Tuesday, March 22, 2022 10:53 AM

Graph Algorithms:

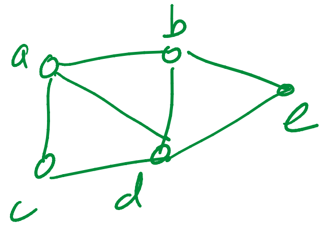
A graph G is (V, E) $\quad |V| = n$, $|E| = m$

V = set of vertices

E = set of edges of the form (u,v) ← directed

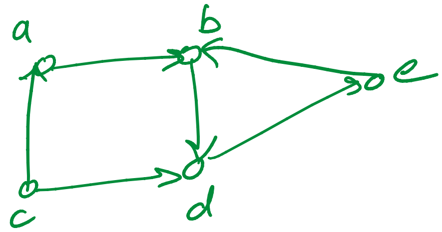
uv ← un-directed. (or $\{u,v\}$)

Ex: Un-directed



$Adj(a) = \{b\}$

$Adj(c) = \{a, d\}$



$V = \{a, b, c, d, e\}$

$E = \{(a,b), (c,a), (b,d), (c,d), (d,e), (e,b)\}$

$E = \{ab, ac, bd, ad, cd, be, de\}$

Appln: Social n/w, road n/w, inter/hyper-link graph

Obs: $(n-1) \le m \le O(n^2)$

if G is connected. (exe: A undirected tree (no cycle) has $(n-1)$ edges)

☆ Basic Concepts: path, connected, cycles, ...

☆ Representation

1. Adjacency Matrix: A $n \times n$ matrix

$A[u,v] = 1$ if $(u,v) \in E$
$\quad\quad = 0$

(if G undirected

(dense graphs)

$A[u,v]$

$= 0$

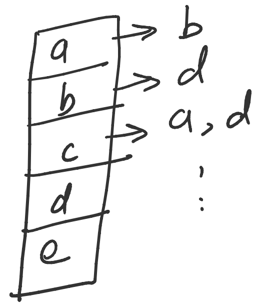
(if G undirected A is symmetric)

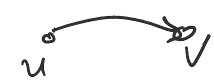
(dense graphs)

$\Theta(n^2)$ space. lookups of edges are $\Theta(1)$

2. Adjacency list :

For each node $u \in V$, $Adj(u) = \{v \mid (u,v) \in E\}$





$$\text{space: } O\Big(n + \sum_{u \in V} \underbrace{|Adj(u)|}_{\text{out-deg}(u)}\Big)$$

$$= O\Big(n + \sum_{u \in V} \text{out-deg}(u)\Big)$$

$$= O(n+m)$$

(Lookup time could be $O(n)$)

(graph is sparse e.g. $m \approx O(n)$)

Basic Question:

- Is there a path from s to t?
- Reachable nodes from s?
- Is G connected?
- Find all connected components of G?

} undirected

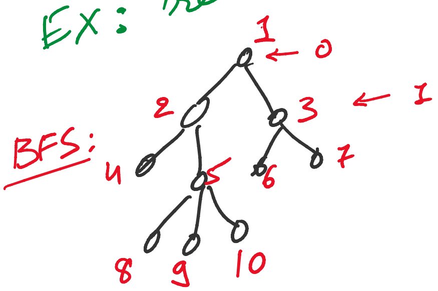
⋮

Basic Search Algorithms:

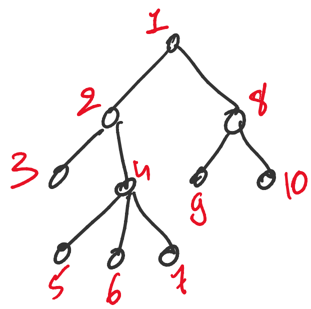
Breadth First Search (BFS)

Depth First Search (DFS)

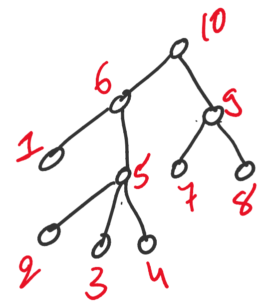
Ex: Tree



DFS



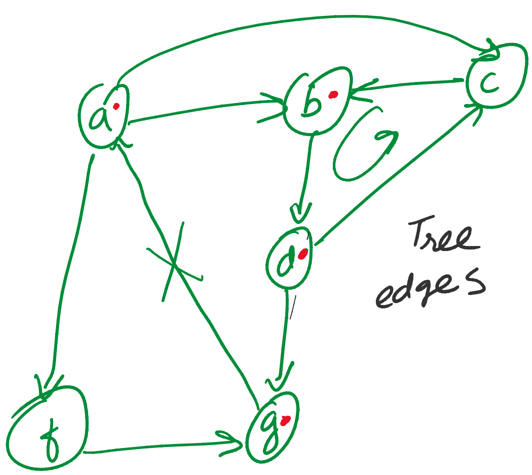
Discovery Order = pre-order



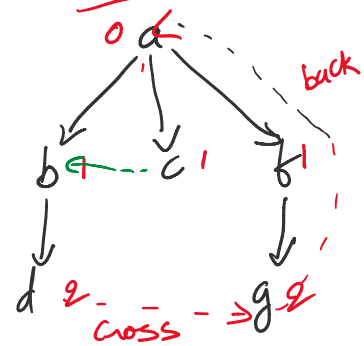
Finish Order. = Post-order.

☆ Graphs (Extension):

Main Idea: Do not revisit a vertex, if seen it before.

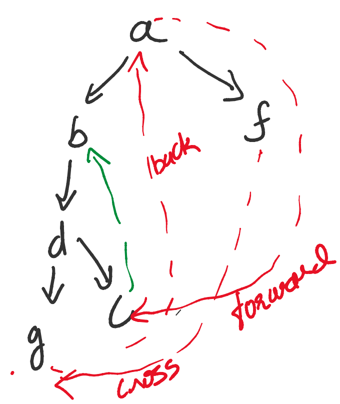


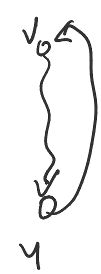
BFS



No forward edges

DFS





☆ Non-tree edges:

back: from a node to it's ancestor

forward: from a node to it's successor.

cross: All other non-tree edges.

Undirected:

Exe: what edges are possible / not-possible forward- ...

cross: All other non- tree edges. | not-possible

☆ Implementation:

BFS (G, s) $s \in V$

// idea 1: Mark visited vertices.

// idea 2: Use a queue data structure Q ...

Q: a

Q: b c f

Q: c f d

1. for each $u \in V$, do unmark u.
2. Insert s in Q. Mark s. level[s] = 0
3. while $Q \neq \emptyset$ {
4. remove a head vertex $\underline{u}$ from Q.
5. for each $\underline{v} \in Adj(u)$, do {
6. if v is unmarked.
7. insert v in Q at tail/end. Mark v.
8. Parent[v] = u. level[v] = level[u] + 1

(Meaning of Q is: nodes that are discovered but their neighbors are yet to be discovered)

Running Time: lines 5-7 $O(|Adj(u)|)$

Total time: $O\left(n + \sum_{u \in V} |Adj(u)|\right)$

$= O(n + m)$

global time = 0

☆ DFS (G, s) {

// Similar, only different data structure: Stack

OR Recursion.

| DFS-all (G)

// similarly

OR Recursion.

→ 1. Mark s. discover[s] = time++
2. for each $v \in Adj(s)$ do
3. if v is unmarked.
4. DFS(G, v); parent[v] = s;
5. finish[s] = time++;

Running Time : $O(n+m)$

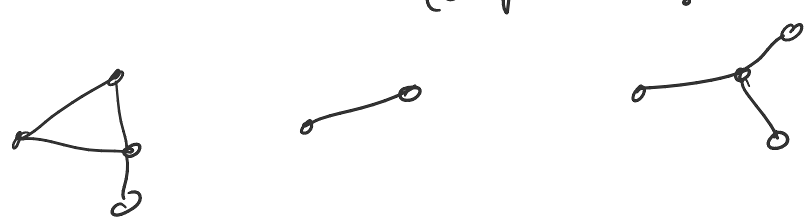
DFS-all (G)
0. unmark all vertices
1. for each $v \in V$
2. if v is unmarked
3. DFS(G, v)

★ Applications:

Ex 1: Given undir/dir G, $s, t \in V$
find shortest path from s to t.

1. Run BFS from s.
2. level [t] is the shortest path distance $s \to t$
s-t path in BFS tree is the shortest path.

Ex 2: Undirected G. Find all connected components.



→ Run DFS-all (G)

→ O/P DFS trees.

Ex 3: Given G un dir, decide if G has a cycle.

→ Run DFS/BFS.

→ check if $\exists$ a non-tree edge.

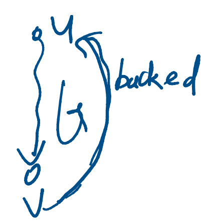
Ex 4: " dir " .

→ Run DFSAll(G)

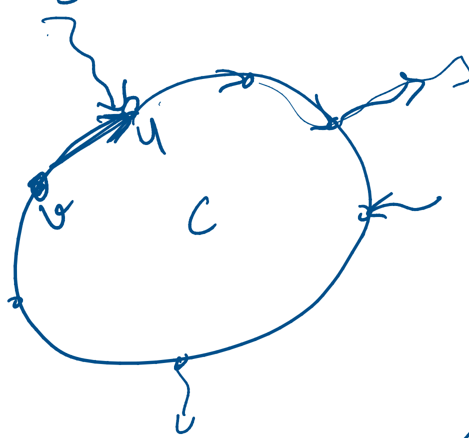
→ If $\exists$ a back edge $\Leftrightarrow$ $\exists$ a cycle.

Proof of Correctness:

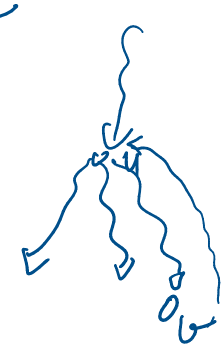
$\exists$ a backedge $\Rightarrow$ $\exists$ a cycle.



$\exists$ a cycle C $\Rightarrow$ $\exists$ a backedge?



- Let u be the first vertex on C encountered by DFS
- & v be the vertex immediately before u on C.



$\Downarrow$

is a back edge in DFS.

$v \longrightarrow u$ is a back edge in DFS.