Ethan Kulman
100454669
HW6

HW 6 Binary Search Trees

The binary search tree dictionary program is made up of the following three files: “bst.h”, “bst.cpp”, and “main-bst.cpp”. The “bst.cpp” file contains all of the definitions of the methods for the “BSTree” class. The declarations of methods and data members for the “BSTree”, and “BSTNode” classes can be found in the “bst.h” header file. The “main-bst.cpp” program is a driver that utilizes the “BSTree”, and “BSTNode” classes to load a text file into memory, and compare the words in that text file to the words contained in two separate dictionary files.

bst.h & bst.cpp

The “bst.h” header file declares two different classes, “BSTNode”, and “BSTree”. The “BSTNode” class contains four different private data members, and a public constructor and destructor. This class is used to build a tree inside of the “BSTree” class. The four different private data members of the “BSTNode” class are as follows:

- “string data” which holds a unique word
- “int frequency” which keeps track of how many times the word has appeared
- “BSTNode *left” is a pointer to a “BSTNode” which represents a word that appears alphabetically lower than the word held in “std::string data”
- “BSTNode *right” is a pointer to a “BSTNode” which represents a word that appears alphabetically greater than the word held in “std::string data”

There is then a constructor and a destructor which are defined as follows:

- “BSTNode(std::string)” is a constructor which takes in a string and sets the “std::string data” member of the BSTNode to the value of the string passed in. It then sets the “frequency” of this word to 1, and sets its “BSTNode *left” and “BSTNode *right” to null.
- ~”BSTNode()” is an empty destructor

The “BSTree” class has three private data members, and six private methods. These are defined as follows:

- “BSTNode *root” is a pointer to the root node of the tree
- “int size” keeps track of the number of inserts that are called
- “int unique_words” is used to keep track of the number of nodes that the tree contains
- “void destroy(BSTNode*)” is a method that recursively moves through each node of the tree and deallocates the memory for that node.
- “BSTNode * find(BSTNode *, std::string)” is a method which takes in a pointer to a “BSTNode” which will be referred to as “BSTNode * r”, and a string which will be referred to as “std::string w”. The string “w” and the node parameters are passed into the “strcmp” function from the “string.h” library, which returns whether or not the word “w” is greater, less than, or equal to the data located in “r”. It then takes the integer return value from “strcmp” function and uses it to evaluate a set of conditionals:
  - If the return value from “strcmp” is 0, then the function will return the node “r”
  - If the return value from “strcmp” is greater than 0, then the function will check if the

    "BSTNode * right" data member of the node "r" is null, and if so it will return the node "r". If the right data member of "r" is not null, then it will return a function call of "BSTNode * find(BSTNode &, std::string)", and it will pass in the right node of "r", which is "r->right", and the word "w".
  - If the return value from "strcmp" is less than 0, then the function will check if the "BSTNode * left" data member of the node "r" is null, and if so it returns the node "r". If the left data member of "r" is not null, then it will return a function call of "BSTNode * find(BSTNode &, std::string)", and it will pass in the left node of "r", which is "r->left", and the word "w".
  - If no conditions are met the function returns the node "r".
- "void increment_frequency(BSTNode *)" increments the "frequency" data member by one for whatever "BSTNode" is passed in
- "void insert(BSTNode**, std::string)" is a function that takes in a double pointer to a "BSTNode" referred to as "currentNode", and a "std::string" referred to as "newStr". It then performs the following:
  - It first checks the "root" data member of the "BSTree" class to see if it is null. If "root" is null, then it dynamically allocates a "BSTNode" and passes the "newStr" into its constructor. It then sets the "root" to the newly allocated node, and sets "size" to 1, and "unique_words" to 1. The function will then return nothing.
  - If the first conditions is not met, then "BSTNode * find(BSTNode *, std::str)" is called and is passed a dereferenced "currentNode" and the word "newStr". This returns a pointer to a BSTNode, the variable "existingNode" is set equal to this return value.
  - Next the "strcmp" function from the "string.h" library is passed the "data" parameter of "existingNode" and the word "newStr". This return value is assigned to the variable "int result".
  - "result" is then passed into a set of conditionals.
    - If "result" is 0, then "newStr" is already in the tree and so "existingNode" is passed into the "increment_frequency" function. Then the "size" data member is incremented by 1.
    - If "result" is greater than 0, then a new BSTNode is dynamically allocated with "newStr" passed into its constructor. This new node is referred to as "addedNode". Then the "right" data member of "existingNode" is set equal to "addedNode". The "size" and "unique_words" data members of the BSTree are then incremented by one.
    - If "result" is less than 0, then a new BSTNode is dynamically allocated with "newStr" passed into its constructor. This new node is referred to as "addedNode". Then the "left" data member of "existingNode" is set equal to "addedNode". The "size" and "unique_words" data members of the BSTree are then incremented by one.
    - The function then returns nothing
- "void print_list(BSTNode*, int, bool)" is function that recursively moves through the BSTree tree structure and prints out information related to the BSTNode that has been passed into its parameters. It does this by performing a pre-order traversal of the binary tree. This implementation allows a boolean to be passed in which modifies the printout of the function. If the boolean passed into the function parameters is true, then the function will print out the "data" and "frequency" data members of the BSTNode. If the boolean passed into the function parameters is false, then it will only print out the "data" data member of the BSTNode.
- "void print_range(std::string, std::string, BSTNode*)" is a method which takes in a pointer to a BSTNode referred to as "currentNode", and two strings which will be referred to as "startWord", and "endWord". If the "currentNode" is not null, then the function performs the following operations:

    - There are two calls made using the “strcmp” function from the “string.h” library which determine whether the “data” data member of “currentNode” appears alphabetically between the “startWord” and “endWord”.
    - If the “data” data member of “currentNode” does appear between the “startWord” and “endWord”, then it will call itself recursively, “print_range(std::string, std::string, BSTNode*)”, once passing the “startWord”, “endWord”, and “currentNode->left”. This will cause the left BSTNode of each BSTNode to be traversed until the “data” is not within the range of “startWord”, and “endWord”. Then “print_list(BSTNode*, int*, bool)” is called with “currentNode” and a pointer to an integer variable with the value of 1. This will print out the “data” data member of “currentNode”. Then it will call itself recursively, “print_range(std::string, std::string, BSTNode*)”, once passing the “startWord”, “endWord”, and “currentNode->right”. This will check the right node of “currentNode” to see if this node contains data that is within the alphabetical range.
    - If the “data” data member of “currentNode” is only less than the “endWord”, then it will call itself recursively, “print_range(std::string, std::string, BSTNode*)”, once passing the “startWord”, “endWord”, and “currentNode->right”.
    - If the “data” data member of “currentNode” is only greater than the “startWord”, then it will call itself recursively, “print_range(std::string, std::string, BSTNode*)”, once passing the “startWord”, “endWord”, and “currentNode->left”.

The “BSTree” class then declares a constructor and destructor, as well as 9 public methods. These are defined as follows:

! “void insert(std::string)” is a method that takes in string as a parameter. It then passes a double pointer that points to the address of the “root” data member of the tree, and the string it received through its parameters into the private “insert” method.
! “void print_list(int)” is a method that takes in an integer “n”. It then calls the private “print_list(BSTNode*, int*, bool)” function and passes this function a pointer to “n”, and the “root” data member. This will cause the private “print_list(BSTNode*, int*, bool)” function to print out the first “n” elements of the tree by performing a preorder traversal of the binary search tree.
! “void print_tree()” is a method which passes the “root” and a copy of the “unique_words” data members of the BSTree class into the private “print_list(BSTNode*, int*, bool)”, and therefore prints out the entire tree.
! “void print_range(std::string, std::string)” is a method which passes the two strings it received as well as the “root” BSTree data member into the private “print_range(std::string, std::string, BSTNode*)” in order to print out all words in the BSTree that appear between the two strings.
! “int get_size()” returns the “size” data member of the BSTree, which is how many times insert has been called.
! “int get_uniques()” returns the “unique_words” data member of the BSTree, which is how many nodes the tree has.
! “bool containsWord(std::string)” utilizes the private find method to check if a word already exists in the BSTree

Main-bst.cpp

The “main-bst.cpp” file defines three supplementary functions as well as a main function. These three supplementary functions are as follows:

! “int loadDictionaryArray(std::string *, std::string)” takes in a pointer to an array of strings, and a string that represents the name of a text file. Then the function will open the file using an ifstream, and will iterate through each word of the file. For each word, the function will strip away any punctuation, make the letters lower case, and then insert that word into the array of strings. This function will then return the number of words that were inserted into the array.

! "bool binSearch(std::string*, std::string, int, int)" implements a binary search to look up words in the dictionaries more quickly. It utilizes the "strcmp" function from the "string.h" library in order to do this.

! "void checkFile(BSTree *, std::string *, int, std::string *, int, std::string, std::string, std::string)" is passed a pointer to a BSTree which will be referred to as "newFile", an array of strings referred to as "dict1" which contains all of the words in the "dictionary.txt" file, an integer that is the number of words in the "dictionary.txt" file which is referred to as "dict1size", an array of strings referred to as "dict2" which contains all of the words in the "dictionary-brit.txt" file, an integer that is the number of words in the "dictionary-brit.txt" file, which is referred to as "dict2size", a string that is the name of the file that will be processed referred to as "response", a string representing the start word for a range query referred to as "s1", and a string representing the end word for a range query referred to as "s2". The function utilizes these parameters to iterate through each word in the text file whose name is stored in "response", and then perform the following:
  - Each word will be stripped of any punctuation and will be set to lower case letters. Then the word will be checked to see if it is alphabetically between s1 and s2, and if this is true a variable holding how many words should appear in the range query will be incremented by one. After this is performed, the word will be passed into the "insert" function of "newFile"
  - Then for each word the function will check if it exists in either "dict1" or "dict2".
  - If the word does not exist in "dict1", then the word will be inserted into an instance of the "BSTree" class that will contain all words not found in the first dictionary.
  - If the word does not exist in "dict2", then the word will be inserted into an instance of the "BSTree" class that will contain all words not found in the second dictionary.
  - Then all of the words not in "dict1", or "dict2" will be printed to the screen. And the "size" and "unique_words" data members of "newFile" will be printed to the screen.
  - Next the "print_range" method of "newFile" will be called and passed "s1", and "s2". This will print all words contained within "newFile" that appear alphabetically between these two words.

! The main function performs the following:
  - It first dynamically allocates a "BSTree" and sets it equal to "newFile".
  - It then opens "dictionary.txt" and reads the number of words in the file and sets this equal to the variable "wc1", it then dynamically allocates an array of string and sets the size to the value of "wc1". This array of strings referred to as "dictionaryArr1", and the string, "dictionary.txt" are passed to the "loadDictionaryArray" function.
  - It then opens "dictionary-brit.txt" and reads the number of words in the file and sets this equal to the variable "wc2", it then dynamically allocates an array of string and sets the size to the value of "wc2". This array of strings referred to as "dictionaryArr2", and the string, "dictionary-brit.txt" are passed to the "loadDictionaryArray" function.
  - Then the user is prompted to enter a name of a text file, and their response is recorded in the variable "response".
  - Then the user is prompted for a start and end word for a range query. These are stored in the variables "s1", and "s2".
  - Then "newFile", "dictionaryArr1", "wordCount1", "dictionaryArr2", "wordCount2", "response", "s1", and "s2" are passed to the "checkFile" function.

- Lastly, “dictionaryArr1”, “dictionaryArr2”, and “newFile” are deallocated.