Assignment 3 Documentation

Adham Makar - 301584092

I learned that I need heap and Hmap class in my indpq class in lecture so, I started by deciding to work around my assignment 2 and try to have the least changes with it since it's an already done Hmap. I took the member functions of heap from the textbook and lecture slides and modified them so they can work using my own thought process using the function swap. Now that I have both Heap and BiMap, I realized I have two options:

1. Have 2 member variables inside indPQ each calling to a different class (BiMap, Heap).
2. Have 1 member variable inside indPQ calling on heap and have heap call on BiMap.

Option 1 seemed more difficult to me, as dealing with 2 different variables seamed messy and I could not think of pseudocode or a way of doing it.

Where Option 2 seamed simple, I can work on one part at a time. And now that I have decided to go with option 2, here are the steps:

1. Once I call on the indpq insert, I wanted to call on the Heap insert and then the heap insert will call on the BiMap insert.

2. Then I realized that my Bimap will be out of order with my Heap order, so I had to call on insert inside the percolate functions, so that every time a change of order happens in the heap, it is reinserted into the BiMap

3. Then The testing phase arrived where I needed to make sure, my insert functions work like I would like them to. Then I faced another problem where I cannot just call on the arrays since they are private in the indpq scope, so I needed to add getfunctions to get the bimap array and the head array and used a for loop to display them and make sure they are correctly ordered.

4. Now that my Insert works, and my logic also works. I only had to implement the functions given in the instructions. This last task was easy as I realized that my BiMap class will not be needing any changes as it needs all the functions it will be using already implemented. I just have to connect these functions to the new function of indpq:

1) I still used the same method from option 2. DeleteMin() calls on heap.DeleteMin which looks at index 0 of the heap array then uses the function back() and pop_back() to remove the corresponding Task and acquires the corresponding task to use it in the removekey()

function in the BiMap. Now the Min is deleted from both classes and then we percolate down and insert the new index and corresponding task into the BiMap using insert

2) The rest of the functions were implemented the same way for example update priority, takes two arguments the task name and the new priority. It uses the task name to get the previous priority ID from BiMap using the getVal function and using the index I can update the priority just by using heapArray[index].priority = newPriority;. I could have used heapArray[Taskid].priority = newPriority; However, as I needed the index to percolate up and down, I saw no harm of doing either methods.

3) As for isEmpty and size, these are just already existing functions that I can use inside my member functions. But still just like all the other indpq member functions it follows option 2, they are calling a heap member function and if needed the heap function will call on a BiMap member function. Example:

```
string isEmpty(){
        return heap.empty();
}
```

4) The remove function was a bit interesting; I was planning to have a similar code for remove and delete min, when I realized that I could just use deletemin as a member function of remove. I did that by setting the desired task to have a negative priority which is the lowest possible priority, then simply using deletemin.

5) After doing the remove function it was simple to observe that the same method could be used for clear, where I used the isEmpty() function to set a while loop to deletemin if isEmpty is false

6) Now only display and ddisplay were left, this is when the get functions of the heap array and and the BiMap array that I did earlier came into handy, and I only needed to add one more get function to get the state of the index (ACTIVE, EMPTY, DELETED). For displayI just used heap.getheaparray and a for loop to go through the heaparray and display the task and priority. For ddisplay I used a size dependednt for loop to go through my heap array to make sure I also get the empty and deleted indexes

Challenges:

A lot of the challenges that I faced were related to the privacy of classes, where I tried to directly reference private members of classes, and it resulted in errors. As it was nothing major, it was annoying as I had to make a lot of repetitive editing. I fixed it by making gt functions

A second challenge and biggest challenge was figuring out how to start, or what my code is supposed to do. I needed help figuring out what should the end goal be, however, just doing the code step by step helped put everything together.

Testing

I thought about testing my percolate functions, but I could not figure out a way to do that without having an actual class, so I decided to test it alongside insert.

After I implemented the insert functions, I tested for a while as it is important that the insert and percolate functions work as they are the base of the whole assignment. I wanted to make sure that BiMap and Heap are both getting the inserted values and everything is getting ordered correctly

Then from there I just tested each function after completing it