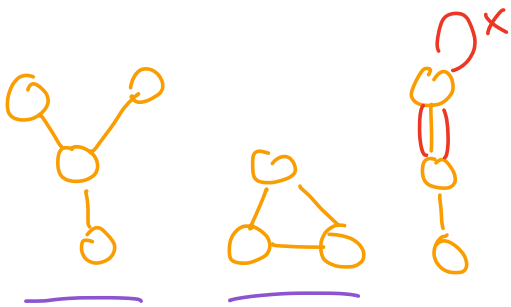
# Graphs!

## Definitions:

Def: set of vertices $V$ & edges $E$



- NO multiedges
  → each pair of vert has 0/1 edges
- No self-loops
- undirected
  so if $(u,v)$ in $E$
  $(v,u)$ also in $E$

Neighbors: $u$ is a neighbor of $v$ if $\exists$ edge $(u,v) \in E$
($v$ is a neighbor of $u$ if " )

Dense graph: $|E| \in O(|V|^2)$

← fully connected graph

$|E| = \binom{|V|}{2} = \frac{|V|(|V|-1)}{2}$

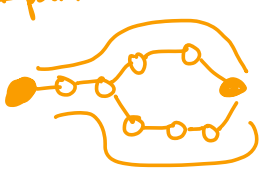
Sparse: $|E| \in O(|V|)$, at most $O(|V| \log |V|)$, definitely not $O(|V|^2)$

Path: list of adjacent vertices

⇒ 2 vert $u$ & $v$ are connected if $\exists$ a path btwn $u$ & $v$

⇒ cycle: ≠ more than 1 path btwn 2 vert

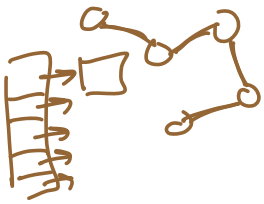
ex)



Tree: connected & acyclic graph

# Representations (n vertices, m edges)

- vertices: # 0 through n-1 (zero-indexing)
  - unsigned int
- edges:
  - ↳ adj list: maintain a list of neighbors for each vert
    - * insert edge ⇒ insert into 2 lists
    - → * better for sparse graphs, take less space

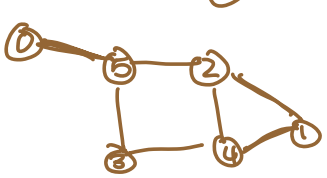


  - ↳ adj matrix: 2D array ⇒ n x n matrix
    - * A[u][v] = 1 if ∃ edge btwn u & v ⇒ A[v][u] = 1
    - * A[u][u] = 0 b/c no self edge
    - * A is symmetric
    - → * better for dense graphs, easy access to check if edge exists

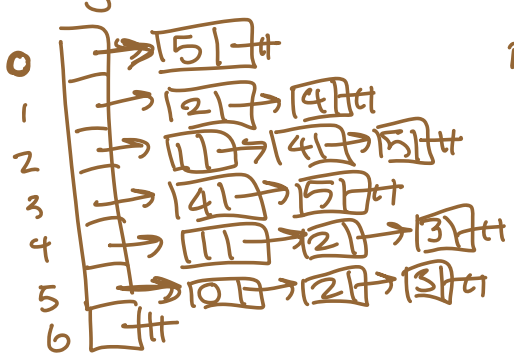
5x5
8

adjlist ** matrix

Ex.)

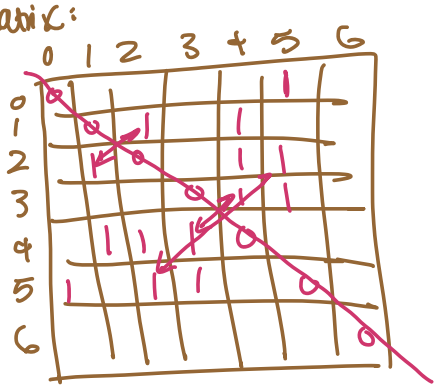



Adj list:




NULL-terminated

Adj Matrix:



# Graph Operations & Complexity (see recording for in-depth explanation!)

| | Adj List | Adj Matrix |
|---|---|---|
| Space | O(V+E) | $O(V^2)$ |
| graph_rmedge | O(min(V,E)) | O(1) |
| graph_addedge | O(1) | O(1) |
| graph_get_neigh | O(1) | O(V) |
| iterate thru neighbors | O(min(V,E)) | O(min(V,E)) |

## Graph Search

Goal: want to see if can reach target vert v from src vert u

### BFS

iterative w/ queues:

1. create queue w/ start vert u, visited set & mark start as visited
2. while queue not empty:
   - ↳ dequeue
   - ↳ check if matches target vertex
   - ↳ enq its neighbors & mark them as visited if not visited yet

### DFS:

iterative w/ stacks (recc):

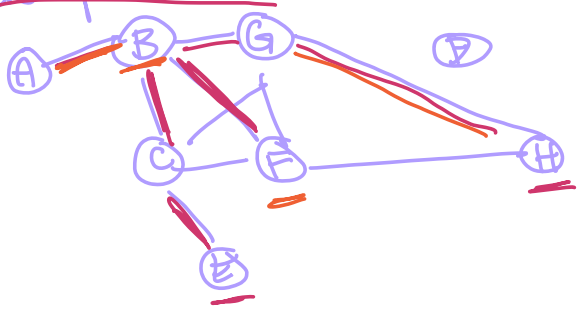
1. create stack w/ start vert, visited set & mark start as visited
2. while stack not empty:

↳ pop
↳ (push neighbors & mark as visited) if haven't seen already

recursive (check):
BC: check if start & target are equal
Else get neighbors of start
& for each neighbor that isn't visited already:
run BFS again at new start, save target

If run BFS/DFS & return false, u & v are not connected !!

Reverse alphabetical order to break ties




* Stop when find target vertex! *
start A
find vert H

mark before enq! (iterative)

~~A B G F C H E~~

BFS: A, B, G, F, C, H

DFS: A, B, G, H

> Compare to recitation results:
Search takes fewer vertices to find target
⇒ how we break ties affects how many vertices we visit