

CMSC 341 - Project 3: Course Registration Algorithm - Spring 2021

Due: Tuesday, April 20, before 9:00 pm

- Please ***do not rename the files***. Linux file names are case sensitive.
- Please ***do not develop code*** in the shared directory.

Addenda

- 04/15/21 - Clarification: The member variable "Student::_priority" is not to be used for determining the priorities in the heap. It is a member variable in the Student class like other member variables which can be used for calculating the priorities. The priorities are to be determined by the calculation in different priorities functions. To determine the priority for every node we need to use the current priority function which is set in "RQueue::priority" member variable.

Objectives

- Practice constructing and using heap data structure as a priority queue ADT
- Practice writing merge operation in skew heap data structure
- Gain additional experience constructing and using binary trees
- Practice using recursion in programs
- Learn to use function pointers

Introduction

The department of Computer Science in Neptune University has limited resources in course offering. The department is searching for a new algorithm which distributes the resources more efficiently. A task is assigned to you to implement a proposed intelligent algorithm for the purpose of simulating and testing the algorithm. This intelligent algorithm automatically registers students in a course based on a priority value for every student. The students will be registered in the order of priority values. The priority for a student is defined based on some criteria. We can change the algorithm which determines the priority value. For example, the priority values can be overwritten by an officer in registrar's office. The algorithm to determine the priority will be implemented in a function. Your job is to implement a skew heap data structure that can take different priority functions. Such an architecture allows us to use different priority functions and compare the results.

Note: This is a fictional scenario.

Skew Heap

A skew heap is a specialized version of a heap data structure which performs the insertion and deletion operations in $O(\log n)$ amortized time. This data structure is a binary tree in which the root always holds the node with the highest priority. Skew heap uses merge operation to perform insertion and deletion.

The major operations supported by a min-skew-heap are insertion of elements, reading the highest priority element, and removing the highest priority element. Reading the highest priority element is just a matter of reading the root node of the heap. The other two operations, insertion and removal, are applications of the merge function:

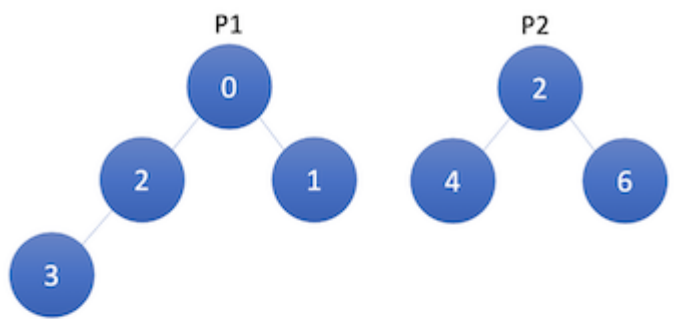
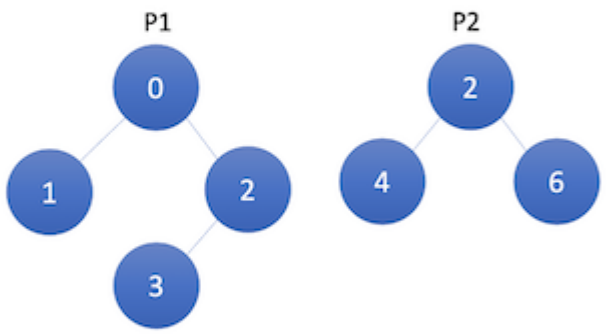
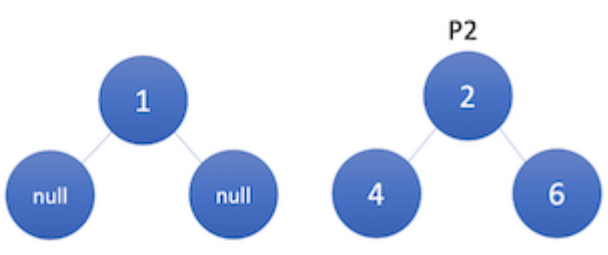
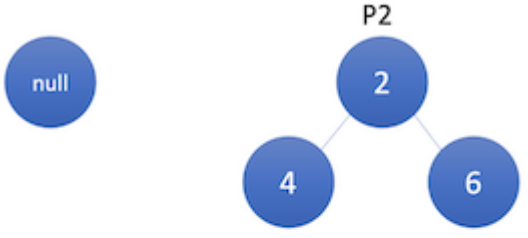
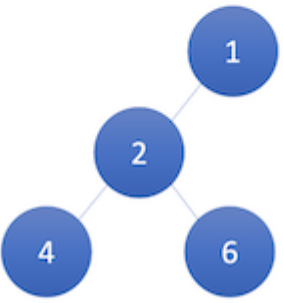
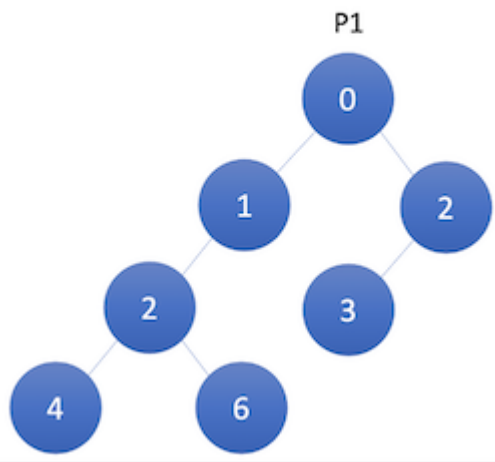
- To insert a new node x into an existing skew heap H , we treat x as a single-node skew heap and merge it with H .
- To remove the node with highest priority value, we delete the root node and then merge the root's left and right sub-heaps.

We see, then, that the merge function is key to all of the major skew heap operations. If we can implement merge correctly, insertion and removal are simple.

The special feature of a skew heap is the merge operation which combines two skew heaps into a single, valid skew heap. Let $p1$ and $p2$ be positions in two skew heaps (e.g. pointers to nodes). The merge operation is defined recursively:

- If $p1$ is Null, return $p2$; similarly, if $p2$ is Null, return $p1$.
- Assume that $p1$ has higher priority than $p2$; if not, swap, $p1$ and $p2$.
- Swap the left and right subtrees of $p1$.
- Recursively merge $p2$ and the left subtree of $p1$, replacing the left subtree of $p1$ with the result of the recursive merge.

The following figure shows two skew heaps and the result of merging the two.

| | |
|---|---|
|  | P1 has a higher priority |
|  | Swap children of P1 |
|  | Merge P2 with left child of P1 Left child of P1 has a higher priority Swap its children |
|  | Left child of P1 left child is null The merge result is P2 |
|  | P2 becomes the left child of P1 left child |
|  | The merge result becomes the left child of P1 |

Assignment

In this project, you will implement a queue class (RQueue) based on a min-skew-heap data structure; it will maintain a min-skew-heap based on the computed priority for each student, where the priority function is provided to the RQueue constructor via a function pointer. Inserting to and extracting from the skew heap uses a simple skew heap merge function which guarantees that the min-heap property is maintained; the comparisons that are part of the merge process will be made on the computed priorities for students in the skew heap. The class allows for the priority function to be changed; in which case the skew heap must be rebuilt.

For this project, you are provided with the following files:

- [rqueue.h](#) – The interface for Student, Node, and RQueue classes.
- [rqueue.cpp](#) – A skeleton for the implementation of the class RQueue
- [driver.cpp](#) – A sample driver program. It contains sample use of RQueue class. There is no test case in this sample driver program.
- [driver.txt](#) – A sample output produced by driver.cpp

Specifications

There are three classes in this project. The class RQueue has a memeber variable of type Node, and the class Node has a member variable of type Student.

Class Student

This class stores the information about a student. The information represents the status of student in the school such as what year the student is, or what the major of the student is. You are not allowed to modify this class. The information is this class can be used to calculate a priority value for the course registration using a priority function. This class provides an alternative constructor that allows for setting all required information. It also provides the required public functions to obtain the information from an object.

Priority Functions

A priority function is a user defined function that can use the information in the class Student to determine a priority value for a student. Two example functions are provided in driver.cpp file. You can define your own functions. The ability of using different functions allows us to adapt to different priority algorithms. The skew heap can accept a prioritization function. This is accomplished by passing a function pointer to the constructor. The function pointer is the address of a function that will be used to compute the priority for the course registration. The function must take a Student object as input and return an integer priority value between 0 and 6. A typedef for the function pointer is provided for you in rqueue.h:

```
typedef int (*prifn_t)(const Student&);
```

This says that prifn_t is a pointer to a function that takes a Student& argument.

| | |
|---|---|
| int priorityFn1(const Student &student); | This function determines a priority value for the course registration for the student and returns the priority value. The algorithm in this function uses the information in the Student class. This algorithm assigns a number between 0 to 6. The lower value means the higher priority for the course enrollement. Note: The implementation of this function is provided to you. You do not need to modify it. |
| int priorityFn2(const Student &student); | This function returns the priority value that is assigned by an officer in the registrar's office. Note: The implementation of this function is provided to you. You do not need to modify it. |

Class Node

This class constructs a node in the heap data structure. It has a member variable of the type Student, i.e. _student. The member variable is initialized throught the Node alternative constructor. The class RQueue is a friend of Node class, it means it has access to private members of Node class. You are not allowed to modify this class.

Overloaded Insertion Functions

There are two overloaded insertion functions for the classes Student and Node to help debugging your project. The implementation is provided to you. You do not need to modify them.

Class RQueue

The class PQueue constructs a min-skew-heap data structure. This class has a member variable called _heap. The member variable _heap presents the root node of the heap data structure and it is of the type Node. The following table presents the list of member functions that need implementation.

| | |
|---|---|
| RQueue::RQueue(prifn_t priFn) | This is the constructor for the RQueue class. It must be provided with a pointer to the prioritization function. |
| RQueue::~~RQueue() | Destructor, all dynamically-allocated data must be deallocated. |
| RQueue::RQueue(const RQueue& rhs) | Copy constructor, must make a deep copy of the rhs object. It must function correctly if rhs is empty. |
| RQueue& RQueue::operator=(const RQueue& rhs) | Assignment operator, remember to check for self-assignment and to free all dynamically allocated data members of the host. You should not call the copy constructor in the implementation of the assignment operator. |
| void RQueue::insertStudent(const Student& input) | Insert a Student into the queue. Must maintain the min-heap property. |

| | |
|---|---|
| Student RQueue::getNextStudent() | Extract (remove the node) and return the highest priority student from the queue. Must maintain the min-heap property. Should throw a domain_error exception if the queue is empty when the function is called. |
| void RQueue::mergeWithQueue(RQueue& rhs) | Merge the host queue with the rhs; leaves rhs empty. Two skew heaps can only be merged if they have the same priority function. If the user attempts to merge queues with different priority functions, a domain_error exception should be thrown. This function requires protection against self-merging. Merging a heap with itself is not allowed. |
| void RQueue::clear() | Clear the queue. Must delete all the nodes in the skew heap, leaving the skew heap empty. |
| int RQueue::numStudents() const | Return the current number of students in the queue. |
| void RQueue::printStudentQueue() const | Print the contents of the queue using preorder traversal. Although the first student printed should have the highest priority, the remaining students will not necessarily be in priority order. Please refer to the sample output file (driver.txt) for the format of this function's output. |
| prifn_t RQueue::getPriorityFn() const | Get the current priority function. |
| void RQueue::setPriorityFn(prifn_t priFn) | Set a new priority function. It must rebuild the heap! |

Additional Requirements

- Private helper functions must be declared in rqueue.h. No other modifications to rqueue.h are permitted!
- You must use a min-skew-heap data structure to store the queue. The skew heap must be ordered according to the prioritization function priority declared in rqueue.h and set by the constructor or setter method. If the prioritization function is changed using the setter method, then the min-heap must be rebuilt using the new prioritization function.
- Computed priority values may not be pre-computed and stored with the student in the queue. They must be computed as needed using the priority function.
- Insertion to and extraction from the min-heap must run in amortized logarithmic time.
- You must use the provided operator<< to output Student and Node objects.

Testing

You must write your own, extensive test driver called mytest.cpp; this file contains your Tester class, all test functions, all test cases, priority functions, and the main function. It must compile with your RQueue class and it must run to completion.

Your driver file must be called **mytest.cpp**; no other name is accepted for the driver program.

Testing RQueue class

Your driver program should test at least the following properties of your RQueue implementation:

- Basic correctness. Creating a queue, inserting students, and reading them out in priority order functions correctly.
- Correctness of copy constructor, assignment operator for normal and edge cases.
- Can change the prioritization function using the setter method and the min-heap is rebuilt correctly. The following algorithm is an example for such a test:
 - Build a heap with a priority function and some data points; data points should have proper information so their priority change once the priority function is changed,
 - Change the priority function; it must rebuild the heap with the same data points,
 - Check whether the rebuilt heap satisfies the priority queue property.
- Efficiency with large queues. Insertion and extraction with a large queue operates in logarithmic time, i.e. $O(\log n)$
- A domain_error exception is thrown If the user attempts to merge queues with different priority functions.
- Attempting to dequeue an empty queue throws a domain_error exception.

Memory leaks and errors

- Run your test program in valgrind; check that there are no memory leaks or errors.
Note: If valgrind finds memory errors, compile your code with the -g option to enable debugging support and then re-run valgrind with the -s and --track-origins=yes options. valgrind will show you the lines numbers where the errors are detected and can usually tell you which line is causing the error.
- Never ignore warnings. They are a major source of errors in a program.

What to Submit

- Please ***do not create sub-directories*** in the shared directory.
- Please ***clean up the shared directory*** after testing. Only the required files for project will remain in the shared directory. Development files such as precompiled gch, object files, your executable files, or vgcore dump files are large;these files consume disk quota in submission folders.

You must submit the following files to the proj3 directory.

- rqueue.h
- rqueue.cpp
- mytest.cpp (This file contains your Tester class, all test functions, all test cases, priority functions, and the main function.)

If you followed the instructions in the [Project Submission](#) page to set up your directories, you can submit your code using the following command:

```
cp rqueue.h rqueue.cpp mytest.cpp ~/cs341proj/proj3/
```

Grading Rubric

The following presents a course rubric. It shows how a submitted project might lose points.

- Conforming to coding standards make about 10% of the grade.
- Correctness and completeness of your test cases (mytest.cpp) make about 15% of the grade.
- Passing tests make about 30% of the grade.

If the submitted project is in a state that receives the deduction for all above items, it will be graded for efforts. The grade will depend on the required efforts to complete such a work.