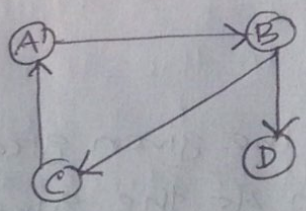
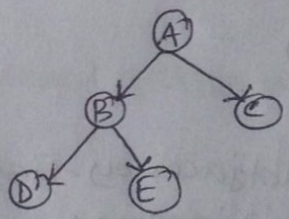
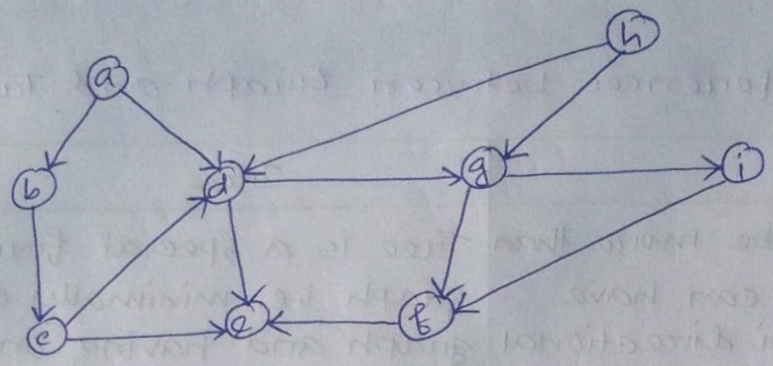
GRAPHS & TREES

What are the differences between Graph and Tree?

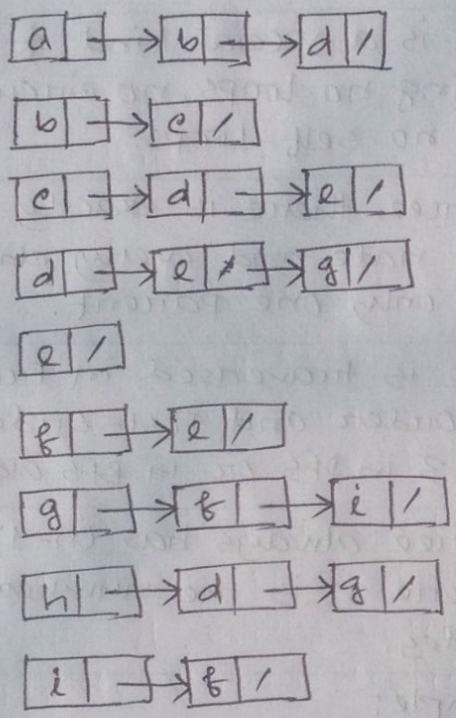
Graph	Tree
In graph, there can be more than one path i.e. graph can have uni-directional or bi-directional paths (edges) between nodes.	Tree is a special form of graph i.e. minimally connected graph and having only one path between any two nodes.
Graph can have loops, circuits as well as can have self-loops.	Tree is a special kind of graph having no loops, no circuits and no self-loops.
In graph, there is no concept of root node.	In tree, there is exactly one root node and every child has only one parent.
Graph is traversed by DFS, BFS algorithm.	Tree is traversed in Pre-Order, In-Order and Post-Order (all 3 in DFS or in BFS algorithm)
In graph, number of edges depends on graph.	A tree always has (n-1) edges where n is the number of nodes.
Example: 	Example: 

Nayeem Mahmood / 4AM / C161026

Show adjacency list and adjacency matrix representation of the following graph.



Soln:



(a)

	a	b	c	d	e	f	g	h	i
a	0	1	0	1	0	0	0	0	0
b	0	0	1	0	0	0	0	0	0
c	0	0	0	1	1	0	0	0	0
d	0	0	0	0	1	0	1	0	0
e	0	0	0	0	0	0	0	0	0
f	0	0	0	0	1	0	0	0	0
g	0	0	0	0	0	1	0	0	1
h	0	0	0	1	0	0	1	0	0
i	0	0	0	0	0	1	0	0	0

(b)

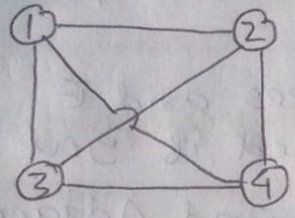
(a) An adjacency-list representation of the given graph
(b) An adjacency-matrix representation of the given graph

Nayeem Mahmood / 4AM / C161026

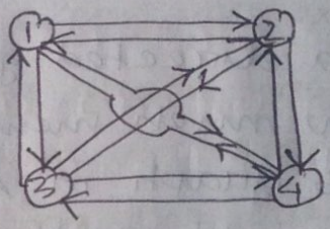
Complete Graph

A complete undirected graph is a simple undirected graph in which every pair of distinct nodes is connected by a unique edge.

A complete directed graph is a simple directed graph in which every pair of distinct nodes is connected by a pair of unique edges (one in each direction).



(a) Complete Undirected Graph with 4 nodes



(b) Complete Digraph with 4 nodes

Strongly Connected Graph

A digraph is strongly connected if there is a path between all pairs of ~~edges~~ nodes.

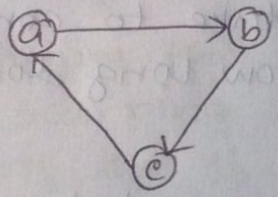
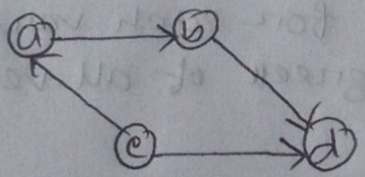


Fig: A strongly connected graph with 3 nodes.

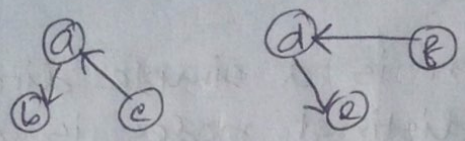
Weakly Connected Graph

A digraph is said to be weakly connected if there exists such a pair (u, v) of nodes where there doesn't exist any path from u to v.



Disjoint Graph

A graph is called disjoint if there doesn't exist any path between two or more components.



(A)

Figure (A) shows a disjoint graph.

Suppose a directed graph has V vertices and E edges. How much memory will be needed if you store this graph in Adjacency matrix and Adjacency list?

Ans: Required memory to store the graph

in Adj-Matrix = $\Theta(V^2)$

in Adj-List = $\Theta(V+E)$

Given an adjacency-list representation of a directed graph, how long does it take to compute the out-degree of every vertex? How long does it take to compute the in-degrees?

Ans: Given an adjacency-list representation Adj of a digraph, the out-degree of a vertex "u" is equal to the length of Adj[u]. Thus the time to compute the out-degree of one vertex is $\Theta(|Adj(u)|)$ and for all vertices is $\Theta(V+E)$.

The in-degree of a vertex "u" is equal to the number of times it appears in all the lists in Adj. If we search all the lists for each vertex, the time to compute the in-degrees of all vertices is $\Theta(VE)$.

Alternatively, we can allocate an array "IN" of size |V| and initialize its entries to Zero. Then we only need to scan the lists in Adj once, incrementing IN[u] when we see "u" in the lists. The values in IN will be the in-degrees of every vertex. This can be done in $\theta(V+E)$ time with $\theta(V)$ additional storage.

⊞ Given an adjacency matrix representation of a digraph, how long does it take to compute the out-degree and in-degree of every vertex?

Ans: The adjacency matrix Adj of any graph has $\theta(V^2)$ entries, ~~negative~~ regardless of the number of edges in the graph. For a digraph, computing the out-degree of a vertex "u" is equivalent to scanning the row corresponding to u in Adj and summing the 1's, so that computing the out-degree of every vertex is equivalent to scanning all entries of Adj. Thus, the time required is $\theta(V)$ for one vertex, and $\theta(V^2)$ for all vertices.

Similarly, computing the in-degree of a vertex "u" is equivalent to scanning the corresponding to u in Adj and summing the 1's. Thus the time required is also $\theta(V)$ for one vertex, and $\theta(V^2)$ for all vertices.

Write down the code segment for finding following information from a given un-weighted directed graph with V vertices which is already implemented using Adjacency matrix named graph.

1) Find the in-degree of each vertex.
2) Find the out-degree of each vertex.
3) Find whether there is any loop in the graph.
4) Find how many pendant vertex are there.
5) Find the transpose of the graph.
6) Find the list of vertices which have more outward edges than inward edges.
7) Make the 3rd vertex disconnected from all other vertices.
8) Check whether the graph is Sparse or Dense graph. [Assume that when number of edges is more than 80% of maximum possible edges, then the graph is dense, sparse otherwise.]

1)

```
int IN_DEG[V+1];
void computeInDeg(int V)
{
    int i,j;
    for(i=1; i<=V; i++) {
        for(j=1; j<=V; j++) {
            if (graph[j][i] == 1) {
                IN_DEG[i]++;
            }
        }
    }
}
```

Nayeem Mahmood / 4AM / C161026

2)

```
int OUT_DEG[V+1];
void computeOutDeg(int V)
{
    int i,j;
    for(i=1; i<=V; i++) {
        for(j=1; j<=V; j++) {
            if(graph[i][j]==1) {
                OUT_DEG[i]++;
            }
        }
    }
}
```

3)

```
bool checkLoop(int V) // returns true if there exists a loop
{
    int i;
    for(i=1; i<=V; i++) {
        if(graph[i][i]==1) {
            return true;
        }
    }
    return false;
}
```

Nayeem Mahmood / 4AM / C161026

4)

```
int IN_DEG[V+1];
int OUT_DEG[V+1];

void computeDeg(int V) // comput in-deg and out-deg
{
    int i, j;
    for(i=1; i<=V; i++) {
        for(j=1; j<=V; j++) {
            if(graph[j][i] == 1) {
                IN_DEG[i]++;
            }
            if(graph[i][j] == 1) {
                OUT_DEG[i]++;
            }
        }
    }
}
int countPendant(int V) // returns the number of pendants
{
    int i, count = 0;
    for(i=1; i<=V; i++) {
        if(IN_DEG[i] == 1 && OUT_DEG[i] == 0) {
            count++;
        }
    }
    return count;
}
```

5)

```
int graphT[V+1][V+1];

void createTranspose(int V)
{
    int i, j;
    for(i = 1; i <= V; i++) {
        for(j = 1; j <= V; j++) {
            graphT[i][j] = graph[j][i];
        }
    }
}
```

6)

```
int IN_DEG[V+1];
int OUT_DEG[V+1];
vector<int> moreOutward;

void computeDeg(int V)
{
    int i, j;
    for(i = 1; i <= V; i++) {
        for(j = 1; j <= V; j++) {
            if(graph[j][i] == 1) {
                IN_DEG[i]++;
            }
            if(graph[i][j] == 1) {
                OUT_DEG[i]++;
            }
        }
    }
}
```

Nayeem Mahmood / 4AM / C161026

```
void createList (int V)
{
    int i;
    for(i=1; i<=V; i++) {
        if (IN_DEG[i] < OUT_DEG[i]) {
            moreOutward.push_back(i);
        }
    }
    if (moreOutward.empty()) {
        printf("There is no such vertex.\n");
    }
    else {
        for(i=0; i< moreOutward.size(); i++) {
            printf("%d\t", moreOutward[i]);
        }
    }
}
```

7)

```
void disconnect3rdVertex (int V)
{
    int i, j;
    for(i=1; i<=V; i++) {
        graph[i][3] = 0;
        graph[3][i] = 0;
    }
}
```

8)

```
bool checkSparseOrDense(int V)
// returns true if Sparse, false if Dense
{
    int i, j, totalEdges, totalPossibleEdges;
    double edgeRate;

    totalEdges = 0;
    totalPossibleEdges = V * (V-1);

    for(i=1; i<=V; i++) {
        for(j=1; j<=V; j++) {
            if(graph[i][j] == 1) {
                totalEdges++;
            }
        }
    }
    edgeRate = (totalEdges * 100.0) / totalPossibleEdges;
    if(edgeRate > 80.0) {
        return false;
    }
    return true;
}
```

☑ Source codes are available at http://bit.ly/KhaboAlgoBamboo

GRAPH TRAVERSAL ALGORITHMS

Breadth-First Search (BFS)

BFS (G, s)

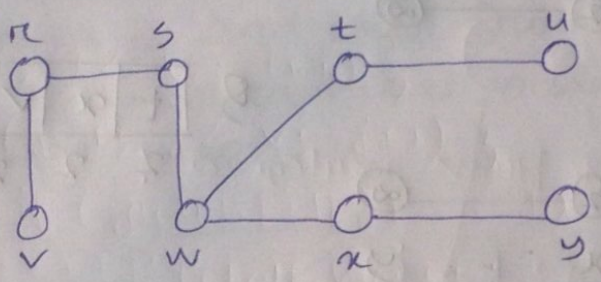
```
1  for each vertex u ∈ G.V - {s}
2        u.color = WHITE
3        u.d = ∞
4        u.π = NIL
5  s.color = GRAY
6  s.d = 0
7  s.π = NIL
8  Q = ∅
9  ENQUEUE (Q, s)
10 while Q ≠ ∅
11       u = DEQUEUE (Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE (Q, v)
18       u.color = BLACK
```

Nayeem Mahmood / 4AM / C161026

What are the information carried by d and π in BFS?

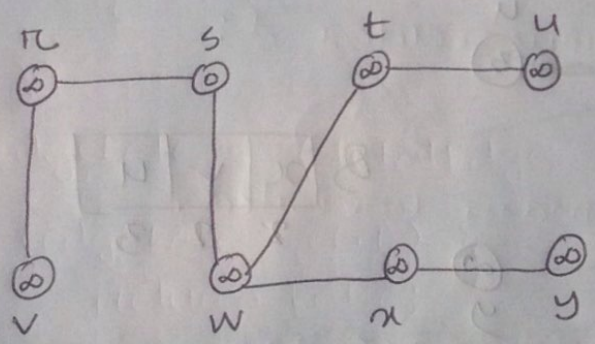
Ans: In Bfs algorithm, every vertex is maintained with 3 attributes. Let u be a vertex in a graph G. Then u.d holds the distance from the source vertex s to the vertex u.
u.π holds the predecessor of the vertex u.

Show the d values that result from running BFS on the following undirected graph with source s. Also show the status of Queue after each step.



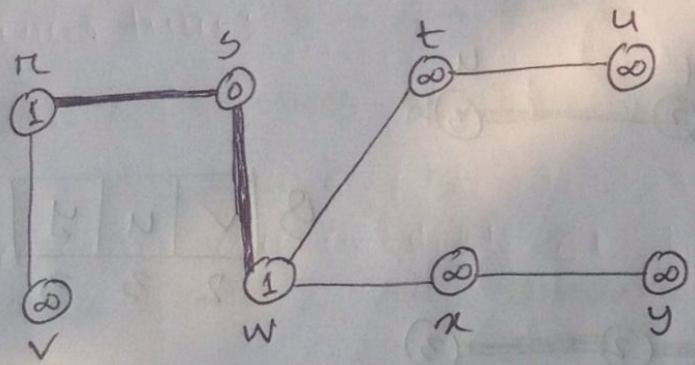
Soln:

(a)



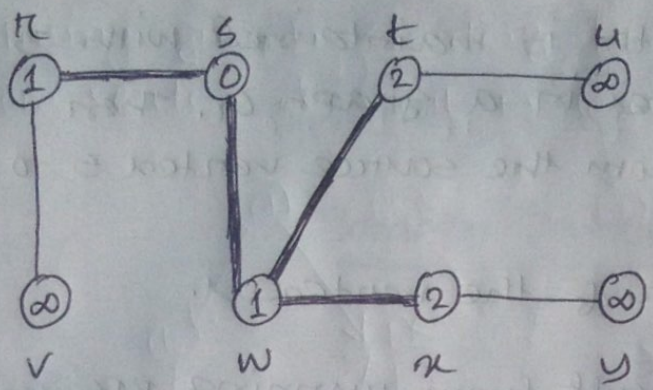
Q | s |
0

(b)



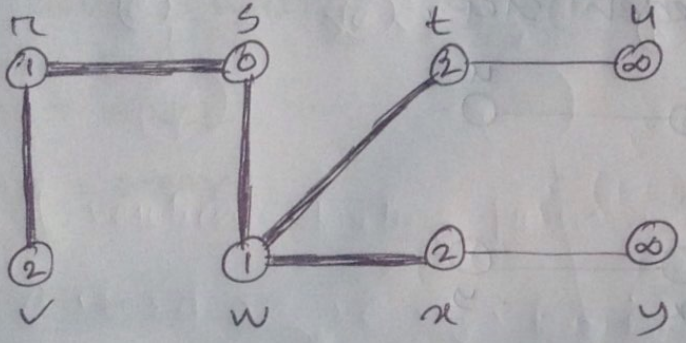
Q | w | r |
1 1

(c)



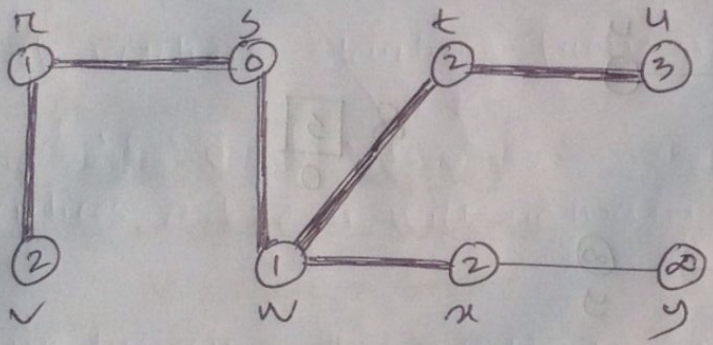
Q | π | t | x |
1 2 2

(d)



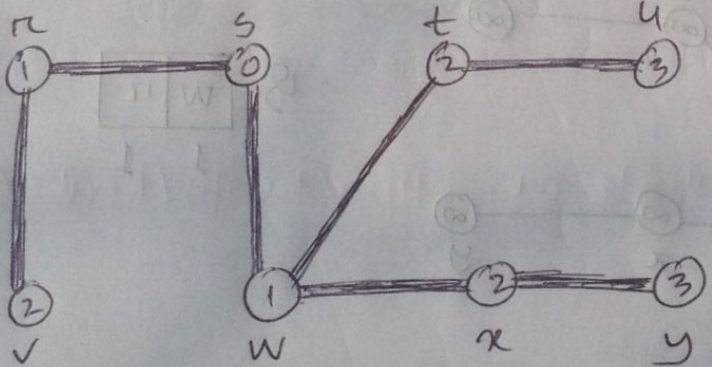
Q | t | x | v |
2 2 2

(e)

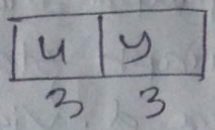


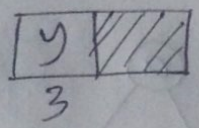
Q | x | v | u |
2 2 3

(f)



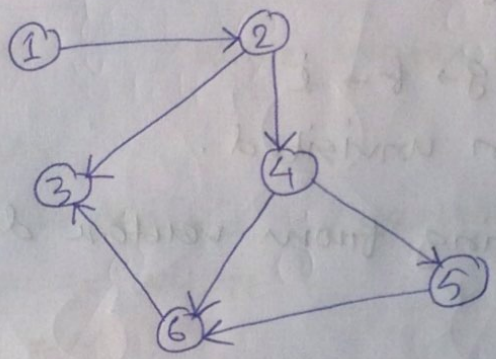
Q | v | u | y |
2 3 3

(g) [Same figure from (f)] & 

(h) [Same figure from (f)] & 

(i) [Same figure from (f)] & $\varnothing$

Show in which order you will visit the vertices if you traverse the following graph using BFS starting from vertex 1 and also from vertex 4. Visit the vertices in lexicographical order. Don't show the intermediate steps.



Soln:

Order of vertices starting from 1:

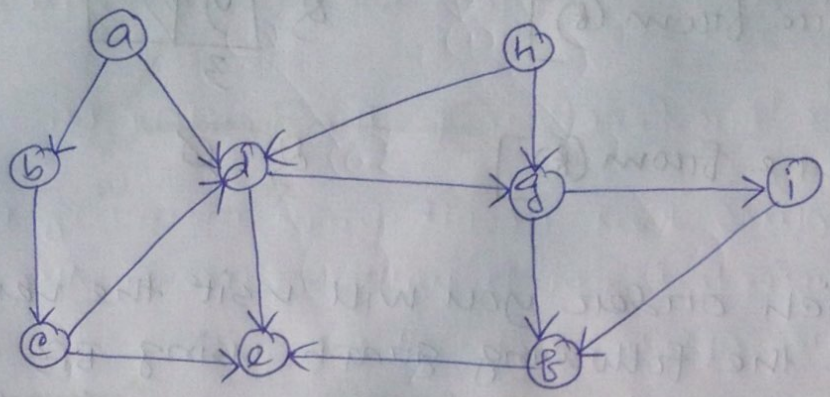
$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

Order of vertices starting from 4:

$4 \rightarrow 5 \rightarrow 6 \rightarrow 3$

Vertex 1 and 2 will remain unvisited.

Show the graph traversal order using BFS for the following graph starting from vertex a and also from vertex d. Visit in lexicographic order.



Ans:

Order of vertices starting from vertex a:

$a \to b \to d \to c \to e \to g \to f \to i$

vertex h will remain unvisited.

Order of vertices starting from vertex d:

$d \to e \to g \to f \to i$

vertices a, b, c, h will remain unvisited.

*** If we traverse the graph using DFS ***

Order of vertices:

$a \to b \to c \to d \to e \to g \to f \to i \to h$

DEPTH FIRST SEARCH (DFS)

DFS (G)

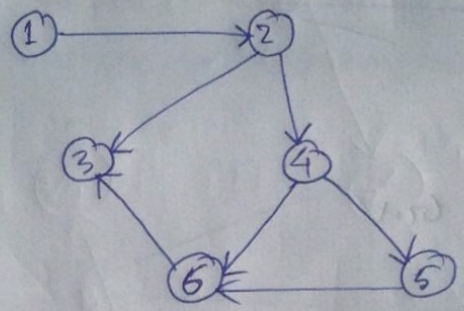
```
1  for each vertex u ∈ G.V
2        u.color = WHITE
3        u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6        if u.color == WHITE
7              DFS-VISIT (G, u)
```

DFS-VISIT (G, u)

```
1  time = time + 1      // while vertex u has just been discovered
2  u.d = time
3  u.color = GRAY
4  for each v ∈ G.Adj[u]    // explore edge (u,v)
5        if v.color == WHITE
6              v.π = u
7              DFS-VISIT (G, v)
8  u.color = BLACK      // blacken u; it is finished
9  time = time + 1
10 u.f = time
```

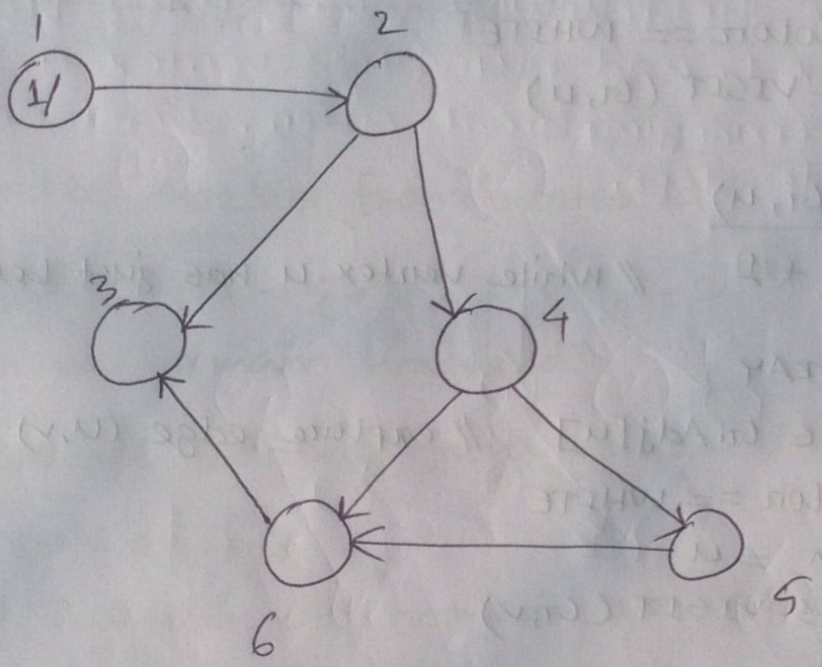
Nayeem Mahmood / 4AM / C161026

Illustrate the steps of DFS for the following graph:

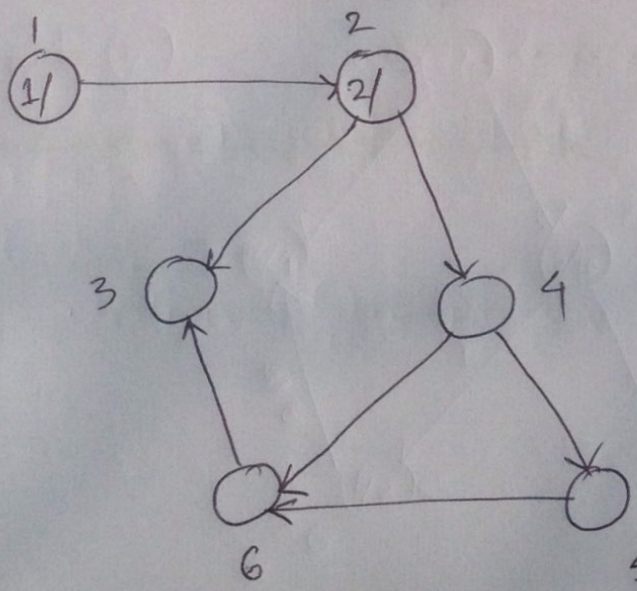


Soln:

(a)

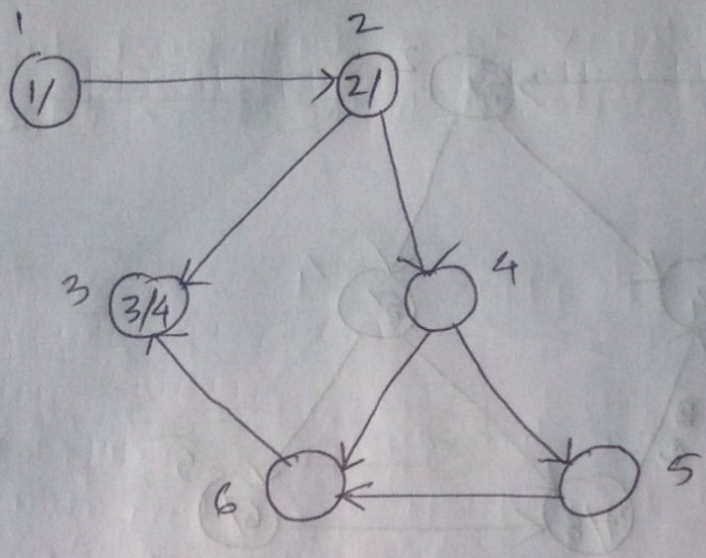


(b)

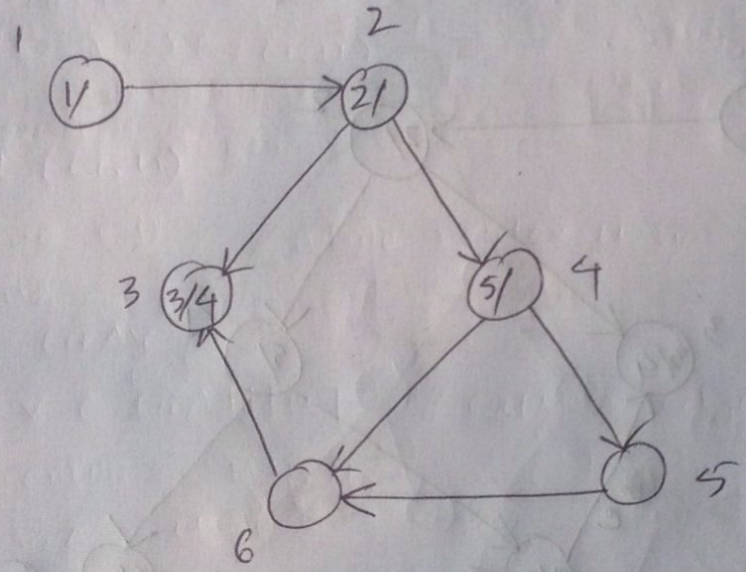


Nayeem Mahmood / 4AM / C161026

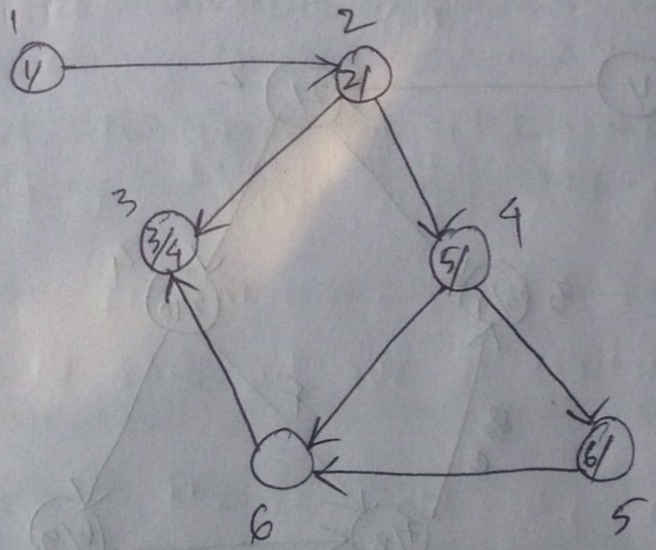
(c)



(d)

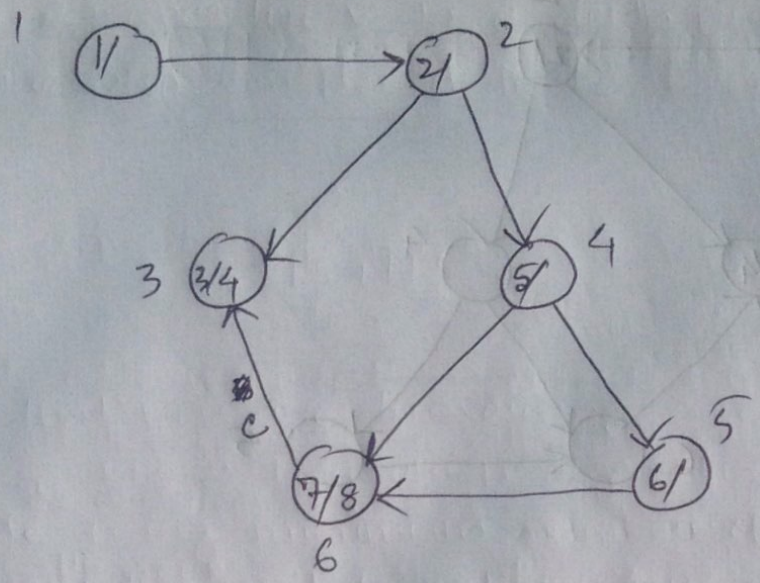


(e)

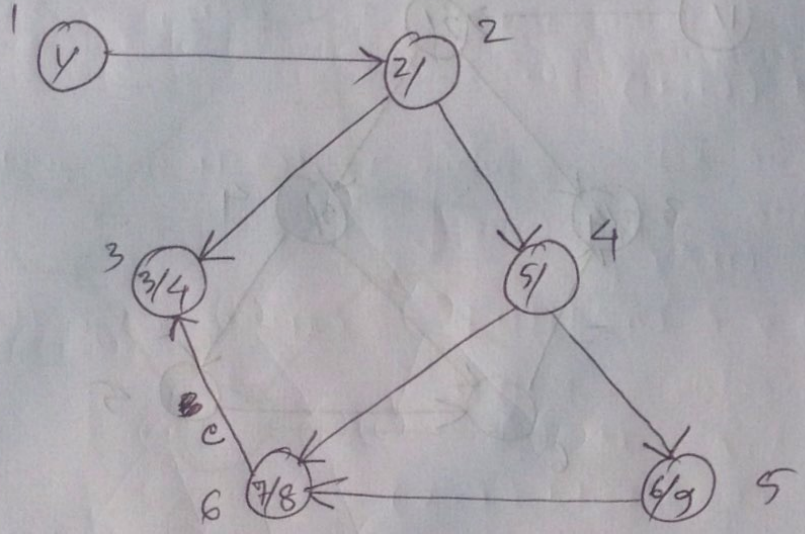


Nayeem Mahmood / 4AM / C161026

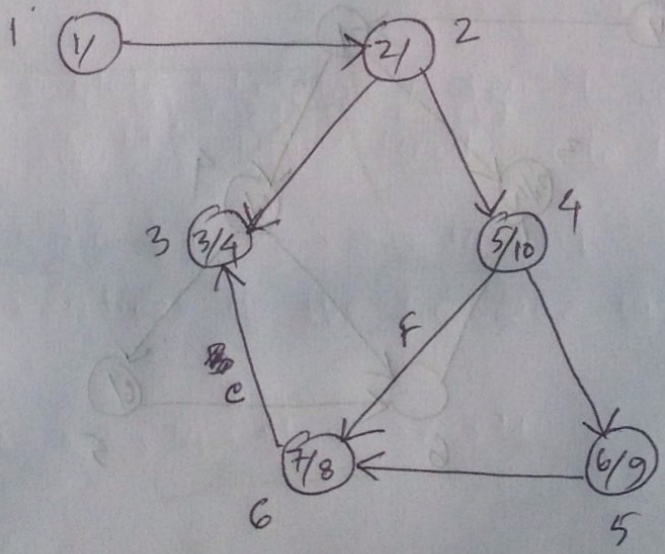
(f)



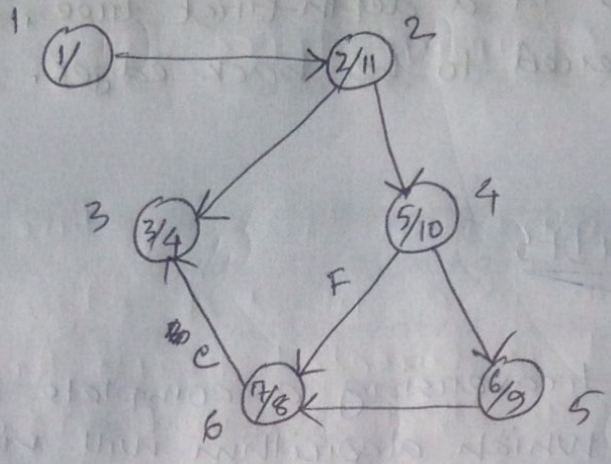
(g)



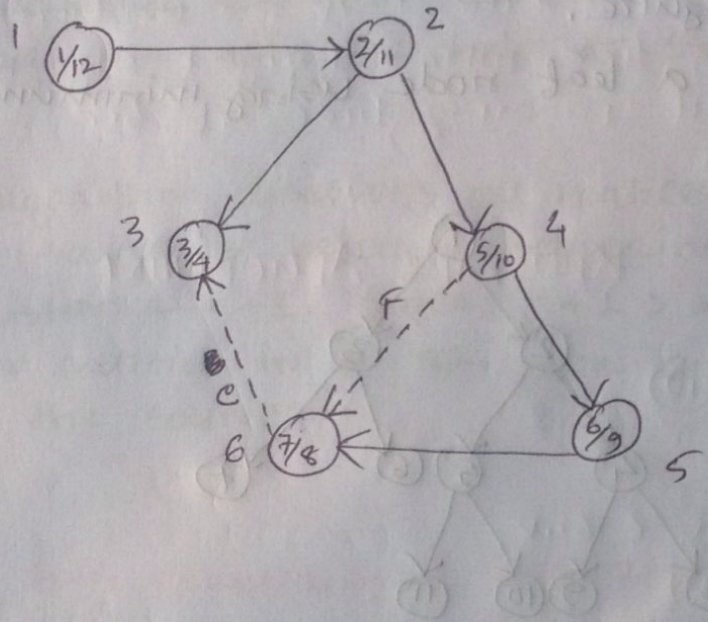
(h)



(i)



(ii)



Q You are asked to write DFS algorithm without using recursion. What data structure should you use?

Ans: I will use that data structure which works in the way of LIFO (Last In First Out) i.e. the stack.

Q How can you detect the presence of a cycle in a graph using DFS?

Ans: DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that connects a

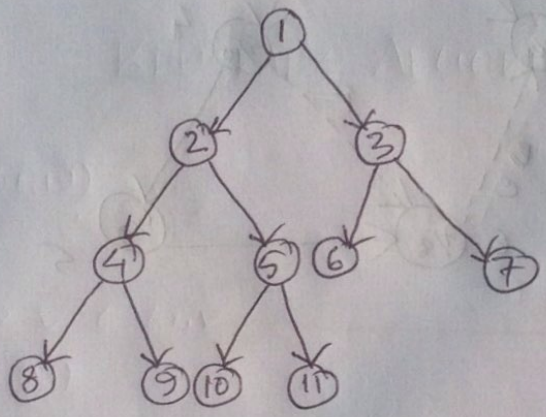
vertex u to an ancestor v in a depth-first tree. self-loops are also considered to be back edges.

BFS + DFS

Suppose that you are traversing a complete binary tree using BFS & DFS. Which algorithm will visit a leaf node using minimum number of steps if you start from the root? Justify your answer using appropriate figure.

Ans: DFS will reach a leaf node using minimum steps.

Justification:



(G)

Let G be a complete BT. The order of vertices if we traverse G using BFS from root node 1 :

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11

The order of vertices if we traverse G using DFS from root node 1:

1 → 2 → 4 → 8 → 9 → 5 → 10 → 11 → 3 → 6 → 7

This shows us that using DFS, a leaf node can be reached using minimum number of steps.

State whether the following statements are true or false and give proper reasoning behind your answer.

i) DFS can be implemented without recursion.

ii) Last visited vertices in a BFS tree are the farthest from the source.

Ans:

i) True.

Any recursive algorithm can be transformed into iterative one with the help of stack data structure.

ii) True.

BFS algorithm discovers all vertices at distance k from source s before discovering any vertices at distance k+1. Since k+1 > k, hence last visited vertices in a BFS tree are the farthest from the source.

*** Running time complexity of both BFS & DFS is $\theta(V+E)$.

Nayeem Mahmood / 4AM / C161026

MINIMUM SPANNING TREES

Write the definition of minimum spanning tree.

A minimum spanning tree is a subset of edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycle and with the minimum possible total edge weight.

What is bottleneck spanning tree?

A bottleneck spanning tree T of an undirected graph G is a spanning tree whose largest edge weight is minimum over all spanning trees of G.

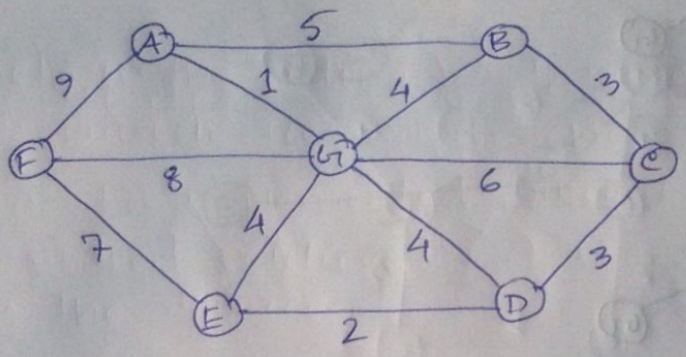
KRUSKAL'S ALGORITHM

MST-Kruskal (G,w)

```
1  A = ∅
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u,v) ∈ G.E, taken in nondecreasing order
   by weight
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u,v)}
8          UNION(u,v)
9  return A
```

Nayeem Mahmood / 4AM / C161026

Construct MST from the following graph using Kruskal's algorithm.



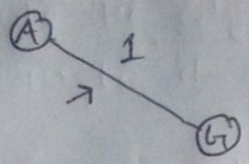
Soln:

Lets sort the edges in nondecreasing order first.

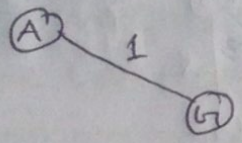
Weight	—	Terminals
1	——	A-G
2	——	D-E
3	——	B-C
3	——	C-D
4	——	B-G
4	——	D-G
4	——	E-G
5	——	A-B
6	——	C-G
7	——	E-F
8	——	F-G
9	——	A-F

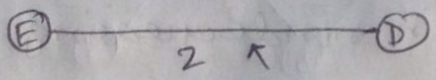
Now we'll construct MST by connecting nodes using edges that cost minimum.

(a)

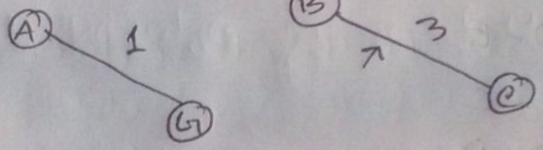


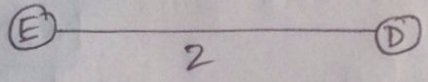
(b)



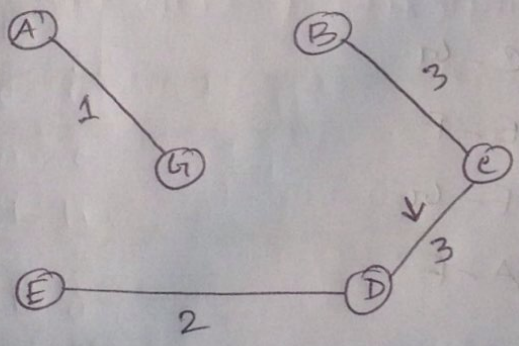


(c)

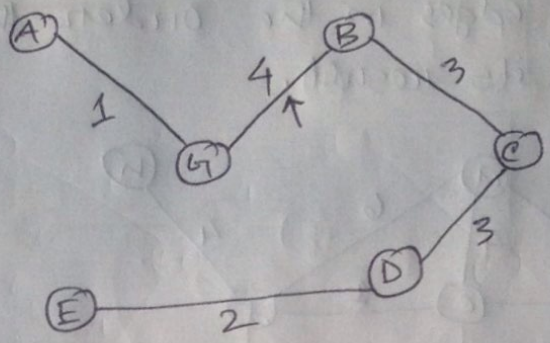




(d)



(e)



(f)

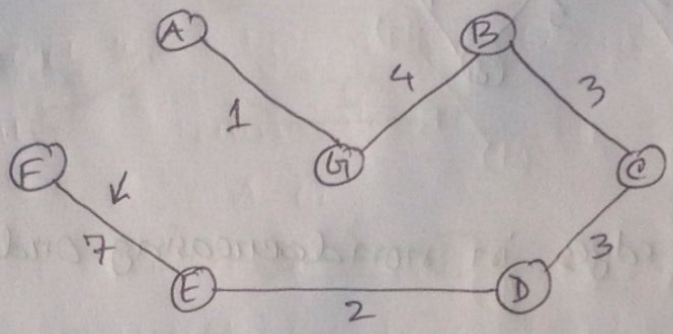
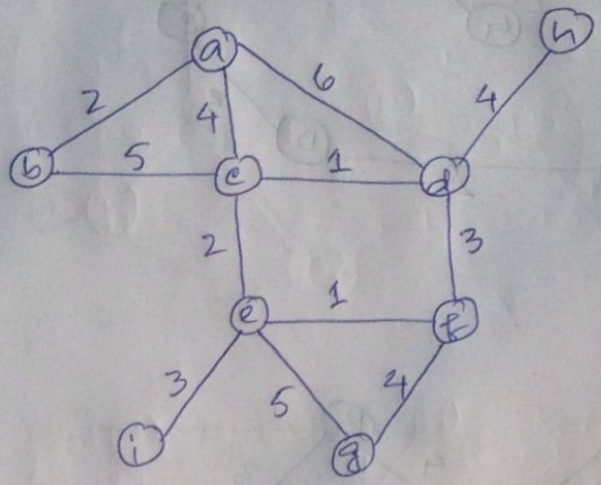


Figure (f) shows one of the possible MSTs which costs 20 in total.

Nayeem Mahmood / 4AM / C161026

Construct MST from the following graph using Kruskal's algorithm. Write the edges in the order that the algorithm would add them to its result.



Soln:

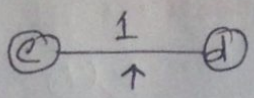
Lets first sort out the edges in non decreasing order of their weights.

Weight	–	Terminals
1	—	c–d
1	—	e–f
2	—	a–b
2	—	c–e
3	—	d–f
3	—	e–i
4	—	a–c
4	—	d–h
4	—	f–g
5	—	b–c
5	—	e–g
6	—	a–d

Nayeem Mahmood / 4AM / C161026

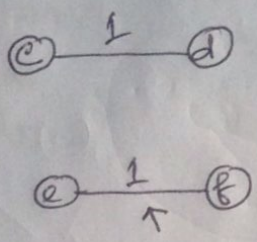
Order of Taking Edges

(1)



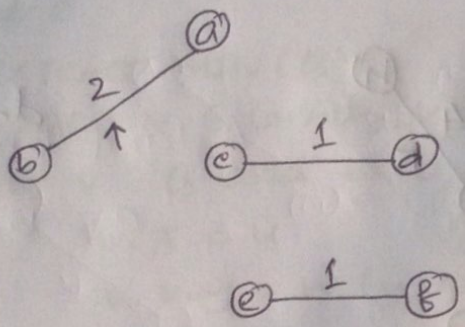
c – d

(2)



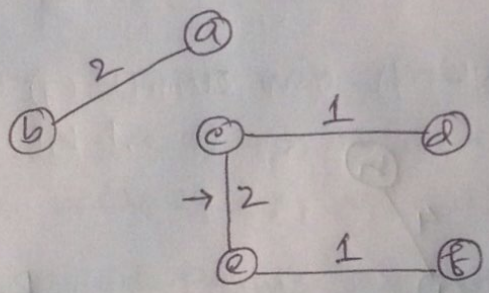
e – f

(3)



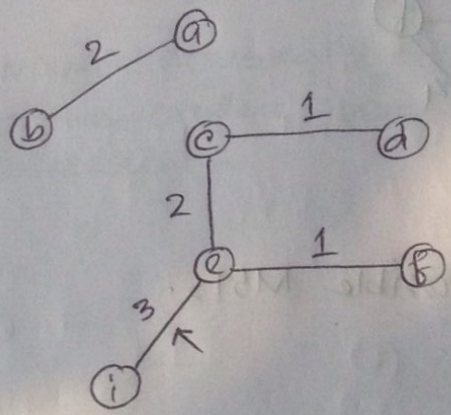
a – b

(4)



c – e

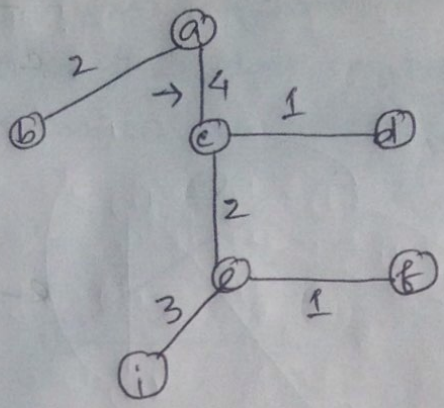
(5)



e – i

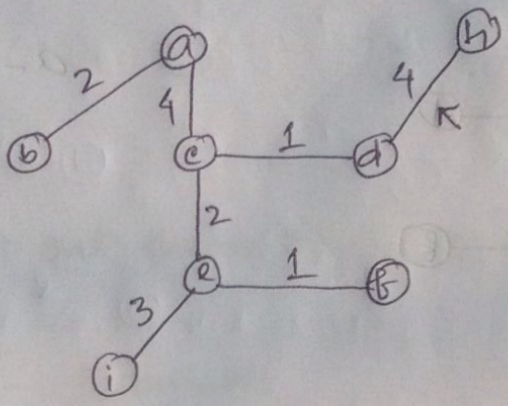
Nayeem Mahmood / 4AM / C161026

(6)



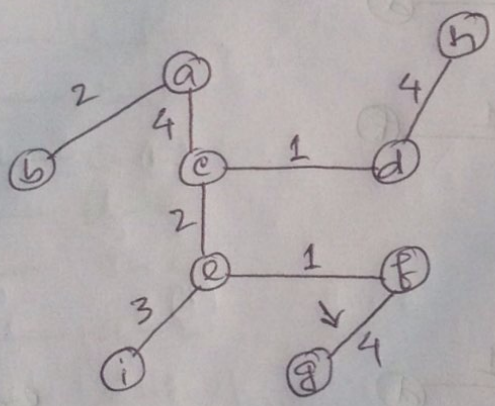
a - c

(7)



d - h

(8)



f - g

Fig (8) shows one of the possible MSTs.

PRIM'S ALGORITHM

MST-PRIM (G, w, r)

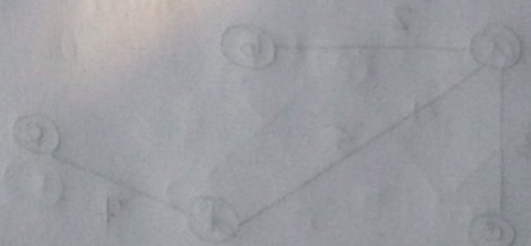
```
1   for each u ∈ G.V
2       u.key = ∞
3       u.π = NIL
4   r.key = 0
5   Q = G.V
6   while Q ≠ ∅
7       u = EXTRACT-MIN(Q)
8       for each v ∈ G.Adj[u]
9           if v ∈ Q and w(u,v) < v.key
10              v.π = u
11              v.key = w(u,v)
```

Consider the graph with the following weights:

$w(a,b)=2$, $w(a,c)=3$, $w(a,d)=5$, $w(a,e)=11$, $w(b,c)=4$, $w(b,d)=6$, $w(b,e)=9$, $w(c,d)=5$, $w(c,e)=8$, $w(d,e)=7$

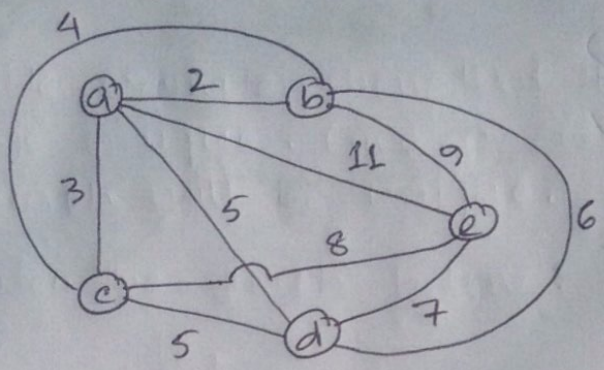
i) What MST would Prim's algorithm produce starting at vertex b? Write the edges in the order that the algorithm would add them to its result.

ii) What MST would Kruskal's algorithm produce? Write the edges in the order that the algorithm would add them to its result.



Nayeem Mahmood / 4AM / C161026

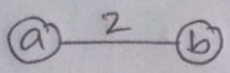
The given graph is :



i)

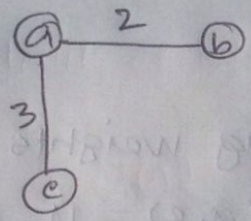
Order of Taking Edges

(1)



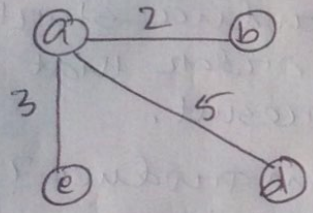
a — b

(2)



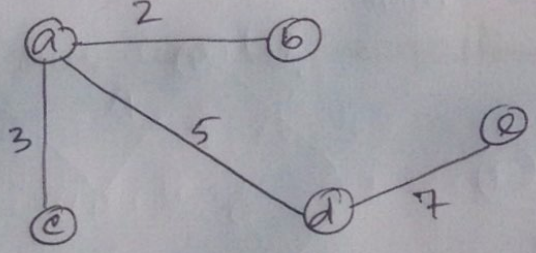
a — c

(3)



a — d

(4)



d — e

Fig (4) shows one of the possible MSTs.

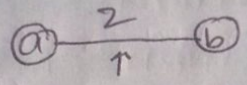
Nayeem Mahmood / 4AM / C161026

ii)

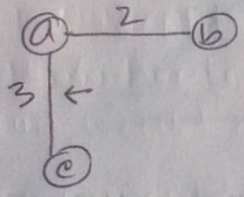
Weight – Terminals

2 — a-b
3 — a-c
4 — b-c
5 — a-d
5 — c-d
6 — b-d
7 — d-e
8 — c-e
9 — b-e
11 — a-e

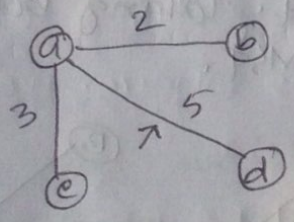
Order of taking edges

(1) 

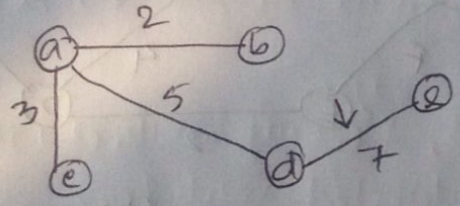
a-b

(2) 

a-c

(3) 

a-d

(4) 

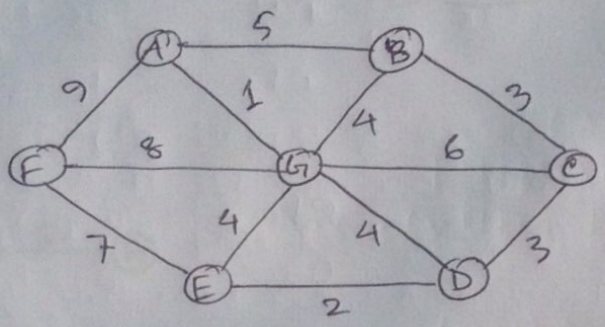
d-e

Nayeem Mahmood / 4AM / C161026

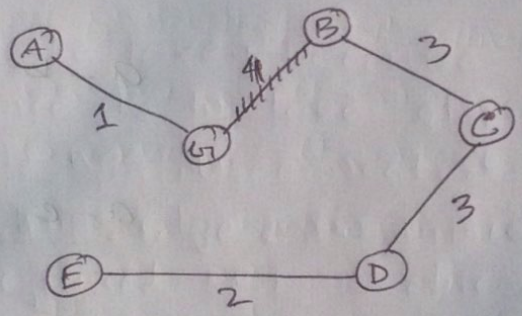
Fig (4) shows the final MST constructed using Kruskal's algorithm.

Using a suitable example, show that the MST of a graph is not always unique. Give a condition for which MST of a graph will be unique.

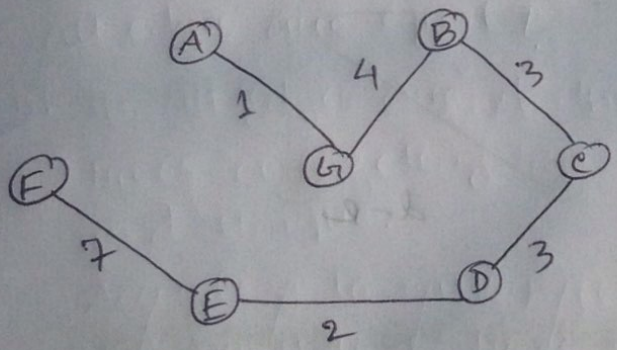
Ans: Let us consider the graph below:



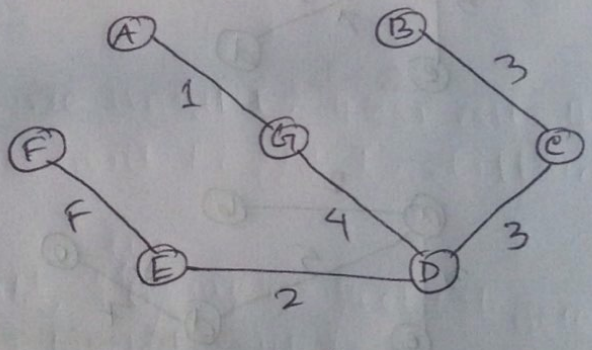
Now lets start construct MST using Kruskal's algorithm.



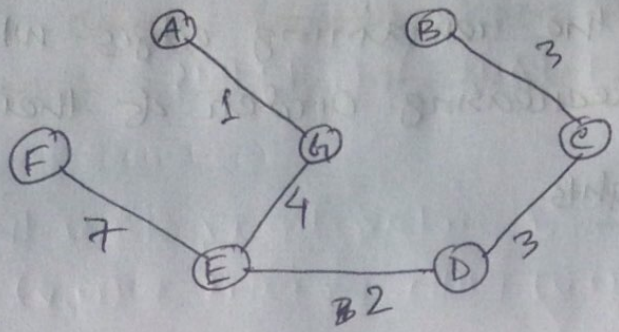
At this point, we have three choices. Either we can take B-G or D-G or E-G, both of them costs 4. Taking both of them in turn gives us three unique MST.



(a)



(b)



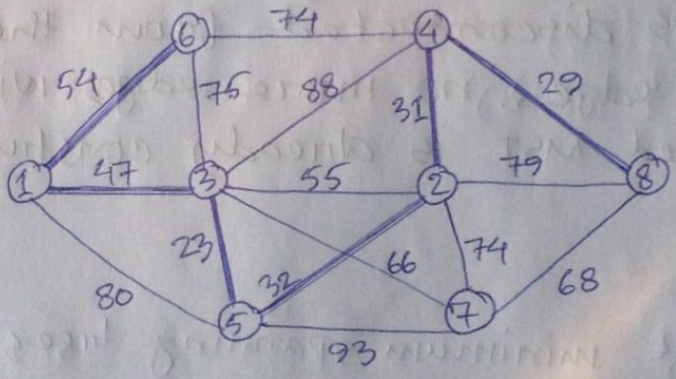
(e)

All 3 MSTs are unique but cost same i.e. 20.

Condition for the uniqueness of MST : A sufficient condition for the uniqueness of MST is

"If all the edges have distinct weight, then there will be only one, unique MST."

Kruskal's algorithm selects the shaded edges for finding MST of the graph below. Which edge will be selected in the next iteration? Justify your choice.



Ans: The edge which costs 66 and connects node 3 & 7 will be selected in the next iteration.

Nayeem Mahmood / 4AM / C161026

Justification: If we sort the remaining edges which are not taken yet (in nondecreasing order of their weights);

	Terminals		Weights
1.	2 - 3	—	55
2.	3 - 7	—	66
3.	7 - 8	—	68
4.	4 - 6	—	74
5.	2 - 7	—	74
6.	3 - 6	—	75
7.	2 - 8	—	79
8.	1 - 5	—	80
9.	3 - 4	—	88
10.	5 - 7	—	93

Edge (1) cannot be taken because it creates a cycle. Then the choice is to take edge (2) which connects node 7 which was disconnected from the tree. After taking this edge, no more edges will be taken because, the required MST is already constructed by taking (3-7) edge.

How many different minimum spanning trees can be formed from a graph of n vertices where weight of all edges are equal?

Ans: If all edges are of the same weight, then all the trees connecting all nodes will be MST, different and unique.

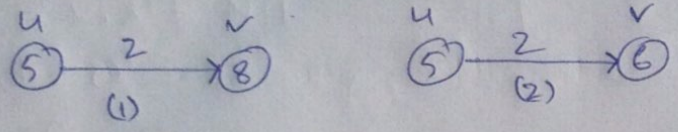
~~For~~ Nayeem Mahmood / 4AM / C161026

According to cayley's formula, "For every positive integer n, the number of tree on n-labeled vertices is n^{n-2}."

Nayeem Mahmood / 4AM / C161026

SHORTEST PATH ALGORITHMS

What do you know about relaxation? Show the result of RELAX for an edge (u,v) with weight w(u,v) for the graphs below:



Ans: The process of relaxing an edge (u,v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating v.d and v.π. A relaxation step may decrease the value of the shortest path estimate v.d and update v's predecessor attribute v.π. The following code performs a relaxation step on edge (u,v) in O(1) time:

RELAX (u,v, w)

```
1  if v.d > u.d + w(u,v)
2      v.d = u.d + w(u,v)
3      v.π = u
```

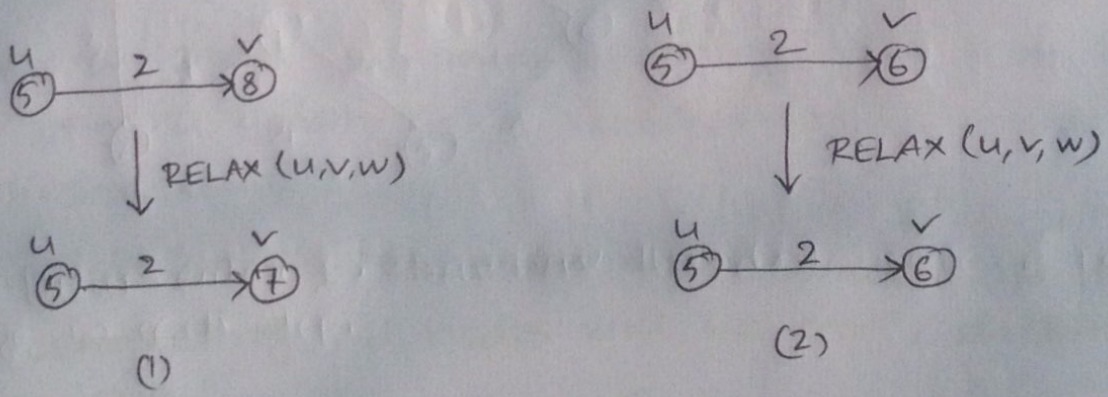


Fig: Result after relaxation

Prove that the subpaths of a shortest path are also shortest paths.

Proof: Given a weighted, digraph $G = (V, E)$ with weight function $w: E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j.

If we decompose path p into $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, then we have that $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$. Now let us assume that there is a path p'_{ij} from v_i to v_j with weight $w(p'_{ij}) < w(p_{ij})$, then $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ is a path from v_0 to v_k whose weight $w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$ is less than $w(p)$, which contradicts the assumption that p is a shortest path from v_0 to v_k.

What is a negative weight cycle?

Ans: A negative weight cycle is a cycle with weights that sum to a negative number.

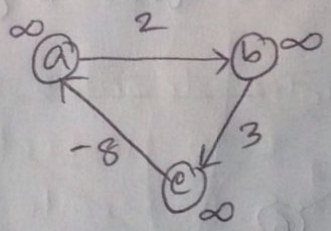


Fig: A negative weight cycle

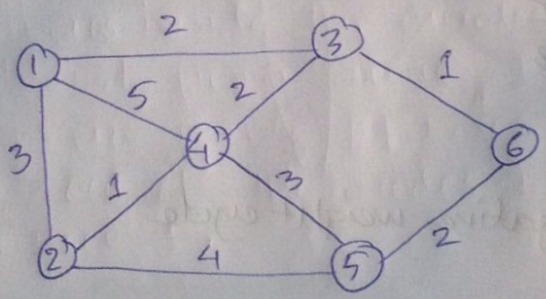
DIJKSTRA'S ALGORITHM

DIJKSTRA (G, W, S)

```
1   for each vertex v ∈ G.V
2       v.d = ∞
3       v.π = NIL
4   s.d = 0
5   S = ∅
6   Q = G.V
7   while Q ≠ ∅
8       u = EXTRACT-MIN(Q)
9       S = S ∪ {u}
10      for each vertex v ∈ G.Adj[u]
11          if v.d > u.d + w(u,v)
12              v.d = u.d + w(u,v)
13              v.π = u
```

Illustrate the operation of Dijkstra algorithm for finding the shortest path for the following graphs:

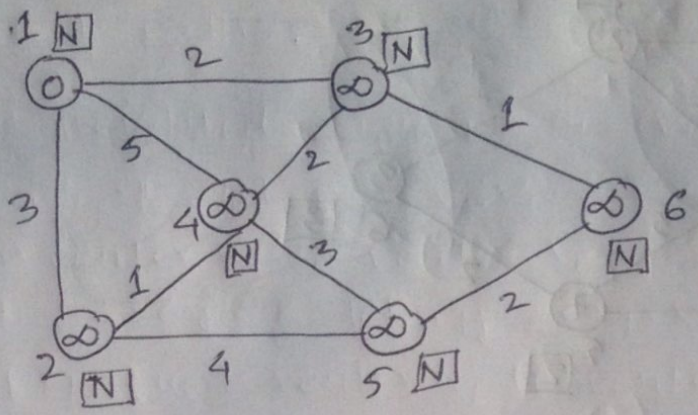
(1)

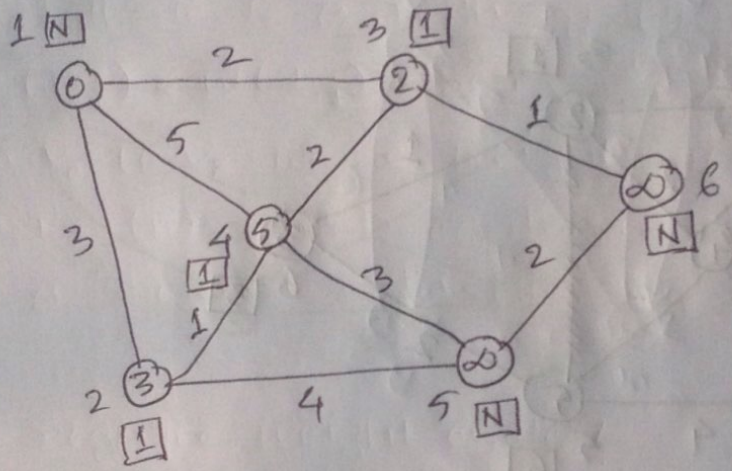


Start from vertex 1, show how the distance and parent change in each step.

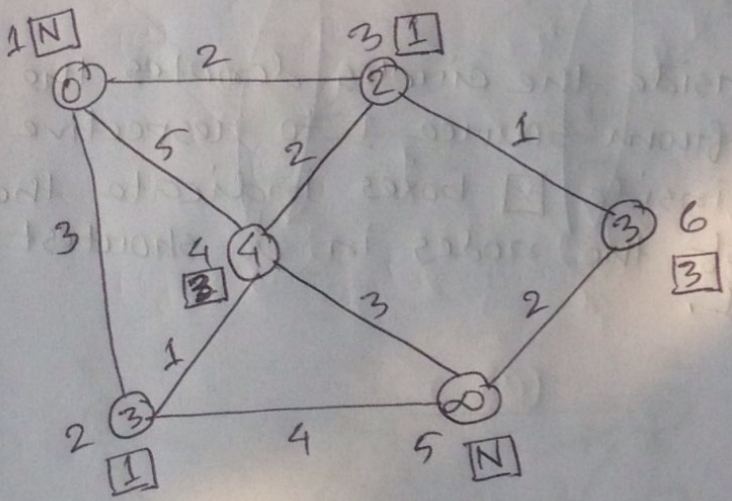
Nayeem Mahmood / 4AM / C161026

Soln:

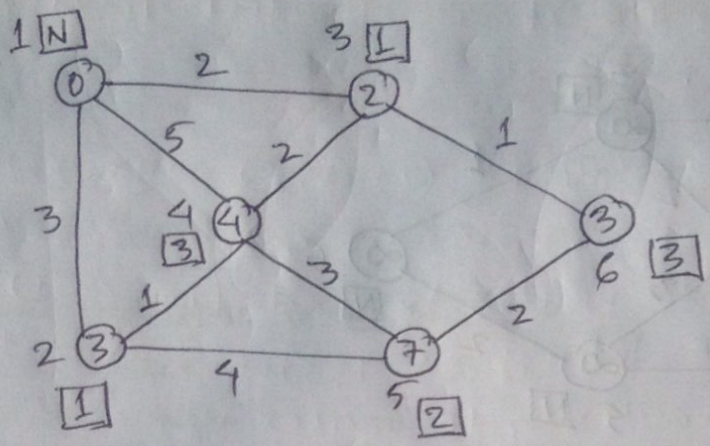




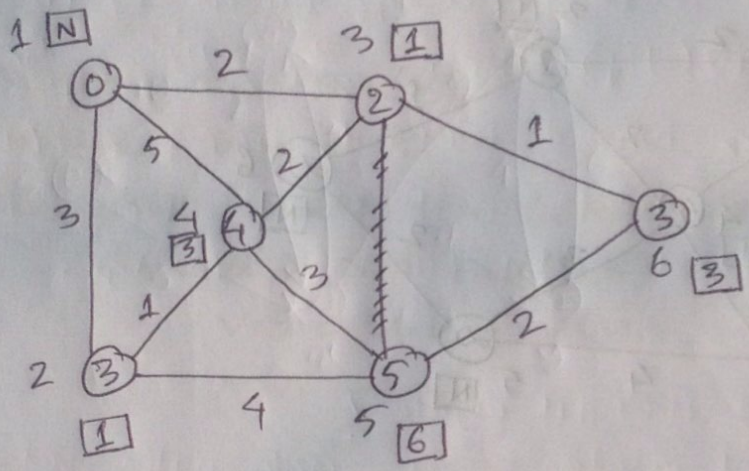
(a)



(b)



(c)

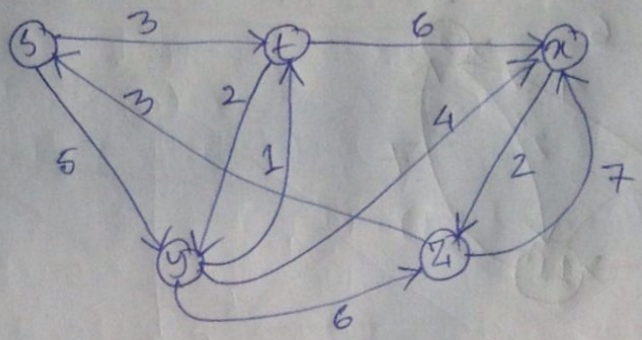


(d)

The numbers inside the circles denotes the minimal cost from source 1 to respective nodes, the numbers inside [x] boxes indicate the predecessor of the nodes in a shortest path from source 1.

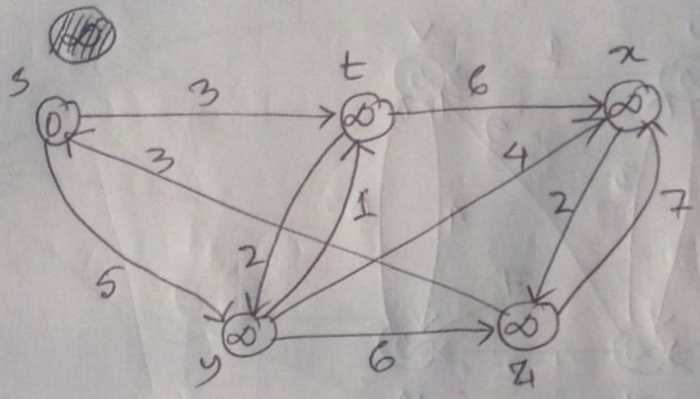
Nayeem Mahmood / 4AM / C161026

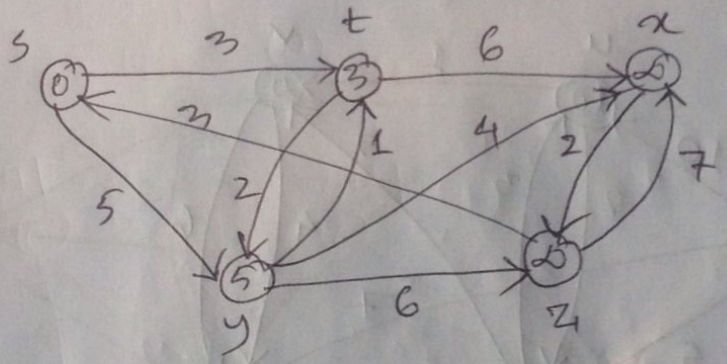
(2)



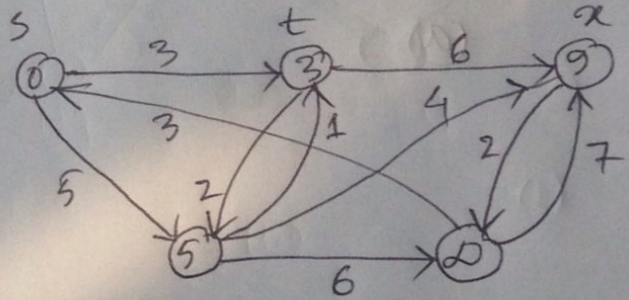
Consider s as source.

Soln:



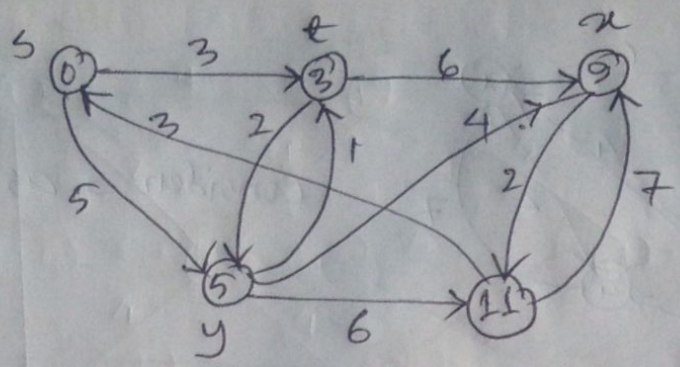


(a)



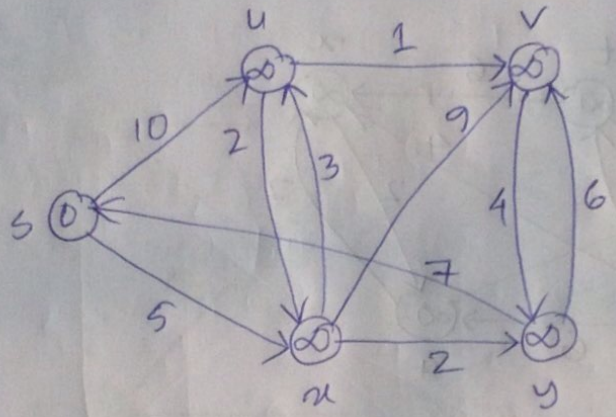
(b)

Nayeem Mahmood / 4AM / C161026

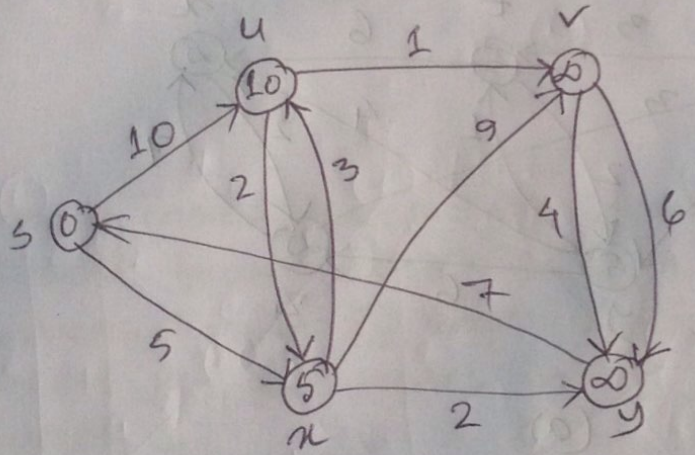


(c)

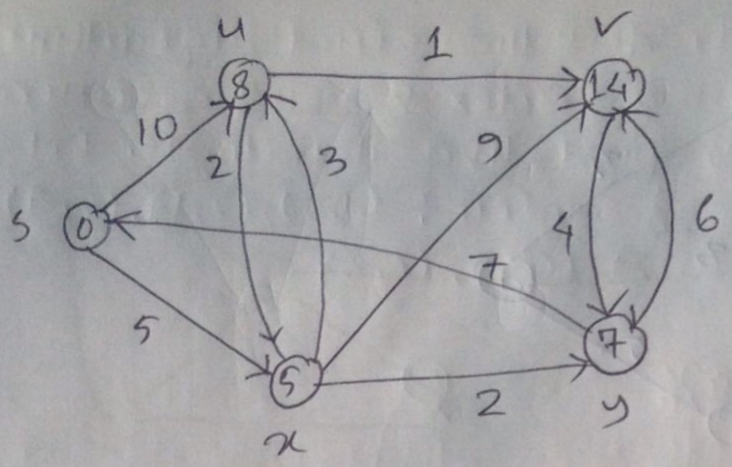
(3)



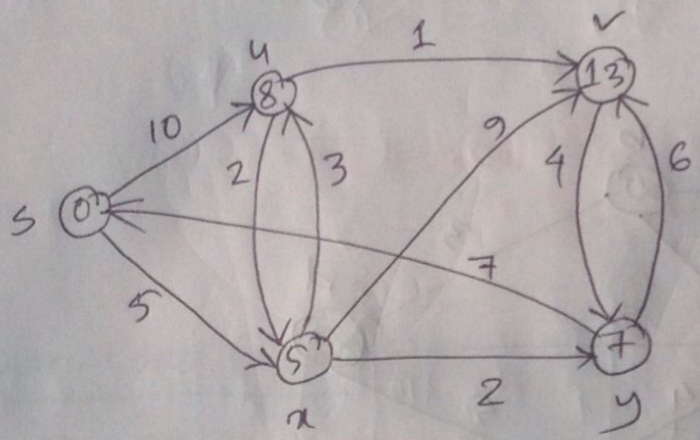
Soln:



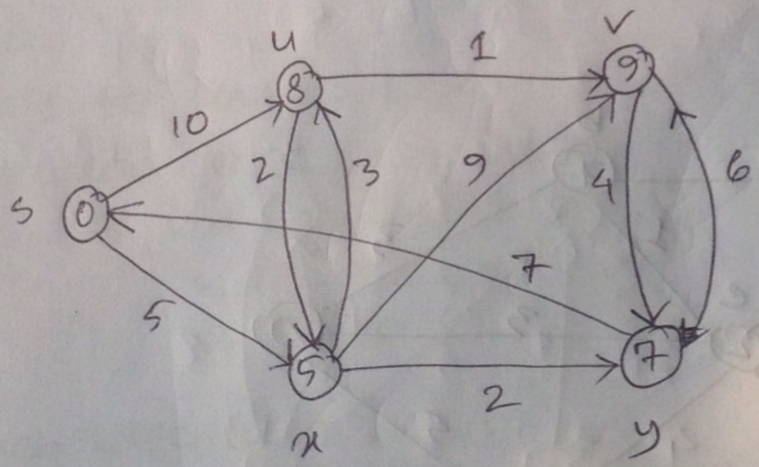
(a)



(b)



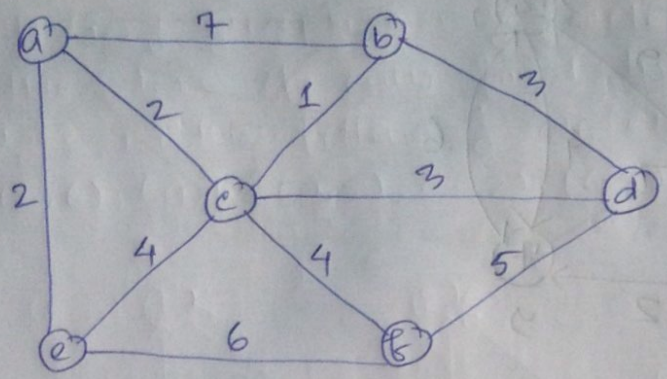
(c)



(d)

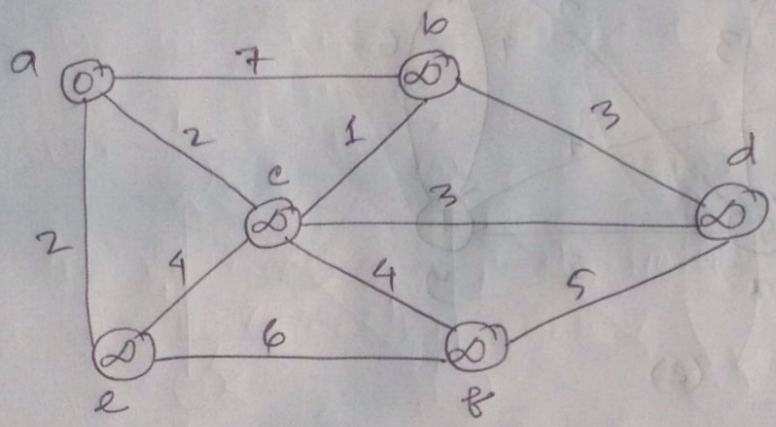
Nayeem Mahmood / 4AM / C161026

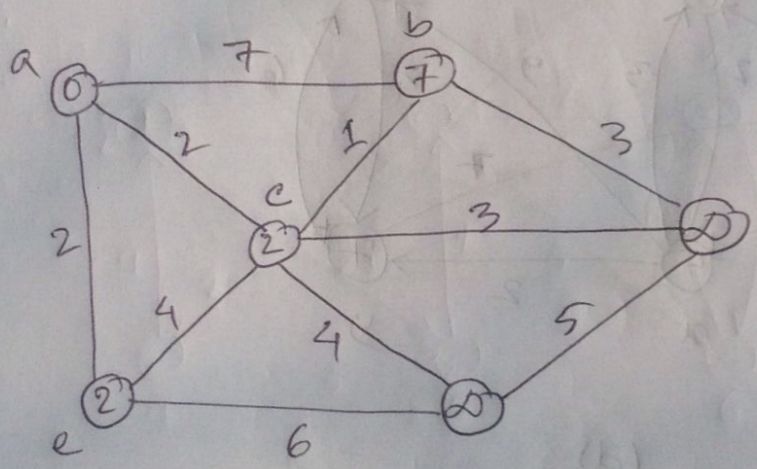
(4)



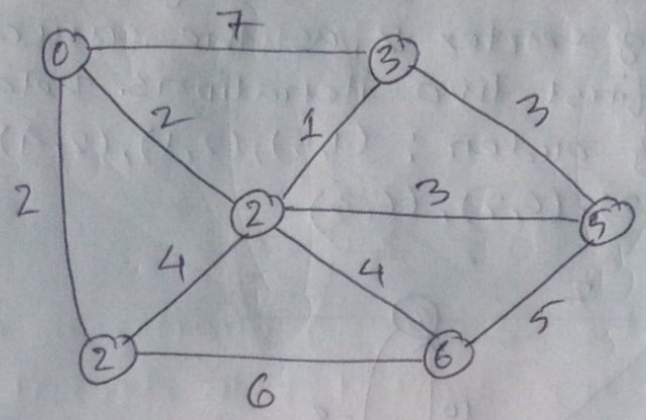
Consider a as source.

Soln:





(1)



12)

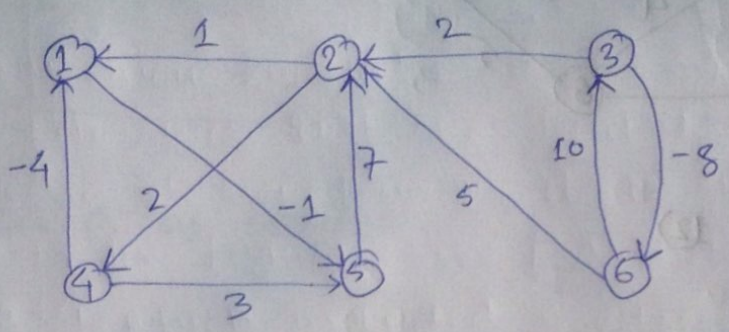
BELLMAN-FORD ALGORITHM

Bellman-Ford (G, w, s)

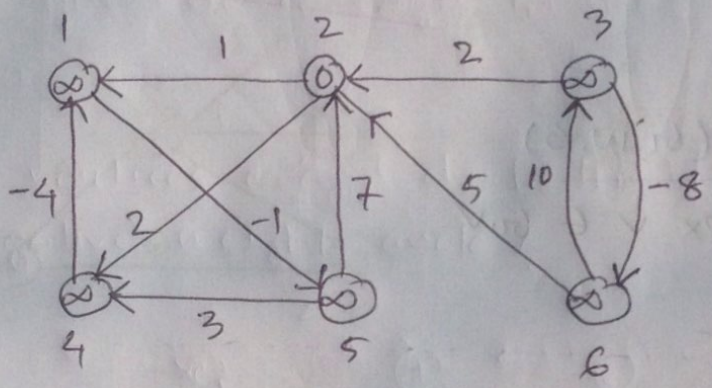
```
1   for each vertex v ∈ G.V
2       v.d = ∞
3       v.π = NIL
4   s.d = 0
5   for i = 1 to |G.V| - 1
6       for each edge (u,v) ∈ G.E
7           if v.d > u.d + w(u,v)
8               v.d = u.d + w(u,v)
9               v.π = u

10  for each edge (u,v) ∈ G.E
11      if v.d > u.d + w(u,v)
12          return FALSE

13  return TRUE
```

Nayeem Mahmood / 4AM / C161026

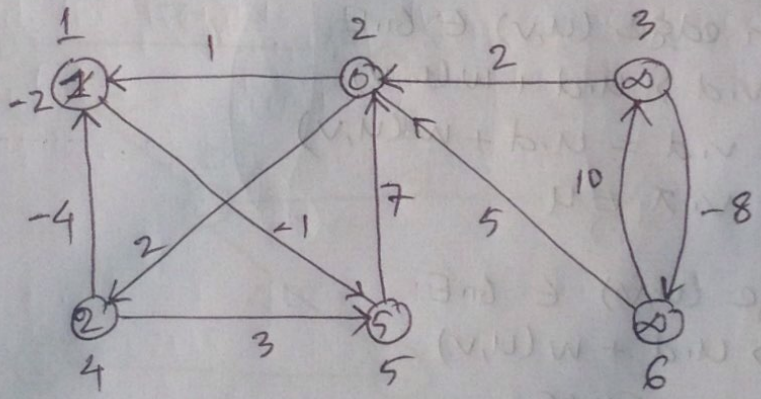
4. Illustrate the operation of Bellman-Ford algorithm on the following graph using vertex 2 as the source. Show the steps of only first two iterations. Relax the edges in the following order: (1,5), (2,1), (2,4), (3,2), (3,6), (4,1), (4,5), (5,2), (6,2), (6,3)



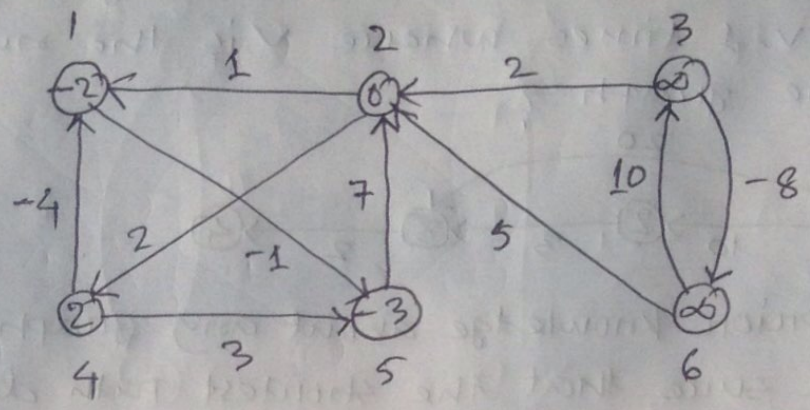
Soln:



Iteration 1:

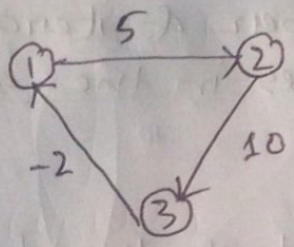


Iteration 2:



Give a simple example of a directed graph with negative weight edge but not with any negative-weight cycle.

Ans:



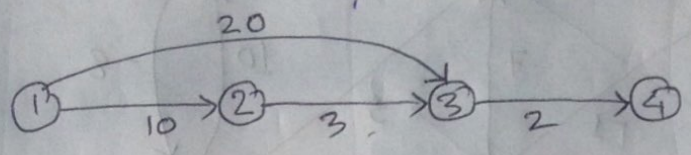
Write down the differences between Dijkstra & Bellman-Ford algorithm.

Dijkstra	Bellman-Ford
This algorithm greedily selects the minimum-weight node that has not yet been processed & performs this relaxation process on all of its outgoing edges.	It simply relaxes all the edges, and does this $\|V\|-1$ times, where $\|V\|$ is the number of vertices.
It can't handle negative-weight cycle.	It can handle negative-weight cycle.
A typical binary heap priority queue implementation gives us time complexity of $O((\|E\|+\|V\|)\log\|V\|)$.	Complexity of this algorithm is $O(\|V\|\|E\|)$.

Nayeem Mahmood / 4AM / C161026

Why the outermost loop of Bellman-Ford's algorithm should run at least V-1 times where V is the number of vertices of the graph?

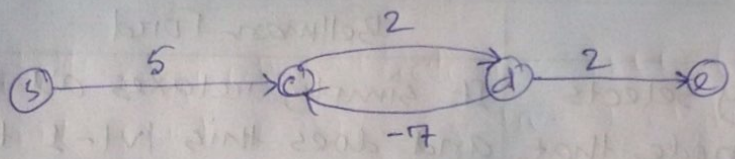
Ans;



Without any prior knowledge about any graph, we can only say for sure that the shortest path doesn't have cycles, and therefore is of length at most V-1 as shown in the figure above [A shortest path from node 1 to 4 which included 3 edges].

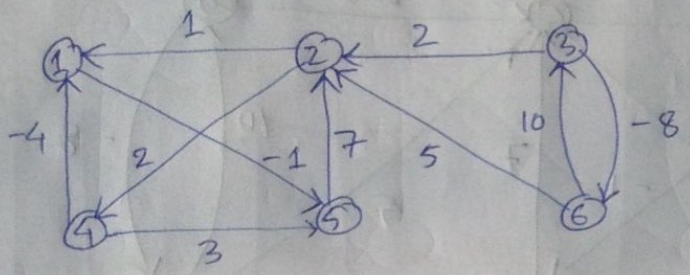
Hence the required number of iterations is the maximum number of edges in the shortest path i.e. V-1.

How do the vertices c and d of the following graph form a negative-weight cycle?

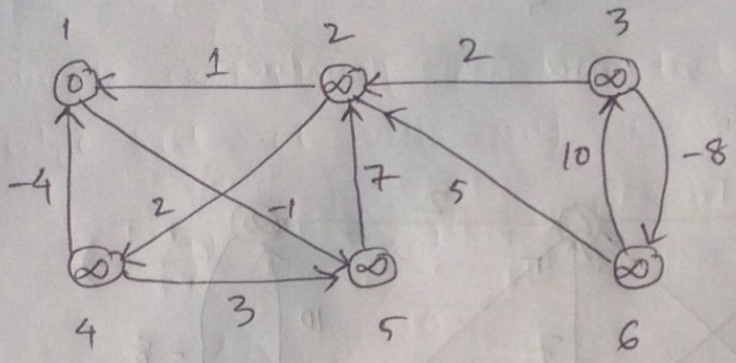


Ans: If we add the weights of the concerned edges, we find 2+(-7) = -5 which tells us that this a negative weight cycle.

Illustrate Bellman-Ford algo on the graph below. Use vertex 1 as the source.

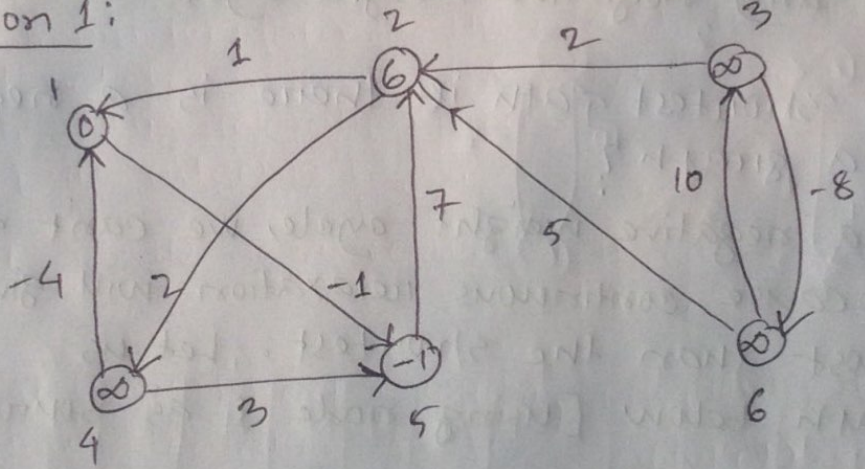


Soln:

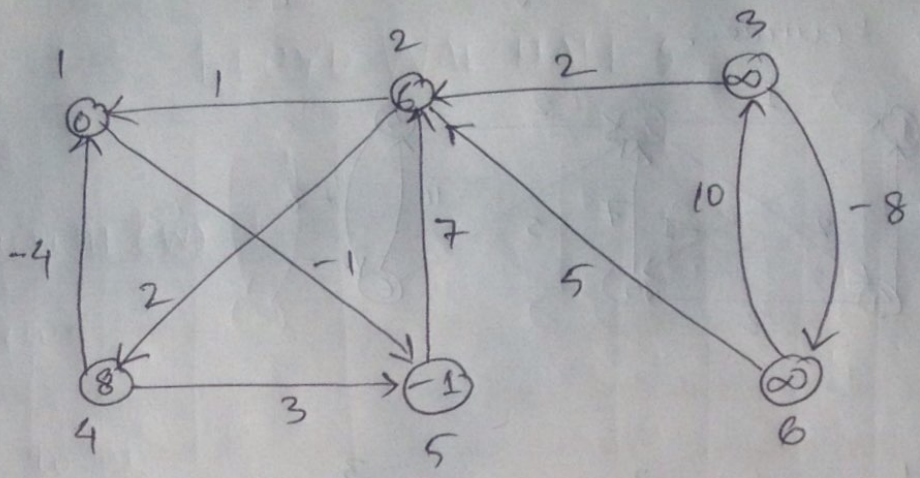


The edges will be relaxed using following order: (1,5), (2,1), (2,4), (3,2), (3,6), (4,1), (4,5), (5,2), (6,2), (6,3)

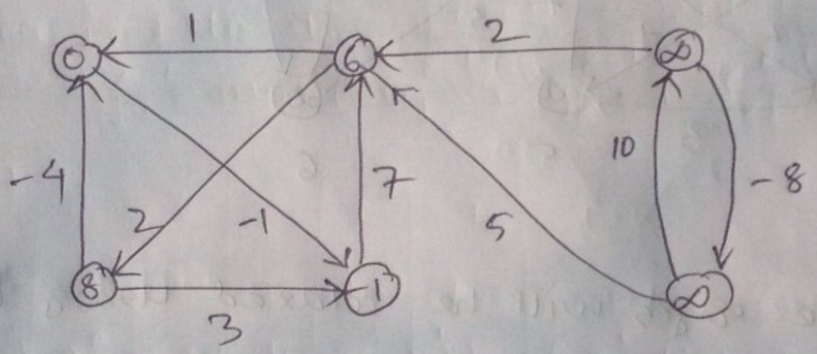
Iteration 1:



Iteration 2:



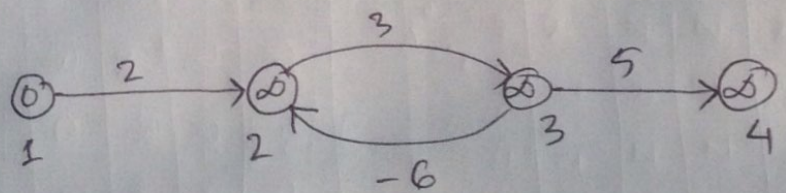
Iteration 3+4+5:



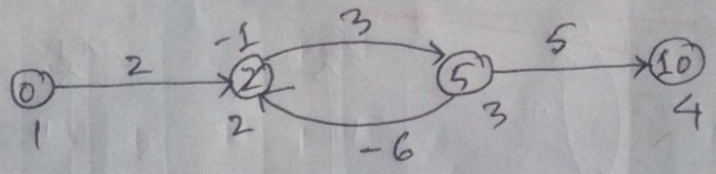
Bellman-Ford algorithm returns TRUE for this graph i.e. there doesn't exist any negative weight cycle.

Q Why can't we get shortest path if there is a negative weight cycle in a graph?

Ans: If there exists a negative weight cycle, we can't get shortest path because continuous relaxation will give us ~~more~~ shorter cost than the shortest. Let us consider the graph below [using node 1 as source]

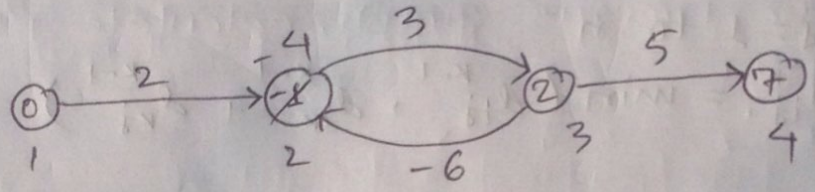


After relaxing for 1 iteration, we could find this



This scenario tells us that the shortes path from node 1 to 2 is : $1 \to 2 \to 3 \to 2$ which costs -1.

Relaxing one more time, we get



Now, following the same path costs us -4!!. If we keep continuing the relaxation, we would find the cost less than this, because there is a negative weight cycle in the graph. So basically, we won't find a shortest path when a negative cycle exists.

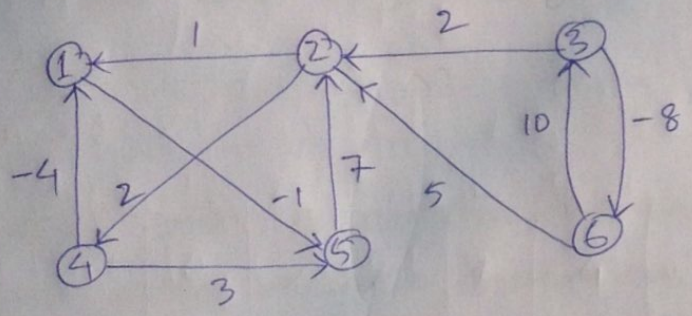
Nayeem Mahmood / 4AM / C161026

FLOYD WARSHALL'S

Floyd-Warshall (W)

```
1   n = W.rows
2   D^0 = W
3   for k = 1 to n
4       let D^k = d_ij^k be a new n x n matrix
5       for i = 1 to n
6           for j = 1 to n
7               d_ij^k = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8   return D^n
```

Find D^0 and π^0 and then D^1 and π^1 using Floyd-Warshall's for the following graph.



Soln:

$D^0 =$

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	∞	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	3	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

$\pi^0 =$

	1	2	3	4	5	6
1	N	N	N	N	1	N
2	2	N	N	2	N	N
3	N	3	N	N	N	3
4	4	N	N	N	4	N
5	N	5	N	N	N	N
6	N	6	6	N	N	N

Nayeem Mahmood / 4AM / C161026

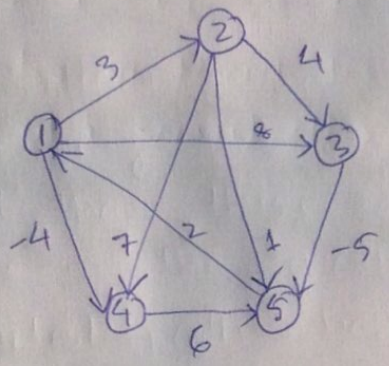
$D^1 =$

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	-5	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

$\pi^1 =$

	1	2	3	4	5	6
1	N	N	N	N	1	N
2	2	N	N	2	1	N
3	N	3	N	N	N	3
4	4	N	N	N	1	N
5	N	5	N	N	N	N
6	N	6	6	N	N	N

Consider the following graph for finding all pairs' shortest path using FW algorithm. Find the matrix π^5 using D^4 and π^4



$D^4 =$

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

$\pi^4 =$

	1	2	3	4	5
1	N	1	4	2	1
2	4	N	4	2	1
3	4	3	N	2	1
4	4	3	4	N	1
5	4	3	4	5	N

Nayeem Mahmood / 4AM / C161026

Soln:

$D^5 =$

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

$\pi^5 =$

	1	2	3	4	5
1	N	5	5	5	1
2	4	N	4	2	1
3	4	3	N	2	1
4	4	3	4	N	1
5	4	3	4	5	N

Considering the previous graph, calculate D^4 using D^3 given below.

$D^3 =$

	1	2	3	4	5
1	0	3	8	7	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0

Soln:

$D^4 =$

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

4. A student has been asked to write down the Floyd-Warshall's algorithm for finding the all pairs shortest paths. So, he has written the following:

```
Floyd-Warshall(W)
    n = W.rows
    D = W
    for i = 1 to n
        for j = 1 to n
            for k = 1 to n
                d_ij = min(d_ij, d_ik + d_kj)
    return D
```

He did a small mistake. Can you find it? Why the above algorithm is not going to work?

Ans: He mistook in the relative ordering of the loops. k should be in the outermost loop while the position of i and j's loops don't matter.

Reason

Considering the ~~follower~~ preceding code segment, we are trying to find a shortest path from i to j, keeping i, j fixed with different values of k. But at that moment, most of the entries in distance matrix is infinity. So, we can't even find that j is reachable from i unless there is a direct edge from i to j.

The correct code segment might be:

Nayeem Mahmood / 4AM / C161026

Floyd-Warshall (w)

```
n = w.rows
D = w
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            dij = min(dij, dik + dkj)
return D.
```

4 How can the concept of intermediate vertex be used to formulate a recursive solution of all pairs shortest paths problem?

Ans: Let $L_{ij}^{(m)}$ be the minimum weight of any path from vertex i to vertex j that contains at most m edges. When $m = 0$, there is a shortest path from i to j with no edges iff $i = j$. Thus,

$$L_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

For $m \geq 1$, we compute $L_{ij}^{(m)}$ as the minimum of $L_{ij}^{(m-1)}$ (the weight of a shortest path from i to j consisting of at most $m-1$ edges) and the minimum weight of any path from i to j consisting of at most m edges, obtained by looking at all possible predecessors k of j. Thus, recursively we define

$$L_{ij}^{(m)} = \min\left(L_{ij}^{(m-1)}, \min_{1 \leq k \leq n}\left\{L_{ik}^{(m-1)} + w_{kj}\right\}\right)$$

$$= \min_{1 \leq k \leq n}\left\{L_{ik}^{(m-1)} + w_{kj}\right\}$$

Dynamic Programming and Greedy Algorithms

What do you mean by optimization problem? Give example.

Ans: Those problems are called optimization problem which can have many possible solution but we are interested in the value of the solution which we expect to be optimal (minimum or maximum).

Examples:

1. Choosing a path to go to Seoul from Dhaka so that no sea is crossed.
2. Finding a way to go to Dhaka from Chittagong with minimum cost.

What is greedy choice? Write the application of greedy method.

Greedy choice: The choice which is made by greedy algorithm that looks best at the moment is called greedy choice. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

Application:

1. Kruskal's algorithm to find MST.
2. Prim's algorithm to find MST
3. Dijkstra's algorithm to find single source shortest path.
4. Fractional knapsack problem.

Nayeem Mahmood / 4AM / C161026

ACTIVITY SELECTION PROBLEM

Greedy-Activity-Selector (s, f)

```
1    n = s.length
2    A = {a1}
3    k = 1
4    for m = 2 to n
5        if s[m] ≥ f[k]
6            A = A ∪ {am}
7            k = m
8    return A
```

Find all possible optimal set for the following data set using activity selector algorithm:

i →	1	2	3	4	5	6	7
Si →	1	3	5	7	6	13	18
Fi →	4	5	6	11	12	18	22

Ans:

All possible optimal sets are:

{1, 3, 4, 6, 7},

{1, 3, 5, 6, 7},

{2, 3, 4, 6, 7},

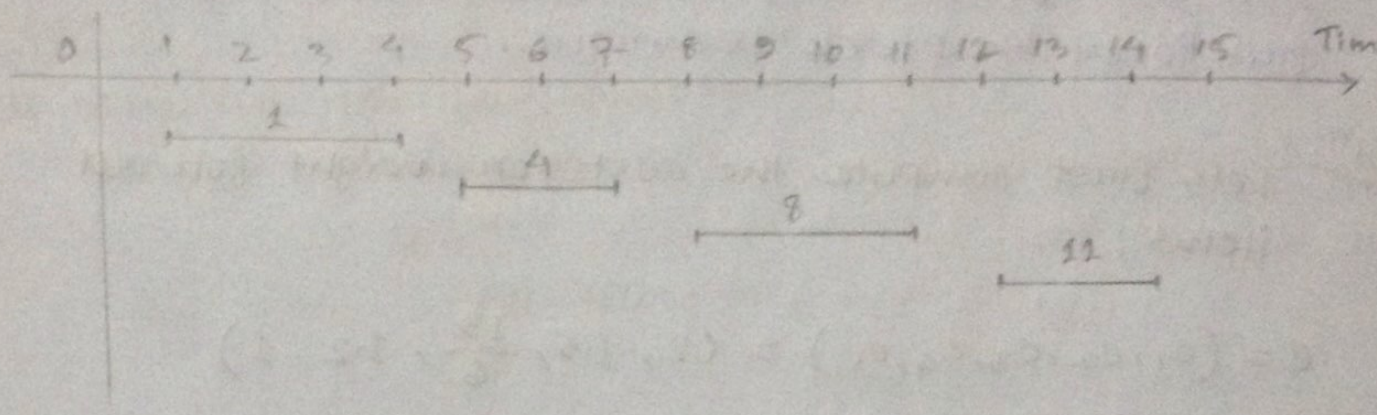
{2, 3, 5, 6, 7}

Nayeem Mahmood / 4AM / C161026

Find possible optimal set for the following data set using activity selector algorithm.

i → 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11
Si → 1 - 3 - 0 - 5 - 3 - 5 - 6 - 8 - 8 - 2 - 12
Fi → 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14

Soln:



So, one of the possible optimal set is {1, 4, 8, 11}.

KNAPSACK PROBLEMS

Find an optimal solution to the knapsack instance $n = 5$, $m = 100$, $(P_1, P_2, P_3, P_4, P_5) = (20, 30, 65, 48, 50)$ and $(w_1, w_2, w_3 w_4, w_5) = (10, 20, 30, 40, 50)$. Use fractional knapsack technique.

Soln:

Lets first compute the cost per weight for all items:

$$c = (c_1, c_2, c_3, c_4, c_5) = (2, 1.5, \frac{13}{6}, 1.2, 1)$$

where, $c_i = \frac{P_i}{w_i}$

Let sort c in non-increasing order.

$$c = (c_3, c_1, c_2, c_4, c_5) = (\frac{13}{6}, 2, 1.5, 1.2, 1)$$

Now, letus fill our knapsack.

1st choice:

We'll take the 3rd item completely.

~~Benefit~~ Profit $= \frac{13}{6} \times 30 = 65$

Remaining space in KS $= 100 - 30$
$= 70$

2nd choice:

We'll take the 1st item completely.

Profit $= 65 + (2 \times 10)$
$= 85$

Rem. space $= 70 - 10$
$= 60$

Nayeem Mahmood / 4AM / C161026

3rd Choice:

We'll take the 2nd item completely.

Profit = 85 + (1.5 × 20) = 115

Rem. Space = 60 − 20

= 40

4th Choice:

We'll take the 4th item completely.

Profit = 115 + (1.2 × 40)

= 163

Rem. Space = 40 − 40

= 0

So, the optimal Profit = 163.

⊞ Find an optimal solution to the knapsack instance $n = 4, m = 16, (P_1, P_2, P_3, P_4) = (45, 10, 30, 45)$, $(w_1, w_2, w_3, w_4) = (3, 5, 9, 5)$.

Soln:

Let us assume data set P contains price per unit weight of the items. Lets sort the P first.

$(P_1, P_4, P_3, P_2) = (45, 45, 30, 10)$

$(w_1, w_4, w_3, w_2) = (3, 5, 9, 5)$

Profit = 0

Remaining space = 16

1st Choice:

We'll take the 1st item completely.

Profit = 0 + (3 × 45) = 135

Rem. Space = 16 − 3 = 13

Nayeem Mahmood / 4AM / C161026

2nd choice:

We'll take the 4th item completely.

$$\text{Profit} = 135 + (5 \times 45)$$
$$= 360$$

$$\text{Rem. Space} = 13 - 5$$
$$= 8$$

3rd choice:

We'll take the 8 units of 3rd item.

$$\text{Profit} = 360 + (8 \times 30)$$
$$= 600$$

$$\text{Rem. Space} = 8 - 8$$
$$= 0$$

So, the optimal profit = 600.

4] A thief robbing a store finds 4 item. The price & weight of each item is given in the table below. The thief wants to ~~make~~ take as valuable a load as possible, but he can carry at most 16 pounds in his knapsack. He can't take a fractional amount of an item or take an item more than once. Find the maximum profit the thief can obtain using Branch and Bound method.

Item Number	1	2	3	4
Price ($)	45	30	45	10
Weight (lb)	3	5	9	5

Soln:

Here, Y denotes that we' the thief takes that item and N denotes doesn't.
The tree shows us that if he takes item 1 & 3, he can obtain maximum profit of 90.

Show that 0-1 knapsack problem can't be solved using greedy approach whereas fractional knapsack can be solved.

Soln: Let us consider a store which has 3 items.

Item	1	2	3
Price	60	100	120
weight	10	20	30

We have a knapsack of capacity 50. Lets first discuss about fractional knapsack.

Item	1	2	3
Price	60	100	120
weight	10	20	30
Price Per Unit weight	6	5	4

Fractional knapsack tells us to take the item first which has maximum price per unit. Accordingly, we can earn profit of

$$(6 \times 10) + (5 \times 20) + (4 \times 20) = 240$$

Let us discuss about 0-1 knapsack now. The greedy technique tells us to take item 1 and 2 since their price per unit weight is greater than that of item 3. But since this is 0-1 knapsack, we can't take fractional portion of item 3. Again we can't take item 3 completely as well because we're out of capacity. So, greedy technique gives us profit of $(60 + 100) = 160$ in total. But a dynamic programming

Nayeem Mahmood / 4AM / C161026

approach would give us Profit of (120+100) = 220 by taking item 2 and 3.

Write down the differences between DP and GT.

Greedy Technique	Dynamic Programming
It makes a choice locally that looks best at the moment and hopes that this choice will lead to a globally optimal solution.	It tries to solve all the possible solution by solving and combining the solution to the sub-problems.
It is basically a top-down approach.	It is basically a bottom-up approach.
Example: Fractional KS, Dijkstra, Kruskal's MST etc.	Example: 0-1 KS, Coin Changing Problem, LCS etc.

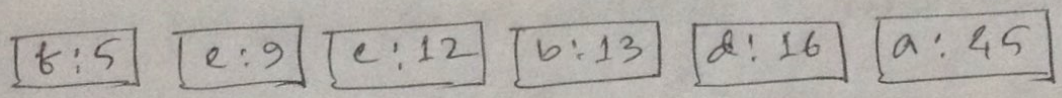
HUFFMAN CODES

Huffman (C)

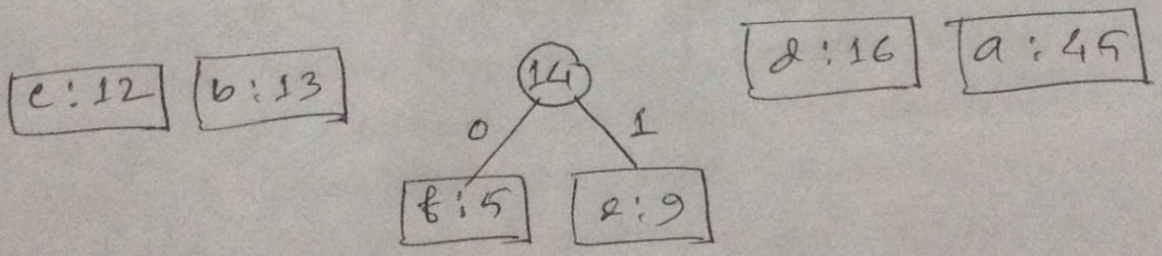
```
1  n = |C|
2  Q = C
3  for i = 1 to n - 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT (Q, z)
9  return EXTRACT-MIN(Q)      // returns the root of the tree
```

A data file contains only the characters a-f with the frequencies : a(45), b(13), c(12), d(16), e(9), f(5). Draw the corresponding Huffman tree and utilizing that tree tell the binary encoding for each letter.

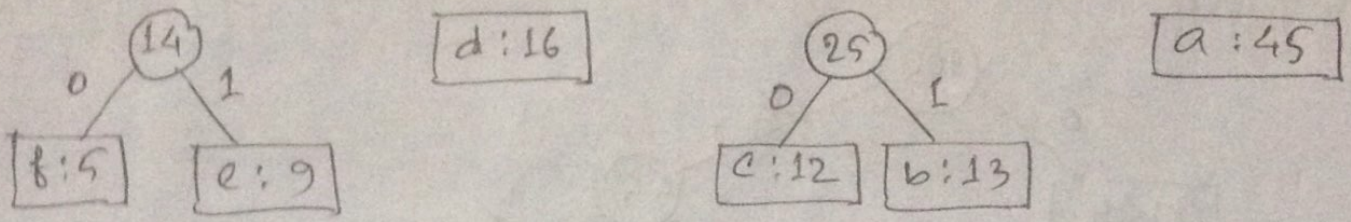
Soln:



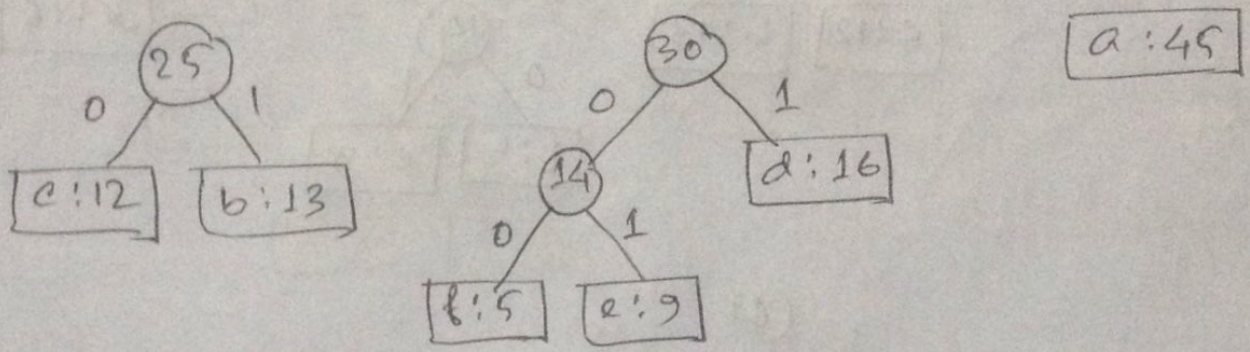
(1)



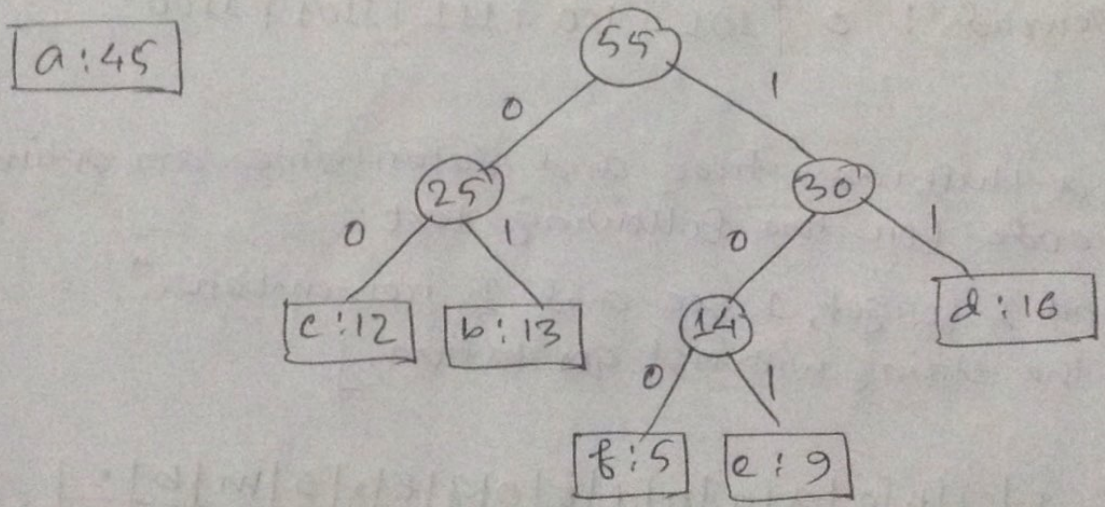
(2)



(3)

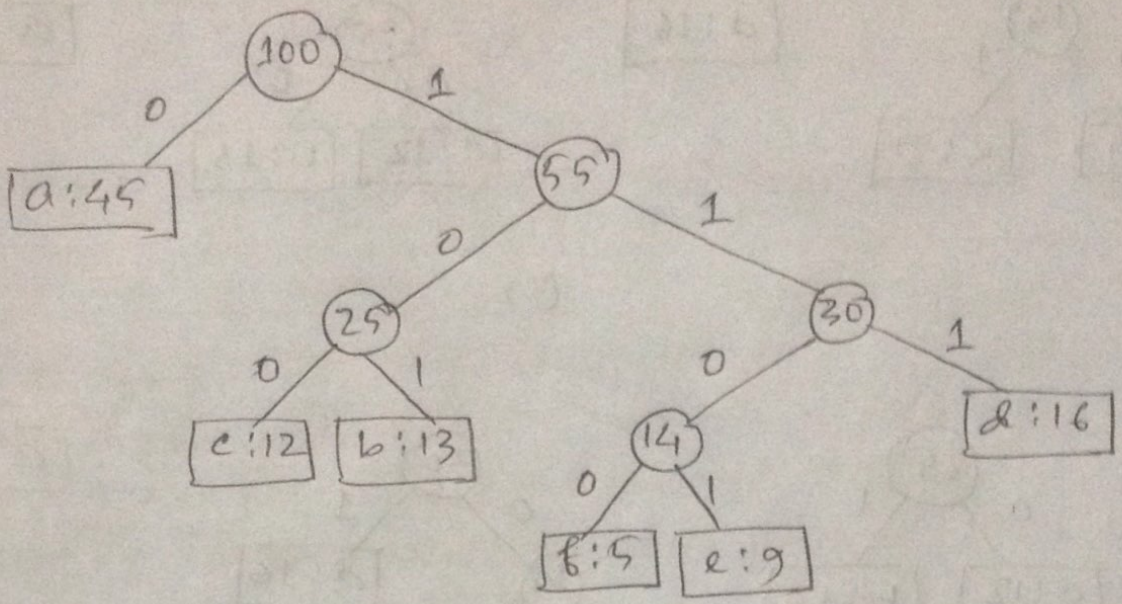


(4)



(5)

Nayeem Mahmood / 4AM / C161026



(6)

Letter :	a	b	c	d	e	f
Binary codeword :	0	101	100	111	1101	1100

[H] Generate a Huffman tree and determine an optimal Huffman code for the following text:
"I hear and I forget, I see and I remember."
[Consider the string without quotation.]

Characters-	I	□	h	e	a	r	n	d	f	o	g	t	,	s	m	b	.
Frequencies-	4	9	1	7	3	4	2	2	1	1	1	1	1	1	2	1	1

□ → Blank Space

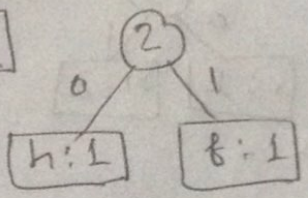
h:1 f:1 o:1 g:1 t:1 ,:1 s:1 b:1 .:1 n:2 d:2

m:2 a:3 I:4 r:4 e:7 □:9

(1)

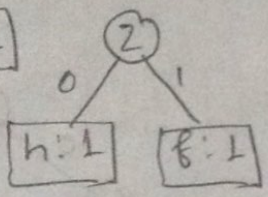
o:1 g:1 t:1 ,:1 s:1 b:1 .:1 n:2 d:2

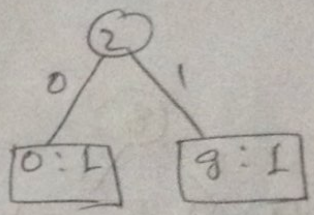
m:2 a:3 I:4 r:4 e:7 □:9

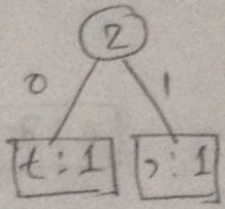


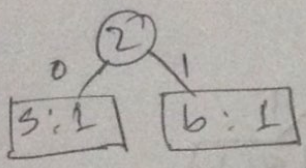
(2)

.:1 n:2 d:2 m:2



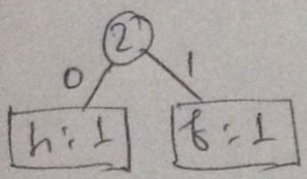


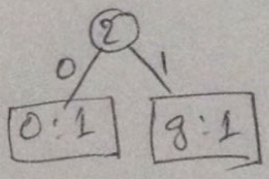


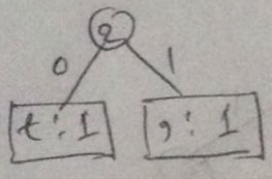


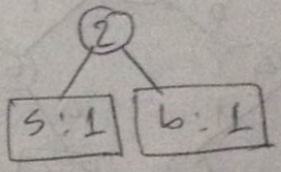
a:3 I:4 r:4 e:7 □:9

~~(1)+(2)~~ (3)+(4)+(5)

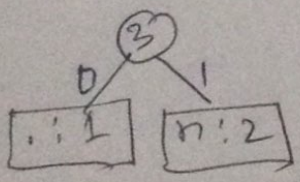




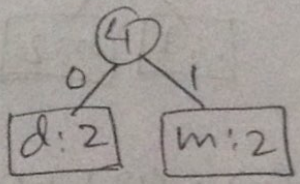




a:3



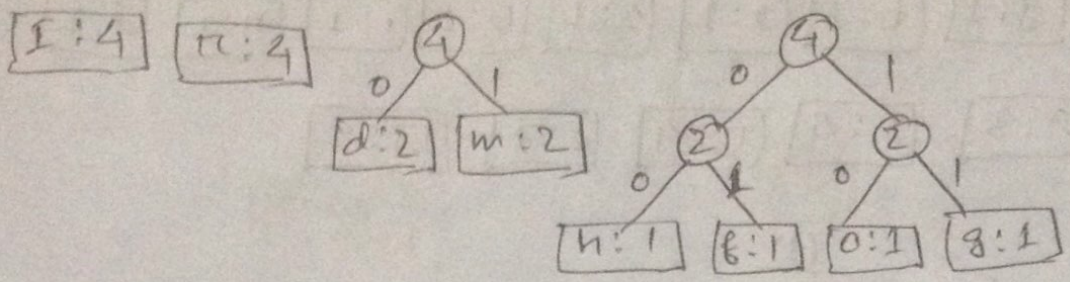
I:4 r:4

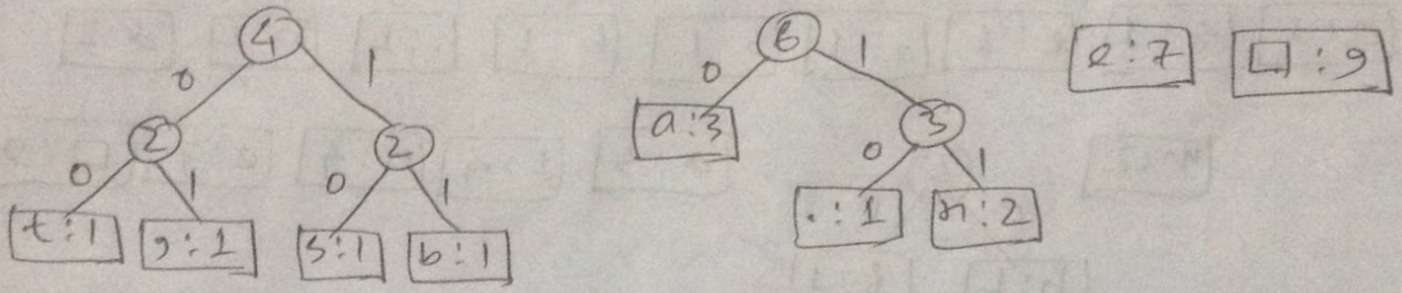


e:7 □:9

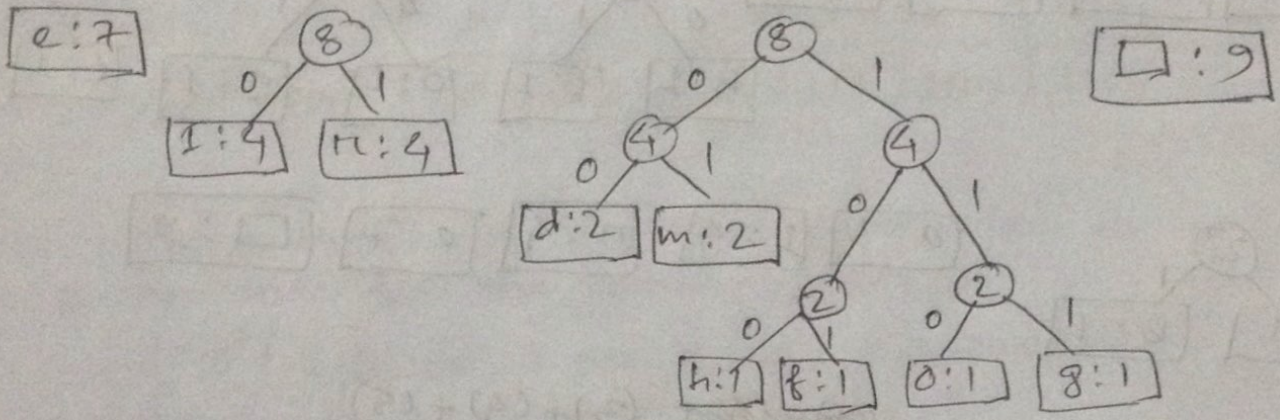
(6)+(7)

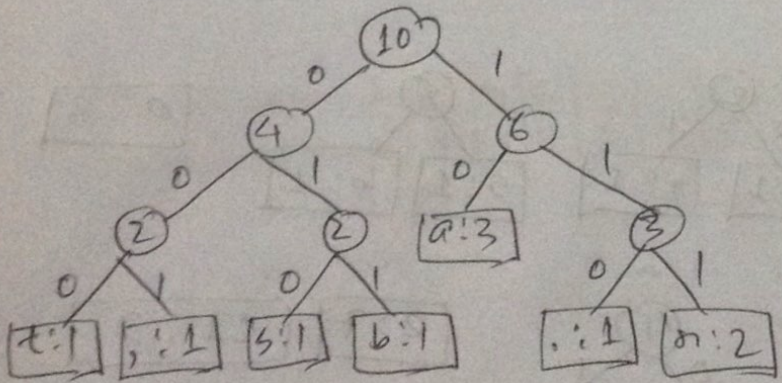
Nayeem Mahmood / 4AM / C161026





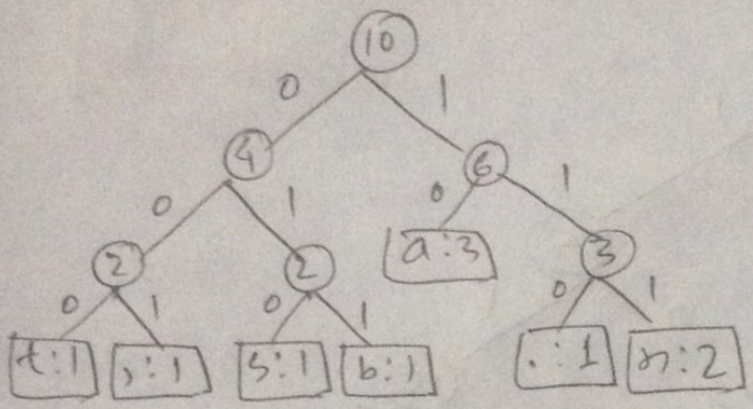
(8) + (9) + (10)

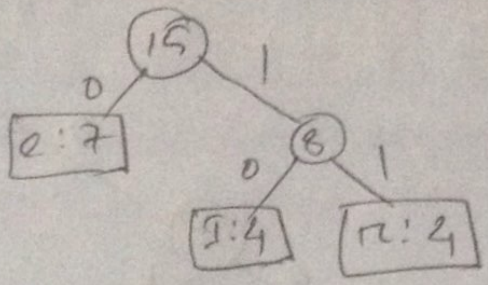


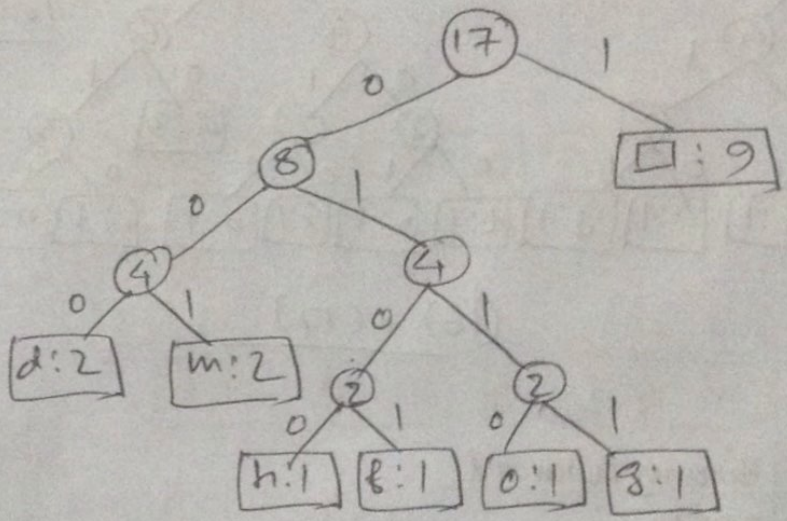


(11) + (12) + (13)

Nayeem Mahmood / 4AM / C161026

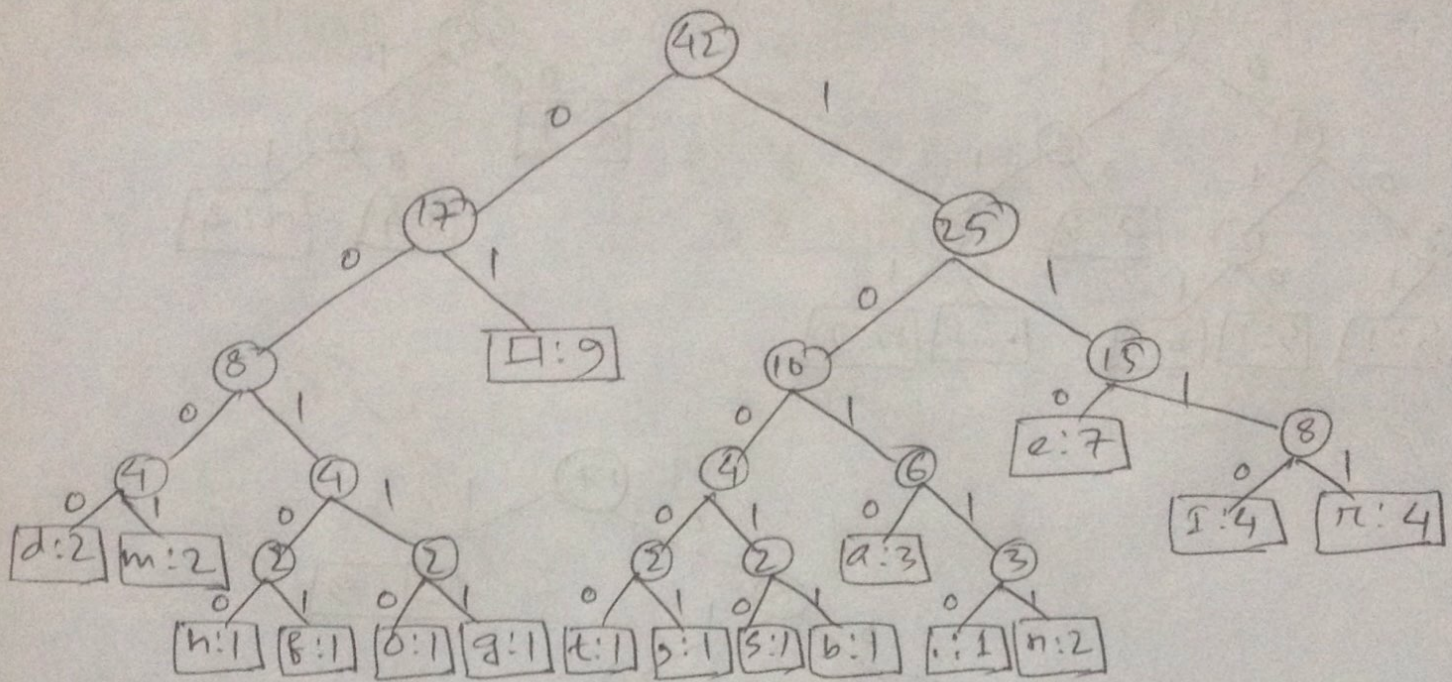






(14) + (15)

Nayeem Mahmood / 4AM / C161026



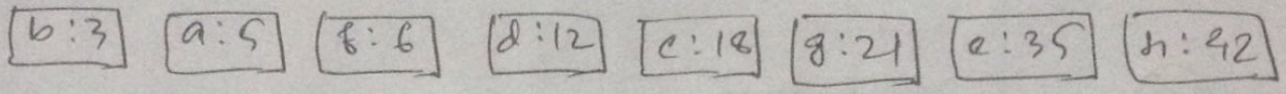
(16) + (17)

characters	Binary Codeword
d	0000
m	0001
h	00100
f	00101
o	00110
g	00111
□	01
t	10000
,	10001
s	10010
b	10011
a	1010
.	10110
n	10111
e	110
I	1110
π	1111

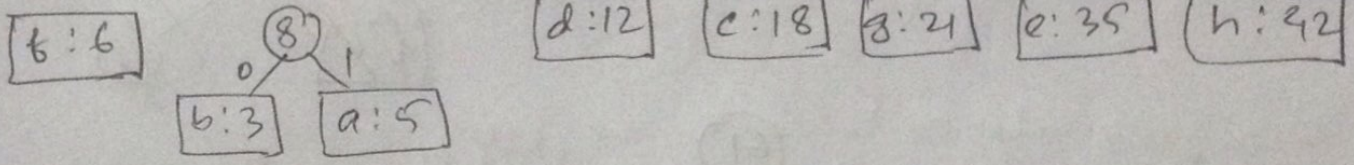
Nayeem Mahmood / 4AM / C161026

[Ex] A data file contains only the characters a-h with the frequencies: a:5, b:3, c:18, d:12, e:35, f:6, g:21, h:42. Draw the corresponding Huffman tree and utilizing that tree tell the binary encoding for each letter.

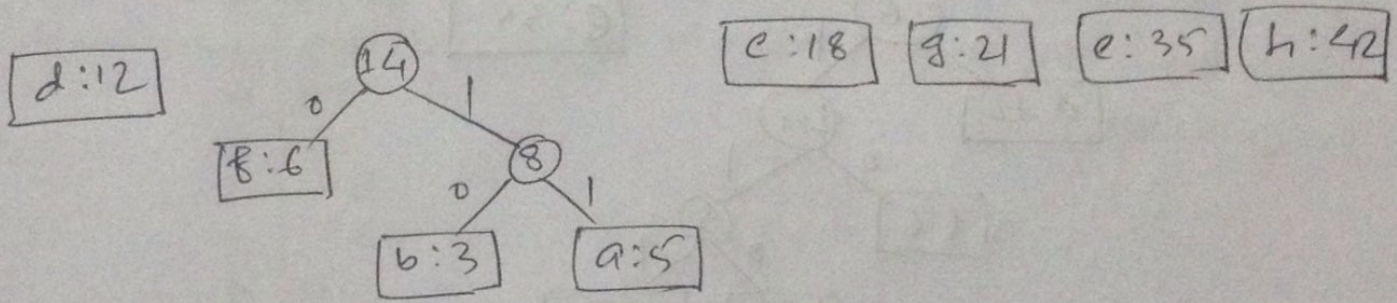
Letter:	a	b	c	d	e	f	g	h
Freq.:	5	3	18	12	35	6	21	42



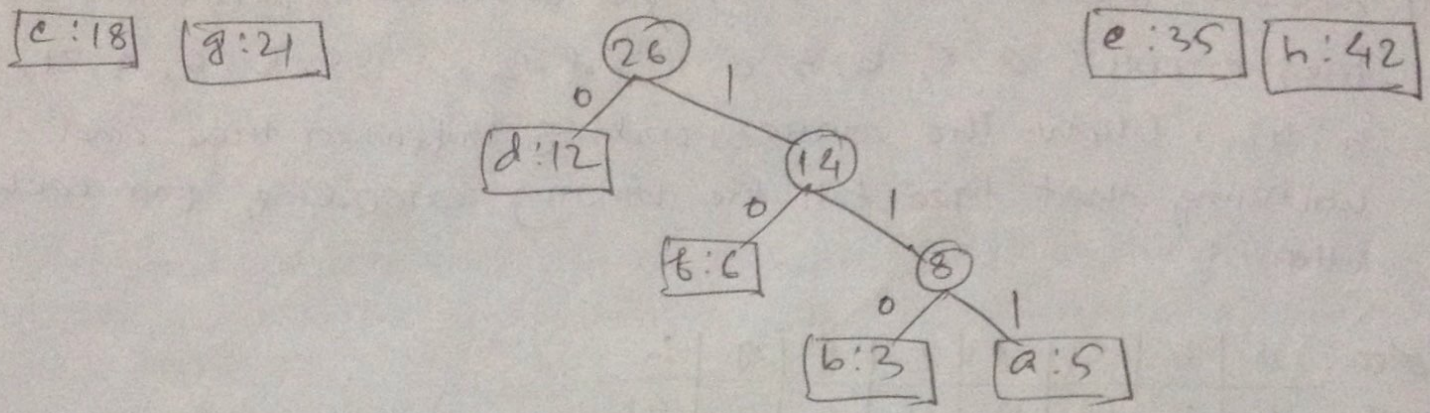
(1)



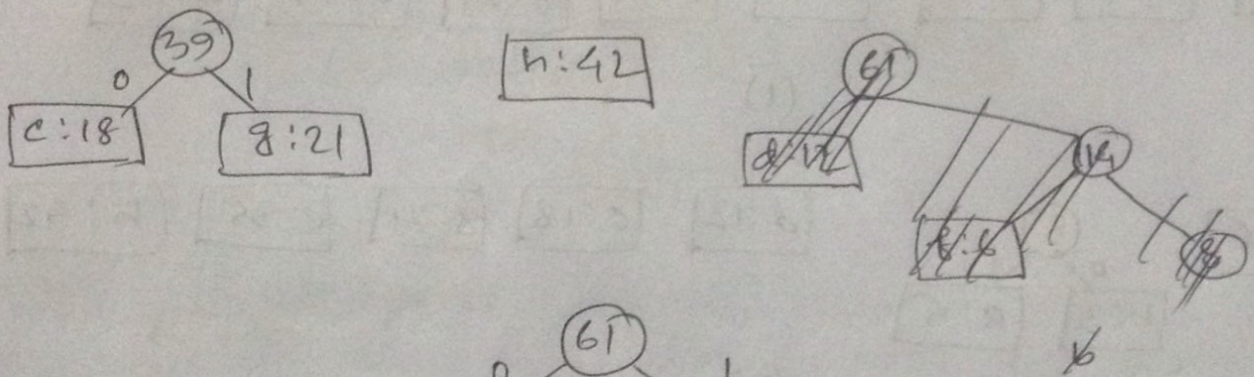
(2)

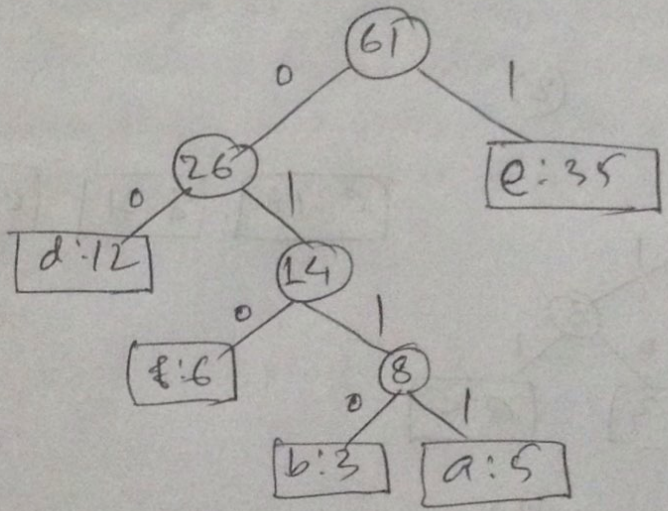


(3)



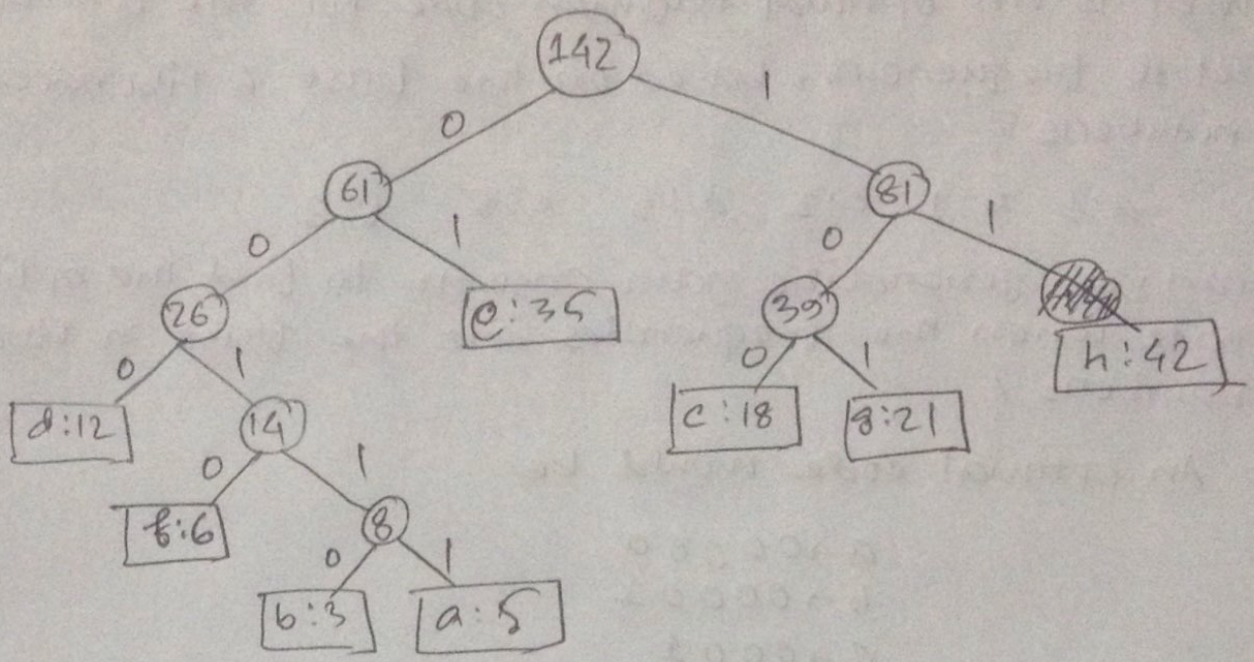
(4)





(5)+(6)

Nayeem Mahmood / 4AM / C161026



(7) + (8)

Letter	Binary Code
d	000
f	0010
b	00110
a	00111
e	01
c	100
g	101
h	11

Nayeem Mahmood / 4AM / C161026

[H] What is an optimal Huffman code for the following set of frequencies, based on the first 6 Fibonacci numbers?

a:1 b:1 c:2 d:3 e:5 f:8

Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?

Ans: An optimal code would be

a → 00000
b → 00001
c → 0001
d → 001
e → 01
f → 1

This generalizes to have the first n Fibonacci numbers as the frequencies in that the $k < n$th most frequent letter has codeward $0^{k-1}1$, and the nth most frequent letter having codeward 0^{n-1}.

In general,

$$\sum_{i=0}^{n-1} F(i) = F(n+1) - 1$$

Nayeem Mahmood / 4AM / C161026

Number Theory

Question: Find the value of (5^{27}) mod 7 using modular exponentiation. Show the steps.

Solution:

[

- (a + b) mod m = {(a mod m) + (b mod m)} mod m
- (a - b) mod m = {(a mod m) - (b mod m)} mod m
- (a x b) mod m = {(a mod m) x (b mod m)} mod m

]

5^{27} mod 7 $= (5^{26} \times 5)$ mod 7

$= \{(5^{26} \text{ mod } 7) \times (5 \text{ mod } 7)\}$ mod 7 ----- (1)

Now,

5^{26} mod 7 $= (5^{13} \times 5^{13})$ mod 7

$= \{(5^{13} \text{ mod } 7) \times (5^{13} \text{ mod } 7)\}$ mod 7 ----- (2)

Again,

5^{13} mod 7 $= (5^{12} \times 5)$ mod 7

$= \{(5^{12} \text{ mod } 7) \times (5 \text{ mod } 7)\}$ mod 7 ----- (3)

Now,

5^{12} mod 7 $= (5^{6} \times 5^{6})$ mod 7

$= \{(5^{6} \text{ mod } 7) \times (5^{6} \text{ mod } 7)\}$ mod 7 ----- (4)

Now,

5^{6} mod 7 $= (5^{3} \times 5^{3})$ mod 7

$= \{(5^{3} \text{ mod } 7) \times (5^{3} \text{ mod } 7)\}$ mod 7 ----- (5)

Again,

5^{3} mod 7 $= (5^{2} \times 5)$ mod 7

$= \{(5^{2} \text{ mod } 7) \times (5 \text{ mod } 7)\}$ mod 7 ----- (6)

Nayeem Mahmood / 4AM / C161026

Now,

$5^2 \bmod 7 = 25 \bmod 7$

$= 4$ ----- (7)

So, from (6),

$5^3 \bmod 7 = (5^2 \times 5) \bmod 7$

$= \{(5^2 \bmod 7) \times (5 \bmod 7)\} \bmod 7$

$= (4 \times 5) \bmod 7$ [using (7)]

$= 20 \bmod 7$

$= 6$ ------ (8)

So, from (5),

$5^6 \bmod 7 = (5^3 \times 5^3) \bmod 7$

$= \{(5^3 \bmod 7) \times (5^3 \bmod 7)\} \bmod 7$

$= (6 \times 6) \bmod 7$ [using (8)]

$= 36 \bmod 7$

$= 1$ ------ (9)

So, from (4),

$5^{12} \bmod 7 = (5^6 \times 5^6) \bmod 7$

$= \{(5^6 \bmod 7) \times (5^6 \bmod 7)\} \bmod 7$

$= (1 \times 1) \bmod 7$ [using (9)]

$= 1$ ------ (10)

So, from (3),

$5^{13} \bmod 7 = (5^{12} \times 5) \bmod 7$

$= \{(5^{12} \bmod 7) \times (5 \bmod 7)\} \bmod 7$

$= (1 \times 5) \bmod 7$ [using (10)]

$= 5$ ------ (11)

So, from (2),

$5^{26} \bmod 7 = (5^{13} \times 5^{13}) \bmod 7$

$= \{(5^{13} \bmod 7) \times (5^{13} \bmod 7)\} \bmod 7$

$= (5 \times 5) \bmod 7$ [using (11)]

$= 25 \bmod 7$

$= 4$ [using (4)] ------ (12)

Finally, from (1),

$5^{27} \bmod 7 = (5^{26} \times 5) \bmod 7$

$= \{(5^{26} \bmod 7) \times (5 \bmod 7)\} \bmod 7$

$= (4 \times 5) \bmod 7$ [using (12)]

$= 20 \bmod 7$

$= 6$

Hence, $5^{27} \bmod 7 = 6$.

Question: Find GCD(300, 21) using Euclid's algorithm.
Solution:

[
Euclid's Algorithm:

```
GCD(a, b):              // Here, a >= b
1.   if b == 0
2.       return a
3.    else
4.       return GCD(b, a mod b)
```

]

GCD(300, 21) = GCD(21, 300 mod 21) = GCD(21, 6)
= GCD(6, 21 mod 6) = GCD(6, 3)
= GCD(3, 6 mod 3) = GCD(3, 0)
= 3

COMPUTATIONAL GEOMETRY

Describe line segment properties.

In geometry, a line segment is a part of a line that is bounded by two distinct end points and contains every point on the line between its endpoints.

Properties:

1. A closed line segment includes both end points.
2. An open line segment excludes both end points.
3. A half-open line segment includes exactly one of the end points.
4. A line segment is a connected, non-empty set.
5. A pair of line segments can be any of these: intersecting, parallel, skew or none of these.

How can we determine if two line segments are colinear?

If the cross product of two line segments is 0, then they are colinear, pointing in either the same or opposite direction.

Suppose that an ant is traveling from point A(2, 1) to point B(6, 4) in straight line, and from there it starts travelling towards point C(4, 6). Did the ant turn to left or right at point B? From point C it travels toward point D(2, 8). What turn did the ant make at point C, left or right?

Nayeem Mahmood / 4AM / C161026

Soln:

Given points,

$A(2,1)$,
$B(6,4)$,
$C(4,6)$,
$D(2,8)$

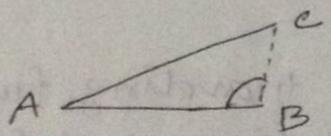
Let, $B' = B - A$
$= (6-2, 4-1)$
$= (4,3)$

and $C' = C - A$
$= (4-2, 6-1)$
$= (2,5)$

Now, $B' \times C' = \det \begin{pmatrix} 4 & 2 \\ 3 & 5 \end{pmatrix}$

$= 20 - 6$
$= 14$

Since $B' \times C'$ is positive, hence AB is clockwise from AC.



Therefore, the ant took left turn at point B.

Now, let

$C'' = C - B$
$= (4-6, 6-4)$
$= (-2, 2)$

$D' = D - B$
$= (2-6, 8-4) \quad = (-4, 4)$

$$C'' \times D' = \det \begin{pmatrix} -2 & -4 \\ 2 & 4 \end{pmatrix}$$

$$= -8 + 8$$

$$= 0$$

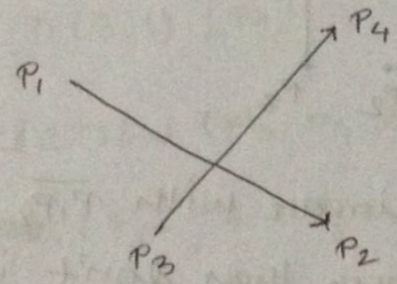
Since $C'' \times D'$ is 0, hence BC and CD segments are colinear. Therefore, the ant didn't take any turn at point C.

⊞ How do you determine whether two line segments intersect or not? Describe the basic idea using figure.

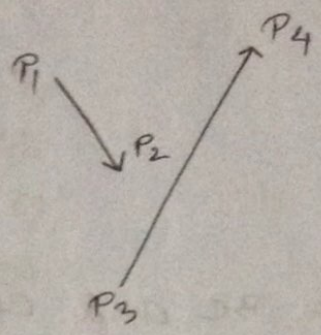
Two line segments intersect iff either (or both) of the following condition holds:

1. Each segment straddles the line containing the other.
2. An endpoint of one segment lies on the other segment.

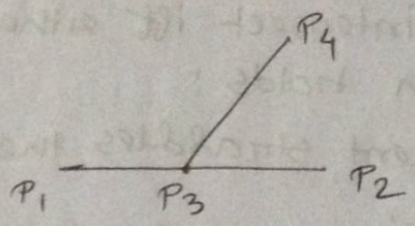
There could ~~Both~~ four kind of cases. Each of them are described below.



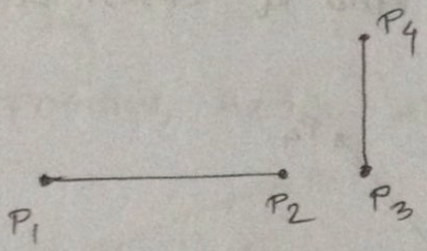
Case (a): $(P_3 - P_1) \times (P_2 - P_1)$ and $(P_4 - P_1) \times (P_2 - P_1)$ differ. Also $(P_1 - P_3) \times (P_4 - P_3)$ and $(P_2 - P_3) \times (P_4 - P_3)$ differ. Thus two segments intersect each other.



Case (b): segment $\overline{P_3P_4}$ straddles the line containing $\overline{P_1P_2}$, but $\overline{P_1P_2}$ doesn't straddle the line containing $\overline{P_3P_4}$. Thus they don't intersect.



Case (c): Point P_3 is colinear with $\overline{P_1P_2}$ and is between P_1 and P_2. Thus they intersect.



Case (d): Point P_3 is colinear with $\overline{P_1P_2}$, but it is not between P_1 & P_2. Thus they don't intersect.

[4] Given one line segment AB and a point C. How can you determine whether point C is on the line segment AB or not?

Let us assume the coordinates of the points

$A(A_x, A_y)$,

$B(B_x, B_y)$,

$C(C_x, C_y)$

C is on the line segment AB iff the following condition holds:

$$\min(A_x, B_x) \leq C_x \leq \max(A_x, B_x) \wedge \min(A_y, B_y) \leq C_y \leq \max(A_y, B_y)$$

[4] Given two line segments AB and CD where coordinates are A(2,2), B(5,3), C(4,6) and D(8,1). Find whether they intersect or not.

Given,

A(2,2) $[P_1]$
B(5,3) $[P_2]$
C(4,6) $[P_3]$
D(8,1) $[P_4]$

$d_1 = \text{DIRECTION}(P_3, P_4, P_1)$

$= \det (P_1 - P_3) \times (P_4 - P_3)$

$= (-2, -4) \times (4, -5)$

$= \det \begin{pmatrix} -2 & 4 \\ -4 & -5 \end{pmatrix}$

$= 10 + 16$

$= 26$

Nayeem Mahmood / 4AM / C161026

$d_2 = \text{DIRECTION}(P_3, P_4, P_2)$

$= (P_2 - P_3) \times (P_4 - P_3)$

$= (1, -3) \times (4, -5)$

$= \det \begin{pmatrix} 1 & 4 \\ -3 & -5 \end{pmatrix}$

$= -5 + 12$

$= 7$

$d_3 = \text{DIRECTION}(P_1, P_2, P_3)$

$= (P_3 - P_1) \times (P_2 - P_1)$

$= (2, 4) \times (3, 1)$

$= \det \begin{pmatrix} 2 & 3 \\ 4 & 1 \end{pmatrix}$

$= 2 - 12$

$= -10$

$d_4 = \text{DIRECTION}(P_1, P_2, P_4)$

$= (P_4 - P_1) \times (P_2 - P_1)$

$= (6, -1) \times (3, 1)$

$= \det \begin{pmatrix} 6 & 3 \\ -1 & 1 \end{pmatrix}$

$= 6 + 3$

$= 9$

Nayeem Mahmood / 4AM / C161026

Since d_3 and d_4 are of opposite sign, segment CD straddles AB but since d_1 and d_2 are of same sign, segment AB doesn't straddle CD. Again they are not colinear. Therefore they don't intersect.

[H] Write an algorithm to determine line segment intersection.

SEGMENT-INTERSECT (P_1, P_2, P_3, P_4)

```
1   d1 = DIRECTION (P3, P4, P1)
2   d2 = DIRECTION (P3, P4, P2)
3   d3 = DIRECTION (P1, P2, P3)
4   d4 = DIRECTION (P, P2, P4)
5   if (d1d2 < 0  ∧ d3d4 < 0)
6           return TRUE
7   else if d1 == 0 and ON-SEGMENT (P3, P4, P1)
8           return TRUE
9   else if d2 == 0 and ON-SEGMENT (P3, P4, P2)
10          return TRUE
11  else if d3 == 0 and ON-SEGMENT (P1, P2, P3)
12          return TRUE
13  else if d4 == 0 and ON-SEGMENT (P1, P2, P4)
14          return TRUE
15  else return FALSE
```

DIRECTION (P_i, P_j, P_k)

1 return $(P_k - P_i) \times (P_j - P_i)$

ON-SEGMENT (P_i, P_j, P_k)

1 if $\min(x_i, x_j) \leq x_k \leq \max(x_i, x_j) \wedge$
$\min(y_i, y_j) \leq y_k \leq \max(y_i, y_j)$
return TRUE
2
3 else return FALSE

4. Define Convex Hull.

The convex hull of a set Q of points, denoted by $CH(Q)$, is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

Question: Consider the following points and find convex hull using Graham's Scan algorithm:
P1(0.7, 2.7), P2(1.8, 3.2), P3(2.6, 0.8), P4(0.9, 2.5), P5(0.6, 0.7), P6(1.5, 0.6), P7(1.5, 2.5), P8(2, 1.5), P9(1.0, 2.0), P10(0.8, 1.5)

Answer: