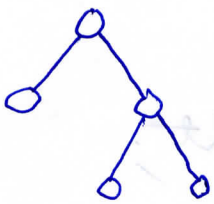
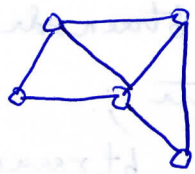
Graphs :- Non-linear data structure from many parents to many children (non-hierarchical)



TREE

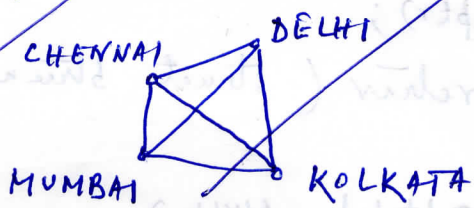


GRAPH

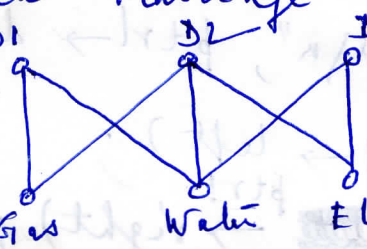
Tree is a special type of graph

Practical Problems :- (1) Cities connected through airlines

Shortest path between two cities ?



(2) Hungarian Marriage Problem :— Tracking of cells in images

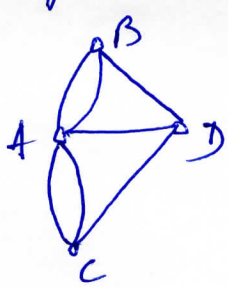


Boys

Girls

Find 1-1 pair s.t. total cost is optimum

(3) Konigsberg's Bridge Problem :—



Four lands : A, B, C, D

Seven Bridges : AB, BA, AC, CA, AD, BD, CD

Starting from any land, have to cross all bridges exactly once and return to land. → Possible ?

Definition :- ① A graph G consists of two sets :—

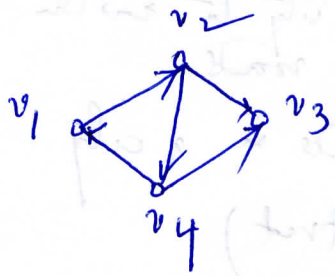
(1) A set V, called the set of all vertices / nodes

(2) A set E, ——————— edges / arcs

$G = G(V, E)$

② Order : # vertices, ③ Size : # edges

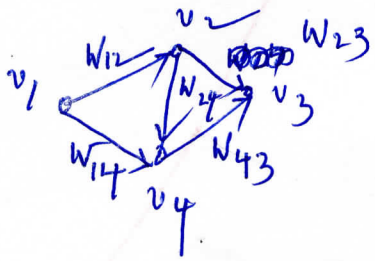
④ **Directed Graph (Digraph) :-** $G = G(V, E)$ — set of ordered pair of elements from $V^{(2)}$

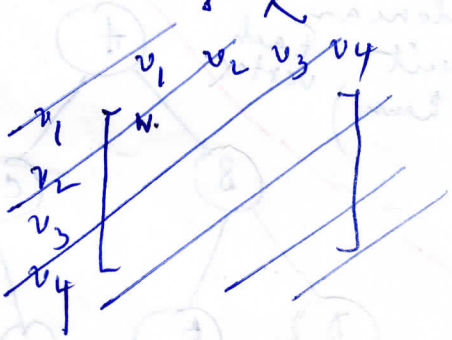


$V = \{v_1, v_2, v_3, v_4\}$

$E = \{(v_1, v_2), (v_2, v_3), (v_2, v_4), (v_4, v_1), (v_4, v_3)\}$

⑤ **Weighted Graph :-** All the edges are labelled in the graph with some weights





⑪ ⑥ **Bipartite Graph :-** $G = (V_1 \cup V_2, E)$

V_1 and V_2 are disjoint

Every edge in E joins a vertex in V_1 to a vertex in V_2

⑨ ⑦ **Path :-** Sequence of vertices is connected by a path. (*)

⑧ **Representation of Graphs :-**

Let n be the no. of vertices ⇒ n×n matrix is used (Adjacency matrix)

	v_1	v_2	...	v_n
v_1				
v_2				
⋮				
v_n				

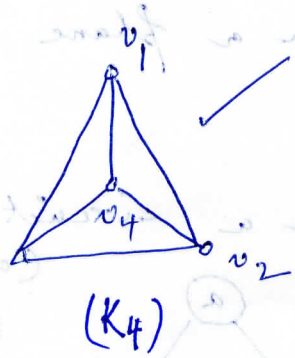
$a_{ij} = 1$ if ∃ edge from v_i to v_j

0 otherwise

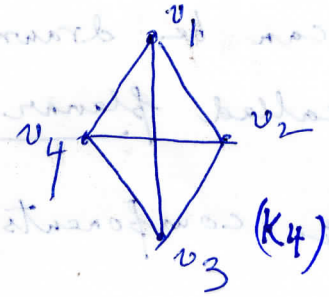
⑩ ⑨ (*) ⑧ **Cycle :-** Path $v_0, v_1, \ldots, v_n$, $n \geq 3$, $v_0 = v_n$.

(6) **Complete Graph :—**

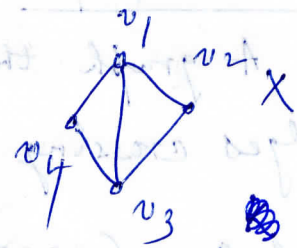
A graph (digraph) G is said to be complete if each vertex v_i is adjacent to every other vertex v_j in G (i.e. there is an edge from v_i to every v_j)



(K_4)



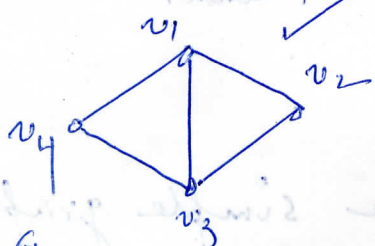
(K_4)



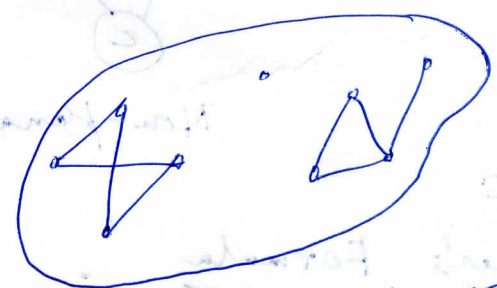
X

(10) **Connected Graph**

A graph is said to be connected if there is a path in G for every pair of distinct vertices v_i, v_j

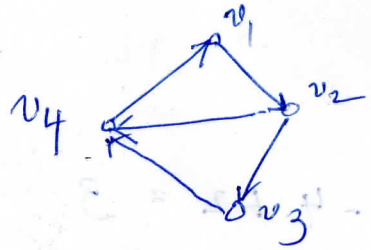


G



X

A digraph is strongly connected if for every pair of distinct vertices v_i, v_j in G, there is a path from v_i to v_j and also from v_j to v_i



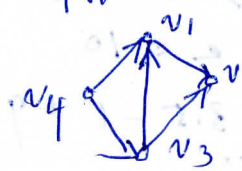
X

(7) (8) **Degree of a vertex**

The number of edges connected with vertex v_i is called the degree of vertex v_i.

For a digraph → indegree : No. of edges incident into v_i

→ outdegree : No. of edges emanating from v_i.



indegree $(v_1) = 2$

outdegree $(v_1) = 1$

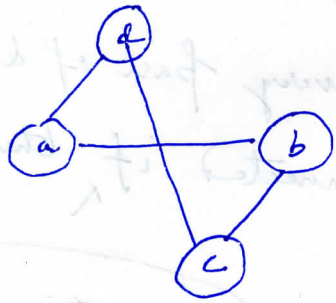
and so on

Various Graph Problems :—

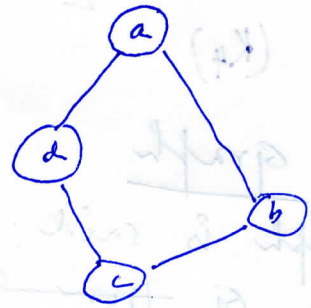
(12) Planar Graph

A graph that can be drawn on a plane without edges crossing is called planar.

e.g. Connecting components on a circuit board. (cheaper if no intersection)



Non-planar

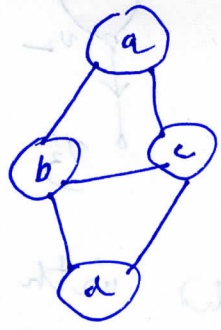


Planar

(13) Euler's Formula

For a connected planar simple graph $G = (V, E)$ with $e = |E|$ and $v = |V|$, if we let r be the no. of regions that are created when drawing a planar representation of the graph, then $r = e - v + 2$.

e.g.



$v = 4$

$e = 5$

$r = 5 - 4 + 2 = 3$.

K_5.

Testing for planarity

For a simple connected planar graph with $v \geq 3$ vertices and e edges, $e \leq 3v - 6$. e.g. $K_5 \rightarrow v = 5$, $e = 10$

$3v - 6 = 3 \cdot 5 - 6 = 9$

$\therefore e > 3v - 6$ it is non-planar.

Elementary Graph Algorithms

Breadth-First-Search (BFS)

Depth-First-Search (DFS)

Representations of Graphs

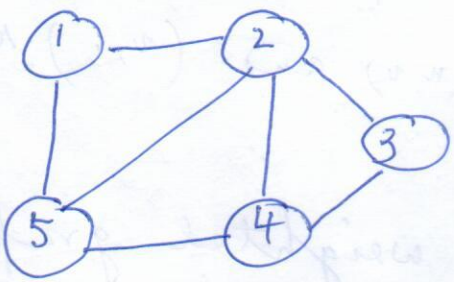
$G = (V, E)$

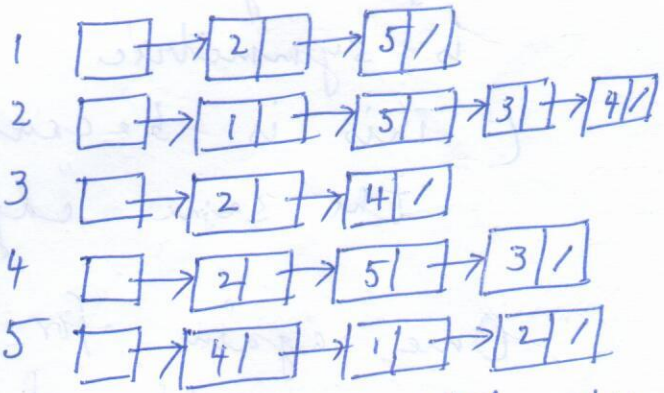
- Collections of adjacency lists : Better for sparse graphs $|E| \ll |V|^2$
- Adjacency Matrix : Better for dense graph $|E| \sim |V|^2$

Adjacency List :-

Consists of an array Adj of $|V|$ lists, one for each vertex V. For each $u \in V$, the adjacency list Adj[u] contains all the vertices v s.t. there is an edge $(u, v) \in E$.

e.g.





Sum of the lengths of all adjacency lists = 14

$= 2 \times 7 = 2 \times |E|$ ((u, v) is an undirected edge, u appears in v's adj. list and vice-versa)

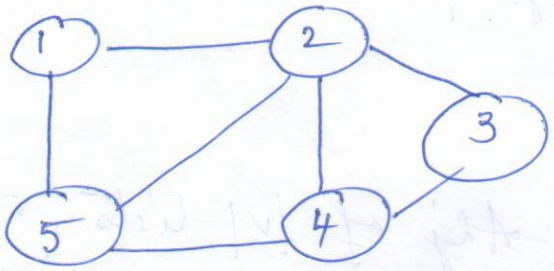
For a directed graph, the above sum = $|E|$.

For a weighted graph, we can store the weight with vertex v in u's adjacency list.

Adjacency - Matrix Representation

Assume that the vertices are numbered $1, 2, \cdots, |V|$ in some manner. Then the adj. matrix $A = (a_{ij})$ of size $|V| \times |V|$ is given by :—

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Adjacency matrix of an undirected graph is symmetric i.e. $A = A^T$

(This is because (u,v) and (v,u) represent the same edge.)

Once again for the weighted graph, we can directly put the weights in the adjacency matrix.

Breadth-first search

Given a graph $G = (V, E)$ and a distinguished source vertex s, breadth-first search explores the edges of G to find every vertex that is reachable from s.

Principle :- The algorithm discovers all vertices at distance k from s before discovering any vertices at distance $k+1$.

Strategy :- BFS colours each vertex white, gray, or black. Initially, all vertices are white. When a vertex is encountered for the first time (= discovered), it becomes non-white. All vertices adjacent to a black vertex have been discovered but this is not true for a gray vertex.

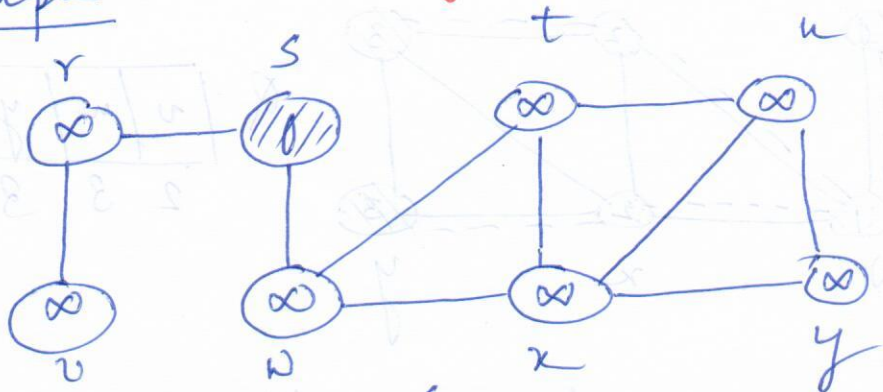
Predecessor / Parent :- Whenever a white vertex v is discovered while scanning the adjacency list of an already discovered vertex u, the vertex v and the edge (u, v) are added to the BFS-tree. We say u is the parent of v in the BFS-tree. $\pi[v] = u$

Distance from the source s to any vertex u is stored in $d[u]$.

```
BFS (G, s)
1  for each vertex u ∈ V[G] - {s}                    INITIALIZE VERTICES
2      do color[u] ← WHITE                           OTHER THAN ROOT
3         d[u] ← ∞
4         π[u] ← NIL
5  colour[s] ← GRAY                                  INITIALIZE
6  d[s] ← 0                                          ROOT
7  π[s] ← NIL
8  Q ← φ                                             INITIALIZE
9  ENQUEUE (Q, s)                                    QUEUE
10 while Q ≠ φ                                       TILL THERE ARE GRAY VERTICES
11     do u ← DEQUEUE (Q)
12        for each v in Adj[u]
13            do if colour[v] = WHITE
14               then colour[v] = GRAY
15                    d[v] = d[u] + 1
16                    π[v] ← u
17                    ENQUEUE (Q, v)
18        colour[u] ← BLACK
```

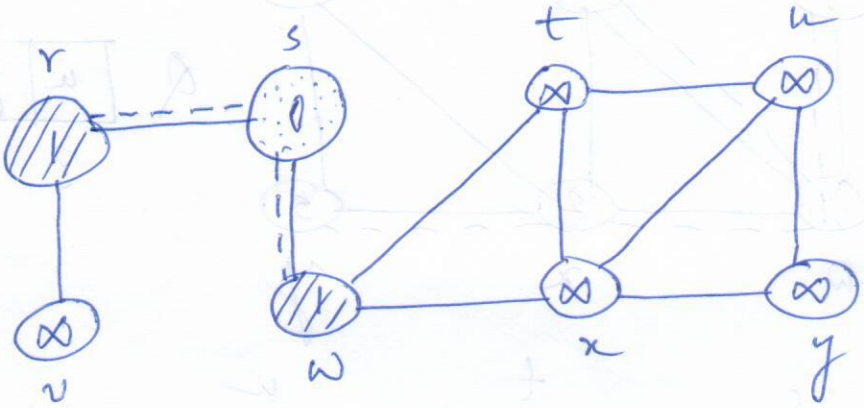
○ : White ◎ Gray ● Black
---- : Edges in BFS-tree

Example

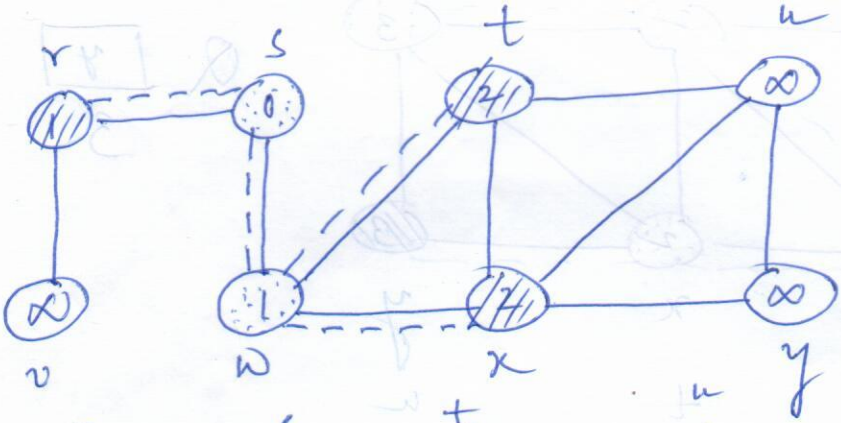


Q: s | 0

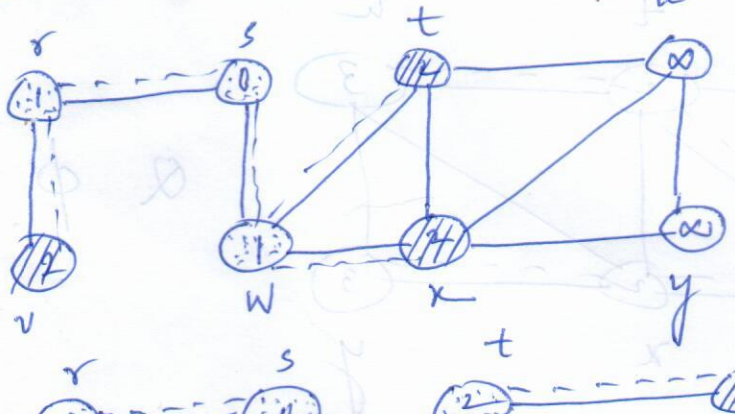
Adj[s] → r, w



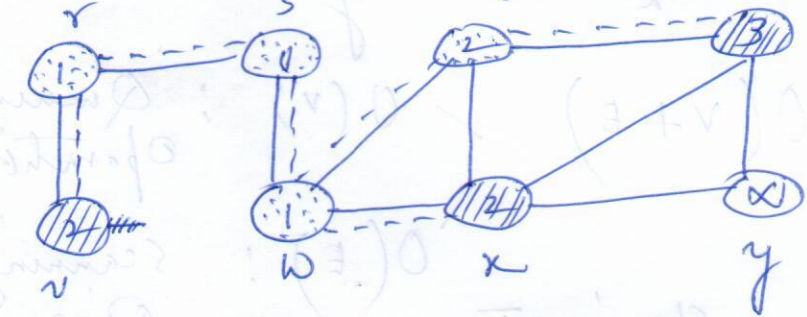
Q: w | r | 1 | 1



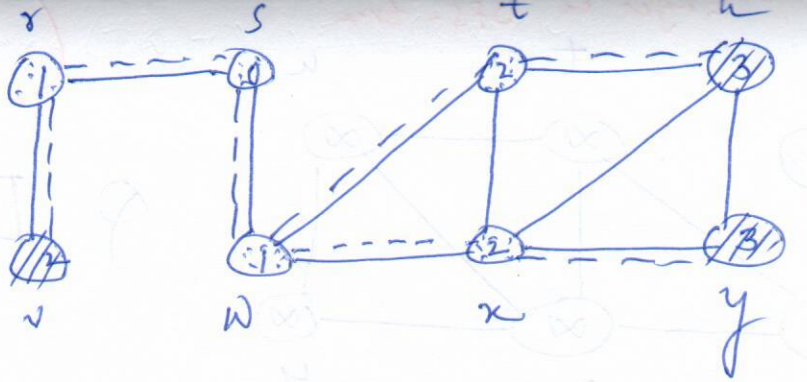
Q: r | t | 2 | 1 | 2 | 2



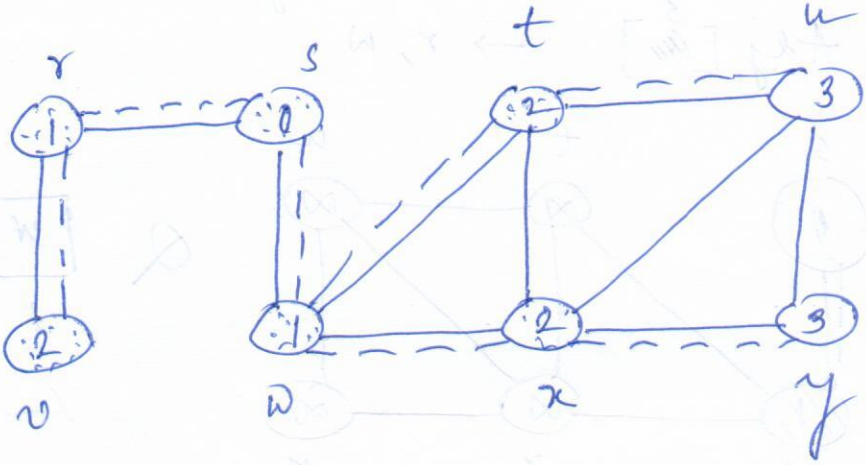
Q: t | x | v | 2 | 2 | 2



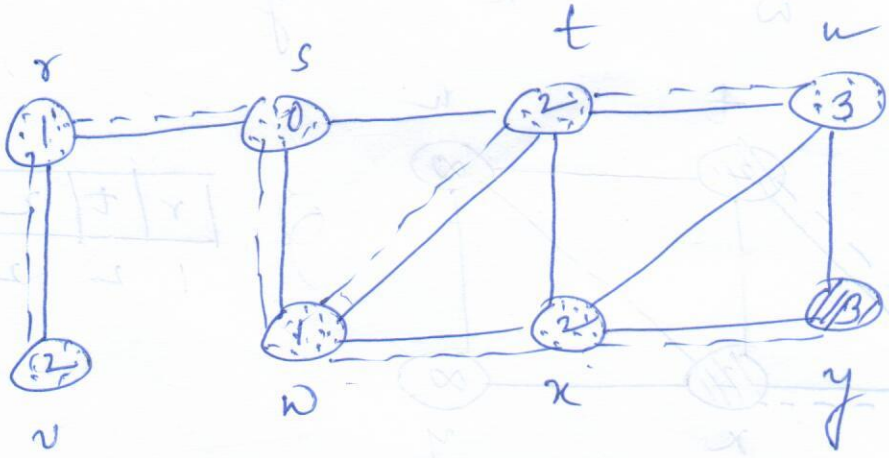
Q: x | v | u | 2 | 2 | 3



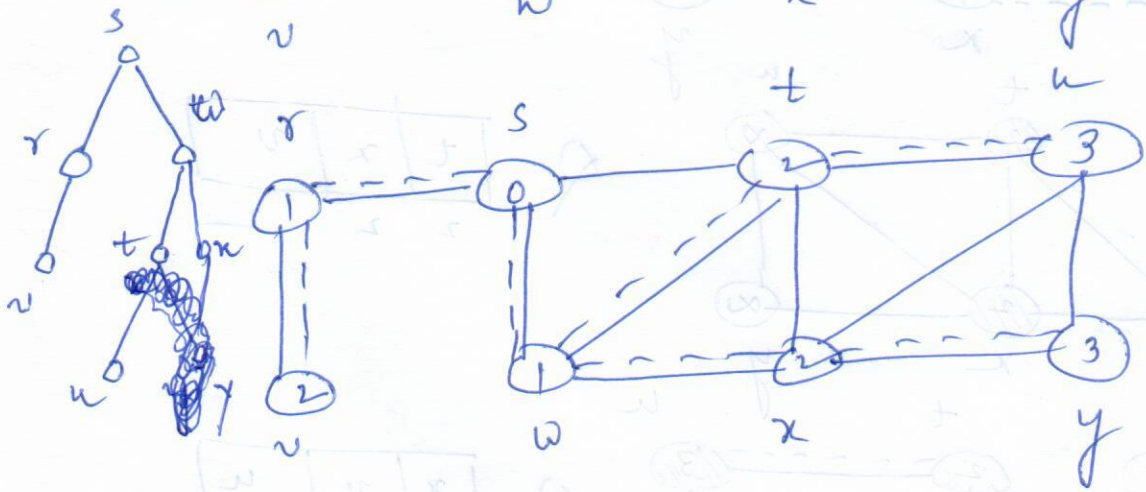
Q | v | u | y |
2 3 3



Q | u | y |
3 3



Q | y |
3



Q ϕ

Complexity : $O(V+E)$ — $O(V)$: Queuing operations
$O(E)$: Scanning adjacency list

Applications :—

Prim's Algo. → Minimum Spanning Tree
→ Single Source shortest Path Algo

Depth-First Search (DFS) :-

Principle :- Edges are explored out of the most recently discovered vertex v that still has unexplored edges leaving it. When all of v's edges have been explored, the search 'backtracks' to explore edges leaving the vertex from which v was discovered. The process continues until we have discovered all the vertices that are reachable from the original source vertex.

The predecessor sub-graph $G_\pi = (V, E_\pi)$ where E_π are tree-edges ↑ Edges in

$E_\pi = \{ (\pi[v], v) : v \in V \text{ and } \pi[v] \neq NIL \}$

→ Forest, composed of several trees, as the search may be repeated from multiple sources (subroutine in other algo.)

vs. Tree, in case of BFS as BFS is limited to a single source. (shortest path distances)

Timestamps (white) ↑

Each vertex v has two timestamps :—

- the first timestamp $d[v]$ when v is first discovered (grayed)
- the second timestamp $f[v]$ when the search finishes examining v's adjacency list (blackened)

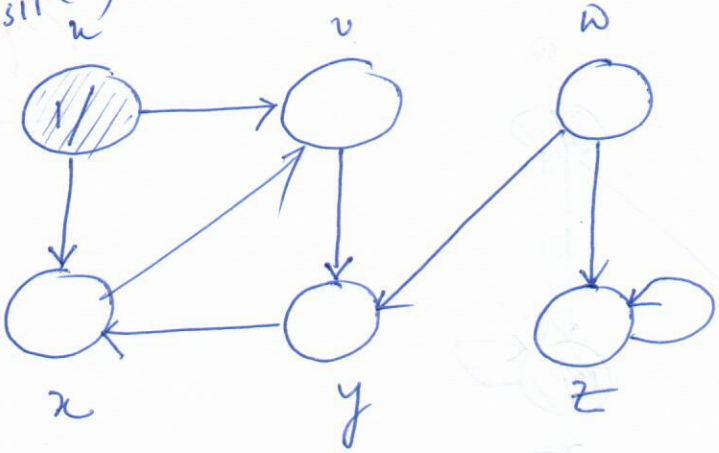
$d[v] < f[v]$

Vertex v : White before $d[v]$

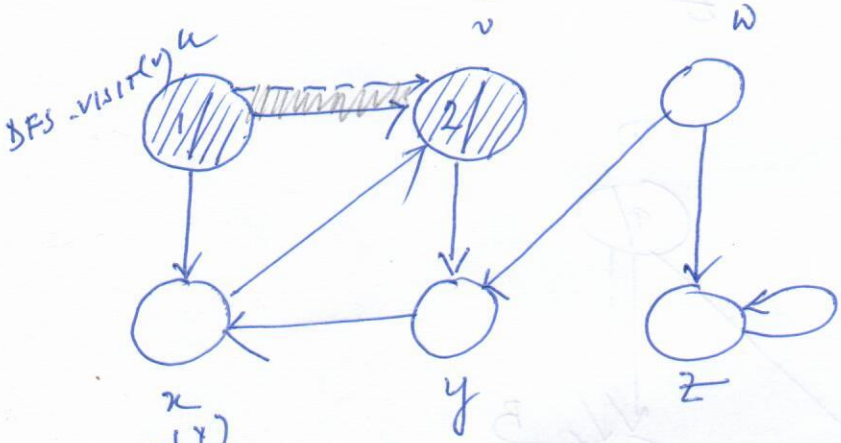
```
DFS(G)
1  for each vertex u ∈ V[G]
2      do colour[u] ← WHITE
3         π[u] ← NIL
4  time ← 0
5  for each vertex u ∈ V[G]
6      do if colour[u] = WHITE
7         then DFS-VISIT(u)

DFS-VISIT(u)
1   colour[u] ← GRAY
2   time ← time + 1
3   d[u] ← time
4   for each v ∈ Adj[u]
5       do if colour[v] = WHITE
6          then π[v] ← u
7               DFS-VISIT(v)
8   colour[u] ← BLACK
9   f[u] ← time ← time + 1
```

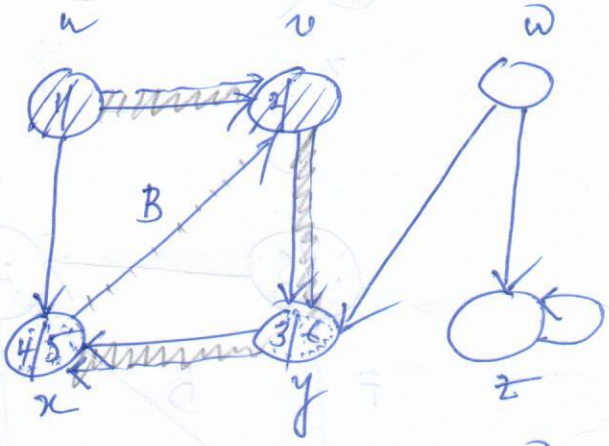
Ex.

DFS-VISIT(u)

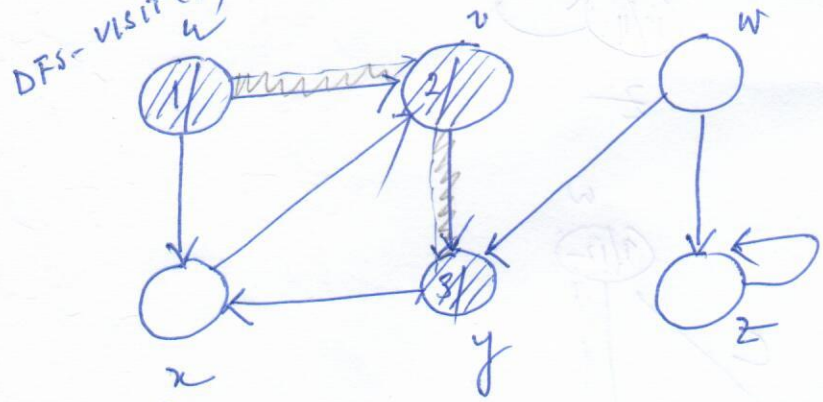


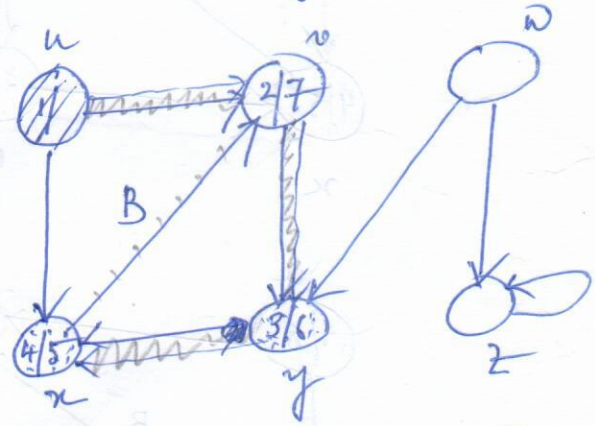
DFS-VISIT(v)



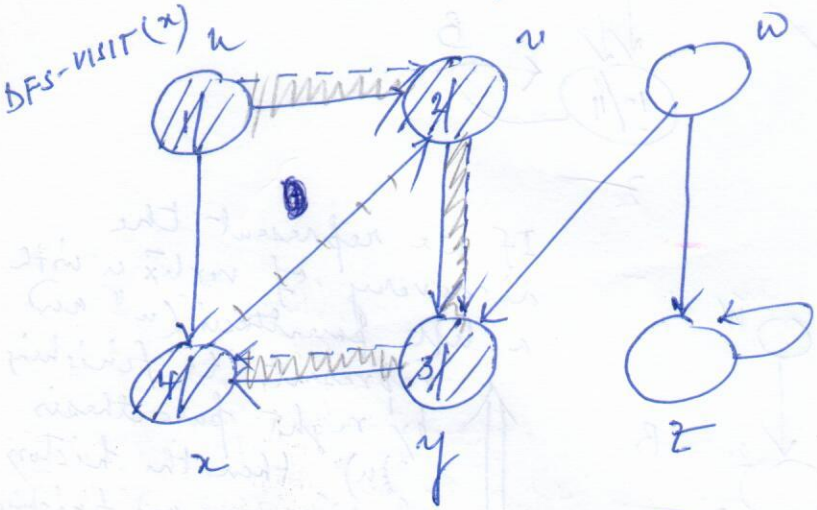


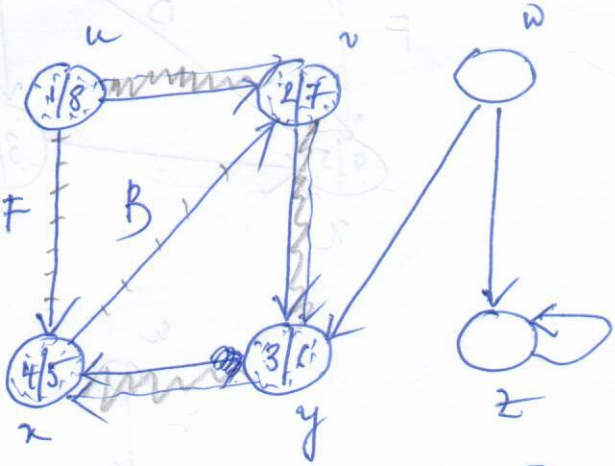
DFS-VISIT(y)



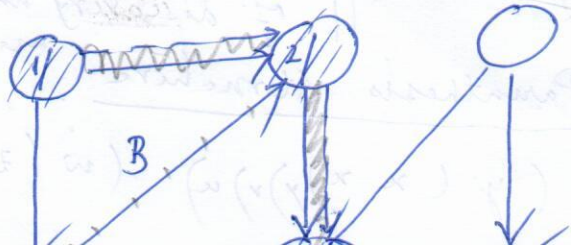


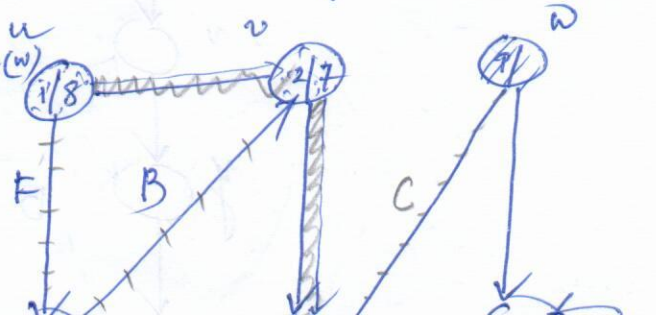
DFS-VISIT(x)



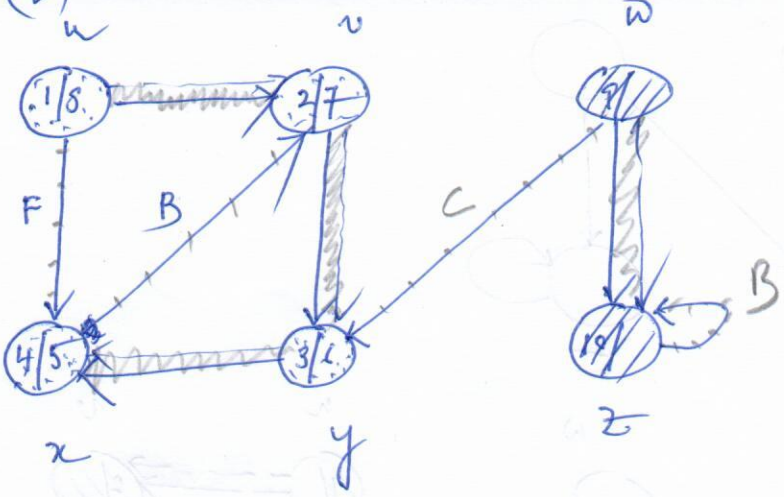


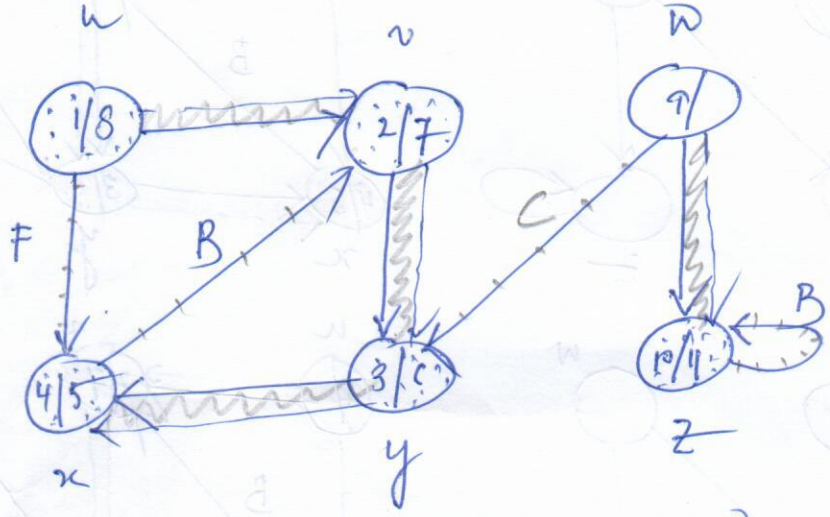
DFS-VISIT(w)

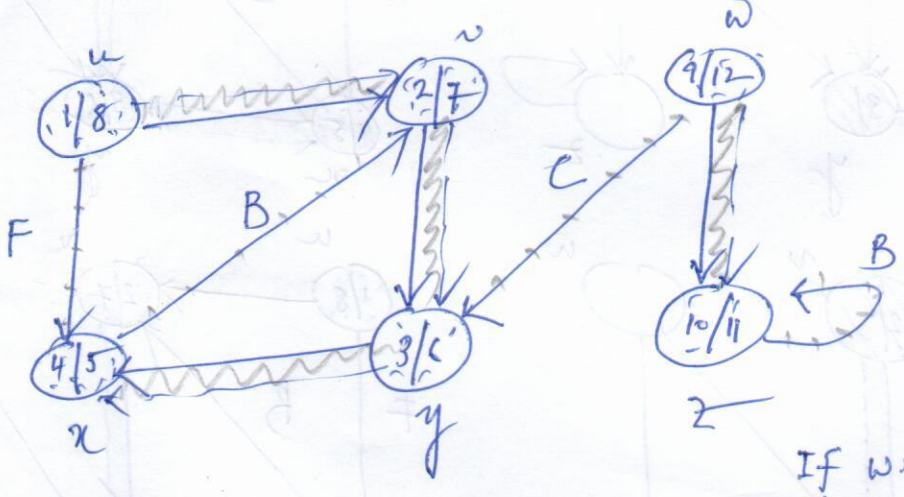


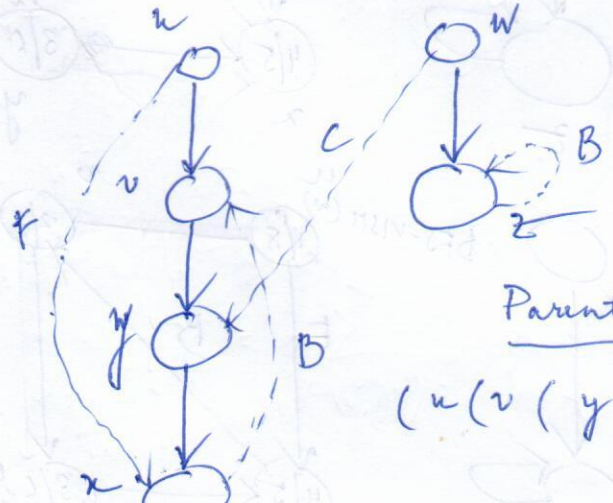


DFS-VISIT(z)









If we represent the discovery of vertex u with a left parenthesis "(u" and represent its finishing by right parenthesis "u)" then the history of discovery and finishings are properly nested.

Parenthesis structure

(u(v(y(x x)y)v)u) (w(z z)w)

Classification of Edges

Four types of edges in a directed graph : — the depth-first forest G_π from

(1) Tree-edges are edges in the depth-first forest G_π. Edge (u,v) is a tree edge if v was first discovered by exploring (u,v).

(2) Back edges are those edges (u,v) connecting a vertex u to an ancestor v in a depth-first tree. Self-loops are back edges.

(3) Forward edges are those nontree edges (u,v) connecting a vertex u to a descendant v in a depth-first tree.

(4) Cross edges are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

Complexity of DFS : —

$\Theta(V+E) = O(V+E)$

DFS-VISIT called exactly for each vertex $v \in V$ $\Theta(V) \Rightarrow O(V)$

DFS-VISIT(v) executed $|Adj[v]|$ times $\Theta(E) \Rightarrow O(E)$

Applications of DFS

① A topological sort of a directed acyclic graph (dag) $G = (V, E)$ is a linear ordering of all its vertices s.t. if G contains an (u, v), then u appears before v in the ordering.

② Strongly connected component of a directed graph $G = (V, E)$ is a maximal set of vertices $C \subseteq V$ s.t. for every pair of vertices (u, v) in C, we have both $u \rightsquigarrow v$ and $v \rightsquigarrow u$. (reachable from one another)

Applications of BFS

① Minimum Spanning Tree :—

An acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight $w(T) = \sum_{(u,v) \in T} w(u, v)$ is minimized.

② Shortest Path Problem :—

Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to R$. The weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges :—

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

A shortest path from u is defined as any path p with weight $w(p) = \delta(u, v)$

shortest path $\delta(u, v) = \begin{cases} \min \{ w(p) : u \rightsquigarrow v \text{ if there is a path from } u \text{ to } v \\ \infty \text{ otherwise} \end{cases}$