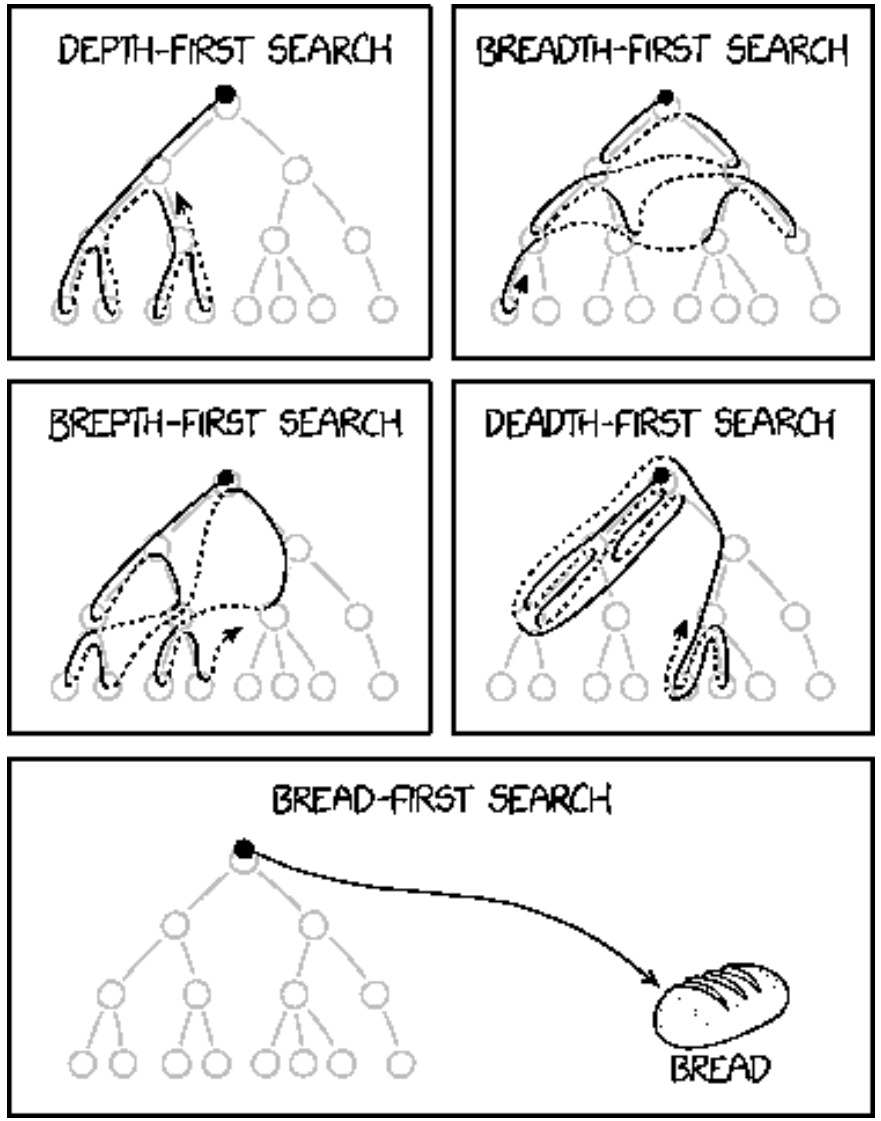
CSCI 3104: Algorithms

Lecture 13: Graph Search, Breadth First Search, Dijkstra's Algorithm, Minimum Spanning Tree

Rachel Cox

Department of Computer Science



What will we learn today?

- ❑ Graph Search
- ❑ Breadth-First Search
- ❑ Depth-First Search
- ❑ Dijkstra’s Algorithm
- ❑ Minimum Spanning Trees

Intro to Algorithms, CLRS:
Sections 22.2, 23.1, 24.3

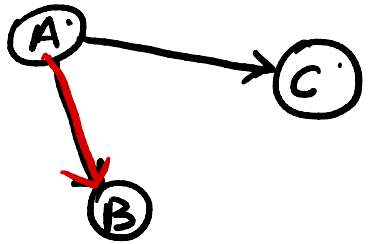
Definition of A Graph

- 7 bridges of Konigsberg
- A simple path is one with no repeated vertices.

- A **graph** is a collection of vertices joined by edges. $G = (V, E)$
- Graph G is composed of a set of vertices V and a set of edges E, where an edge is defined as $e = (i, j) \in V$.

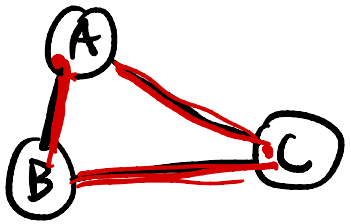
Types of Graphs:

- Directed Graph



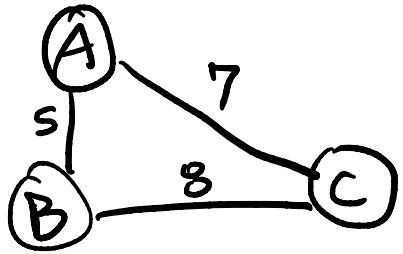
e.g. Twitter

- Undirected Graph



e.g. Facebook

- weighted graph



e.g.
- nodes are towns
- edges are roads
- values are distances

Graph Search

* **Uninformed Search** breadth-first search, depth-first search
  - no information about the goal ahead of time

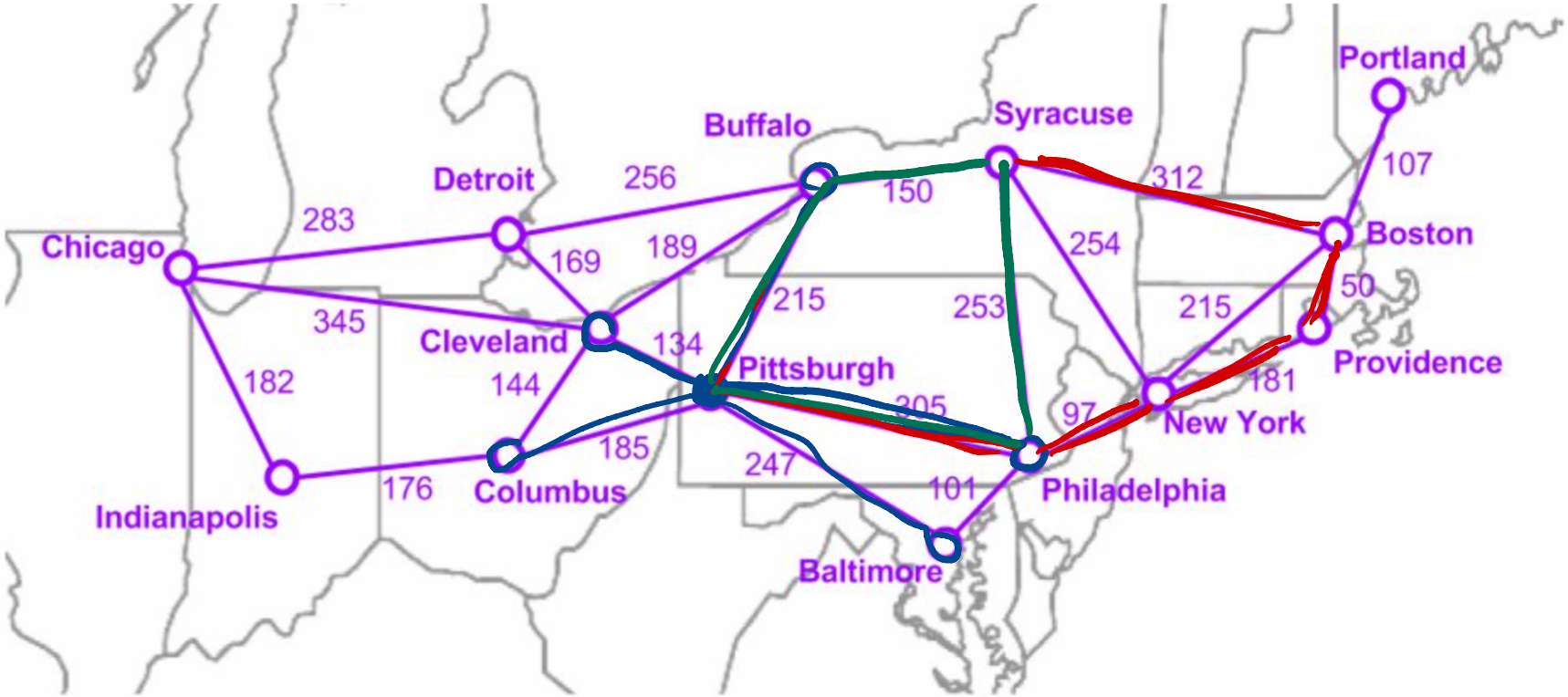
Informed Search

Graph Search

A few variables:

Branching factor: *b*, maximum number of successors of any node

Depth: *d*, the depth (in the search tree) of the shallowest goal node



Graph Search

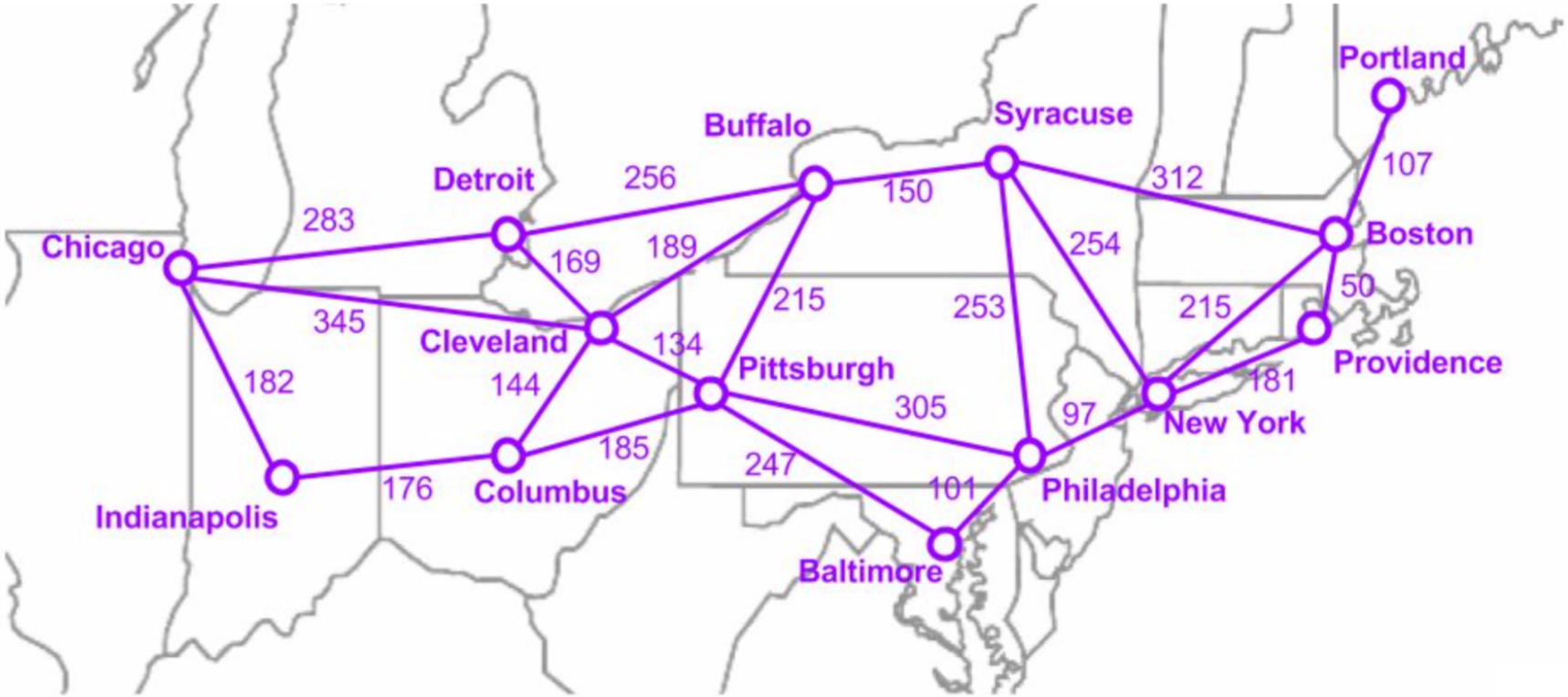
Things to think about:

Completeness: Is the algorithm guaranteed to find a solution, when one exists?

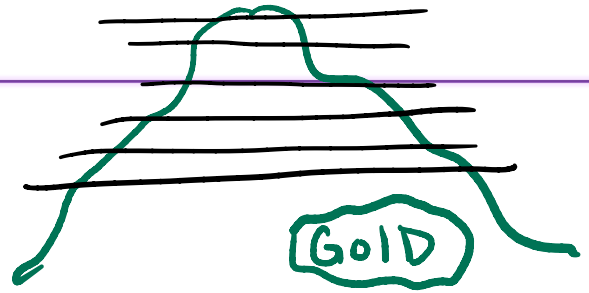
Optimality: Is the algorithm guaranteed to find the optimal solution (i.e. lowest path cost)

Time Complexity: How long does it take to find a solution?

Space Complexity: How much memory is needed to find the solution?



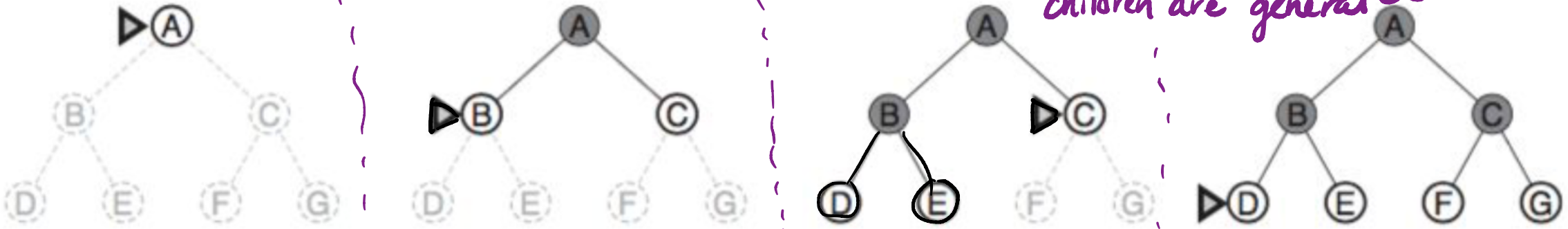
Breadth-first Search (BFS)



First in First out queue

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

- A node is generated when it is created, added to queue (aka frontier)
- A node is expanded when its children are generated.



Explored:	{ }	{A}	{A, B}	{A, B, C}
Frontier:	A	B, C	C, D, E	D, E, F, G . .

Breadth–first Search (BFS)

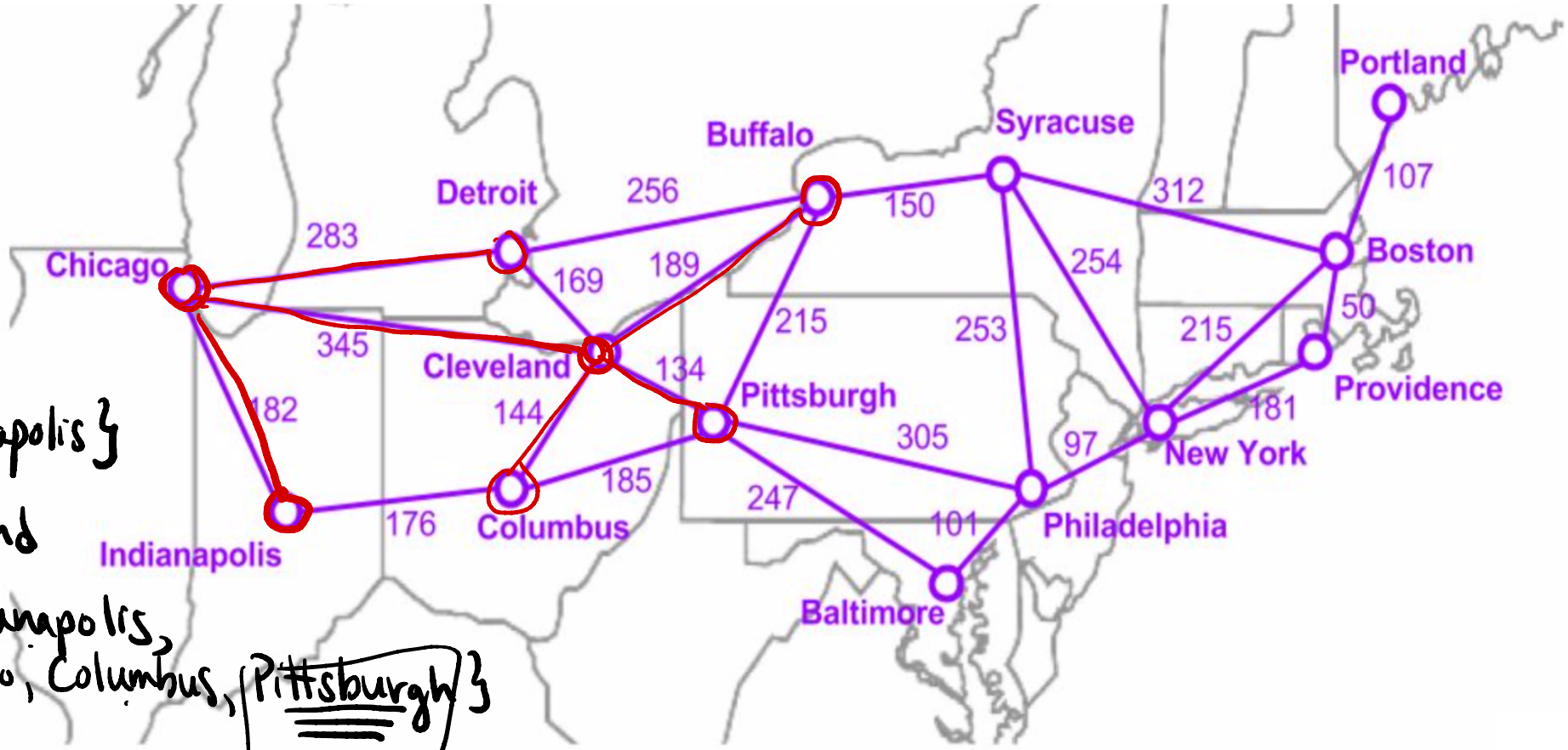
Example: Traveling in the US northeast

Path: Chicago → Cleveland → Pittsburgh

Start: Chicago

Goal: Pittsburgh

Step costs: miles between cities along major highways



Step 1:

Explored: Chicago

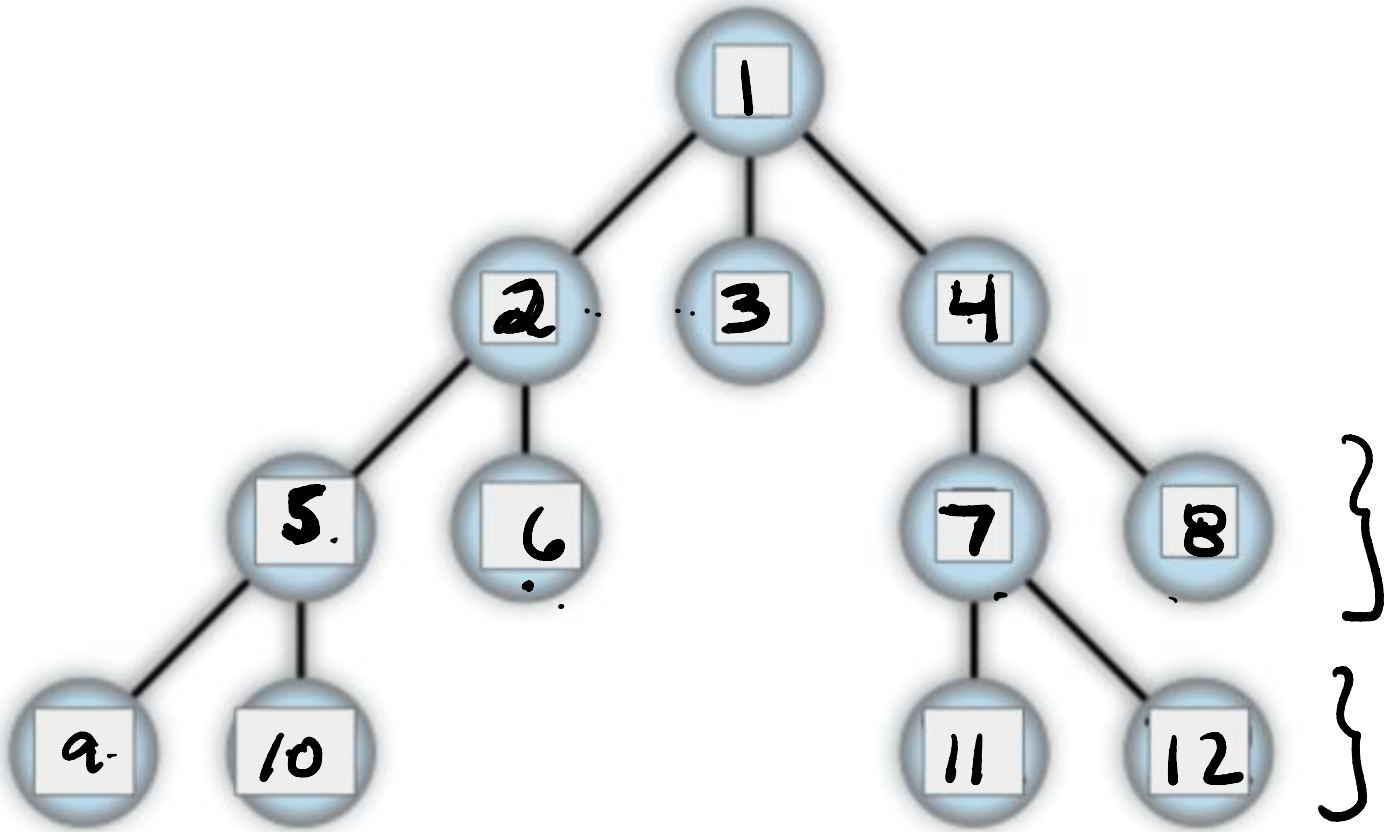
Frontier: {Cleveland, Detroit, Indianapolis}

Step 2: Explored: Chicago, Cleveland

Frontier: {Detroit, Indianapolis, Buffalo, Columbus, Pittsburgh}

Breadth–first Search (BFS)

Example: Number the nodes in the search tree according to the order in which they would be expanded using BFS. Assume that the goal is not found, and nodes within a layer are expanded from left to right.



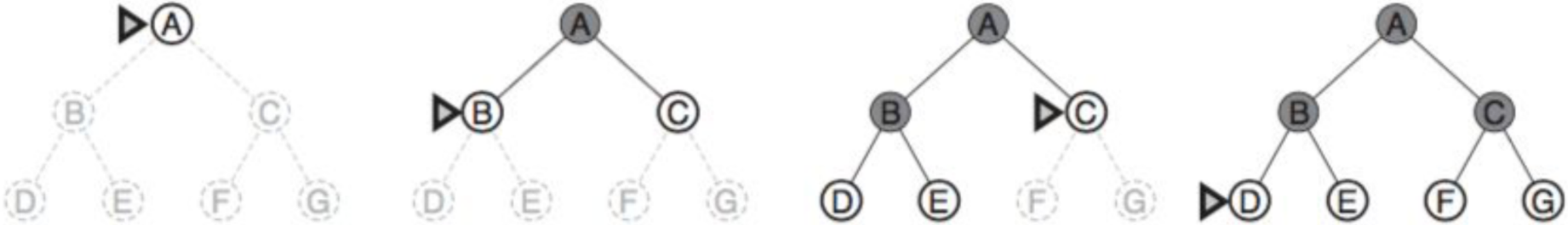
Breadth–first Search (BFS)

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



Breadth–first Search (BFS)

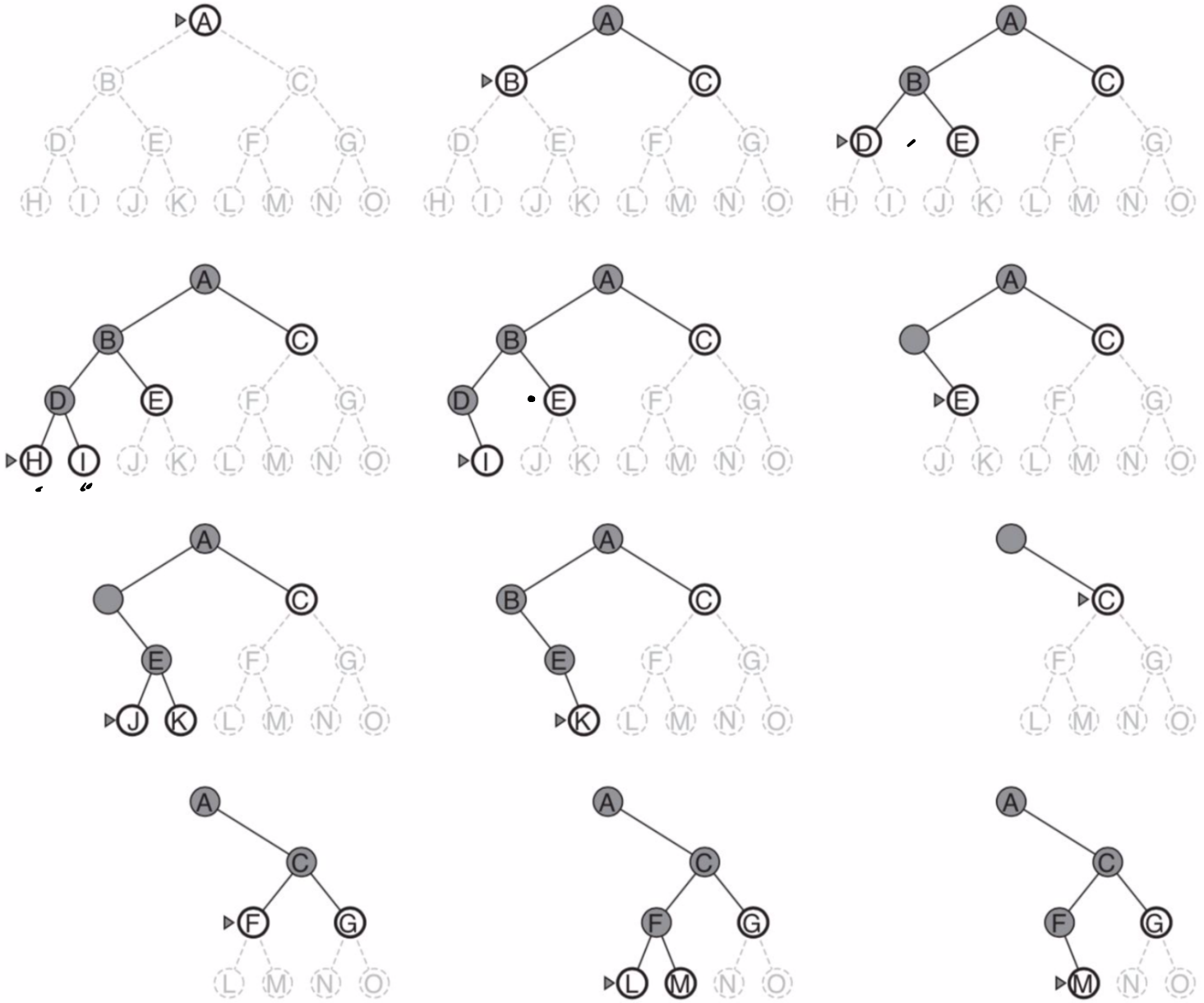
Memory requirements are a problem.

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

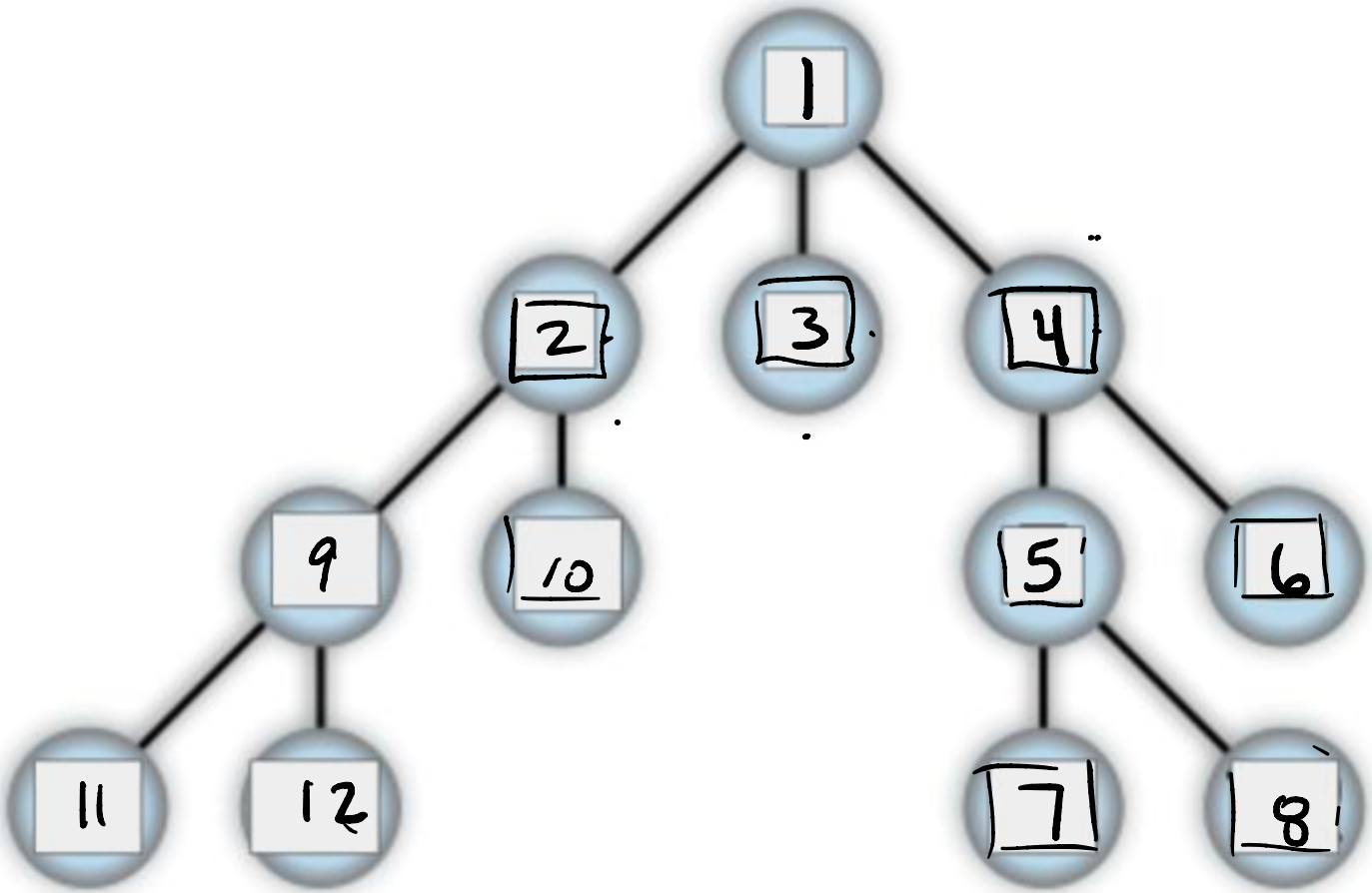
Depth–First Search (DFS)

- Uninformed
- Expand deepest node first (LIFO)
  aka stack
- “Back up” to next-deepest node with unexplored successors



Depth–First Search (DFS)

Example: Number the nodes in the search tree according to the order in which they would be generated using DFS. Assume that the goal is not found, and nodes are added to the frontier stack from left to right.



Depth–First Search (DFS)

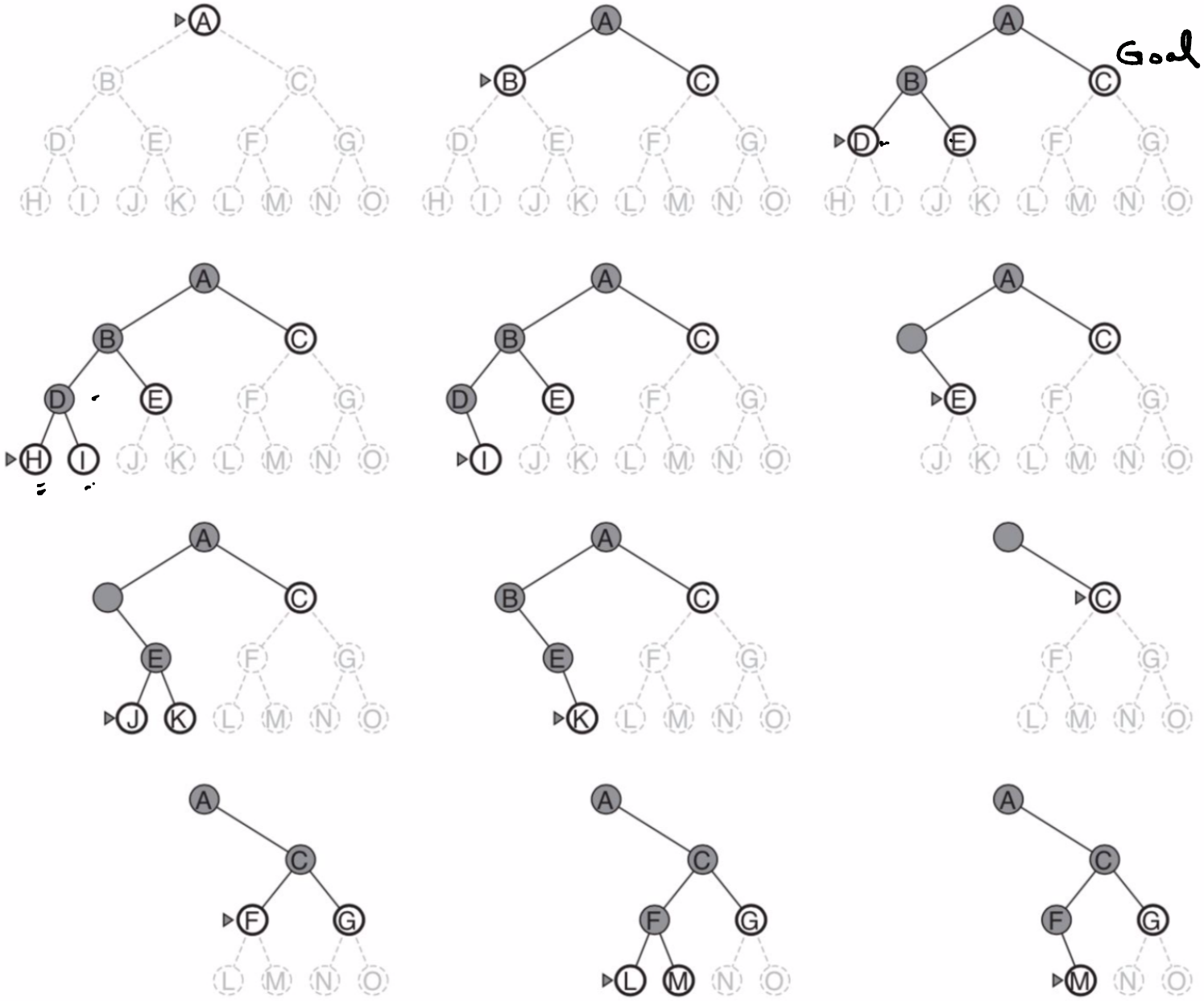
Time Complexity:

- 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: $\mathcal{O}(b^m)$



Dijkstra's Algorithm

An algorithm for finding the shortest path between nodes on a graph.

→ many variants

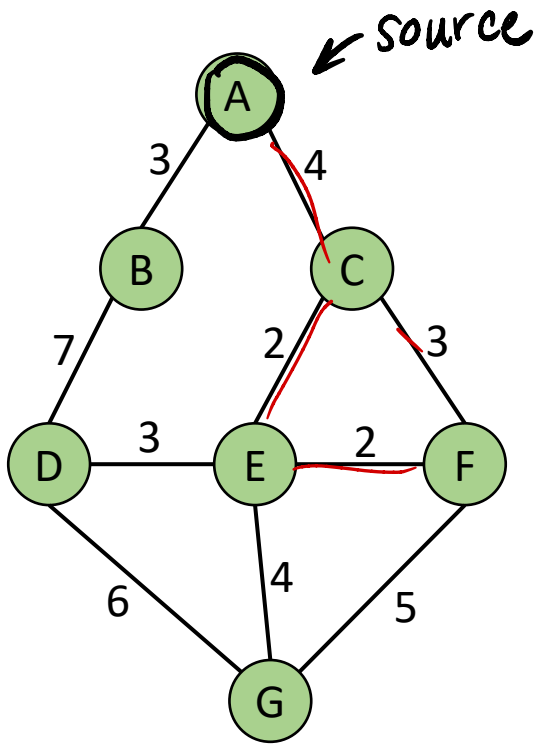
Here: Look to find the shortest path to all nodes.

→ Greedy algorithm to find the optimal solution (minimum/shortest path)

- Note: BFS only considers the number of steps, it doesn't consider the step cost.

Dijkstra's Algorithm

Example: Use Dijkstra's Algorithm to find the shortest path to each node. Let A be the source vertex.



Idea: Begin by initializing

dist(Source) = 0

distance(Vertex) = ∞ ← all vertices

Queue = {(A, 0), (B, ∞), (C, ∞), (D, ∞), (E, ∞), (F, ∞), (G, ∞)}

Current = A

Adjacent = B, C

- for v = B, C

v = B: d = dist(A) + edge(A, B)
= 0 + 3
= 3

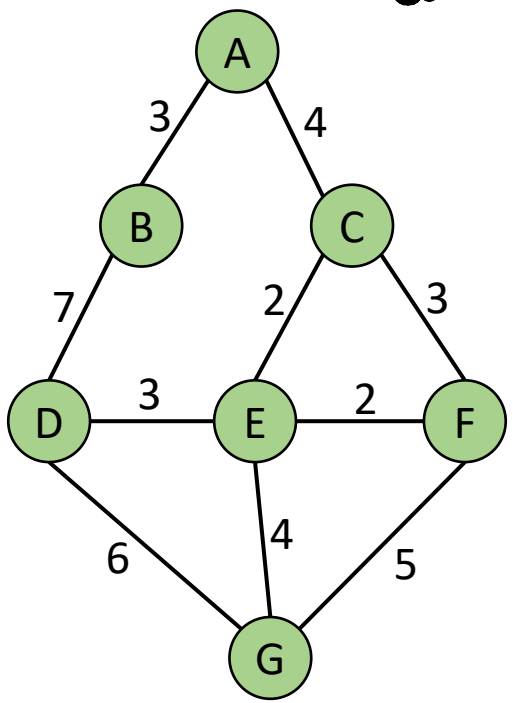
if 3 < dist(B)
dist(B) = 3
B.previous = A

v = C
d = dist(A) + edge(A, C)
= 4
if 4 < dist(C)
dist(C) = 4 ; C.previous = A

Vertex	Shortest Distance from Source	Previous
(Source) ~~A~~	0	
→ B	~~∞~~ 3	
C	~~∞~~ 4	
D	∞	
E	∞	
F	∞	
G	∞	

Dijkstra's Algorithm

Example: Use Dijkstra's Algorithm to find the shortest path to each node. Let A be the source vertex.



So now, Queue = {(B,3), (C,4), (D,∞), (E,∞), (F,∞), (G,∞)}

Current = B

B.adjacent = D

for v = D

d = dist(B) + edge(B,D)

= 3 + 7

= 10

if 10 < dist(D)

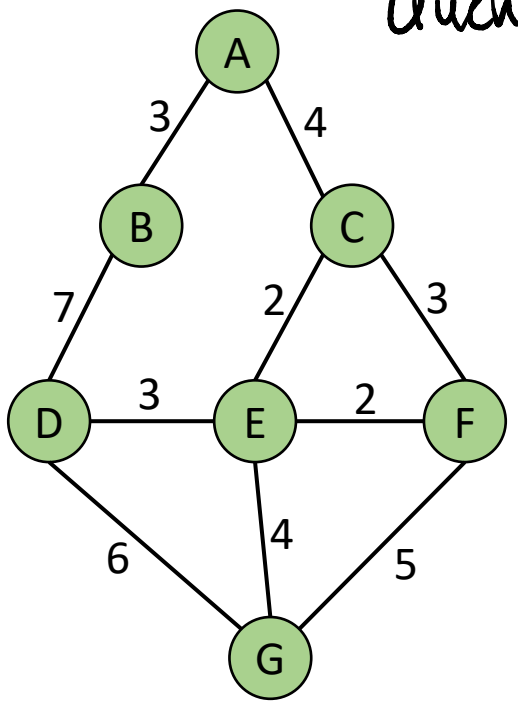
dist(D) = 10

D.previous = B

Vertex	Shortest Distance from Source	Previous
A	0	
B	3	A
C	4	A
D	~~∞~~ 10	B
E	∞	
F	∞	
G	∞	

Dijkstra's Algorithm

Example: Use Dijkstra's Algorithm to find the shortest path to each node. Let A be the source vertex.



Queue = {(C,4), (D,10), (E,∞), (F,∞), G,∞)}

* Current = C
C.adjacent = E, F

for V = E

d = dist(C) + edge(C,E)
= 4 + 2
= 6

if 6 < dist(E) (=∞)
dist(E) = 6
E.previous = C

for V = F

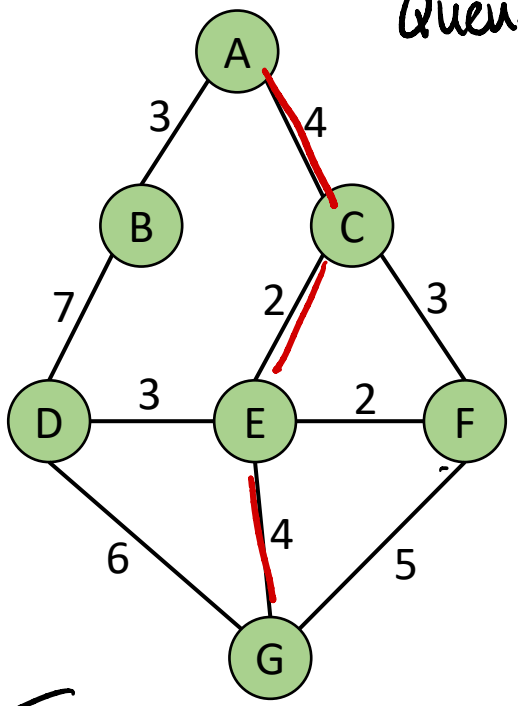
d = dist(C) + edge(C,F)
= 4 + 3

if 7 < dist(F) (=∞)
dist(F) = 7
F.previous = C

Vertex	Shortest Distance from Source	Previous
A	~~∞~~ 0	
B	~~∞~~ 3	A
C.	4	A
D	10	B
E.	~~∞~~ 6	C.
F	~~∞~~ 7	C
G	∞	

Dijkstra's Algorithm

Example: Use Dijkstra's Algorithm to find the shortest path to each node. Let A be the source vertex.



Queue: $\{(D,10), (E,6), (F,7), (G,\infty)\}$

Current : E

E adjacent = D, F, G

for v = F
d = dist(E) + edge(E,F)
= 6 + 2
= 8
if 8 < dist(F) = 7 No

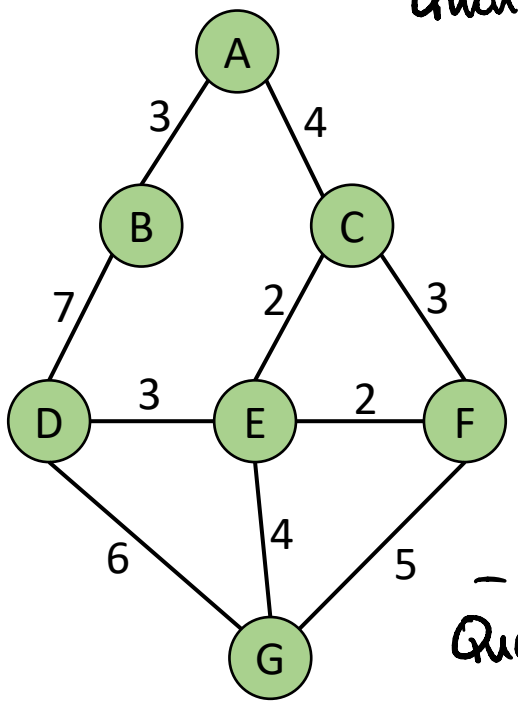
for v = D
d = dist(E) + edge(E,D)
= 6 + 3 = 9
if 9 < dist(D) = 10
dist(D) = 9 D.previous = E

for v = G
d = dist(E) + edge(E,G)
= 6 + 4 = 10
if 10 < dist(G) = ∞
dist(G) = 10 , G.prev = E

Vertex	Shortest Distance from Source	Previous
A	0	
B	3	A
C	4	A
D	~~10~~ 9	E
E	6	C
F	7	C
G	~~∞~~ 10	E

Dijkstra's Algorithm

Example: Use Dijkstra's Algorithm to find the shortest path to each node. Let A be the source vertex.



Queue: = {(D, 9), (F, ~~7~~), (G, 10)}

current = F

adjacent = G

for v = G

d = dist(F) + edge(F, G)

= 7 + 5 = 12

if 12 < dist(G) = 10 No

Queue: {(D, ~~9~~), (G, 10)}

current = D

adjacent = G

for v = G

d = dist(D) + edge(D, G)

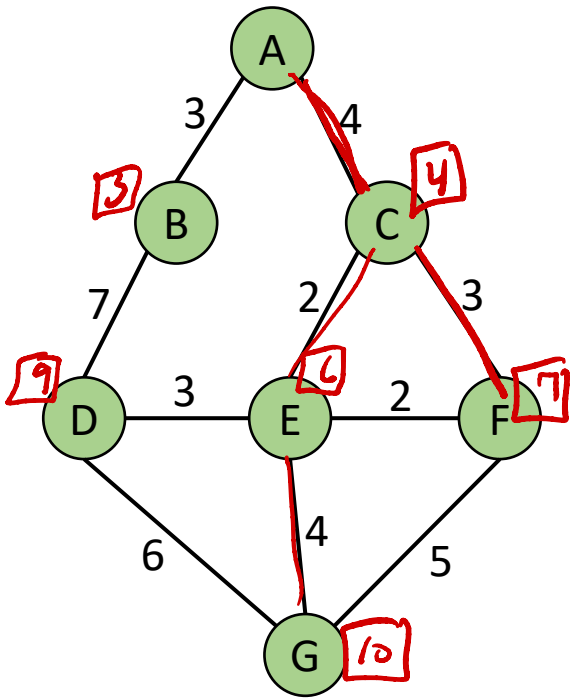
= 9 + 6 = 15

no update

Vertex	Shortest Distance from Source	Previous
A	0	
B	3	A
C	4	A
D	9	E
E	6	C
F	7	C
G	10	E

Dijkstra's Algorithm

Example: Use Dijkstra's Algorithm to find the shortest path to each node. Let A be the source vertex.



Queue = { (G, 10) }

Current = G

adjacent = ∅

exit . .

e.g. shortest path to F:

A → C → F

Shortest path to G:

A → C → E → G

	Vertex	Shortest Distance from Source	Previous
	A	0	null
	B	3	A
→	C	4	A
	D	9	E
→	E	6	C
→	F	7	C
→	G	10	E

Dijkstra's Algorithm

Dijkstras(G, S)

Source node = S

Initialize $distance, previous$
$distance(S) = 0$
$distance(V) = \infty$

Represents the set of all vertices other than Source (S)

$Q = \min priority\ queue$
$Q.add(V)$

$while\ Q\ !empty$

$u = Q.pop()$

$for\ each\ v\ in\ u.adjacent$
$d = distance(u) + e_{u,v}$

$if\ d < distance(v)$
$distance(v) = d$
$previous(v) = u$

$return\ G$

Runtime:

Dijkstra's Algorithm

Analyzing Dijkstra's Algorithm - Proof of Correctness

Prove: For the set S with starting vertex s_0, for each $u \in S$, the path p_u is the shortest s_0 to u path.

- We stopped here!

Remaining Slides Moved to Next Lecture Set

Minimum Spanning Trees

Given connected, undirected graph $G = (V, E)$, a spanning tree is a subgraph that is an undirected tree and contains all vertices in G.

In a weighted graph, the weight of the subgraph is the sum of edges in the subgraph.

A **minimum spanning tree** (MST) is a spanning tree with minimum weight.

Minimum Spanning Trees

Example: Consider the following graph: