Representing Graphs.

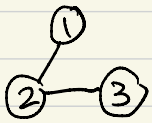
Method 1: (Adjacency Matrix)

For an n-node graph $G=(V,E)$. $\left(\begin{array}{l} n = \text{\# of nodes} \\ m = \text{\# of edges} \end{array}\right)$.

Assume $V = \{1, 2, \dots, n\}$.

The adjacency matrix of G is an $n \times n$ matrix A where

$$A[u,v] = 1 \quad \text{iff} \quad (u,v) \in E.$$



$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

This takes $\Theta(n^2)$ space.

Method 2: (Adjacency List).

For each $u \in V$, we have an array $Adj[u]$ that contains all nodes v where $(u,v) \in E$.

$Adj[1] = \boxed{2}$

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

$Adj[3] = \boxed{2}$

Claim: The adjacency list representation takes $O(n+m)$ space.

Proof: First we need pointers to $Adj[i]$. $\Rightarrow O(n)$

Each edge appears in exactly two adj. lists $\Rightarrow O(m)$

Implementing BFS/DFS:

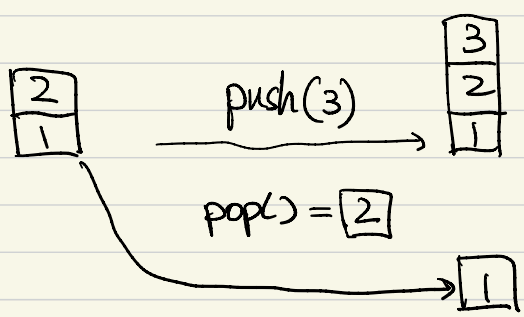
Stack: First-In-Last-Out

Queue: First-In-First-Out

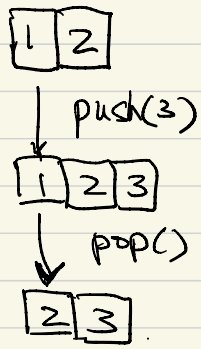
push: add an element to the data struture (DS).

pop: remove an element.

Stack:



Queue:



Implementing BFS:

```
BFS(s):
    Set discovered[s]=1 and discovered[v]=0, ∀v≠s;
    queue Q = empty.
    Q.push(s)   ← Q contains all nodes that are discovered but not explored.
    while ( Q is not empty)
        u = Q.pop();
        For each edge (u,v) incident to u
            If discovered[v] = 0
                discovered[v]=1
                Add edge (u,v) to the BFS tree T_B.
                Q.push(v)
            End if
        End for
    End while
```

How do we iterate over all neighbors of u?

- If G is represented by an adjacency matrix A:
  we loop over $v=1\ldots n$ and check if $A[u][v]=1$.
- If G is represented by adjacency lists $Adj[i]$:
  we iterate over $Adj[u]$.

Runtime of BFS: (Assume Q.push(x) and Q.pop() runs in $O(1)$ time.)

- In either adj. matrix or adj. lists representation,
  BFS runs in time $O(n^2)$.
  Because WHILE loop runs at most n times,
  and <u>each</u> for loop runs at most n times.

- In the adj. lists representation:
  BFS runs in time $O(m+n)$. $= O(n^2)$
  Because the FOR loop of node u runs $|Adj(u)|$ times.
  <u>Total</u> number of time we run the FOR loop
  is $O(\sum_u |Adj(u)|) = O(2m) = O(m)$.
  Additional $O(n)$ to set up and manage the length n array "discovered"