Description of project:

I implemented Hollow Heap for the CMPUT 403 final project. The data structure is a simpler and better improvement of the Fibonacci heap. It is a heap structure that extends the "tree" to "dag". Compared to standard Binary Heap, the insert, decrease_key, and merge functions of Hollow Heap use O(1) time, rather than O(log N) or O(n). If the problem requires lots of insertion, decrease_key, or merge operations, then the Hollow Heap wins significantly.

Running time Analysis:

Insert()	O(1)
meld()	O(1)
find_min()	O(1)
decrease_key()	O(1)
delete_min()	O(logn)
link()	O(1)
add_child()	O(1)
delete()	O(logn)

All operations except delet_min() cost constant time, since almost all of these functions, mainly do value swaps and value assignments without loops. However, for the delete() function, if the node to be deleted is not the root, then the operation costs constant time. But if we need to delete the root, then deleting the minimum element is done by destroying hollow roots and then doing links to reduce the number of roots to at most log N nodes. (Theorem 2 in Hollow paper: "The maximum rank of a node in a hollow heap of N nodes is at most $\log_\varphi N$"). So the upper bound for both delete_min() and delete() is O(logn).

List of resources:

The only source used is the Hollow Heaps paper: https://arxiv.org/abs/1510.06535#:~:text=Hollow%20heaps%20combine%20two%20novel,the%20data%20structure%20its%20name.

Instructions for compiling and running:

The implementation of the hollow heap is contained in the hollow.py, three tests and one user_portal file is included in the project. For each of them, simply type the commands:

python correctness_test.py
python runtime_test.py
python exception_test.py
python user_portal.py

All the above files import the hollow class from hollow.py, so there is no need to run hollow.py, above files will demonstrate the utility and correctness of hollow heap.

Tests included:

correctness_test.py:

The file test each function included in the hollow heap one by one. It generates many random integers as keys, then compares the result of each function operation to the corresponding operation achieved by the default python list. The same execution is repeated thousands of times with different random keys. If there are mismatches occur, the program prints "Failure" to indicate there are mistakes in hollow implementation.

It also includes a comprehensive test that combines all operations together, the test use randomly generated integers as keys and values, and the correctness is indicated by applying the same operation on the python standard list.

runtime_test.py:

The file measures the running time of three operations, insert, delete_min and decrease_key. The test execution thousand times the same hollow heap operation repeatedly, and record its running time. At the same time, do the same repetitive operations on heapq standard library. Then print out the running times of the two structures respectively.

exception_test.py

The file demonstrates the ability of my implementation to handle exceptions. Different exceptional cases considered in my coding process are shown clearly.

user_portal.py

The file provided the user a portal to directly use my hollow heap, as the file doesn't consider many exceptions, users have to follow the printed guideline strictly. All operations are supported in the user_portal except the meld function.

Assumptions:

The keys must be numeric.

All_files:

–hollow

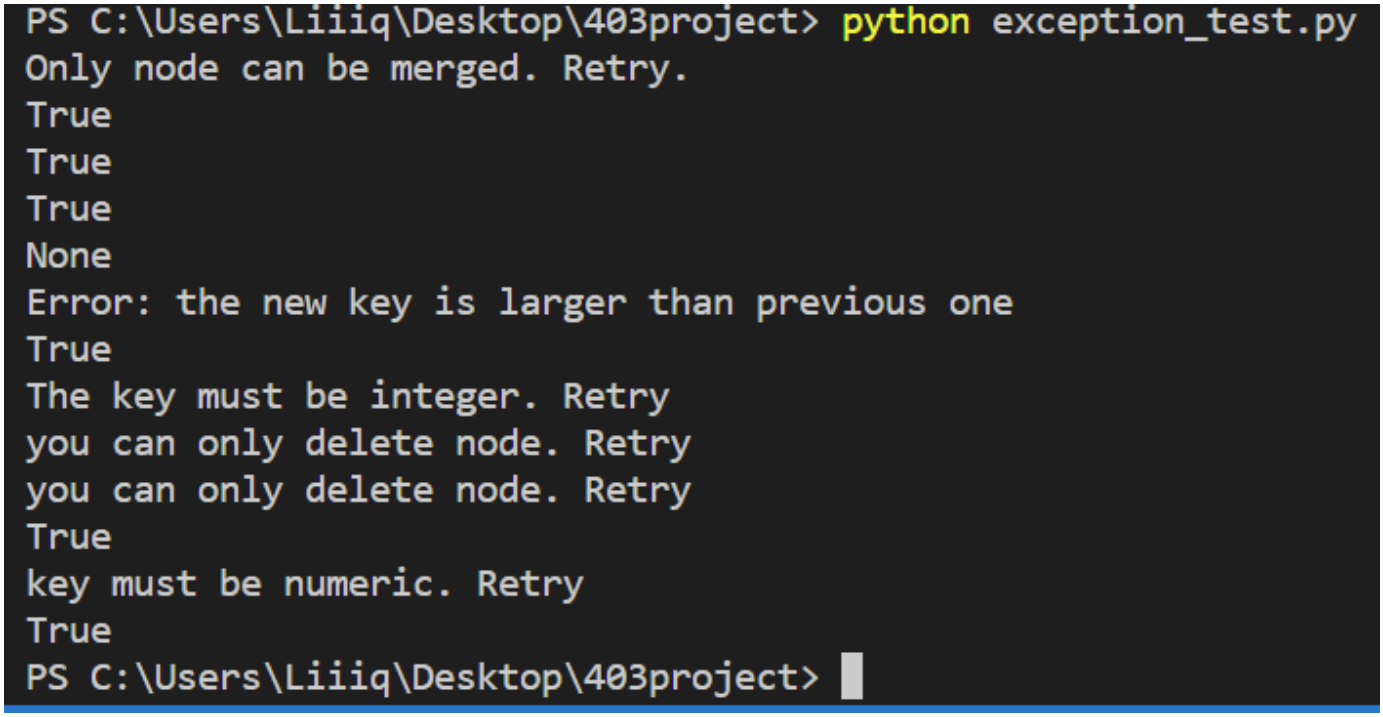
- – hollow.py
- – correctness_test.py
- – runtime_test.py
- – exception_test.py
- – user_portal.py
- – README.pdf

Explanation of result:

Below is the result of runtime_test. It is clear that the running times of the three operations are printed separately.

```
PS C:\Users\Liiiq\Desktop\403project> python runtime_test.py

.........Insertion time comparsion...........

Binary insertion: Elapsed 261555.100 micro secs.
Hollow insertion: Elapsed 3623945.300 micro secs.

.................End comparsion.................


.........delete_min time comparsion...........

Binary delete_min: Elapsed 434571.100 micro secs.
Hollow delete_min: Elapsed 10868014.500 micro secs.

.................End comparsion.................


.........decrease_key time comparsion...........

Binary decrease_key: Elapsed 1916.100 micro secs.
Hollow decrease_key: Elapsed 3629.000 micro secs.

.................End comparsion.................

PS C:\Users\Liiiq\Desktop\403project>
```

Below is the result of exception_test. A True indicates the test case passed the assertion, and other printed strings indicate different exceptional scenarios. I explained all cases in the code comments clearly.



Below is the result of correctness_test, if there is some mistake occurs, there will be a “Failure” printout. For example, the “Do Insert Test” line indicates that the insert_test function is running, when there is no “Failure” printed before the “End Test” line, there is no mistake in this part of code implementation.

```
PS C:\Users\Liiiq\Desktop\403project> python correctness_test.py

.........Do Insert Test..........
............End Test..............

.........Do Insert pair Test..........
............End Test..............

.........Do Find_Min Test..........
............End Test..............

.........Do Delete_Min Test..........
............End Test..............

.........Do Delete Test..........
............End Test..............

.........Do Decrease Key Test..........
............End Test..............

.........Do Meld Test..........
............End Test..............


########## DO COMPREHENSIVE TEST ############

......doing insertion.....

......doing find min & delete.....

......doing meld.....

......doing decrease key test.....

......doing delete min test.....

############ END COMPREHENSIVE TEST ############

PS C:\Users\Liiiq\Desktop\403project>
```