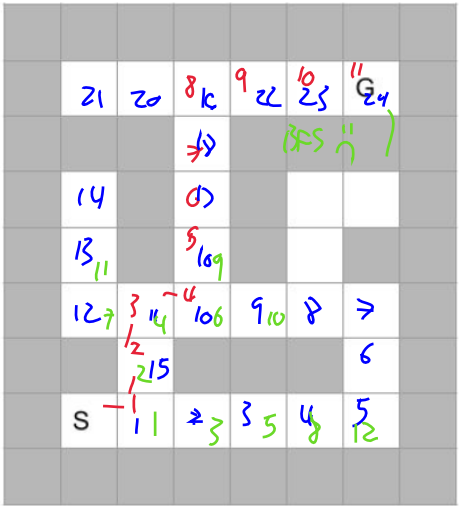
Sept 9: Uniform Cost Searches



Imagine telling an agent (Roomba?) to navigate this maze. We have either BFS or DFS, and we decide to "explore" actions in a clockwise order (N, E, S, W).

Which is faster? By how much?

Announcements and To-Dos

Announcements:

1. Make sure you check Piazza (`piazza.com/colorado/fall2020/csci3202`) for common HW questions and concerns!

Last time we learned:

1. Depth-first searching.

To do:

1. HW01 due tonight!

#5 getChildren portion of Node( ) can be inst. w/ Nodes

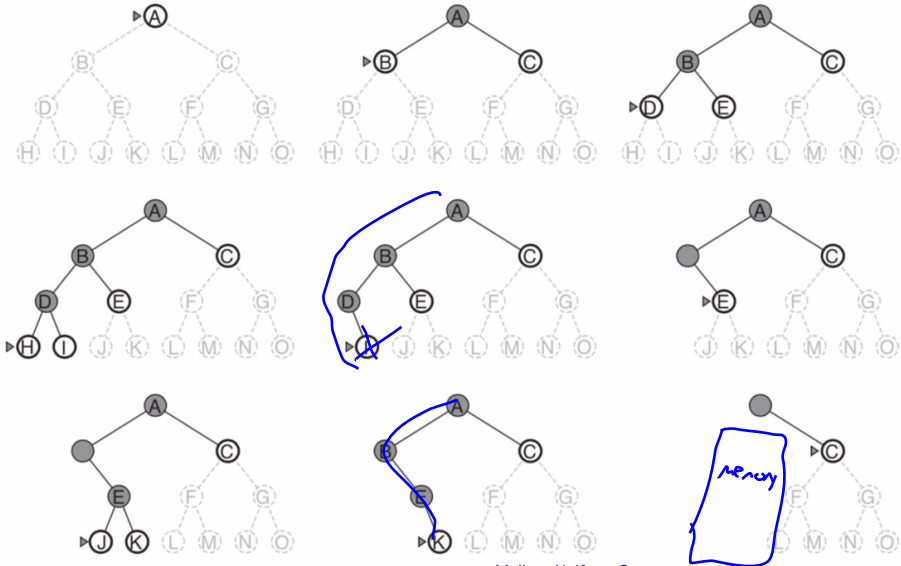
DFS Recap:

Last time we talked about **Depth-First Searching** (~~BFS~~ DFS).

Given a tree representing a state space, the DFS search performs the following unfolding of the tree:

1. At each iterative step, expand the deepest node first (last in, first out).
2. If we reach a redundant state or dead end, "back up" to the next-deepest node with unresolved successors.
3. Apply a goal test to each node *as it is generated*

DFS



Recall: Chicago to Boston DFS/BFS

neither used distances/miles

17

Now we explore **Uniform-Cost Searching** (UCS).

Unlike BFS and DFS, the UCS search can take into account edge weights. The UCS operates on a tree representation of a state space where spaces also have *costs* associated with moving between states. It then performs the following *uninformed* unfolding of the tree:

1. At each iterative step, expand the cheapest-to-reach frontier node first (lowest path cost).
2. In addition to the stacks/queues (DFS/BFS), UCS must track the path cost to reach each state on the frontier. This leads to a *priority* queue.
3. Goal test: when a goal state node is *selected* for expansion. This is different than BFS/DFS, which stopped when goal state reached frontier!

c Both
d 1 or neither

Is UCS **complete** and/or **optimal**?

17

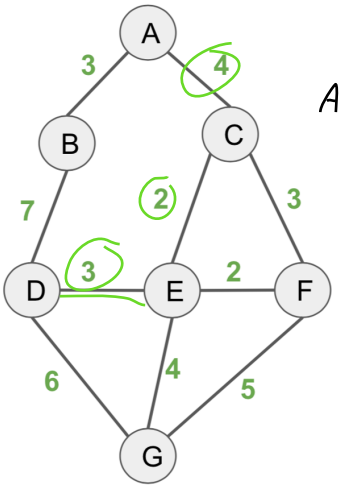
optimal ⊆ complete

1. The goal is when a node is selected.
2. Because we know we've taken the lowest cost path to that goal, UCS is **optimal**.
3. It's also **complete**, because it's a a more general form of BFS!

17

UCS Practice

Perform a UCS on this graph, with goal to move from A to G.



A-C-E-D

step 0: . frontier {(A,0)}

explored {0}

2nd argument: min distance from state to A

{(B,3),(C,4)} {(A,0)}

{(C,4),(D,10)} {(A,0),(B,3)}

{(D,10),(E,6),(F,7)} {(A,0),(B,3),(C,4)}

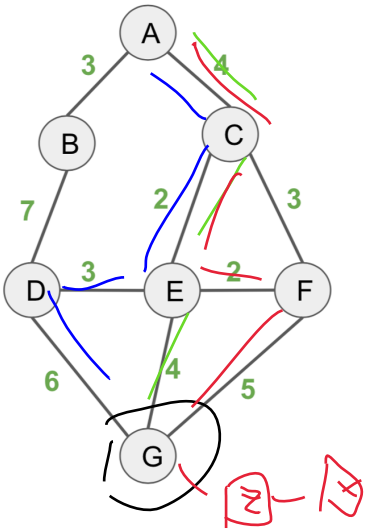
in C + cost(C,E)

via C via C

{(D,10),(D,9) . . . }

UCS Practice

Perform a UCS on this graph, with goal to move from A to G.



0 Explored: $\{\}$; Frontier: $[(A, 0)]$.

1 Explored: $\{(A, 0)\}$; Frontier: $[(B, 3), (C, 4)]$.

2 Explored: $\{(A, 0), (B, 3)\}$; Frontier: $[(C, 4), (D, 10)]$.

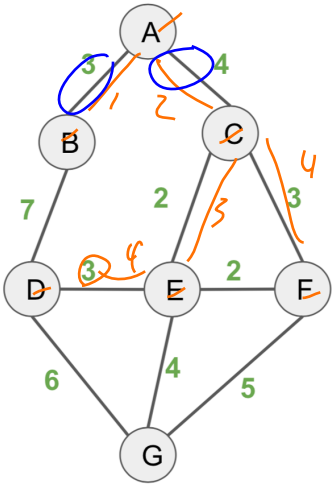
3 Explored: $\{(A, 0), (B, 3), (C, 4)\}$; Frontier: $[(D, 10), (E, 6), (F, 7)]$.

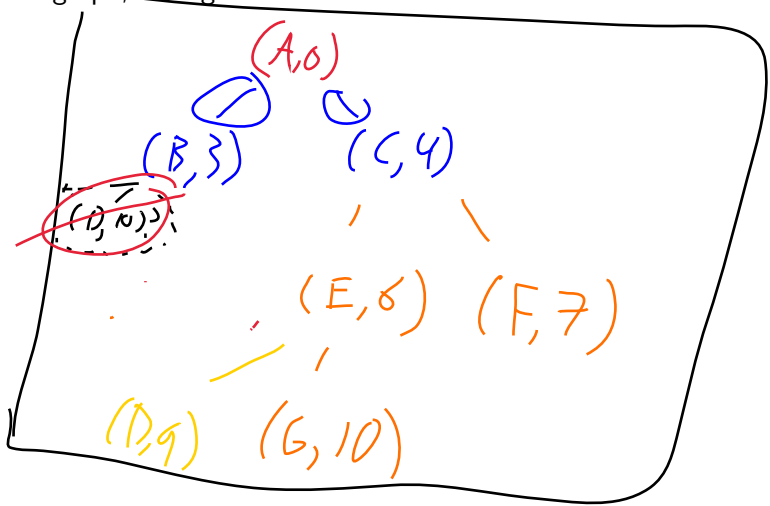
$\vdots$

k After expanding both D and E, we find $(G, 10)$ **and** $(G, 15)$ on the frontier. Not only can we already remove $(G, 15)$, we are done once we choose to expand $(G, 10)$

UCS Practice

Draw the tree depicting UCS on this graph, with goal to move from A to G.



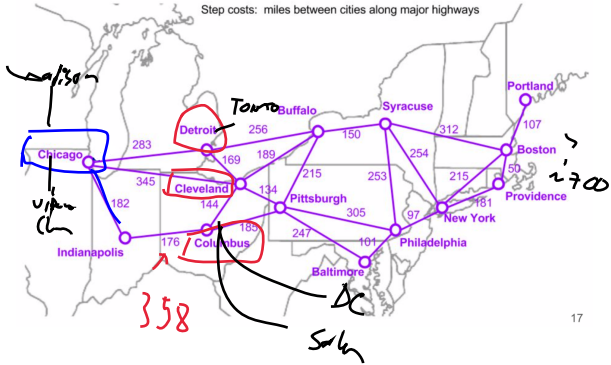


Consider our map problem, with the goal of traveling from Chicago to Boston. What are the first few steps of UCS, here?

{(CHI, 0), (IND, 182)}

{(CHI, 0), (IND, 182), (DET, 283)}

CLE, ... COL, ..., PIT, BUF

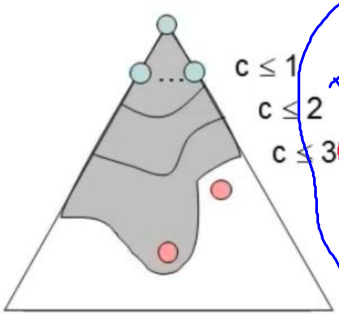


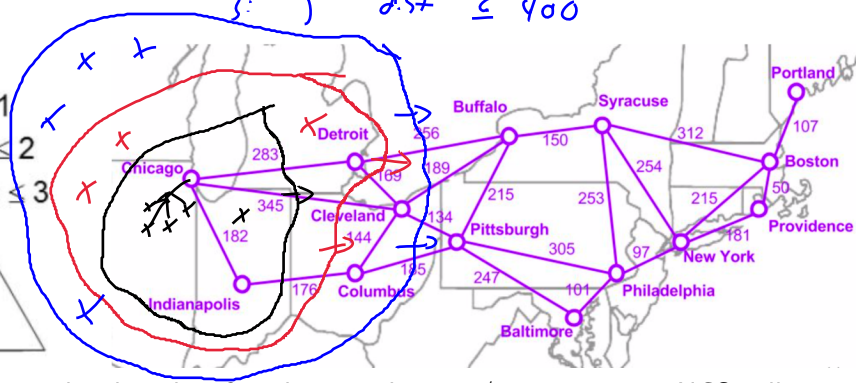
UCS complexity

Time Complexity Suppose as usual our UCS problem has *branching factor* b, and the optimal solution costs C^*. UCS can get stuck or struggle if there are sequences of no-cost actions: it will continually have to expand that branch of the tree.

0. The worst-case complexity depends on how many actions it might take to reach C^*. We shorthand this by denoting the minimal cost action ε.
1. UCS is $\boldsymbol{O}(b^{1+\lfloor C^*/\varepsilon \rfloor})$
2. This is a similar cost to BFS's $\boldsymbol{O}(b^d)$: consider C^* the depth of solution and $\varepsilon = 1$.
3. Can be inefficient without an informed approach because it explores in every "direction" from the starting point.

Countoured exploration



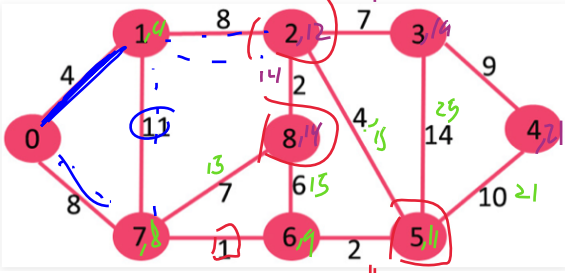


Exploring every direction can be thought of as drawing distance/cost contours: UCS will explore *every* location within one contour before moving to the next!

Dijkstra's Shortest Path Algorithm

You may have seen some shortest path algorithms before. UCS is a variation of Dijkstra's Shortest Path Algorithm.

Use UCS/Dijkstra's algorithm to find the shortest path from 0 to all other nodes. This creates a *shortest path tree*.



Contour: 8, 9, 11, 12, 14

to 4

(7, 15) via 1

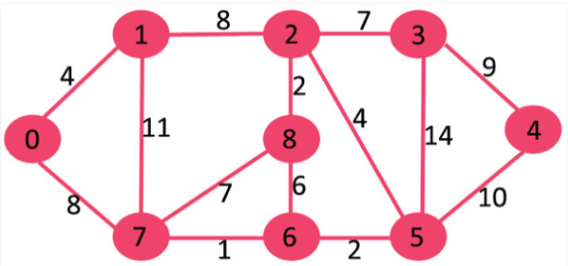
(7, 8) via 0

Vertex	Dist from 0	Prior State
0	0	NA
1	4	0
2	12	1
3	19	2
4	21	5
5	11	6
6	9	7
7	8	0
8	14	2

Dijkstra's Shortest Path Algorithm

You may have seen some shortest path algorithms before. UCS is a variation of Dijkstra's Shortest Path Algorithm.

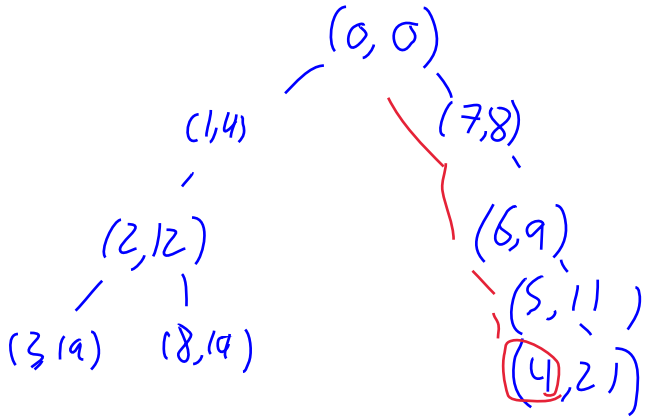
Use UCS/Dijkstra's algorithm to find the shortest path from 0 to all other nodes. This creates a *shortest path tree*.



Vertex	Dist from 0	Prior State
0	0	NA
1	4	0
2	12	1
3	19	2
4	21	5
5	11	6
6	9	7
7	8	0
8	14	2

Dijkstra's Shortest Path Algorithm

Drawing a Shortest path tree from the prior slide:



Moving Forward

- This week:
  1. HW1 Due ~~Friday~~ Today!: main issue seems to be challenges with typing of your getChildren() function: they can be Nodes! Check the couple of relevant Piazza posts.
  2. HW 2 for probable release around the weekend.
- Next time: A^* (Astar) searching!

Search Recap

Consider going from Arad to Bucharest. How does

1. BFS route us?

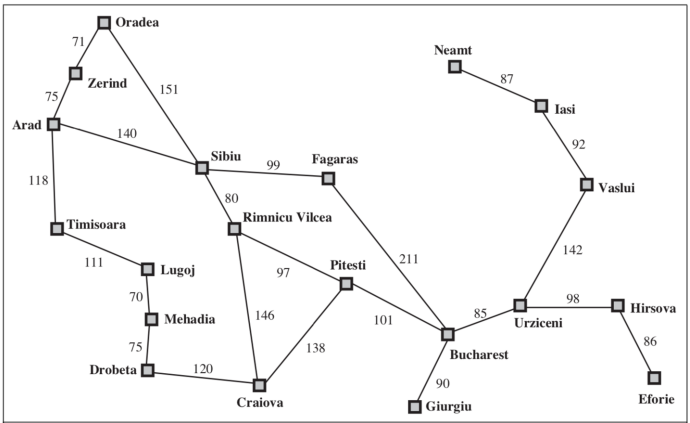


Figure 3.2 A simplified road map of part of Romania.

Search Recap

Consider going from Arad to Bucharest. How does

2 DFS route us?

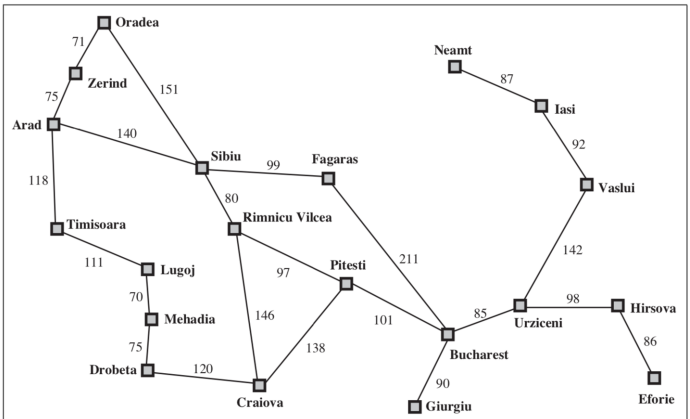


Figure 3.2 A simplified road map of part of Romania.

Search Recap

Consider going from Arad to Bucharest. How does

3 UCS route us?

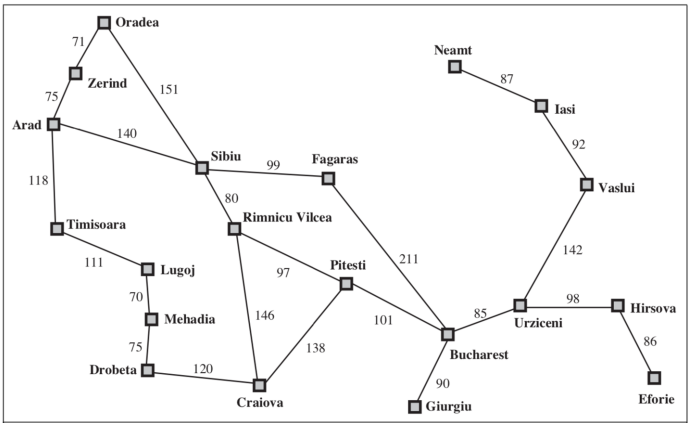


Figure 3.2 A simplified road map of part of Romania.