Andrew Taylor

Atayl136

Homework 4 – Queues and Lists

1. Develop an ADT specification for a priority queue. A priority queue is like a FIFO queue except that items are ordered by some priority setting instead of time. In fact, you may think of a FIFO queue as a priority queue in which the time stamp is used to define priority.

A priority queue is a type of queue, so we will start with the ADT as presented in class, with some modifications, and then add some optional methods. It is assumed items cycle from low priority to high priority (but possibly be defined vice-versa) or else they would never leave the queue. Likewise, inserting or deleting an item of intermediate priority would be an operation that could be better suited for a List. These are considerations to take into account when understanding the clients requirements. However, since the structure is not FIFO it anticipates the situation where items come in and out of the queue both before and after each other, so random access can be exploited to find and delete intermediate items, quickly if in terms of their priority because the priority number can be directly accessed with no comparison/search needed. Because the question does not mention the requirement of data search and so on, I'm going to limit my ADT to a modification of the "anchor" method discussed in class, with deferred shifting. This is an O(n) approach with an additional field for priority. This was chosen because insertion is still O(1), it avoids false overflow, wrap-around wouldn't support intermediate insertion/deletion very well, and a linked implementation forces sequential access. It's the client applications job to determine which item gets deleted when, but with the anchor and shifting approach they can do it according to priority.

ADT Priority Queue

Data: An empty list of values with a reference to the last item in the queue as well as a field for a priority value, an integer. Queue anchored to the front.

Methods:

QEmpty

Input:	None
Precondition:	None
Process:	Checks if the queue contains data items
Postcondition:	None
Output:	Return 1 or True if queue is empty and 0 or False otherwise

QPriorityDelete

Input:	None
Precondition:	Queue contains meaningful data values
Process:	Remove the item with the specified priority by changing its priority to null.

Overwrite the data value to “some string of blanks” this must be chosen carefully.
Run a less than comparison rule on all priority fields in the queue, and for the first and all subsequent values that fail the test decrement that priority value by 1. Null is not compared.

Postcondition: The queue has one less data item in the correct spot, quickly checked by comparing the data field to “some string of blanks”
Length decremented by 1
Output: Return the deleted Item

QPriorityInsert

Input: A data item to be stored at a specified priority
The specified priority, default: lowest
(but possible drag and drop input)
Precondition: If Length<AllocationSize continue elif
resize if dynamic allocation
else call QShiftCopy if static allocation. (or garbage collection necessary eventually.)
Process: Store the item at the back of the queue using rear pointer
Store the items priority value in its field
Run a less than comparison rule on all priority fields in the queue, and for the first and all subsequent values that fail the test increment that priority value by 1. Null is not compared.
Length incremented by 1
Postcondition: The queue contains one additional item.
Output: None

QPeek

Input: None
Precondition: Queue contains meaningful data values
Process: Retrieve the value of the data item of the specified priority
Postcondition: The queue is unchanged
Output: Return the value of the data retrieved from the item specified

Optional Method

QShiftCopy

Input: Queue of Data values and priority values in place
Precondition: IsEmpty confirms the queue is not empty
Process: Another queue of equal size is created with priority numbers prefilled sequentially up to length. Starting from the anchor of the original queue the process compares the priority field to null and 1.

If null it moves to the next item. It compares each priority to the list of priority numbers up to sequential length starting with 1 and when equal copies the data value over to that item in the new queue with no gaps. When the incrementing comparison reaches length it completes the process (depending on indexing)

Postcondition: The new queue is not empty.

Output: None

Another option would be to rely on the garbage collection of Java or Python.

2. Write an algorithm to reverse a singly linked list, so that the last element become the first and so on. Do NOT use Deletion - rearrange the pointers.

Initialize the 4 pointers that usually come with a singly linked list: The List Pointer "L" pointing to the head, A temp or "T" pointer, the next "T.N" pointer at each node and at the tail is null.

1) Declare a variable called New.Next which stores a specific node pointer
2) Declare a variable called Old.Next which stores a specific node pointer
3) T starts at the head pointed out by L

```
        New.Next = T
        Old.Next = T.N
        T = T.N
        Old.Next = null
While T != null
        New.Next = T
        Old.Next = T.N
        T = T.N
        Old.Next = New.Next
```

This leaves the head in place but the last element is the first and the order is reversed, and it only uses two variables there is no need for new objects. The pointers have been rearranged with these variables. I think this works because you use the variables to modify the nodes (objects).

3. What is the average number of nodes accessed in search for a particular element in an unordered list? In an ordered list? In an unordered array? In an ordered array? Note that a list could be implemented as a linked structure or within an array.

The average number of nodes accessed in a search for a particular element in an unordered list, and ordered list, and an unordered array is n/2 in all three cases. In an ordered array, things are different because the array is sorted. It can be searched in less time, using for example binary search. The average number of nodes accessed in search for a particular element in an ordered array is lg n or (lg n)-1.

4. Write a routine to interchange the *m*th and *n*th elements of a singly-linked list. You may assume that the ranks m and n are passed in as parameters. Allow for all the ways that m and n can occur. You must rearrange the pointers, not simply swap the contents

```
Def Swap(List,m,n)

        MTH = Node.mth
        NTH = Node.nth
        MN = mthnext variable
        MP = mthprevious variable
        mCounter = 0
        nCounter = 0
        T = L
        For i in range (m-1)                    #loops and counters find m and n
                mCounter +1
                if  MTH == L                    # special handling of head case
                        MN = T.N
                        MP = null
                If mCounter = m-1         # saving variables to be used later
                         MP = T
                If mCounter = m
                        MN = T.N
                If T.N == null               # this logic increments temp single linked only
                        T = L                  the tail will be null
                else
                        T = T.N
        T = L
        For i in range (n-1)                    #loops and counters find m and n
                nCounter +1
                If nCounter = n-1
                        T.N = MTH
                elIf nCounter = n             # special handling of head and tail cases
                        if T.N == null
                                MTH.next = null      # moving over m
                        else
                                MTH.next = T.N
                If T.N == null                              # this logic increments temp single linked only
                        T = L                                       the tail will be null
                else
                        T = T.N
        NTH.next = MN                               # move nth over and finished
        If MP != null                               # move nth over and finished
                MP.Next = NTH
```