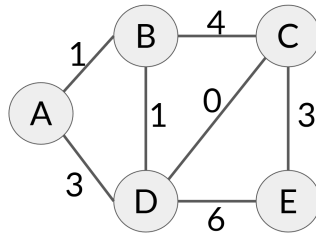


1 Dijkstra's, A*

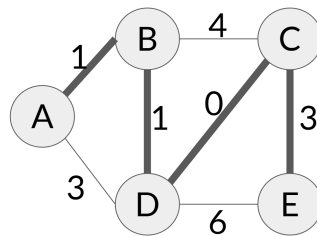


- (a) Run Dijkstra's Algorithm on the graph above starting from vertex *A*. Break ties alphabetically.
- Fill in how the priority values change below. When you remove a node from the priority queue, mark it with a check, and leave it blank for the subsequent rows.
 - Sketch the resulting shortest paths tree in the end.

Solution:

| Node | A | B | C | D | E |
|--------|---|----------|----------|----------|----------|
| Start | 0 | ∞ | ∞ | ∞ | ∞ |
| Iter 1 | ✓ | 1 | ∞ | 3 | ∞ |
| Iter 2 | | ✓ | 5 | 2 | ∞ |
| Iter 3 | | | 2 | ✓ | 8 |
| Iter 4 | | | ✓ | | 5 |
| Iter 5 | | | | | ✓ |

The SPT of the whole graph is drawn below for your reference. You should be able to “read-off” the edgeTo pointers by locating the last time the priority value of a node changes, and search for where the “checkmark” is for that row - that row would be the edge you came from in the SPT!



- (b) We have the following heuristic values. Run A*, starting from *A* and with *E* as a goal. In the table below, fill in the distance to A, followed by the priority value of each node, separated by a comma.

| | | | | | |
|-----------|---|---|---|---|---|
| u | A | B | C | D | E |
| $h(u, E)$ | 8 | 6 | 5 | 2 | 0 |

| | | | | | |
|--------|------|----------|----------|----------|----------|
| Node | A | B | C | D | E |
| Start | 0, 8 | ∞ | ∞ | ∞ | ∞ |
| Iter 1 | ✓ | | | | |
| Iter 2 | | | | | |
| Iter 3 | | | | | |
| Iter 4 | | | | | |
| Iter 5 | | | | | |

Solution:

| | | | | | |
|--------|------|----------|----------|----------|----------|
| Node | A | B | C | D | E |
| Start | 0, 8 | ∞ | ∞ | ∞ | ∞ |
| Iter 1 | ✓ | 1, 7 | ∞ | 3, 5 | ∞ |
| Iter 2 | | 1, 7 | 3, 8 | ✓ | 9, 9 |
| Iter 3 | | ✓ | 3, 8 | ∞ | 9, 9 |
| Iter 4 | | | ✓ | | 9, 9 |
| Iter 5 | | | | | 6, 6 |

The shortest path given by A* is: $A \rightarrow D, D \rightarrow C, C \rightarrow E$.

- (c) Is the heuristic admissible (it never overestimates the true distance to the goal for each vertex)? Is the heuristic consistent (does taking roundabouts through direct neighbors gives no better distance than the heuristic)?

Solution: The heuristic is not admissible, because the heuristic value of B is 6, which overestimates the shortest distance of B to E (which would be 4). The heuristic is also not consistent, because the taking the edge $B \rightarrow D$ (with weight 1) and the heuristic value of D (2) gives you a total estimated distance of 3, which is less than the heuristic value of B (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.

Solution: True. If all edges are equal in weight, then the shortest path from A to each node is proportional to the number of nodes on the path, so breadth first search will return the shortest path.

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

Solution: False. Consider a case of 3 nodes where AB is 3, AC is 5, and BC is 2. Here, the two possible paths from A to C both are of length 5.

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

Solution: False. Consider a case of 3 nodes A, B, and C where AB is 1, AC is 2.5 and BC is 1. Clearly, the best path from A to C is through B, with weight 2. However, if we add 1 to each edge weight, suddenly the path going through B will have weight 4, while the direct path is only 3.5.

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

Solution: True. Suppose we have arbitrary nodes u and v . Let's say the shortest path from u to v , before the multiplication by k , was of total weight w . This implies that every other path from u to v was of total weight greater than w . After multiplying each edge weight by k , the total weight of the shortest path becomes $w * k$ and the total weight of every other path becomes some number greater than $w * k$. Therefore, the original shortest path doesn't change.

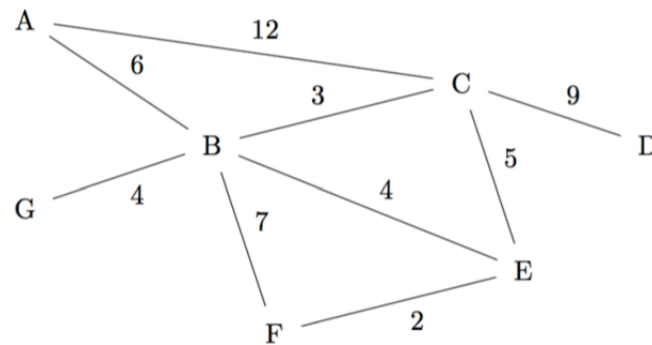
3 Shortest Paths Algorithm Design

Two countries, Mondstadt and Fontaine, are located in the fictional world of Teyvat. The railroad system of Teyvat can be modeled as a **weighted directed graph**, with V vertices, E edges, and weights being the length of the railway. Circle the traveler wishes to take railway from Mondstadt to Fontaine, and needs to determine the shortest railway distance between them. Define the set M to be all cities in Mondstadt, and F to be all cities in Fontaine. There can be cities that belong to neither country along the way. The shortest distance between the two countries is the shortest distance between any city c_M in Mondstadt and c_F in Fontaine.

Describe an algorithm that compute the minimum railway distance from Mondstadt to Fontaine, in $O((V + E) \log V)$ time. You are able to utilize all graph algorithms you learned in class. *Hint: Consider modifying the graph so that running a graph algorithm yields an equivalent answer to solving the original problem.*

Solution: Create a dummy vertex d . For each city in Mondstadt, create an edge from d to that city, with edge weight 0. This creates a new graph G' . Run Dijkstra, starting from dummy vertex d , on the graph G' . This would yield the distance from dummy vertex d to all cities in the Teyvat world. Now, because we are only interested in the minimum distance to any city in F , we can iterate through all cities in F and take the one with the minimum distance.

4 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:

Kruskal's algorithm order:

Solution: 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.

Solution: True, either this smallest edge (now with weight +1) is included, or this smallest edge is not included and some larger edge takes its place since there was no other edge of equal weight. Either way, the total weight increases.

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

Solution: True, the cut property states that the minimum weight edge in a cut must be in the MST. Since all weights are unique, the minimum weight edge is always unique, so there is only one possible MST.

- (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 .

Solution: False, consider vertices C and E in the graph above. The shortest path between C and E uses the edge CE , but it is not part of the MST of the graph.