**Data Structures and Algorithms II**

**Fall 2023**

Student name: Hyeonseo Kim

Assignment: Program #2

Grade: 65 (see final comment for explanation)

Comments:

The program does not work for the provided test case, there is incorrect output. I'll try to track this down before running my much larger tests. For the provided test case, when I compare my output to yours with "diff", this is what I see:

$ diff ../../TESTS/myout1.txt o1.txt

145c145

< Deleted item with string id "string one" and key 50

---

> Deleted item with string id "" and key 0

165c165

< Deleted item with string id "string five" and key 60

---

> Deleted item with string id "string one" and key 50

176c176

< Deleted item with string id "string seven" and key 70

---

> Deleted item with string id "string five" and key 60

185c185,187

< Call to 'deleteMin' returned: 1

---

> Call to 'deleteMin' returned: 0

>

> Deleted item with string id "string seven" and key 70

The lines starting with < are in my output, the lines starting with > are the replacement lines in your output.

I'll look through your heap code first; I'll comment on any issues I notice.

This is not a bug, but this is strange in heap's insert routine:

if(pv == nullptr){data[new\_position].pData = nullptr;}

else{data[new\_position].pData = pv;}

data[new\_position].pData = pv;

The if/else statement is not needed, and the assignment you do after it (setting the pData field to pv no matter what) is better, and redundant. Also, you don't need to reset new\_position to 0 before returning, there is no need. -1 point

I believe I see a bug in heap::remove here:

if (data[current\_size].key < data[pos].key) {percolateUp(pos);}

else {percolateDown(pos);}

You have already overwritten the node at data[pos], and you've already reset the node at current\_size. So, at position "pos", you have the new key, and the old key is lost. You need to store the old key before you overwrite it. -5 points

I fixed this, but I still see incorrect output (but not all the same as before).

I didn't notice other heap issues (although I could be missing some), so I am moving on to inspect the hash code for now.

In hashTable::insert, some code is not indented, which affects readability. -2 points

In hashTable::insert, you should only break out of the loop and return 1 if the duplicate key that matches what you are looking for is not deleted. -5 points

The check for rehashing in in a bad place. You need to do it at the start of the routine, or right before the return. Otherwise, the insertion causing the rehash will wind up in the wrong place. Also, you are doing integer division there, which won't work; it will be 0 until it is 1! You need to cast one of the items as double (or there are other ways to make it work). -5 points

I fixed these issues, but still see incorrect output.

There is another major issue with the hash table's insert function; you are NOT setting the pv value! You need to do this here (as well as in setPointer). -5 points

In findPos, you need to make sure that the matching key has not been lazily deleted before returning the pos; otherwise, keep searching. -5 points

In rehash, you need to make sure that each item has not been deleted, and you need to reinsert the void pointers along with the keys. (You also don't need to explicitly compare to true and false; for a Boolean, just do if (b) or if (!b).) -5 points

After all of these fixes, I am still seeing incorrect output for the provided test case.

When you lazily delete an item, you should NOT set isOccupied to false; the item is still occupied (and also lazily deleted). This would break other member functions. -5 points

Not a bug, but in several locations, you should not call findPos twice. Just call it once and store the return value. That avoids linear probing twice. -3 points

I can't keep debugging, there is still incorrect output for the provided test case. You are going to need to get this working to use your hash and heap classes for your third program. Start with all the fixes indicated here, and see if you can find other issues. You can resend it to me to check that it is working on the heap test cases, but it won't affect your second program grade.

I am going to give you a 65 here, rather than take off all the individual points noted above, which would be worse, because the program is no the right track; but you clearly still have a lot of debugging to do. Please make sure it works for the provided test data before sending it to be rechecked. Of course, feel free to come talk to me if you are having trouble.