 : Cells have fresh oranges
2 : Cells have rotten oranges
So, we have to determine what is the minimum time required to rot all oranges. A rotten orange at index [i,j] can **r**ot other fresh orange at indexes [i-1,j], [i+1,j], [i,j-1], [i,j+1] (**up**, **down**, **left** and **right**) in unit time. If it is impossible to rot every orange then simply return -1.
Input:
The first line of input contains an integer T denoting the number of test cases. Each test case contains two integers r and c, where r is the number of rows and c is the number of columns in the array a[]. Next line contains space separated r*c elements each in the array a[].
Output:
Print an integer which denotes the minimum time taken to rot all the oranges (-1 if it is impossible).
Constraints:
1 <= T <= 100
1 <= r <= 100
1 <= c <= 100
0 <= a[i] <= 2
Example:
Input:
2
3 5
2 1 0 2 1 1 0 1 2 1 1 0 0 2 1
3 5
2 1 0 2 1 0 0 1 2 1 1 0 0 2 1

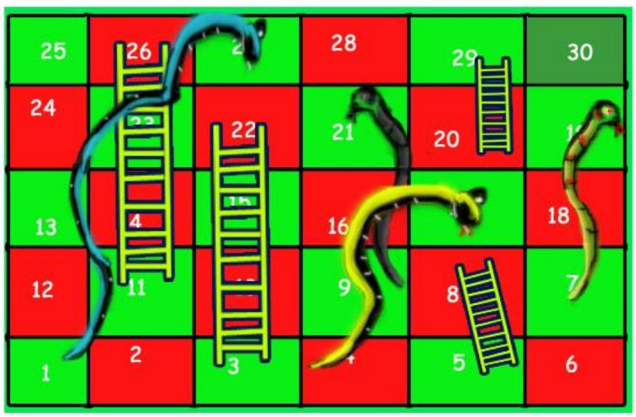
Output:
2
-1
Explanation:
Testcase 1:
2 | 1 | 0 | 2 | 1

1 | 0 | 1 | 2 | 1
1 | 0 | 0 | 2 | 1
Oranges at positions {0,0}, {0, 3}, {1, 3} and {2, 3} will rot oranges at {0, 1}, {1, 0}, {0, 4}, {1, 2}, {1, 4}, {2, 4} during 1st unit time. And, during 2nd unit time, orange at {1, 0} got rotten and will rot orange at {2, 0}. Hence, total 2 unit of time is required to rot all oranges.

45. Snake and Ladder Problem

Given a snake and ladder board of order 5x6, find the minimum number of dice throws required to reach the destination or last cell (30th cell) from source (1st cell) .

Example



For the above board output will be 3
For 1st throw get a 2
For 2nd throw get a 6
For 3rd throw get a 2

Input:

The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. Each test case contains two lines. The first line of input contains an integer N denoting the no of ladders and snakes present. Then in the next line are 2*N space separated values a,b which denotes a ladder or a snake at position **'a'** which takes to a position **'b'**.

Output:

For each test case in a new line print the required answer denoting the min no of dice throws.

Constraints:

1 <= T <= 100
1 <= N <= 10
1 <= a <= 30
1 <= b <= 30

Example:

Input:

2
6
11 26 3 22 5 8 20 29 27 1 21 9
1
2 30

Output:

3
1

Explanation:

Testcase 1:

For 1st throw get a 2, which contains ladder to reach 22

For 2nd throw get a 6, which will lead to 28
Finally get a 2, to reach at the end 30. Thus 3 dice throws required to reach 30.

46. **Steps by Knight**

Given a square chessboard of N x N size, the position of Knight and position of a target is given. We need to find out minimum steps a Knight will take to reach the target position.

Input:

The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains an integer n denoting the size of the square chessboard. The next line contains the X-Y coordinates of the knight. The next line contains the X-Y coordinates of the target.

Output:

Print the minimum steps the Knight will take to reach the target position.

Constraints:

1<=T<=100
1<=N<=20
1<=knight_pos,targer_pos<=N

Example:

Input:

2
6
4 5
1 1
20
5 7
15 20

Output:

3
9

HASHING

47. **Match specific pattern**

Given a dictionary of words and a pattern. Every character in the pattern is uniquely mapped to a character in the dictionary. The task is to complete a function **findMatchedWords(dict,pattern)** that returns a vector of strings matching with given pattern. The function takes two argument the first argument is an array of strings **dict** which denote the dictionary and the second argument is the **pattern** to match.

Input:

The first line of input contains an integer **T** denoting the number of test cases. Each testcase contains 3 lines. The first line contains an integer **N** denoting the number of strings in the dictionary,second line N space separated strings denoting the strings of the dictionary. The third line contains pattern.

Output:

The output for each test case will be the space separated strings that matches the given pattern in lexicographical order.

User Task:

Since this is a functional problem you don't have to worry about input, you just have to complete the function **findMatchedWords()**

Constraints:

1 <= T <= 100
1 <= N <= 10

Input

1

4
abb abc xyz xyy
foo
Output
abb xyy
Explanation
Testcase 1: In the above test case xyy and abb have same character at index 1 and 2 like the pattern.

48. Union of Two Linked Lists

Given two linked lists, your task is to complete the function **makeUnion(),** that returns the union of two linked lists. This union should include all the distinct elements only.
Input:
The function takes 2 arguments, reference pointer to the head of the first linked list (**head1**) and reference pointer to the head of the second linked list (**head2**).
There are multiple test cases and for each test case this function will be called separately.
Output:
The function should return reference pointer to the head of the new list that is formed by the union of the two the lists.
Note: The new list formed should be in non-decreasing order.
User Task:
The task is to complete the function **makeUnion**() which makes the union of the given two lists.
Constraints:
1<=T<=100
$1<=N<=10^3$
Example:
Input:
1
6
9 6 4 2 3 8
5
1 2 8 6 2
Output:
1 2 3 4 6 8 9
Explanation:
Testcase 1: Union of the given two lists have elements 1, 2, 3, 4, 6, 8 and 9 in the list.

49. Count pairs with given sum

Given an array of integers, and an integer ‘K’ , find the count of pairs of elements in the array whose sum is equal to 'K'.
Input:
First line of the input contains an integer T, denoting the number of test cases. Then T test cases follow. Each test case consists of two lines. First line of each test case contains 2 space separated integers N and K denoting the size of array and the sum respectively. Second line of each test case contains N space separated integers denoting the elements of the array.
Output:
Print the count of pairs of elements in the array whose sum is equal to the K.
Constraints:
1<=T<=50
1<=N<=50
1<=K<=50
1<=A[i]<=100

Example:
Input
2
4 6
1 5 7 1
4 2
1 1 1 1
Output
2
6

50. Nuts and Bolts Problem

Given a set of **N** nuts of different sizes and N bolts of different sizes. There is a one-one mapping between nuts and bolts. Match nuts and bolts efficiently.
Comparison of a nut to another nut or a bolt to another bolt is not allowed. It means nut can only be compared with bolt and bolt can only be compared with nut to see which one is bigger/smaller.
Input:
The first line contains '**T**' denoting the number of testcases. Then follows description of T testcases: Each case begins with a single positive integer N denoting the number of nuts/bolts. Then follows the array of nuts, each element separated by a space. And finally the bolts array, again, each element is separated by a space here. Array of Nuts/Bolts can only consist of the following elements:{'@', '#', '$', '%', '^', '&', '~', '*', '!'}. And no element can be repeated.
Output:
For each test case, output the matched array of nuts and bolts in separate lines, where each element in the array is separated by a space. Print the elements in the following order ! # $ % & * @ ^ ~
Constraints:
1 <= T <= 70
1 <= N <= 9
Example:
Input:
2
5
@ % $ # ^
% @ # $ ^
9
^ & % @ # * $ ~ !
~ # @ % & * $ ^ !
Output:
$ % @ ^
$ % @ ^
! # $ % & * @ ^ ~
! # $ % & * @ ^ ~

MATRIX

51. Sorted matrix

Given an n x n matrix, where every row and column is sorted in non-decreasing order. Print all elements of matrix in sorted order.
Input:
The first line of input contains an integer T denoting the number of test cases. Then T test cases

follow. Each test case contains an integer n denoting the size of the matrix. Then the next line contains the n x n elements in row wise order.

Output:

Print the elements of the matrix in sorted order.

Constraints:

1<=T<=100

1<=n<=100

1<=a[n][n]<=100

Example:

Input:

2

4

10 20 30 40 15 25 35 45 27 29 37 48 32 33 39 50

3

1 3 4 2 6 7 5 8 9

Output:

10 15 20 25 27 29 30 32 33 35 37 39 40 45 48 50

1 2 3 4 5 6 7 8 9

52. Is Sudoku Valid

Given an incomplete Sudoku configuration in terms of a 9x9 2-D square matrix (mat[][]) the task to check if the configuration has a solution or not.

Input:

The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. Each test case contains 9*9 space separated values of the matrix mat[][] representing an incomplete Sudoku state where a 0 represents empty block.

Output:

For each test case in a new line print 1 if the sudoku configuration is valid else print 0.

Constraints:

1<=T<=10

0<=mat[]<=9

Example:

Input:

2

3 0 6 5 0 8 4 0 0 5 2 0 0 0 0 0 0 0 0 8 7 0 0 0 0 3 1 0 0 3 0 1 0 0 8 0 9 0 0 8 6 3 0 0 5 0 5 0 0 9 0 6 0 0 1 3 0 0 0 0 2 5 0 0 0 0 0 0 0 0 7 4 0 0 5 2 0 6 3 0 0

3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9 7 3 6 1 2 9 3 1 9 4 7 8 4 5 0 3 6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7 4 5 2 5 4 7 4 4 3 0 7

Output:

1

0

53. Maximum sum Rectangle

Given a 2D array, find the maximum sum subarray in it. For example, in the following 2D array, the maximum sum subarray is highlighted with blue rectangle and sum of this subarray is 29.

1	2	-1	4	-20
-8	-3	4	2	1
3	8	10	-8	3
-4	-1	1	7	-6

Input:
First line of every test case consists of T test case. First line of every test case consists of 2 integers R and C , denoting number of rows and columns. Second line consists of R*C spaced integers denoting number of elements in array.
Output:
Single line output, print the Max sum forming a rectangle in a 2-D matrix
Example:
Input:
1
4 5
1 2 -1 -4 -20 -8 -3 4 2 1 3 8 10 1 3 -4 -1 1 7 -6
Ouptut:
29

54. Boolean Matrix Problem

Given a boolean matrix **mat[M][N]** of size **M** X **N**, modify it such that if a matrix cell **mat[i][j]** is **1** (or true) then make all the cells of ith row and jth column as **1**.
Input:
The first line of input contains an integer **T** denoting the number of test cases.
The first line of each test case is r and c, r is the number of rows and c is the number of columns.
The second line of each test case contains all the elements of the matrix in a single line separated by a single space.
Output:
Print the modified array.
Constraints:
$1 \le T \le 100$
$1 \le r, c \le 1000$
$0 \le A[i][j] \le 1$
Example:
Input:
3
2 2
1 0
0 0
2 3
0 0 0
0 0 1
4 3
1 0 0
1 0 0
1 0 0
0 0 0
Output:
1 1
1 0
0 0 1
1 1 1
1 1 1
1 1 1
1 0 0

Explanation:
Testcase1: Since only first element of matrix has 1 (at index 1,1) as value, so first row and first column are modified to 1.