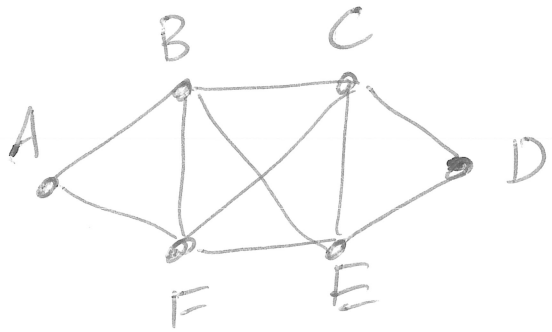
COP 3503 - 3/1/2022

Graphs

Undirected, Unweighted

$G = \langle V, E \rangle$

- V: Vertices → dots
- E: edges → lines btw dots.



$V = \{A, B, C, D, E, F\}$

$E = \{(A,B), (B,C), (A,F), (B,E), (B,F), (E,F), (C,D), (D,E), (C,E), \cancel{(B,F)}\ (C,F)\}$

Useful because they can model lots of real world things.

$\Rightarrow$ Cities + Roads connecting ~~cities~~ cities.

Storage

1. Adjacency Matrix → Floyd-Warshall's ... → Calculate # Paths of length k ...
2. Adjacency List - Use this MOST of the time!!!

↳ Usually takes less storage

for each vertex we need a list of vertices

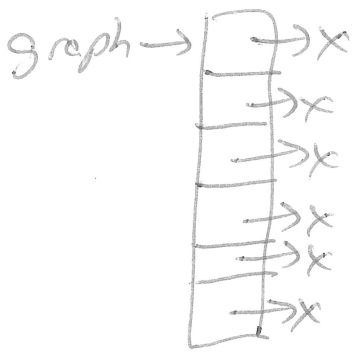
A: B, F
B: A, C, E, F
C: B, D, E, F
D: C, E
E: B, C, D, F
F: A, B, C, E

In practice, how to store

Array of ArrayList, vertices numbered 0 to n-1.

```
ArrayList<Integer>[] graph =
        new ArrayList[n];

for (int i=0; i<n; i++)
    graph[i] = new ArrayList<Integer>();
```

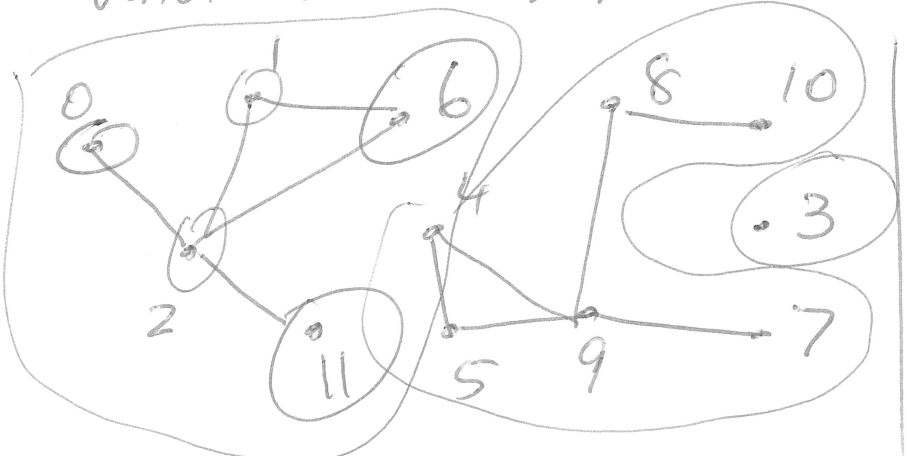



```
for (int i=0; i<numE; i++) {
    int u = stdin.nextInt()-1;
    int v = stdin.nextInt()-1;
    graph[u].add(v);
    graph[v].add(u);
}
```

Today's Algorithms

1. Depth First Search
2. Breadth First Search

Goal of both is to explore every reachable vertex in a graph from a starting vertex.



Problem:
Given a graph and vertex, u, mark each reachable vertex from u.

```
DFS(Graph G, vertex v, boolean[] visited) {
    visited[v] = true;
    for (Integer next = G[v])
        if (!visited[next]) //ONLY GO TO NEW PLACES
            DFS(G, next, visited);
}
```

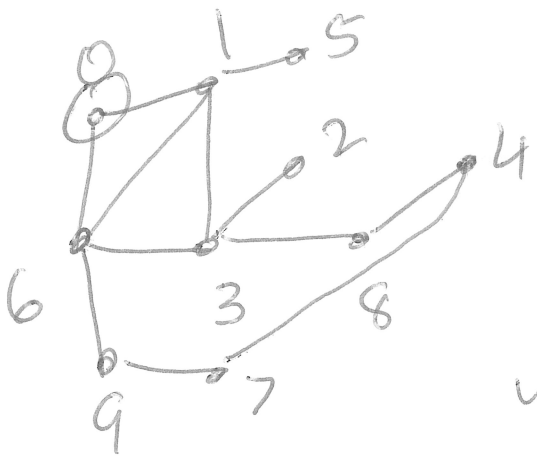
~~DFS(0)~~
~~DFS(1)~~ ~~DFS(11)~~
~~DFS(2)~~
~~DFS(0)~~ 0, 2, 1, 6, 11

Breadth First Search

Input : G, Starting Vertex : v

Goal: Mark every where we can visit

Goal #2: Mark the shortest distances from the starting vertex to all other vertices.



	0	1	2	3	4	5	6	7	8	9
dist	0	~~∞~~ 1	~~∞~~ 3	~~∞~~ 2	~~∞~~ 4	~~∞~~ 2	~~∞~~ 1	~~∞~~ 3	~~∞~~ 3	~~∞~~ 2

Queue: ~~0, 1, 6, 8, 3, 9, 2, 8, 7, 4~~

```
while (queue not empty) {
    int cur = remove next queue item
    for each of cur's neighbors: next
        if (dist[next] != -1) continue
        dist[next] = dist[cur] + 1;
        Add next to queue
}
```

If implemented w/ Adjacency Lists

RUN in $O(V+E)$.

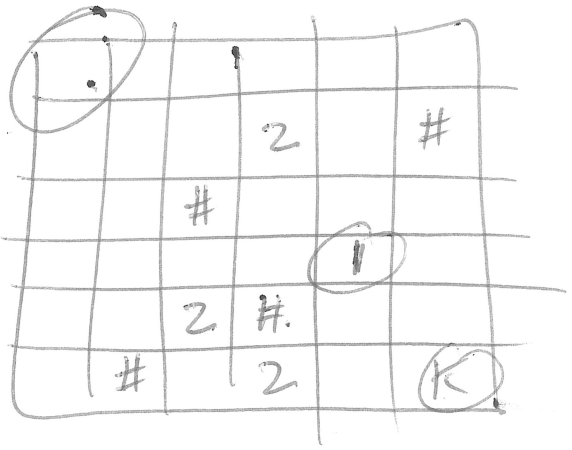
Elevator

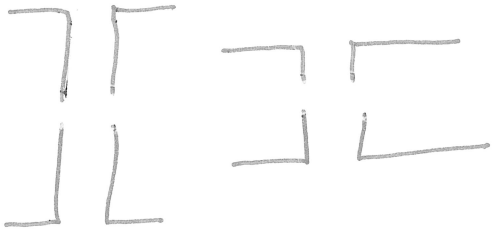
10 floors
S = 5
goto = 8
U = 4
d = 3

edges btwn each pair of floors I can travel

DIRECTED!

Knight Moves





$dx = \{-2, -2, -1, -1, 1, 1, 2, 2\}$

$dy = \{-1, 1, -2, 2, -2, 2, -1, 1\}$