Assignment 2

By: Shivam Negi

1. Give a brief definition for the following:

a. Tree graph

A tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any acyclic connected graph is a tree. A forest is a disjoint union of trees.
A tree is an undirected graph G that satisfies any of the following equivalent conditions:

- G is connected and has no cycles.
- G is acyclic, and a simple cycle is formed if any edge is added to G.
- G is connected, but is not connected if any single edge is removed from G.
- G is connected and the 3-vertex complete graph K3 is not a minor of G.
- Any two vertices in G can be connected by a unique simple path.

b. Adjacency List

An adjacency list is a collection of unordered lists used to represent a finite graph. Each list describes the set of neighbors of a vertex in the graph. This is one of the representation of graphs for use in computer programs.

c. 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 minimum possible number of edges. In general, a graph may have several spanning trees, but a graph that is not connected will not contain a spanning tree.
A tree is a connected undirected graph with no cycles. It is a spanning tree of a graph G if it spans G (that is, it includes every vertex of G) and is a subgraph of G (every edge in the tree belongs to G). A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no cycle, or as a minimal set of edges that connect all vertices.

d. **Breadth-first search** (**BFS**):

BFS is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbours.

This non-recursive implementation is similar to the non-recursive implementation of depth-first search, but differs from it in two ways:

- it uses a queue (First In First Out) instead of a stack (First In Last Out) and
- it checks whether a vertex has been discovered before enqueueing the vertex rather than delaying this check until the vertex is dequeued from the queue.

e. Admissible Heuristic: A heuristic function is said to be admissible if it never overestimates the cost of reaching the goal, i.e. the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path

2. Arrange the following functions in increasing order of asymptotic growth:

5n^5

0.33^n

5n^3

(n ^2) √n

5^n

log n

√n

Sol:

d≪loglogn≪log^an≪nb≪cn≪n!≪n^n

1. 0.33^n (Exponentially decreasing)
2. log n
3. √n
4. n^2 √n
5. $5n^3$
6. $5n^5$
7. 5^n (Exponentially increasing)

3. T(n) = 8T (n/2)+ n

Sol:

For T(n) = 8T (n/2) + n, a=8, b=2, d=1

logba > d => log28 = 3 > 1

T(n)= Theta (nd) = Theta(n1)

T(n)= Theta (n)

4. T(n) = n^2 T (n/2) + log n

Sol:

Master theorem doesn't apply. As 'a' is not constant, and can be less than 0.

5. T(n) = 4T (n/2)+ n^2

Sol:

T(n) = 4T (n/2) + n2 , a=4, b=2, d=2

logba = d => log4/log2 = 2 = d

T(n)= Theta (n^dlog (n)) = Theta(n^2log(n))

T(n)= Theta (n^2log (n))

6. (22, 13, 26, 1, 12, 27, 33, 15) Merge Sort

Mid point division

(22, 13, 26, 1) (12, 27,33,15)

(22,13) (26,1) (12,27) (33,15)

22 13 26 1 12 27 33 15

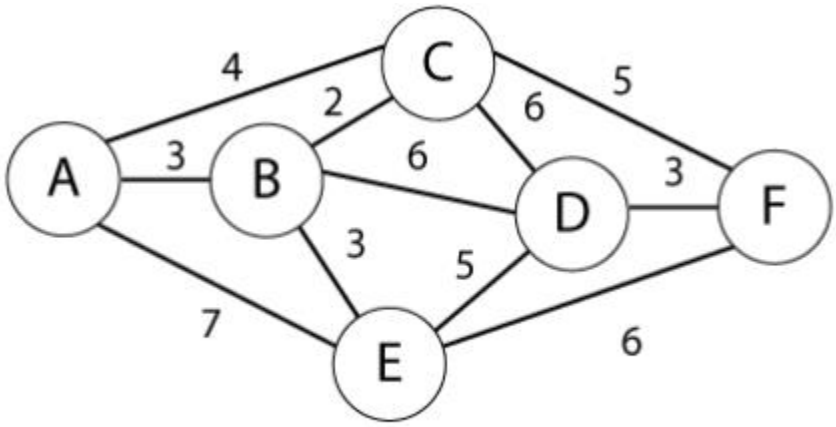
(13,22) (1,26) (12,27) (15,33)

(1, 13, 22, 26) (12, 15,27,33)

[**1,12,13,15,22,26,27,33]**

Sorted Array.

7. Kruskal's algorithm,



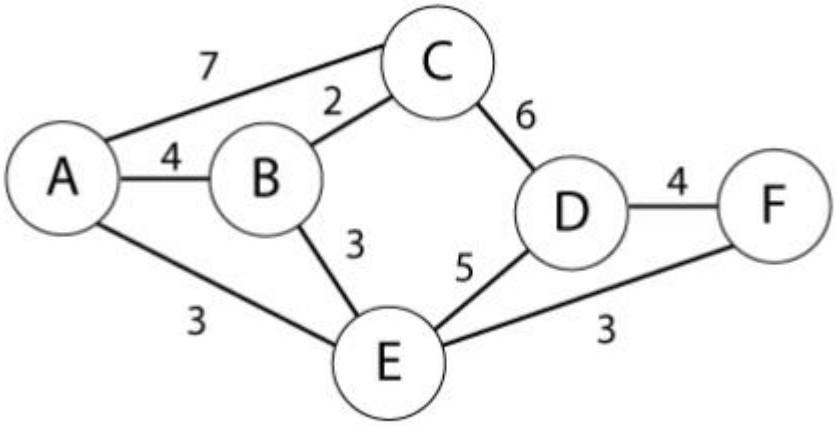
Sol:

1. Connect B-C(2)
2. Connect A-B(3)
3. Connect D-F(3)
4. Connect B-E(3)
5. Connect E-D(5)

Stop. MST formed(5 edges) 6 vertices

MST=[A-B, B-C, D-F,B-E, E-D]

8. Prim's algorithm



Sol:

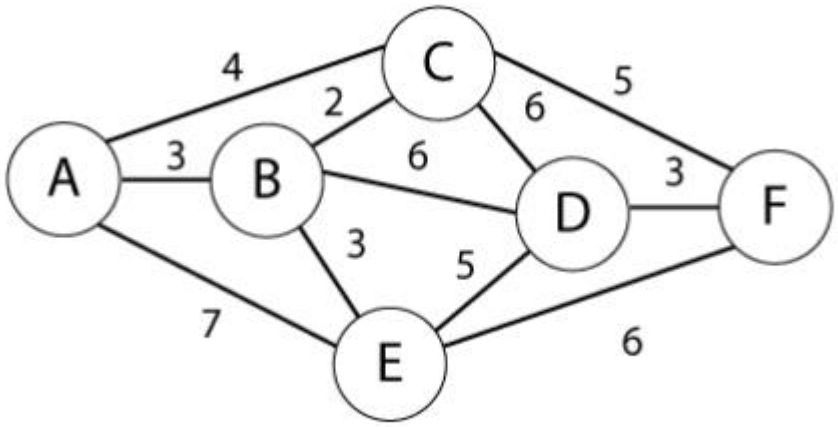
0. A S{A}

1. Choosing the lowest cut A-E(3) S={A,E}
2. E-F(3) or B-E(3) , choosing mincut E-F(3) S={A,E,F}
3. E-B (3) S={A,E,F,B}
4. B-C(2) S={A,E,F,B,C}
5. F-D(4) S={A,E,F,B,C,D}

DONE N-1 EDGES 5

MST={A-E,E-F,E-B,B-C,F-D}
Weight =3+3+3+2+4=15

9. Dijkstra's Algorithm

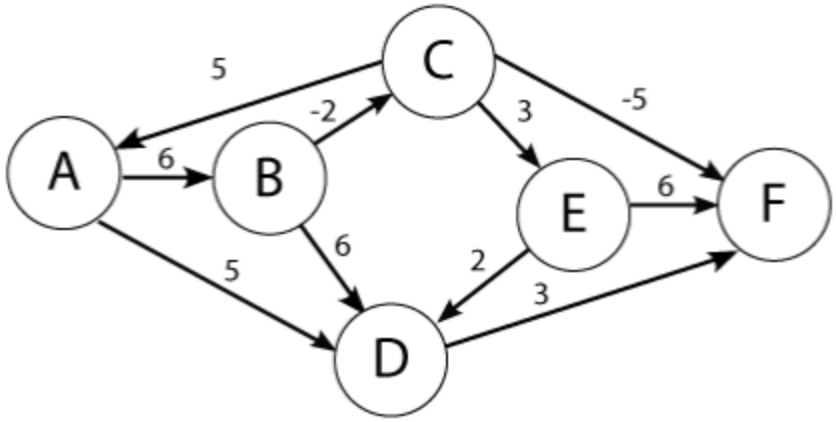


Sol: Source A

		A	B	C	D	E	F
1: A {A}	A	0	(3,A) **	(4,A)	INF	(7,A)	INF
2: B {A,B}	B	0	(3,A)	(4,A)**	(9, B)	(6,B)	INF
3: C {A, B,C}	C	0	(3,A)	(4,A)	(9,B)	(6,B)**	(9,C)
4:E{A,B,C,E}	E	0	(3,A)	(4,A)	(9,B)**	(6,B)	(9,C)
5: D {A,B,C,E,D}	D	0	(3,A)	(4,A)	(9,B)	(6,B)	(9,C)**
6:F { A,B,C,E,D,F}	F	0	(3,A)	(4,A)	(9,B)	(6,B)	**(9,C)**

So the Shortest Path would be **A->C->F COST(4+5)=9**

10 Bellman-Ford algorithm to find the shortest path from node A to F



6 vertices therefore 5 iteration

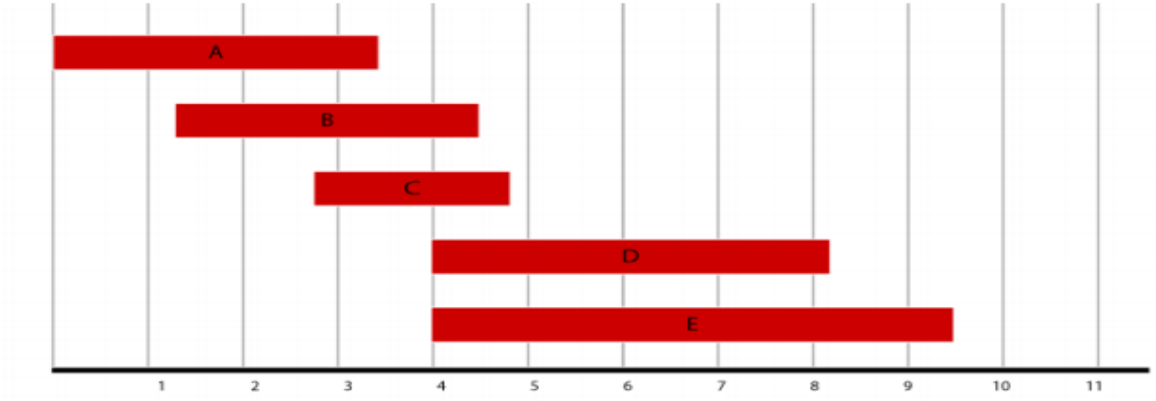
Edges	Weights
AB	6
AD	5
BC	-2
BD	6
CA	5
CE	3
CF	-5
DF	3
ED	2
EF	6

There will be a total of (n-1) iterations where 'n': number of vertices

Initial:	A (0)	B (INF)	C (INF)	D (INF)	E (INF)	F (INF)
Iteration 1:	A (0)	B (6)	C (4)	D (5)	E (7)	F (-1)
Iteration 2:	A (0)	B (6)	C (4)	D (5)	E (7)	F (-1)

Since there are no changes in both iterations – stop the algo.

11. Use dynamic programming. Show your work.



Interval	Value
A	2
B	3
C	2
D	3
E	2

Interval	Value	Previous	Max
A	2	NA	Max (0, 2) = 2
B	3	NA	Max (2, 3) = 3
C	2	NA	Max (3, 2) = 3
D	3	A	Max (3, 3+2) = 5
E	2	A	Max (5, 2+2) = 5

Interval	Trace	S
E	2+2 < 5	{}
D	3+2=5, jump to A	{D}
C	Jump to A	
B	jump to A	
A	2=2	{D, A}

S= {D, A}

2+_3=5

12: Use dynamic programming. Show your work. Solve Knapsack.

Item	Value	Weight
1	3	4
2	2	3
3	4	2
4	4	3

Sol:

Capacity of knapsack = 6
Algorithm: w[4, 3, 2, 3] and v[3, 2, 4, 4]:

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

Items 3, 4 are combined value of 8 in the knapsack

S={3,4}