Dijkstra notes

Single source shortest paths.

* Show sample directed graph:
* 
  + How do I get from A to F?
  + actual answer is ABEF (dist = 6)
* all based around the following idea.  
  + Imagine I live at A, and I want to figure out the shortest path to all other vertices in my graph.
  + And I know a friend who lives at each of the other nodes B, C, etc.
  + So I start making a table, storing the distance from where I live at A to each other possible node in the graph.
  + So I look at all the places I could go from A (which are B and C).
  + So I write down in my table of distances: B: 2 and C: 4 (because right now I know that I could get from A to B by walking 2 miles and C by walking 4 miles).
  + Now I call up my friend at node B. And I say, "Hey Bob, what are all the vertices you can visit directly and how far away from you are they? And bob says C is 1 mile away from me, D is 4 miles away, and E is 2 miles away.
  + So I look at my table so far, and I see that I can get from A to C in 4 miles. But Bob just said he can get from B to C in 1 mile, and I know that I can get from A to B in 2 miles. So that's a total of 3 miles from A to C, if I go through node B. So I update my table: C=3 now.
  + Plus added new entries for D = 2+4=6, and E: 2 + 2 = 4.
  + Now I call up my friend Carla, who lives at node C, and I ask her the same thing.
  + Carla, where can you go directly? And she says, well I can go to E, and it's 3 miles away. And I know that right now I can get to Carla in 4 miles, and if she can go to E in 3 miles, that's a total of 7 miles to get to E if I go through Carla. But I already know by my table, that I can get to E in 4 miles (by going through Bob). So I'm not going to change my table.
* So that's the algorithm:
  + Keep a table of the "best distance found so far from the start node to each other node.
  + Visit each node of the graph in turn (in increasing distance away from the start node). Call this node U.
    - Look at each neighbor of the node, (call the neighbor V) and consider if it would be faster to visit V by going through U, or sticking to the best path to V I've already found in my table. [DRAW PIC BELOW]
    - If the path to V going through U is better than the one I've already found, then update my table.  
        
      A picture containing shape

      Description automatically generated
    - Ask: is dist[u] + weight(u, v) < dist[v]?
    - If so, update our table
  + When we've visited all the nodes in the graph, we're done, and we'll have a table of all the best distances to each other node.
* One caveat --- this algorithm doesn't let us reconstruct the best path, we only have the distance.
* So the way to fix this is every time we make a change to our "best distances" table, we're also going to store the "best previous node" – meaning the node that would come before this one in the best path.  
    
  Every time make a change in our distances table, meaning we've found a better path to some node V, we're also going to write down U, the node that will come before V in that path.