So as I understand it, we should be using a modified Prim's alg. that we saw in class for our Dijkstra implementation. Is this correct?
for each vertex in adj. list array
  u.key = infinity
  u.predecessor = null
root/source.key = 0
//Added the line below due to investigation in textbook
disjoint set S = newSet
build priority queue with vertex pointers
while queue is not empty
  u = extract-min
  //Added the line below due to investigation in textbook
  S = union(S,u)
  for each vertex v in the adj. list of u
    //This is the only modification to Prims, the condition below. Is this right for Dijkstra?
    if(v is in queue and u.key + weight(u,v) < v.key)
      v.predecessor = u
      v.key = weight(u,v)
But I'm still confused as to how there's a MST at the end. For instance, this seems to modify the original graph, and also excludes parts of the original graph that were disconnected, and therefore need to have their own MST produced.

EDIT: I guess the idea is that the original graph keeps it's description in the adj. list array, but each vertex now has a predecessor pointer which, with a specialized print function, can print the MST. Then, if the algorithm produced multiple MSTs, you can know that simply if your print function doesn't touch every vertex; it can then identify what vertices are left, and go from there?

EDIT 2: According to the textbook, page 658, there should be a disjoint set structure S which has UNION called on it for every vertex right after the EXTRACT-MIN. So the MST is there, and within S there will be one or multiple sets which are the product of Dijkstra and will be the MSTs.

Subject: Re: Correct Modification of Prim's?
Posted by lusth on Mon, 20 Mar 2017 16:35:21 GMT

Neither Prim's nor Dijkstra's uses disjoint sets (though Kruskal's does).
So this is what should happen then (and the result for our shortest path trees will be described by the altered predecessor pointers and keys (shortest distance) of each vertex in the adj. list array, and with a special print function we can go through and display each tree):

for each vertex in adj. list array
  u.key = infinity
  u.predecessor = null
root/source.key = 0
build priority queue with vertex pointers
while queue is not empty
  u = extract-min
  for each vertex v in the adj. list of u
    if(v is in queue and u.key + weight(u,v) < v.key)
      v.predecessor = u
      v.key = weight(u,v)

Subject: Re: Correct Modification of Prim's?
Posted by luth on Mon, 20 Mar 2017 17:58:13 GMT
View Forum Message <> Reply to Message

The key holds the length of the shortest path found so far. You're mixing and matching Prim's and Dijkstra's.

Subject: Re: Correct Modification of Prim's?
Posted by davidmccoy on Mon, 20 Mar 2017 18:28:11 GMT
View Forum Message <> Reply to Message

Whoops, right. So:
if(v is in queue and u.key + weight(u,v) < v.key)
  v.predecessor = u
  v.key = u.key + weight(u,v)