MTE 140 - Spring 2009. Project 3.

Trees: a Priority Queue and a Binary Search Tree.

Due Date: Friday Tuesday June 30th, 11:59pm.

Hanan Ayad

1 Project Description

In this project, you will implement two abstract data types. The first is a priority queue represented as a heap (where the array representation of a complete binary tree is used). The second is a binary search tree using a linked representation (it is **not** required for the binary search implemented in this project to be balanced). You are given the following source files:

1. The interface (header) file `priorityQ_asHeap.h`, which has the type definitions and the function declarations for the Priority Queue represented as a Binary Max Heap (discussed in class and presented in the textbook - see Sec 9.4 and 9.5).
2. An incomplete implementation file `priorityQ_asHeap.c`, for which you are required to implement the body of each of the functions, based on the heap representation (where the array representation of a complete binary tree is used).
3. The interface (header) file `binarySearchTree.h`, which has the type definitions and the function declarations for the Binary Search Tree.
4. An incomplete implementation file `binarySearchTree.c`, for which you are required to implement the body of each of the functions, based on the linked representation of binary trees.

2 Specifications of the Priority Queue Operations

The functions you will be implementing for the priority queue are described as follows:

1. `void PQ_initialize( PriorityQueue *PQ )`
   This function takes as an argument a pointer PQ to a PriorityQueue (which is a struct), and initializes its count field.
2. `int PQ_isEmpty( PriorityQueue *PQ )`
   This function takes as an argument a pointer PQ to a PriorityQueue and returns 1 if the queue is empty, 0 if it has one or more items, and -1 if PQ is `NULL` (as indication of error).
3. `int PQ_isFull( PriorityQueue *PQ )`
   This function takes as an argument a pointer PQ to a PriorityQueue and returns 1 if the queue is full, 0 if it has one or more empty positions, and -1 if PQ is `NULL` (as indication of error).

4. `int PQ_length( PriorityQueue *PQ )`
   This function takes as an argument a pointer PQ to a PriorityQueue and returns the number of items in the queue. It returns -1 if PQ is `NULL` (as indication of error).

5. `int PQ_enqueue( PQItem value, PriorityQueue *PQ )`
   This function takes as arguments a PQItem `value` and a pointer PQ to a PriorityQueue. It inserts the value into the queue, which requires a heapify-up process to restore the tree as a heap. It returns 1 if the item is enqueued, 0 if the queue is full, and -1 if PQ is `NULL` (where 0 and -1 indicate errors).

6. `int PQ_dequeue( PQItem *itemPtr, PriorityQueue *PQ )`
   This function takes as arguments a pointer `itemPtr` to a PQItem and a pointer PQ to a PriorityQueue. It removes the highest priority item (item with the largest value) and copies it into the value pointed to by `itemPtr`. This removal requires a heapify-down process to restore the tree as a heap. It returns 1 if the item is dequeued, 0 if the queue was empty before dequeueing, and -1 if any of its arguments is `NULL` (where 0 and -1 indicate errors).

7. `int PQ_findMax( PQItem *itemPtr, PriorityQueue *PQ )`
   This function takes as arguments a pointer `itemPtr` to a PQItem and a pointer PQ to a PriorityQueue. It copies the value of the highest priority item (item with maximum value) into the value pointed to by `itemPtr`. It returns 1 if the item is copied, 0 if the queue is empty, and -1 if any of its arguments is `NULL` (where 0 and -1 indicate errors).

8. `void PQ_print( PriorityQueue *PQ )`
   This function takes as an argument a pointer PQ to a Priority Queue and prints the queue items in the order in which they are stored in the array $[1:n]$, where n is the number of tree nodes.

3 Specifications of the Binary Search Tree Operations

In this project, the implementation of the binary search tree follows the recursive definition of trees; the binary tree is a either empty (`NULL`) or is a node that has left and right subtrees that are binary trees. Many of functions you will be implementing for the Binary Search Tree would be expressed recursively. It is assumed, in this project, that the values in the binary search tree are unique (no duplicate key values). The functions are described as follows:

NOTE: When declaring a variable of type SearchTree in `main`, set it first to `NULL` as an initialization to indicate that it is an empty tree.

1. `NodePtr tree_find( searchKey value, SearchTree T )`
   This function takes as arguments a searchKey value and a SearchTree T and returns a pointer to the node in the tree that has its key (data) equals to value. It returns `NULL` is no such node is found.

2. `NodePtr tree_findMin( SearchTree T )`
   This function takes as an argument a searchTree T and returns a pointer to the smallest item in the tree (which is the leftmost node). It returns `NULL` is T is `NULL`.

3. `NodePtr tree_findMax( SearchTree T )`
   This function takes as an argument a searchTree T and returns a pointer to the largest item in the tree (which is the rightmost node). It returns `NULL` is T is `NULL`.

4. `SearchTree tree_insert( searchKey value, SearchTree T )`
   This function takes as arguments a searchKey value and a searchTree T. It proceeds down the tree, searching for the proper position to insert a node with this value. If the value is found in the tree, the function doesn't insert, otherwise, it inserts a node with key equals to value at the last spot of the traversed path. It returns the modified searchTree.

5. `SearchTree tree_delete( searchKey value, SearchTree T )`
   This function takes as arguments a searchKey value and a searchTree T. It proceeds down the tree, searching for the node to be deleted. If the node is found, several cases should be considered (the node is a leaf, the node has one child, and the node has two children). It returns the modified searchTree.

6. `unsigned int tree_size( SearchTree T )`
   This function takes as an argument a searchTree T and returns the number of nodes in the tree. It returns 0 if T is `NULL`.

7. `int tree_depth( SearchTree T )`
   This function takes as an argument a searchTree T and returns the depth of the deepest node of the tree. It returns -1 if T is `NULL`.

8. `void tree_print( SearchTree T )`
   This function takes as an argument a SearchTree T and prints the nodes in increasing order of their key values. This can be obtained by performing an "inorder" traversal of the tree, where for each node, the function recurs left, print the node data, then recurs right.

4 Documentation and Programming Style

It is required that your name and ID appear at the top of all your submitted text files. Your code should be clear and understandable; you are required to use proper spacing and indentation to enhance the clarity of your code, use meaningful variable names, and useful comments. You should include comments for each function indicating its specifications. This includes a description of what it does, its parameters, return values, and any assumptions that it makes.

5 Submission Requirements

For the submission of this project, you are required to follow the steps outlined below:

1. Create a directory with the name `uwuserid_NNNNNNNNp3`, where `uwuserid` is your UW user ID, `NNNNNNNN` is your UW student ID number, and p3 denotes project 3.

2. Save all the sources files under this directory (do not create subdirectories). The files are: `priorityQ_asHeap.h`, `priorityQ_asHeap.c`, `binarySearchTree.h`, `binarySearchTree.c`.

3. Implement the project and generate all the object (.o) files. You are not required to sumit a `main` function for this project (but you will need to write one for yourself to test your functions).

4. Zip the directory into one .zip file with the same name. That is, `uwuserid_NNNNNNNNp3.zip`. The unzipping should result in creating a subdirectory under test with the name `uwuserid_NNNNNNNNp3`, with all the project files being extracted under this subdirectory.

5. Submit the file `uwuserid_NNNNNNNNp3.zip` into the dropbox designated for project 3 on UW-ACE (under MTE 140).