If I understand correctly, we will create a new object to be inserted into the binomial heap which contains the edge weight (which will be the key) and the two vertices it connects actually, the heap will just be storing each vertex, which will have an associated minimum distance and a list of neighboring vertices. The graph does not change after execution, hence this data in the heap shouldn't either. (although obviously, the heap itself will)

So what exactly is the update function used for? I cannot think of an implementation where the value itself would need to know where it is in the priority queue/heap. Properties of the graph are independent from the heap used to store them.

Subject: Re: Binomial Heap Update Function
Posted by davidmccoy on Sat, 18 Mar 2017 16:44:28 GMT

I thought we were using Prim's that we learned in class, but just modifying the distance check to be appropriate for Dijkstra instead of Prim. In that case, the binomial heap contains pointers to the vertices (which are kept in the adj. list array), and those vertices have a value (which should correspond with their index in the array, so perhaps this needn't be kept), a key (which is the current best known minimum distance to that vertex from the source/root), and a parent (which will be helpful when displaying the MST when we're done).

EDIT: Actually really confused. I know at the end we're supposed to have a MST, but how is that manifested? If we're not altering the original graph to be a MST, then do we create a new adjacency list to describe the new MST? And also, if the original graph is disconnected, then do we created multiple MSTs for each partitioned subgraph?

Subject: Re: Binomial Heap Update Function
Posted by SSisincho on Sat, 18 Mar 2017 17:07:13 GMT

Actually, you're right. I do believe we should be storing vertices, not edges.

I made another mistake. I think we actually do update a part of the value being stored in our heap, that being the current minimum distance. The minimum distance for each value will start at infinity, and eventually be updated to reflect the appropriate minimum distance for the MST. When printing output, we use the same values that were inserted with the original graph, just with their updated distance.
The algorithm starts by queueing all vertices, giving it a distance value of infinity (except for the source node). If the graph has disconnected portions, the queue will not be empty while all neighbors in the "connected" portion have been visited. So the program should know to select the next vertex in the queue for the new source node, set the distance to zero, and continue in the other portion.

Still, once we decreaseKey, the value shouldn't need to know weather or not it moved in the heap. So what's the purpose of the update function?

Subject: Re: Binomial Heap Update Function
Posted by davidmccoy on Sat, 18 Mar 2017 17:14:01 GMT

According to the "Rough Draft" spec sheet,
Quote: Note: a null pointer may be passed in as the updater, if this feature is not needed by the application. In this case, the updater function is never called.
So maybe we don't need it in this application, but it should be there as a generic feature of a binomial heap. The dynamic array and binomial heap classes we are producing, I guess, are to be made outside the realm of our specific implementation of Dijkstra's, and therefore may have features like update which are never called, but nonetheless will be tested.

Subject: Re: Binomial Heap Update Function
Posted by SSinischo on Sun, 19 Mar 2017 01:39:23 GMT

I think I understand the purpose of the update function.

We will have to create a Vertex object, with properties such as distance, neighbors, visited, and of course, actual value. BUT the decreaseKey function needs to take a pointer to whatever node it is changing. There is no way to find this node unless the Vertex object also has a BinomialNode as part of its struct.

When Vertex objects bubbleUp in the binomial heap, the update function needs to be called so that they know which BinomialNode they now belong to, hence if decreaseKey is called again, the binomial heap knows which one of its nodes to change.

Subject: Re: Binomial Heap Update Function
Posted by lusth on Mon, 20 Mar 2017 17:55:55 GMT

^ This is exactly right. If it possible for decreaseKey to be called multiple times on the same vertex (and it is in this application), then the updater function must be used to inform a vertex that it has a
different node in the heap holding it.