Programming Assignment 4

Minimum Priority Queue

Objectives

In this assignment you are asked to implement three variants of the Minimum Priority Queue (MPQ) data structure: vector, linked list, and binary heap.

This assignment also explores Object Oriented Programming (OOP) design patterns and continues to explore uses of templates.

Report & Turn In

There is no written report for this assignment. There will be an oral report over the contents of this assignment in lab the week following the due date.

`!IMPORTANT` By submitting code to Mimir and/or Canvas you acknowledge and are bound by the Aggie Honor Code:

> *On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work*
>
> *An Aggie does not lie, cheat, or steal, or tolerate those who do.*

Your submission will be taken in place of a digital/physical signature.

Turn in your code to Mimir. You should submit the following files:

- `MPQ.h` , `SortedMPQ.h` , `UnsortedMPQ.h` , `BinaryHeapMPQ.h` , `BinaryHeap.h` , and `cpu-job.h`

Please also submit a zipped or tarball version of these files to Canvas.

Provided Materials

Starter code is provided on Github: `https://github.tamu.edu/csce221/pa4`

Additionally, sample test, main, and make files are provided. Part of this assignment is a `100,000` element stress test. Writing efficient code and testing locally will save you time over repeatedly uploading to Mimir.

The Minimum Priority Queue ADT

The minimum priority queue consists of four functions:

- `T remove_min()` - remove and return the minimum element
- `T min()` - return, without removing, the minimum element
- `bool is_empty()` - return *true* iff the queue is empty
- `void insert(const T& data)` - insert *data* into queue. Throw any exception if the queue is empty.

The file `MPQ.h` contains the function signatures for these functions. A couple of notes:

- These functions are `virtual` . This means that calling them from a super class will cause the one in the derived class to be called instead; even if the object is cast into another type.

- These functions are `pure virtual` (the part where `= 0`). This means that derived classes *must* implement these functions. Additionally, it is not possible to instantiate any class with a `pure virtual` function. Specifically, the constructors are `deleted` .

You should not need to edit the `MPQ.h` file, but if needed you can.

Unsorted MPQ

Implement an Unsorted MPQ built on top of *std::vector* (or alternatively collection). `insert()` should be `O(1)` and `remove_min()` should be `O(n)` .

Implement the four functions from the MPQ ADT for the Unsorted MPQ. Starter code is provided for you. Feel free to add any additional data members or methods as you deem necessary, probably an instance of *std::vector*. You may not need to implement a default constructor, but feel free to do so if necessary.

Sorted MPQ

Implement an Sorted MPQ built on top of *std::list* (or alternatively any other list implementation). `insert()` should be `O(n)` and `remove_min()` should be `O(1)` .

Implement the four functions from the MPQ ADT for the Sorted MPQ. Starter code is provided for you. Feel free to add any additional data members or methods as you deem necessary, probably an instance of *std::vector*. You may not need to implement a default constructor, but feel free to do so if necessary.

Binary Heap MPQ

Implement a MPQ built on top of a binary heap.

But first implement a binary heap. The binary heap is a form of binary tree with two properties.

- The binary tree is *proper*. There are no holes. The height is minimized and nodes are shifted as far left.
- The value at a node is less than the value of either child node. Nodes can be promoted (`upheap`) and demoted (`downheap`) to maintain this property.

> *Note: this is for a minimizing heap. There are also maximizing heaps, but we will not use those in this assignment.*

This second property gives a nice feature: the tree can be implemented on top of an array (or more realistically a *std::vector*). The starter code in `BinaryHeap.h` contains the formulas for getting the index of the left and right child from the index of the parent.

Once you complete the Binary Heap, implement a MPQ based on a binary heap. Both `insert()` and `remove_min()` should be `O(logn)` time (except where the underlying vector reallocates).

CPU Job (MPQ Applied)

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.

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

- 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.
- The output format for a CPU Job should be as follows: `Job 382 with length 3 and priority -7` where the integers represent job id, length, and priority respectively.
- 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.

IMPORTANT: Please do not create a `cpu-job.cpp` . Instead, implement all the functions inside the `cpu_job.h` file. This is generally bad practice, but for this assignment will be required to make Mimir work.

Hints

- Complete `UnsortedMPQ` , test with `unsortedmpq-main`
- Complete `SortedMPQ` , test with `sortedmpq-main`
- Complete `BinaryHeapMPQ` , test with `mpq-main` . Complete `BinaryHeap` before attempting `BinaryHeapMPQ`
- `BinaryHeap` does not inherit `MPQ` . `BinaryHeapMPQ` does inherit `MPQ`

Default Constructor Inference In C++ (>=11), if you do not define a default constructor, a default constructor will be generated which default constructs each data member in the order they are declared. For example, a class `A` containing a `std::vector v` will have a default constructor equivalent to: `A(void) : v() {};` This behavior is cancelled if

- Any data member cannot be default constructed
- The default constructor is explicitly declared: `A(void): v(100) {}`
- The default constructor is explicitly deleted: `A(void) = delete;`
- Any other non-default constructor is provided (ex `A(int i) ...`). In this case, the default constructor can be reenabled with `A(void) = default;`

For the above reason, you should only need to implement the four MPQ functions for each of the MPQ classes. You may implement constructor(s) as you see fit, but they are not necessary for this assignment.