**Data Structures**

**(CSE22101)**

**Assignment 4:**

**Saken Kenzhegulov**

**(20132027)**

Binary Search Tree

BST() – assigning to our root NULL since we creating empty tree;

~BST() – assigning NULL to our root and deleting our root, other nodes will be deleted by TreeNode class

TreeNodeType\* Search(const KeyType k) – first creating temp pointer which first points to the root of our tree if it exists, otherwise returning NULL. Then, searching according to Key, if temp’s current Key is less we go to the left child if it exists, if bigger than Key then to the right child if it exists, when there is no child on the right or left then we return NULL;

Void Insert(const KeyType k, const ValType v) – first we create a new node for that. If our tree is empty we make that Node as a root, otherwise going through tree while we found Node with the same Key, updating the Value of that Node, or make it one Node’s left or right child, by linking them.

bool Remove(const KeyType k) – first we are searching for Node with Key equal to k. If this Node doesn’t exist then return false. Otherwise, removing this Node. Instead of this Node we put left child if that Node had only left child, if it had both, then we put its right child’s leftmost Node instead removed Node (this Node can be right child of removed Node). Linking new Node with removed Node’s parent if it exists, and unlinking from its old parent.

void PrintInorder() – printing using LVR order, by using additional function for recursion. First we go to the leftmost Node and print it, go back to parent and print it and then go to the right child and do the same thing.

Word Frequency

void ReadText(const char\* filename, const MapType t) – reading text using ifstream and word by word just using string sub. Using function tolower we make our word lowercase. Then removing all non-alphabetical characters from our word and calling IncreaseFrequency(ss) function, where ss is the converted string.

int GetFrequency(const std::string word) – using Search function of BST class, we searching for word. Since the word is saved as Key and frequency as Value of BST we return value of Node with Key equal to word, if no Key with word found we return 0;

void IncreaseFrequency(const std::string word) – using Insert function of BST class we add that word as Key to our tree with Value v. Since we just need to increase to 1 our frequency of word, if this word already exist in our tree, we assign Value of this Node with Key word to our v. If there is no Key with word then v = 1;

void PrintAllFrequency() – just calling PrintInorder function of BST class, we print all word with its frequency in our BST tree.