CSCI 2270 Lecture Notes

2/15/19

* Binary Search Tree Intro
  + special case of a binary tree
  + ordered structure
* For any node in the tree
  + all nodes in the left sub-tree of that node are less than that node value
  + all nodes in the right sub-tree of that node are greater that or equal to that node value
* Building a BST
  + if(newKey < currentKey){

go left

}else(>=){

go right

}

* + a balanced tree has better performance, although unbalanced are valid
  + Self-balancing trees exist
    - red black trees
    - AVL
    - …. many more
* Why does balanced trees matter?
  + Assume a perfect tree
    - the number of comparison operations to find a specific value will be significantly reduced compared to a similar search in something else such as an array
    - the worst case number of operations to find a value is the height of the tree (how many levels the tree has), where a similar search in something such as an array could be n, the number of elements in that data structure
* Node definition for BST
  + struct Node{

int key;

Node \*parent;

Node \*LC; // left child

Node \*RC; //right child

}

* BST ADT
  + private:

root

searchRecursive(node, value)

public:

init();

search(value)

insert(value)

disp() // ?

delete(value)

deleteTree() //destructor

* Insert Function
  + Given an existing tree, insert a node with a given key
    - Create new node and assign key value
    - Two pointers for traversal
      * temp = root
      * prev = null
    - Drill down to find next available empty spot
      * while(temp != nullptr){

prev = temp;

//check which way to traverse

if((n->key) < (temp)){

temp = temp->leftchild

}else{  
 temp = temp->rightchild

}

}