4 Feb 22

Date :

Page No.

GRAPHS

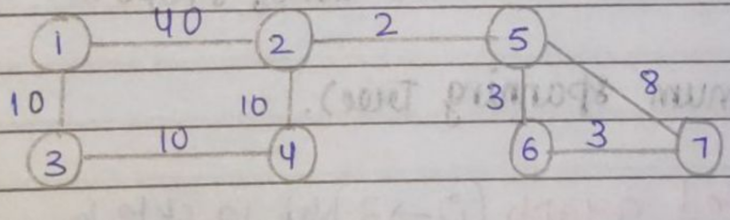
A graph is a non-linear Data structure consists of nodes and edges.

The nodes are sometimes also referred as vertices & the edges are the lines or arcs that connect any 2 nodes in the graph.

Usage -

→ Graph are used to solve many real-life problems.

→ Graphs are used to represent networks. The networks may include paths in a city or telephone network or ckt network. Graphs are also used in social networks like linkedln, fb. Ex - in fb, each person is represented with a vertex (or node). Each node is a structure and contains information person id, name, gender, locale etc.



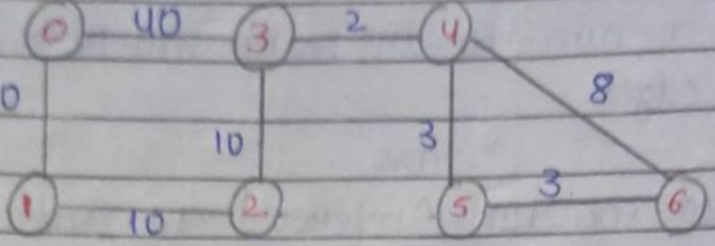
vertices 1, 2, 3, 4, 5, 6, 7

Edges. 12, 13, 34, 24, 25, 56, 67, 57.

Date :

Page No.

(NDG)



Questions ?

(1) Does a path exist b/w 0 & 6 cities ?

(2) Print all path of 0 & 6 ?

- 0123456. → SP weigth (38 km)
- 012346
- 0346 → Shortest path (Sbse km vertices visit hui)
- 03456.

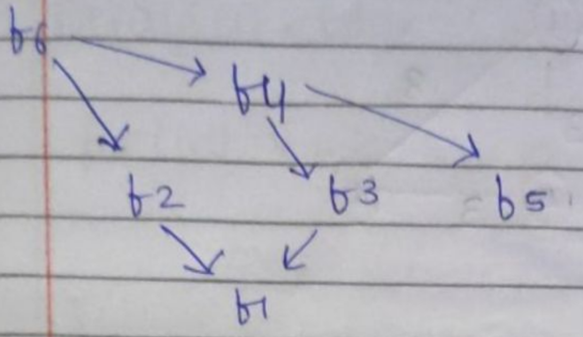
(i) smallest Path in terms of km / weight ?

(ii) '' no. of stops ? (0346)

(3) MST ? (Minimum spanning Tree).

* (Non directed graph (0→3 bhi ja skte h , 3→0 bhi))

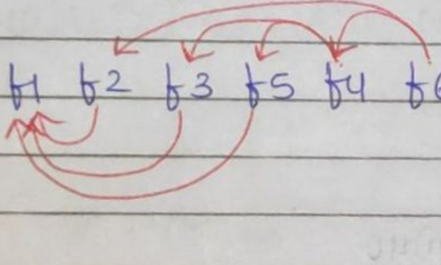
(Directed graph)



(1) Order of compilation ?

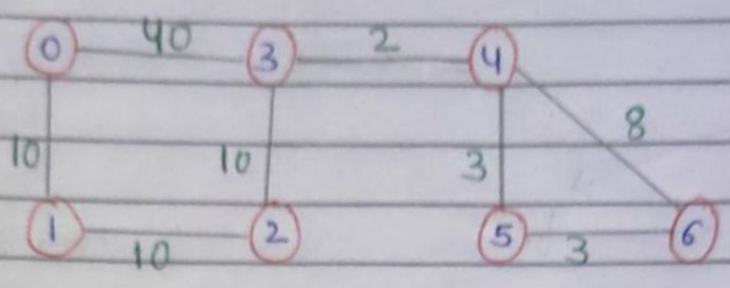
f1 f2 f3 f5 f4 f6. → TOPOLOGICAL SORT
(sbse phle f1 compile hogi
or sb).

(2) Dependency ?



(f6 Ko compile krna h to phle
f2f4 compile honi chie)

graph Representation



★ Adjacency List , Adjacency Matrix.

→ Most of the time will use Adjacency list

7 size Array (7 vertices h to)

ArrayList < Edge > [ ] graph

(ALof edge ka array h)

v1 v2 weight.

★ Vertices -

Vertices		Edges		
0	→	01@10	03@40	← Edges ki AL
1	→	10@10	12@10.	
2	→	21@10	23@10.	
3	→	30@40	32@10	34@2
4	→	43@2	45@3	46@8
5	→	54@3	56@3	
6.	→	64@8	65@3.	

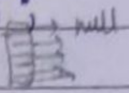
★ Degree - 3 ki degree 3 h (uspe kitni Edge aarhi h)

★ Incident - 03, 23; 34 are incident on 3.

AL<Edge>?. ek vertex pe aane vli edges h
[] → Hr vertex ke saamne uski edges store krni h.

Jitni vertices h

```
ArrayList<Edge>[] graph = new ArrayList[5];
```



```
for(int v = 0 ; v < graph.length;v++) {
        graph[i] = new ArrayList<>();
}
```

graph nhi bna pr vertices bn gai
↳ sbke saamne AL aagai

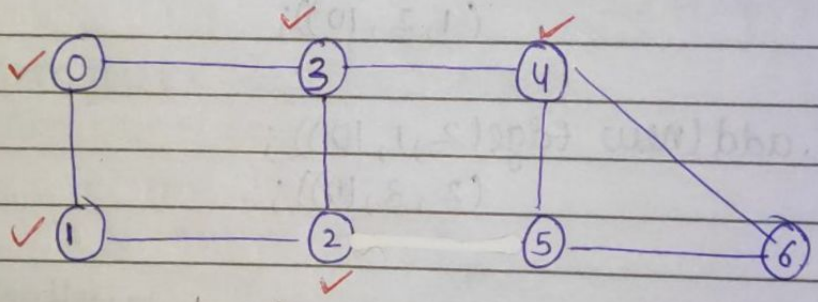
```
graph[0]. add(new Edge(0,1,10)) ;
graph[0]. add(new Edge (0,3,40));

graph[1]. add(new Edge(1,0 ,10));
                      (1,2,10));

graph[2].add(new Edge(2,1,10)) ;
                      (2,3,10));
```

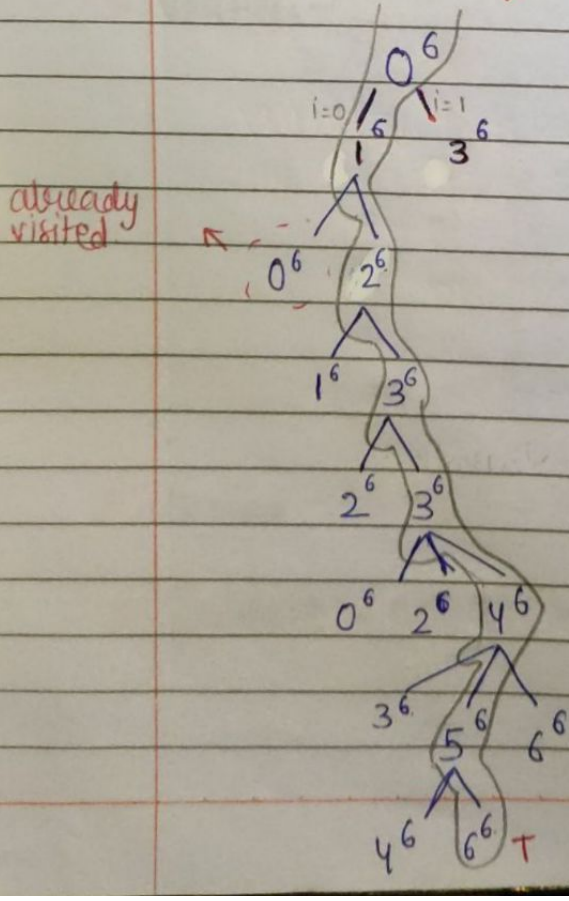
Date :
Page No.

1. HAS Path?-

I/P. No. of vertices - 7
No. of edges - 8
(then 8 edges) → 0 1 10
↘ 1 2 10
:
:

src 0 ⌉ → Does Path exist
dest 6 ⌋ B/W 0 & 6. (Destination)
(src)





Date :
Page No.

```
jo ek baar visit ho gya vha dubara nhi jaana
                                    jitni vertices hutna bda array bnega
boolean[] visited = new boolean [vtces];

boolean pathExists = hasPath(graph, visited, src, dest);
soutln(pathExists);
}.
                    (is graph me is src se is dest ki tref koi rsta h ki nhi?)
public static boolean hasPath(ArrayList<Edge>[] graph,

                        boolean[] visited, int src, int dest){
// is the dest a direct nbr of src
  if(src == dest){
      return true;
  }.
//does a path exist from any of src's neighbours
//ye src to visit krliya <- visited[src] = true;
                     (src ke apr edges aati h vo mil gai)
  for(int i=0 ; i<graph[src].size(); i++){ // ye loop unke liye jo visit nhi hue h
                                                (inke liye nhi).
      Edge edge = graph[src].get(i);
                        //src ke saamne ki saari Edges mili
                          or ith edge utha li
      int nbr = edge.nbr; -> //OR uska nbr dekh liya

      if(visited[nbr] == false){ //agr ye nbr unvisited h

          boolean pathExists = hasPath(graph, visited, nbr, dest);
                        (neighbour se pucha dest tk ka
                         path h ya nhi!)
          if(pathExists){      ↓path h to return T
                return true;
          }.
      }
  }.
return false;
```

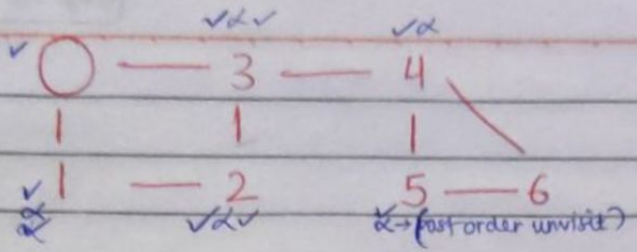
Date :
Page No.

2. Print All Paths -

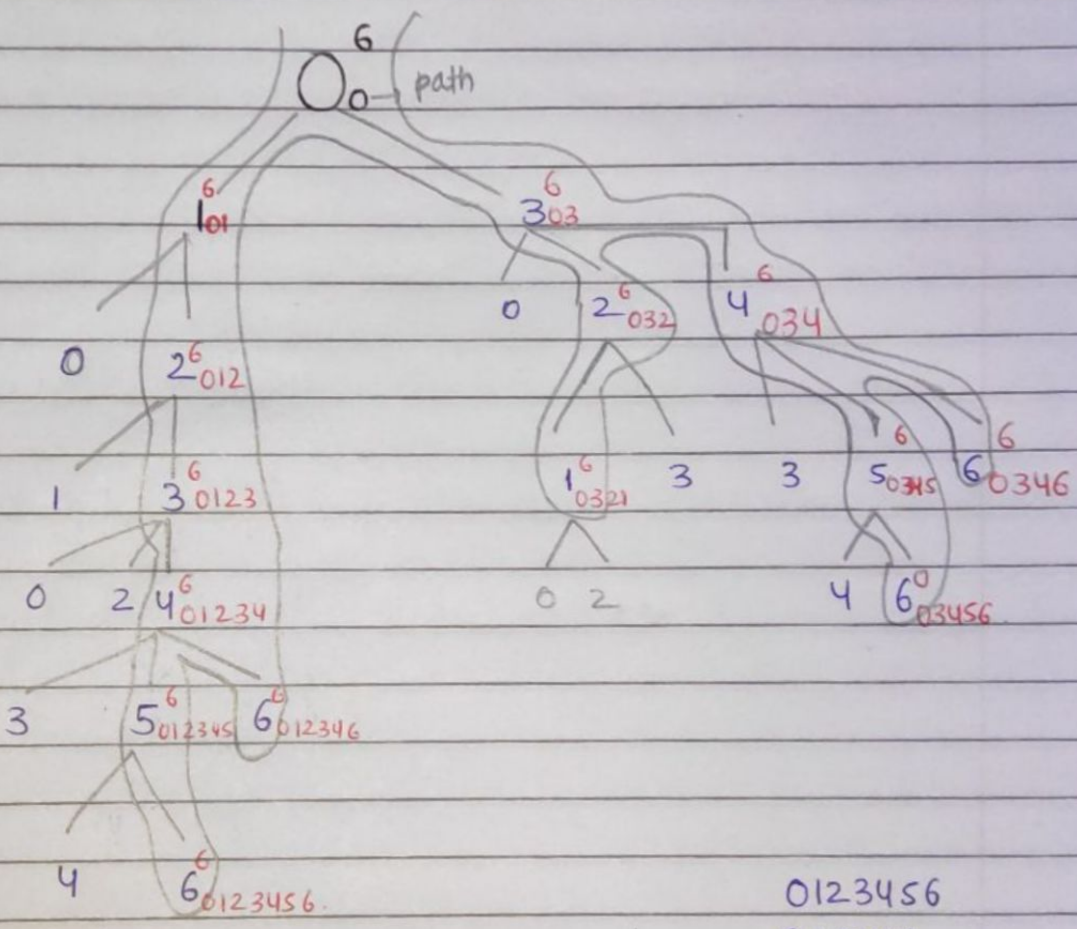
→ print all paths b/w source & destination.

→ Pre → visit
→ visit the unvisited neighbours.
→ Post → unvisit

```
boolean[] visited = new boolean[Vtces];
printAllPaths( graph, visited, src, dest, src + " " );
                                          (ose path start hoga.)
}.

          print krna h,
        ↱ return nhi
public static void printAllPaths(ArrayList<Edge>[] graph,
                              boolean[] visited, int src,
                              int dest, String psf) {
                                              (path so far)
                                              (Abhi tk ka rsta).
    if(src = =dest){
        soutln(psf);
        return;
    }

    visited[src] = true;                         //Neighbour's me loop
    for(int i=0; i< graph[src].size(); i++){
        Edge e = graph[src].get(i);
        int nbr = e.nbr;
                      unvisit nbrs ko visit krenge
        if(visited[nbr] = = false) { [nbr unvisited h tow uspr jae]
            printAllPaths(graph, visited, nbr, dest, psf + nbr + " ");
        }.
    }.
    visited[src] = false; (Post me unvisit taaki saare path
                             visit hoske).
}.
```



0→01@10 , 03@40



0123456
0,12346
03456
0346

3. MULTISOLVER -

Smallest, Longest, ceil, Floor, KthLargest Path.

I/P. 7-vertexs, 9-Edges.

I/P. 7

9

0 1 10

1 2 10

2 3 10

0 3 40

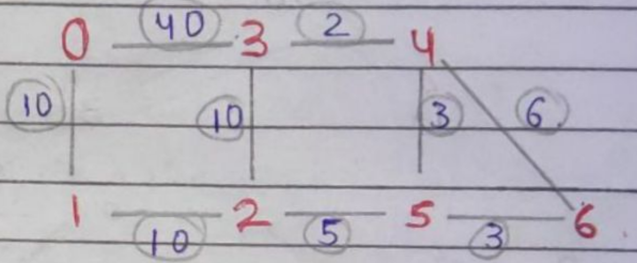
3 4 2

4 5 3

5 6 3

4 6 8

2 5 5



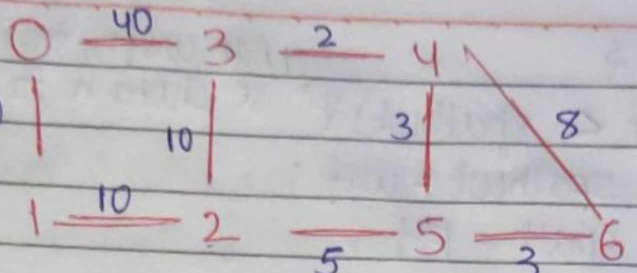
0] Ke path ① smallest path in terms of wt. 01256@28 wt.
6 ② Largest Path. 032546@66.

30 ③ Just largest Path than 30 012546@36. [-<] CEIL
4 ④ Just smaller Path than 30 01256@28 [->] FlOOR

⑤ 4th largest Path. 03456@48 (pq)

ceil - just greater (bdo me sbse chota).

floor - just smaller (choto me sbse bda).



0123456@38
012346@40
012546@36
01256@28
032546@66
03256@58
03456@48
0346@50

smallestpathwt → null@∞ (38, ∞ se km h) (wt)
~~0123456@~~ 38
~~012546@36~~, $\boxed{01256 @ 28}$.

largestpathwt → null@ ~~-∞~~
~~@38~~, ~~@40~~, $\boxed{032546@66}$

ceilpathwt → null @ ∞ (wt) (criteria(40)) se bda, 66 se chota. (criteria se bda ceil se chota).
(crieteria se bda ~~@66~~, ~~@58~~
wt se chota) @48

Floorpathwt → criteria 40 → 40 ke choto me sbse bda.
null @ -∞ @38

3rd largest →
(PQ)

~~38~~ ~~40~~ ~~36~~
66 58
50

~~36~~ ~~38~~ ~~40~~ ~~48~~ 50

```
                                                (wt.sofar)
        if (src == dest) {                      // Agr wsf , smallest pathwt
// smallest path.   if (wsf < spathwt) {         se chota h to update.
                        spathwt = wsf ;
                        spath = psf ;
                    }

// largest path.    if(wsf > lpathwt) {
                        lpathwt = wsf;
                        lpath = psf ;
                    }.

// ceilpath         if(wsf > criteria && wsf < cpathwt){
(bda me sbse            cpathwt = wsf;
 chota)                 cpath = psf;
                    }.

// floorpath         if(wsf < criteria && wsf > fpathwt){
                        fpathwt = wsf ;
                        fpath = psf ;
                     }.

                     if (pq.size() < K) {
                          pq.add(new Pair(wsf, psf));
                     }else {
                          if(wsf > pv.peek().wsf) {
                              pq.remove() ;
                              pq.add (new Pair(wsf ,psf));
                          } .
                     }.
                     return;
                }.
```

```
visited[src] = true;

for(int i =0 ; i < graph[src].size(); i++) {
    Edge e = graph[src].get(i);
    int nbr = e.nbr;

    if(visited[nbr] ==false) { unvisited nbr h to hi jaana h.
        multisolver( griph , nbr ,dest , visited , criteria, K,
                                     psf +nbr, wsf + e.wt);
    }.
}
visited[src] =false;
```