KIT205 Data Structures and Algorithms

Week 4 Tutorial

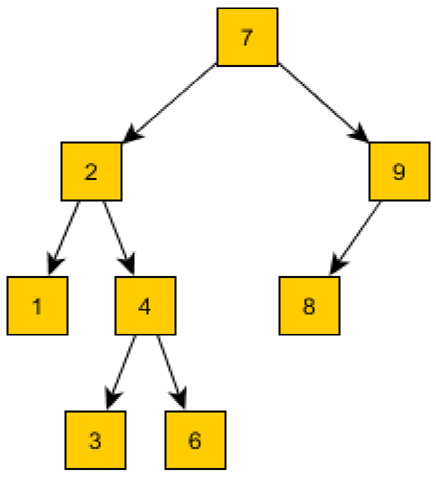
In addition to the work below, you will also be visited by the tutor to conduct a brief lab test based on previous tutorial work. You should have your list code from the first tutorial open and in a runnable state, so that you can quickly switch to that code when the tutor asks. You will be expected to demonstrate your code and answer simple questions to demonstrate your understanding.

Implementing a Binary Search Tree

In this tutorial you will be implementing and testing some BST code. Start by copying the relevant code from the lectures as follows:

1. Create a new C project in Visual Studio.
2. Copy the bst *struct* definitions from Lecture 3 into a new file called *bst.h*
3. Copy the *find_bst/find_bst_node* functions; and the *insert_bst/insert_bst_node* functions; the *min_node* function; and the *delete_bst/delete_bst_node* functions into a new file called *bst.c*
4. Add function prototypes (for the bst functions only – not the bst_node functions) for these functions to *bst.h*
5. Add a function called *new_bst* to create an empty bst.

Next we need some way of visualising the tree. We will only be using console output, so this will not be a particularly pretty representation of the tree (if you want better output, you could work on that later). Instead we are going to write one-line print functions that print a pre-order, in-order, or post-order traversal.



For the above tree we will use the following formats:

```
pre-order:   ( 7 ( 2 ( 1 __)( 4 ( 3 __)( 6 __)))( 9 ( 8 __)_))
in-order:    (((_ 1 _) 2 ((_ 3 _) 4 (_ 6 _))) 7 ((_ 8 _) 9 _))
post-order:  (((__ 1 )((__ 3 )(__ 6 ) 4 ) 2 )((__ 8 )_ 9 ) 7 )
```

The code for the pre-order print is:

```c
void print_pre_order_bst_node(BSTNodePtr self){
	if (self!= NULL){
		printf("(");
		printf(" %d ", self ->data_item);
		print_pre_order_bst_node (self ->left);
		print_pre_order_bst_node (self ->right);
		printf(")");
	} else {
	    printf("_");
	}
}

void print_pre_order_bst(BST *self) {
	print_pre_order_bst_node(self->root);
}
```

6. Add the code for pre-order print and then copy and modify it to get the in-order and post-order prints.
7. Next write a *main* method to build and print a tree as follows:

```c
int main(){
	BST tree = new_bst(0);
	int quit = 0;
	int data;
	while (quit == 0){
		printf("Enter some data: ");
		scanf("%d", &data);
		if (data != 0){
			insert(tree, data);
		} else {
			quit == 1;
		}
	}
	print_pre_order_bst(tree);
	printf("\n");
	print_in_order_bst(tree);
	printf("\n");
	print_post_order_bst(tree);
	printf("\n");
}
```

8. Experiment by building some trees and checking to make sure that you get the correct output.
9. Finally, write a (recursive) function to print the *height* of a tree