I have a series of questions regarding consolidation. The first question is regarding this line in the pseudocode, via the insert() function:
Quote: consolidate the root list of b using b’s comparator
I am assuming that it simply means to call the consolidation function via:
consolidate(binheap);
and the ‘using b’s comparator’ part is simply baked into the comparison function. Is my thought correct?
Second, where and how is the comparator function specifying the order of the heap? The first time I noticed that the comparator function is being used is in the consolidate() function.
Quote: update b’s extreme pointer if it’s null or b’s comparator indicates that D[i] is more extreme
I know to call the comparator function like
b->comparator(vertex_a, Vertex_b)
I am assuming that I am comparing D[i]->value and b->extreme->value, but I am unsure how to tell which is more extreme?
How does it know if the smaller key is more extreme or if the larger key is more extreme?
Is this where the function is deciding if it is a min-heap or a max-heap, or should something already know that?
The third and final question revolves around the updateConsolidationArray() function. Specifically this line in the pseudocode:
Quote: combine spot and D[degree], setting spot to the combined subheap
Are we combining spot and D[degree] using the linked list union() function? And I cannot figure out what ‘setting spot to the combined subheap’ means to save my life.

Subject: Re: Consolidation
Posted by btlindow on Thu, 17 Nov 2016 01:30:09 GMT

According to his pseudocode, yes, you would just call consolidate(b) assuming your binheap is named "b".

Consolidate takes all the items in the root list and merges them all together to minimize the rootlist. For instance: after an extractMin operation, we take all of the extreme’s children and throw them into the rootlist of b. But this is no longer an accurate binomial heap because we could have multiple items in the rootlist with the same degree. So, we must merge the children in the rootlist with like-degree subheaps. This is where we use the comparator. When we merge two like-degree subheaps, we want the Node with the smaller key value to be the parent of the other. Recall that degree is simply the amount of children that a Node has.

You aren’t necessarily comparing the extreme value with the item in the i'th entry of the array. You are comparing a like-degree Node with the i'th entry of the array. Depending on which one is smaller, one will take over as parent BUT it will go into the i+1'th entry because it’s degree has increased after absorbing the larger Node. I didn’t really worry about the extreme value UNTIL it came time to start putting the items in the array back into the rootlist (linked list form). This is
because it doesn't matter which value is the most extreme when consolidating them, we just need to get them back into binheap standard. We can worry about FINDING the extreme when it comes time to pushing them back into the rootlist from the array.

The comparator is what defines the binheap to be a min or max ordered heap. Simply flipping some values around in the comparator and you will return a value indicating the larger of the two values instead of the smallest.

To answer your final question, his pseudocode works like this:

ExtractMin does some stuff and calls Consolidate.
Consolidate calls UpdateConsolidationArray.
UpdateConsolidationArray calls Combine.

---

Subject: Re: Consolidation
Posted by bmbaker1 on Thu, 17 Nov 2016 02:40:48 GMT
View Forum Message <> Reply to Message

Wow! Thank's again! This is very helpful. I can't work on the 201 project right now, but I'm sure that I will have many followup questions when I do get to test all of this out.