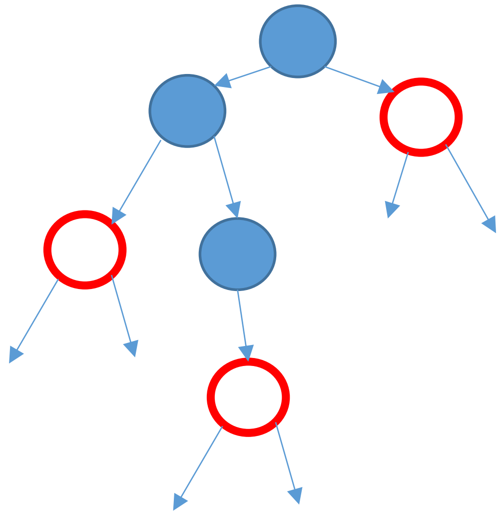
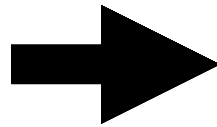
CS440/ECE448 Lecture 3: Search Order

Slides by Mark Hasegawa-Johnson, 1/2020

Including some slides written by Svetlana Lazebnik, 9/2016

CC-BY 4.0: You are free to: copy and redistribute the material in any medium or format, remix, transform, and build upon the material for any purpose, even commercially, if you give appropriate credit.





Outline

- Uniform cost search (UCS) = slightly different implementation of Dijkstra’s Algorithm
- Breadth-first search (BFS) = special case of UCS
- Depth-first search (DFS)

Dijkstra's Shortest Path Algorithm

- Initialize:
  - $d_{nl} =$ distance from n to l
  - $V_n = \infty$ for all vertices n
  - Unvisited = $\{all\ nodes\ but\ start\}$
  - k = Start Node
- While Goal ∈ Unvisited
  - For n ∈ Neighbor(k)
    - $V_n = \min(V_n, V_k + d_{kn})$
  - k ← $\underset{l \in Unvisited}{\text{argmin}}\ V_l$

By Subh83 - Own work, CC BY 3.0,
https://commons.wikimedia.org/w/index.php?curid=14916867

Uniform Cost Search

- Initialize:
  - $d_{nl} =$ distance from n to l
  - **$V_k = 0$ for start_node k, only**
  - **Frontier = {}**
  - k = Start Node
- While Goal $\neq k$
  - For n ∈ Neighbor(k)
    - **Frontier** $\leftarrow n: \min(V_n, V_k + d_{kn})$
  - k $\leftarrow \underset{l \in \textit{Frontier}}{\operatorname{argmin}} V_l$

By Subh83 - Own work, CC BY 3.0,
https://commons.wikimedia.org/w/index.php?curid=14916867

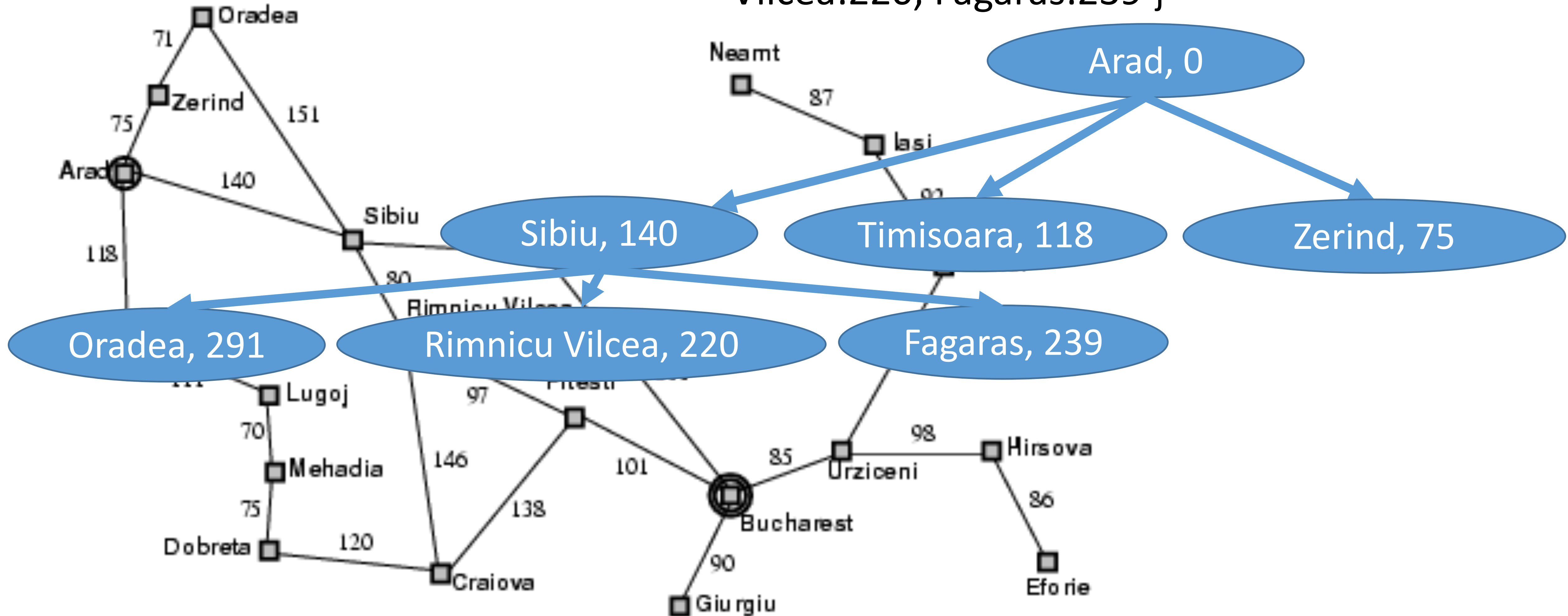
Dijkstra’s algorithm vs. Uniform Cost Search

- They evaluate the same nodes k, in exactly the same order.
- They give the same (minimum-cost) path as a result.
- The only difference:
    - Dijkstra’s algorithm keeps track of V_n for all nodes in the search space
    - UCS keeps track of V_n only for nodes you’ve explored

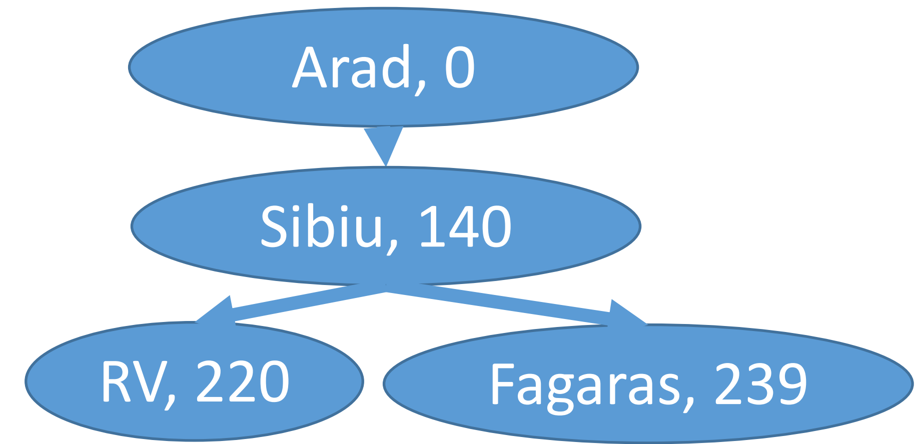
Example: Romania

Frontier: { Zerind:75, Timisoara:118, Oradea:291, Rimnicu Vilcea:220, Fagaras:239 }

Explored: { Arad:0, Sibiu:140, Zerind:75, Timisoara:118, Oradea:291, Rimnicu Vilcea:220, Fagaras:239 }



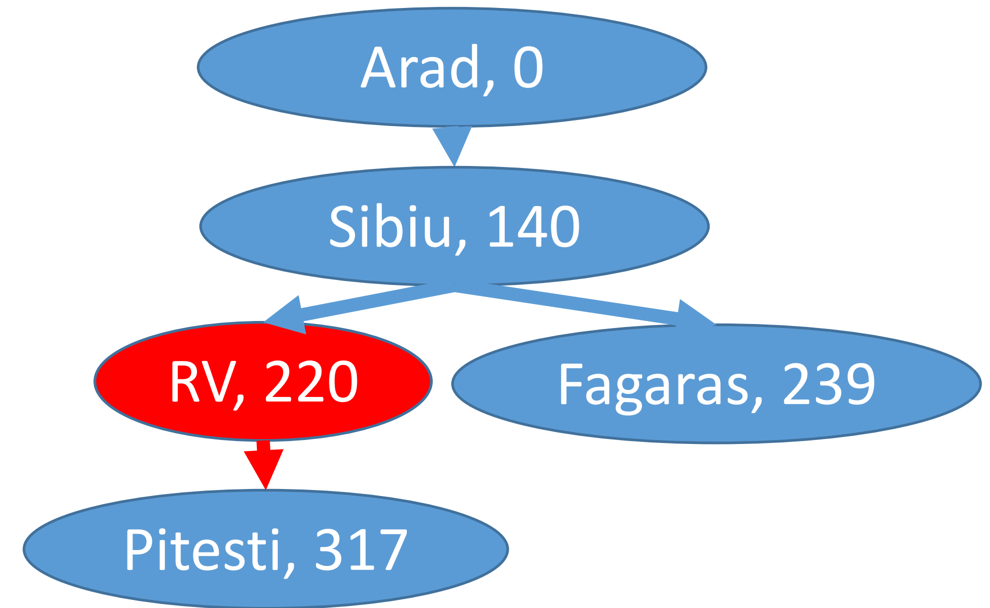
Why it Works



- $d_{nl} \geq 0$ means that every node on the best path to the Goal, G, has a cost, V_n, less than or equal the cost of the goal, $V_n \leq V_G$

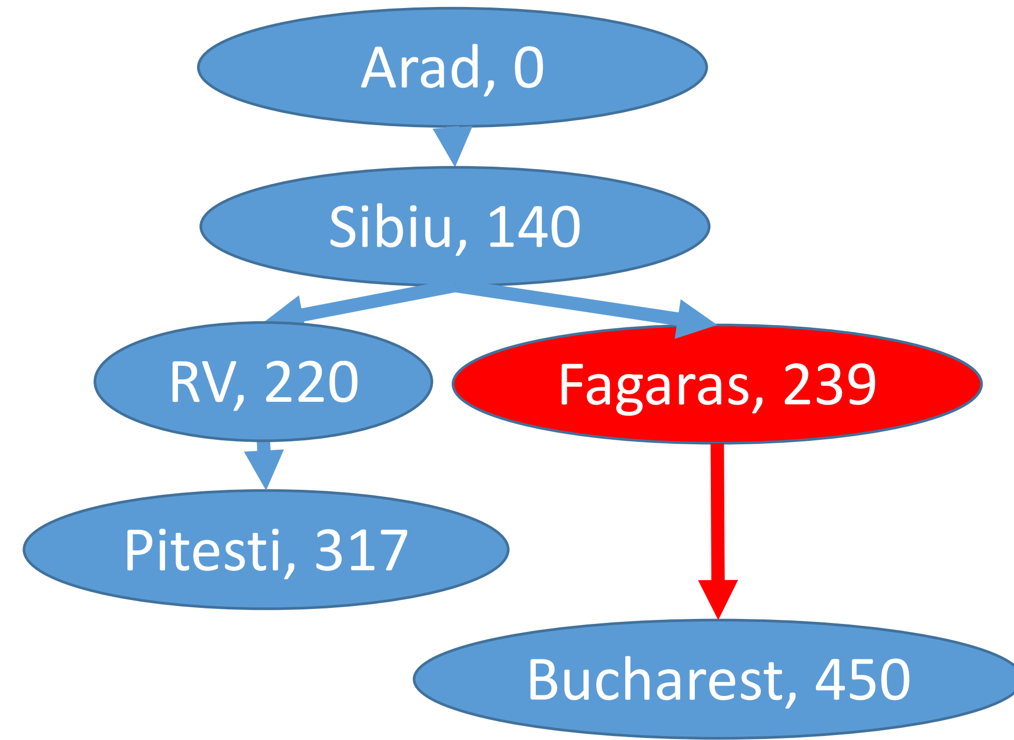
Why it Works

- k ← $\underset{l \in Frontier}{argmin}\ V_l$ means that the lowest-cost nodes are expanded first



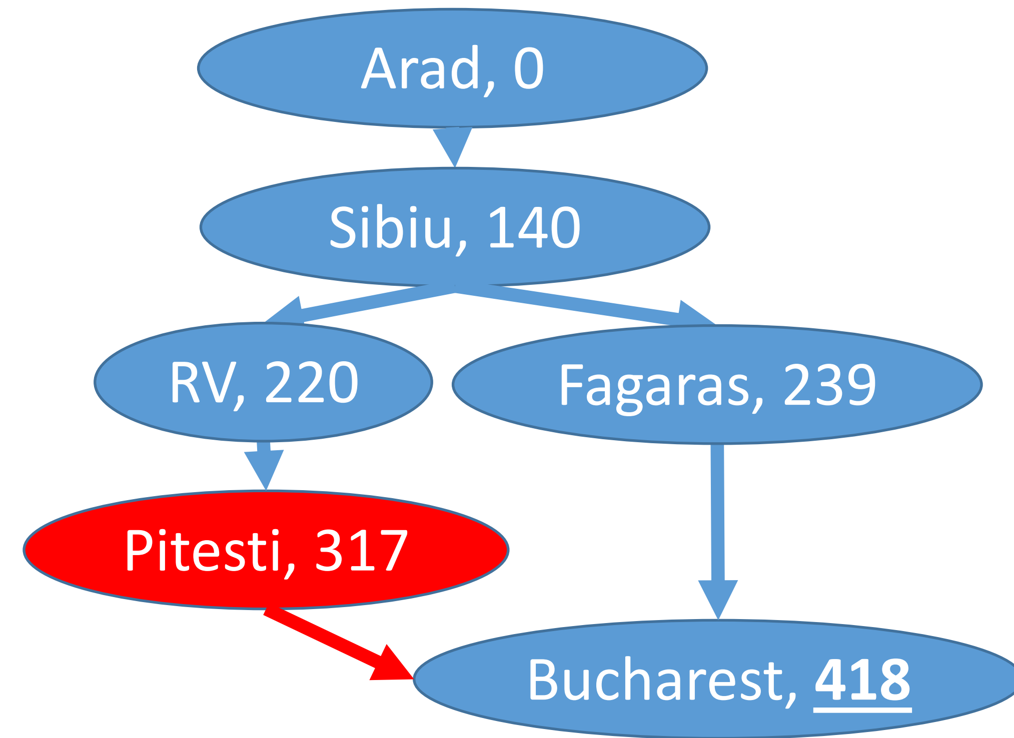
Why it Works

- We don’t end when Goal is placed on the **Frontier**, we only end when Goal is **expanded**



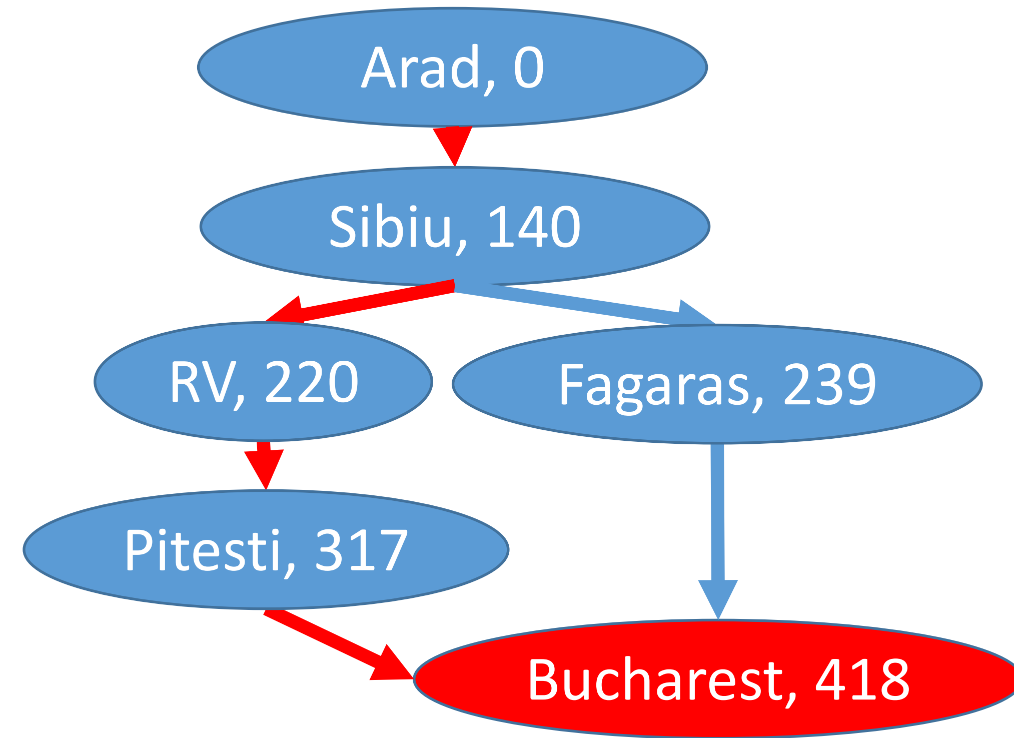
Why it Works

- k ← $\underset{l \in Frontier}{argmin}\ V_l$ means that every predecessor with a cost less than V_G is expanded before the goal is expanded



Why it Works

- Therefore we always find the shortest path



Computational Considerations

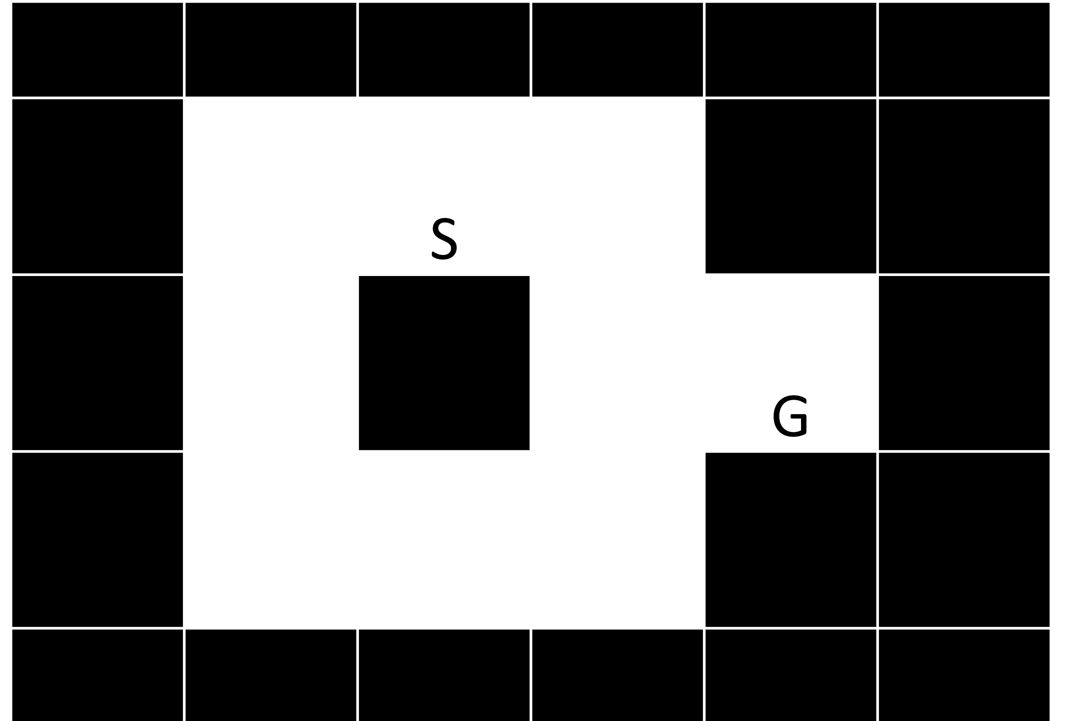
- Suppose there are N world states
- k ← $\underset{l \in Frontier}{\text{argmin}} \; V_l$
  - Naïve implementation: requires you to sort through the whole frontier, to find the smallest. Complexity: O{N} per search step!!
  - Better implementation: keep the frontier sorted, as a **priority queue**. Then complexity is O{logN} to insert a node into the frontier, and O{1} to retrieve the minimum.
- Frontier ← $n\text{: } \min(V_n, V_k + d_{kn})$
  - "Explored list" is a **hash table** (python: a dict), so that, given a world state n, you can immediately tell (O{1}) whether or not that state has been explored.
  - Each state in the "Explored list" has a pointer to corresponding state in the Frontier, if that state is still in the frontier (O{1}).

Outline

- Uniform cost search (UCS) = slightly different implementation of Dijkstra's Algorithm
- Breadth-first search (BFS) = special case of UCS
- Depth-first search (DFS)

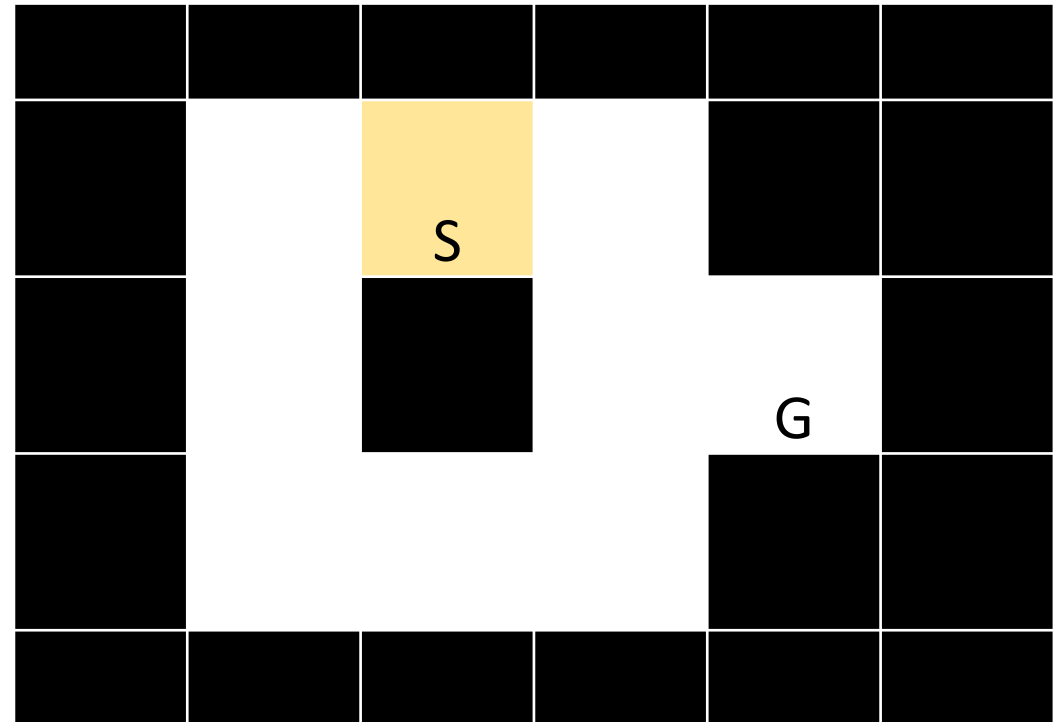
Breadth-first search (BFS) = special case of UCS

- … when every step has exactly the same cost, $d_{nl} = 1$
- Example: solving a maze



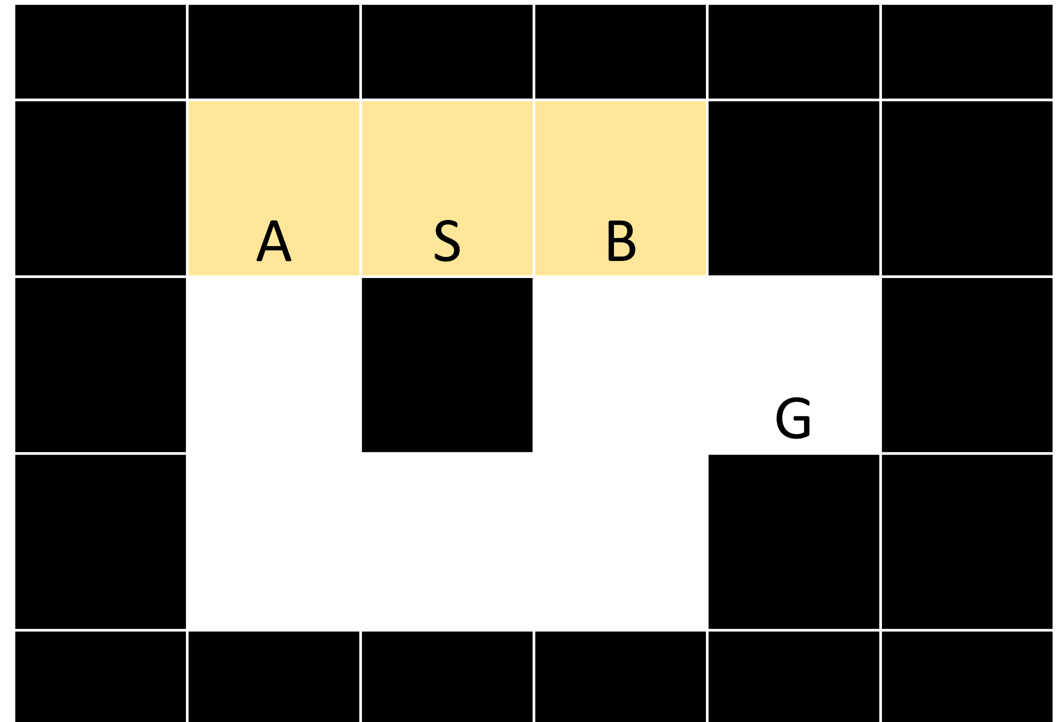
BFS Computational Savings

- Frontier doesn't have to be a priority queue
- It can just be a regular first-in, first-out (FIFO) queue
- Example: Frontier = {S}



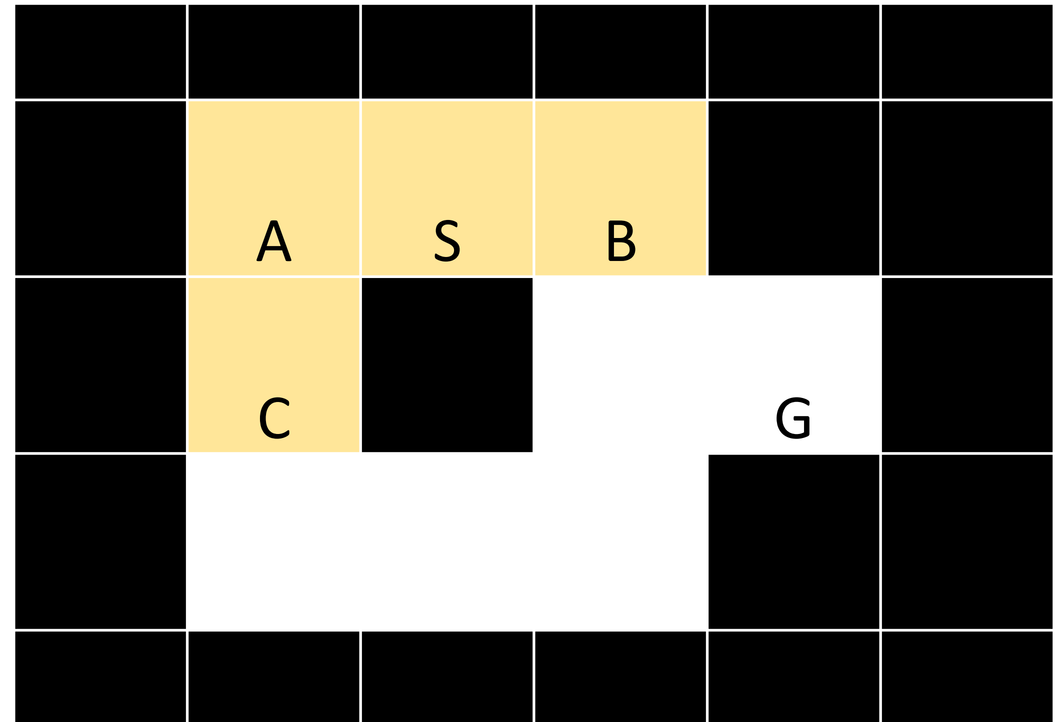
BFS Computational Savings

- Frontier doesn't have to be a priority queue
- It can just be a regular first-in, first-out (FIFO) queue
- Example: Frontier = {A,B}
- Pop A from the queue, expand it



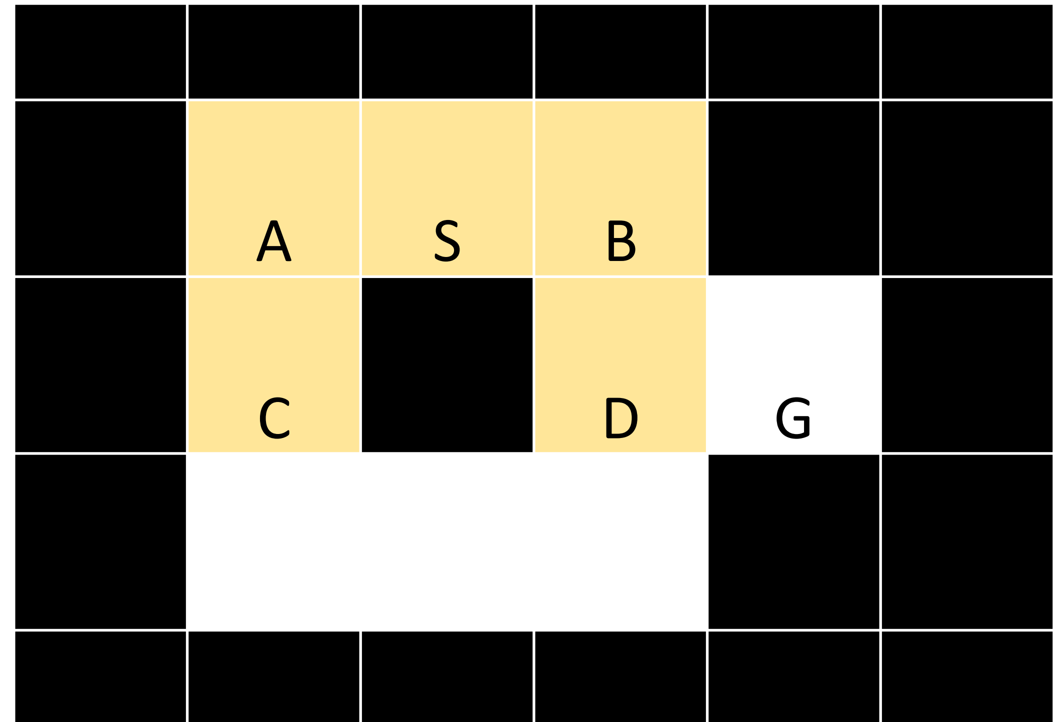
BFS Computational Savings

- Frontier doesn't have to be a priority queue
- It can just be a regular first-in, first-out (FIFO) queue
- Example: Frontier = {B,C}
- Pop B from the queue, expand it



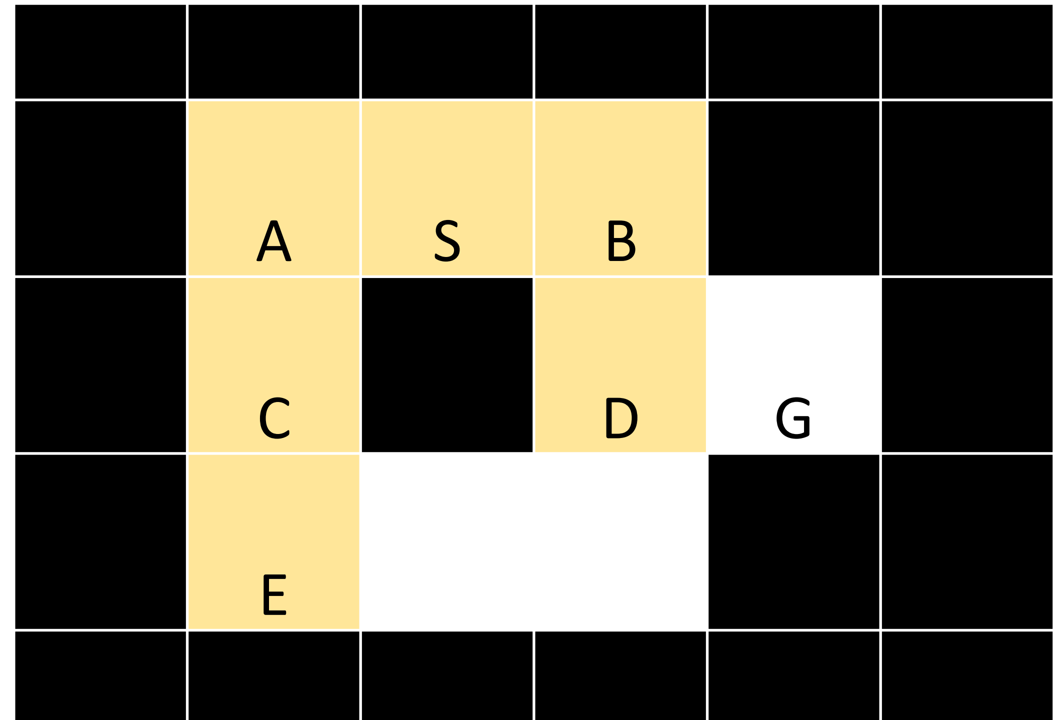
BFS Computational Savings

- Frontier doesn't have to be a priority queue
- It can just be a regular first-in, first-out (FIFO) queue
- Example: Frontier = {C,D}
- Pop C from the queue, expand it



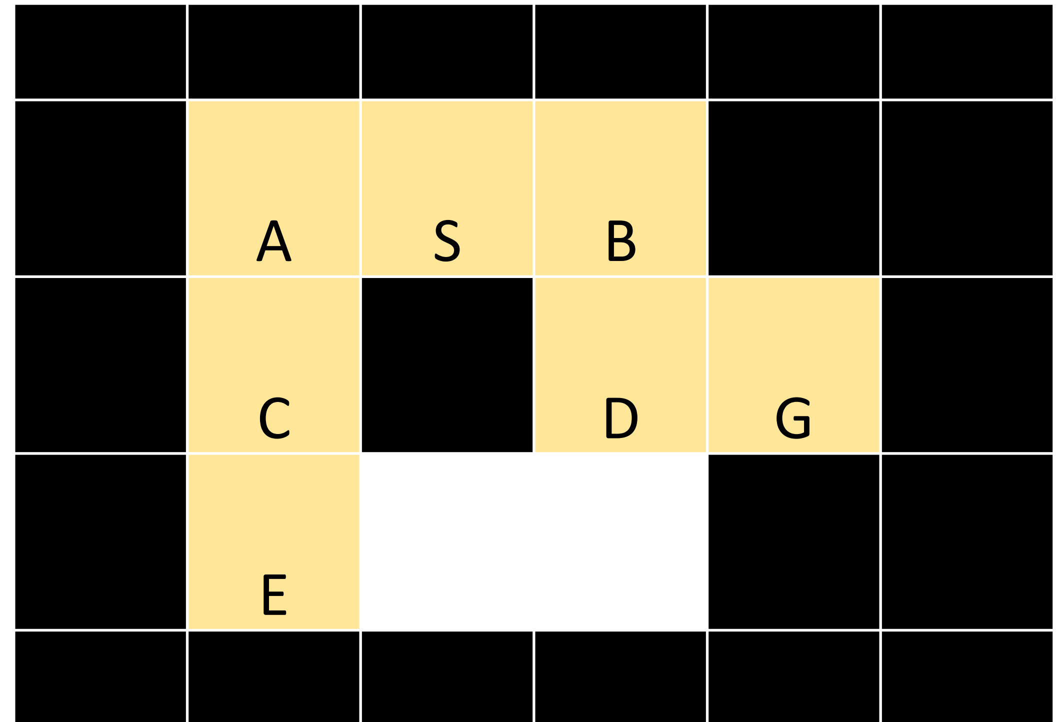
BFS Computational Savings

- Frontier doesn't have to be a priority queue
- It can just be a regular first-in, first-out (FIFO) queue
- Example: Frontier = {D,E}
- Pop D from the queue, expand it



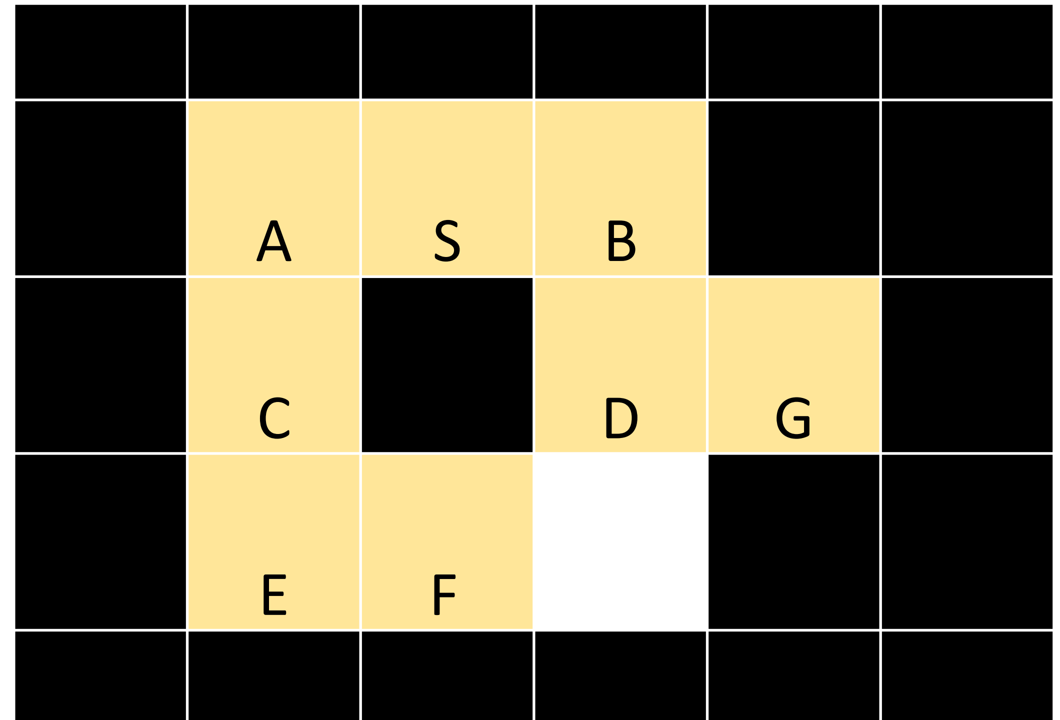
BFS Computational Savings

- Frontier doesn't have to be a priority queue
- It can just be a regular first-in, first-out (FIFO) queue
- Example: Frontier = {E,G}
- Pop E from the queue, expand it



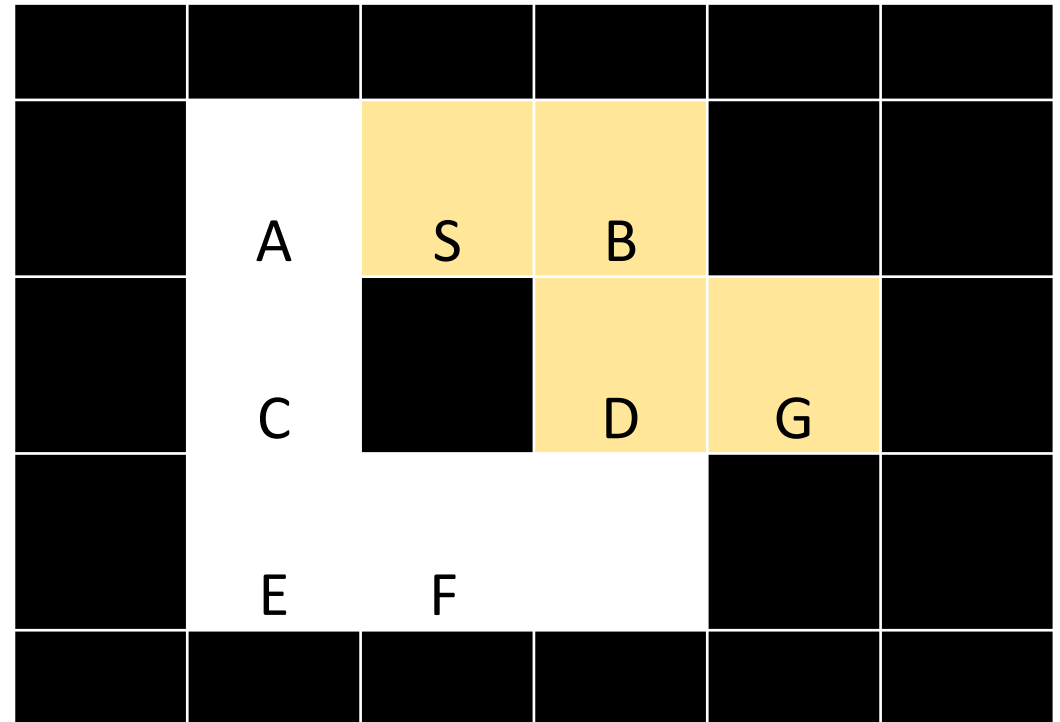
BFS Computational Savings

- Frontier doesn't have to be a priority queue
- It can just be a regular first-in, first-out (FIFO) queue
- Example: Frontier = {G,F}
- Pop G from the queue, expand it



BFS Computational Savings

- Frontier doesn't have to be a priority queue
- It can just be a regular first-in, first-out (FIFO) queue
- Example: G expanded, we learn that it's the goal, we found the best path



Analysis of search strategies

- Strategies are evaluated along the following criteria:
  - **Completeness:** does it always find a solution if one exists?
  - **Optimality:** does it always find a least-cost solution?
  - **Time complexity:** number of nodes generated
  - **Space complexity:** maximum number of nodes in memory
- Time and space complexity are measured in terms of
  - ***b****:* maximum branching factor of the search tree
  - ***d****:* depth of the optimal solution
  - ***m***: maximum length of any path in the state space (may be infinite)

Properties of breadth-first search

- **Complete?**
  Yes (if branching factor b is finite).
  Even w/o explored-set checking, it still works!
- **Optimal?**
  Yes – if cost = 1 per step (uniform cost search will fix this)
- **Time?**
  Number of nodes in a b-ary tree of depth d: $O(b^d)$
  (d is the depth of the optimal solution)
- **Space?**
  $O(b^d)$

- Space is the bigger problem (more than time)

Properties of uniform-cost search

- **Complete?**
  Yes (if branching factor b is finite).
  Even w/o explored-set checking, it still works!
- **Optimal?**
  Yes – even if cost $\neq$ 1 per step
- **Time?**
  Number of nodes in a b-ary tree of depth d: $O(b^d)$
  (d is the depth of the optimal solution)
- **Space?**
  $O(b^d)$

- Space is the bigger problem (more than time)

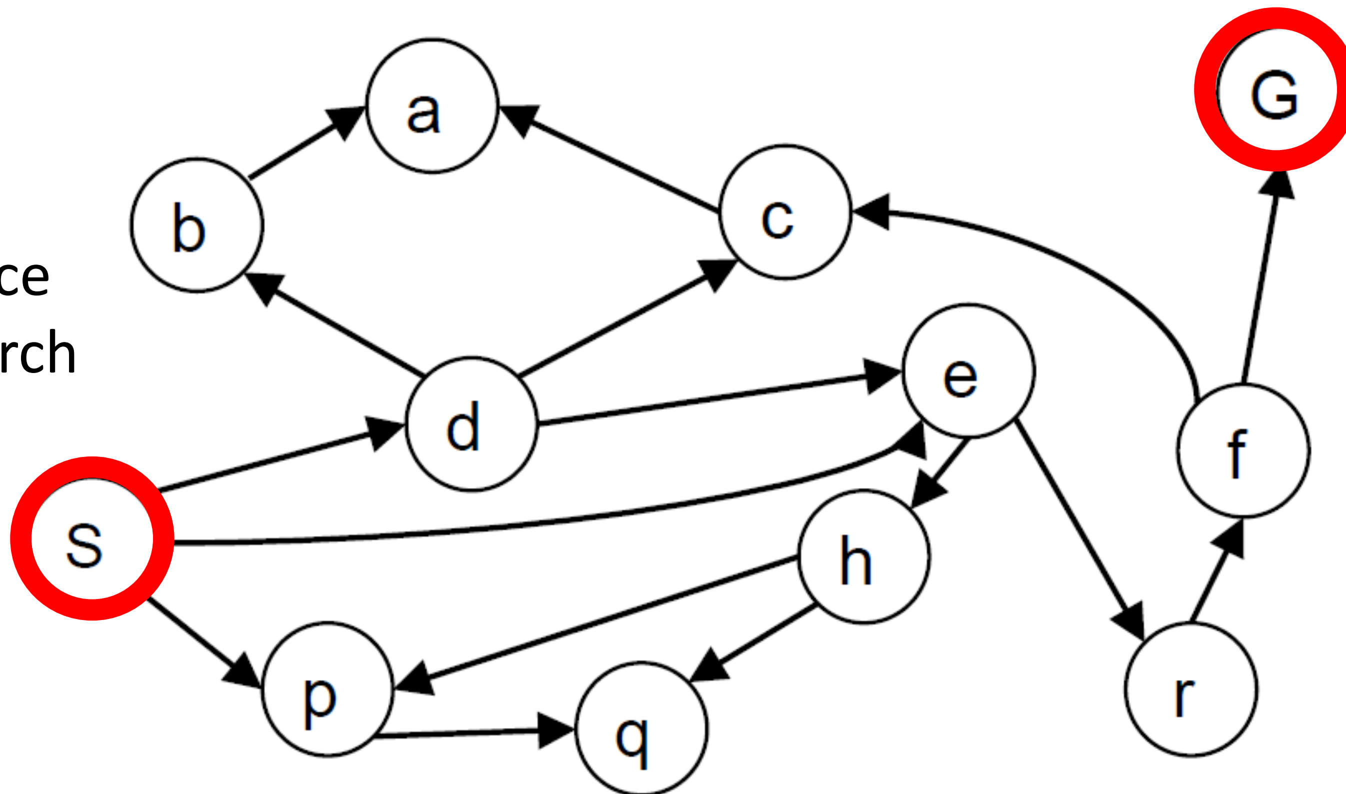
Outline

- Uniform cost search (UCS) = slightly different implementation of Dijkstra’s Algorithm
- Breadth-first search (BFS) = special case of UCS
- Depth-first search (DFS)

Depth-first search

- Expand ***deepest*** unexpanded node (BFS: ***shallowest***)
- Implementation: Frontier is a ***last-in-first-out (LIFO)*** stack (BFS:FIFO)

Example state space graph for a tiny search problem



Example from P. Abbeel and D. Klein

Depth-first search

Frontier:

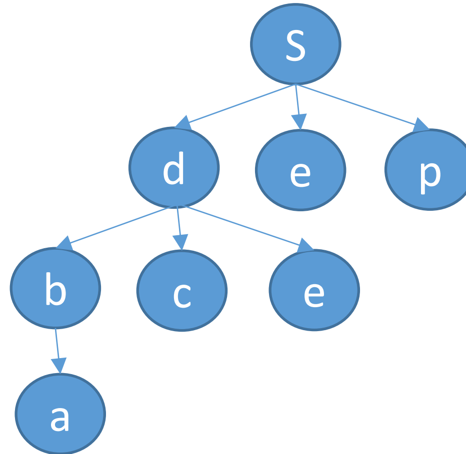
Step 0: {S}

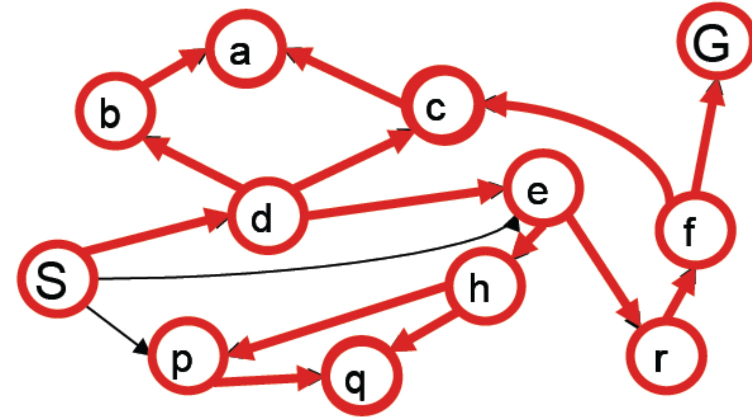
Step 1: {d,e,p}

Step 2: {b,c,e,p} – FIFO

Step 3: {a,c,e,p}

Step 4: {c,e,p}

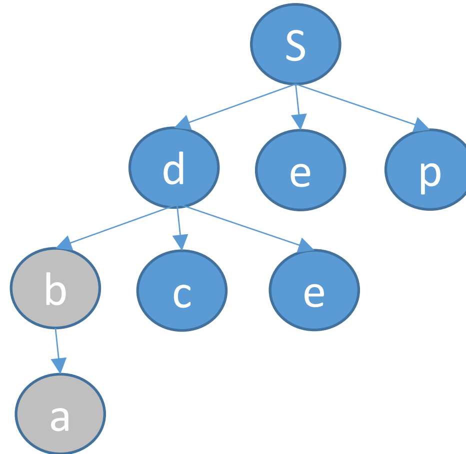


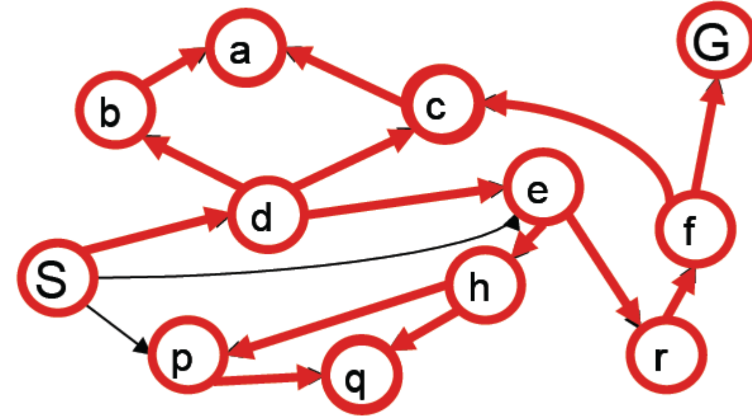


Example from P. Abbeel and D. Klein

The reason DFS is useful: Space

When we know that Sdba is a dead end, we can remove it from the tree!





Example from P. Abbeel and D. Klein

Breadth-first search

Computational complexity:

(s,

d,e,p,

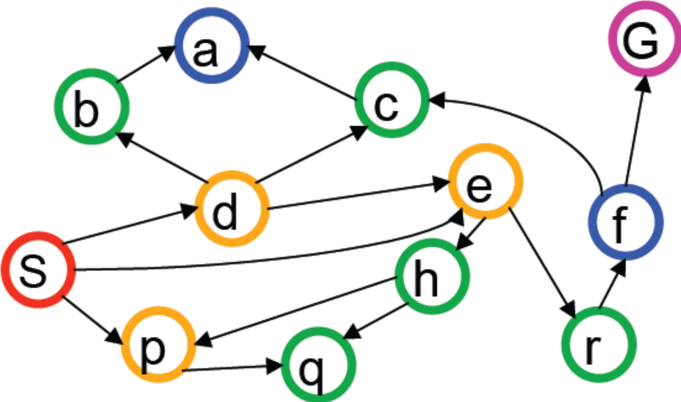
b,c,e,h,r,q,

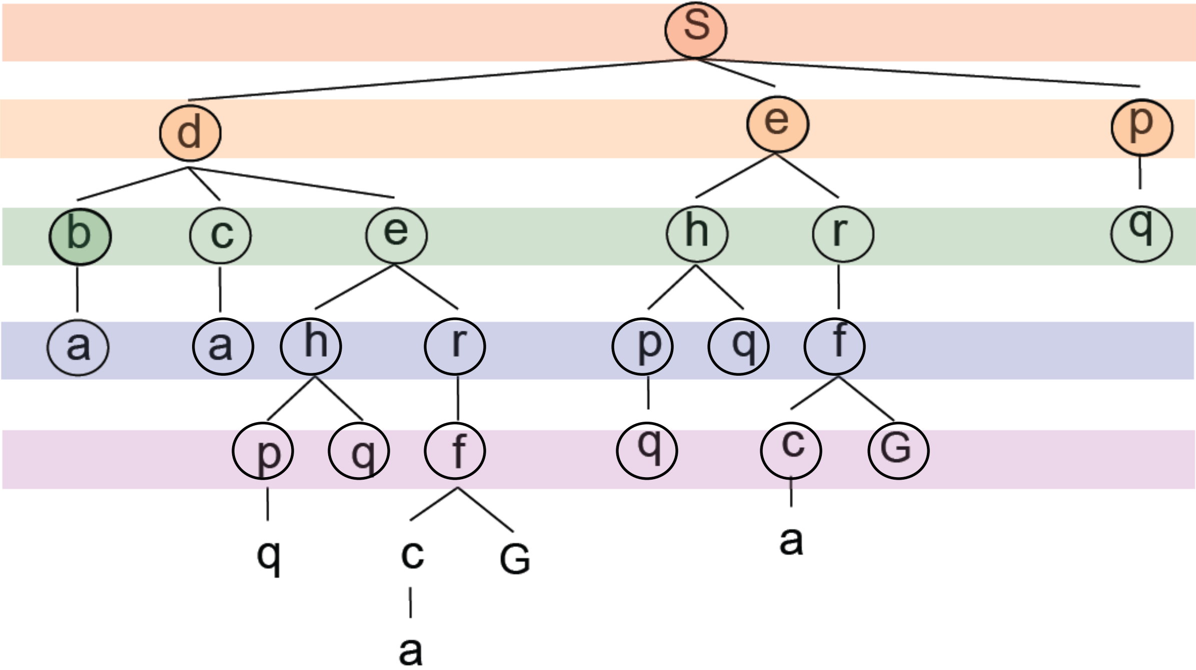
a,a,h,r,p,q,f,

p,q,f,q,c,G)

Space complexity:

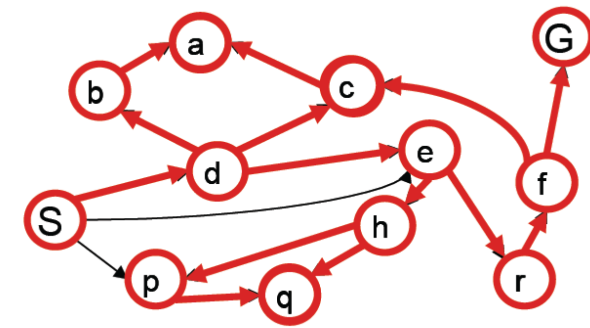
We have to store the whole tree!



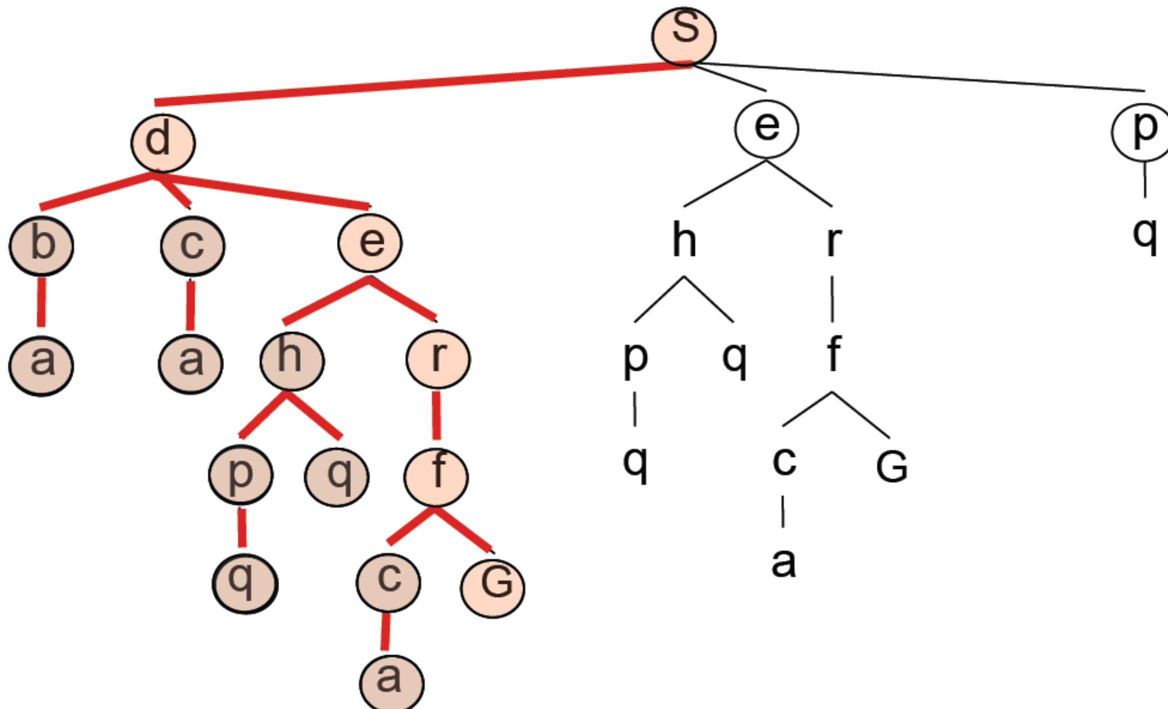


Example from P. Abbeel and D. Klein

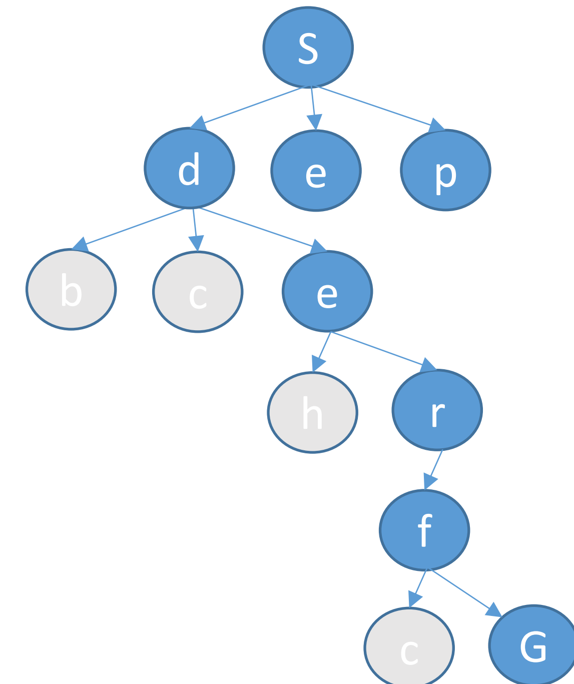
Depth-first search



Computational complexity: here are the nodes we expanded



Space complexity: here's the part of the tree that we still have in memory



Example from P. Abbeel and D. Klein

Analysis of search strategies

- Strategies are evaluated along the following criteria:
  - **Completeness:** does it always find a solution if one exists?
  - **Optimality:** does it always find a least-cost solution?
  - **Time complexity:** number of nodes generated
  - **Space complexity:** maximum number of nodes in memory
- Time and space complexity are measured in terms of
  - ***b**:* maximum branching factor of the search tree
  - ***d**:* depth of the optimal solution
  - ***m***: maximum length of any path in the state space (may be infinite)

Properties of depth-first search

- **Complete? (always finds a solution if one exists?)**
  Fails in infinite-depth spaces
  Fails if there are loops (unless you keep an "Explored Set")
- **Optimal? (always finds an optimal solution?)**
  No – returns the first solution it finds
- **Time? (how long does it take, in terms of b, d, m?)**
  $O(b^m)$ (remember BFS was $O(b^d)$)
  Terrible if m is much larger than d
- **Space? (how much storage space?)**
  $O(bm)$, i.e., linear space!
  The frontier doesn't need to keep track of failed paths, only the currently active path

Comparison of Search Strategies

Algorithm	Complete?	Optimal?	Time complexity	Space complexity	Implement the Frontier as a…
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$	Queue
DFS	No	No	$O(b^m)$	$O(bm)$	Stack
UCS	Yes	Yes	Number of nodes w/ $V_n \leq V_G$	Number of nodes w/ $V_n \leq V_G$	Priority Queue sorted by V_n