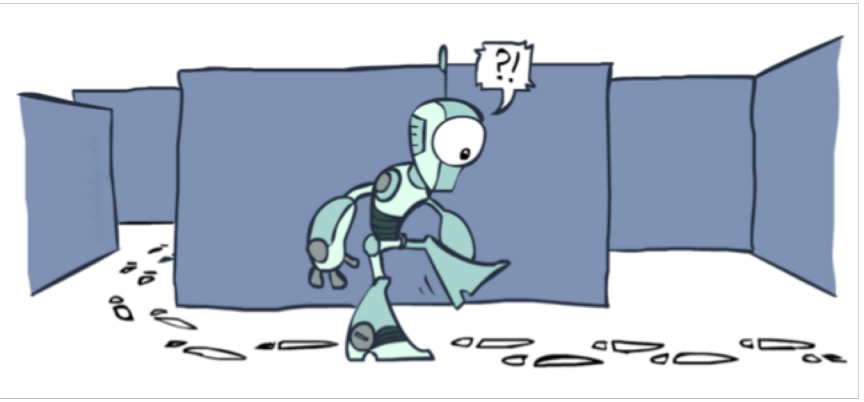
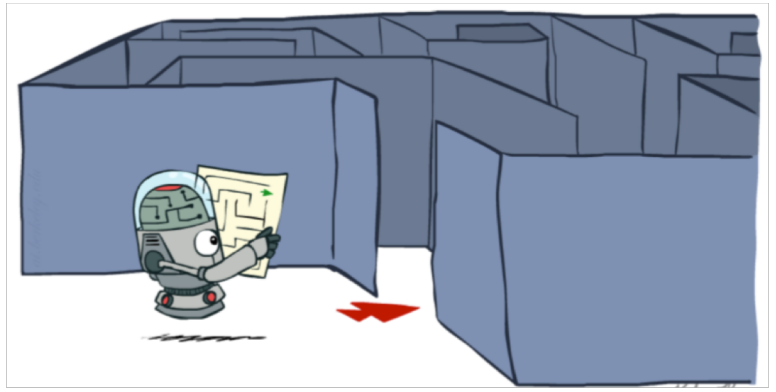
CS 1501: Intro to Robotics

Autonomy, AI, and Applications

Motion Planning II

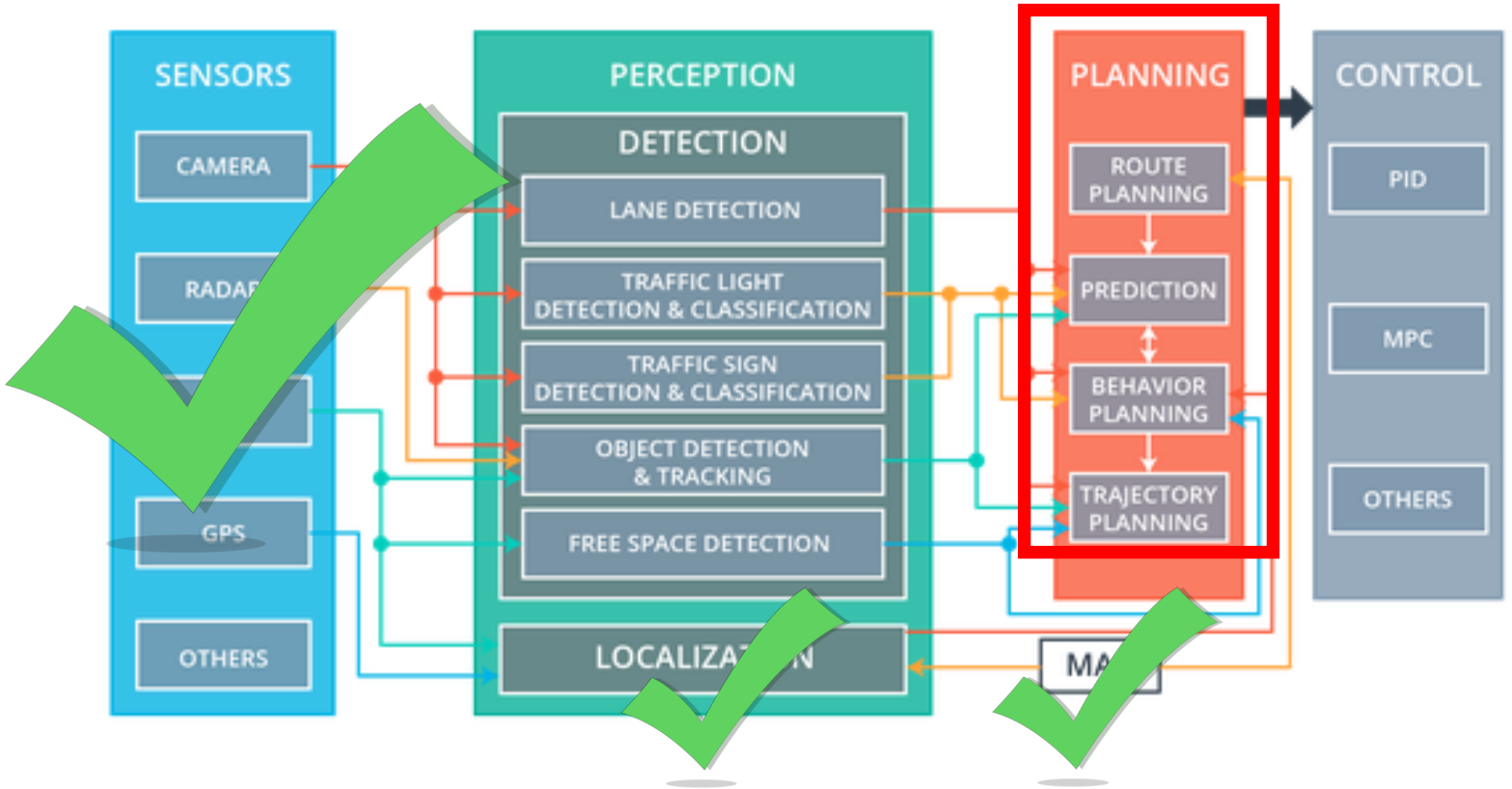




Rohan Raval

Monday 1-1:50pm, MEC 213

Recap: See-Think-Act

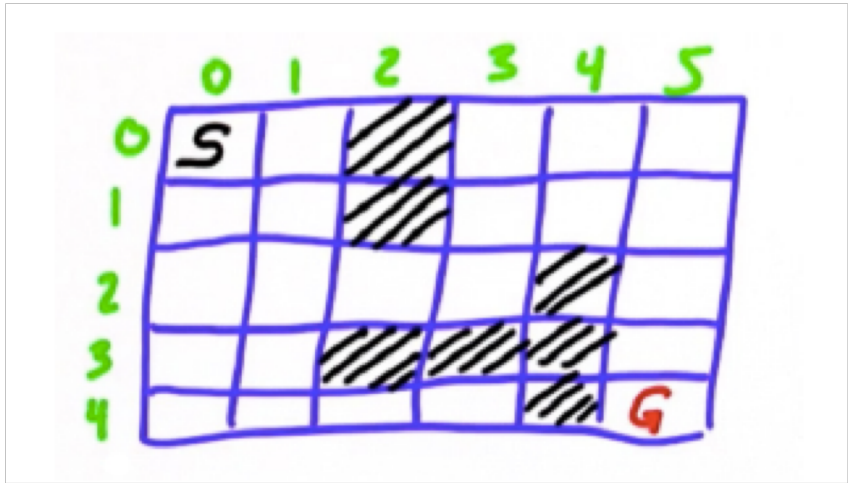


Recap: What is Motion Planning?

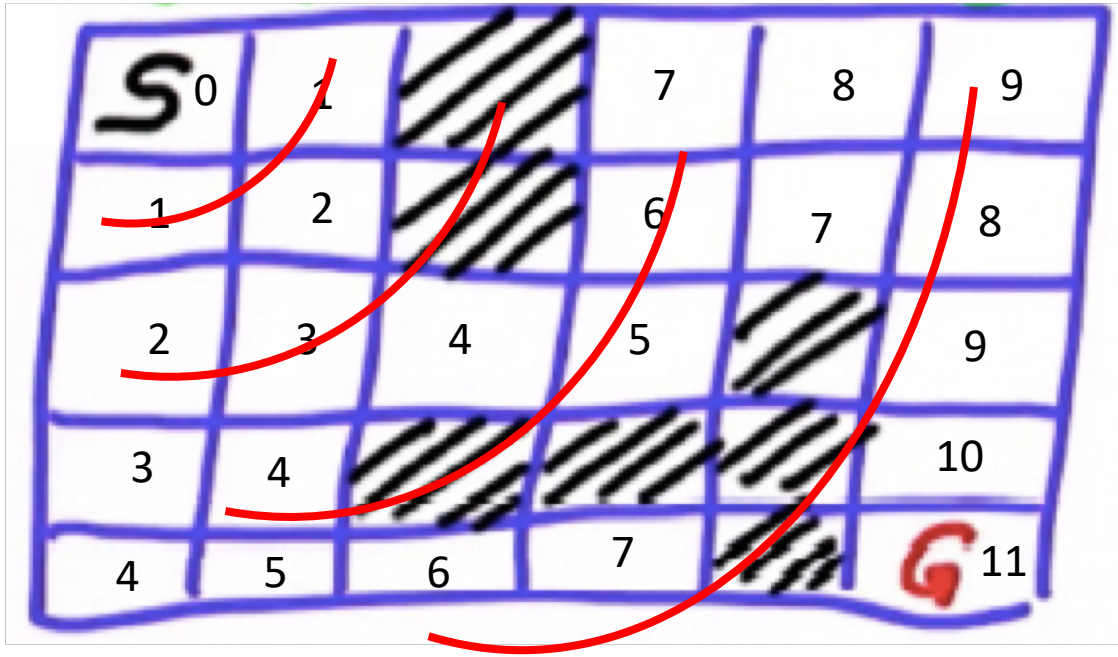
Robot Motion Planning Problem:

- Get from *location* A to *location* B on a map

- **Given:**
  - Map
  - Start Location
  - End Location (Goal)
  - Cost (in our case, distances between locations)
- **Find:**
  - Minimum-cost path (in our case, **shortest-path**)



Recap: Grassfire Algorithm



Grassfire Algorithm

Key idea: keep track of the order in which the "fringe" cells expand

How?

- Maintain a dynamic list ("queue") of unvisited cells

Pseudo-code:

```
For each node n in the graph
    n.distance = infinity

Create an empty list
Set goal.distance = 0 and add goal to list

While list is not empty
    Let current = first node in list, remove current from list
    For each node n that is adjacent to current
        If n.distance = infinity
            n.distance = current.distance + 1
            Add n to back of the list
```

Grassfire Algorithm

Pros:

- If path exists between start and goal, it will find the *shortest path*
- If no path exists, it will discover this
- Simple and easy to program

Cons:

- Computationally inefficient
  - Storing too much!
  - Computation effort increases *linearly* with number of cells
  - **Do we need to explore all possible fringes? Or are there better ones...**
- **Grid is a limiting representation...**

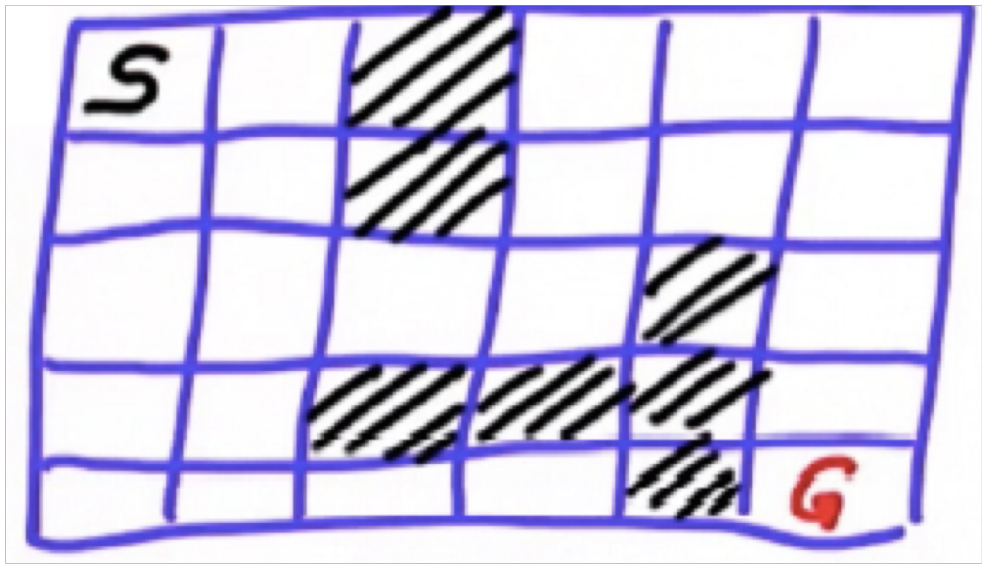
Intro to Graphs

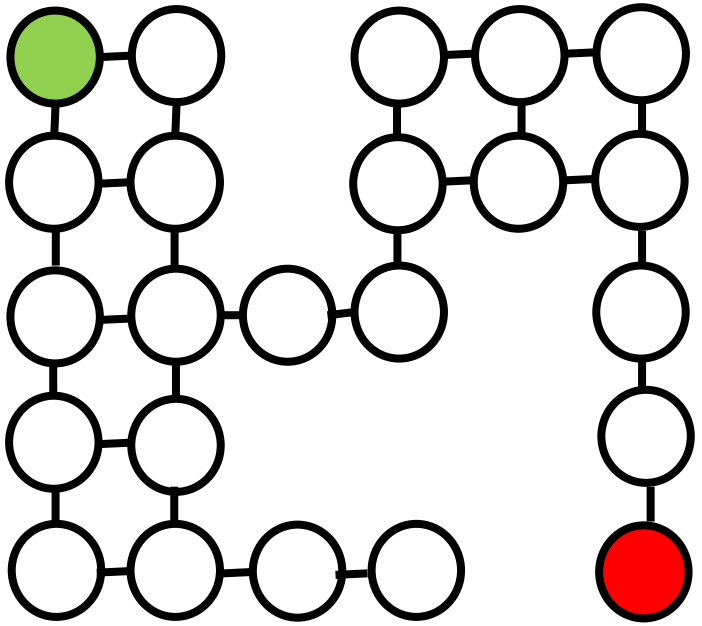
Representation Matters!

- Let's model our problem as a GRAPH search problem (instead of GRID)...

What is a Graph?

- V = "Vertices" or "Nodes"
- E = "Edges"
- G = (V, E)

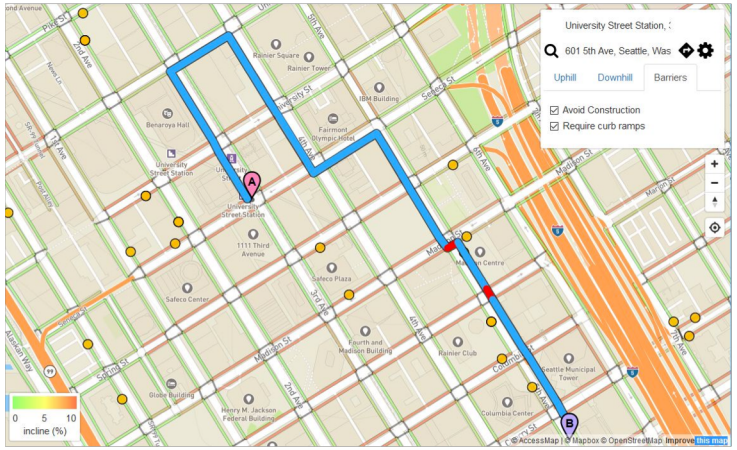


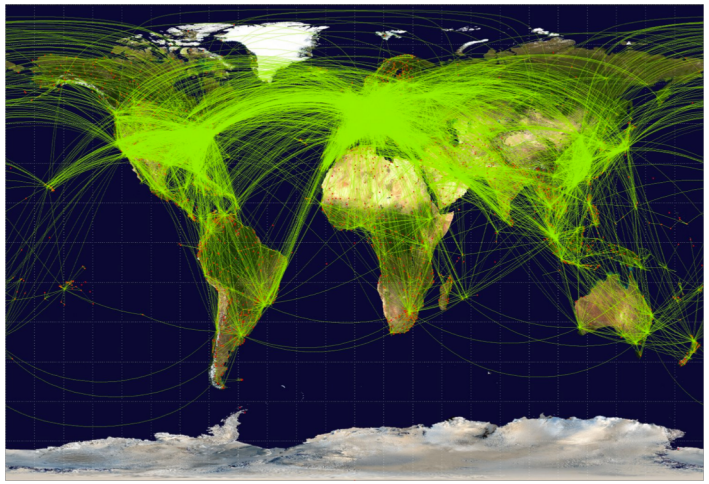


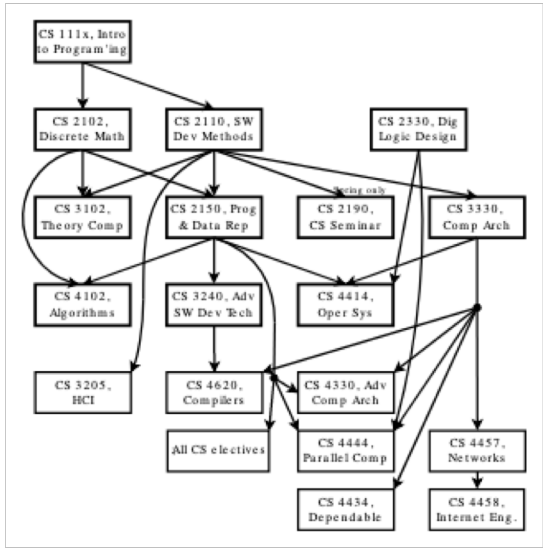
Intro to Graphs

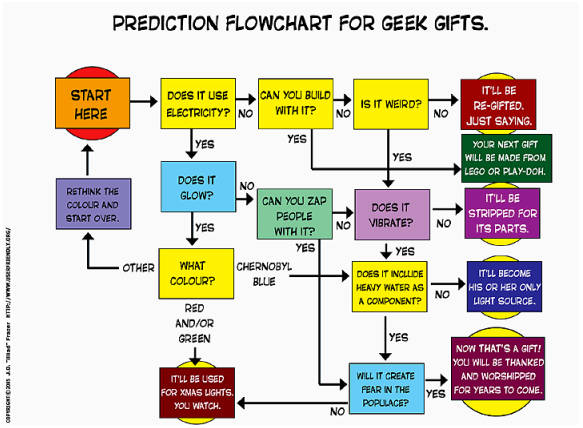
Why Graphs?

- Road Map Routing
- Airline Routing
- Internet Traffic Routing
- Flowcharts
- Prerequisite Diagrams
- Pancakes
- So much more....





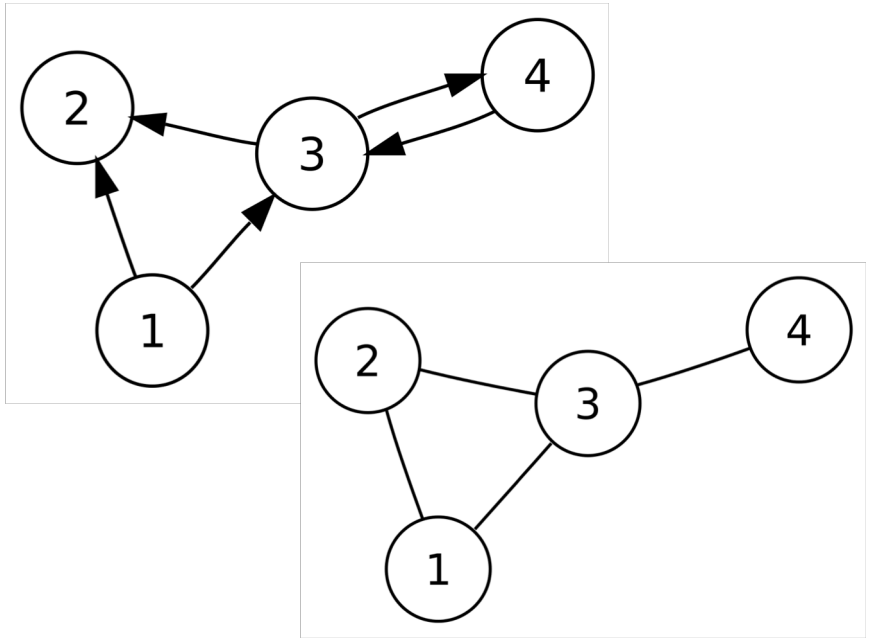




Intro to Graphs

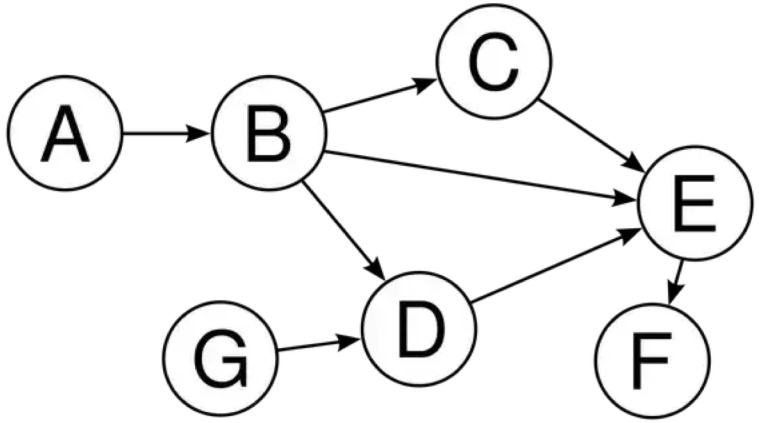
Terminology

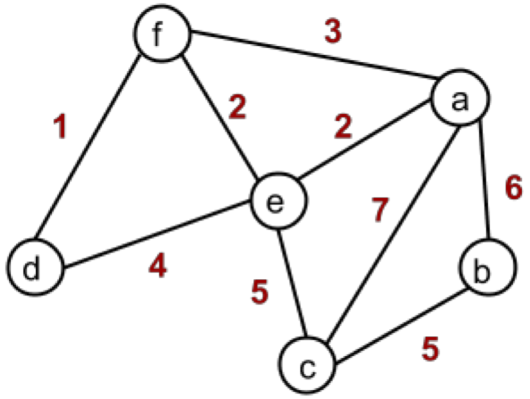
- **Directed** vs **Undirected** Graphs
- "DAG" = Directed Acyclic Graph
- **Weights** or **Costs** associated with each edge
- **Path** = sequence of vertices connected by edges
- **Length of a Path** = number of edges



What to do with a Graph?

- Search!

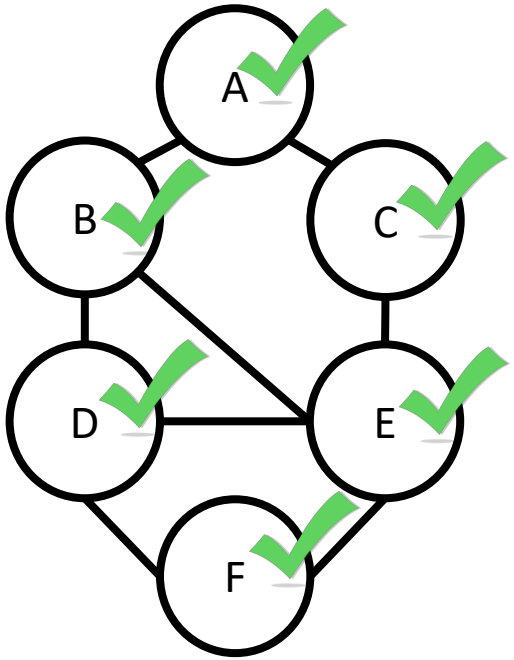


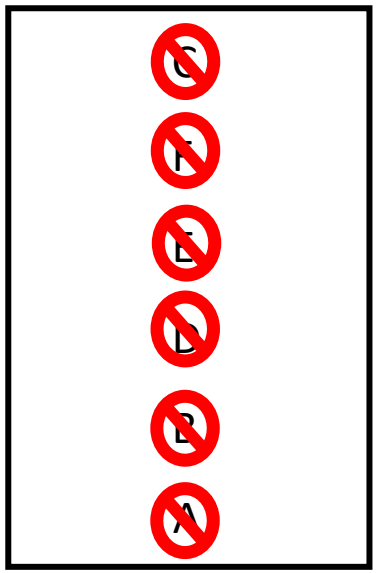


Graph Search 1: DFS

DFS = Depth First Search

Idea: go forward in depth while you can, after that backtrack



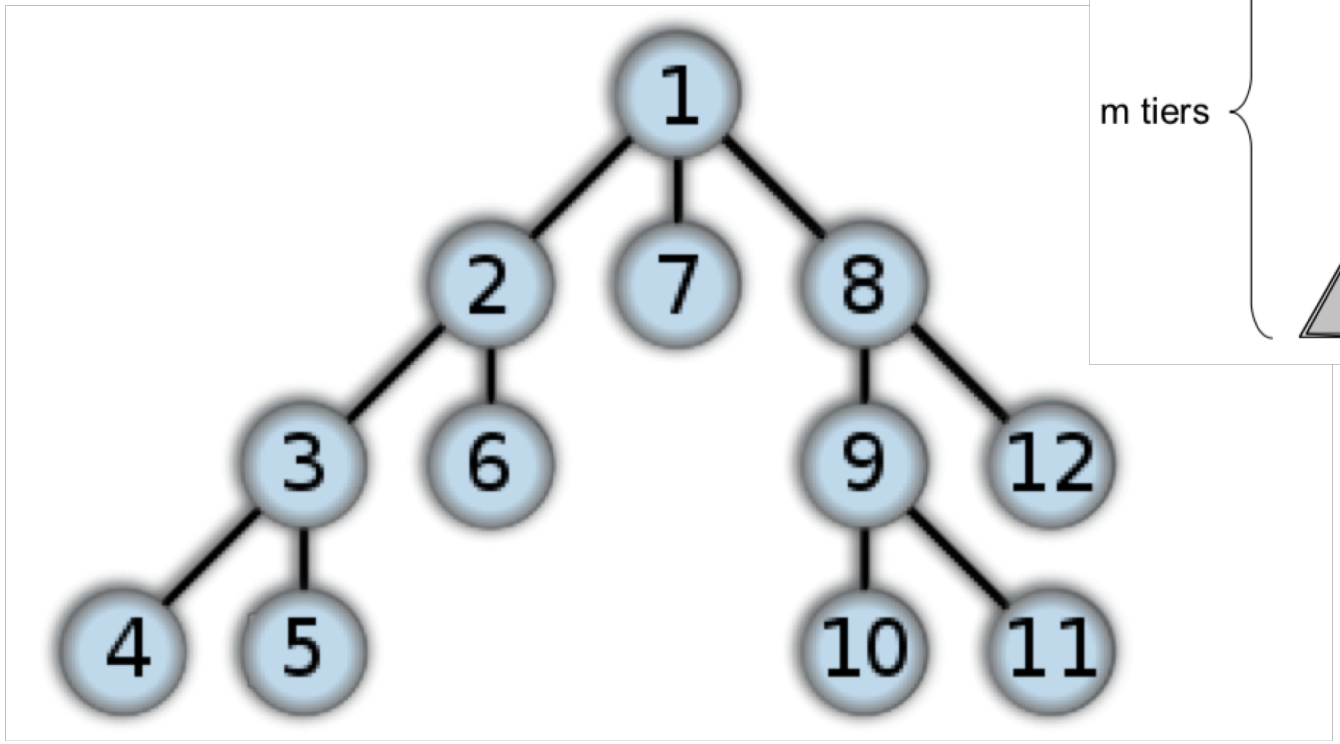


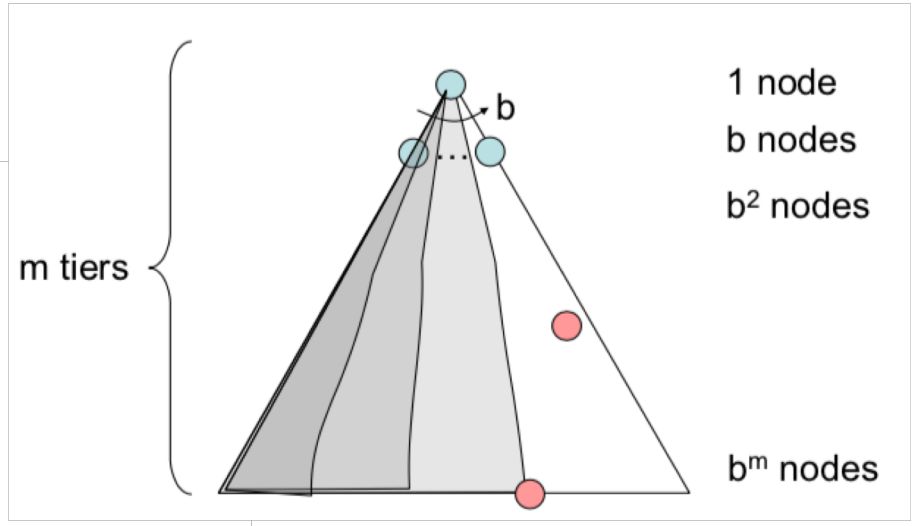
STACK (recursion)

Traversal: A B D E F C

Graph Search 1: DFS

Try it yourself!

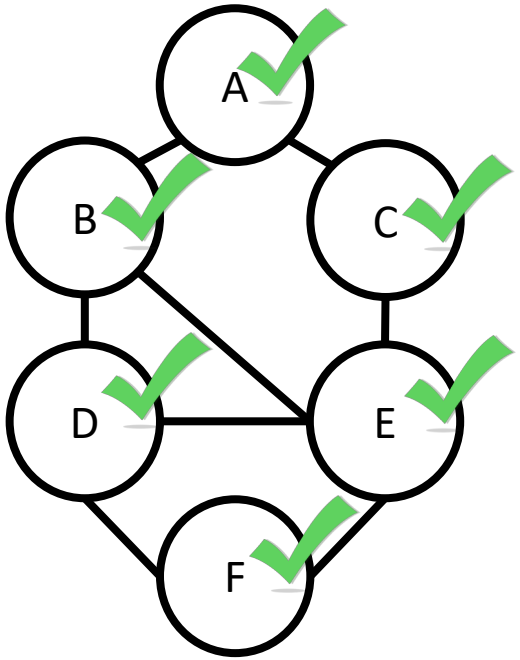




Graph Search 2: BFS

BFS = Breadth First Search

Idea: visit all your node's neighbors, then visit theirs, ...

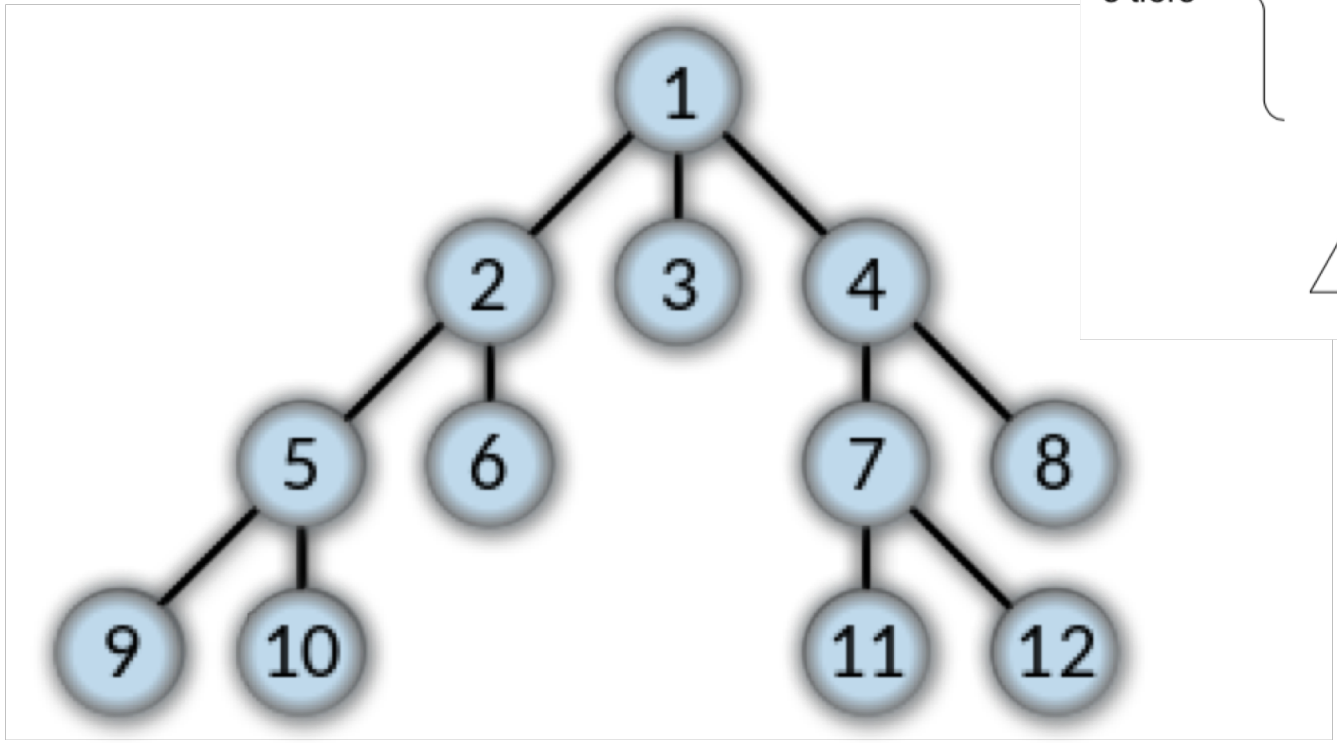


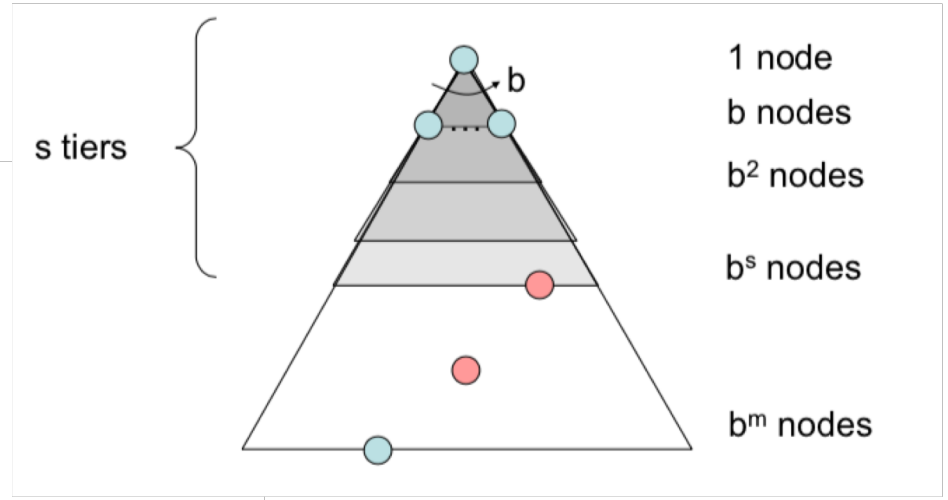
QUEUE A B C D E F

Traversal: A B C D E F

Graph Search 2: BFS

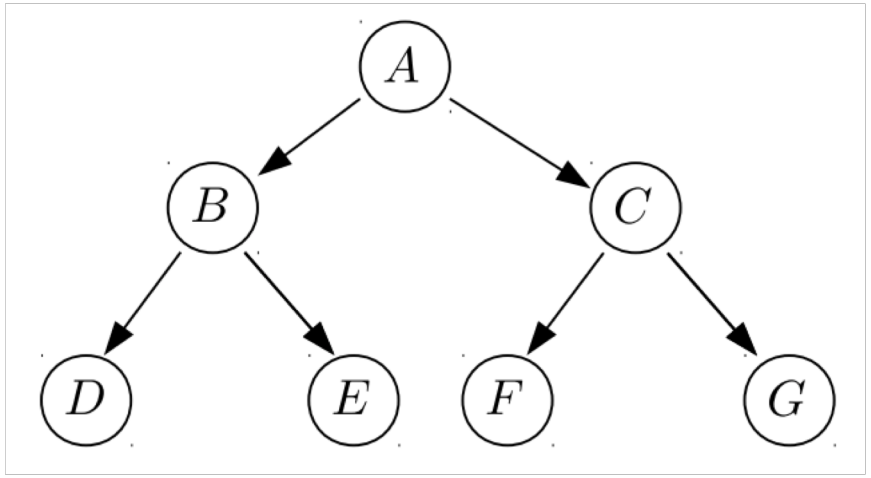
Try it yourself!





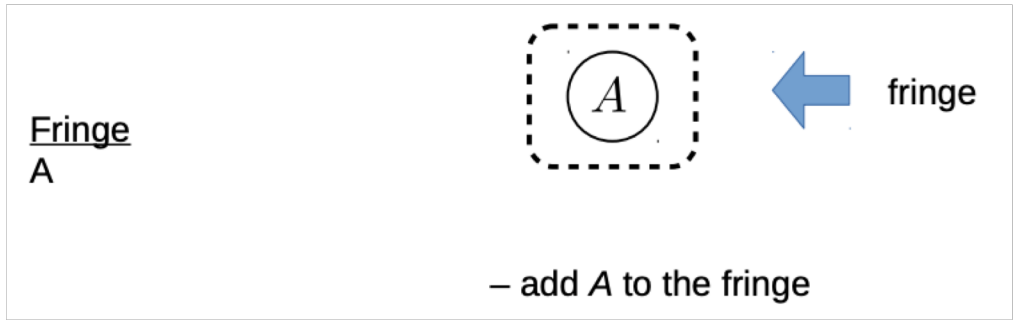
Graph Search 2: BFS

Maintaining a “fringe”



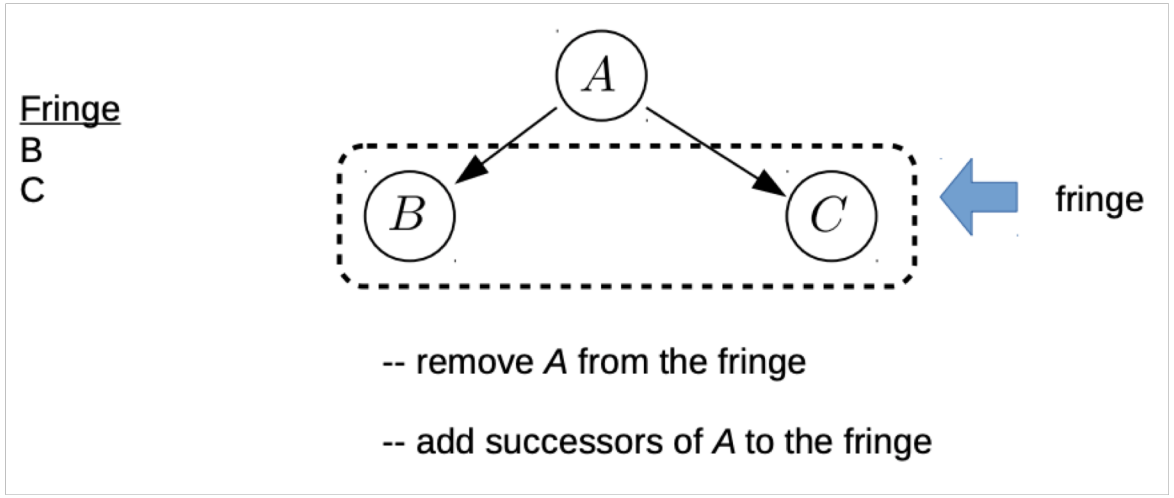
Graph Search 2: BFS

Maintaining a “fringe”



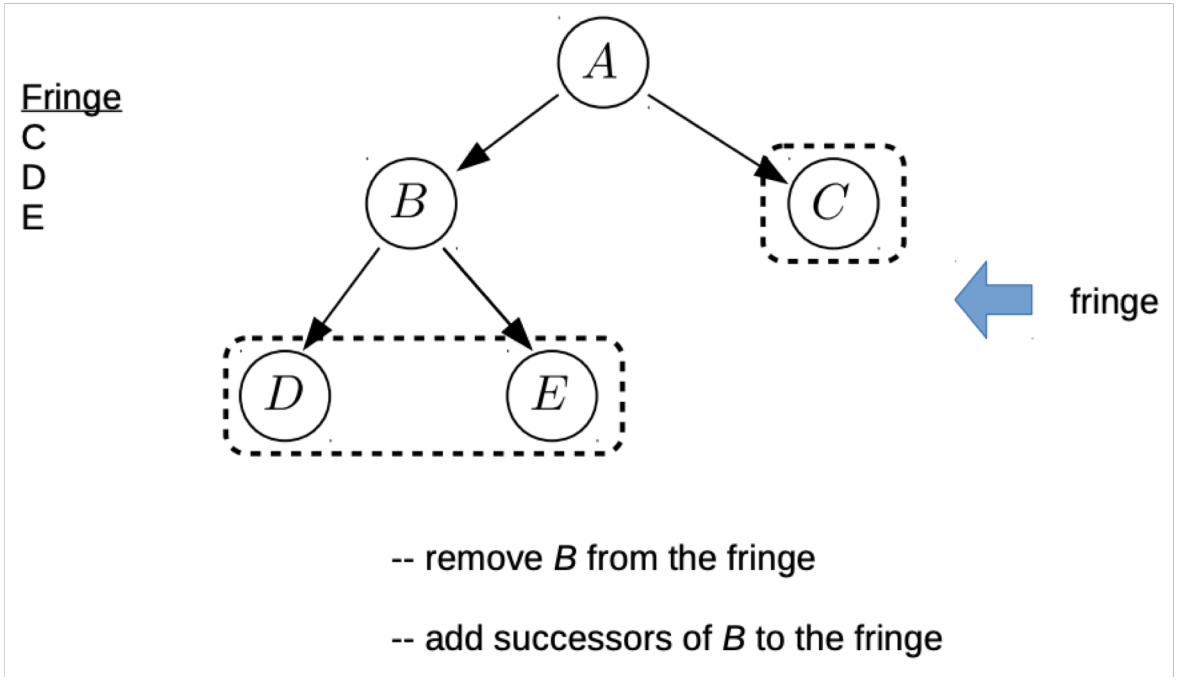
Graph Search 2: BFS

Maintaining a “fringe”



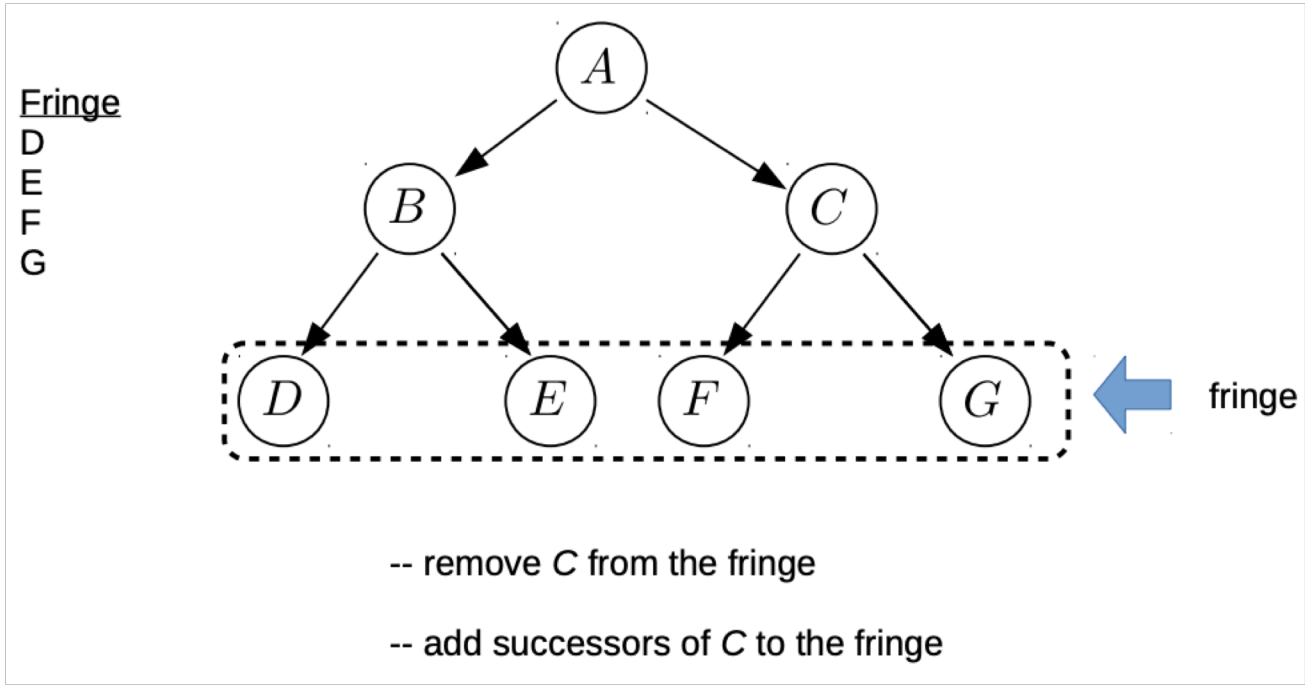
Graph Search 2: BFS

Maintaining a “fringe”



Graph Search 2: BFS

Maintaining a “fringe”

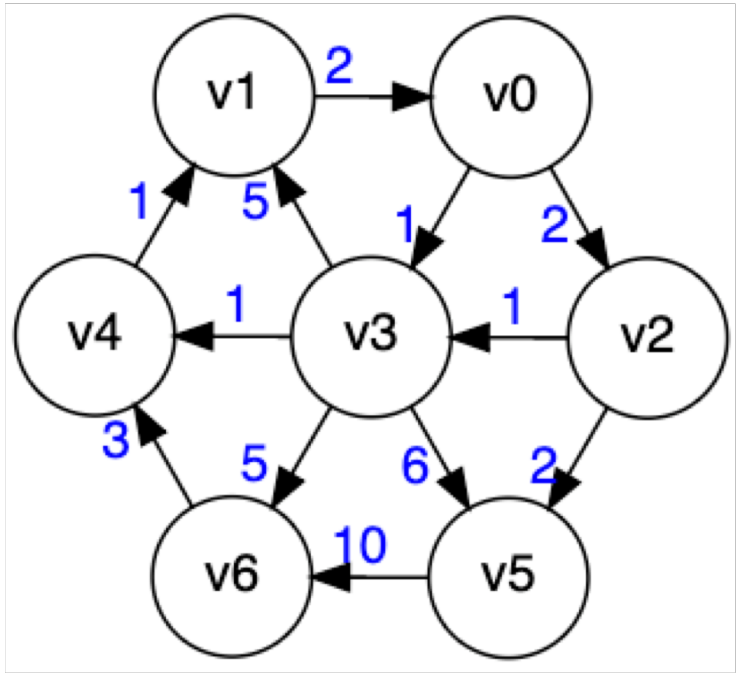


Dijkstra's Algorithm

An implementation of Breadth-first Search (BFS)

Example 1: want to get from v0 to v6...

Best path: v0 → v3 → v6



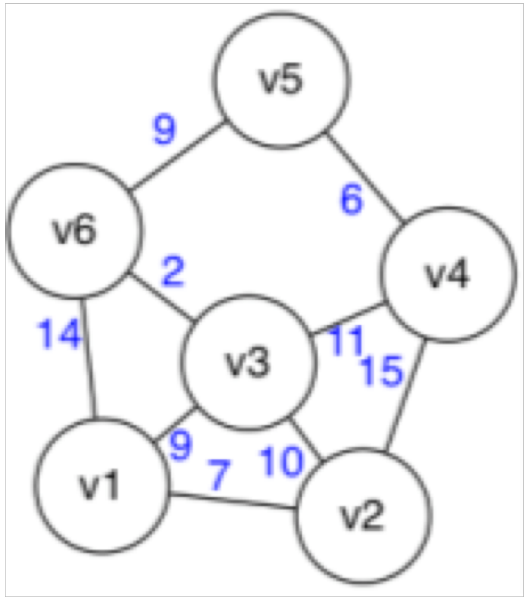
V	Known?	Distance	Path
v0	✓	0	
v1	✓	~~6~~ 3	~~v3~~ v4
v2	✓	2	v0
v3	✓	1	v0
v4	✓	2	v3
v5	✓	~~7~~ 4	~~v3~~ v2
v6	✓	6	v3

Dijkstra's Algorithm

Try it yourself!

Example 2: want to get from v1 to v5...

Best path: v1 → v3 → v6 → v5



V	Known?	Distance	Path
v1	✓	0	
v2	✓	7	v1
v3	✓	9	v1
v4		~~21~~ 20	~~v2~~ v3
v5	✓	20	v6
v6	✓	~~14~~ 11	~~v1~~ v3