Kyle Tranfaglia

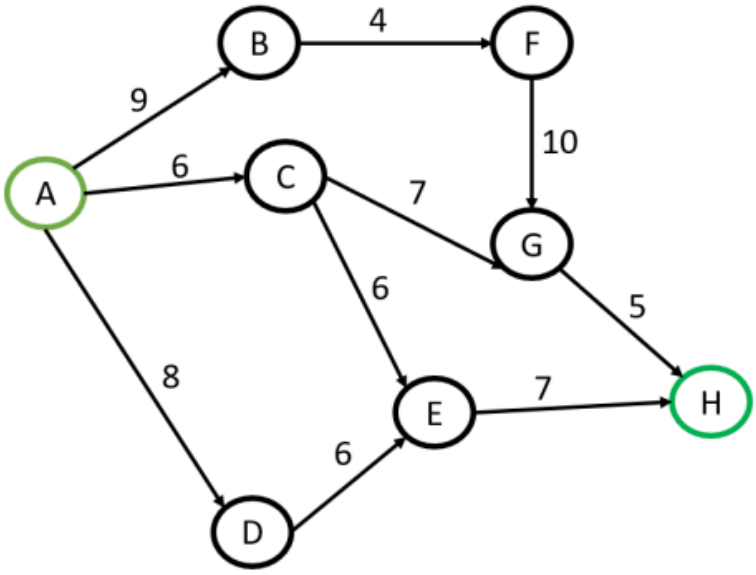
COSC 411

Homework 02

20 Oct. 2024

Uninformed and Informed Searches

1. Find the path from the start node A to the end node H. In a tie event, use the priority order: A > B > C > D > E > F > G > H.




Heuristic function:
- h(A) = 19
- h(B) = 20
- h(C) = 11
- h(D) = 13
- h(E) = 6
- h(F) = 8
- h(G) = 2
- h(H) = 0

Note the data structure for each search algorithm (queue, stack, etc.) represents the fringe: The nodes to explore/expand. This guarantees a path if one exists, either the leftmost or rightmost path, depending on order priority.

A) Depth-First Search: Explore as deeply as possible along a branch before backtracking

1. Start at A | Stack [A] | Expanded []
2. Expand node A | Stack [D, C, B] (Priority order utilized) | Expanded [A]
3. Expand node B | Stack [D, C, F] | Expanded [A, B]

4. Expand node F | Stack [D, C, G] | Expanded [A, B, F]
5. Expand node G | Stack [D, C, H] | Expanded [A, B, F, G]
6. Expand node H | Stack [D, C] | Expanded [A, B, F, G, H]
7. Goal reached: stop search. The path is the expanded nodes in order without including nodes that were dead-ends.

   Final Path: A -> B -> F -> G -> H

   Total Cost: 9 + 4 + 10 + 5 = 28

B) Breadth-First Search: Explore all neighbors at the present depth level before moving on to nodes at the next level. This guarantees the shortest path in terms of nodes & edges traveled.

1. Start at A | Queue [A] | Expanded []
2. Expand node A | Queue [B, C, D] (Priority order utilized) | Expanded [A]
3. Expand node B | Queue [C, D, F] | Expanded [A, B]
4. Expand node C | Queue [D, F, G, E] | Expanded [A, B, C]
5. Expand node D | Queue [F, G, E] | Expanded [A, B, C, D]
6. Expand node F | Queue [G, E] | Expanded [A, B, C, D, F]
7. Expand node G | Queue [E, H] | Expanded [A, B, C, D, F, G]
8. Expand node E | Queue [H] | Expanded [A, B, C, D, F, G, E]
9. Expand node H | Queue [] | Expanded [A, B, C, D, F, G, E, H]
10. Goal reached: stop search. The path is a traceback from the goal node to its parents, traveling the least amount of edges & nodes and handling ties on the level (breadth) using an alphabetical priority order.

    Final Path: A -> C -> G -> H

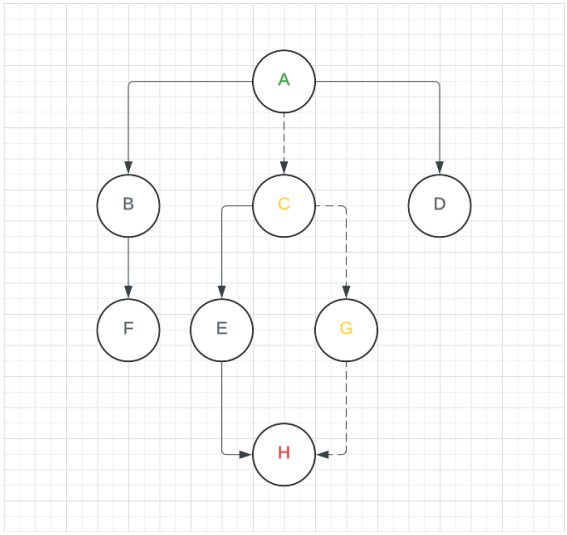
    Total Cost: 6 + 7 + 5 = 18

C) Uniform Cost Search: Expand the least-cost node first using path costs. It is guaranteed to find the shortest path regarding total cost but is relatively slow. *Priority queue is sorted by cost.

1. Start at A | Priority Queue [(A, 0)] | Expanded []
2. Expand A | Priority Queue [(C, 6), (D, 8), (B, 9)] | Expanded [A]
3. Expand C| Priority Queue [(D, 8), (B, 9), (E, 12), (G, 13)] | Expanded [A, C]
4. Expand D | Priority Queue [(B, 9), (E, 12),(G, 13)] | Expanded [A, C, D]
5. Expand B | Priority Queue [(E, 12), (F, 13), (G, 13)] (Priority order utilized)| Expanded [A, C, D, B]
6. Expand E | Priority Queue [(F, 13), (G, 13), (H, 19)] | Expanded [A, C, D, B, E]
7. Expand F | Priority Queue [(G, 13), (H, 19)] | Expanded [A, C, D, B, E, F]
8. Expand G | Priority Queue [(H, 18), (H, 19)] | Expanded [A, C, D, B, E, F, G]
9. Expand H | Priority Queue [(H, 19)] | Expanded [A, C, D, B, E, F, G, H]
10. Goal reached: stop search. The path is a traceback from H to the nodes that path to it with the lowest cost.

    Final Path: A -> C -> G -> H

    Total Cost: 6 + 7 + 5 = 18

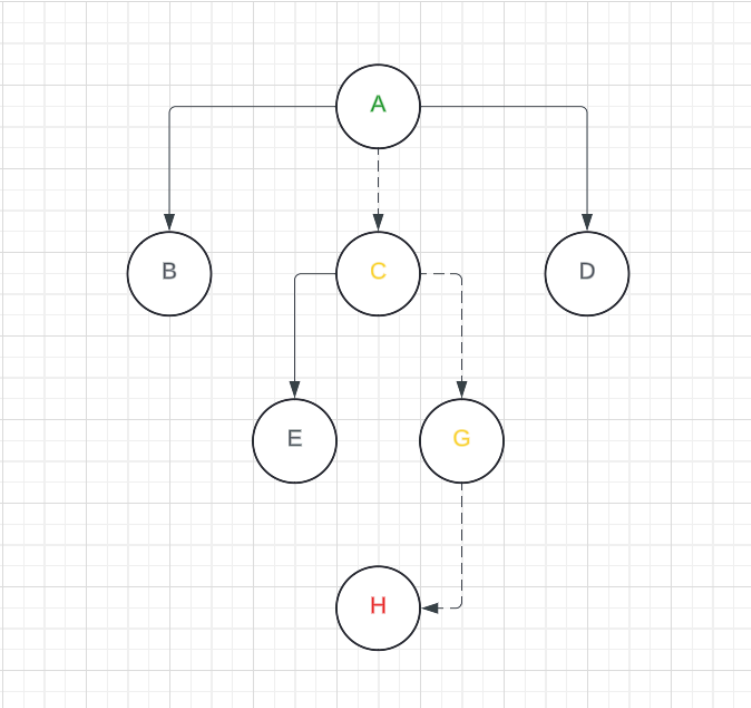


D) Greedy Search: Expands the node with the lowest heuristic value. The goal is reached (relatively) quickly, but it does not guarantee the optimal path. *Priority queue is sorted by heuristic.

1. Start at A | Priority Queue: [(A, 19)] | Expanded []
2. Expand A | Priority Queue: [(C, 11), (D, 13), (B, 20)] | Expanded [A]
3. Expand C | Priority Queue: [(G, 2), (E, 6), (D, 13), (B, 20)] | Expanded [A, C]
4. Expand G | Priority Queue: [(H, 0), (E, 6), (D, 13), (B, 20)] | Expanded [A, C, G]
5. Expand H | Priority Queue: [(E, 6), (D, 13), (B, 20)] | Expanded [A, C, G, H]
6. Goal reached: stop search. The path is a traceback from H to the nodes that path to it with the lowest heuristic values (nodes expanded in order).

   Final Path: A -> C -> G -> H

   Total Cost: 6 + 7 + 5 = 18

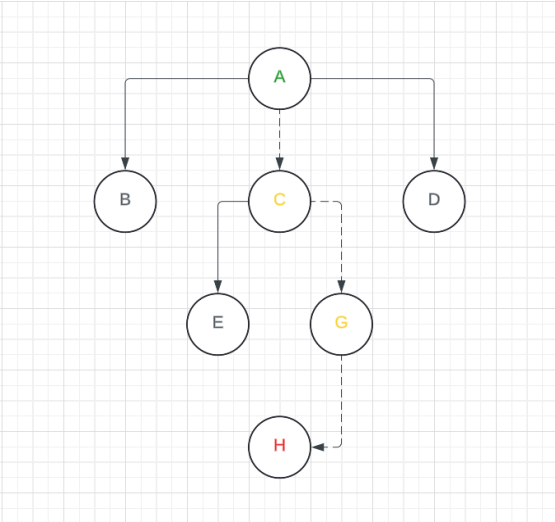


E) A* Search: combines features of both Greedy Search and Uniform Cost Search (UCS), using the path cost and a heuristic to determine the most promising node to explore next. It is optimal if the heuristic function h(n) is admissible (never overestimates the true cost to reach the goal). It is generally faster than UCS but slower than Greedy Search. Overall, it balances the immediate cost and the future potential cost. *Cost function f(n)=g(n)+h(n) is used to determine the next node to explore, where g(n) is the actual path cost, h(n) is the heuristic (the estimated cost to reach the goal), and f(n) is the sum of the two costs.

1. Start at A | Priority Queue: [(A, 19)] | Expanded []
2. Expand A |Priority Queue: [(C, 17), (D, 21), (B, 29)] | Expanded [A]
3. Expand C | Priority Queue: [(G, 15), (E, 18), (D, 21), (B, 29)] | Expanded [A, C]
4. Expand G | Priority Queue: [(H, 18), (E, 18), (D, 21), (B, 29)] | Expanded [A, C, G]
5. Expand H | Priority Queue: [(E, 18), (D, 21), (B, 29)] | Expanded [A, C, G, H]
6. Goal reached: stop search. The path is a traceback from H to the nodes that path to it with the lowest f(n) value (nodes expanded in order).

   Final Path: A -> C -> G -> H

   Total Cost: 6 + 7 + 5 = 18



2. Find the path from the start node S to the end node G. In a tie event, use the priority order: S > A > B > C > D > E > F > G.

A) Depth-First Search

1. Start at S | Stack [S] | Expanded []
2. Expand node S | Stack [D, A] (Priority order utilized) | Expanded [S]
3. Expand node A | Stack [D, B] | Expanded [S, A]
4. Expand node B | Stack [D, E, C] | Expanded [S, A, B]
5. Expand node C | Stack [D, E] | Expanded [S, A, B, C]
6. Expand node E | Stack [D, F] | Expanded [S, A, B, C, E]
7. Expand node F | Stack [D, G] | Expanded [S, A, B, C, E, F]
8. Expand node G | Stack [D] | Expanded [S, A, B, C, E, F, G]
9. Goal reached: stop search. The path is the expanded nodes in order without including nodes that were dead-ends.

   Final Path: S -> A -> B -> E -> F -> G

   Total Cost: 2 + 1 + 5 + 4 + 3 = 15

B) Breadth-First Search

1. Start at S | Queue [S] | Expanded []
2. Expand node S | Queue [A, D] (Priority order utilized) | Expanded [S]
3. Expand node A | Queue [D, B] | Expanded [S, A]
4. Expand node D | Queue [B, E] | Expanded [S, A, D]
5. Expand node B | Queue [E, C] | Expanded [S, A, D, B]
6. Expand node E | Queue [C, F] | Expanded [S, A, D, B, E, C]
7. Expand node C | Queue [F] | Expanded [S, A, D, B, E, C, F]

8. Expand node F | Queue [G] | Expanded [S, A, D, B, E, C, F, G]
9. Expand node G | Queue [] | Expanded [S, A, D, B, E, C, F, G]
10. Goal reached: stop search. The path is a traceback from the goal node to its parents, traveling the least amount of edges & nodes and handling ties on the level (breadth) using an alphabetical priority order.

    Final Path: S -> D -> E -> F -> G

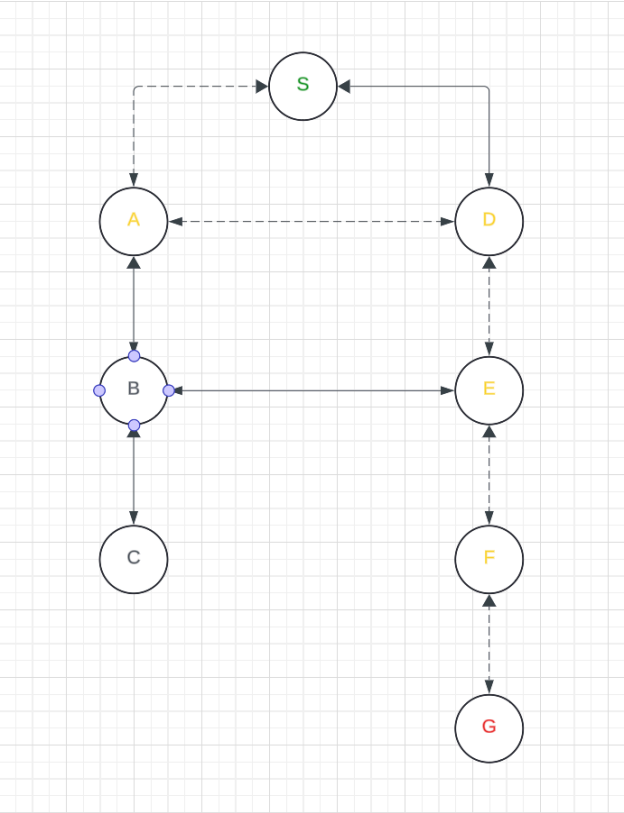
    Total Cost: 5 + 2 + 4 + 3 = 14

C) Uniform Cost Search

1. Start at S | Priority Queue [(S, 0)] | Expanded []
2. Expand node S | Priority Queue [(A, 2), (D, 5)] | Expanded [S]
3. Expand node A | Priority Queue [(B, 3), (D, 4), (D, 5)] | Expanded [S, A]
4. Expand node B | Priority Queue [(D, 4), (C, 7), (E, 8), (D, 5)] | Expanded [S, A, B]
5. Expand node D | Priority Queue [(E, 6), (C, 7), (E, 8)] | Expanded [S, A, B, D]
6. Expand node E | Priority Queue [(C, 7), (F, 10)] | Expanded [S, A, B, D, E]
7. Expand node C | Priority Queue [(F, 10)] | Expanded [S, A, B, D, E, C]
8. Expand node F | Priority Queue [(G, 13)] | Expanded [S, A, B, D, E, C, F]
9. Expand node G | Priority Queue [] | Expanded [S, A, B, D, E, C, F, G]
10. Goal reached: stop search. The path is a traceback from the goal node to its parents, traveling the least amount of edges & nodes and handling ties on the level (breadth) using an alphabetical priority order.

    Final Path: S -> A -> D -> E -> F -> G

    Total Cost: 2 + 2 + 2 + 4 + 3 = 13

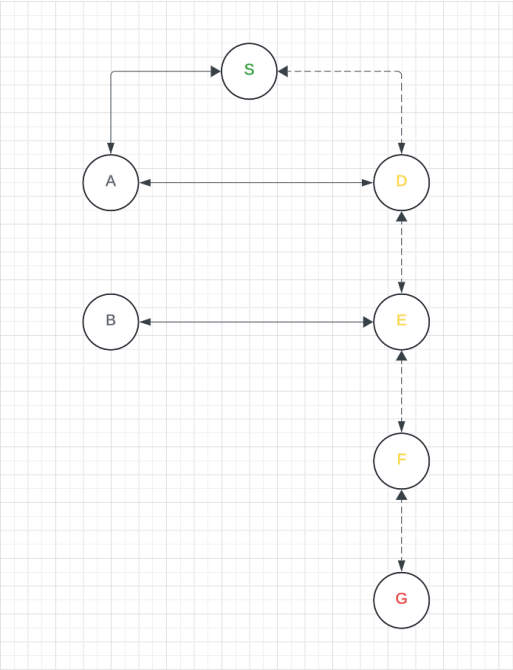



D) Greedy Search

1. Start at S | Priority Queue: [(S, 11)] | Expanded []
2. Expand S | Priority Queue: [(D, 8.9), (A, 10.4)] | Expanded [S]
3. Expand D | Priority Queue: [(E, 6.9), (A, 10.4)] | Expanded [S, D]
4. Expand E | Priority Queue: [(F, 3), (B, 6.7), (A, 10.4))] | Expanded [S, D, E]
5. Expand F | Priority Queue: [(G, 0), (B, 6.7), (A, 10.4)] | Expanded [S, D, E, F]
6. Expand G | Priority Queue: [(B, 6.7), (A, 10.4)] | Expanded [S, D, E, F, G]
7. Goal reached: stop search. The path is a traceback from G to the nodes that path to it with the lowest heuristic values (nodes expanded in order).

   Final Path: S -> D -> E -> F -> G

   Total Cost: 5 + 2 + 4 + 3 = 14

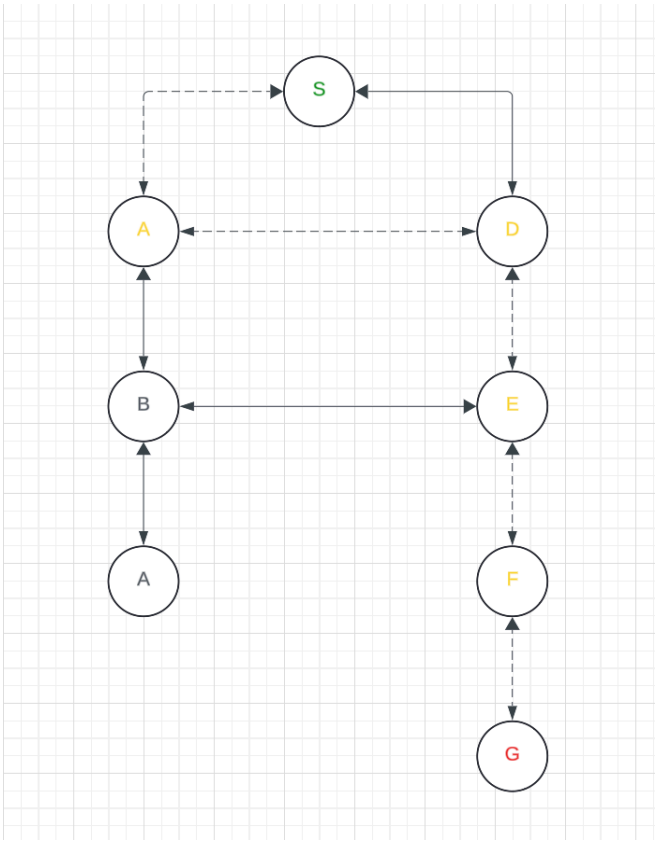



E) A* Search

1. Start at S | Priority Queue: [(S, 11)] | Expanded []
2. Expand S |Priority Queue: [(A, 12.4), (D, 13.9)] | Expanded [S]
3. Expand A | Priority Queue: [(B, 9.7), (D, 12.9), (D, 13.9)] | Expanded [S, A]
4. Expand B | Priority Queue: [(C, 11), (D, 12.9), (D, 13.9), (E, 14.9)] | Expanded [S, A, B]
5. Expand C | Priority Queue: [(D, 12.9), (D, 13.9), (E, 14.9)] | Expanded [S, A, B, C]
6. Expand D | Priority Queue: [(E, 12.9), (D, 13.9), (E, 14.9)] | Expanded [S, A, B, C, D]
7. Expand E | Priority Queue: [(F, 13), (13.9, D), (14.9, E)] | Expanded [S, A, B, C, D, E]
8. Expand F | Priority Queue: [(G, 13), (13.9, D), (14.9, E)] | Expanded [S, A, B, C, D, E, F]
9. Expand G | Priority Queue: [(13.9, D), (14.9, E)] | Expanded [S, A, B, C, D, E, F, G]
10. Goal reached: stop search. The path is a traceback from G to the nodes that path to it with the lowest f(n) value (nodes expanded in order).

    Final Path: S -> A -> D -> E -> F -> G

    Total Cost: 2 + 2 + 2 + 4 + 3 = 13


S
A
D
B
E
A
F
G