Name: Lakhan Kumawat
Roll No: 1906055
Branch: CSE-1
Course: CS4403

03/05/2021

Solution 1:

Simple Solution

Approach: The simple idea is to traverse the array and search element one by one

Algorithm:

1. Run a nested loop, one outer loop for row and inner loop for column.
2. Check every element with x and if element found print "element got".
3. If element is not present print "element absent."

Time Complexity: $O(n^2)$ Space Complexity: $O(1)$

Yes we can design $O(n)$ solution.

a) Efficient Solution: ←

- Approach Simple idea is to remove a row or column in each comparison until an element is found. Start searching from the top-right corner of matrix.

Possible cases :-

1. Given number is greater than current number: This will ensure, that all the elements in the current row are smaller than the given number as the pointer is already at the right-most element and row is sorted. Thus, entire row gets eliminated and continue search on next row.

2. Given number smaller than current number: This ensures that ~~all~~ given elements is in current ~~column~~ row ~~are greater~~ than given number. Thus, and ~~entire column gets eliminated~~ and continue search on previous column. we have only search in that particular i^{th} row.

3. The given number is equal to current number: This will end the search

Name: Lakhan Kumawat
Roll NO: 1906055
Branch: CSE-1
CS4403
03/05/2021

5 Steps Algorithm:

Step 1: Let the given element be x, create two variable i=0, j=n-1 as index of row and column.

Step 2: Run a loop until i=0.

Step 3: Check if the current element is greater than x then decrease count of j Exclude the current column.

Step 4: check if the current element is less than x the increase count of i. Exclude current row.

Step 5: If element is equal than print the position and end.

Time Complexity: O(n) Space Complexity = O(1)

Name : Lokhan kumawat
Roll NO:1906055
Branch: CSE-1
CS4403
03/05/2021

Solution 2:

Given: Vertex Set {1, 2, 3, 4, 5}

W =

	1	2	3	4	5
1	0	1	8	1	4
2	1	0	12	4	9
3	8	12	0	7	3
4	1	4	7	0	2
5	4	9	3	2	0

Step 1: To get Minimum Spanning Tree with 1 as leaf node, we must first remove node 1 from the graph.

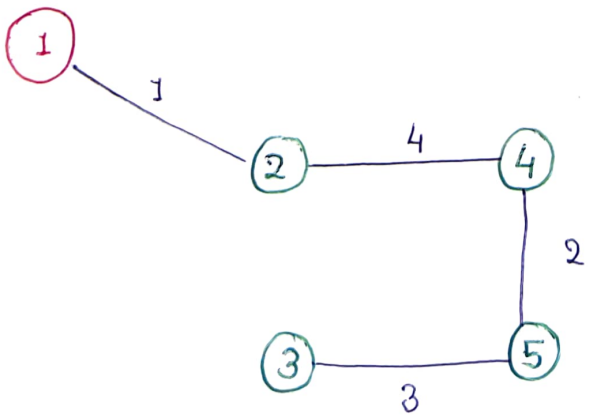
Step 2: Then we must find MST of the remaining graph and then join 1 to obtained MST with minimum weight edge

Now as node 1 is removed, we won't Consides 1st row and 1st column.

Now list down edges in increasing order of their weight.

4—5 ⇒2 , 3-5 ⇒ 3, 2-4 ⇒ 4 and so on.

To get MST, we must form edges in above increasing order such that there is no cycle until vertices are covered.



Name: Lakhan kumawat
Roll No: 1906055
CSE-1
CS4403

03/05/2021

Design and Analysis of Algorithms

Joining node 1 to the MST with minimum edge weight

So minimum possible weight

$= 2 + 3 + 4 + 1$

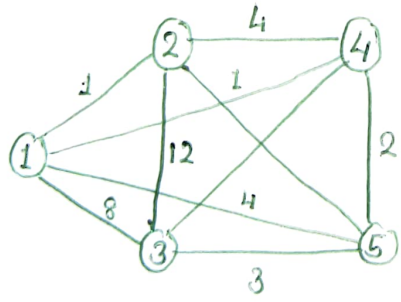
Minimum weight = 10

Solution: 3 Given graph.

Time Complexity of Dijkstra's Single Source Shortest-path

$O(V^2)$

[Time Complexity = $O(n^2)$]



The order in which vertices are finalised in Dijstra's algorithm is given below :-

(1) (1) ← Source vertex.

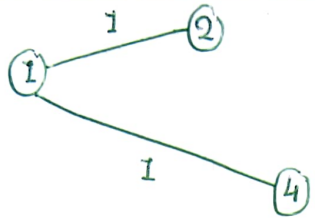
vertex	1	2	3	4	5
distance	0	∞	∞	∞	∞

(2) (1) —1— (2) Vertex (2) & vertex (4) have same minimum value 1.

vertex	1	2	3	4	5
Distance	0	1	∞	∞	∞

Lakhan kumawat
1906055
CSE-1
CS4403
03/05/2021

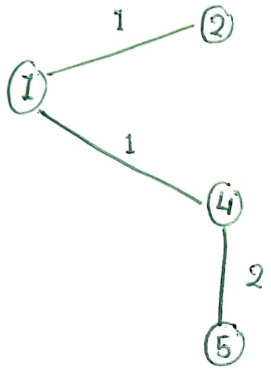
(3)



Selecting 1

vertex	1	2	3	4	5
distance	0	1	∞	1	∞

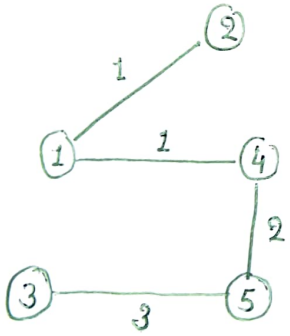
(4)



Selecting next minimum. 5.
added at cost 2.

vertex	1	2	3	4	5
distance	0	1	∞	1	3

(5)



vertex	1	2	3	4	5
distance	0	1	6	1	3

Final
← Shortest path

Order of finalization: ① → ② → ④ → ⑤ → ③

So Shortest path length for vertex 1 is

① → ② ⇒ 1
① → ③ ⇒ 6 (through 4 & 5)
① → ④ ⇒ 1
① → ⑤ ⇒ 3 (through 4)

Name: Lakhan Kumawat
Roll No: 1906055
Branch: CSE-1

CS4403
03/05/2021

Solution 4:

Approach: We traverse through the array and for every index, find the number of smaller elements on its right side of array. This can be achieved using nested loop. Sum up all the counts for all index in array and print sum.

Algorithm:

Step 1: Traverse through the array start to end.

Step 2: For every element, find the count of elements smaller than current number up to that index using another loop.

Step 3: Sum up the count of inversions for every index

Step 4: print the count of inversions.

Implementation :-

```
int getInversionCount(int n, int arr[])
{
    int inversion_count = 0;

    for (int i=0; i<n-1; i++)
        for (int j= i+1; j<n; j++)
            if (arr[i] > arr[j])
                inversion_count++;

    return inversion_count;

}
main()
{
    cout << getInversionCount(n, arr);
}
```

Name: Lakhan Kumawat
Roll No: 1906055
Branch: CSE-1
CS4403
03/05/2021

Time Complexity : $O(n^2)$

Two nested loops are needed to traverse the array from start to end so time complexity is $O(n^2)$

Name: Lakhan Kumawat
Roll No: 1906055
CSE-1

CS4403
03/05/2021

Solution 5:

Statement 1: The problem of determining whether there exists a cycle in an undirected graph is in P.

Solution: we can find the presence of a cycle inside an undirected graph by using DFS technique

Time complexity for doing so is [O(E+V)]

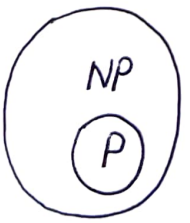
which is obviously polynomial time order, hence it lies in class P.

So, i) is True.

Statement 2: The problem of determining whether there exists a cycle in an undirected graph is in NP.

Solution: We know that every problem which is in class P. is also in class NP from the fact that class P ⊆ NP.

$$P \subseteq NP$$



So, ii) is True.

Name: Lakhan Kumawat
Roll No: 1906055
CSE-1
CS4403
03/05/2021

Statement 3> : The problem A is NP-Complete, there exists a non-deterministic polynomial time algorithm to solve A.

Solution: problem A is NP-complete.

which means A belongs to the language class NP and if any other problem of NP class can be reduced to A for solving.

Since A belongs to NP class.

hence it will take non-deterministic polynomial time to solve A.

Hence iii> is also True.

So the options:

[(A) (i), (ii) and (iii) is correct.] ✓