M, 09/16/19

Fall'19 CSCE 629

Analysis of Algorithms

Fang Song
Texas A&M U

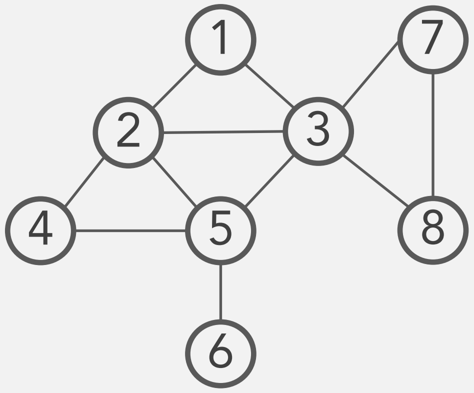
Lecture 7

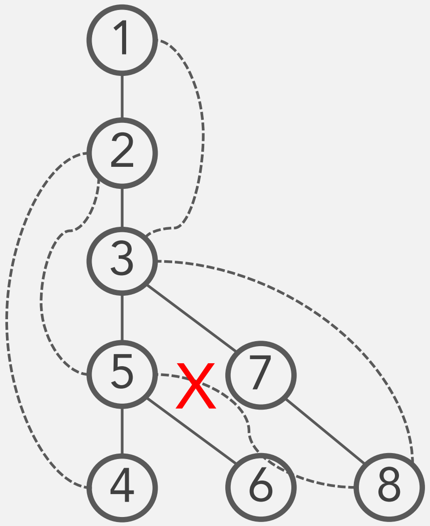
- Graph representations
- BFS/DFS implementations
- Connected component

Credit: based on slides by A. Smith & K. Wayne

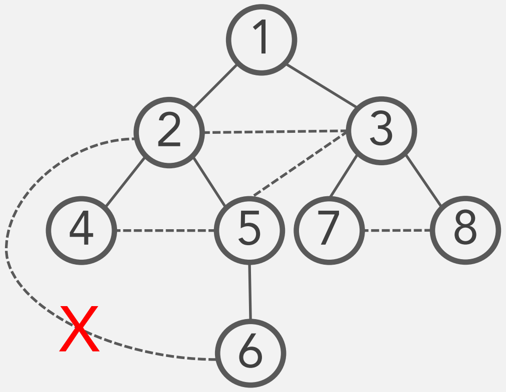
DFS Recap

- Constructing DFS tree





Contrast with
BFS tree



- Running time: linear $O(|V| + |E|)$ (more to come)
- Let T be a DFS tree of G, and let u & v be nodes in T. Let (u, v) be an edge of G that is not an edge of T. Then one of u or v is an ancestor of the other.

Implementing (B/D)FS

Generic traversal algorithm

1. $R = \{s\}$
2. While there is an edge (u, v) where $u \in R$ and $v \notin R$, add v to R.

To implement it, need to choose

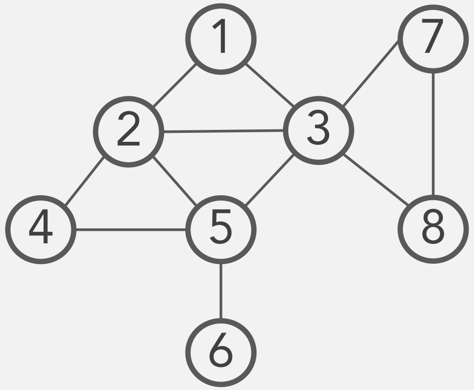
- Graph representation
- Data structures to track…
  - Vertices already explored
  - Edge to be followed next

These choices affect the order of traversal

Graph representation 1: adjacency matrix

$G = (V, E), |V| = n, |E| = m$

- Adjacency matrix A: n–by–n. $A_{uv} = 1$ iff. (u, v) is an edge
  - Lookup an edge: $\Theta(1)$ time
  - List all neighbors: $\Theta(n)$
  - Symmetric (undirected graph)
  - Space: $\Theta(n^2)$, good for dense graphs



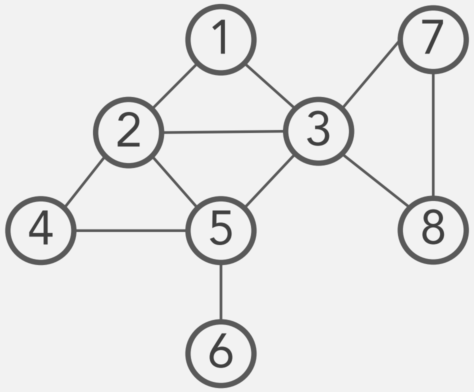
A	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Graph representation 2: adjacency lists

$G = (V, E), |V| = n, |E| = m$

- Adjacency list. $\forall u \in V, Adj[u] = \{v : v \text{ adjacent to } u\}$
  - Lookup an edge (u, v): $\Theta(\deg(u))$ time
  - Space: $\Theta(n + m)$, good for sparse graphs

How many entries in the lists?
$\sum_u \deg(u) = 2m$



$Adj[1] = \{2,3\}$

$Adj[2] = \{1,3,4,5\}$

$Adj[3] = \{1,2,5,7,8\}$

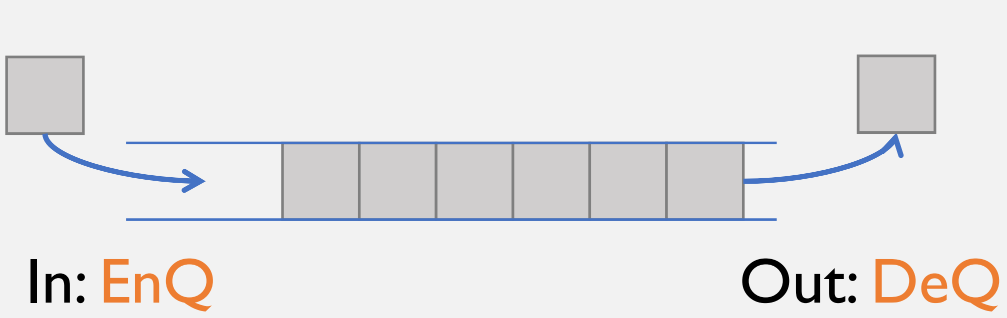
$\vdots$

$Adj[8] = \{3,7\}$

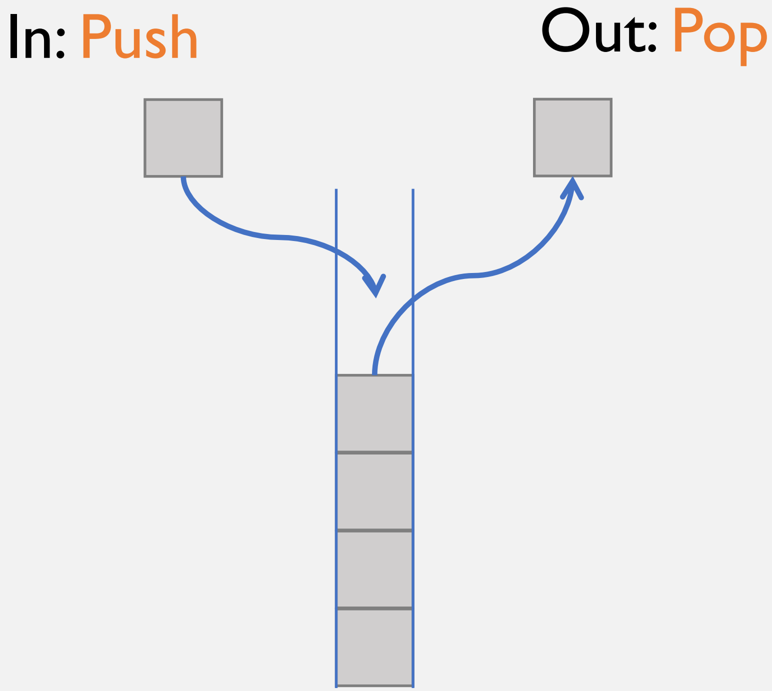
Review: queue & stack

Two options for maintaining a set of elements

1. Queue: first-in first out (FIFO)



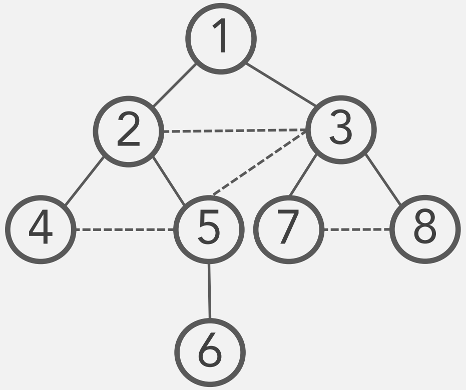
2. Stack: last-in first out (LIFO)

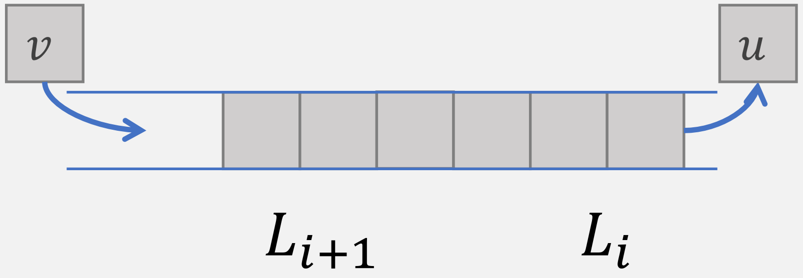


BFS implementation

- Input: $G = (V, E)$ by adjacency list Adj. Start node s.
- Output: BFS tree T (rooted at s). Initialized to empty.

BFS(s): // Discovered[1,...,n]: array of bits (explored or not) – initialized to all zeros.
// Queue $Q \leftarrow \emptyset$
1. Set Discovered[s] = 1
2. EnQ(s) // add s to Q
3. While Q not empty, DeQ(u)
    For each (u, v) incident to u
        If Discovered[v] = 0 then
            Set Discovered[v] = 1
            Add edge (u, v) to T
            EnQ(v)





BFS running time

BFS(s): // Discovered[1,...,n]: array of bits (explored or not) – initialized to all zeros.
// Queue $Q \leftarrow \emptyset$

1. Set Discovered[s] = 1
2. EnQ(s) // add s to Q
3. While Q not empty, DeQ(u)
   - For each (u, v) incident to u
     - If Discovered[v] = 0 then
       - Set Discovered[v] = 1
       - Add edge (u, v) to T
       - EnQ(v)

$O(1)$, run once for all

$O(1)$, run once per vertex

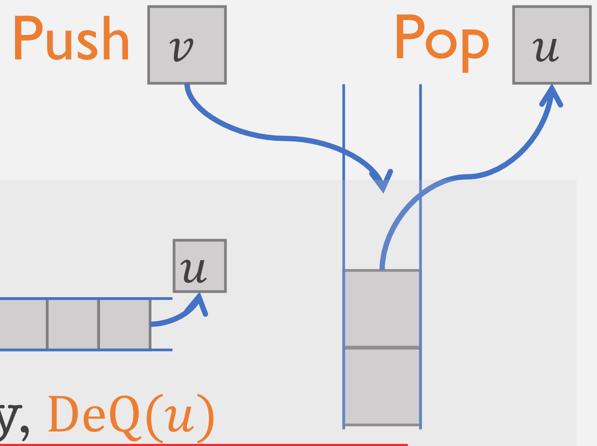
$O(1)$, run $\leq$ twice per edge

Theorem. **BFS** takes $O(m + n)$ time (linear in input size).

DFS implementation

Theorem. **DFS** takes $O(m+n)$ time (linear in input size).

DFS(s): // Discovered[1,…,n]
// Stack $S \leftarrow \emptyset$
1. Set Discovered[s] = 1
2. Push(s) // add s to S
3. While S not empty, Pop(u)
    If Discovered[u] = 0 then
        Set Discovered[u] = 1
        For each (u,v) incident to u
            Push(v)



BFS(s):
…

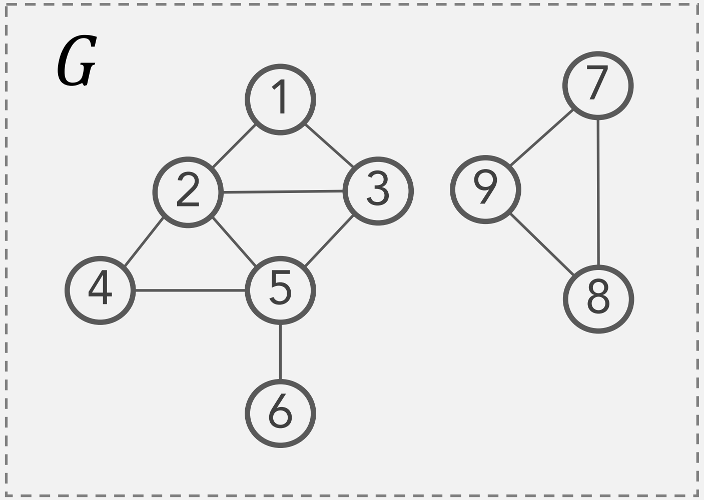
3. While Q not empty, DeQ(u)
    For each (u,v) incident to u
        If Discovered[v] = 0 then
            Set Discovered[v] = 1
            Add edge (u,v) to T
            EnQ(v)

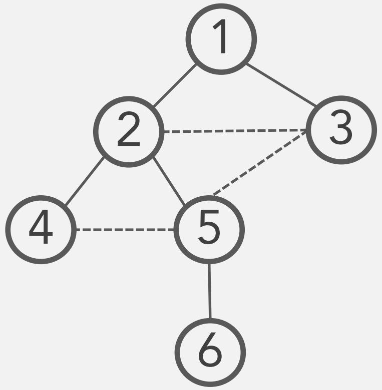
Exercise. How to build DFS tree T along the way?

Connected components

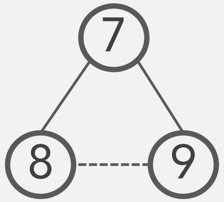
- B/DFS actually tells more than s–t conectivity…

> Connected component of G containing s:
> all nodes reachable from s





$BFS(1)$



$BFS(7)$

- Claim. For any two nodes s and t, their connected components are either identical or disjoint.

The set of all connected components

- In-class discussion
  - How to find all connected components?
  - How fast?
  - Why care?

- Iterate over V, run B/DFS
- $\sum_i O(n_i + m_i) = O(m+n)$
- Basic topology about G