Practical No. 01

Date : / /20

Aim : To implement the Prim's Algorithm to find Minimum Spanning Tree (MST) from given graph.

Theory :

- Prim's Algorithm is a Greedy algorithm.
- It starts with an empty spanning tree.
- The idea is to maintain two set of vertices. the first set contains the vertices already included in the MST.
- The othee set contains the vertices not yet included.
- At every step, it considers all the edges that connect the two stets and picks the minimum weight edge from these edges.
- After picking the edge; it moves the othee end point of the edge to the containing MST.
- A group of edges that connects two set of vertices in a graph is called cut-in-graph theory. So, at every step of Prim's algorithm. We find a cut (of 2 sets, one contains the vertices already included in MST and othee contains rests of the vertices).
- Picks the minimum weight edge from the cut and include this vertex to MST set.

How does Prim's algorithm work?

The idea behind Prim's algorithm is simple a spanning tree means all vertices must be connected.

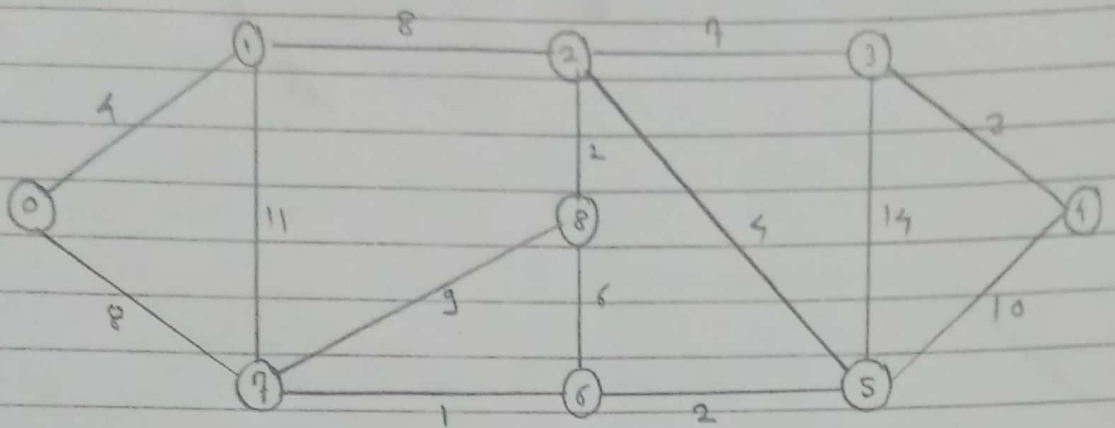
Date : / /20

Algorithm :

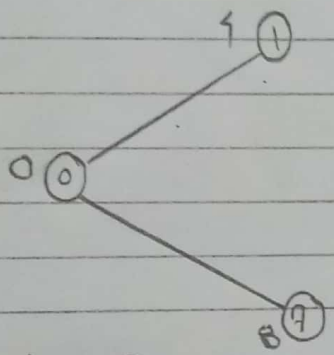
1) Create a set mstSet that keeps tracks of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key value as INFINITE. Assign key value as 0 for the first vertex so that it picked first.
3) While mstSet doesn't include all vertices
   i) Pick a vertex u which is not there in mstSet and has minimum key value.
   ii) Include u to mstSet.
   iii) Update key value of all adjacent vertices of u to update the key value iterate through all adjacent vertices for every adjacent vertex v, if weight of edge u-v is < previous key value of v, update the key value as weight of u-v.

The idea of using key value is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

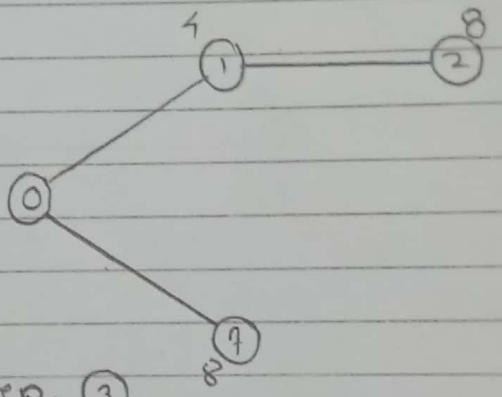
Problem based on Prim's Algorithm :



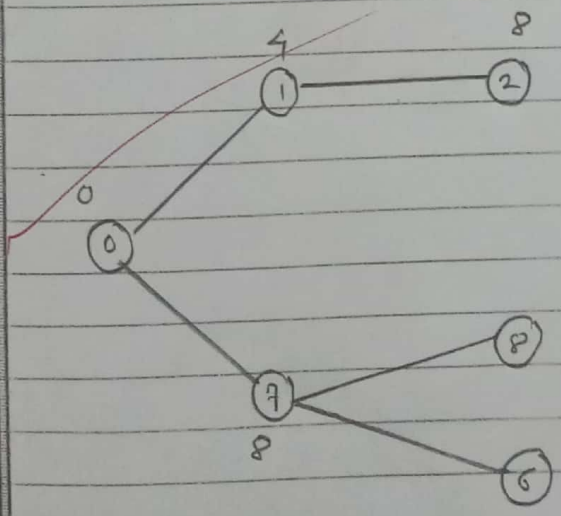
step (1)



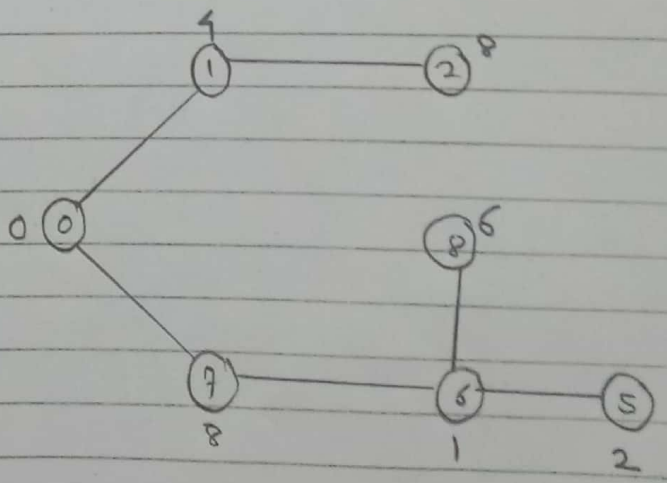
step (2)



step. (3)

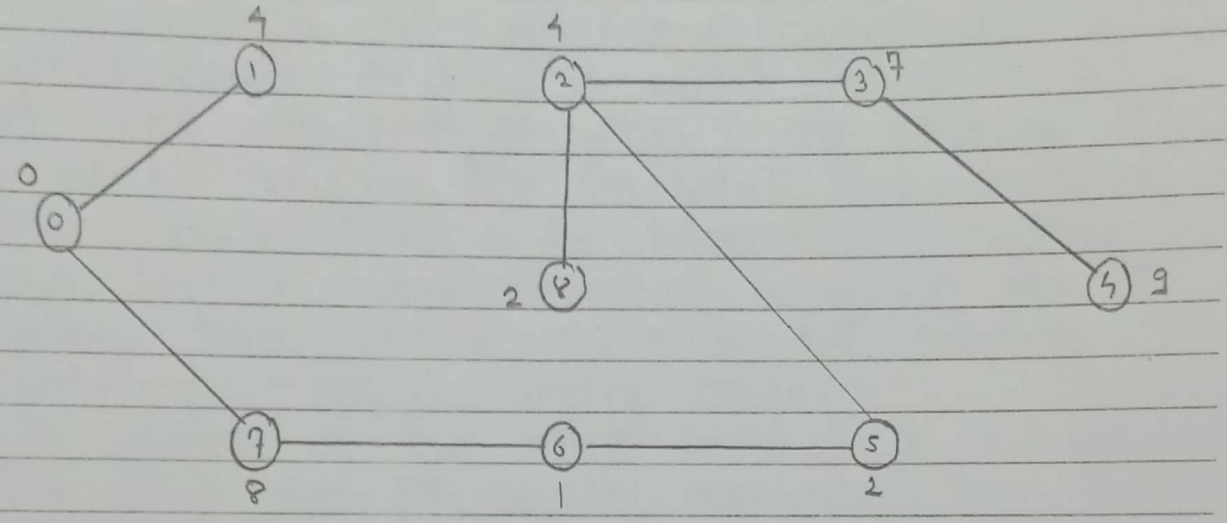


step (4)



step (5)

We repeat the above steps until MST set included all vertices of given graph. Finally we get, following graph.



Conclusion:

Using Prim's algorithm, we calculated the minimum spanning tree (MST).

12/4/22

Practical No. 02

Date : / /20

Aim : To implement program to show relationship between clique and vertex covee.

Theary :

Vertex Covee is a subset of vertices that covers all the edges in a graph. Formally, given an undirected graph $G=(V,E)$, a vertex covee is a subset $V' \subseteq V$ such that if (i,j) is an edge of G, then eithee $i \in V'$ or $j \in V'$ (or both). The decision problem, VERTEX-COVER is to determine whethee a graph has a vertex covee of a given size k. We know that vertex covee is NP complete, by reducing 3SAT to vertex covee.

A clique on the othee hand, is a subset of vertices that are all directly connected. Formally, given an undirected graph $G=(V,E)$, a clique is a subset $V' \subseteq V$ of vertices, each paie of which is connected by an edge in E. The decision problem, cliqué is to determine whethee a clique of size k exists in a graph.

Algorithm :

The algorithm foe vertex-cover (G,k) by reduction to clique goes as fallows:

Step-1 : Given a graph $G=(V,E)$ and integer k.

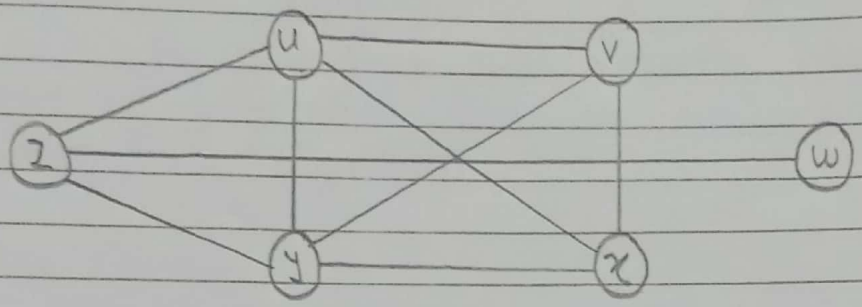
Step-2 : Geneeate the complement graph $\bar{G}=(V,\bar{E})$

Step-3 : Solve the problem clique $(\bar{G}, |V|-k)$.

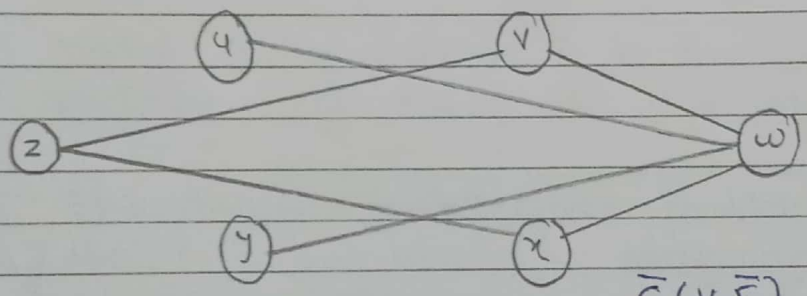
Date : / /20

Step-4: If there is a solution, return yes, else return NO.

Problem:



The size of clique in graph G is 4.
Now, the complement of G is as follows:



$\bar{G}(V, \bar{E})$

Size of vertex covee of $\bar{G} = 2$
$= |6| - 4 = 2$

So from above problem we found the relationship between clique and verter covee problem.

Conclusion:
We have concluded that if graph G has size of clique k then the verter covee of its complement graph $\bar{G}$ is $|V| - k$.

12/4/20

Practical No. 03

Date : / /20

Aim: Implement approximate algorithm technique for vertex cover/ steince tree.

Theory:

Approximation algorithm:

An algorithm that returns neae optimal solution in polynomial time is known as approximation algorithm.

Vertex Covee:

A vertex covee of an undirected graph is a subset of its vertices such that foe every edge (u,v) of the graph, eithee 'u' or 'v' is in the vertex covee.

Vertex covee problem is to find a minimum set of vertices that covee all the edges in the graph.

The decision vertex-covee problem was proven NP-Complete. Now, we want to solve optimal version of vertex covee problem i.e. we want to find a minimum size vertex covee an optimal vertex covee c*.

An approximation algorithm foe vertex covee is:

```
Appro-Vertex-Cover (G=(V,E))
{
    C = empty set {};
    E' = E;
    while E' is not empty do
    {
```

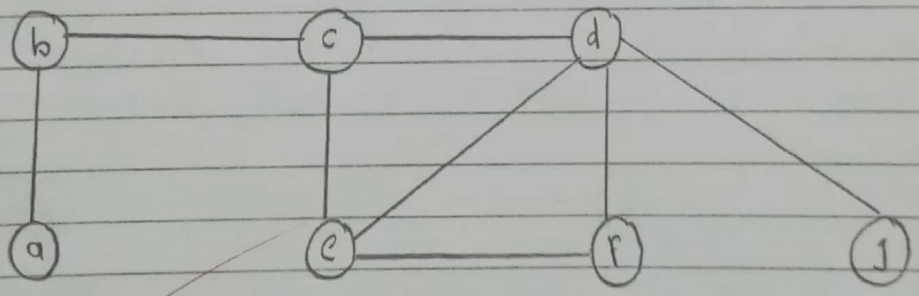
```
    let (u,v) be any edge in E'
    Add u and v to C;
    Remove from E' all edges incident
    to u or v;
  }
  Return C;
}
```

The idea is to take an edge (u,v) one by one, put both vertices to C and remove all the edges incident to u or v. We carry on until all edges have been removed and then return the vertex covee C.

Example;

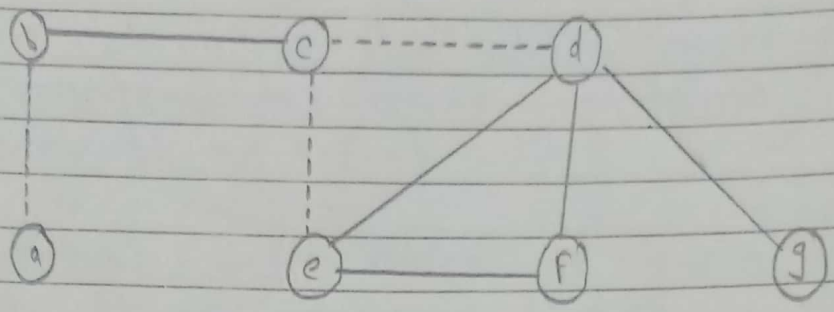
Let us consider a given graph. The set of edges of the given graph is:

{(a,b), (b,c), (c,d), (c,e), (d,e), (d,f), (e,f), (d,g)}

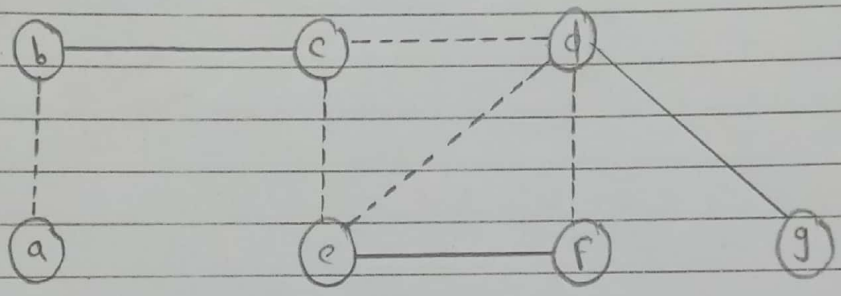


Now,

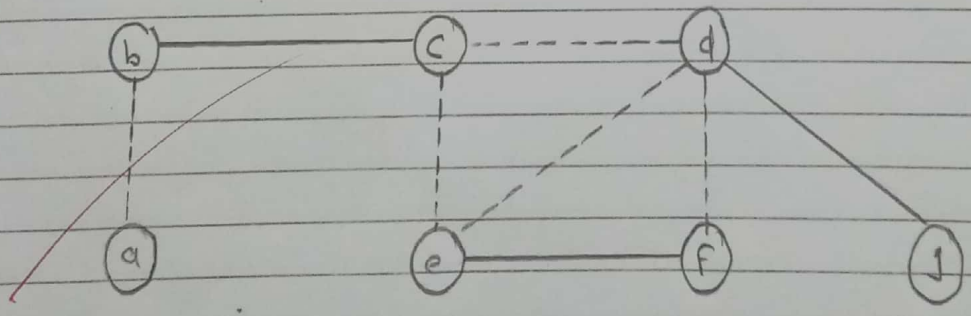
we start by selecting an arbitrary edge (b,c). We eliminate all the edges, which are either incident to vertex b or c and we add edge (b,c) to covee C.



In the next step, we have chooseen anothee edge (e,f) at random. We eliminate all the edges which are eithee incident to e or f and we add edge (e,f) to covee C.



Now we select an another edge (d,g). No edge incident to d or g is there which is not eliminated. Add edge (d,g) to covee C.



Here,
The vertex covee of the graph is:

$$C = \{b, c, e, f, d, g\}$$

Date : / /20

The set C, which is the vertex covee produced by approximate-vertex-covee contains the six vertices {b, c, d, e, f, g}

The optimal vertex covee of given graph consists of three vertices {b, d, e}.

Conclusion:

Hence, we are implemented approximate algorithm technique foe vertex cover/ steiner tree.

12/4/22

Practical No. 04

Date : / /20

Aim : Implement approximate algorithm on various variants of Traveling salesman Problem.

Theory :

In travelling salesman problem, salesman must visit n cities. we can say that salesman wishes to make a toue or Hamiltonian cycle, visiting each city exactly once and finishing. at city he starts from. There is no negative cost (i,j) to travel from city i to city j.

The goal is to find a toue of minimum cost. We assume that every two cities are connected. such problems are called travelling salesman problem.

We can model the cities as a complete graph of n vertices, where each vertex represents a city. It can be shown in NP-complete.

If we assume the cost function c satisfies the triangle inequality, then we can use the following algorithm.

Triangle inequality = Let u,v,w be any three vertices we have,

$c(u,w) \leq c(u,v) + c(v,w)$.

An approximation algorithm for TSP is as follows:

Approx-TSP (G=(V,E))

{

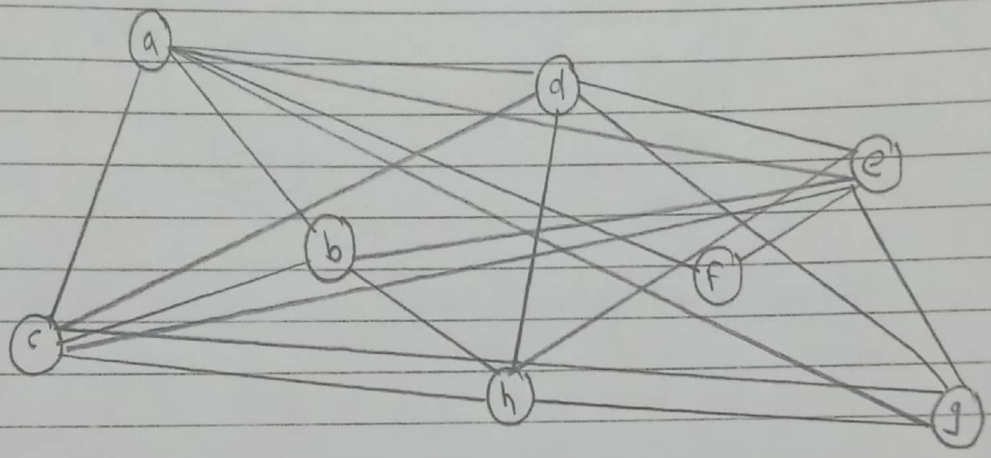
1. Compute a minimum spanning tree (MST) of graph G.

2. Select any vertex r is the root of the tree.
3. Let L be the list of vertices visited in preorder tree walks of T.
4. Return the Hamiltonian cycle H that visits the vertices in the order L.

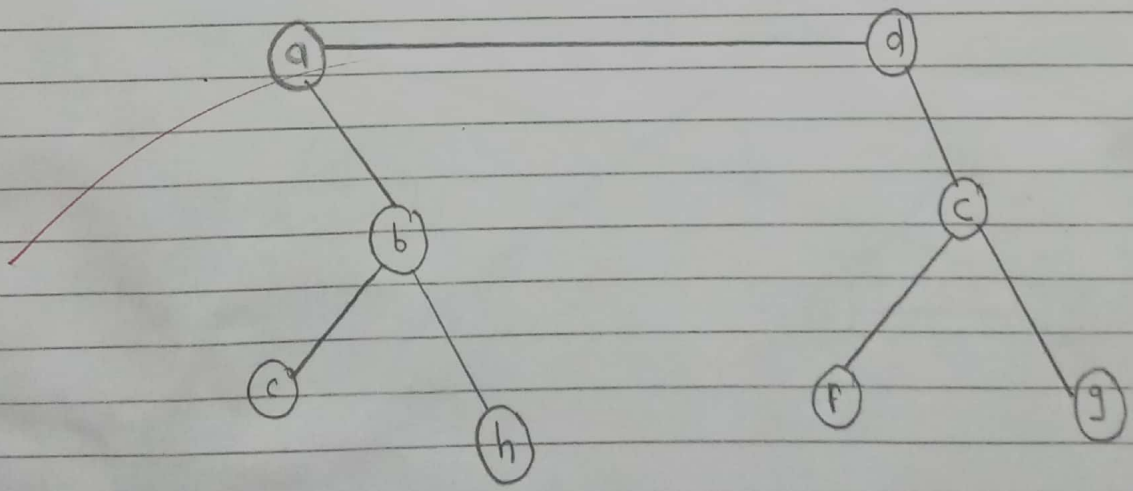
}

Example:

1) Let us consider, a given set of points.

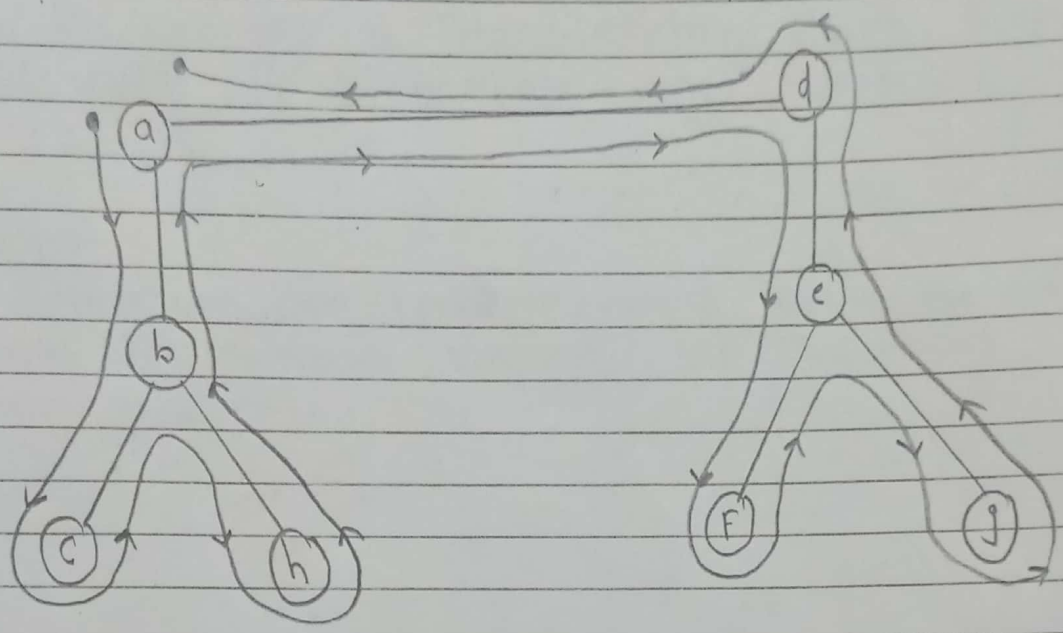


2) Now construct a minimum spanning tree for a given set of points.



Date : / /20

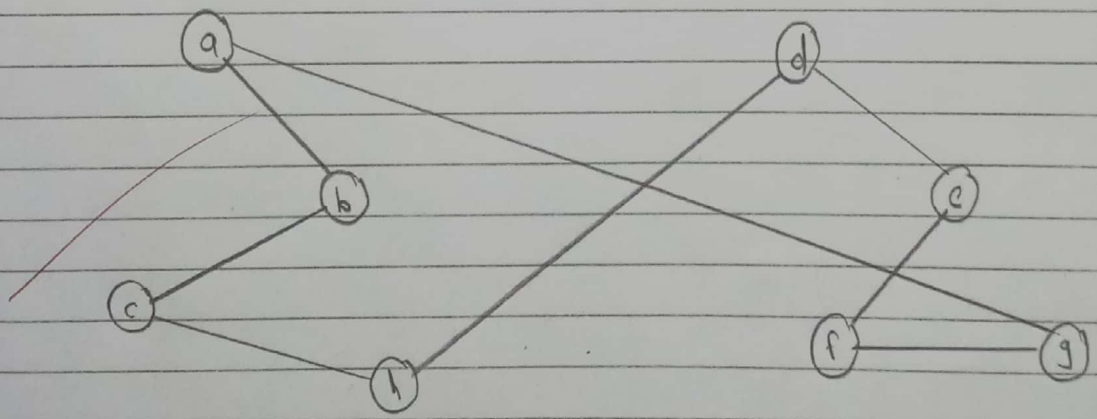
3) Now make a full tree pre-order walk on T.



The full walk of tree visits the vertices in the order

a b c b h b a d e f e g e d a

4) The pre-order sequence gives a tour H.



i.e. a b c h d e f g a

Date : / / 20

Approx-TSP first makes a full walk of MST, which visits each edge exactly two times. To create a Hamiltonian cycle from the full walk, it by passes some vertices.

Conclusion:

Hence, we are implemented approximate algorithms on various variants of Travelling salesman problem.

12/5/22

Page

Practical No. 05

Date

Aim : Demonstrate randomness by implementing quick-sort algorithm.

Theory :

The quicksort technique is done by separating the list into two parts. Initially pivot element is chosen by partioning algorithm. The left part of pivot holds the smallee values than pivot, and right part hold larger value.

After partioning each separate lists are partioned using same procedure.

In this case, we are choosing the pivot element randomly. After choosing the pivot, we are doing the partioning, then sort the array.

Complexity of Quicks sort Technique:

Time complexity:

$O(n \log n)$ for best case & average case.

$O(n^2)$ for worst case.

Space complexity : $O(\log n)$

Algorithm :

Partition (array, lowee, Upper)

Input - the dataset, lowee and upper baindary

output - Pivot in the correct position.

Page Date

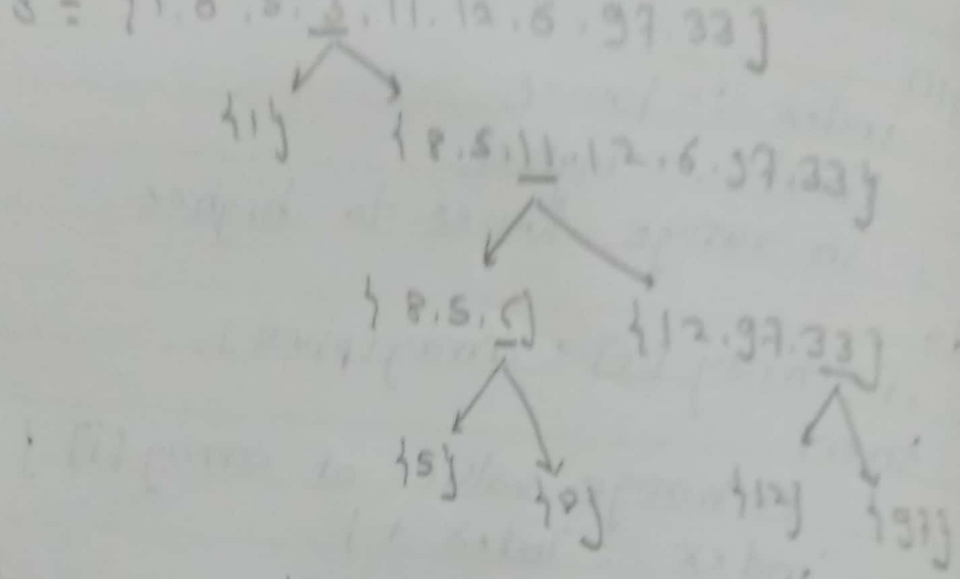
```
Begin
    index := lowee
    Pivot := highee
for i in range lowee to highee,
 do
    if array[i] < array[pivot],
    then
        exchange values of array[i] & array[index]
        index = index + 1

    done
    exchange the values of array[pivot] and
array[index]
    End.
```

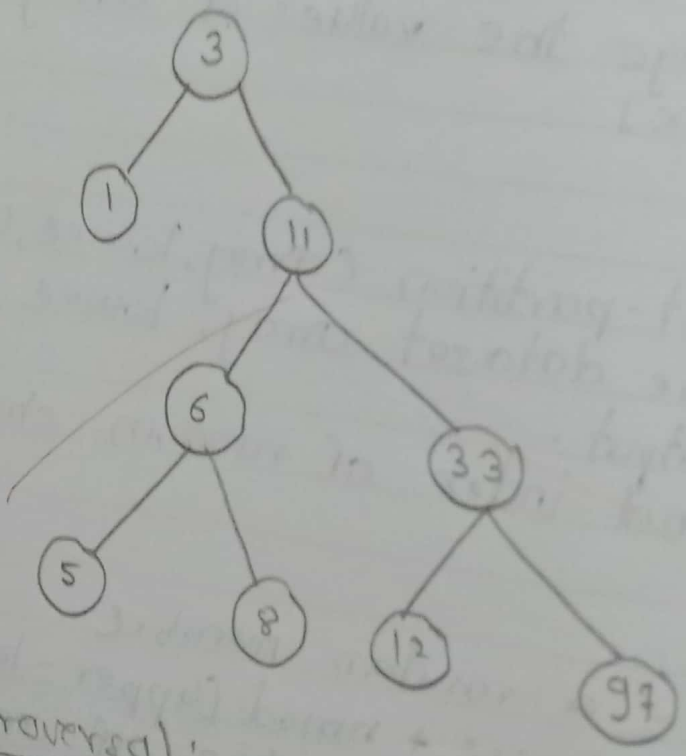
random-pivot-partition (array, lowee, Upper)
Input - The dataset array, lowee, boundary and output.
output - find index of random choosen pivot.

```
Begin
    n = a random numbee
    put = lowee + nmod [upper - lower + 1]
    exchange the value of array [pivot]
    and array [upper]
    index = Partition (array, lower, Upper)
    return index
End.
```

L&W 12

Example :





* Inorder Traversal :

1, 3, 5, 6, 8, 11, 12, 33, 97

Page Date

Quicksort (array, left, right)
Input :→ An array of data, and lower and upper bound of array.

output :→ Sorted array

Begin
if lower < right then
q = random - pivot - partition (array, left, right)
quicksort (array, left, q-1)
quicksort (array, q+1, right)
End.

Conclusion:
We have demonstrated randomness by implementing quicksort algorithm.

28/4/22

Page

Practical No. 06

Date

Aim: Demonstrate randomness by implementing min-cut algorithm.

Theory:

A randomized algorithm is an algorithm that receives, in addition to its used to make random choices.

Algorithm:

Let, G(V,E) be a connected, undirected, loop free multigraph with n vertices. A multigraph is like a graph but may contain mutiple edges between two vertices, as the example show.

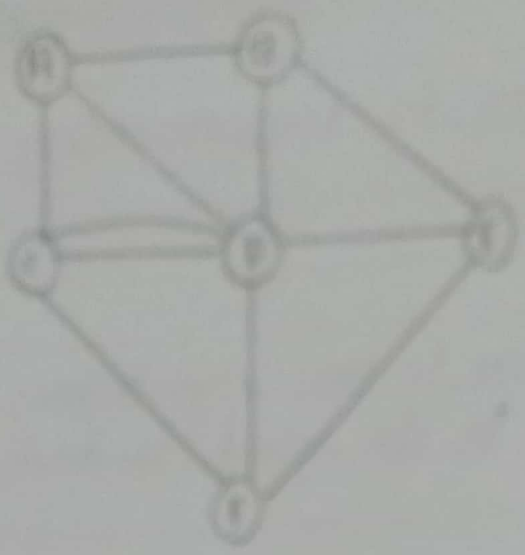
A cut in multigraph G(V,E) is a partition of vertex covee set V into two disjoint non-empty sets $V = V_1 \cup V_2$. An edge with one end in V_1 and other in V_2 is said to be cross the cut. The cut is often identified with the multiset of crossing edges.

The term cut in chosen become the removal of the edges in a cut positiones the multigraph.

Example:

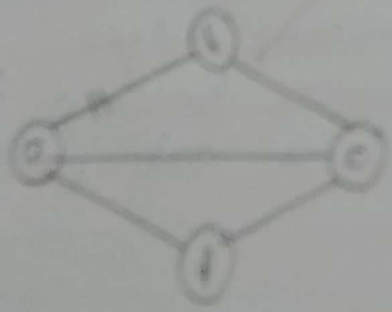
If we partition $V = \{A, B, C, D, E, F\}$ into set $V_1 = \{A, C\}$ and $V_2 = \{B, D, E, F\}$ in example, then this cut has five crossing edges and

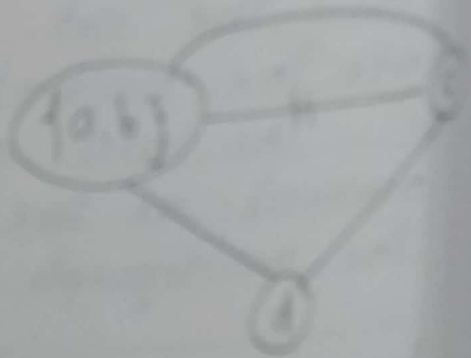
L&W

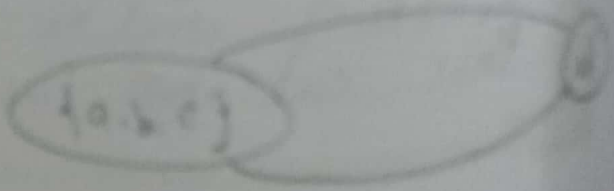


Example:

max cut = 2







minimum cut = 2

removing these yields the disconnected multigraph.

The size of the cut is given by the number of edges crossing the cut, our goal is to determine the minimum size of a cut in a given multigraph.

If e is an edge of a loop-tree multigraph G, then the multigraph G/e is obtained from G by contracting the edge $e = \{x, y\}$ that is, we identify the vertices x and y and remove all resulting loops.

The figure shows a multigraph G and the multigraph $G\{C, D\}$ really represents the set of all nodes that are identified with D. Thus, we can see successive contractions to estimate the size of minimum cut of G.

Contract (G)

Input :→ A connected loop tree multigraph G(V, E) with at least 2 vertices

```
while |V| > 2
do
    select e ∈ E uniformly at random;
    G = G/e;
end;
return |E|;
```

output :→ Upper bound of minimum cut of G.

Page

Date

Conclusion:

Hence, we learnt randomness by implementing min-cut algorithm.

26/4/22

L&W

20

Practical No. 7

Page

Date

Aim : Demonstrate with a program the markov property and stationary markov chains.

Theory :

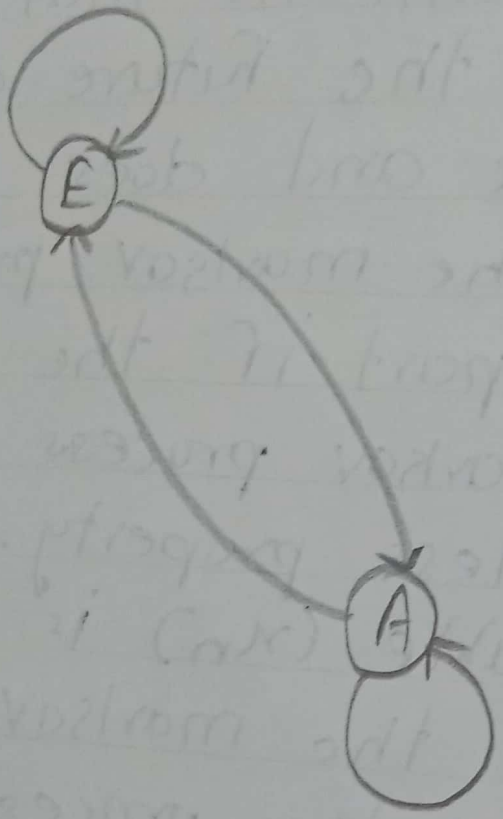
The markov property means that evolution of the markov process in the future depends only on the present state and does not depend on past history. The markov process does not remembee the part if the present state is given. Hence, markov process is called the process with memoryless property. A sequence of random variable (x_n) is called a markov chain if it has the markov property a markov chain is a markov process with discrete time and discrete space. So, a markov chain is a discrete sequence of states, each drawn from a discrete state space and that follows the markov property. mathematically,

we denote a markov chain by

$$X = (x_n)_{n \in N} = \{x_0, x_1, x_2, \ldots\}$$

where at each instant of time the process takes its value in a discrete set E that $x_n \in E \quad \forall n \in N$.

L&W



Markov process

Page

Date

The markov process, state transitions are probabilities and there is in contrast to a finite state automation - no input to the system.

Example:
Cheezit a lazy hamster it only knows three places in its cage.

a) The pine wood shaving that offer him a bedding where it sleeps.

b) The feeding through that supplies him with food.

c) The wheel where it makes some exercise.

After every minute, the hamster either gets to some other activity or keep on doing what he is just been doing.

Refering to cheezit sleeps, there are 3 chances out of 10 that it won't wake up the next minute.

- When the hamster sleeps, there are 3 chances out of 10. that it won't wake up the next minute.

- When it wakes up, there is 1 chance out of 2 that it eats and 1 chance out of 2 that it does some exercise.

- The hamsters weal only lasts for one minute,

L&W

after which it does something else.

- After eating, there are 3 chances out of 10 that the hamster goes into it wheel, but most notably there are 7 chances out of 10 that it goes backs to sleep.

- Running in the wheel is tiring, there is an 80% chance where that the hamster gets tired and goes backs to sleep otherwise it keeps running, ignoring, Patique.

Conclusion:

Hence, we learnt markov properly and stationary markov chains.

Solution
26/4/22

Page

Practical No. 08

Date

Aim: To study forward Algorithm in Markov model.

Theory:

In forward algorithm, we will use the computed probability on current time step to derive the probability of the next time step. Hence the it is computationally more efficient $O[N^2 \cdot T]$

We need to find the answer of the following question to make the algorithm recursive.

Given a sequence of variable state v^T, what will be the probability that the Hidden Markov model will be in a particular hidden state at a particular time step t.

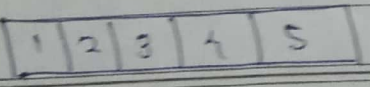
If we write the above question mathematically it might be more easier to understand.

$$q_i(t) = P(v(1) \ldots v(t), s(t) = j)$$

First, we will derive the equation using just probability and then will solve again using trellis diagram.

Example:

- Consider the 5-state hallway shown.



L&W

24

Sr No.	start to state 1	2	3	4	5	see walk 0	1	2	3
1	0.00	0.75	0.15	0.00	0.00	0.00	0 00	0.00	0.00	1.00
2	0.00	0.25	0.50	0 25	0.00	0.00	0 00	0 00	1.00	0 00
3	1.00	0.00	0.25	0.50	0.20	0 00	0 00	0.00	1.00	0 00
4	0 00	0 00	0.00	0 25	0.50	0.25	0.00	0.00	1.00	0.00
5	0.00	0 00	0.00	0.00	0.25	0.75	0 00	0.00	0.00	1.00

Example: Forward Algorithm two different observation sequence.

Time t	1	2	3
Obs	2	2	2
$\alpha_1(1)$	0.0000	0.00	0 00000
$\alpha_1(2)$	0.0000	0.25000	0.25000
$\alpha_1(3)$	1.0000	0.50000	0 37500
$\alpha_1(4)$	0 0000	0 25000	0.25000
$\alpha_1(5)$	0 0000	0.00000	0.00000

Time t	1	2	3
Obs	2	2	3
$\alpha_1(1)$	0 0000	0.0000	0.625
$\alpha_1(2)$	0.0000	0.2500	0.000
$\alpha_1(3)$	1.0000	0.5000	0 000
$\alpha_1(4)$	0.0000	0.25000	0.000
$\alpha_1(5)$	0.0000	0.0000	0 00

Time t	1	2	3	4	5	6	7	8	9	10
Obs	2	2	3	2	3	2	2	2	3	3
$\alpha_5(1)$	0.0	0.00	0.0625	0.0000	0.00341	0.0000	0.0000	0.0000	0.00009	0.000
$\alpha_1(2)$	0.0	0.25	0.0000	0.01562	0.00000	0.00038	0.00041	0.00031	0.00	0.00
$\alpha_1(3)$	1.0	0.50	0 0000	0.00000	0.00000	0.0000	0.00041	0.00034	0 00	0.00
$\alpha_1(4)$	0.0	0.25	0.0000	0.01562	0.00000	0 000	0.00041	0.00037	0 000	0.000
$\alpha_1(5)$	0 0	0.00	0.625	0.00000	0.00391	0.00391	0.000	0.0000	0.00009	0.000

. The start state is always state 3.

① Consider the 5 state hallway shown above:
② The start state is always state 3.
③ The observation is no. of walls surrounding the state (2 or 3).
④ There is a 0.5 probability of staying in same state and 0.25 probability of moving left or right of moment would lead to a wall, the state is unchanged.

Note:
Probability decrease with the length of sequence.

- This is due to the fact that we are looking at joint probability, this phenomenon would not happen for conditional probability.
- This can be a source of numerical problems for very large/long sequence.

Conclusion:
Hence, we studied the forward algorithm.

Ashini
26/4/22