GRAPH THEORY

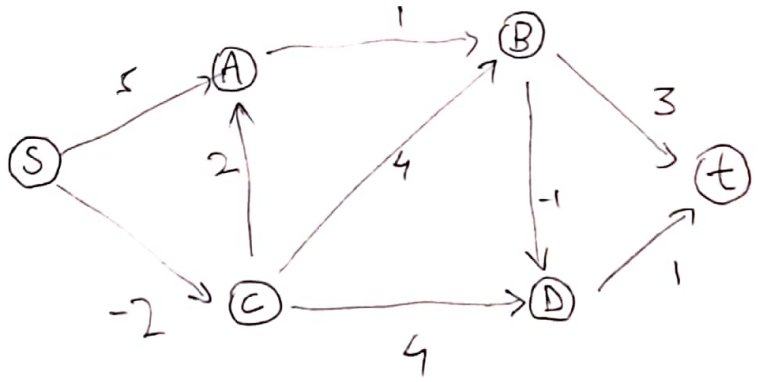
MC-405

ASSIGNEMENT-II

ANISH SACHDEVA
DTU/2K16/MC/13

~~30th October~~
30th September 2020

Q) Using Bellman Ford Algorithm, find the shortest Path between S and all other vertices.



We will work over every edge in the graph and for every edge we will use edge relaxation.

```
function Relax ( Edge u,v ):
    if distance(Source, v) > distance( Source, u) + e(u, v):
        distance(source, v) = distance (source, u) + e(u,v)
        parent[v] = u
```

We perform the following operations by maintaining an Edge List over which we iterate and we maintain a distances map and also a parent Map. Initially the distances map has infinity (∞) for all vertices except Source and parent map is set to NULL for every vertex except source.

Distance Map

S	0
A	∞
B	∞
C	∞
D	∞
t	∞

Parent Map

S	—
A	NULL
B	NULL
C	NULL
D	NULL
t	NULL

Edge List

S → A

S → C

A → B

C → A

C → B

C → D

B → D

B → t

D → t

I) First Iteration

i) S → A

$d[S,A] > d[S,S] + e(S,A)$

$\infty > 0 + 5$

$\infty > 5$ TRUE

$d(S,A) = 5$ parent[A] = S

Distance Map

S	0
A	5
B	∞
C	∞
D	∞
t	∞

Parent Map

S	—
A	S
B	NULL
C	NULL
D	NULL
t	NULL

ii) S → C

$d(S,C) > d(S,S) + e(S,C)$

$\infty > 0 + -2$

$\infty > -2$ TRUE

$d(S, C) = -2$ & Parent[C] = S

Distance Map

S	0
A	S
B	∞
C	-2
D	∞
t	∞

Parent Map

S	—
A	S
B	NULL
C	S
D	NULL
t	NULL

iii) A → B

$d(S, B) > d(S, A) + e(A, B)$

$\infty > 5 + 1$

$\infty > 6$

$d(S, B) = 6$ parent[B] = A

Distance Map

S	0
A	S
B	6
C	-2
D	∞
t	∞

Parent Map

S	—
A	S
B	A
C	S
D	NULL
t	NULL

iv) C → A

$d(S, A) > d(S, C) + e(C, A)$

$5 > -2 + 2$

$5 > 0$ TRUE

$d(S, A) = 0$ parent[A] = C

Distance Map

S	0
A	0
B	6
C	-2
D	∞
t	∞

Parent Map

S	—
A	C
B	A
C	S
D	NULL
t	NULL

v) C → B

$d(S,B) > d(S,C) + e(C,B)$

$6 > -2 + 4$

$6 > 2$ TRUE

$d(S,B) = 2$ parent[B] = C

Distance Map

A	0
S	0
B	2
C	-2
D	∞
t	∞

Parent Map

S	—
A	C
B	C
C	S
D	NULL
t	NULL

vi) C → D

$d(S,D) > d(S,C) + e(C,D)$

$\infty > -2 + 4$

$\infty > 2$ TRUE

$d(S,D) = 2$ parent[D] = C

Distance Map	
S	0
A	0
B	2
C	-2
D	2
t	∞

Parent Map	
S	—
A	C
B	C
C	S
D	C
t	NULL

vii) B → D

$d(S,D) > d(S,B) + e(B,D)$

$2 > 2 + -1$

$2 > 1$ TRUE

$d(S,D) = 1$ parent[D] = B

Distance Map	
S	0
A	0
B	2
C	-2
D	1
t	∞

Parent Map	
S	—
A	C
B	C
C	S
D	B
t	NULL

viii) B → t

$d(S,t) > d(S,B) + e(B,t)$

$\infty > 2 + 3$

$\infty > 5$ TRUE

$d(S,t) = 5$ parent[t] = B

Distance Map	
S	0
A	0
B	2
C	-2
D	1
t	5

Parent Map	
S	—
A	C
B	C
C	S
D	B
t	B

iv) D → t

$d(s,t) > d(s,D) + e(D,t)$

$5 > 1 + 1$

$5 > 2$ TRUE

$d(s,t) = 2$ Parent[t] = D

Distance Map		Parent Map	
S	0	S	–
A	0	A	E
B	2	B	C
C	-2	C	S
D	1	D	B
t	2	t	D

II) Iteration 2

i) S → A

$d(S,A) > d(S,S) + e(S,A)$

$0 > 0 + 5$

$0 > 5$ False

ii) S → E

$d(S,C) > d(S,S) + e(S,C)$

$-2 > 0 - 2$

$-2 \not> -2$ False

iii) A → B

$d(S,B) > d(S,A) + e(A,B)$

$2 > 5 + 1$

$2 \not> 6$ False

iii) $A \rightarrow B$

$d(S,B) > d(S,A) + e(A,B)$

$2 > 0 + 1$

$2 > 1$ True

Distance $(S,B) = 1$ Parent $[B] = A$

Distance Map

S	0
A	0
B	1
C	-2
D	1
t	2

Parent Map

S	-
A	C
B	A
C	S
D	B
t	D

iv) $C \rightarrow A$

$d(S,A) > d(S,C) + e(C,A)$

$0 > -2 + 2$

$0 > 0$ False

v) $C \rightarrow B$

$d(S,B) > d(S,C) + e(C,B)$

$1 > -2 + 4$

$1 > 2$ False

vi) $C \rightarrow D$

$d(S,D) > d(S,C) + e(C,D)$

$1 > -2 + 4$

$1 > 2$ False

vii) B → D

$d(S, D) > d(S, B) + e(B, D)$

$1 > 1 + -1$

$1 > 0$ True

$d(S, D) = 0$ Parent[D] = B

Distance Map

S	0
A	0
B	1
C	-2
D	0
t	2

Parent Map

S	—
A	C
B	A
C	S
D	B
t	D

viii) B → t

$d(S, t) > d(S, B) + e(B, t)$

$2 > 1 + 3$

$2 > 4$ False

ix) D → t

$d(S, t) > d(S, D) + e(D, t)$

$2 > 0 + 1$

$2 > 1$ True

$d(S, t) = 1$ Parent[t] = D

Distance Map

S	0
A	0
B	1
C	-2
D	0
t	1

Parent Map

S	—
A	C
B	A
C	S
D	B
t	D

Now, we one again run an iteration on all edges:-

III) Iteration 3:

i) S → A

$d(S,A) > d(S,S) + e(S,A)$

$0 > 0 + 5$

$0 > 5$ False

ii) ~~d(S,~~ S → C

$d(S,C) > d(S,S) + e(S,C)$

$-2 > -0 + -2$

$-2 > -2$ False

iii) A → B

$d(S,B) > d(S,A) + e(A,B)$

$1 > 0 + 1$

$1 > 1$ False

iv) C → A

$d(S,A) > d(S,C) + e(C,A)$

$0 > -2 + 2$

$0 > 0$ False

v) C → B

$d(S,B) > d(S,C) + e(C,B)$

$1 > -2 + 4$

$1 > 2$ False

vi) C → D

$d(S,D) > d(S,C) + e(C,D)$

$0 > -2 + 4$

$0 > 2$ False

vii) B → D

$d(S,D) > d(S,B) + e(B,D)$

$0 > 1 + (-1)$

$0 > 0$ False

viii) B → t

$d(S,t) > d(S,B) + e(B,t)$

$1 > 1 + 3$

$1 > 4$ False

ix) D → t

$d(S,t) > d(S,D) + e(D,t)$

$1 > 0 + 1$

$1 > 1$ False

As in this iteration, no distance was updated, we will not run any more iterations and the final result is as follows

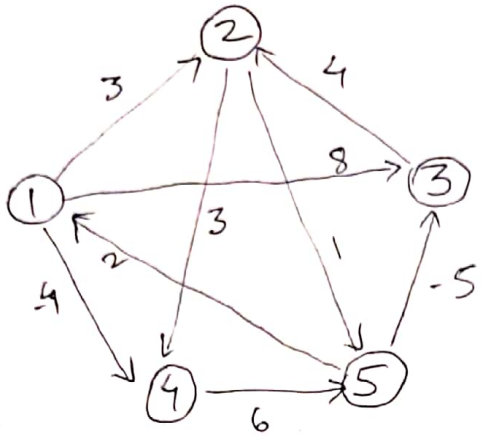
Distance Map

S	0
A	0
B	1
C	-2
D	0
t	1

Parent Map

S	—
A	C
B	A
C	AS
D	B
t	D

Q2) Using Floyd-Warshall Algorithm, find the shortest path between all pair of vertices.



We create the distance matrices as follows :—

$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & -4 & \infty \\ \infty & 0 & \infty & 3 & 1 \\ \infty & 4 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & 6 \\ 2 & \infty & -5 & \infty & 0 \end{bmatrix} \end{array}$$

Now, calculating D^1 as $D[i,j] = \min(D[i,j], D[i,1] + D[1,j])$

$D[2,3] = \min(D[2,3], D[2,1] + D[1,3])$

$\min(\infty, \infty + 8) = \infty$

$D[2,4] = \min(D[2,4], D[2,1] + D[1,4])$

$\min(3, \infty + -4) = 3$

$D[2,5] = \min(1, \infty + \infty) = 1$

$D[3,2] = \min(D[3,2], D[3,1] + D[1,2])$

$\min(4, \infty + 3) = 4$

$D[3,4] = \min[\infty, \infty + -4) = \infty$

$D[3,5] = \min(\infty, \infty - \infty) = \infty$

$D[4,2] = \min(\infty, \infty + 3) = \infty$

$D[4,3] = \min[\infty, \infty + 8) = \infty$

$D[4,5] = \min[6, \infty + \infty] = 6$

$D[5,2] = \min(\infty, 2 + \infty) = \infty$ 5

$D[5,3] = \min(-5, 2+8) = -5$

$D[5,4] = \min(\infty, 2+(-4)) = -2$

$D^1 =$

	1	2	3	4	5
1	0	3	8	-4	∞
2	∞	0	∞	3	1
3	∞	4	0	∞	∞
4	∞	∞	∞	0	6
5	2	5	-5	-2	0

Now, we calculate paths through vertex 2 :-

$D[1,3] = \min(8, 3 + \infty) = 8$

$D[1,4] = \min(-4, 3+3) = -4$

$D[1,5] = \min(\infty, 3+1) = 4$

$D[3,1] = \min(\infty, 4 + \infty] = \infty$

$D[3,4] = \min(\infty, 4+3) = 7$

$D[3,5] = \min(\infty, 4+1) = 5$

$D[4,1] = \min(\infty, \infty + \infty) = \infty$

$D[4,3] = \min(\infty, \infty + \infty) = \infty$

$D[4,5] = \min(6, \infty + 1) = 6$

$D[5,1] = \min(2, 5 + \infty) = 2$

$D[5,3] = \min(-5, 5 + \infty) = -5$

$D[5,4] = \min(-2, 5+3) = -2$

$D^2 =$

	1	2	3	4	5
1	0	3	8	-4	4
2	∞	0	∞	3	1
3	∞	4	0	7	5
4	∞	∞	∞	0	6
5	2	5	-5	-2	0

We now calculate paths from vertex (3) :-

$D[1,2] = \min(D[1,2), D[1,3] + P[3,2]) = \min(3, 8+4) = 3$

$D[1,4] = \min(-4, 8+7) = -4$

$D[1,5] = \min(4, 8+5) = 4$

$D[2,1] = \min(\infty, \infty+\infty) = \infty$

$D[2,4] = \min(3, \infty+7) = 3$

$D[2,5] = \min(1, \infty+5) = 1$

$D[4,1] = \min(\infty, \infty+\infty) = \infty$

$D[4,2] = \min(\infty, \infty+4) = \infty$

$D[4,5] = \min(6, \infty+5) = 6$

$D[5,1] = \min(2, -5+\infty) = 2$

$D[5,2] = \min(5, -5+4) = -1$

$D[5,4] = \min(-2, -5+7) = -2$

$D^3 =$

	1	2	3	4	5
1	0	3	8	-4	4
2	∞	0	∞	3	1
3	∞	4	0	7	5
4	∞	∞	∞	0	6
5	2	-1	-5	-2	0

Solving for all paths going through ④ :-

$D[1,2] = \min(3, -4+\infty) = 3$

$D[1,3] = \min(8, -4+\infty) = 8$

$D[1,5] = \min(4, -4+6) = 2$

$D[2,1] = \min(\infty, 3+\infty) = \infty$

$D[2,3] = \min(\infty, 3+\infty) = \infty$

$D[2,5] = \min(1, 3+6) = 1$

$D[3,1] = \min(\infty, 7+\infty) = \infty$

$D[3,2] = \min(4, 7+\infty) = 4$

$D[3,5] = \min(5, 7+6) = 5$

$D[5,1] = \min(2, -2+\infty) = 2$

$D[5,2] = \min(-1, -2+\infty) = -1$

$D[5,3] = \min(-5, -2+\infty) = -5$

$$D^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & -4 & 2 \\ \infty & 0 & \infty & 3 & 1 \\ \infty & 4 & 0 & 7 & 5 \\ \infty & \infty & \infty & 0 & 6 \\ 2 & -1 & -5 & -2 & 0 \end{bmatrix} \end{array}$$

Solving for all paths going through ⑤ :-

$D[1,2] = \min(3, 2+-1) = 1$

$D[1,3] = \min(8, 2+(-5)) = -3$

$D[1,4] = \min(-4, 2+(-2)) = -4$

$D[2,1] = \min(\infty, 1+2) = 3$

$D[2,3] = \min(\infty, 1+(-5)) = -4$

$D[2,4] = \min(3, 1+(-2)) = -1$

$D[3,1] = \min(\infty, 5 + 2) = 7$

$D[3,2] = \min(4, 5 + (-1)) = 4$

$D[3,4] = \min(7, 5 + (-2)) = 3$

$D[4,1] = \min(\infty, 6 + 2) = 8$

$D[4,2] = \min(\infty, 6 + (-1)) = 5$

$D[4,3] = \min(\infty, 6 + (-3)) = -4$

$$D^5 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & -3 & -4 & 2 \\ 3 & 0 & -4 & -1 & 1 \\ 7 & 4 & 0 & 3 & 5 \\ 8 & 5 & 1 & 0 & 6 \\ 2 & -1 & -5 & -2 & 0 \end{bmatrix} \end{array}$$

This resulting distance matrix represents the minimum distance between vertices.

Q3) Prove that the following are equivalent for an n-vertex graph T.

a) T is a tree.

This is simply a statement which we will prove true and will show all the following statements are equivalent and are indeed a t a tree T.

It is sufficient to show that all the following statements are equivalent to a tree T.

b) T is connected and has no cycles.

Here a graph T is connected and acyclic, but a tree T by definition is a connected acyclic graph, hence by definition it is a tree and is equivalent to (a).

c) For $u, v \in V(T)$, T has exactly one $u-v$ path.

We will prove both the sufficiency and necessity that for a tree T there must exist $u, v \in V(T)$ such that there is unique path between them and also show that if this condition holds then it is infact a tree.

First let us suppose that T is a tree and hence by definition it is completely connected and acyclic. So, for any 2 distinct vertices $u, v \in V(T)$ it contains a $u \pm v$ path. If it were to contain 2 paths then that would imply a cycle, but a tree contains no

cycle which is a contradiction, hence all paths must be unique. This proves the necessity, we now prove the sufficiency

Let there be a graph G_1 for which there exists a unique path between every 2 distinct vertices $u, v \in V(G_1)$. As there exists a path between every 2 vertices the graph G_1 is connected. Now, we need to show that the graph G_1 is acyclic,

Proof by contradiction :-

Let there be a cycle in the graph G_1 and hence there would exists 2 paths from vertex $u, v \in V(T)$ which is a contradiction $\Rightarrow\Leftarrow$ Hence, the graph G_1 is acyclic and hence $G_1 \cong T$ (True).

G_1 is equivalent to (a).

d) T is connected and has exactly n-1 edges.

We are already given that the graph is connected, so all we need to prove that it is acyclic as well.

Proof by contradiction :-

Let there be a cycle C in G_1. Let e be any edge in cycle $e \in C$. Now, we take

$$G_1' = G_1 \setminus e$$

After removing cyclic edge (Non-bridge edge) we know that $G_1 \setminus e$ must still be connected. So, we know that every connected graph has $\geq n-1$ edges, but in G_1' :-

$|E(G)| = n-1$

$|E(G \setminus e)| = n-2$

$|E(G')| = n-2$

But $|E(\mathcal{G})| \geq n-1 \quad \forall \mathcal{G} \in \mathcal{G}$ (Family of graphs)

Hence this is a contradiction $\Rightarrow\Leftarrow$ and our assumption that G contains a cycle is incorrect, hence G must be clearly acyclic. Hence G is connected acyclic graph and is a tree. Hence is equivalent to (a)

e) T is connected, but $T \setminus e$ is disconnected for every edge e.

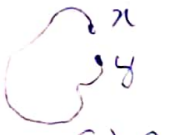
Here we are given that T is connected, so all we have to prove is that T is acyclic. We are also given that for every edge $e \in E(T)$ $T \setminus e$ is disconnected, hence $\forall e \in E(T)$ is a bridge edge. So, all we need to prove is that no bridge edge can lie on a cycle and our proof will be done.

Proof By Contradiction :-

Let e be a bridge edge and let it lie on a cycle C.

Cycle C
Graph G

Now, we remove e from G and obtain



$G \setminus e$

Here the components x and y are still connected and hence e can't be a bridge edge which is a contradiction $\Rightarrow\Leftarrow$

Hence, no bridge edge can be on a cycle and our graph T is acyclic and connected and hence is a Tree. So, it is equivalent to (a).

f) T has no cycle, but T + xy does, for any 2 non-adjacent vertices $x, y \in V(T)$.

We are given that the graph T is acyclic, so all we need to show is that it is also connected.

Proof By Contradiction :-

Let the graph T not be connected, Now let there be 2 vertices $x, y \in V(T)$ such that there doesn't exist any path between x and y. x and y belong to 2 disjoint disconnected components. Now, let us add an edge between x and y. So, we obtain T + xy which must contain a cycle as per the initial condition, and if there is a cycle then there must exist 2 paths (distinct) from x to y, but we have stated earlier that x and y are disconnected and xy is the only unique path between x and y $\Rightarrow\Leftarrow$. This is a contradiction, hence our initial assumption that the graph T is disconnected was incorrect, and the graph T is in fact connected and acyclic and is hence a Tree. ■

g) T has n-1 edges and no cycles.

We first show that for any tree T, it is acyclic and contains exactly n-1 edges which is the necessity condition. We then prove that if a graph has n-1 edges and is acyclic, it is a tree which is the sufficiency condition.

→ To prove: If T is a tree then it is acyclic and has m = n-1 edges.

By definition if T is a tree it is connected and acyclic, so we need only prove it has n-1 edges.

Proof By Mathematical Induction:-

Base Case: For trivial tree with one vertex |V(T)| = 1, we have |V(T)| = 1 (n-1) edges, m = 0.

•

K_1

Induction Hypothesis:

Let all trees of order n have n-1 edges, so now we need to show that for all trees with n+1 order must have n edges.

Let T be a tree of order n+1. We know that in a tree T there exists at least 2 leaves.

Let $v \in V(T)$ with degree 1, i.e. it is a leaf.

Let $T' = T - v$

$|V(T')| = n$ vertices therefore

T' has $n-1$ edges

Therefore we can see that $|E(T)| = |E(T')| + 1$

$|E(T)| = (n-1)+1$

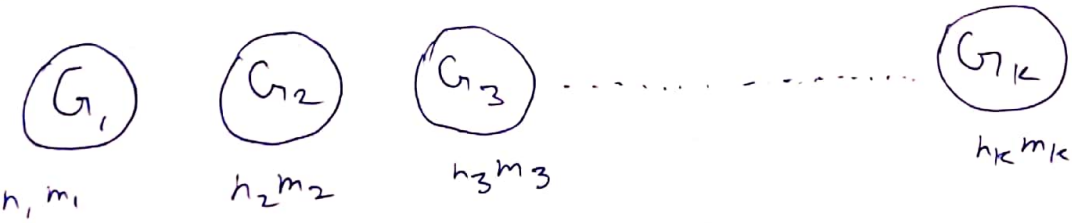
$|E(T)| = n$

This proves the inductive Hypothesis.

→ Given an acyclic graph with $m = n-1$, show that it is a tree.

We now prove the sufficiency. We are already given that the graph T is acyclic, so all we need to prove is that it is connected.

Let us assume that T has some connected components forming a family of graphs G_f.



We know that each component $G_i \in G_f$ is acyclic and fully connected, hence each component is a tree.

So, $|E(G_i)| = |V(G_i)| - 1 \quad \forall \; G_i \in G_f$

$m_i = n_i - 1 \quad \forall \quad i = 1, 2 \ldots k$

Now,

$$\sum_k m_i = \sum_k (n_i - 1)$$

$m = n - k$ where $|E(G)| = m$ and $|V(G)| = n$

and there are k components in the graph, but we are already given that :-

$m = n-1$, hence $k = 1$. There is only one component and the graph is fully connected and hence T was a tree, and is equivalent to (a).

Q4) Show that there are n^{n-2} Labelled trees with n vertices, $n \geq 2$ (Cayley's Formula)

Prüfer Encoding -

The most straightforward method of showing that a set has a certain number of elements is to find a bijection between the set and some other set with a known number of elements. In this case we find a bijection between the set of Prüfer sequences and the set of spanning trees.

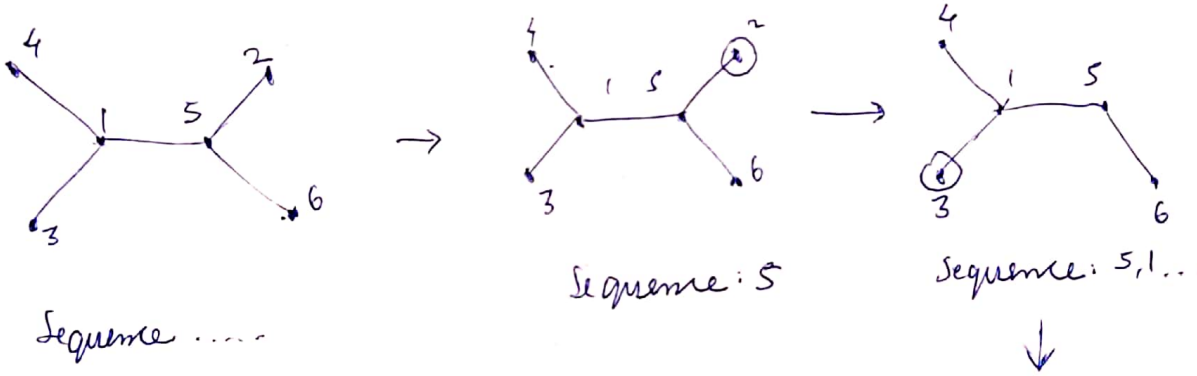
A prüfer sequence is a sequence of $n-2$ numbers, each being one of the numbers 1 through n. We should initially note that indeed there are n^{n-2} prüfer sequences for any given n. The following is an algorithm that can be used to encode any tree into a Prüfer sequence:

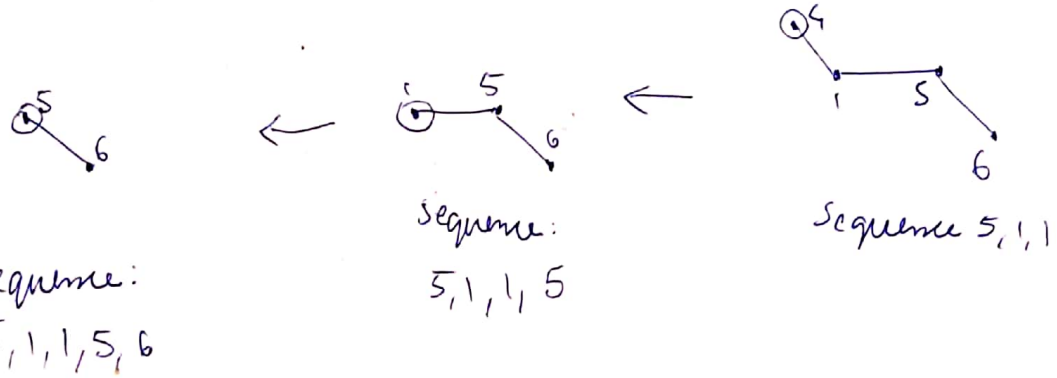
i) Take any tree, $T \in T_n$ whose vertices are labelled from 1 to n in any manner.

ii) Take the vertex with the smallest label whose degree is 1, delete it from the tree and write down the value of it's neighbour.

iii) Repeat the process with the new smaller tree. Continue until only one vertex remains.

This algorithm will give us a sequence of $n-1$ terms, but we know that the last term will always be the number n because even if initially $d(n)=1$, there will always be another vertex of degree 1 with smaller label. Since we already know the number of vertices on our graph by the length of our sequence, we can drop the last term as it is redundant. So now we have a sequence of $n-2$ elements encoded from our tree. Below is an example of encoding a tree on 6 vertices:—





After encoding we end up with our tree with sequence 5, 1, 1, 5, 6. Then we can drop the ending 6 and end with our Prüfer sequence and denote it by P. P = 5, 1, 1, 5. So, what should make us think that this is the only tree that gives us this sequence? It follows that every vertex has degree 1 + a

where a is the number of times that vertex appears in our sequence. This way of analyzing the Prüfer sequence provides us with a way of reconstructing an encoded tree.

The algorithm goes as follows:-

1. Find the smallest number from 1 to n that is not the sequence P and attach the vertex with that number to the vertex with the first number in P.
2. Remove the first number of P from the sequence. Repeat this process considering only the numbers whose vertices have not yet attained their correct degree.
3. Do until there are no numbers left in P. Remember to attach the last number in P to vertex n.

Since there are no ambiguities on how to encode or decode the sequence, we can see that for every tree there is exactly one corresponding tree. More formally, the encoding function can be thought of as taking a member of the set of spanning trees on n vertices T_n, to the set of prüfer sequences with n-2 terms P_n. Decoding would then be the inverse of the encoding function, and we have seen that composing those two functions results in the identity map. If we let f be the encoding function, then the above statements can be summarized as follows:-

$f: T_n \to P_n$ $\quad f^{-1}: P_n \to T_n$ $\quad f^{-1} \circ f = Id$

Since we have found a bijective function between T_n and P_n, we know that they must have the same number of elements. We know that $|P_n| = n^{n-2}$ and so $|T_n| = n^{n-2}$.

Hence proved. ■

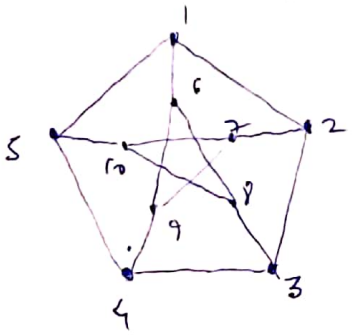
Q5) Define a spanning Tree of a graph. Find the spanning tree in the Peterson Graph.

Spanning Tree

A spanning tree T of an undirected graph G is a subgraph that is a tree which includes all of the vertices of G, with a minimum possible number of edges.

In general a graph may have several spanning trees, but a graph that is not connected will not contain. If all of the edges of G are also edges of a spanning tree T of G, then G is a tree and is identical.

Spanning Tree in Peterson Graph



We implement Kruskal's algorithm:-

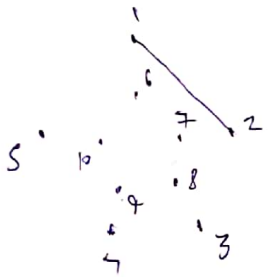
Edges

1 - 2	1 - 6	6 - 8
2 - 3	2 - 7	6 - 9
3 - 4	3 - 8	7 - 10
4 - 5	4 - 9	7 - 9
5 - 1	5 - 10	8 - 10

We iterate on each edge 1 by one and add it in the resulting graph if it doesn't form a cycle.

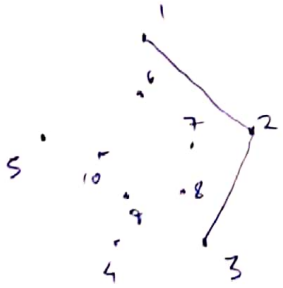
i) 1-2

We add this edge.



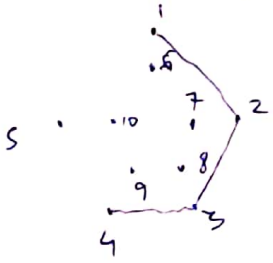
ii) 2-3

We add this edge



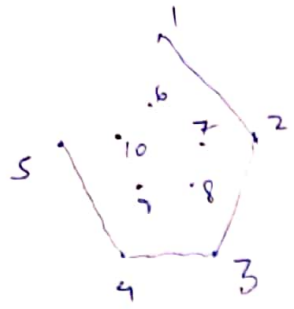
iii) 3-4

We add this edge



iv) 4-5

We add this edge

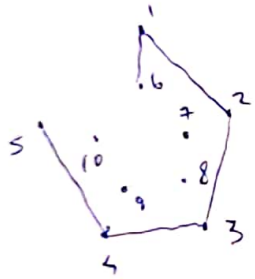


v) 5-1

This edge will create a cycle, so we do not add it.

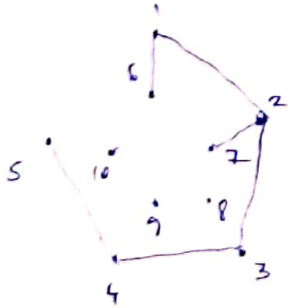
vi) 1-6

We add this edge



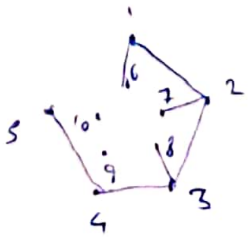
vii) 2-7

We add this edge



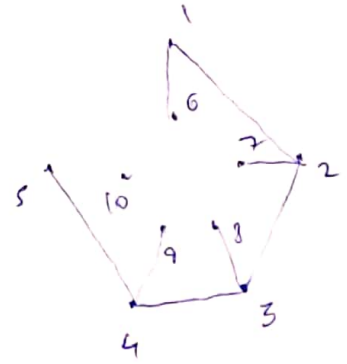
viii) 3-8

We add this edge



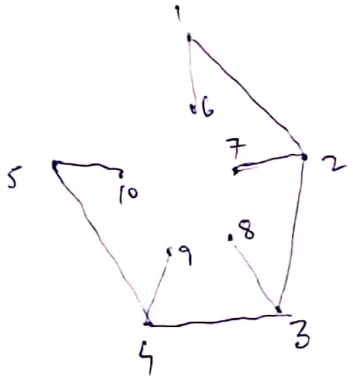
ix) 4-9

We add this edge in the spanning tree.



x) 5-10

We add this edge in the spanning tree.



This is the spanning tree obtained from Petersen Graph and we stop our iteration here as we have obtained $n-1 = 9$ edges.

Q.6) Show that a Hamiltonian Path in a graph is a Spanning tree?

A spanning tree is a tree which covers all the vertices in a graph G. A Hamiltonian path is a path in G such that traversing on the path covers all the vertices on the graph G without any repititions. This also implies that no edge is repeated as a repeated edge would automatically imply repeated vertices. So, we can see that a Hamiltonian path on G, H covers all the vertices in V(G) just as is done by a spanning tree.

So, all we need to prove is that the Hamiltonian path H is a tree i.e. acyclic and connected.

We know that for any 2 distinct vertices $u, v \in V(G)$ there exists a path between them on the Hamiltonian path as it covers all vertices hence the Hamiltonian path is fully connected. We now prove it is acyclic.

Proof By Contradiction :-

Let there exist a cycle in the Hamiltonian path $C = (v_1 v_2 v_3 \ldots\ldots v_n v_1)$ of length n in the graph. If $v_1 v_n$ there would exist 2 edges from v_1, namely $v_1 v_2$ and $v_1 v_n$ so, when we traverse the path we

will arrive at vertex v_1 twice which isn't allowed in a Hamiltonian Path as each vertex is traversed only once, which is a contradiction $\Rightarrow\Leftarrow$.

So, our initial assumption was incorrect and the path H is acyclic. Hence H is a tree and as it covers all vertices, it is a spanning tree. ∎

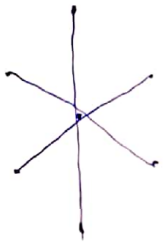
Q7) Prove / Disprove that every spanning tree is a Hamiltonian path.

We can disprove this statement by giving examples of graphs with spanning trees that are not Hamiltonian Paths.

$K_{1,3}$

This is a tree and it's own spanning tree, but doesn't contain a Hamiltonian Path.

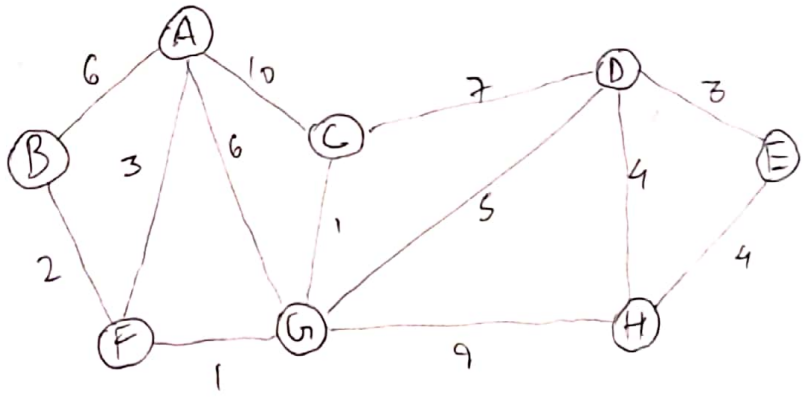
$K_{1,4}$



$K_{1,6}$

Graphs of type $K_{1,n}$ $(n \geq 3)$ also called stars are all spanning trees, but only are not Hamiltonian Paths.

(Q 8) Find the Minimum Spanning tree using Prim's and Kruskal's Algorithm.



Kruskal's Algorithm

In Kruskal's algorithm, we add all edges in a MinHeap data structure and iterate over them one by one, we add an edge only if it doesn't form a cycle, if an edge would form a cycle we don't add it. The pseudo code can be represented as:-

```
function Kruskal (G : Graph):
    result ← new Graph from vertex set of G
    edges ← E(G)
    minheap ← Add Edges in Heap
    for edge e in minheap:
        if adding e in result forms cycle:
            Do Nothing
        else:
            add e in result
    return result
```

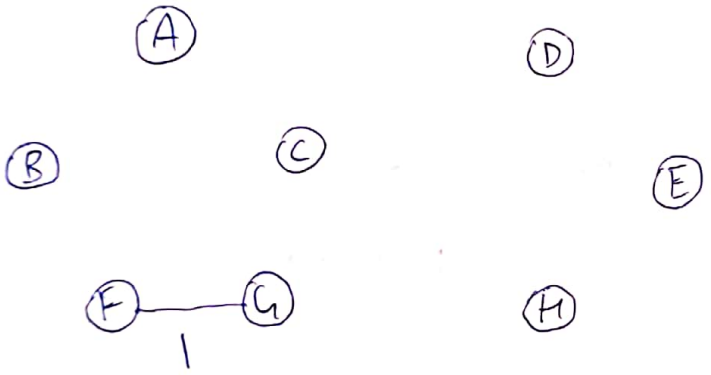
Edge List (sorted by Weights) -

i) FG – 1
ii) GC – 1
iii) BF – 2
iv) AF – 3
v) DE – 3
vi) DH – 4
vii) HE – 4
viii) GD – 5
ix) AB – 6
x) AG – 6
xi) CD – 7
xii) GH – 9
xiii) AC – 10

Iterating over edges :-

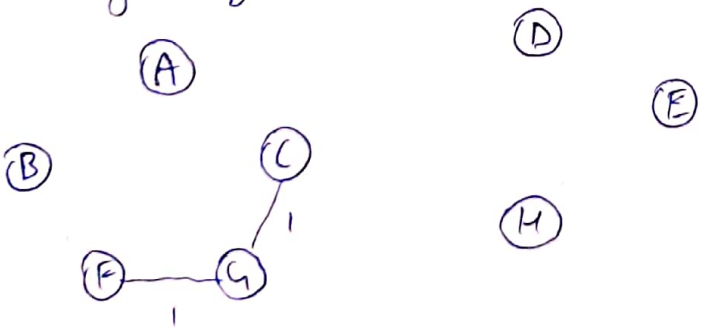
i) FG – 1

Adding Edge



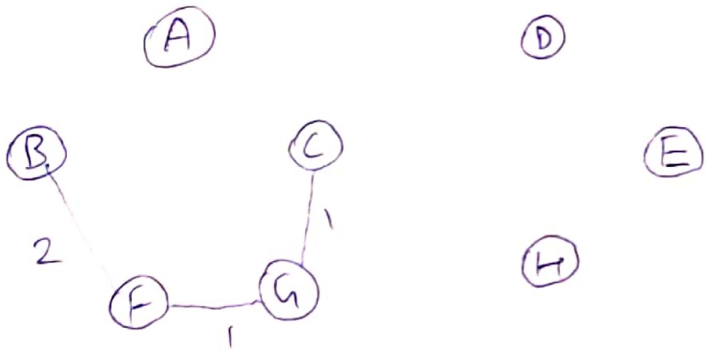
ii) GC – 1

Adding Edge



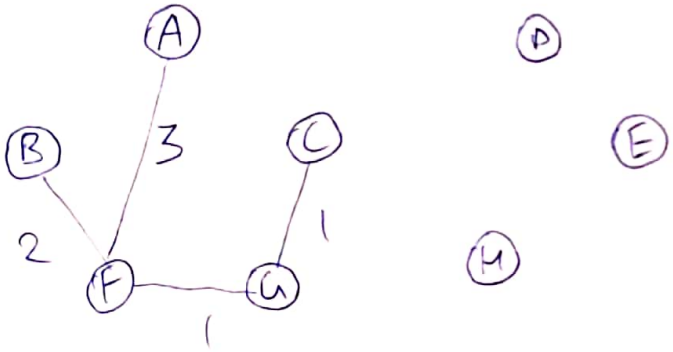
iii) BF -2

Adding Edge



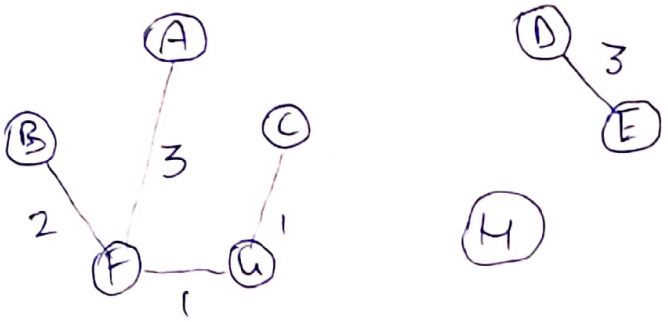
iv) AF - 3

Adding Edge



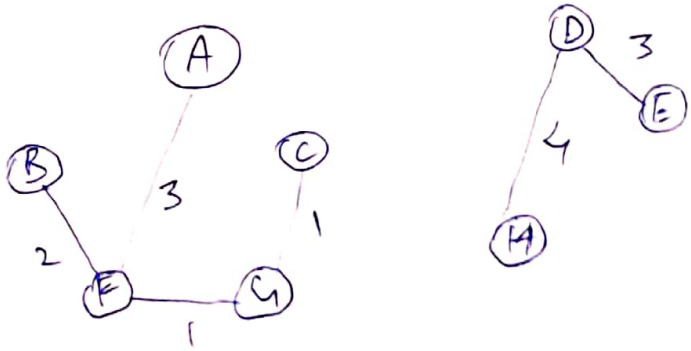
v) DE - 3

Adding Edge



vi) DH - 4

Adding Edge

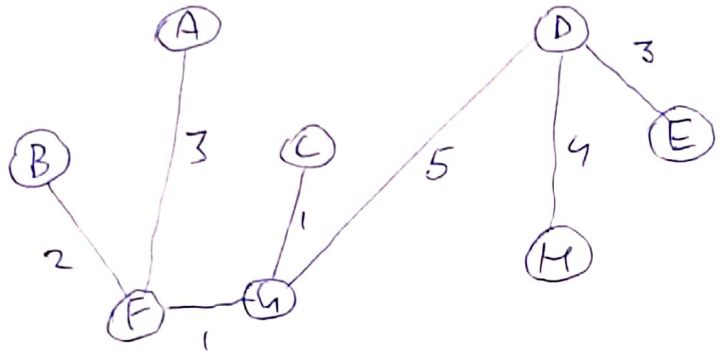


viii) HE – 4

This will form a cycle, hence we will not add this edge.

viii) GD – 5

Adding Edge



ix) AB – 6

This will form a cycle, hence we will not add edge.

x) AG – 6

This will form a cycle, hence we won't add.

xi) CD – 7

This will form a cycle, hence will not add it.

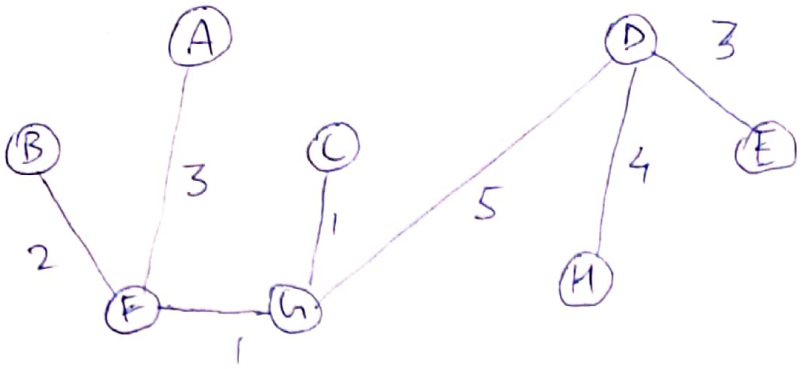
xii) GH – 9

This will form a cycle, hence we will not add.

xiii) AC – 10

This will form a cycle, hence we will not add it.

Hence, we obtain the following minimum spanning tree:-



Prim's Algorithm

In Prim's algorithm, we arbitrarily choose a starting point or starting vertex from the graph and add it's edges into the minimum heap. From this heap we select the smallest edge (with smallest weight) and add to resulting graph. We continue to do so until we have run out of edges from the heap. Also, we will only add an edge if that vertex hasn't already been used. We maintain a set of vertices to check this.

```
function prims(G: Graph):
    result ← New empty graph
    v ← Arbitrary vertex from G
    minheap ← Add edges incident to v in heap
    for every edge e in minheap:
        if e not in result and v_A(e) & v_B(e) not in
            result:
            . add e in result
              heap ← add edges incident to v_B(e)
    return result
```

We start by taking A as the initial vertex. Adding edges incident on A in the heap.

Edges

AB – 6
AF – 3
AG – 6
AC – 10

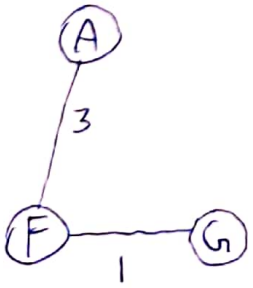
We select (AF – 3)

Adding edges incident on F

Edges

AB – 6
AG – 6
AC – 6
FB – 2
FG – 1

We select (FG – 1)

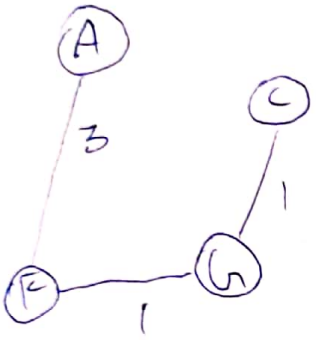


Adding Edges incident on G in the heap.

Edges

AB – 6
AG – 6
AC – 10
FB – 2
GC – 1
GD – 5

We select (GC – 1)

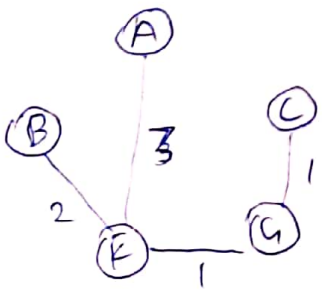


Adding edges incident to C in the heap

Edges

AB – 6 GD – 5
AG – 6
AC – 10
FB – 2
CD – 7

We select (FB – 2)

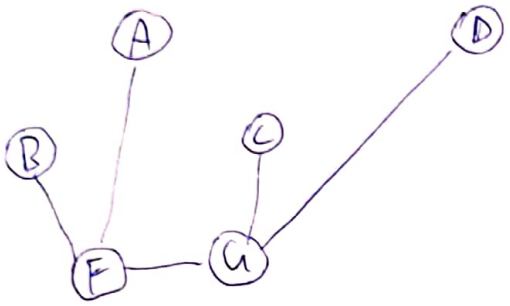


Adding edges incident to B in the heap :–

Edges

AB – 6
AG – 6
AC – 10
CD – 7
GD – 5

We select (GD – 5)

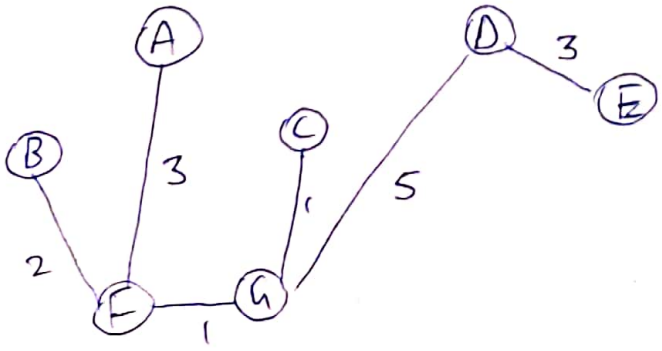


Adding Edges incident on (D) in the graph.

Edges

AB – 6
AG – 6
AC – 10
CD – 7
~~GD – 5~~
DH – 4
DE – 3

We select edge (DE – 3)

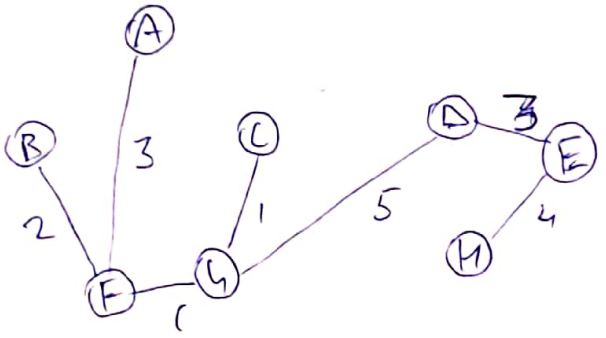


We add edges incident on (E) :-

Edges

AB – 6
AG – 6
AC – 10
CD – 7
DH – 4
EH – 4

We select (EH – 4)



Edges

AB – 6
AG – 6
AC – 10
CD – 7
DH – 4

We select (DH – 4), but both vertices are already present.

Edges

AB – 6
AG – 6
AC – 10
CD – 7

We select (AB – 6), but both vertices already exist.

Edges

AB - 6
AL - 10
CD - 7

We select (AB-6) but both vertices already exist.

Edges

AL - 10
CB - 7

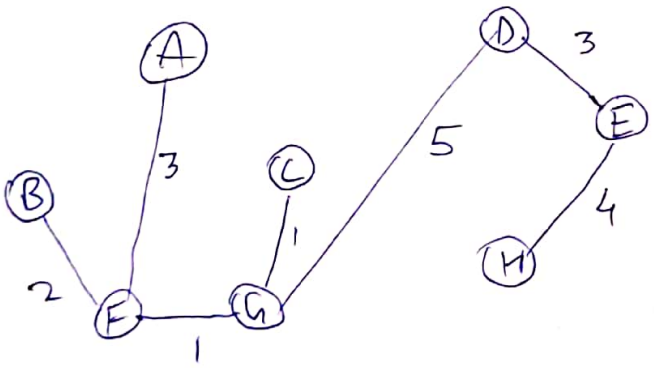
We select (CD-7), but both vertices already exist.

Edges

AL - 10

We select (AL-10), but both vertices are already present.

We get the minimum spanning tree as :-



Q9) Describe the Chinese Postman Problem and travelling Salesman Problem. Give examples for both. Also, mention the main differences among the two.

Travelling Salesman Problem

The travelling Salesman Problem asks the following question "Given a list of cities and the distances between each pair of cities, what is the shortest path that visits each city

exactly once and returns to the origin city?"

This is a NP-hard problem and has many practical applications in applications such as planning, Logistic and the manufacture of microchips. In the Travelling Salesman problem or TSP we take every city as a vertex on the graph and the distance between each pairwise city is taken as the or weighted undirected Edge. So, the TSP assumes a weighted undirected Graph and visiting every vertex is necessary, but not every edge, hence we form a Hamiltonian Cycle when we perform the Travelling Salesman Problem. The time complexity for travelling Salesman Problem is $O(n^2 2^n)$ where n is the number of vertices in the graph. Some examples where this problem is used in real life are:-

i) Imagine Dominos / Zomato delivery Person wants to deliver food to multiple homes and wants to optimize this for optimal fuel efficiency.

ii) Imagine Amazon calculating delivery path for all it's delivery persons to optimize fuel and time.

iii) An old application of the TSP is to schedule the collection of coins from payphones throughout a given region.

iv) A semi-conductor manufacturer has used Lourenço's implimentation of the Chained Lin-Kernighan heuristic in experiments to optimize scan chains in Integrated circuits. Scan chains are routes included on a chip for testing purposes and it is useful to minimize their lengths for both timing and power reasons.

Chinese Postman Problem / Route Inspection Problem

Chinese Postman Problem is to find a shortest closed Path or Circuit that visits every edge of a connected undirected graph. When the graph has an Euler Circuit (A closed walk that covers every edge exactly once). That circuit is an optimal solution. Otherwise, the optimization problem is to find the smallest number of graph edges to duplicate so that the resulting multigraph does have an Euler Circuit. The Chinese Postman Problem can be solved in Polynomial Time.

The main difference between Chinese Postman problem and TSP is that in TSP, all vertices in the graph must be visited exactly once whereas in Chinese Postman Problem, all edges must be visited at least once and hence vertices can be repeated.

Some applications of the Chinese Postman Problem are:-

i) Finding a maximum cut in a planar graph.

ii) Finding a minimum mean length circuit in an undirected graph.

iii) Routing snow ploughs through a city so that all roads are cleared of snow in minimum time. Similarly this is also applicable for Trash collection, Road Sweeping, Highway Lawnmowers, transmission Line inspections, School Bus Routing etc.