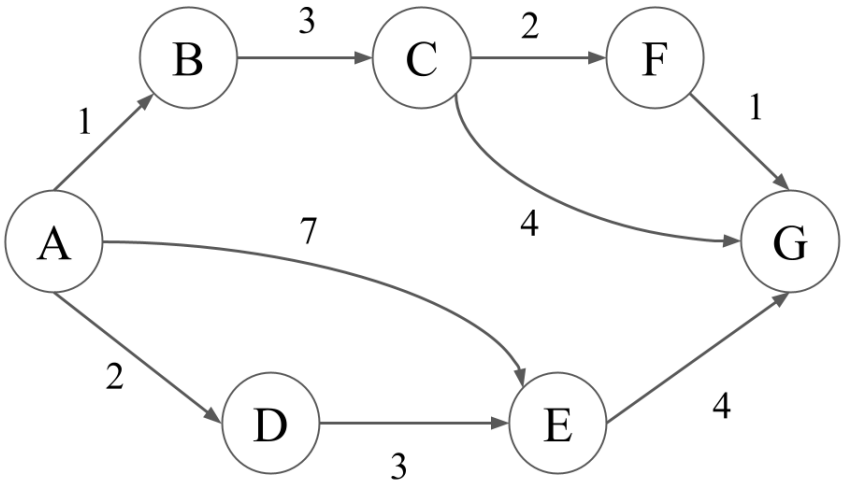
CS 61B Spring 2021

Shortest Paths and MSTs

Discussion 10: March 29, 2021

1 The Shortest Path To Your Heart

For the graph below, let `g(u, v)` be the weight of the edge between any nodes `u` and `v`. Let `h(u, v)` be the value returned by the heuristic for any nodes `u` and `v`.



Below, the pseudocode for Dijkstra's and A* are both shown for your reference throughout the problem.

Dijkstra's Pseudocode

```
PQ = new PriorityQueue()
PQ.add(A, 0)
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  popNode, popPriority = PQ.pop()

  for child in popNode.children:
    if PQ.contains(child):
      potentialDist = distTo[popNode] +
        edgeWeight(popNode, child)
      if potentialDist < distTo[child]:
        distTo.put(child, potentialDist)
        PQ.changePriority(child, potentialDist)
```

A* Pseudocode

```
PQ = new PriorityQueue()
PQ.add(A, h(A))
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  poppedNode, poppedPriority = PQ.pop()
  if (poppedNode == goal): terminate

    for child in poppedNode.children:
      if PQ.contains(child):
        potentialDist = distTo[poppedNode] +
          edgeWeight(poppedNode, child)

        if potentialDist < distTo[child]:
          distTo.put(child, potentialDist)
          PQ.changePriority(child, potentialDist + h(child))
```

(a) Run Dijkstra's algorithm to find the shortest paths from A to every other vertex. You may find it helpful to keep track of the priority queue and make a table of current distances.

B = 1 ; C = 4 ; D = 2 ; E = 5 ; F = 6 ; G = 7

Explanation:

For the best explanation, it is recommended to check the slideshow linked on the website or watch the walkthrough video, as the text explanation is verbose.

We will maintain a priority queue and a table of distances found so far, as suggested in the problem and pseudocode. We will use {} to represent the PQ, and (()) to represent the distTo array.

{A:0, B:inf, C:inf, D:inf, E:inf, F:inf, G:inf}. (()).

Pop A.
{B:inf, C:inf, D:inf, E:inf, F:inf, G:inf}. ((A: 0)).

changePriority(B, 1). changePriority(D, 2). changePriority(E, 7).
{B:1, D:2, C:inf, E:7, F:inf, G:inf}. ((A: 0)).

Pop B.
{D:2, C:inf, E:7, F:inf, G:inf}. ((A: 0, B: 1)).

changePriority(C, 4).
{D:2, C:4, E:7, F:inf, G:inf}. ((A: 0, B: 1)).

Pop D.
{C:4, E:7, F:inf, G:inf}. ((A: 0, B: 1, D: 2)).

changePriority(E, 5).
{C:4, E:5, F:inf, G:inf}. ((A: 0, B: 1, D: 2)).

Pop C.
{E:5, F:inf, G:inf}. ((A: 0, B: 1, D: 2, C: 4)).

changePriority(F, 6). changePriority(G, 8).
{E:5, F:6, G:8}. ((A: 0, B: 1, D: 2, C: 4)).

Pop E.
{F:6, G:8}. ((A: 0, B: 1, D: 2, C: 4, E: 5)).

potentialDistToG = 9, which is worse than our current best known distance to G. No updates made.

Pop F.
{G:8}. ((A: 0, B: 1, D: 2, C: 4, E: 5, F: 6)).

potentialDistToG = 7. changePriority(G, 7).
{G:7}. ((A: 0, B: 1, D: 2, C: 4, E: 5, F: 6)).

```
Pop G.
{}. ((A: 0, B: 1, D: 2, C: 4, E: 5, F: 6, G: 7)).
```

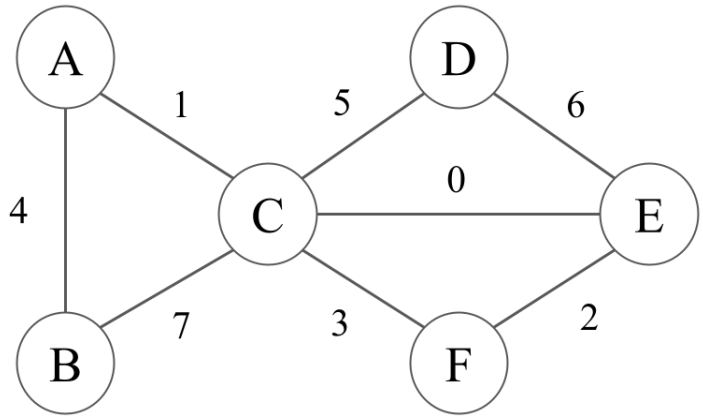
(b) *Extra:* Given the weights and heuristic values for the graph above, what path would A* search return, starting from A and with G as a goal?

A* would return $A - B - C - F - G$. The cost here is 7.

Explanation: A* runs in a very similar fashion to Dijkstra's. We got the same answer for the shortest path to G, though we actually explored less unnecessary nodes in the process (we never popped D and E off the queue). The main difference is the priority in the priority queue. For A*, whenever computing the priority (for the purposes of the priority queue) of a particular node n, always add $h(n)$ to whatever you would use with Dijkstra's. Additionally, note that A* will be run to find the shortest path to a particular goal node (as our heuristic is calculated as our estimate to our specific goal node), whereas Dijkstra's may be run with a specific goal, or it may be run to find the shortest paths to ALL nodes. In the solutions above, we found the shortest paths to all nodes, but if we only needed to know the shortest path to E, for example, we could have stopped after visiting E.

2 Minimalist Moles

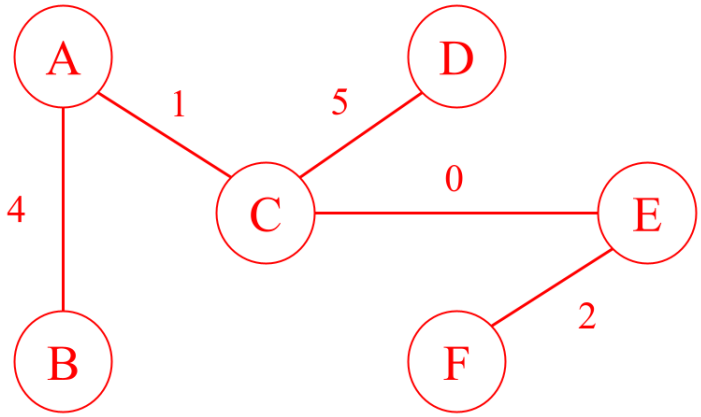
Mindy the mole wants to dig a network of tunnels connecting all of their secret hideouts. There are a set few paths between the secret hideouts that Mindy can choose to possibly include in their tunnel system, shown below. However, some portions of the ground are harder to dig than others, and Mindy wants to do as little work as possible. In the diagram below, the numbers next to the paths correspond to how hard that path is to dig for Mindy.



(a) How can Mindy figure out a tunnel system to connect their secret hideouts while doing minimal work?

This problem can be solved by finding a minimum spanning tree! This will connect all the secret hideouts (a tree will include all nodes in the graph) and it will take the minimum total work, since an MST will have the minimum total edge weight.

(b) *Extra:* Find a valid MST for the graph above using Kruskal's algorithm, then Prim's. For Prim's algorithm, take A as the start node. In both cases, if there is ever a tie, choose the edge that connects two nodes with lower alphabetical order. Both Prim's and Kruskal's give the MST below.



(c) *Extra:* Are the above MSTs different or the same? Is there a different tie-breaking scheme that would change your answer?

In this particular case, the trees for Prim's and Kruskal's are the same. There is NOT a different tie breaking scheme that would change this answer, because there is only one possible correct MST when all the edge weights are distinct! However, if a tree has edges with duplicate weights, then **it would be possible for Prim's and Kruskal's to give different answers**.