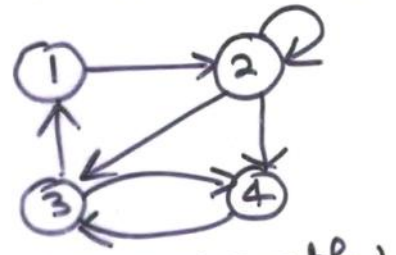
Graph

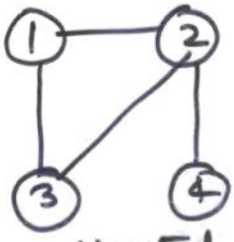
Graphs = a set of nodes (vertices) with edges (links) between them

Notation :-

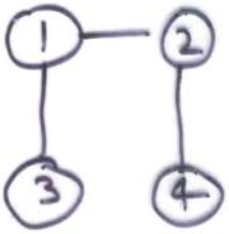
* $G = (V, E)$ - graph
* V = set of vertices $|V| = n$
* E = set of edges $|E| = m$



(Directed graph)



(undirected graph)



Acyclic graph.

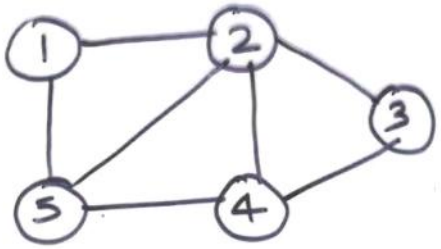
Graph representation

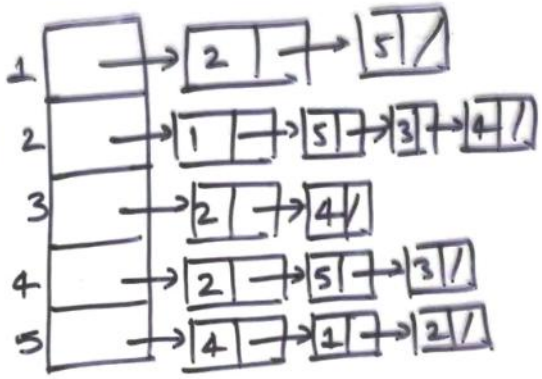
Adjacency link representation :-

- An array of $|V|$ lists, one for each vertex in V
- Each list adj[u] contains all the vertices v such that there is an edge between u and v

Note: Adj[u] contains the vertices adjacent to u (in arbitrary order).

- can be used for both directed and undirected graphs



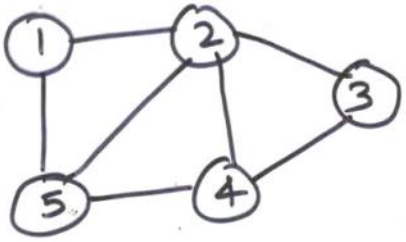


→ Adjacency matrix representation of $G=(V,E)$

— Assume vertices are numbered $1, 2, \ldots |V|$

— the representation consists of a matrix $A_{|V| \times |V|}$

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



	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

Searching a graph.

Graph searching = systematically follow the edges of the graph so as to visit the vertices of the graph

→ Two basic graph searching algorithms
- breadth-first search
- Depth-first search

- The difference between them is in the order in which they explore the unvisited edges of the graph.

Breadth-first-search (BFS)

Input:
- A graph $G = (V, E)$ (directed or undirected)
- source vertex $s \in V$

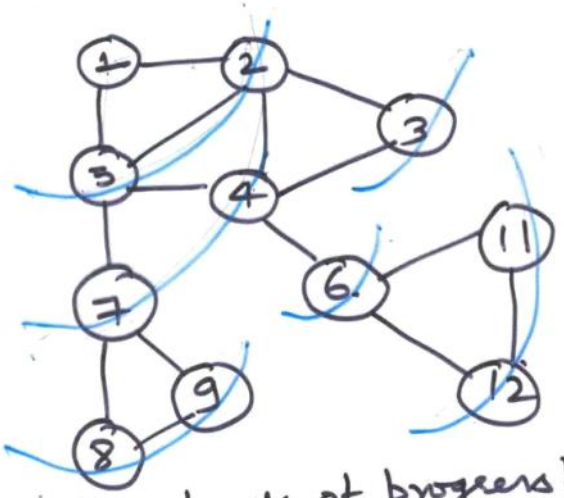
Goal:
- Explore the edges of G to "discover" every vertex reachable from s, taking the ones closest to s first

Output:
- $d[v]$ = distance (smallest # of edges) from s to v, $\forall v \in V$
- A "breadth-first-tree" rooted at s that contains all reachable vertices

Breadth-first-search

→ Discover vertices in increasing order of distance from the source s- search in breadth not depth

*→ find all vertices at 1 edge from s, then all vertices at 2 edges from s, and so on

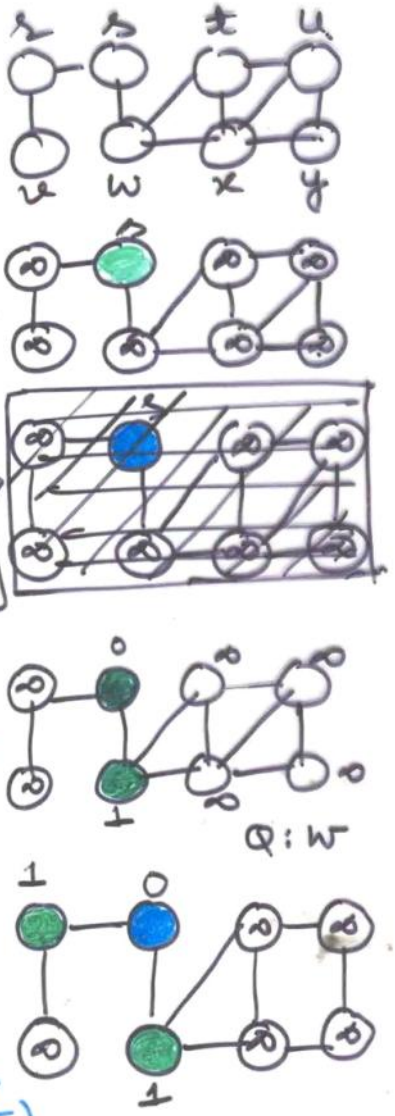


* { keeping track of progress }
- Color each vertex is either white gray or black
- Initially all vertices are white
- when being discovered a vertex becomes gray
- → after discovering all it's adjacent vertices, node becomes black
- use FIFO queue Q to main gray vertices

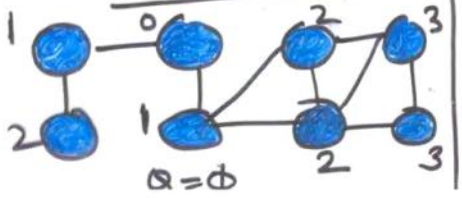
BFS (V, E, s)

```
1.  for each u ∈ V-{s}                          O(V)
2.      do Colour[u] ← White
3.         d[u] ← ∞
4.         π[u] = NIL
5.  Color[s] ← Gray                             Θ(1)
6.  d[s] ← 0
7.  π[s] = NIL
8.  Q ← ϕ
9.  Q ← ENQUEUE(Q, s)
10. while Q ≠ ϕ
11.     do u ← DeQueue(Q)  ← Θ(1)
12.        for each v ∈ Adj[u]  → *
13.            do if color[v] = white
14.               then Color[v] = gray
15.                    d[v] ← d[u]+1
16.                    π[v] = u
17.                    ENQUEUE(Q,v)  ← Θ(1)
18.    Colour[u] ← Black
```

Complexity $O(V+E)$



final figure



Colour used in SLIDES

- ● → black color
- ● → Gray

* sum of length of adj arrays = $\Theta(E)$

(5)

Depth-first search

Input:
- $G = (V, E)$ (No source vertex given)

Goal:
- Explore the edges of G to "discover" every vertex in V starting at the most currently visited node
- search may be repeated from multiple sources

Output:
- 2 timestamp on each vertex
  - $d[v]$ = discovery time
  - $f[v]$ = finishing time

DFS(V,E)

```
1. for each u ∈ V                    } Θ(V).
2.     do color[u] ← white
3.        π[u] ← NIL
4. time ← 0
5. for each u ∈ V                    } Θ(V) - exclusive of time for DFS_visit
6.     do if color[u] = white
7.        then DFS-visit(u)
```

→ Every time DFS-visit(u) is called, u becomes the root of a new tree in the depth-first forest

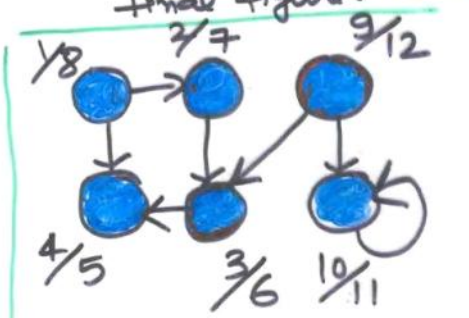
DFS-visit(u)

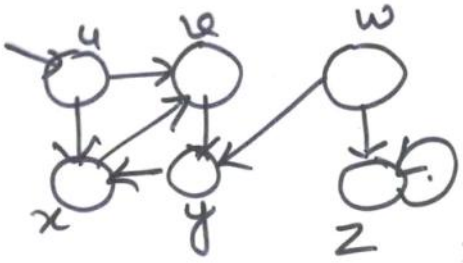
```
1. Color[u] ← GRAY
2. time ← time+1
3. d[u] ← time
4. for each v ∈ Adj[u]
5.     do if color[v] = WHITE
6.         then π[v] ← u
7.              DFS_visit(v)
8. Color[u] ← BLACK
9. time ← time+1
10. f[u] ← time
```

each loop takes $|Adj[v]|$.

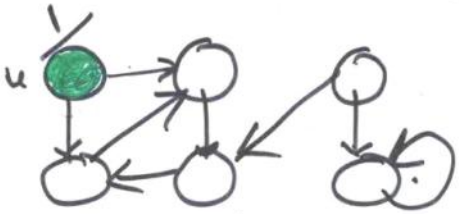
Total = $\sum_{u \in V} |Adj[v]| + \Theta(V) = \Theta(V+E)$

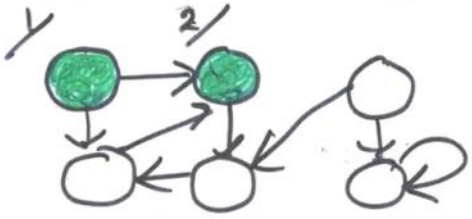
final figure.





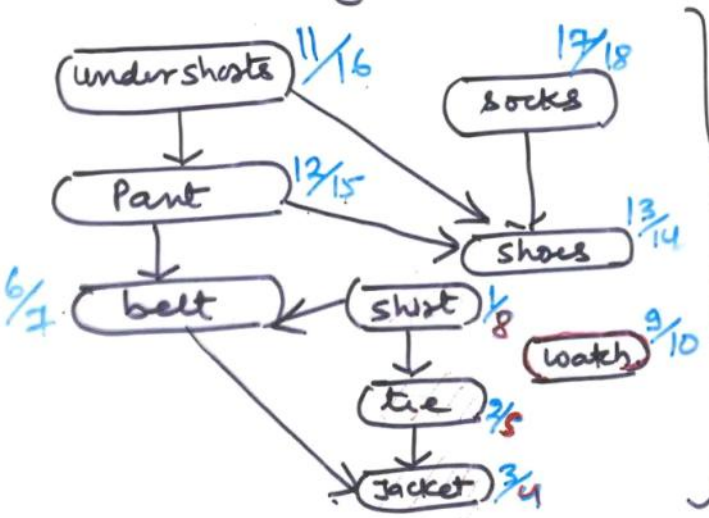
time = 1





Topological sort

Topological sort of a directed ~~graph~~ acyclic graph $G = (V, E)$: a linear order of vertices such that if $\exists$ an edge (u, v), then u appears before v in the ordering

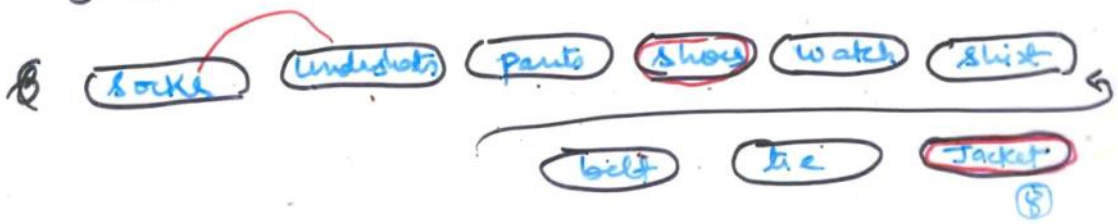


Topological sort : an ordering of vertices along a horizontal line so that all directed edges go from left to right

Topological-sort (V, E)

1. Call DFS (V, E) to compute finishing time for each vertex
2. When each vertex is finished, insert it onto the front of a linked list
3. Return the linked list of vertices

$\Theta(V+E)$



Strongly Connected Components

Given directed graph $G = (V, E)$

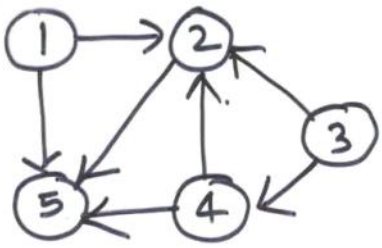
A strongly connected components (SCC) of G is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices $U, V \in C$, we have both $U \Rightarrow V$ and $V \Rightarrow U$

* Note the following terms

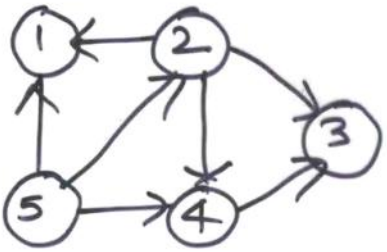
G^T = transpose of G

→ G^T is G with all edges reversed

→ $G^T = (V, E^T)$, $E^T = \{(U,V) : (V,U) \in E\}$.



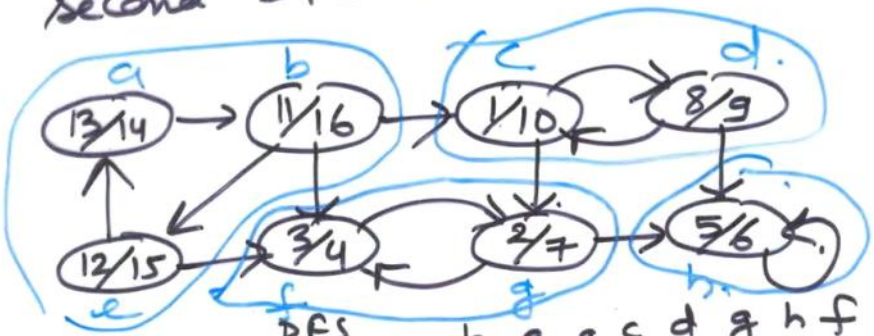
G



G^T

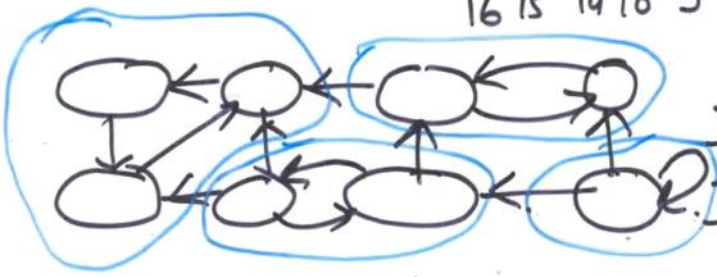
STRONGLY Connected Components (G)

1. Call DFS(G) to compute finishing time f[u] for each vertex u.
2. Compute G^T
3. Call DFS(G^T), but in the main loop of DFS, consider vertices in order of decreasing f[u]
   (as computed in the first DFS)
4. Output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC



DFS

b e a c d g h f
16 15 14 10 9 7 6 4



DFS on G^T.

→ start at b: visit a, e
→ start at c: visit d
→ start at g: visit f
→ start at h.