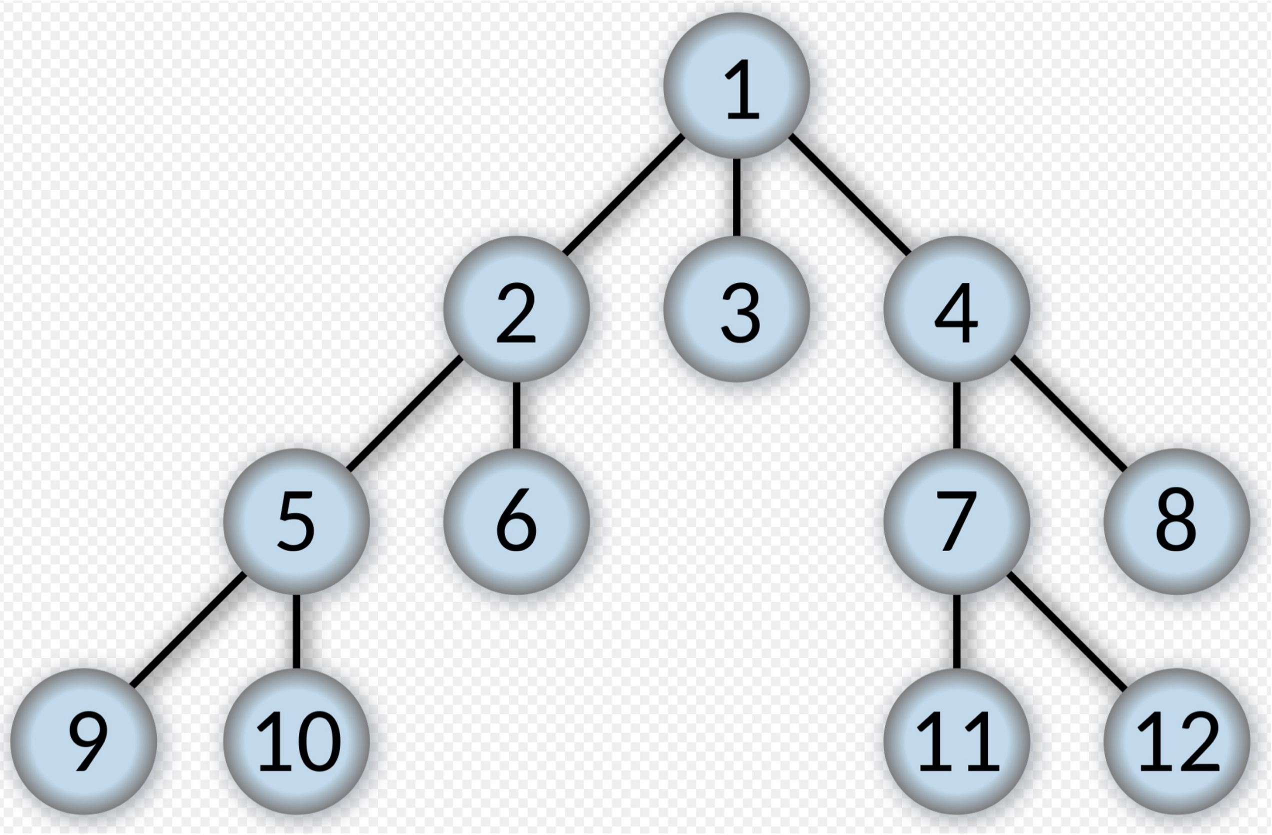
CSCI 3202: Intro to Artificial Intelligence

Uninformed search, (BFS), (DFS), Uniform cost

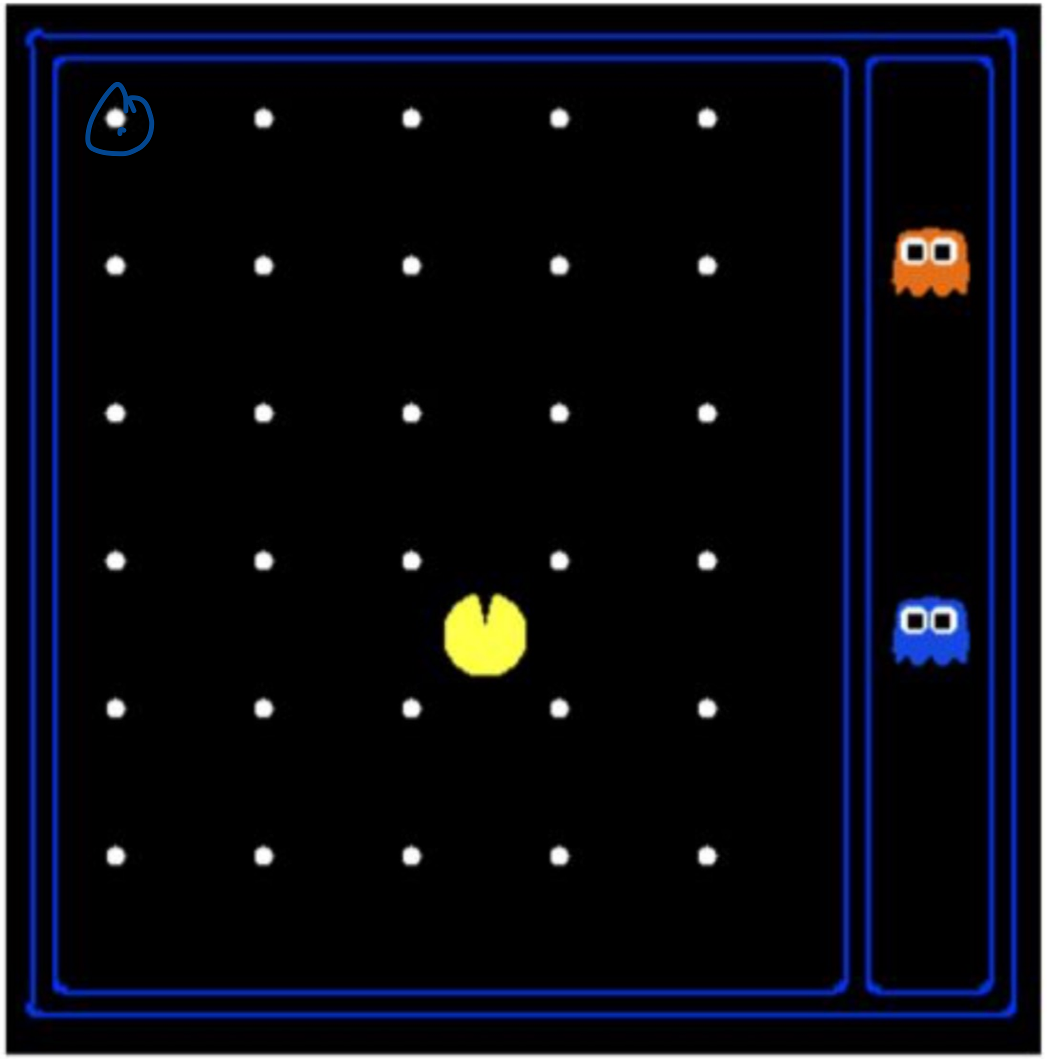
Rhonda Hoenigman

Department of Computer Science



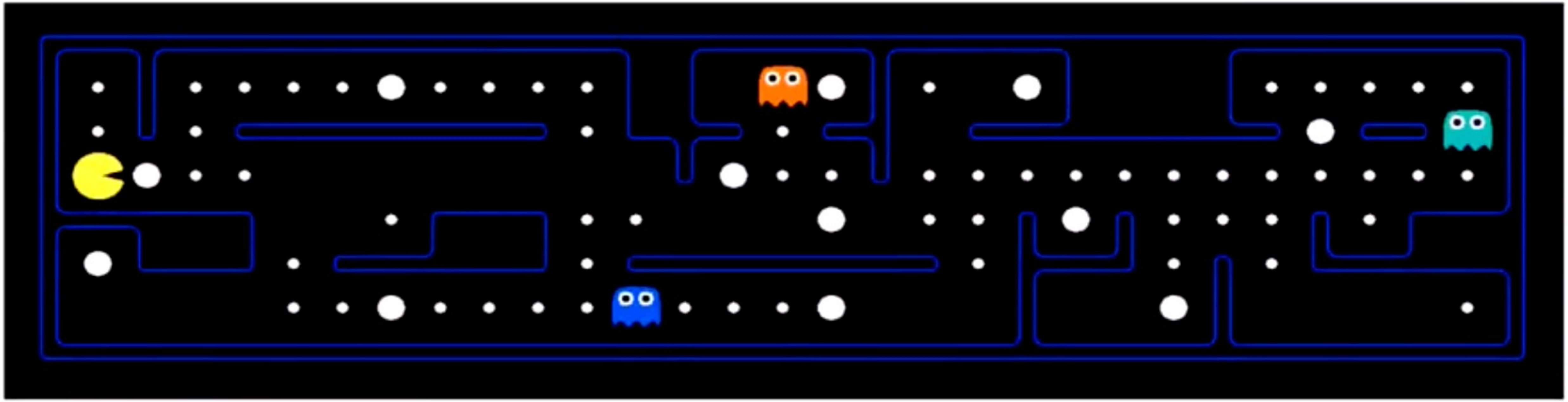
States

Example: What is the size of the state space for this Pac-Man agent? White dots are consumable food, grid is 10x12.



States Activity

Example: Suppose your goal is to eat all of the food while keeping the ghosts "scared" constantly. What information would your state space need to include?



Search

1. **State space**
   a. What are all the possible ways the world could look?
   b. Forms a directed graph.
   c. A path in the state space is a sequence of states, connected by actions.
   d. A path cost function assigns a numeric cost (might be defined as in utility) to each path.
   e. Sum of the step costs (typically)

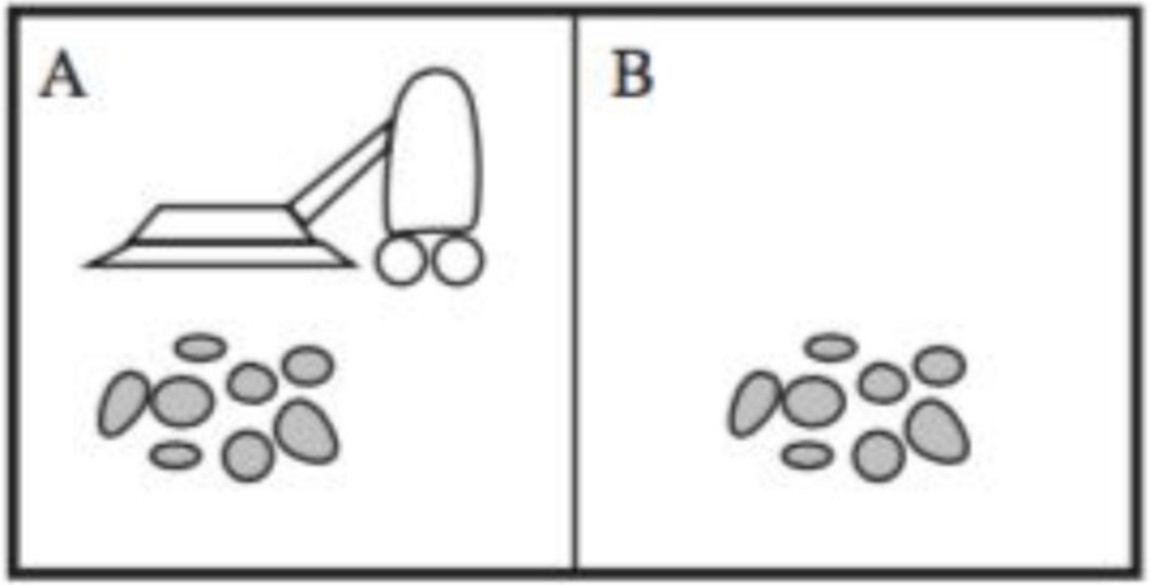
2. **Transition model**
   a. function that returns state_new that results from doing an action to state_old
   b. “successor”: any state reachable from a given state by a single action.

3. **Actions**
   a. What can the agent do? (operations on the environment)

Search

4. Initial state
    a. e.g. $[A, 'dirty']$ for the vacuum

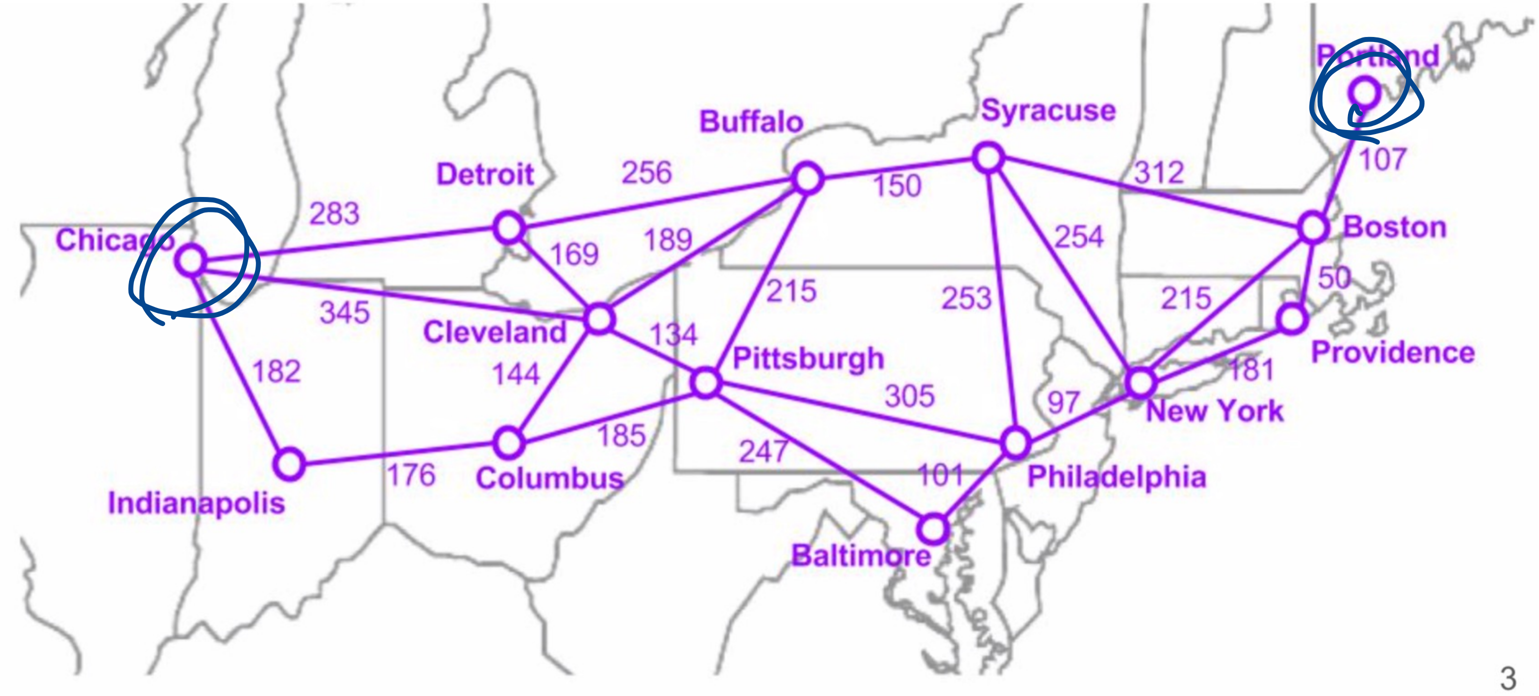
5. Goal test
    a. Determines whether a given state is the goal state.



Search

A **search problem** consists of:

1. State space
2. Transition model
3. Actions – Drive
4. Initial state –
5. Goal test –
6. Solution –



Search

Example: Traveling in the US northeast

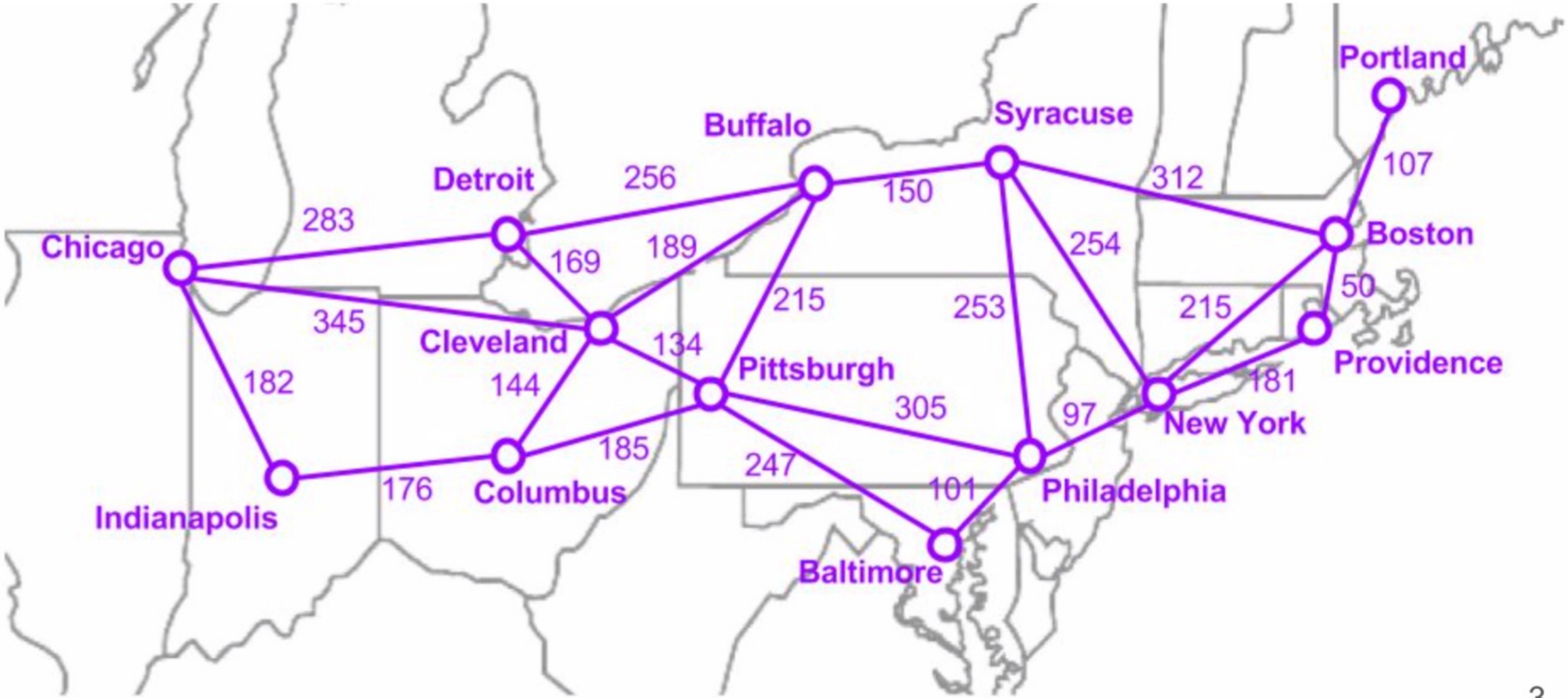
1. State space
2. Transition model
3. Actions
4. Initial state
5. Goal test
6. Solution

Search algorithms

Uninformed Search - no additional information about states beyond that in the problem definition
Informed Search - Some idea of which non-goal states are “more promising” than others

hueristic.

BFS



Search

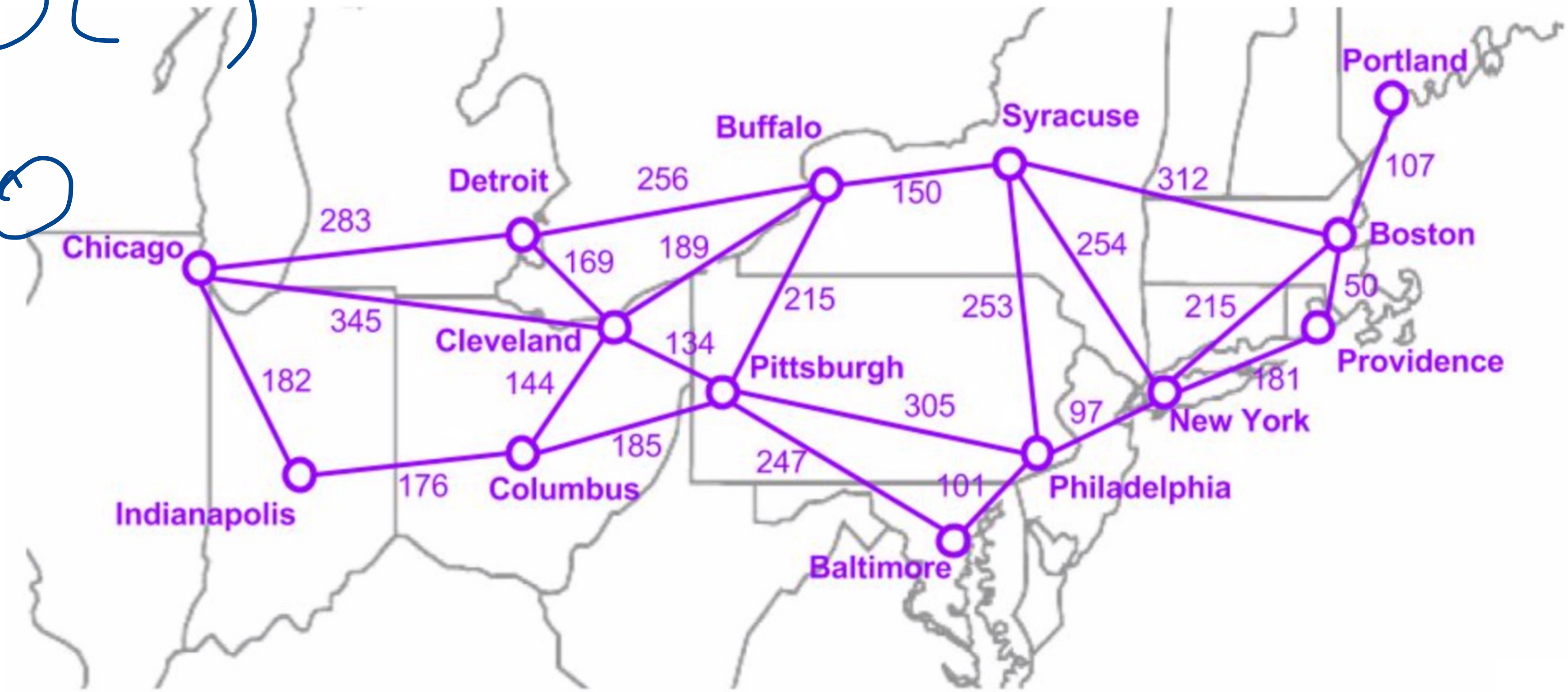
Things to think about:

Completeness: FINDS A PATH IF IT EXISTS.
RETURNS NOTICE IF PATH DOESN'T EXIST.

Optimality: → RETURNS SHORTEST POSSIBLE PATH.

Time Complexity: – O( )

Space Complexity: → O



Search strategies this week

Breadth-first search (BFS) – search across the tree before searching deeper into the tree.

UNWEIGHTED

Depth-first search (DFS) – search deeper into the tree before searching across the tree

Uniform Cost Search – BFS strategy with additional logic

A* - BFS strategy, informed

Many, many, many search algorithms built on basic premise of BFS or DFS.

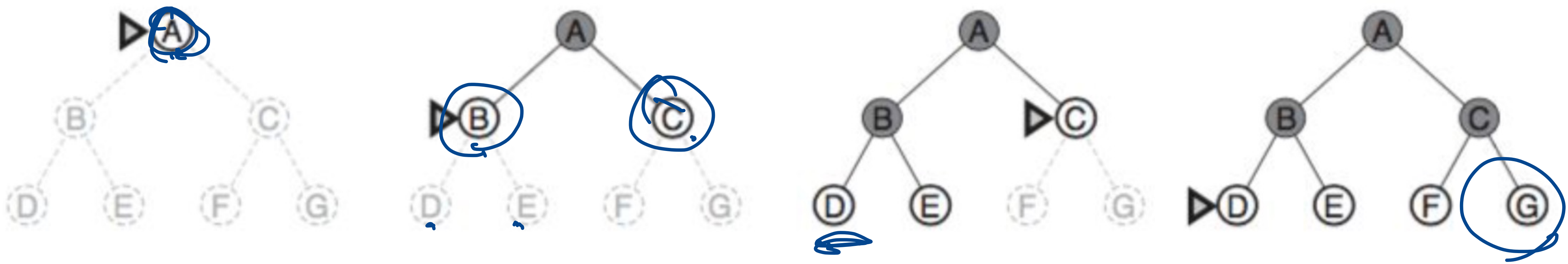
GREEDY

Breadth–first Search (BFS)

DFS

STACK
LIFO,

- Uninformed
- Expand all nodes at a given depth before proceeding into to the next layer (FIFO)
- Apply a goal test to each node



SET
Explored: A, B, C, D, E, F, G

QUEUE
Frontier: ~~B~~, ~~C~~, ~~D~~, ~~E~~, ~~F~~, G

Breadth–first Search (BFS) – implementation

```
BFS(graph, start_node, end_node):
    frontier = new Queue()
    frontier.enqueue(start_node)
    explored = new Set()

    while frontier is not empty:
        current_node = frontier.dequeue()
        if current_node in explored: continue
        if current_node == end_node: return success

        for neighbor in graph.get_neigbhors(current_node):
            frontier.enqueue(neighbor)

        explored.add(current_node)
```

Search

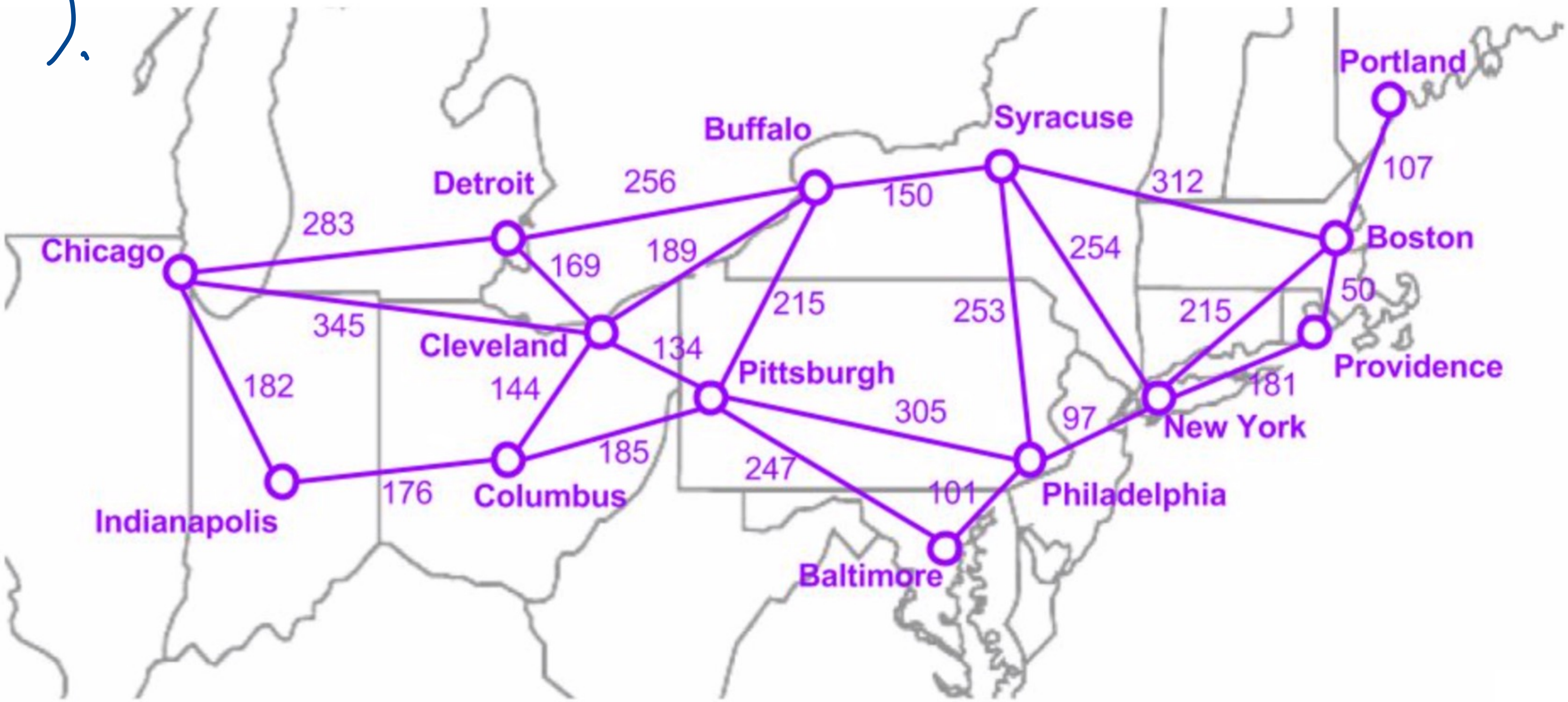
Things to think about:

Completeness:

Optimality:

Time Complexity: $O(b^d)$.

Space Complexity:



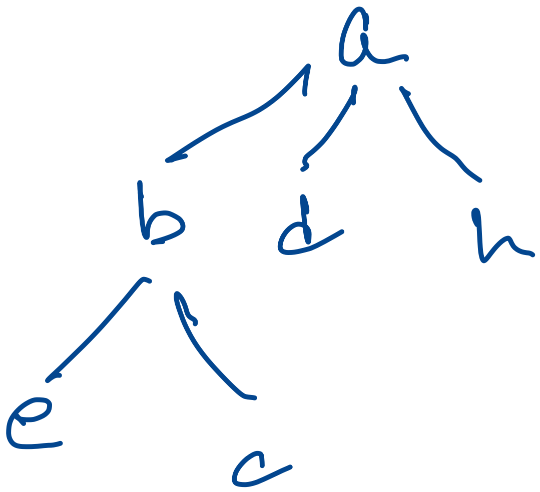
Breadth–first Search（BFS）

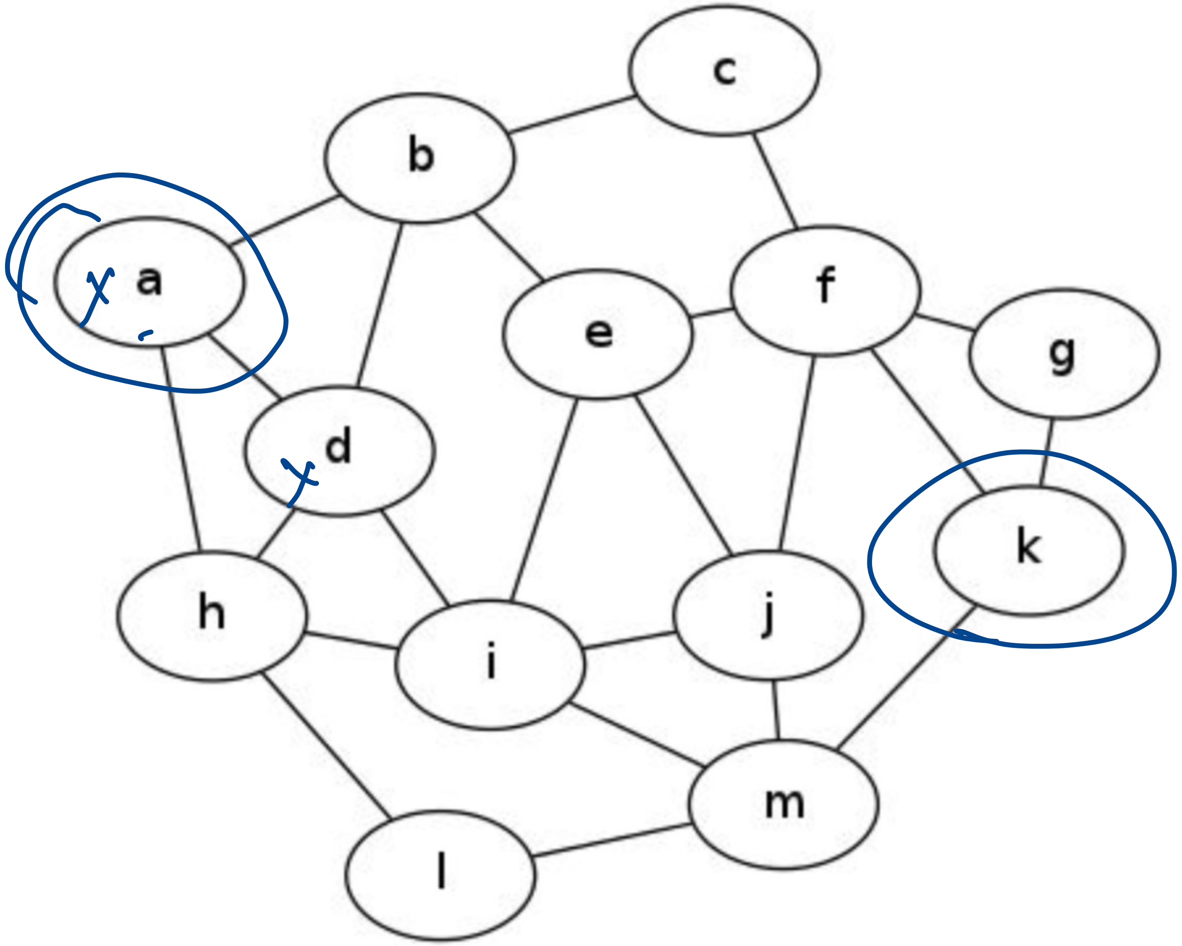
Example: Traveling in the US northeast

Define step costs:
- Number of cities to goal (unweighted)
- Miles between cities along major highways (weighted)
- Time to travel to next city (weighted)

Breadth-first Search (BFS)

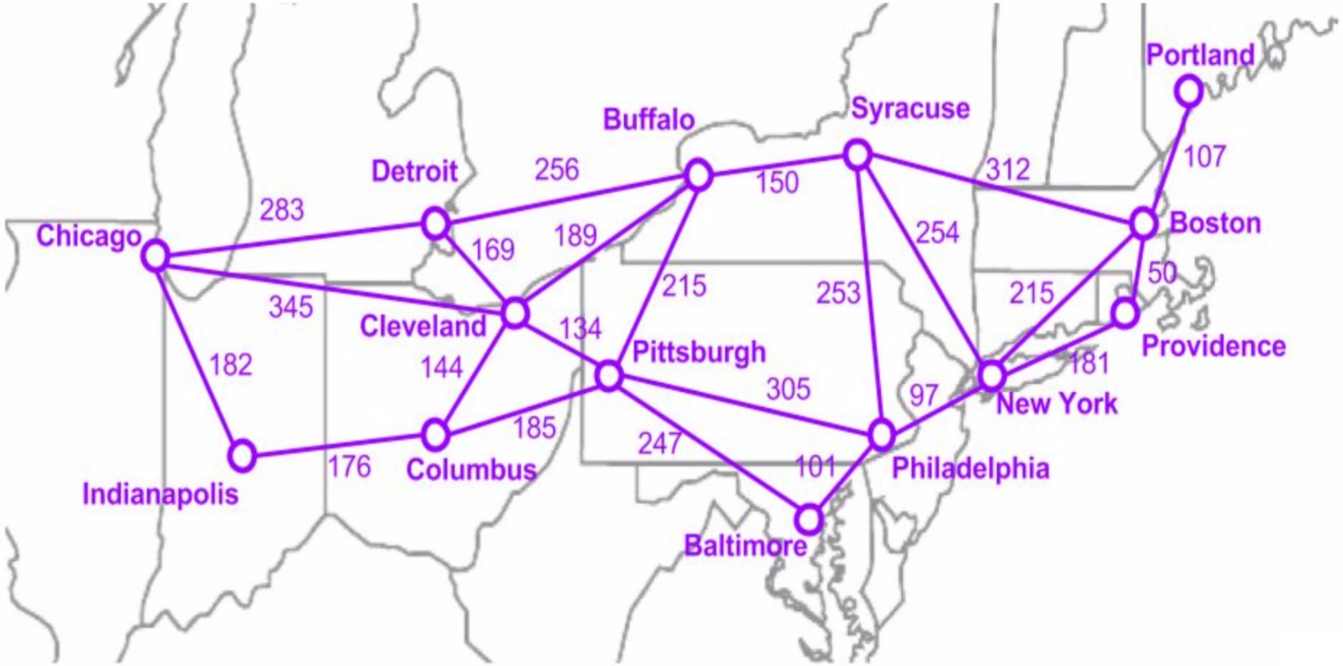
Example: Build a search tree from the nodes in the graph according to the order in which they would be expanded using BFS to find a path from a to k. Assume that nodes within a layer are expanded in alphabetical order. Edges are unweighted.



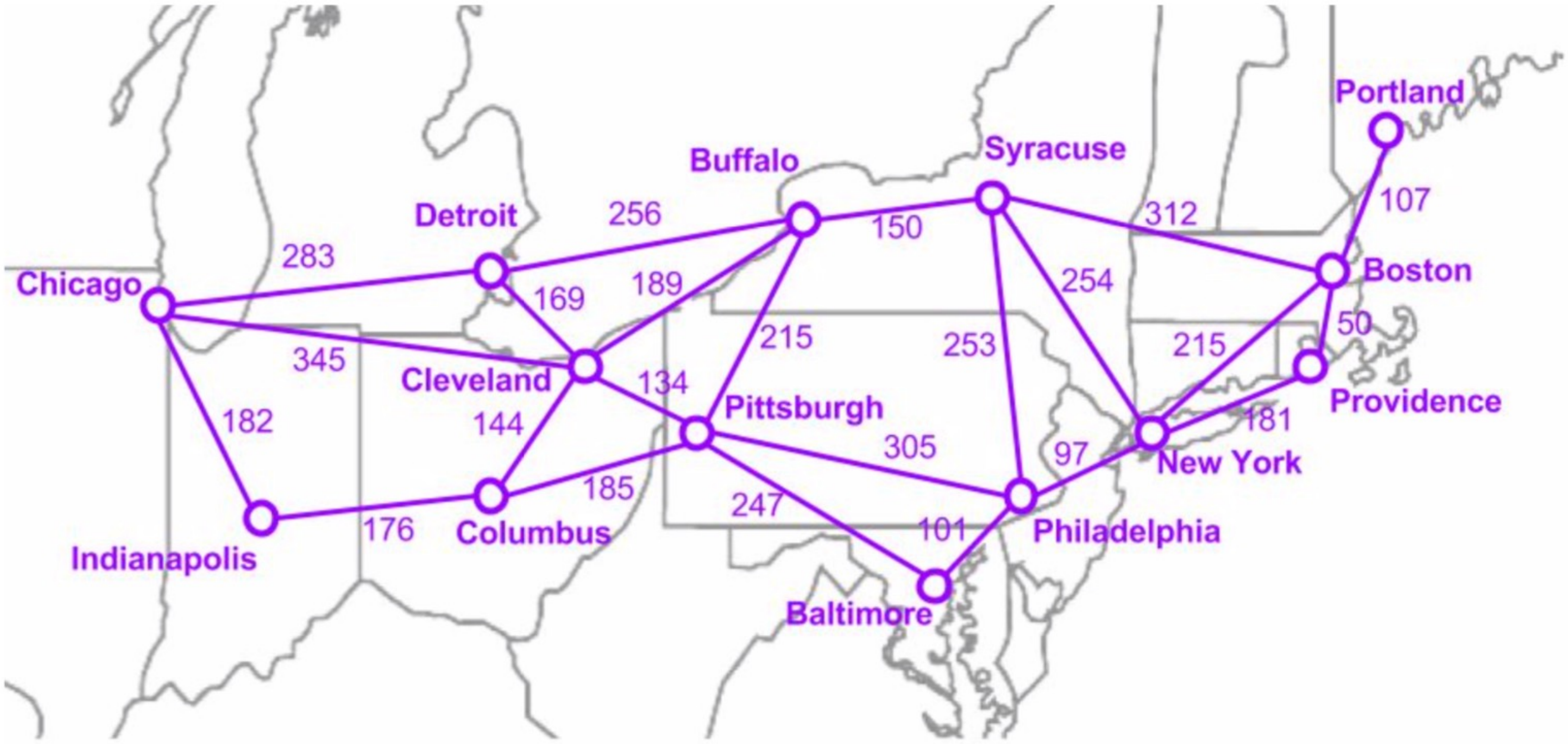


Breadth–first Search（BFS）

Example: Traveling in the northeast again. Sketch a search tree with Chicago as the initial state.



Breadth–first Search（BFS）



Complete?

Optimal?

Breadth–first Search (BFS)

Time Complexity: Suppose that each layer generates b nodes (calling b the "branching factor") and the search problem has d total layers.

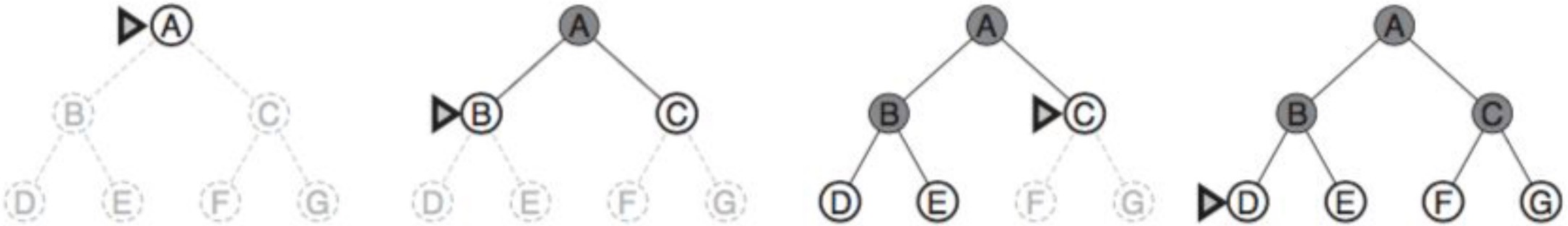
- layer 0 (root) generates $b^0 = 1$ node

- layer 1 generates $b^1 = b$ nodes

- layer 2 generates b^2 nodes

… and so on …

total: $\boldsymbol{1 + b + b^2 + b^3 + \ldots + b^d = \mathcal{O}(b^d)}$

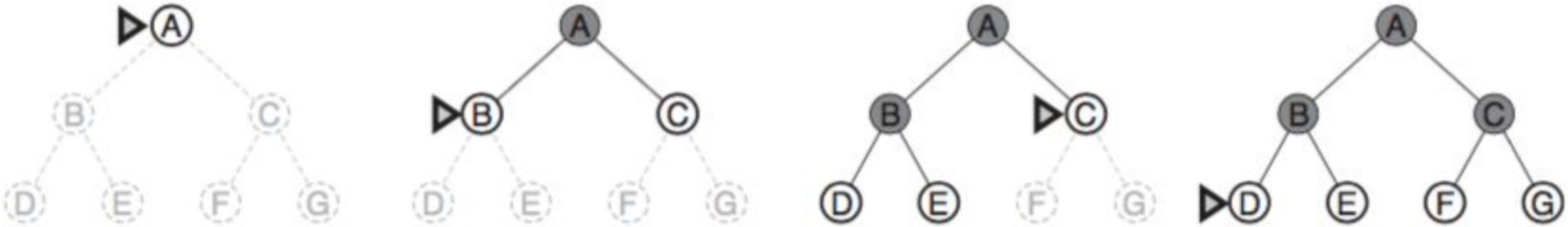


Breadth–first Search (BFS)

Space Complexity: assumes need to store every node in the explored set $= \mathcal{O}(b^{d-1})$

and every node on the frontier $= \mathcal{O}(b^{d})$

- $\boxed{\mathcal{O}(b^{d})}$



Depth_First Search (DFS) – iterative and recursive implementations

```
DFS-iterative (G, s):                                   //Where G is graph and s is source vertex
  let S be stack
  S.push( s )              //Inserting s in stack
  mark s as visited.
  while ( S is not empty):
      //Pop a vertex from stack to visit next
      v  =  S.top( )
     S.pop( )
     //Push all the neighbours of v in stack that are not visited
    for all neighbours w of v in Graph G:
        if w is not visited :
                 S.push( w )
                mark w as visited

DFS-recursive(G, s):
    mark s as visited
    for all neighbours w of s in Graph G:
        if w is not visited:
            DFS-recursive(G, w)
```

STACK, S

VISITED (SET)

V = VERTEX

GRAPH NEIGHBORS

W = NEIGHBOR

VISIT WHEN WE PUSH IT ON STACK

VISIT

LIST OF NEIGHBORS

S = SOURCE (START)

W = NEIGHBOR

VISITED (SET)

* HOW DOES RECURSION STORE VARIABLES?
  - STORE CALLING ARGUMENTS
  - STORE LOCAL VARIABLES.

  } NOT EXACTLY TRUE IN PYTHON.

* DEFAULT LIMIT OF 1000 CALLS IN PYTHON

* MEMORY AND STACK SIZE LIMITS APPLY.

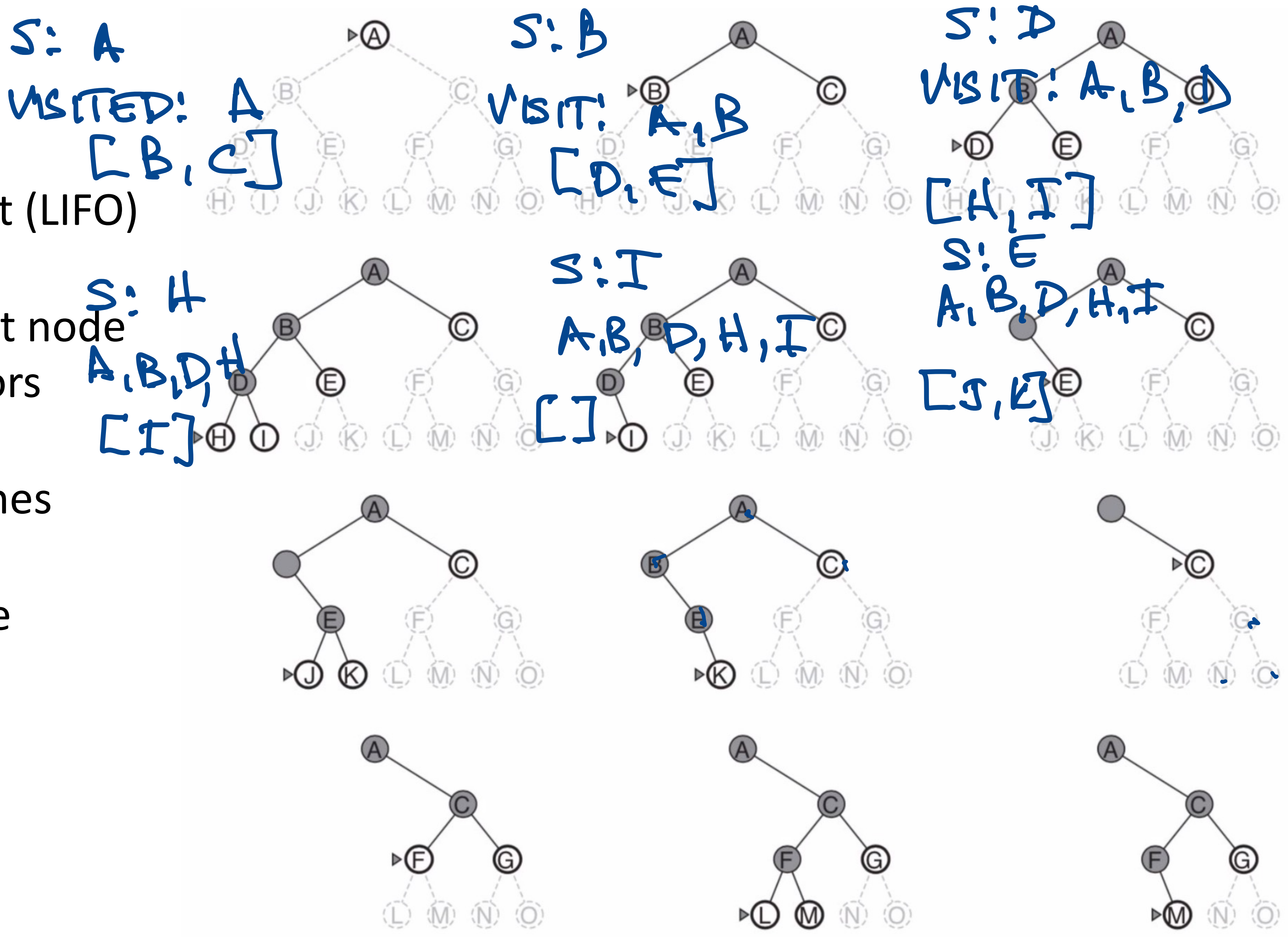
* USE ITERATIVE VERSION FOR LARGE GRAPHS OR TREES

WHAT IS THE DIFFERENCE BETWEEN A GRAPH AND TREE VERSION OF DFS?

Depth–First Search (DFS)

RECURSIVE

- Uninformed
- Expand deepest node first (LIFO)
- “Back up” to next-deepest node with unexplored successors
- Implementation determines nodes explored
  - Iterative and recursive versions

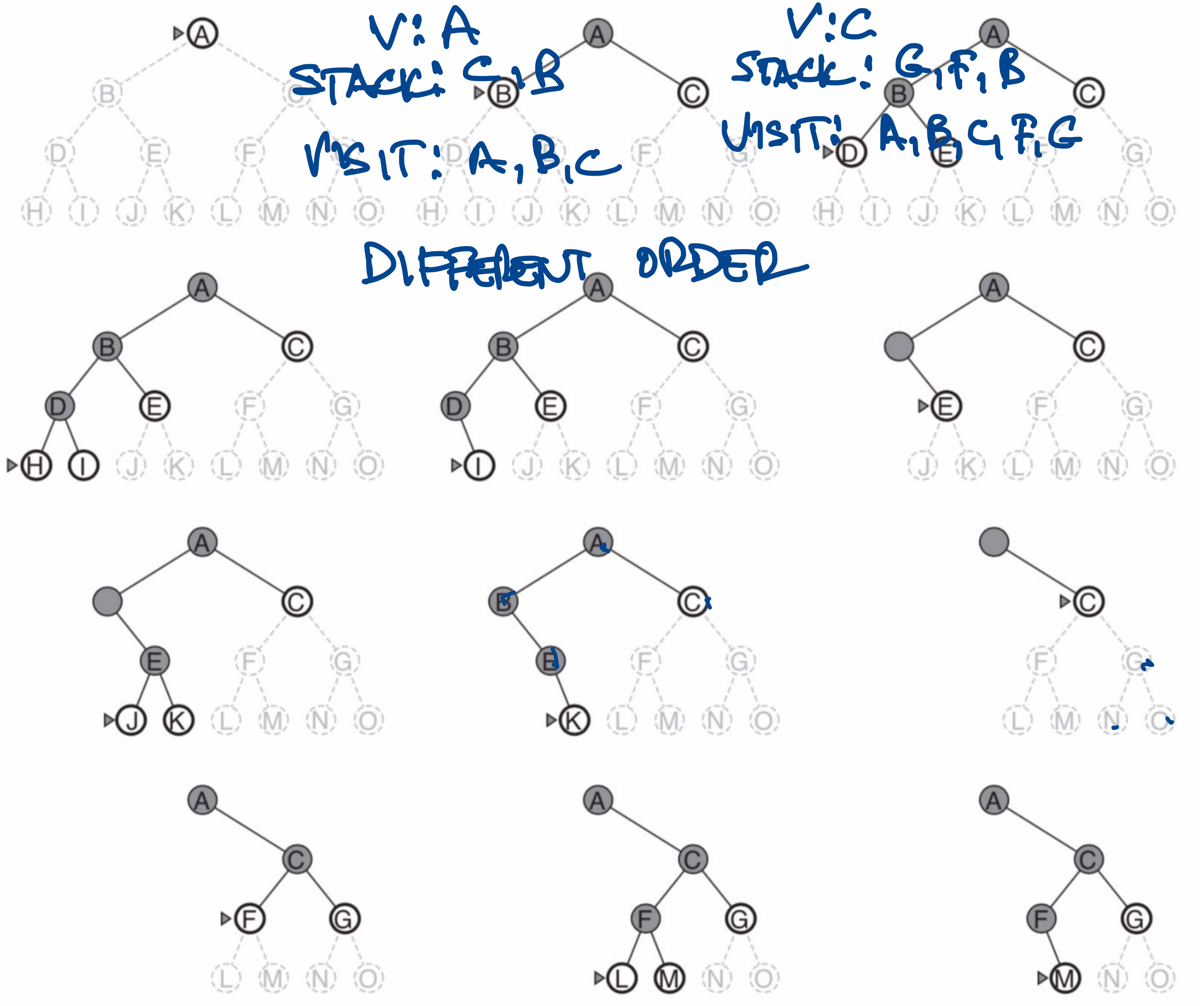


Depth_First Search (DFS)

ITERATIVE

S : A
VISIT: A
STACK

- Uninformed
- Expand deepest node first (LIFO)
- “Back up” to next-deepest node with unexplored successors
- Implementation determines nodes explored
  - Iterative and recursive versions



Depth_First Search (DFS) – iterative and recursive implementations

```
DFS-iterative (G, s):                                   //Where G is graph and s is source vertex
  let S be stack
  S.push( s )                //Inserting s in stack
  mark s as visited.
  while ( S is not empty):
      //Pop a vertex from stack to visit next
      v  =  S.top( )
     S.pop( )
     //Push all the neighbours of v in stack that are not visited
    for all neighbours w of v in Graph G:
        if w is not visited :
                 S.push( w )
                mark w as visited
```

STACK, S
VISITED (SET)
V = VERTEX
GRAPH NEIGHBORS
W = NEIGHBOR
VISIT WHEN WE PUSH IT ON STACK

```
DFS-recursive(G, s):
    mark s as visited
    for all neighbours w of s in Graph G:
        if w is not visited:
            DFS-recursive(G, w)
```

VISIT
LIST OF NEIGHBORS

S = SOURCE (START)
W = NEIGHBOR
VISITED (SET)

Depth–First Search (DFS)

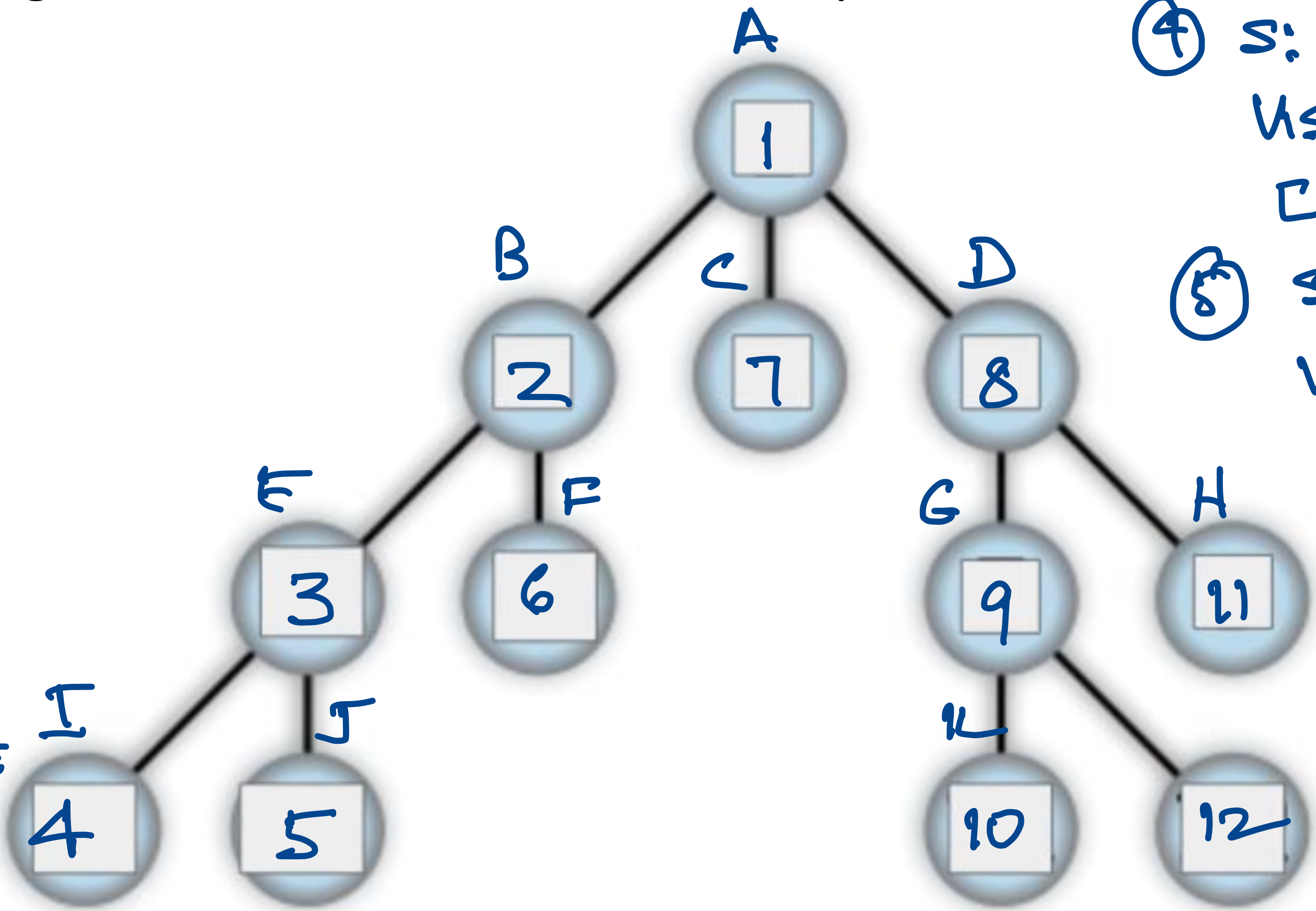
RECURSIVE

Example: Number the nodes in the search tree according to the order in which they would be added to visited using DFS. Show both iterative and recursive versions of the algorithm. Assume that the goal is not found, and nodes are processed from left to right.

1 S: A
VISIT: A
[B, C, D]

2 S: B
VISIT: B
[E, F]

3 S: E
VISIT: A, B, E
[I, J]



4 S: I
VISIT: A, B, E, I
[ ]

5 S: J
VISIT: A, B, E, I, J

6 S: F
VISIT: A, B, E, I, J, F

7

Depth–First Search (DFS)

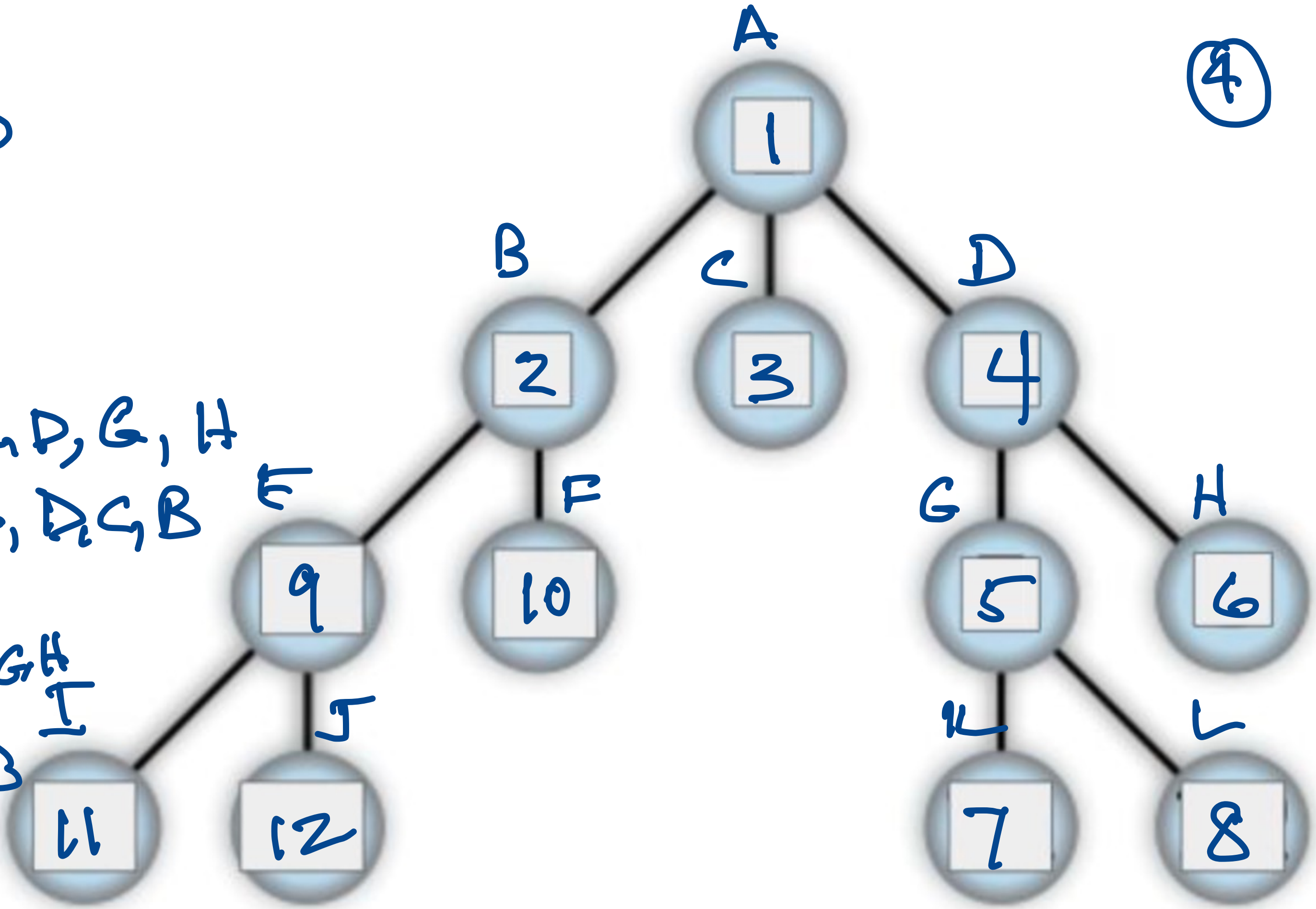
ITERATIVE

Example: Number the nodes in the search tree according to the order in which they would be added to visited using DFS. Show both iterative and recursive versions of the algorithm. Assume that the goal is not found, and nodes are processed from left to right.

1) S: A
VISIT: A, B, C, D
STACK D, C, B

2) V: D
VISIT: A, B, C, D, G, H
STACK: H, G, D, C, B

3) V: H
VISIT: A, B, C, D, G, H, I
STACK: G, D, C, B



4) V: G
VISIT: A, B, C, D, G, H, K, L
STACK: L, K, C, B

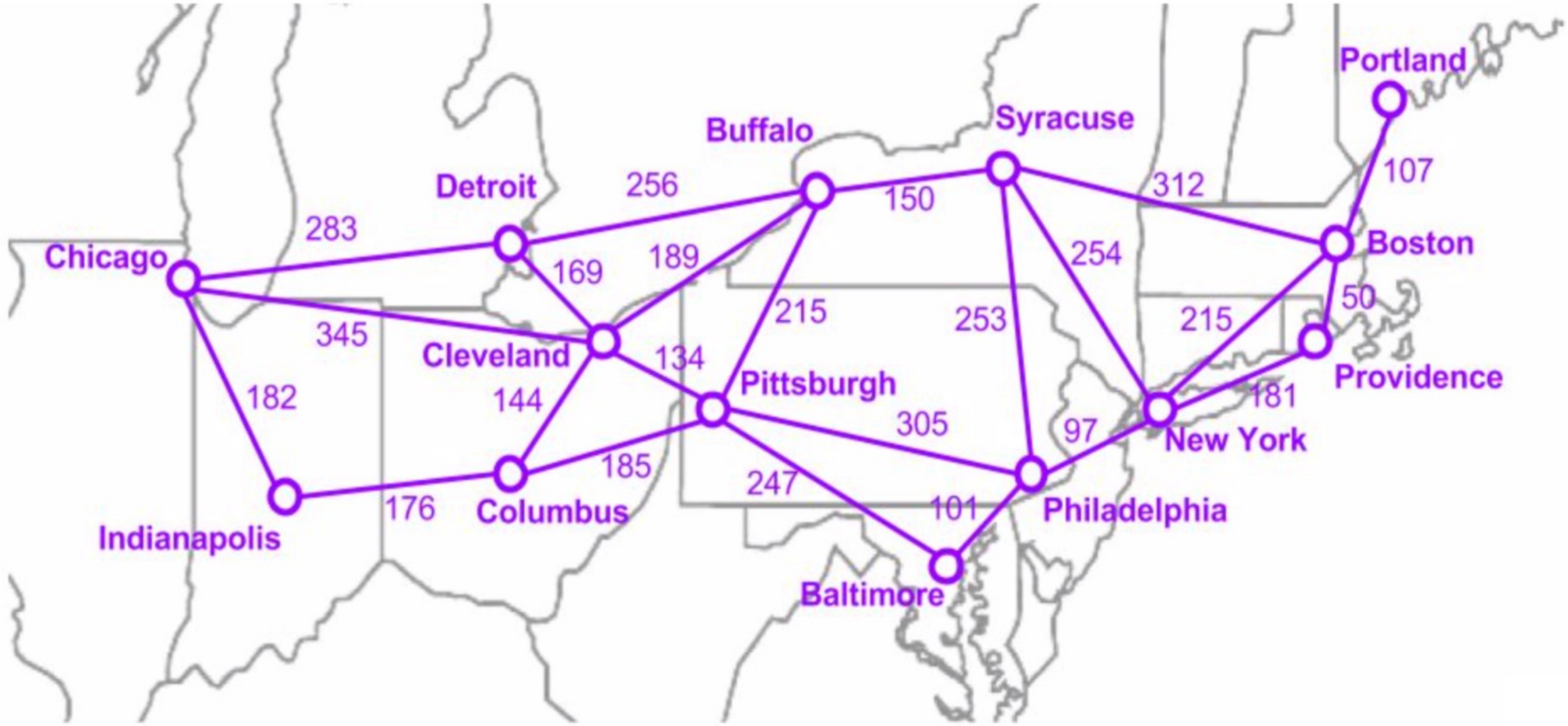
⋮

V: B
VISIT: A→H, K, L
STACK F, E

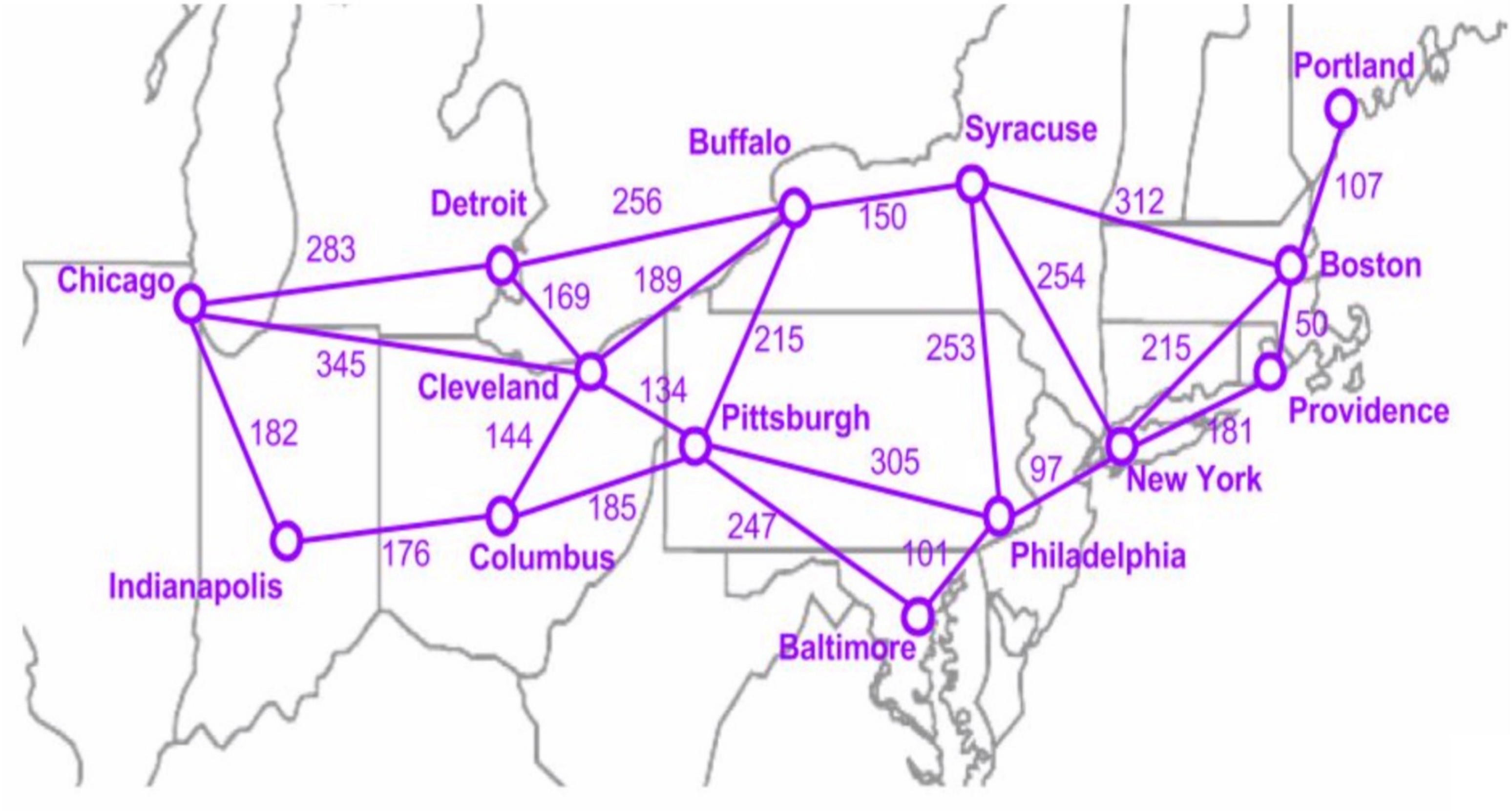
Depth–First Search (DFS)

Example: Traveling in the US northeast. **Question**: Would changing the step cost function change our DFS result?

Step costs: estimated travel time (minutes) along major highways at 5PM east coast time on a Friday



Depth–First Search (DFS)



Complete?

Optimal?

Depth–First Search（DFS）

Time Complexity:

- branching factor b
- maximal depth of m layers
- shallowest goal state in layer d

➢ might need to generate all b^m states

➢ could be substantially more than just going to shallowest goal state b^d

➢ total worst case: $\mathcal{O}(b^m)$