Subject: Vertex class  
Posted by jgmurphy1 on Tue, 08 Nov 2016 18:39:19 GMT  
View Forum Message <> Reply to Message 

The void *value in the node class points to a vertex class correct?  
In class the vertex class is a struct with int key, vertex *predecessor, and node *owner. Would it  
also have a int value field for the number of the vertex? If not where is that value stored?

Subject: Re: Vertex class  
Posted by lusht on Tue, 08 Nov 2016 21:09:59 GMT  
View Forum Message <> Reply to Message 

Yes, a vertex object would have it’s vertex number.

Subject: Re: Vertex class  
Posted by btlindow on Mon, 14 Nov 2016 03:11:36 GMT  
View Forum Message <> Reply to Message 

I have a question about the vertex->owner. When we utilize bubble up within the binheap class,  
we are swapping values inside the Nodes but not actually updating the vertex->owners  
themselves. I found a fix, but I wanted to make sure it was the right fix before continuing on with it.

To keep the binheap class generic, I wrote another "wrapper" function like the comparator function  
and just pass my void* to that, cast to vertex, and make the swap there. Is this the right idea or  
was this handled somewhere else?

Thanks.

Subject: Re: Vertex class  
Posted by sestephens on Sun, 20 Nov 2016 00:40:46 GMT  
View Forum Message <> Reply to Message 

btlindow wrote on Sun, 13 November 2016 21:11To keep the binheap class generic, I wrote  
another "wrapper" function like the comparator function and just pass my void* to that, cast to  
vertex, and make the swap there. Is this the right idea or was this handled somewhere else?

What class is your wrapper function a member of? Binary heap, node, or vertex?

I'm pretty confused as to how to do this while maintaining generality.

Subject: Re: Vertex class  
Posted by sestephens on Sun, 20 Nov 2016 01:02:34 GMT
The only way that I've been able to think of to do this swap while maintaining generality is to actually just displace and swap the two nodes rather than swapping the values they contain. Swapping the values requires knowledge of what those values are in order to set the vertex->owner field; but if we just switch the nodes themselves, I think (read: hope) we're good.

So, I essentially switch each node's next, prev, parent, and child pointers while maintaining their values. Note that this does mean that for the line return bubbleUp(b,n's parent); you're actually still on n rather than on n's parent.

My b->comparator lives in my prim class, so I saw it fitting to put my b->swapper function inside prim's as well. And just as the comparator casts the void* to vertex*, I am doing the same in my swapper.

```cpp
swapper(void* a, void* b) {
    vertex* v = (vertex*) a;
    vertex* w = (vertex*) b;
    Node* t = v->owner;
    v->owner = w->owner;
    w->owner = t;
}
```

It's a void function that I call every time I perform a bubble up. I consider it to be generic still because I am not accessing any vertex functions in my binheap class. But I could be wrong. I just didn't see a different way to swap the two owners without using a class that has access to vertex's functions.

My concern with that is that, if I am not mistaken, you now have a second generic function that binheap needs to be initiated with (since bubbleUp is a binheap routine). Thus, when you call newBinHeap, you're going to need to pass in a second parameter to initialize b->swapper (something like void (*)(void *, void *)). But the assignment specifies that our function definition must look identical to this:

```cpp
binheap *newBinHeap(int (*)(void *,void *));
```
Meaning that it can only be initiated with a single function parameter.

Subject: Re: Vertex class
Posted by btlindow on Sun, 20 Nov 2016 04:51:10 GMT
View Forum Message <> Reply to Message

I guess it all comes down to whether it is acceptable to use my swapper or not. I mean, it works.

Subject: Re: Vertex class
Posted by lusth on Mon, 21 Nov 2016 17:10:10 GMT
View Forum Message <> Reply to Message

^ your swapper function is the right way to do this (it's known as a callback). I'll talk more about this in class Tuesday.

Subject: Re: Vertex class
Posted by oamohamed@crimson.ua.edu on Thu, 24 Nov 2016 16:28:27 GMT
View Forum Message <> Reply to Message

is the value and key in the vertex class is of type int, and if they are of type void*, how they would be cast to int?.(sorry if this has been answered in class or the forum).

Subject: Re: Vertex class
Posted by jgmurphy1 on Thu, 24 Nov 2016 18:20:36 GMT
View Forum Message <> Reply to Message

I am pretty sure everything in the vertex class can be hardwired to a data type since the the vertex object itself is used with void *.

Subject: Re: Vertex class
Posted by oamohamed@crimson.ua.edu on Thu, 24 Nov 2016 20:42:43 GMT
View Forum Message <> Reply to Message

this is a relieve, thanks.

also, my understanding about the predecessors is those vertices that have an edge with the vertex considered (not quit sure), and they help at the time of running breadth first, but my question is: at what point they got updated?
Lets say you are looking at all of the edges that leave vertex 1. And vertex 1 is connected to vertex 2 and vertex 4. If the edge weight that is connecting vertex 1 and vertex 2 is more extreme than is already stored in the key value of vertex 2, then you would update vertex 2's key value to that edge weight and set vertex 2's predecessor to 1 and we are done. Now, if vertex 4 already has a key value that is more extreme than the edge connecting it to vertex 1, then we leave that vertex alone.

In the case of this project, "more extreme" means smaller.

that means vertex 4 has been added to the minimum spanning tree right.

I am testing my linked list(kind of late for this assignment),how to go around the void * value inside the node struct to cast that value to an int? if I have a loop of int value and want to insert those values would assignment like void *i=i,before i do some else involve insert with the prev value?

Not necessarily. It is possible that Vertex 4 had it's key updated once before but never made it to the top of the priority queue. For instance:

1 5 1 ;
5 4 10 ;
5 6 2 ;
6 4 8 ;

1 would update 5 and put it at the top of the queue. Then we would update all the keys that touched 5, in this case 4 (key value of 10) and 6 (key value of 2) and 6 would make it to the top of the queue.
Then 6 would re-update 4's key value to 8 because it is lower than the value it had earlier, 10.

When you extract min off the top of your binheap, that is the vertex you add to the minimum spanning tree and would no longer update that vertex when you came across it during the "relaxation" stage.

Subject: Re: Vertex class
Posted by napatton on Sat, 26 Nov 2016 04:43:12 GMT
View Forum Message <> Reply to Message

To clarify for myself: It is acceptable to change newBinHeap(comparator) to newBinHeap(comparator, swapper)?

Subject: Re: Vertex class
Posted by lusth on Mon, 28 Nov 2016 15:52:04 GMT
View Forum Message <> Reply to Message

napatton wrote on Fri, 25 November 2016 22:43: To clarify for myself: It is acceptable to change newBinHeap(comparator) to newBinHeap(comparator, swapper)?

Yes.