Lecture 18: Search

Mark Hasegawa-Johnson

Johannes Hevelius observing with one of his telescopes (1647).

Outline

- Search Problems: start, goal, neighborhood
- Depth-first search (DFS): completeness, admissibility, & optimality
- Breadth-first search (BFS)
- Uniform-cost search (UCS)

Search problems

Defined by:

- A discrete (possibly infinite) set of states or nodes, $n \in \mathcal{N}$
  - The agent must start in a "start state" s.
  - The agent must reach any "goal state" $t \in \mathcal{T}$, where $\mathcal{T} \subset \mathcal{N}$.
- A set of transitions
  - $\Gamma(n)$ = the set of states that are neighbors of n.
  - $h(m, n)$ = cost of the shortest path from m to n, $h(m, n) > 0$.

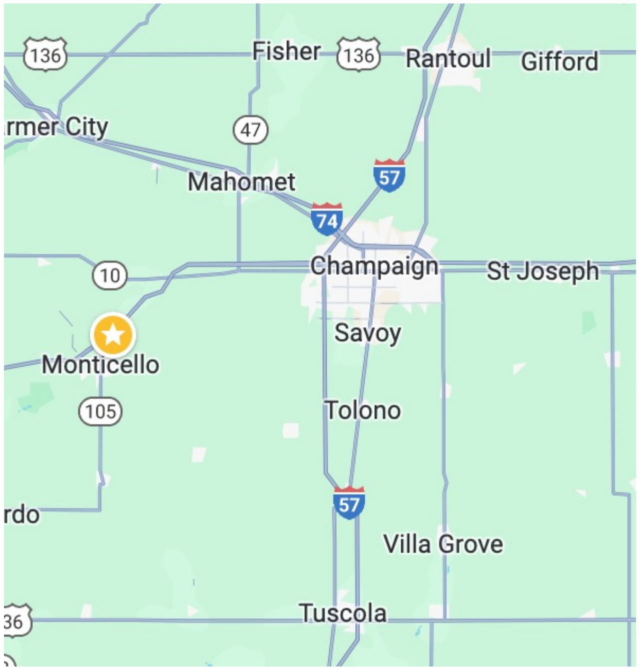
Example: Road Trip

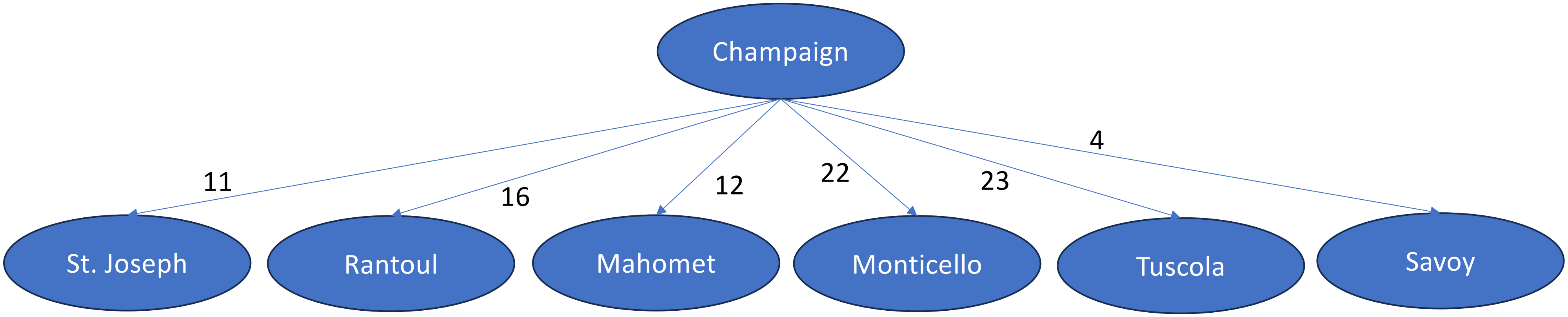
We're in Champaign-Urbana. We want to plan a road trip to see New York and Washington, D.C.

- State definition: physical location $\times$ have we visited New York yet? $\times$ have we visited DC yet?
  - Initial state: Urbana, no, no
  - Final state: Anywhere, yes, yes
- $\Gamma(n)$ $=$ set of cities reachable from current location
  - If we reach NY, set NY=True
  - If we reach DC, set DC=True
- $h(m,n)$ $=$ distance, in miles, from m.loc to n.loc

Neighborhood

- The neighborhood function, $\Gamma(n)$, finds the neighbors of a node
- It also gives you the distance $h(n, m)$ from n to each neighbor





Solution strategies

- Random walk: Just start driving
  - Advantages: No thinking required
  - Disadvantages: We might never get there
- Planned walk: Explore every possible path, and choose the shortest
  - Advantages: Reach goal, Spend the least possible amount of gas
  - Disadvantages: Lots of computation

Search algorithms compute a path to the goal (possibly the shortest) by describing many partial paths (description = list of states on each path).

Outline

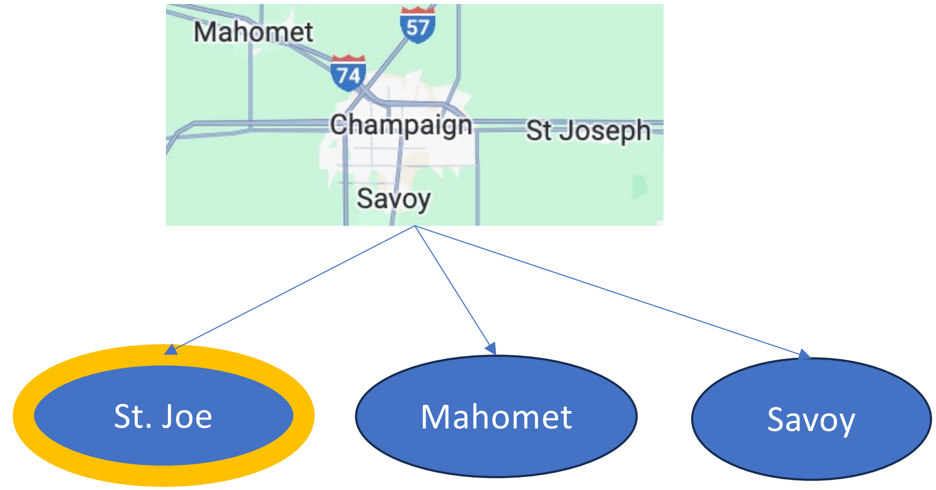
- Search Problems: start, goal, neighborhood
- Depth-first search (DFS): completeness, admissibility, & optimality
- Breadth-first search (BFS)
- Uniform-cost search (UCS)

Depth-first search

- Depth-first search is sort of like a random walk, but in software, not in real life
- Advantage: if the random walk doesn't reach the goal, then we have only spent electricity, not gas

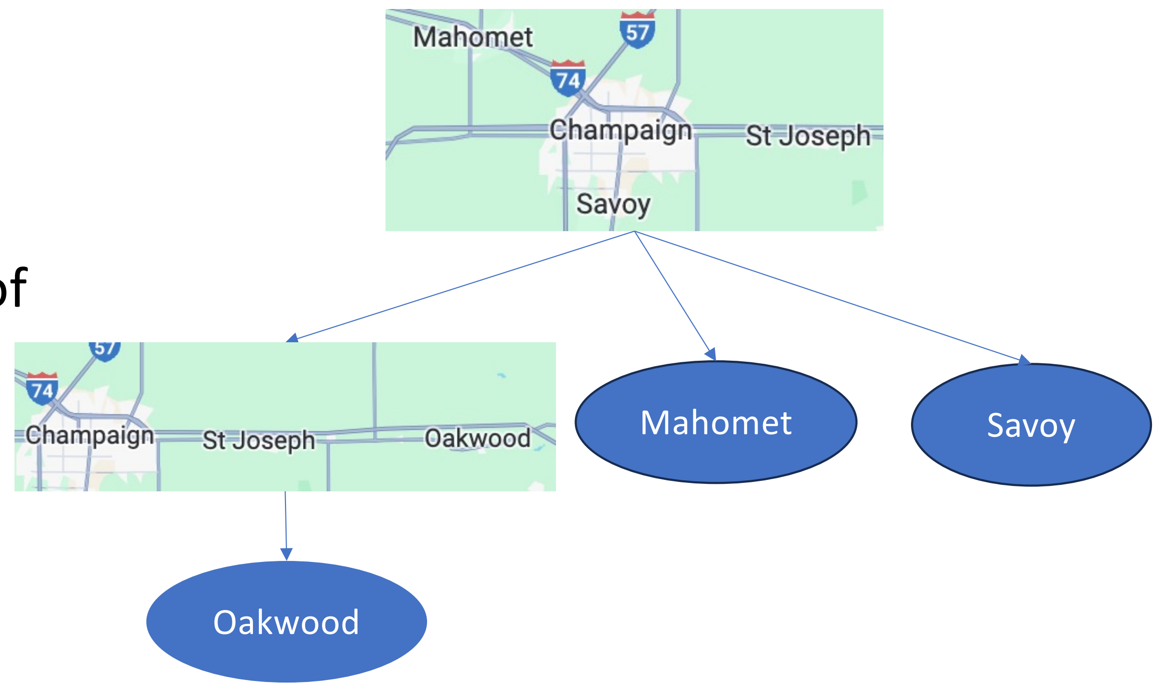
Depth-first search

- Choose, at random, n_1 =one of the neighbors of s



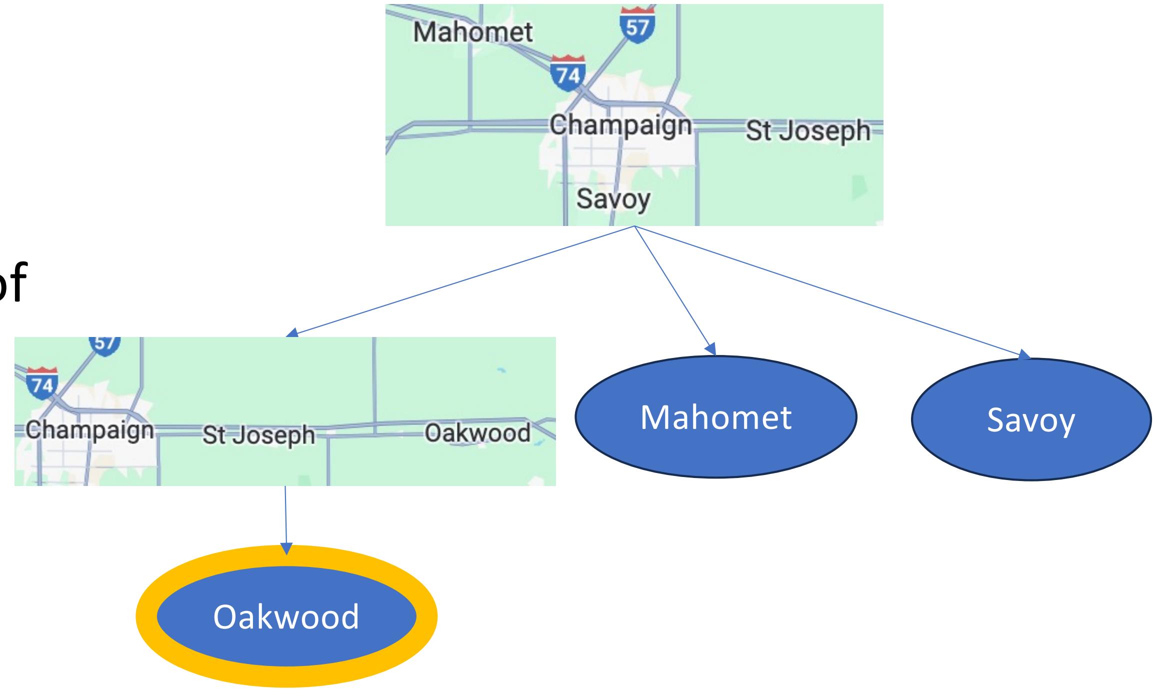
Depth-first search

- Choose, at random, n_1 =one of the neighbors of s
- EXPAND it.
  - Definition: Find out what its neighbors are



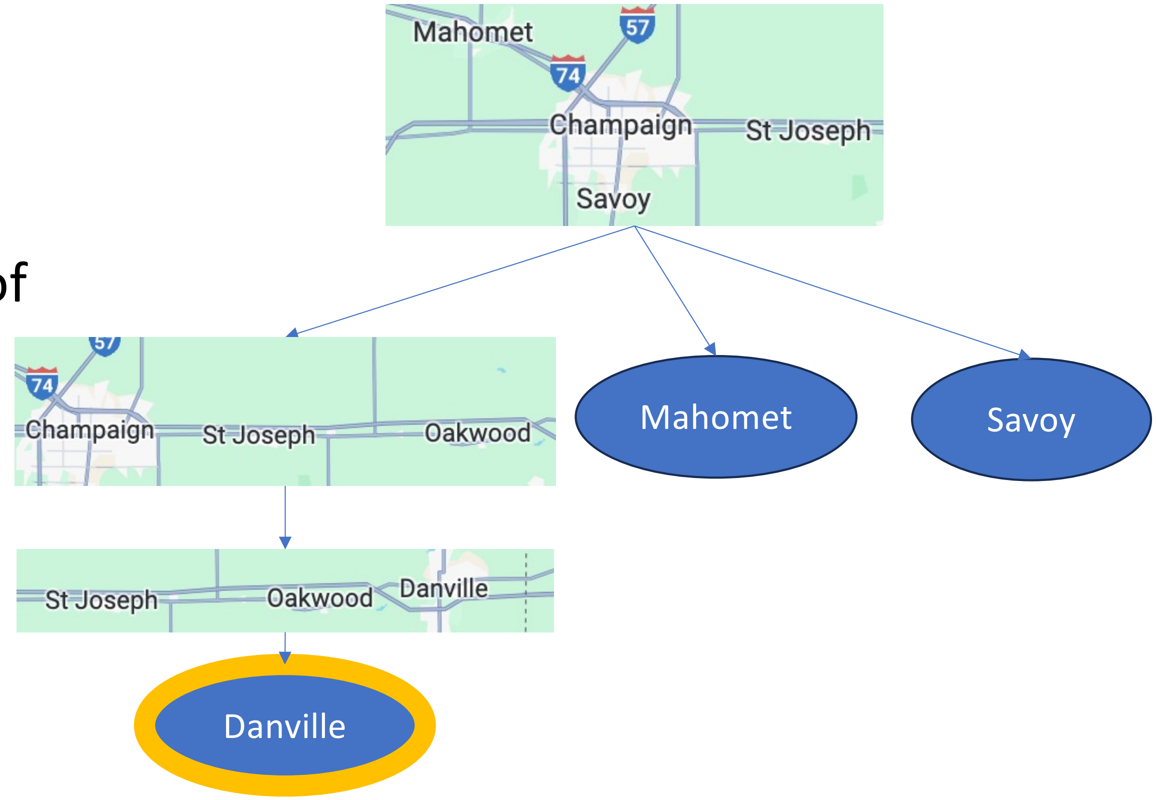
Depth-first search

- Choose, at random, n_1 =one of the neighbors of s
- EXPAND it.
  - Definition: Find out what its neighbors are
- Choose, at random, n_2 = one of the neighbors of n_1.
  - Make sure not to choose a state you've already explored



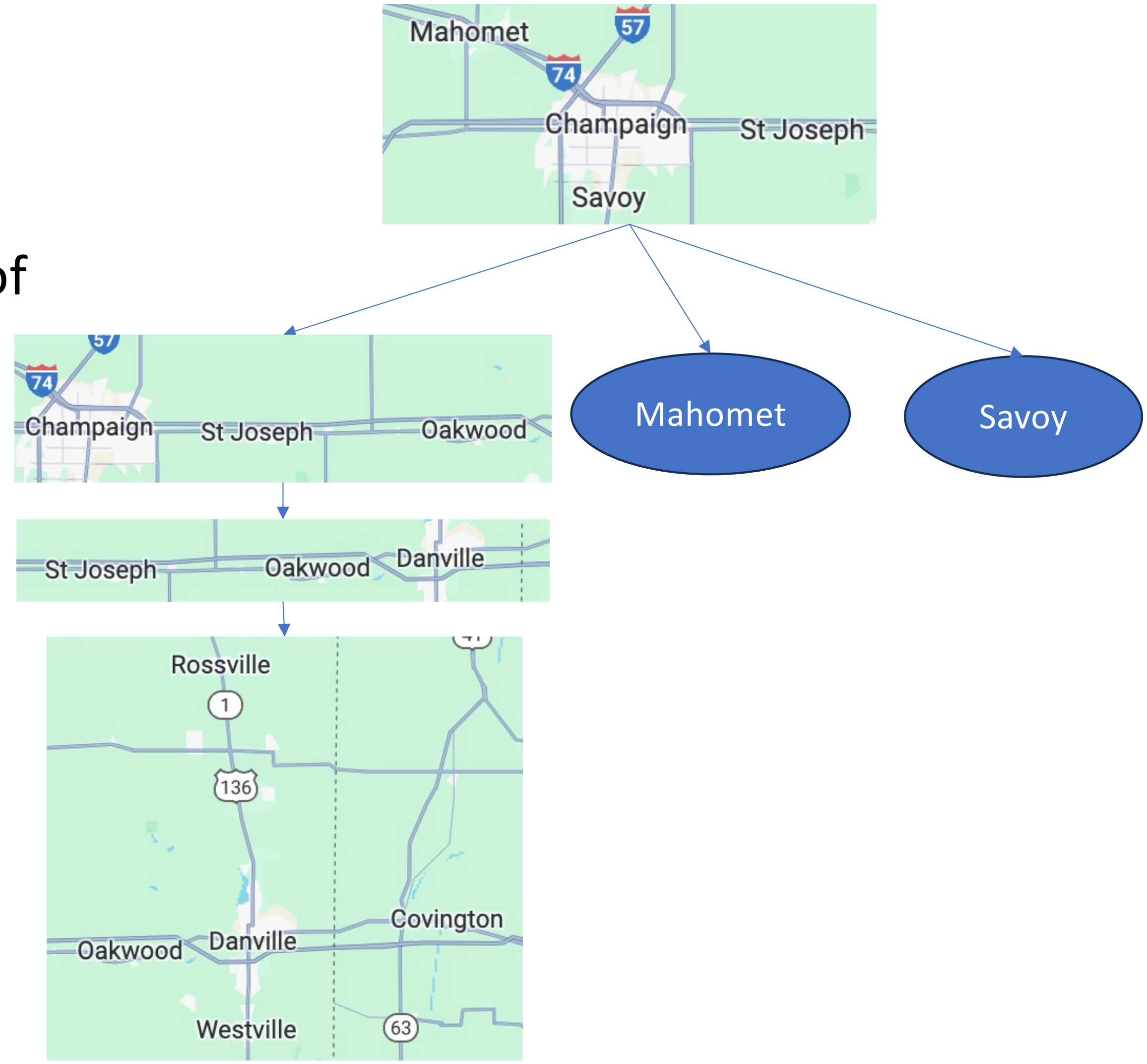
Depth-first search

- Choose, at random, n_1 =one of the neighbors of s
- EXPAND it.
  - Definition: Find out what its neighbors are
- Choose, at random, n_2 = one of the neighbors of n_1.
  - Make sure not to choose a state you've already explored
- Repeat



Depth-first search

- Choose, at random, n_1 =one of the neighbors of s
- EXPAND it.
  - Definition: Find out what its neighbors are
- Choose, at random, n_2 = one of the neighbors of n_1.
  - Make sure not to choose a state you've already explored
- Repeat
- Repeat



Problems with depth-first search

- It might run forever, without ever finding a path to the goal
- If it finds a path to the goal, there's no guarantee it finds the shortest path
- Even if it finds the shortest path, it might require an unreasonable amount of computation

Desirable properties of a search algorithm

- **<u>Complete</u>**: If there is a finite-length path to the goal, the algorithm finds it in a finite amount of time
- **<u>Admissible</u>**: If there is a path, it finds the shortest path
    - Shortest path = smallest path cost (e.g., miles traveled)
- **<u>Optimal</u>**: If there is a path, it uses the least possible amount of computation to find the path
    - Computation = number of states on which the neighborhood function, $\Gamma(n)$, must be evaluated.

Depth-first search (DFS) has none of these properties.

Outline

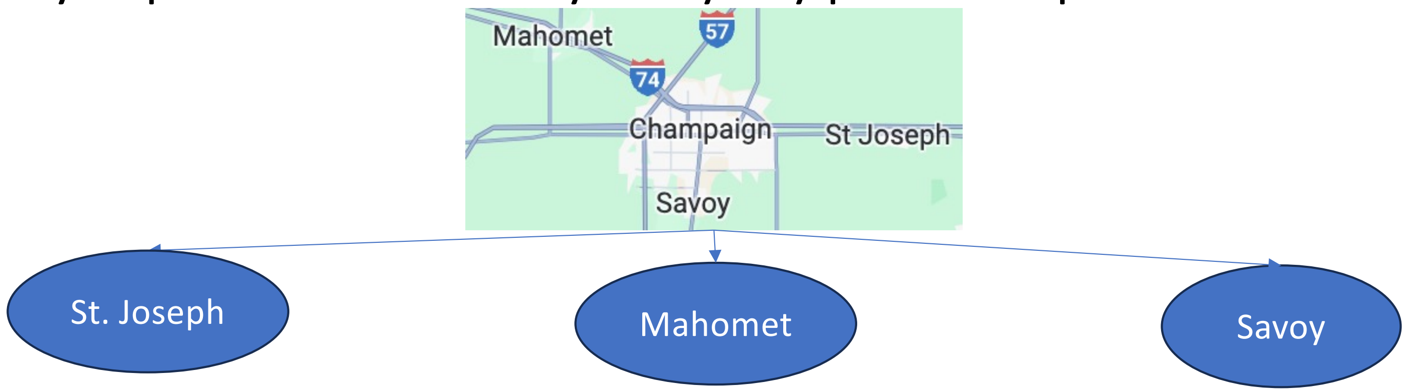
- Search Problems: start, goal, neighborhood
- Depth-first search (DFS): completeness, admissibility, & optimality
- Breadth-first search (BFS)
- Uniform-cost search (UCS)

Depth of a search

- Suppose that reaching our goal requires passing through d nodes
- We call d the ***depth*** of the path
- How can we guarantee that we find a path of depth d, if it exists?
- Answer: try every path of length d before we try any paths of length $d + 1$

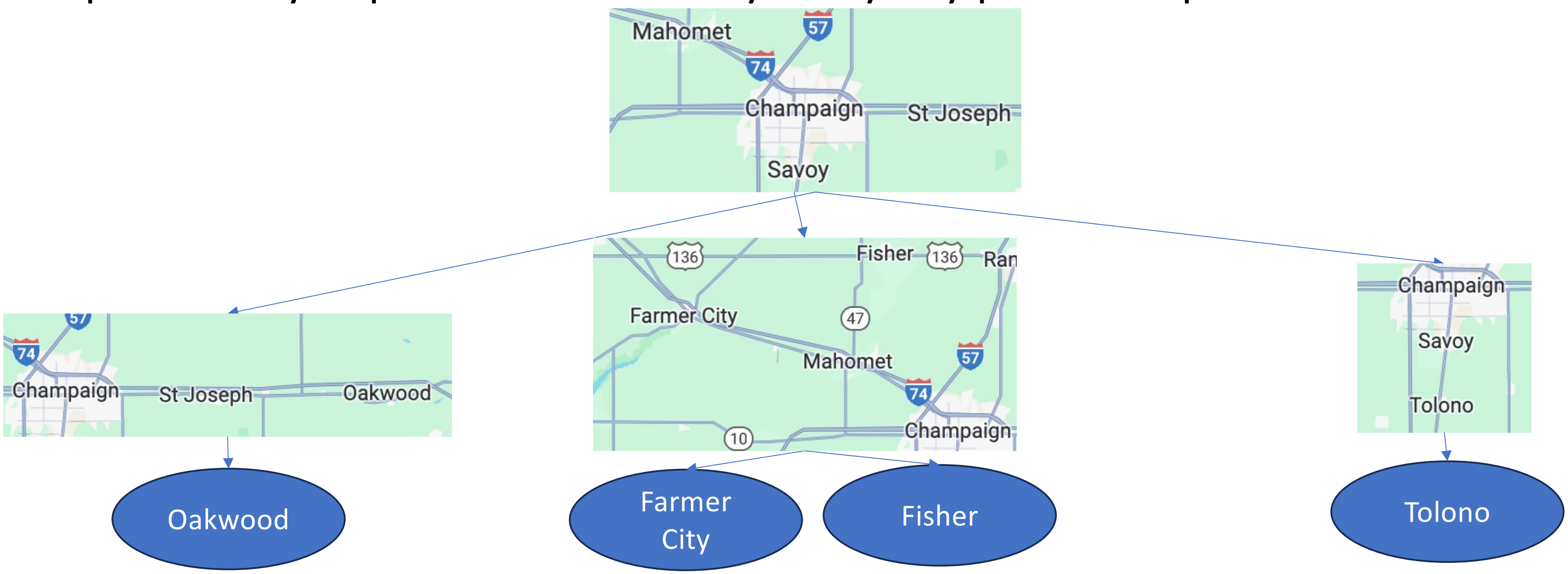
Breadth-first search

Expand every depth-0 node before you try any path of depth 1.



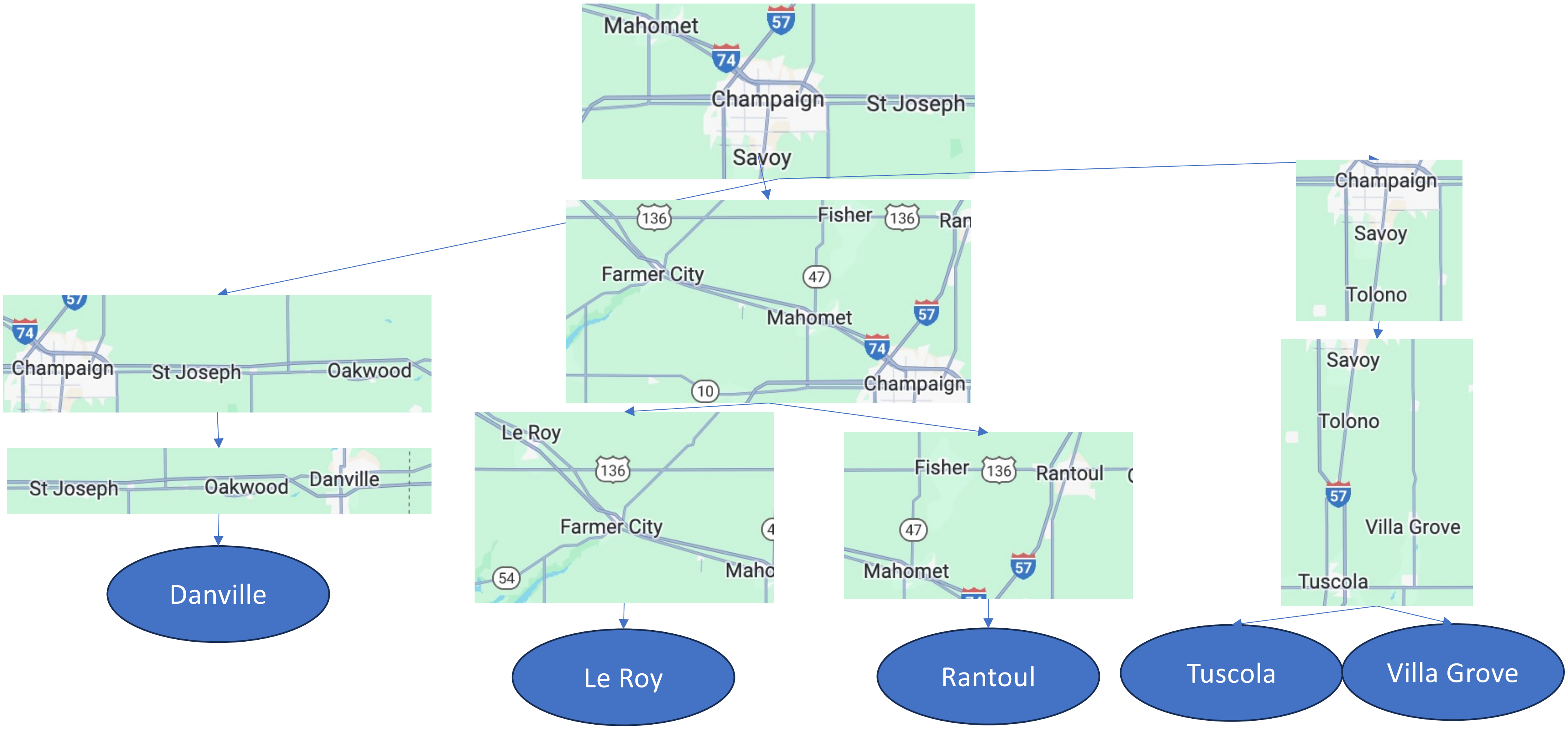
Breadth-first search

Expand every depth-1 node before you try any path of depth 2.



Breadth-first search

Expand every depth-2 node before you try any path of depth 3.



Analysis of breadth-first search

- **Complete?** Yes
  - If the goal can be reached in a path of depth d, BFS will find it at a depth of d
- **Admissible?** Only if all steps have the same cost
  - If each step has a cost of 1, then the best path has a cost of d, and BFS finds it
  - If different steps have different costs, then BFS may not find the shortest
- **Optimal?** No
  - There are other algorithms that require less computation

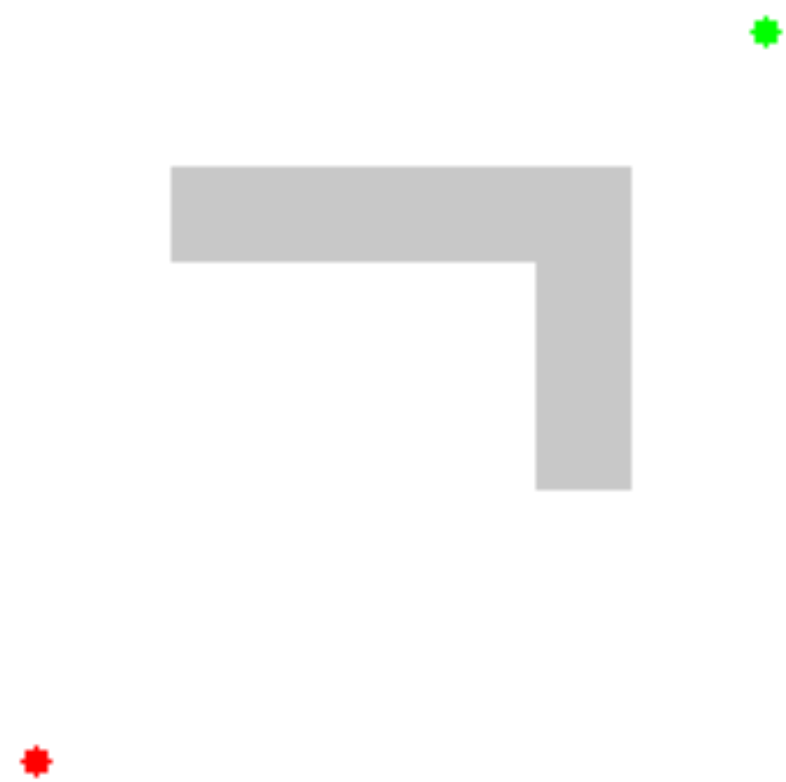
Computational complexity of BFS and DFS

- **Parameters**
  - $b =$ Branching factor (largest number of neighbors any node can have)
  - $d =$ Depth of the best path to goal
  - $m =$ Depth of the longest path to any state (may be infinite)
- **Time complexity**: (# evaluations of $\Gamma(n)$)
  - BFS: Time complexity = $\mathcal{O}\{b^d\}$
  - DFS: Time complexity = $\mathcal{O}\{b^m\}$
- **Space complexity**: (# nodes that must be stored during search)
  - BFS: Space complexity = $\mathcal{O}\{b^d\}$
  - DFS: Space complexity = $\mathcal{O}\{bm\}$

Completeness of BFS (animation)

- $b = 8$
- $d = 28$
- $m =$ not shown (infinite?)

- **<u>Time complexity</u>**:
  - BFS: Time complexity = $\mathcal{O}\{b^d\}$
  - DFS: Time complexity = $\mathcal{O}\{b^m\}$
- **<u>Space complexity</u>**:
  - BFS: Space complexity = $\mathcal{O}\{b^d\}$
  - DFS: Space complexity = $\mathcal{O}\{bm\}$

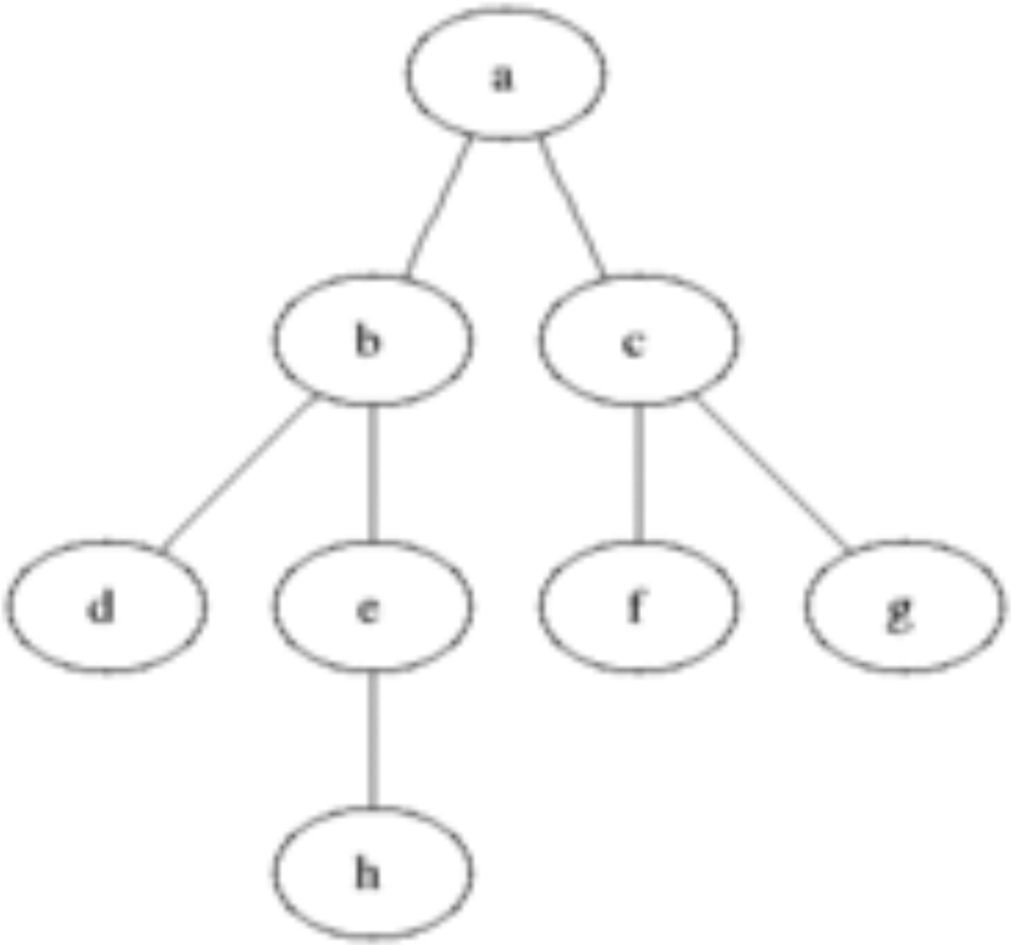


Dijkstra's progress, CC-BY 3.0, Subh83, 2011
https://commons.wikimedia.org/wiki/File:Dijkstras_progress_animation.gif

BFS search order (animation)

- $b = 2$
- $d = 3$
- $m = 3$

- **Time complexity**:
  - BFS: Time complexity = $\mathcal{O}\{b^d\}$
  - DFS: Time complexity = $\mathcal{O}\{b^m\}$
- **Space complexity**:
  - BFS: Space complexity = $\mathcal{O}\{b^d\}$
  - DFS: Space complexity = $\mathcal{O}\{bm\}$

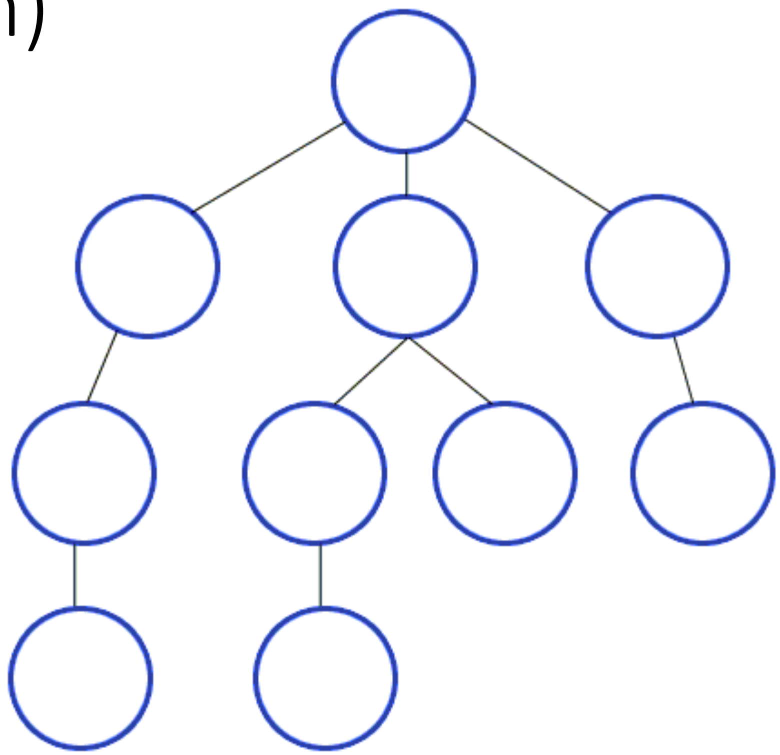


Animated-BFS. CC-SA 3.0, Blake Matheny, 2007
https://commons.wikimedia.org/wiki/File:Animated_BFS.gif

DFS search order (animation)

- $b = 3$
- $d = 3$
- $m = 3$

- **Time complexity**:
  - BFS: Time complexity = $\mathcal{O}\{b^d\}$
  - DFS: Time complexity = $\mathcal{O}\{b^m\}$
- **Space complexity**:
  - BFS: Space complexity = $\mathcal{O}\{b^d\}$
  - DFS: Space complexity = $\mathcal{O}\{bm\}$



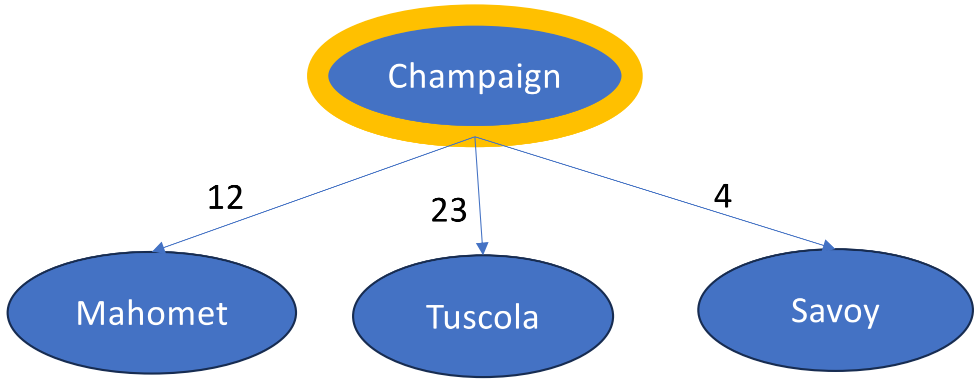
Depth-first-search. CC-BY-SA 3.0, Mre, 2009
https://commons.wikimedia.org/wiki/File:Depth-First-Search.gif

Outline

- Search Problems: start, goal, neighborhood
- Depth-first search (DFS): completeness, admissibility, & optimality
- Breadth-first search (BFS)
- Uniform-cost search (UCS)

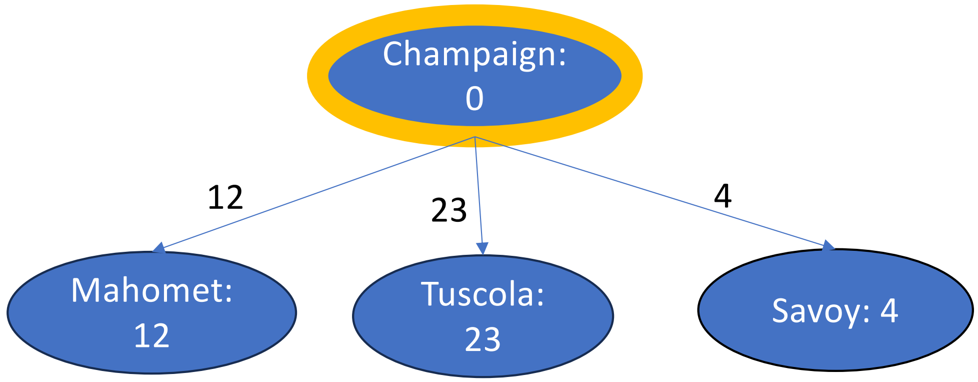
What about cost?

- Remember that not all edges have the same cost
- How can we guarantee that a search returns the path with the minimum total cost?



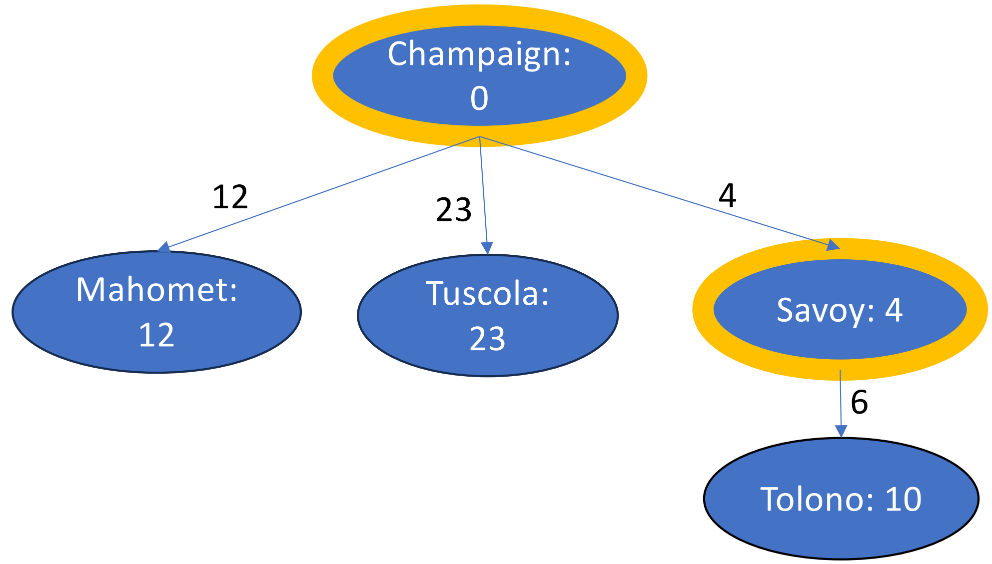
Uniform Cost Search

- Keep track of $g(n) =$ the cost of the shortest path from the start node to n
- The next node to expand = the node with the smallest cost



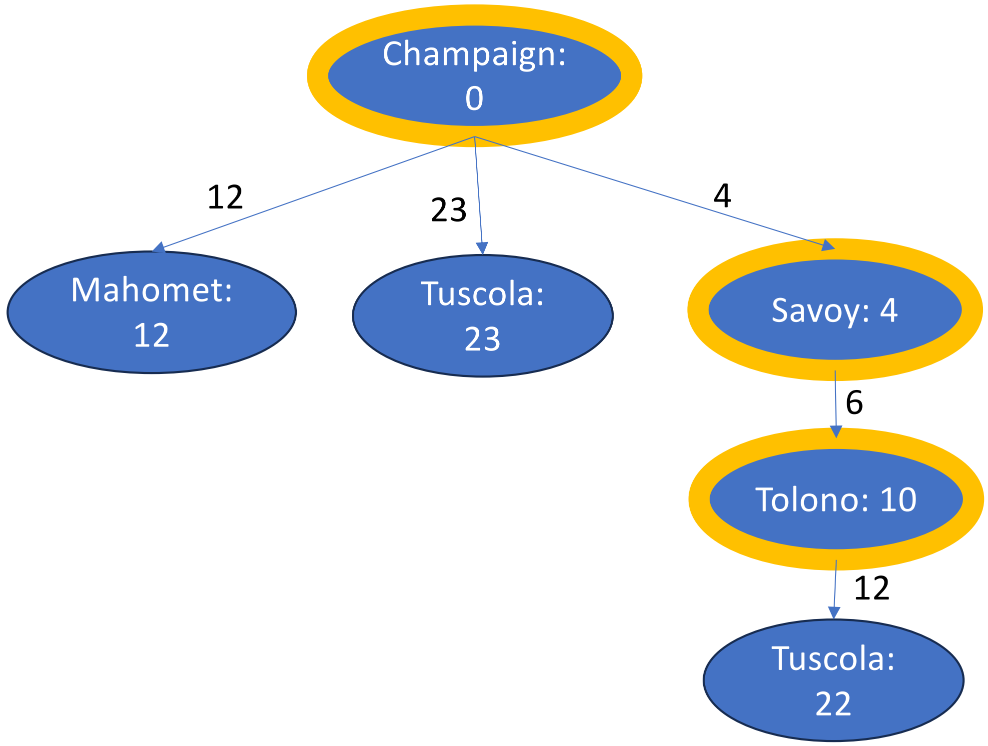
Uniform Cost Search

- Keep track of $g(n)$ = the cost of the shortest path from the start node to n
- The next node to expand = the node with the smallest cost



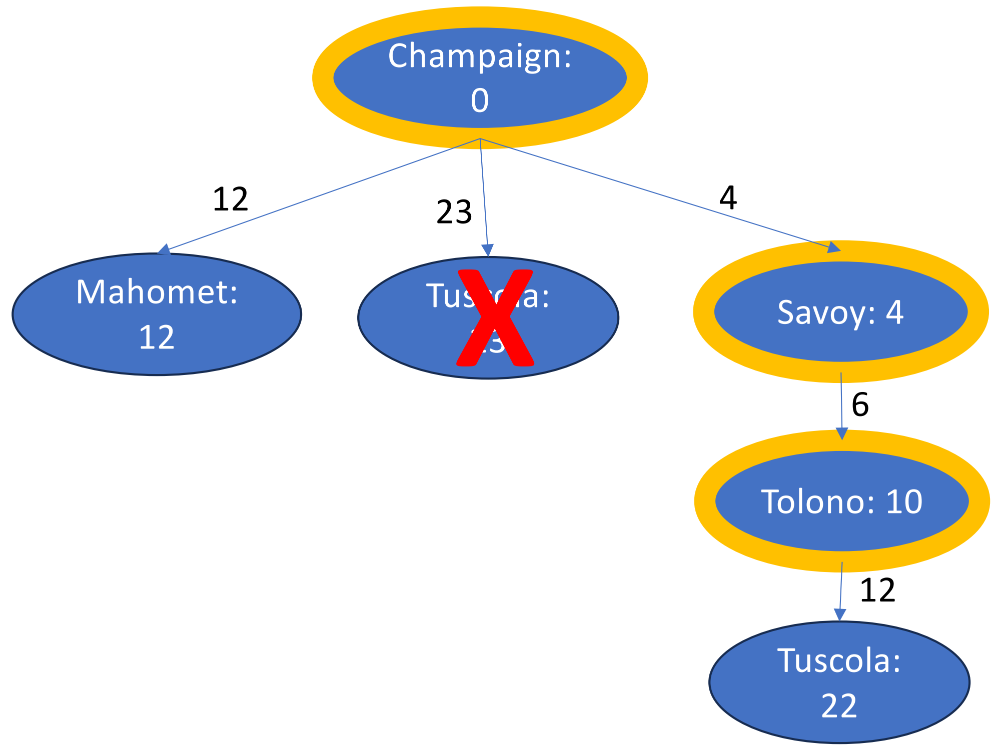
Uniform Cost Search

- Keep track of $g(n)$ = the cost of the shortest path from the start node to n
- The next node to expand = the node with the smallest cost



Uniform Cost Search

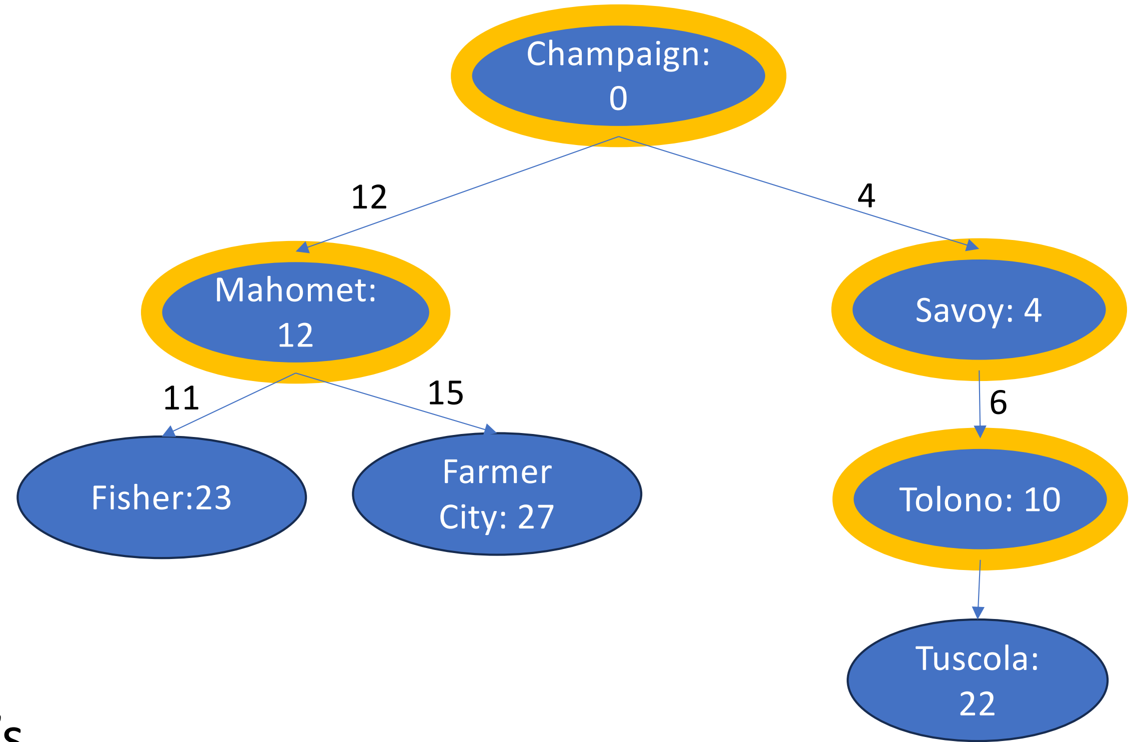
- If you find a shorter path to a node you are waiting to explore (we say this node is in your ”frontier”), keep only the shortest path



Uniform Cost Search

- Keep track of $g(n)$ = the cost of the shortest path from the start node to n
- The next node to expand = the node with the smallest cost

- Comment: also known as Dijkstra's algorithm
- Comment: if each step has the same cost, then UCS = BFS



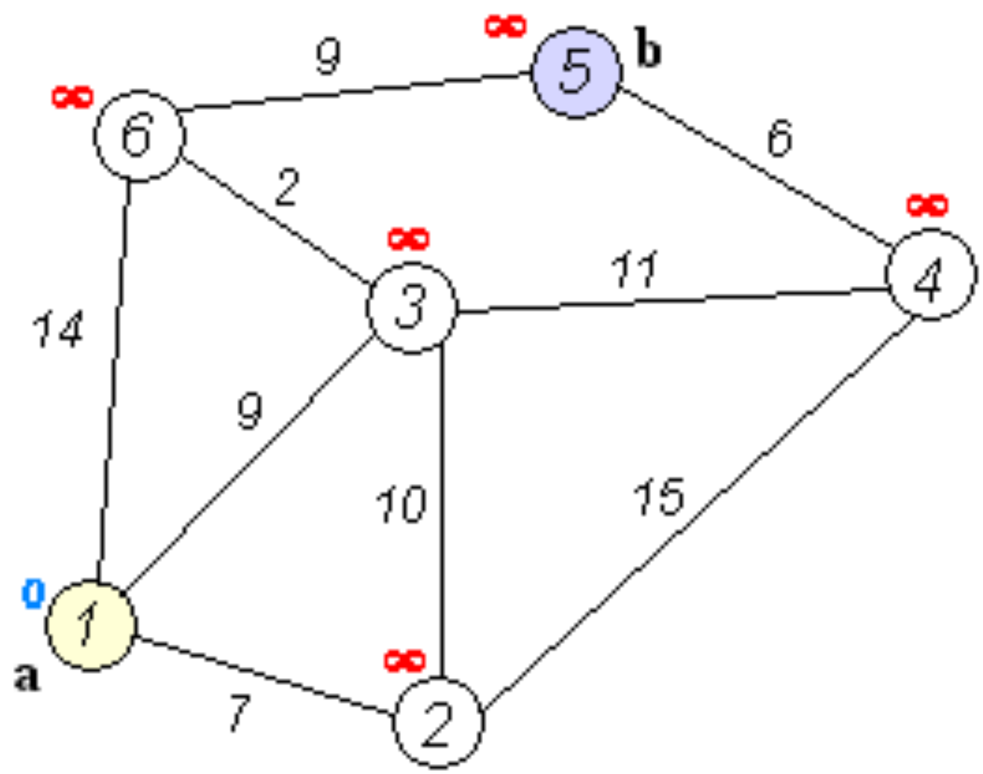
Try the quiz

Try the quiz!

Analysis of uniform-cost search

- **Complete?** Yes
  - If the goal can be reached with a total cost of $g^* = \min_{t \in \mathcal{T}} g(t)$, UCS will find a path with a cost of g^*
- **Admissible?** Yes
  - If the shortest total path cost is g^*, then UCS will find it
- **Optimal?** No
  - There are other algorithms that require less computation
- **Time Complexity=** # nodes with $g(n) \leq g^*$
- **Space Complexity=** # nodes with $g(n) \leq g^*$

Search order of UCS (animation)



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

Conclusions

- Depth-first search (DFS)
  - incomplete, inadmissible, non-optimal
  - Time complexity = $\mathcal{O}\{bm\}$, Space complexity = $\mathcal{O}\{b^m\}$
- Breadth-first search (BFS)
  - complete, inadmissible (unless each edge has cost 1), non-optimal
  - Time complexity = Space complexity = $\mathcal{O}\{b^d\}$
- Uniform-cost search (UCS)
  - complete, admissible, non-optimal
  - Time complexity = Space complexity = # nodes with $g(n) \leq g^*$