Shortest Paths and MSTs

Exam Prep 10

Announcements

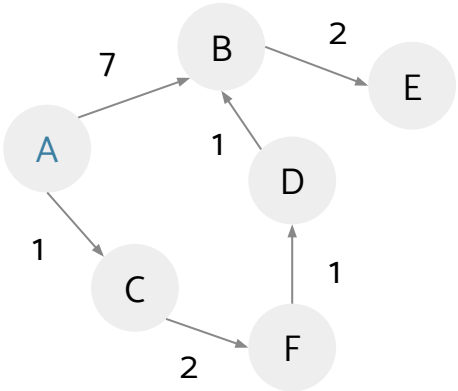
- Congrats on finishing the midterm!
- Project 2B Checkpoint due Monday 10/23
    - Project 2B due Monday 10/30
- Lab 09 due Friday 10/27
- Weekly survey due Monday 10/23

Content Review

Dijkstra's Algorithm

We've learned that BFS can help us find paths from the start to other nodes with a minimum number of edges. However, neither BFS or DFS account for finding shortest paths based off edge weight.

Dijkstra's algorithm is a method of finding the shortest path from one node to every other node in the graph. You use a priority queue that sorts vertices based off of their distance to the root node.

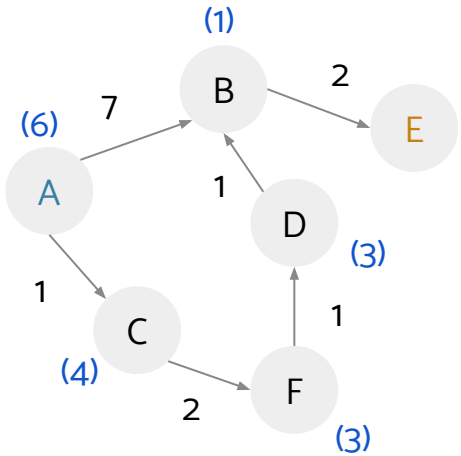


Steps:

1. Pop node from the front of the queue - this is the current node.
2. Add/update distances of all of the neighbors of the current node in the PQ.
3. Re-sort the priority queue (technically the PQ does this itself).
4. Finalize the distance to the current node from the root.
5. Repeat while the PQ is not empty.

A*

A* is a method of finding the shortest path from one node to a specific other node in the graph. It operates similarly to Dijkstra's except for that we use a (given) heuristic to estimate a vertex's distance from the goal.



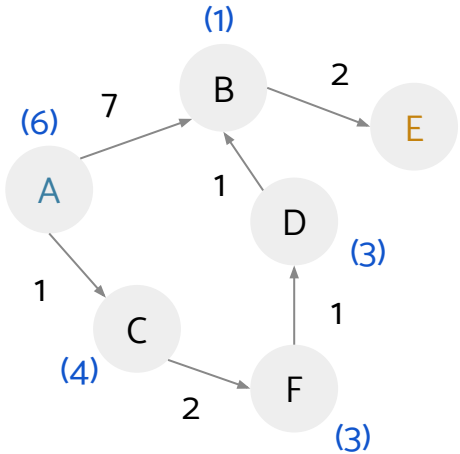
We're guaranteed to get the shortest path if our heuristic is admissible (never overestimates the true distance to the goal) and consistent (estimate always <= the estimated distance from any neighboring vertex to the goal + the cost of reaching that neighbor).

A*

A* is a method of finding the shortest path from one node to a specific other node in the graph. It operates similarly to Dijkstra's except for that we use a (given) heuristic to estimate a vertex's distance from the goal.

Steps:

1. Pop node from the top of the queue - this is the current node.
2. Add/update distances of all of the children of the current node. This distance will be the sum of the distance up to that child node and our guess of how far away the goal node is (our heuristic).
3. Re-sort the priority queue.
4. Check if we've hit the goal node (if so we stop).
5. Repeat while the PQ is not empty.



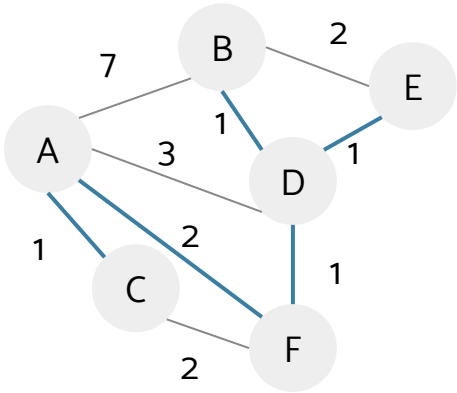
Note: the heuristic may not always be very good and might lead us down a path that isn't the shortest!

Minimum Spanning Trees

Minimum Spanning Trees are set of edges that connect all the nodes in a graph while being of the smallest possible weight.

MSTs may not be unique if there are multiple edges of the same weight.

There are two main algorithms for finding MSTs in this class: Prim's and Kruskal's. Both are based on the cut property: if we "cut" across any edges and separate the graph into two groups, the minimum weight edge that falls along that cut will be in some MST.



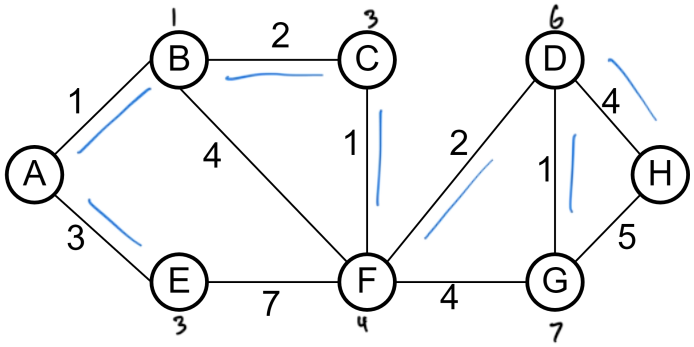
Worksheet

CS 61B Fall 2023

Shortest Paths, MSTs

Exam-Level 10: October 23, 2023

1 Dijkstra's, A*



(a) Run Dijkstra's Algorithm on the graph above starting from vertex A, breaking ties alphabetically. Fill in how the priority values change below. When you remove a node from the fringe, mark it with a check, and leave it blank for the subsequent rows. Stop when you remove G. Also sketch the resulting shortest paths tree in the end.

Node	A	B	C	D	E	F	G	H
Start	0	∞	∞	∞	∞	∞	∞	∞
Iter 1	✓	1	↓	↓	3	↓	↓	↓
Iter 2		✓	3	↓	↓	5	↓	↓
Iter 3			✓	↓	↓	4	↓	↓
Iter 4				↓	✓	↓	↓	↓
Iter 5				6		✓	8	↓
Iter 6				✓			7	10
Iter 7							✓	↓
								✓

(b) The heuristic distance from all nodes to G is defined below. Run A*, starting from A and with G as a goal. For each entry in the table below, fill in the distance, followed by the priority value of each node, separated by a comma. Is the heuristic admissible? Inadmissible: $h(A,G) > dist(A,G)$, $9 > 7$

u	A	B	C	D	E	F	G	H
$h(u,G)$	9	7	4	1	10	3	0	5

Node	A	B	C	D	E	F	G	H
Start	0, 9	∞	∞	∞	∞	∞	∞	∞
Iter 1	✓	1, 8	↓	↓	3, 13	↓	↓	↓
Iter 2		✓	3, 7	↓	↓	5, 8	↓	↓
Iter 3			✓	↓	↓	4, 7	↓	↓
Iter 4				6, 7	↓	✓	8, 8	↓
Iter 5				✓	↓		7, 7	10, 15
Iter 6					↓		✓	↓

2 Conceptual Shortest Paths

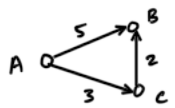
Answer the following questions regarding shortest path algorithms for a **weighted, undirected graph**. If the statement is true, provide an explanation. If the statement is false, provide a counterexample.

(a) (T/F) If all edge weights are equal and positive, the breadth-first search starting from node A will return the shortest path from a node A to a target node B.

True - can divide every edge weight to be 1, then it becomes BFS

(b) (T/F) If all edges have distinct weights, the shortest path between any two vertices is unique.

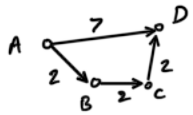
False - consider a counterexample that has a different # of edges



A → B : 5
A → C → B : 5

(c) (T/F) **Adding** a constant positive integer k to all edge weights will not affect any shortest path between two vertices.

False - shortest paths could still be affected by # of edges



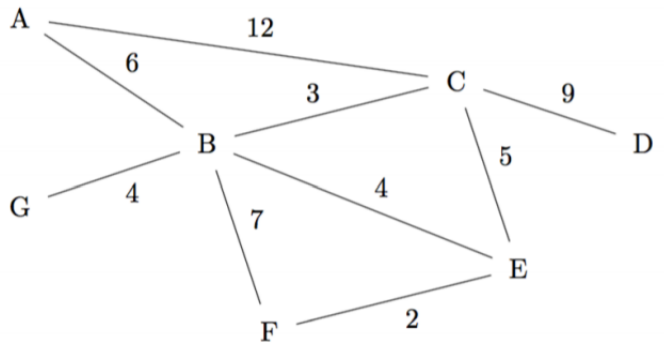
Set k to be 1
↳ A → D is 8
A → B → C → D is 9

(d) (T/F) **Multiplying** a constant positive integer k to all edge weights will not affect any shortest path between two vertices.

True - will scale up everything

Think about Dijkstra's as adding more nodes to a regular BFS

3 Introduction to MSTs



(a) For the graph above, list the edges in the order they're added to the MST by Kruskal's and Prim's algorithm. Assume Prim's algorithm starts at vertex A. Assume ties are broken in alphabetical order. Denote each edge as a pair of vertices (e.g. AB is the edge from A to B)

Prim's algorithm order: AB, BC, BE, EF, BG, CD

Kruskal's algorithm order: EF, BC, BE, BG, AB, CD

(b) True/False: Adding 1 to the smallest edge of a graph G with unique edge weights must change the total weight of its MST

True - this edge would be in MST as first edge from Kruskal's

(c) True/False: If all the weights in an MST are unique, there is only one possible MST.

True - Kruskal's would have only one ordering

(d) True/False: The shortest path from vertex u to vertex v in a graph G is the same as the shortest path from u to v using only edges in T, where T is the MST of G.

False - MST ≠ Dijkstra's

C and E in graph above