Raymond Chan
10/08/18
ECS 36c
Joël Porquet, PhD

Program #1: Binary tree succinct decoding

The program will read same file twice in order to prevent the use of containers. The first file will be reading the first line (structure line), while the second file will be reading the second line by skipping the first line (using getline to skip structure line, and access data line). Now the program could build the tree as it reads through the file.

The program will first construct root node if the first element of structure line is 1, otherwise (0) the tree should be empty. Then the program could run the recursive function. While running the recursive function, the program will first check if a node should be insert into the left of root node. If structure says 1, then make new left child node and call recursive function on this new left child node. If not, then move on to the right child node, do the same check, and call recursive function upon success.

The program should check if structure line is available (rather the file is empty or not); if data line is available (rather the file have a second line); if the datas are valid (are all datas having the same type, or that there are no shortage of data when building tree, or that there are no leftover datas after the tree finished building); if the structures are valid (all structures must be 0 or 1). Any datas or structures invalidation will cause corresponding files to close, and print out respective error.

Program #2: Binary search tree

Floor and ceil function both find their respective target by traversing the b_tree by deep (this means, if current key is larger than input key, the search will traverse to the left, else traverse to the right): the floor function will compare current key with its input key everytime it traverse through a node, and if current key is less than or equals to input key, that means it has the potential to “floor” the current key. Since every key after floor key should be greater than input key (due to our method to traverse the tree), the smallest key should be the floor key; finding ceil key is a bit tricky with my search method, in that I will need to check if the left child key is less than or equals to key, or left child is just nullptr, and if current key is greater than or equals to key, and if both are true, current key could be potential ceil. We need to check these condition because knowing the next small key could help us determine if the current key could be ceil of input key or not. If the left child is just nullptr, than we can directly compare current key to input key, and put current key as potential ceil if it is bigger than input key.

If program could not find ceil, then it should throw an overflow exception, since the key must be bigger than the largest value contained by the binary tree. Same but opposite for floor, the input key must be smaller than the smallest of binary tree, thus underflow exception must be thrown. To test these error, the program just needs to try to run ceil function with huge input and floor function with low input. (side note: I added many more test numbers to see if my program could find the appropriate floor/ceil if it should be able to)

For the kth_small function, the program used queue to store information from the tree. All we need to do is to pop out the front of the queue k-1 times, and the remain front of the queue must be the kth smallest item in the tree. The program should throw an out_of_range error if the function fail to find kth_small, this is because input kth must be bigger than number of items in tree, or smaller than 0. In other word, input is out of the range of the binary tree. To verify if this program could check the error, the program could try to catch out_of_range_error if input for kth_small function call is bigger than any elements in binary tree or is less than 0.

In addition, when detected empty tree, the program should throw a runtime_error, because empty tree was detected during run time.