

CMSC 341 - Project 3: Manufacturing Scheduler - Fall 2021

Due: Tuesday, November 16, before 9:00 pm

Addenda

- 10/30/2021 - modifications: The file mqueue.h has been modified to reflect correct values for HIGHESTPRIO and LOWESTPRIO constants. Please download the latest version from the link on this page.

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 current pandemic situation caused many disruptions in the world. Manufacturing plants are late on fulfilling their orders. There are different reasons for this, for example, they do not receive the raw materials on time. Some plants decided to prioritize their production. You are assigned a task to implement a scheduler application that can be used by production managers. The manufacturing plant makes electronic chips for auto industry.

The application automatically prioritizes the production of orders. The prioritization is based on some specific criteria. The algorithm that uses the criteria to calculate the priorities can change. Therefore, the application should have the possibility of changing the order priorities. For example, the production manager may decide to overwrite the calculated priority for an order. An algorithm that can determine the priority will be implemented in a function. Your job is to implement a skew heap data structure that takes different priority functions. Such an architecture allows us to use different priority functions.

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:

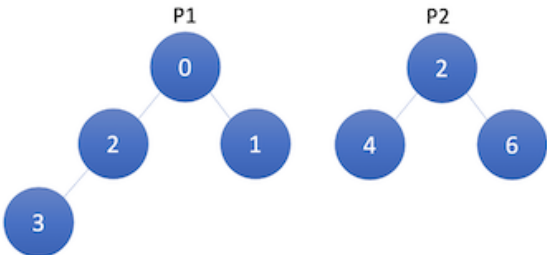
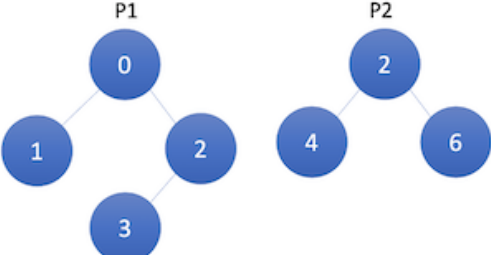
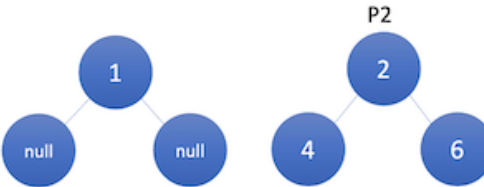
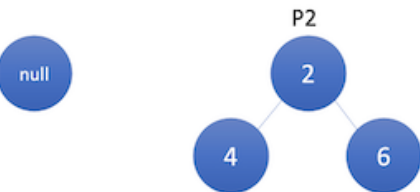
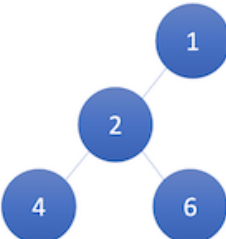
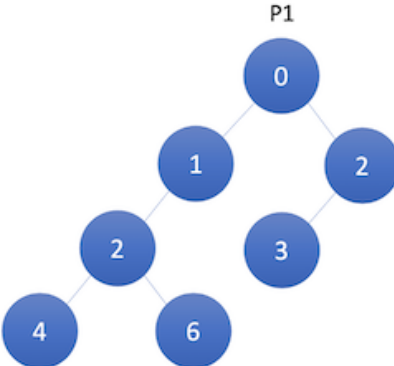
- To insert a new node x into an existing skew heap H , we treat x as a single-node skew heap and merge it with H .
- To remove the node with highest priority value, we delete the root node and then merge the root's left and right sub-heaps.

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 p_1 and p_2 be positions in two skew heaps (e.g. pointers to nodes). The merge operation is defined recursively:

- If p_1 is Null, return p_2 ; similarly, if p_2 is Null, return p_1 .
- Assume that p_1 has higher priority than p_2 ; if not, swap, p_1 and p_2 .
- Swap the left and right subtrees of p_1 .
- Recursively merge p_2 and the left subtree of p_1 , replacing the left subtree of p_1 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 (MQueue) based on a min-skew-heap data structure; it will maintain a min-skew-heap based on the computed priority for each incoming order, where the priority function is provided to the MQueue 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 the orders 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:

- [mqueue.h](#) – The interface for Order, Node, and MQueue classes.
- [mqueue.cpp](#) – A skeleton for the implementation of the class MQueue
- [driver.cpp](#) – A sample driver program. It contains sample use of MQueue 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 MQueue has a memeber variable of type Node, and the class Node has a member variable of type Order.

Class Order

This class stores the following information about an order:

- m_customer stores the name of customer.

- m_quantity stores the number of items ordered by the customer. The values stored in this member variable are defined in the enum type Quantity.
- m_material stores the probablity of raw material availability. The values stored in this member variable are defined in the enum type Material.
- m_arrivalTime stores the time that a customer places an order. This variable can take a value between 1 and 50. It indicates the placement of order. Using this value we can see what customer placed the order earlier comparing to other customers. For example, the value 50 indicates that the customer was the last one to place an order.
- m_overWritten stores priority value which is set by the production manager.

Priority Functions

A priority function is a user defined function that can use the information in the class Order to determine a priority value for an order. 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 manufacturing. The function must take an Order object as input and return an integer priority value between 0 and 58. A typedef for the function pointer is provided for you in mqueue.h:

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

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

int priorityFn1(const Order &order);	This function determines a priority value for the manufacturing and returns the priority value. The algorithm in this function uses the information in the Order class. This algorithm assigns a number between 0 to 58. The lower value means the higher priority. Note: The implementation of this function is provided to you. You do not need to modify it.
int priorityFn2(const Order &order);	This function returns the priority value that is assigned by the production manager. 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 Order, i.e. m_order. The member variable is initialized throught the Node alternative constructor. The class MQueue 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 Order and Node to help debugging your project. The implementation is provided to you. You do not need to modify them.

Class MQueue

The class MQueue constructs a min-skew-heap data structure. This class has a member variable called m_heap. The member variable m_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.

MQueue::MQueue(prifn_t priFn)	This is the constructor for the MQueue class. It must be provided with a pointer to the prioritization function.
MQueue::~~MQueue()	Destructor, all dynamically-allocated data must be deallocated.
MQueue::MQueue(const MQueue& rhs)	Copy constructor, must make a deep copy of the rhs object. It must function correctly if rhs is empty. This function creates an exact same copy of rhs.
MQueue& MQueue::operator=(const MQueue& 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. This function creates an exact same copy of rhs.
void MQueue::insertOrder(const Order& input)	Insert an Order into the queue. Must maintain the min-heap property.
Order MQueue::getNextOrder()	Extract (remove the node) and return the highest priority order from the queue. Must maintain the min-heap property. Should

	throw a <code>domain_error</code> exception if the queue is empty when the function is called.
<code>void MQueue::mergeWithQueue(MQueue& rhs)</code>	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 <code>domain_error</code> exception should be thrown. This function requires protection against self-merging. Merging a heap with itself is not allowed.
<code>void MQueue::clear()</code>	Clear the queue. Must delete all the nodes in the skew heap, leaving the skew heap empty.
<code>int MQueue::numOrders() const</code>	Return the current number of orders in the queue.
<code>void MQueue::printOrderQueue() const</code>	Print the contents of the queue using preorder traversal. Although the first order printed should have the highest priority, the remaining orders will not necessarily be in priority order. Please refer to the sample output file (driver.txt) for the format of this function's output.
<code>prifn_t MQueue::getPriorityFn() const</code>	Get the current priority function.
<code>void MQueue::setPriorityFn(prifn_t priFn)</code>	Set a new priority function. It must rebuild the heap!

Additional Requirements

- Private helper functions must be declared in `mqueue.h`. No other modifications to `mqueue.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 `mqueue.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 order 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 `Order` 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 `MQueue` 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 MQueue class

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

- Basic correctness. Creating a queue, inserting orders, 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.

- mqueue.h
- mqueue.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 mqueue.h mqueue.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.