Programming Assignment 4 (100 points)

Due April 14, 2021

Objectives:

- Implement the Minimum Priority Queue (MPQ) ADT based on three different data structures: vector, linked list, and binary heap.
- Use OOP by extending the parent class `MPQ<T>` with three child classes.
- Use generic programming to allow the use of any data type with an overloaded < operator.
- Test your implementations on large dataset.

1. **(30 points) Vector and Linked List MPQ**

   Implement the minimum priority queue `UnsortedMPQ<T>` and `SortedMPQ<T>` that are child classes of the provided `MPQ<T>` class. The functions from MPQ<T> that are virtual functions (`remove_min()`, `is_empty()`, `min()`, and `insert()`) must be implemented in the child classes. The functions `remove_min()` and `min()` should throw an exception if the minimum priority queue is empty.

   For the `SortedMPQ<T>` class, you will use a **linked list** to implement the minimum priority queue functions. The `insert()` function should be $O(n)$ and the `remove_min()` function should be $O(1)$.

   For the `UnsortedMPQ<T>` class, you will use a **vector** to implement the minimum priority queue functions. The `insert()` function should be $O(1)$ and the `remove_min()` function should be $O(n)$.

   Note: The use of **STL vector and linked list (`list`) are allowed** as well as work done in the previous assignments.

2. **(20 points) Binary Heap MPQ**

   For this section, first implement a binary heap class based on Chapter 6 and the lecture slides on "Binary Heap" and "Priority Queue" named `BinaryHeap<T>`. You can use the STL vector to implement the binary heap data structure. This binary heap should then be used to make `BinaryHeapMPQ<T>` similar to the previous part but using the binary heap as the underlying data structure. We have provided several private heap functions to help you implement the `Heap<T>` class.

3. **(30 points) MPQ Application and Stress Test**

   Once you have completed your three MPQ implementations, we have provided a main file (`main.cpp`) for running simulations and testing the efficiency of your code. The application is based on a CPU job scheduler that needs to order the execution of processes being run on a computer.

   (a) You will use the `CPU_Job` struct to represent a computer process which is found in `cpu-job.h`.

   (b) The jobs to run are read from an input file (in the directory `InputFiles`) where each line represents a job. The format of each line is 3 integers: *job ID*, *length*, *priority* (*job ID* numbers are unique).

   (c) A job priority is represented by an integer from -20 to 19 where lower numbers are prioritized allowing you to use a minimum priority queue. Jobs are ordered by priority then by length (lower before higher) while ties are broken with job IDs.

   (d) The output format for a `CPU_Job` should be as follows.
   `Job 382 with length 3 and priority -7`

   (e) Test your implementations with the given main file on the input files we provide and submit to Mimir for the stress test (sample size 1,000, 10,000, and 100,000). You can also create your own test files.

4. **(20 points) Report**

   Answer the questions in the file "CSCE221_Report_Instructions." (You may use .lyx or .tex or .pdf formats) available in "report" folder of this assignment.

5. **Submission Instructions**

   (a) You should **only** submit `MPQ.h`,`SortedMPQ.h`, `UnsortedMPQ.h`, `BinaryHeapMPQ.h`, `BinaryHeap.h`, **and** `cpu-job.h` **to Mimir.** (which implies that these are the only files you need to complete. MPQ.h may not need any modifications.)

   (b) Compile your code with the provided makefile.

   (c) Test you program on sizes up to 1000 **BEFORE SUBMITTING TO MIMIR**. The timing simulation should take less than a second; however, the full simulation could take significantly longer since you are printing data to a file.

   (d) Submit your **report to Canvas.**

6. **Recommendations**

   (a) You can complete your assignment section wise in the following order.

      i. Take a look at `MPQ.h`. It may not need any modifications but those are the operations you will be implementing in **ii., iii., and v**.

      ii. Complete `UnsortedMPQ.h` and test it with `unsortedmpq-main.cpp`

      iii. Complete `SortedMPQ.h` and test it with `sortedmpq-main.cpp`

      iv. Complete `BinaryHeap.h`. Some functions are already implemented. Implement `insert`, `is_empty`, `min`, `remove_min`, `up_heap`, `down_heap` functions. Also please note that `BinaryHeap.h` does not inherit `MPQ.h`

      v. Complete `BinaryHeapMPQ.h` and test it with `binaryheap-mpq-main.cpp`. `BinaryHeapMPQ.h` can call functions in `BinaryHeap.h`. Hence, complete iv before v.

      vi. Complete `cpu-job.h` and test it with `cpu-job-main.cpp`

      vii. Test using `main.cpp`. All the above mentioned files need to be working properly.

   (b) All the functions that need to be implemented are direct concepts. You can refer the slides in module 10.

   (c) `makefile` need not be modified.