LECTURE 6: 1/30

Recall BFS from last class

DISCOVERED[u] = false for $u \neq s$

DISCOVERED[s] = true

$L[0] \leftarrow s$; $i \leftarrow 0$

WHILE L[i] IS NOT EMPTY

$L[i+1] \leftarrow \phi$

For each vertex $u \in L[i]$

For each neighbor v of u ($v \in A[u]$)

If DISCOVERED[v] = true, then

do Nothing.

Else,

Set DISCOVERED[v] $\leftarrow$ true,

Add v TO $L[i+1]$.

$i \leftarrow i+1$.

→ CHECK IF DISCOVERED[t] = true.

Run-time of BFS:

Claim: Using BFS to solve s-t connectivity takes $O(|V|+|E|)$ time.

Proof: The for loop gets run at most once for each vertex.

$\Rightarrow$ # total runs of the outer FOR loop is at most $|V|$.

Inner-For loop: # of iterations is at most degree of u.

$\leq |V|$.

$\leq |E|$.

$\Rightarrow$ Total run-time

$\leq$ (max # Outer for loop iterations).

(max # inner for loop iterations).

(time per iteration)

$\leq |V| \cdot |E| \cdot O(1) = O(|V| \cdot |E|)$.

$\leq |V| \cdot |V| \cdot O(1) = O(|V|^2)$

Can we do better? YES.

$\forall$ vertex $u \rightarrow$ do outer for loop once

$\downarrow$

inner for loop iterations is degree(u).

$\Rightarrow$ total run-time of loops $= \sum_{u \in V} \text{degree}(u) \cdot O(1)$

$= O(1) \cdot \sum_{u \in V} \text{degree}(u)$.

$= O(1) \cdot 2 \cdot |E| = \boxed{O(|E|)}$.

$\Rightarrow$ total run-time $= O(|V| + |E|)$.

Implementation: **Breadth-First Search (BFS)**

DISCOVERED[u] = false for $u \neq s$ → $O(|V|)$ time.

DISCOVERED[s] = true

$L[0] \leftarrow s$; $i \leftarrow 0$

WHILE L[i] IS NOT EMPTY → $O(|E|)$ time.

$L[i+1] \leftarrow \phi$

→ For each vertex $u \in L[i]$

→ For each neighbor v of u ($v \in A[u]$)

If DISCOVERED[v] = true, then

do Nothing.

Else,

Set DISCOVERED[v] ← true,

Add $v \leftarrow L[i+1]$.

$i \leftarrow i+1$.

→ CHECK IF DISCOVERED[t] = true.

|V| total runs.

degree(u) iteratio

O(1) time.

Properties of BFS:

Lemma: The algorithm solves s-t Connectivity Correctly.

Proof: Induction on "distance"

BFS tree

INPUT: $G = (V, E)$ and a vertex s.

OUTPUT: A tree contained in G.

Breadth-First Search Tree

DISCOVERED[u] = false for $u \neq s$

DISCOVERED[s] = true . $T = \phi$

$L[0] \leftarrow s$; $i \leftarrow 0$

WHILE L[i] IS NOT EMPTY

$L[i+1] \leftarrow \phi$

For each vertex $u \in L[i]$

For each neighbor v of u $(v \in A[u])$

If DISCOVERED[v] = true, then

do Nothing.

Else,

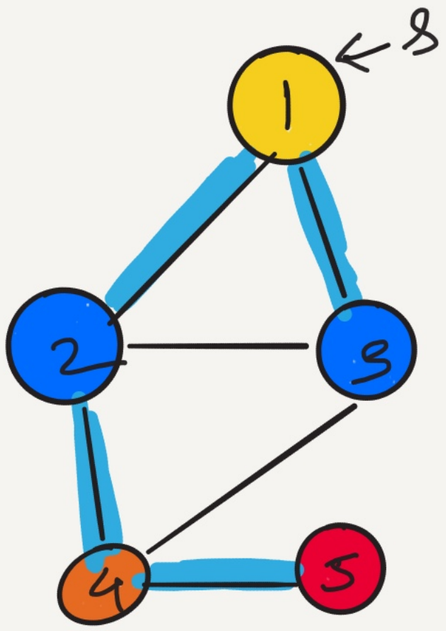
set DISCOVERED[v] $\leftarrow$ true,

Add v TO L[i+1].

ADD Edge {u,v} to T.

$i \leftarrow i+1$.

Example:

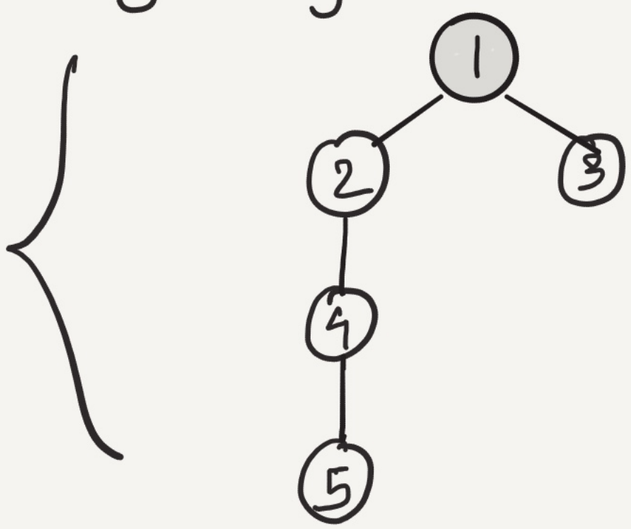


DISCOVERED

	1	2	3	4	5	6	7	8
$L_0 = [s]$	t	f	f	f	f	f	f	f
$L_1 = \{2, 3\}$	t	t	t	f	f	f	f	f
$L_2 = \{4\}$	t	t	t	t	f	f	f	f
$L_3 = \{5\}$	t	t	t	t	t	f	f	f
$L_4 = \emptyset$								

The tree that you get is:

BFS(s)

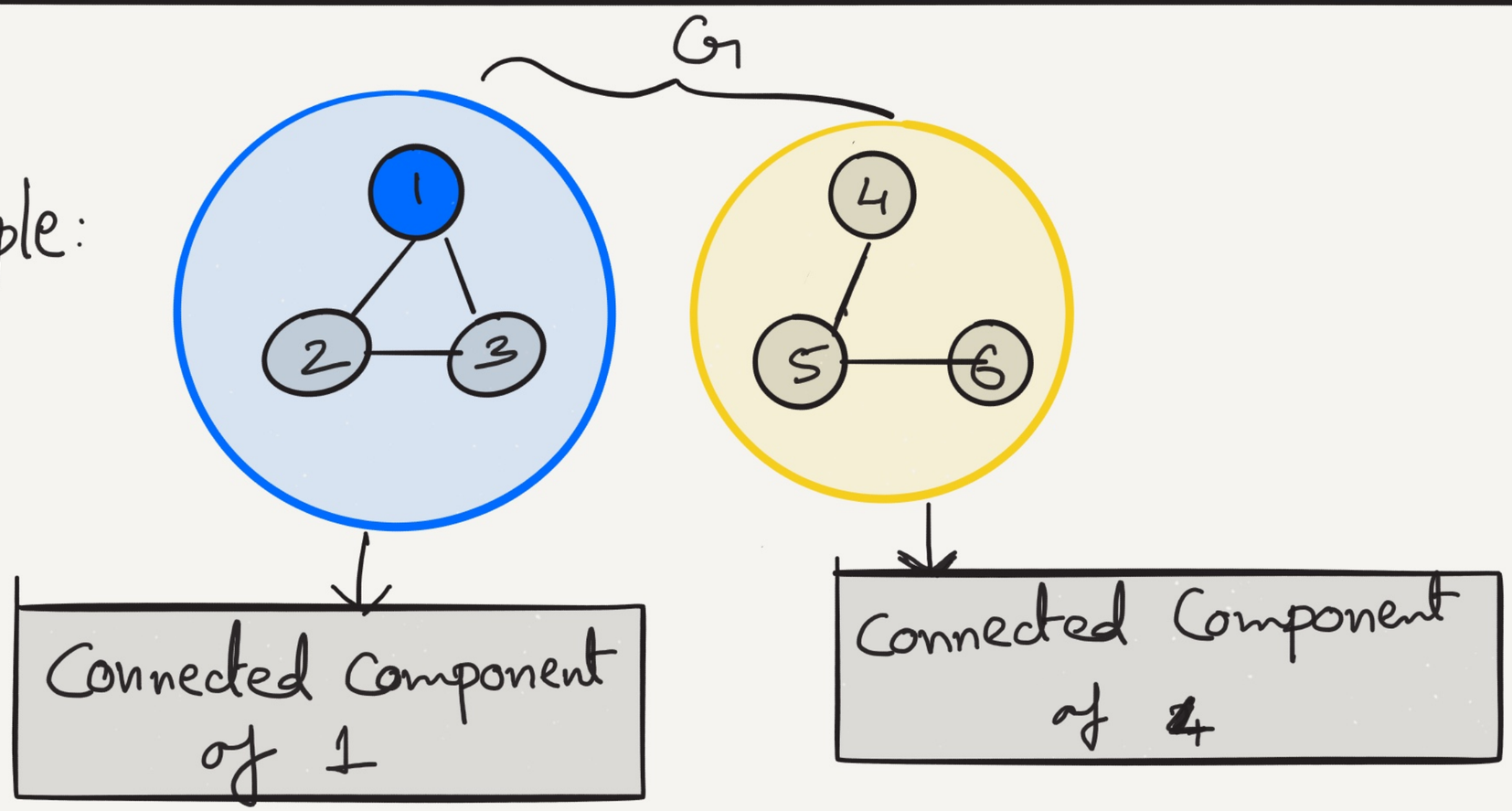


CLAIM: T Constructed during BFS(s) is a tree

Proof: Induction based on distance.

CONNECTED COMPONENT: Connected Component of s is the graph Gs restricted to vertices connected to s.

Example:



Identifying Connected Components:

→ All vertices with DISCOVERED marked true when running BFS(s) are the vertices in the Connected Component of s.

Splitting a graph into Connected components:

→ Is an important primitive in graph theory.

→ Run BFS of a vertex s to find its Connected Component.

Repeat:

→ Run BFS again from a vertex not already discovered.

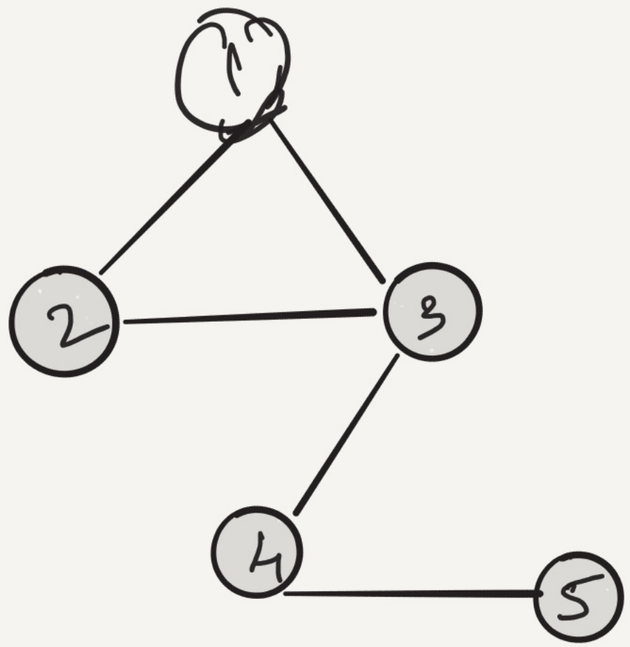
BFS can also solve "shortest path" on unweighted graphs

DISTANCE BETWEEN VERTICES:

$G = (V, E)$ and two vertices u, v.

$$\text{distance}(u,v) = \begin{cases} \text{shortest length of a path from } u \text{ to } v \text{ if } u, v \text{ are connected} \\ \infty \text{ if they are disconnected.} \end{cases}$$

Claim: Vertices in layer i when doing BFS(s) are at distance exactly i from s.

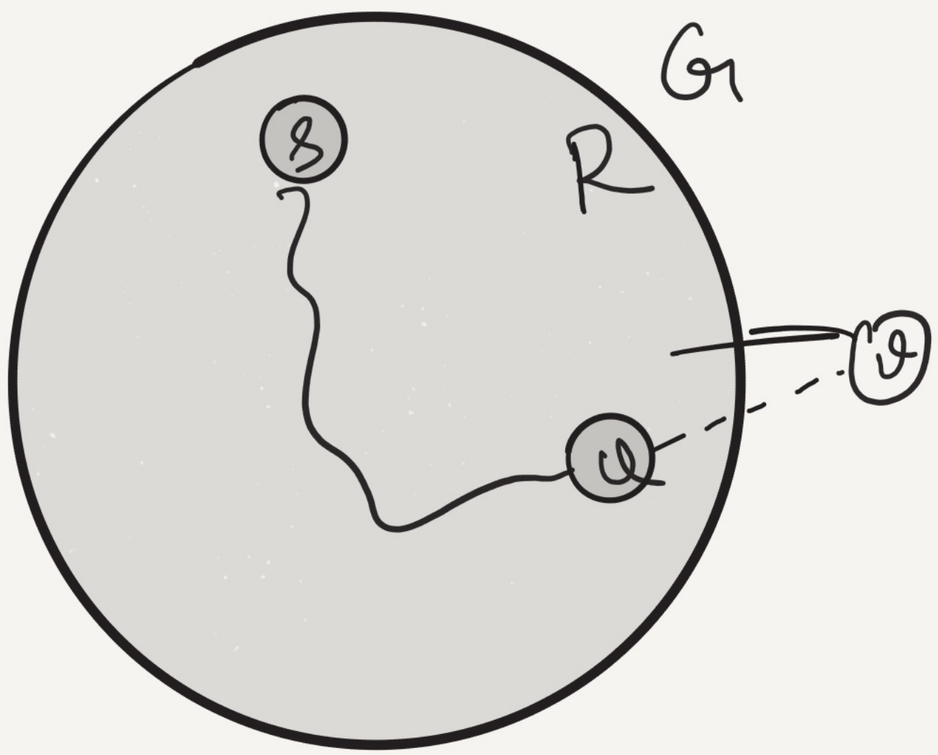


distance(1,3) = 1

distance(2,4) = 2

distance(1,5) = 3

Graph Exploration:



Goal:

Find all vertices reachable/connected from s.

→ $R \leftarrow \{s\}$

→ While there is an edge (u,v) with $u \in R$ and $v \notin R$

→ Add v to R.

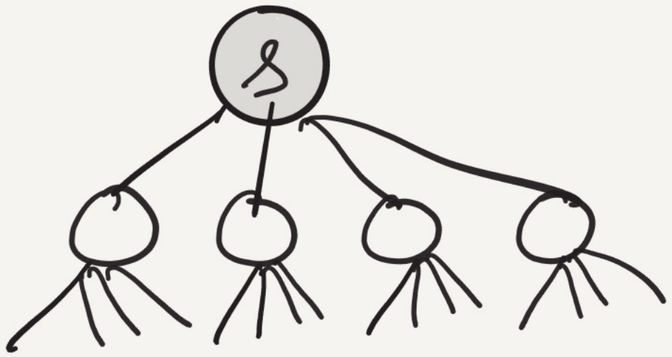
BFS: Corresponds to one way of running the while loop (the order in which edges are explored).

DFS: Depth-First-Search.

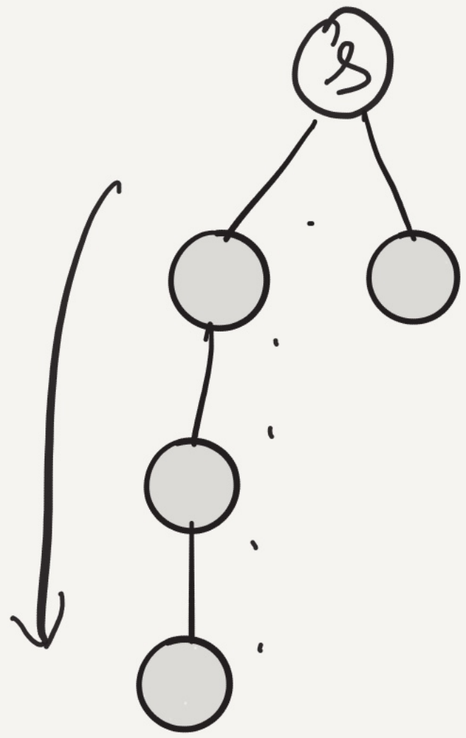
Is also a graph exploration algorithm.

Where you go as "deep" as possible first.

BFS



DFS



DFS Algorithm:

STACK : LIFO ("Last In First Out").

INPUT: $G = (V, E)$ in adjacency list representation.
(∀ any vertex v, $A[v] \leftarrow$ list of neighbors)

OUTPUT: A graph T and an array "EXPLORED"
↓
Marks whether or not a vertex is connected to s.

→ Initialize an array of pointers : Parent[u] = ϕ. $\boxed{T = \phi}$.

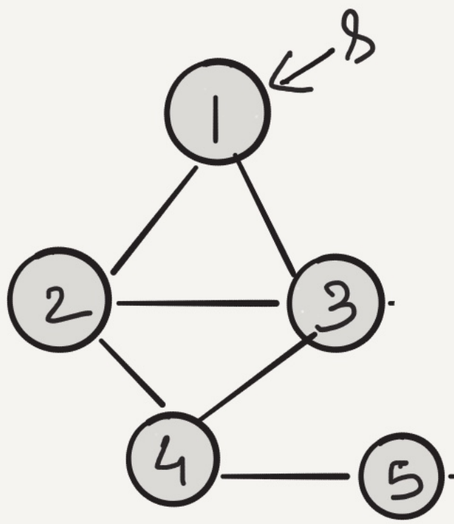
→ Initialize a stack R with one element s.

→ Initialize an array EXPLORED with
EXPLORED[u] = false ∀ u.

→ While R is not empty
→ Take a node u from R (POP(R)).

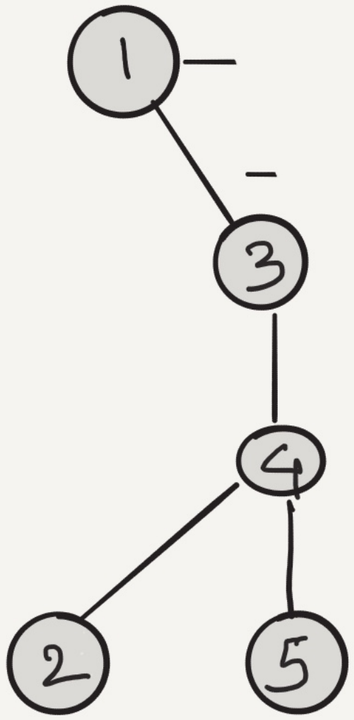
→ If EXPLORED[u] = false, then
→ Set EXPLORED[u] = true
→ Add (Parent[u], u) to the tree T.
→ Add all neighbors of u to R.
→ ∀ vertex v ∈ A[u], PUSH v to R.
→ Parent[v] ← u.

Example:

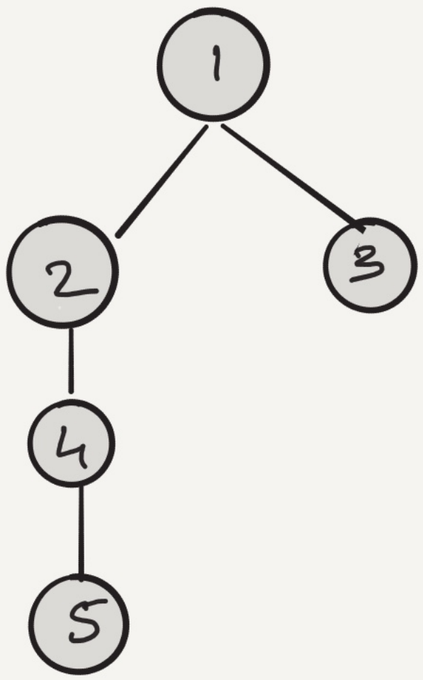


Stack R	EXPLORED					PARENT 1	2	3	4	5
3	0	0	0	0	0	ϕ	ϕ	ϕ	ϕ	ϕ
2 3	1	0	0	0	0	ϕ	1	1	ϕ	ϕ
2 1 2 4	1	0	1	0	0	3	3	1	3	ϕ
2 1 2 2 3 5	1	0	1	1	0	3	4	4	3	4
2 1 2 2 3 4	1	0	1	1	1	3	4	4	5	4
2 1 2 2	1	0	1	1	1	3	4	4	5	4
	1	1	1	1	1	2	4	2	2	4

Tree T

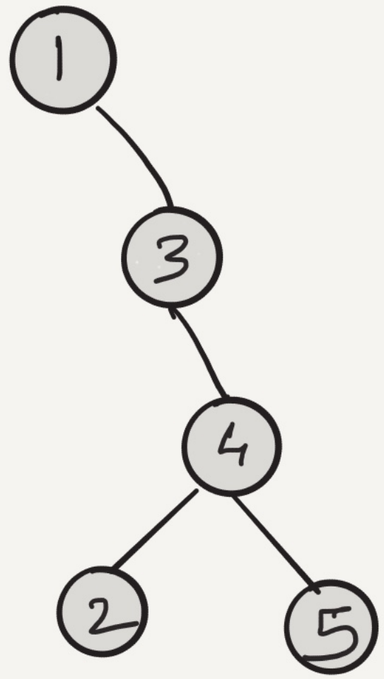


BFS(1)



1. $O(|V|+|E|)$ Run time
2. Connected component(s) = Vertices in T
3. Can tell us distance

DFS(1)



1. $O(|V|+|E|)$
2. Connected component(s) = vertices of T.
3. Does not.