Middle East Technical University   Department of Computer Engineering

CENG 213

Ceng 213 Data Structures

Fall 2014-2015

Programming Assignment 3

Due date: 05.01.2015, 23:55

1 Objective

This homework aims to help you familiarize with the Priority Queue and Hash Table data structures.

Keywords: *Priority Queue (PQ), Dynamic Priority Queue (DPQ), Hashtable, Binary Heap*

2 Problem Definition

In this assignment, you will implement a new data structure DynamicPriorityQueue (DPQ) which is a mixture of Priority Queue and Hash Table. DPQ is a data structure that extends Priority Queue (PQ) so that each item on the queue is associated with both a Key and a Priority value. Priority values of the items in the DPQ can be referred with their keys and updated.

DPQ is useful in cases where

- Each item in the queue is associated with a unique key yet the items need to be sorted with respect to another attribute that is not unique among the items.
- The priority values of items on the queue need to be updated very frequently and the order of the queue changes as a result of the updates to priority values.

A real life example where DPQ structure can be used is a queuematic application. The queuematic application is used in a bank to determine the serving order of customers. Assume that the bank associates each customer with a priority value and serves the customers in order with respect to these priority values. In this case, a PQ would be appropriate. The customers will be kept in a PQ and the bank officer would call the customer with the highest priority on each serve session.

However, if there is a possibility that the priority of the customer changes while he/she is already waiting in the bank, this requirement cannot be met with the PQ interface because the priority values on a PQ cannot be updated. In this context, DPQ will be the appropriate data structure. DPQ both provides PQ interface and also enables access and updates to the priority values on the queue.

DPQ stores both Key and Priority for each item. Like the PQ interface, the client of DPQ can consume the items in order of Priority values one by one. DPQ provides the following access methods:

- **insert(Key,Priority)**: This method enables insertion of an item. The operation of this method is the same with the insert method of PQ except the insertion of *Key* of the item. The method has no effect on the items that are already on the queue and only inserts new items.

- **Key deleteMin()**: This method returns the Key of the item which has the smallest priority value. The operation of this method is the same with the deleteMin method of PQ except it returns Key instead of Priority value. You may think that the bank officer calls the customer with id instead of the priority value.
- **Priority getPriority(Key)**: This method enables key based access to the priority values on the queue. This method is similar to the *find* method of the Hashtable and contributes Hashtable behaviour to DPQ. A frequent use of this method is to check if an item is on the queue.
- **updatePriority(Key,Priority)**: This is the method that enables updates to the priority values of the items on the queue. The method re-orders the queue if the priority updates require. The method has no effect if the item is not on the queue.

3 Specifications

You are expected to implement the **DynamicPriorityQueue** class. **You are expected to use a hashtable and a binary heap to implement the DPQ interface.** These two data structures are mandatory. However, you may use additional data structures like arrays, vectors or lists if needed. There are no other design constraints.

Hashtable is necessary for access with key to the items on the queue and binary heap is necessary to realize the Priority Queue operations. Your DPQ implementation should maintain the two data structures in parallel in order to support all the interface methods. The hashtable should keep references to the items on the priority queue so that the priority value of an item can be accessed in constant time. You should also keep references to the keys from the PQ so that the references in the hashtable are maintained after operations that update PQ. Both structures should be updated whenever there is a change in ordering.

insert and *deleteMin* methods of DPQ has the same time complexity with the PQ methods of same name. Recall that with a binary heap implementation of PQ, *deleteMin* and *insert* methods require O(log N) time where N is the number of items on the queue. The *updatePriority* method may require a re-ordering like *deleteMin* or *insert* methods therefore it will have O(log N) worst case complexity with binary heap and hashtable implementation. *getPriority* method provides constant time access like the *find* method of Hashtable data structure.

You will be provided with

- *DynamicPriorityQueue.h* that includes the interface of DPQ class. Examine the file carefully and make sure that you stick to notes which guides you about where you can insert your code.
- *Hashtable.h* that includes the interface and implementation of Hash Table data structure

You are expected to submit

- an updated *DynamicPriorityQueue.h* file that includes your implementation of DPQ interface. Make sure that you do not update method signatures in the original interface file. You are expected to add only private members or methods to the header file.

The DPQ interface is given below. Note that it is a template class based on *Key* and a *Priority*. The const variable KEY-NOT-FOUND specifies the value to be returned when a key is not found on the queue. The other const variable QUEUE-EMPTY is the value to be returned by the *deleteMin* method when the queue is empty.

In case of equal priority values, the order in which the items are comsumed by deleteMin method is not important.

```
template <class Key, class Priority>
class DynamicPriorityQueue
{
public:

    DynamicPriorityQueue(Priority notFound,Key queueEmpty,int capacity)
      :KEY_NOT_FOUND(notFound),
        QUEUE_EMPTY(queueEmpty),
            capacity(capacity){}


    void insert( const Key& Key, const Priority& priority);
    const Key deleteMin( );

    void updatePriority(const Key& Key, const Priority& priority);
    const Priority& getPriority(const Key& Key) const;

private:
    int capacity;
    const Priority KEY_NOT_FOUND;
    const Key QUEUE_EMPTY;
};
```

4 Regulations

1. **Programming Language**: You should code your program in C++ language. **Your submission will be compiled and evaluated on department's inek machines.** You are expected to make sure that your code compiles with g++ correctly.

2. **Late Submission**: You have a total of 7 days for late submission of the assignments. However, one can use at most 3 late days for any assignment. No penalty will be incurred for submitting within late 3 days. Your assignment will not be accepted if you submit more than 3 days late or you used up all 7 late days.

3. **Cheating**: We have zero tolerance policy for cheating. Sharing and copying any piece of code from each other and Internet is strictly forbidden. People involved in cheating will be punished according to the university regulations.

4. **Newsgroup**: You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis. Please ask your questions related to the homework on the newsgroup instead of sending e-mails.

5. **Evaluation**: Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.

6. **Not Graded**: Below are some situations that your homework will not be graded (automatically assigned zero): (c) Implementing DPQ class without HashTable. (d) Implementing DPQ class without a binary heap.

5 Submission

- Submission will be done via COW.
- Submit a tar.gz file named **hw3.tar.gz** that contains only **DynamicPriorityQueue.h**.

- *DynamicPriorityQueue.h* submitted by you will be placed under a directory that includes the original *HashTable.h* file that we provided. Besides these files, a driver file like *DPQTest.cpp* will be placed under this directory. Then, the following command sequence is executed to run your solution:

```
$ g++ DPQTest.cpp -o dpqTest
$ ./dpqTest
```