Aaron Williams

Project 1

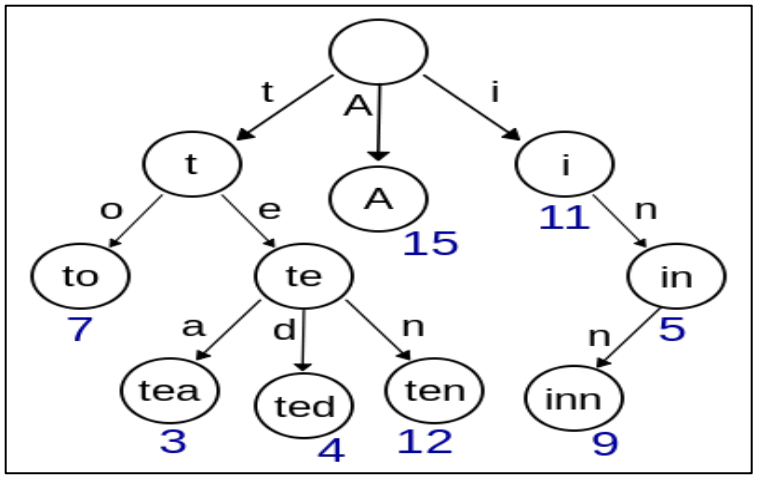
10 October 2017

Assignment Description

The purposes of this project were to create a Trie data structure based on a provided dictionary file, and then use the created Trie to implement a command-line auto complete search query interface. This project is split into two separate parts. The first part is the Trie construction. The second part is the implementation of the auto complete search query functionality.

Logic and Outputs

1) The Trie data structure created in this project was based on the interpretation provided by Wikipedia (https://en.wikipedia.org/wiki/Trie). The Trie data structure is used to store strings and is composed of two components- Nodes and Edges. The Trie contains a root Node member when it is constructed, but all other Nodes and all Edges are constructed based on the strings that are inserted. The root Node contains no data, but has multiple Edges that each has a child Node. Each Edge stores a letter or number (of data type char) that leads to a child Node containing a substring of the original string ending in the letter or number stored by the aforementioned edge. Once the last Node in the sequence of Nodes is created, it is assigned a number value (of data type int) that indicates the data contained by the node is a valid search query. Below is a visual representation of the Trie data structure as described.



https://en.wikipedia.org/wiki/Trie

The first step of constructing the Trie data structure was to build the associated Node and Edge classes. Since the classes reference each other, they had to be declared in the program header. The Node class was built first. Each Node instance contains an integer value, qv, which indicates if the Node contains a valid search query. Each Node also contains a string member, data, which is the string or substring stored in the Node. Lastly, each Node contains an array of Edge pointers set to a size of thirty-seven with all elements initially set to NULL. The size of thirty-seven was chosen to represent the possibility of each Edge storing a special character (EX: '_'), number (0-9), or a letter (a-z). Additionally, an assign method was added that sets the data member to the passed string.

Node Class:

```
21  //Node class: represents a node in the trie.
22  class Node {
23      public:
24          Node() {
25              qv = 0;
26              //Set all elements of edge array to NULL.
27              for(int i = 0; i < 37; i++) {
28                  edges[i] = NULL;
29              }
30          }
31          ~Node() {}
32          //Query value assigned to valid node.
33          int qv;
34          //String or substring contained in node.
35          string data;
36          //Array of pointers to potential edges of the node.
37          Edge* edges[37];
38          //Assigns the current substring to the node.
39          void assign(string data) {
40              this->data = data;
41          }
42  };
```

The Edge class was built following the Node class. Each Edge contains a letter member (of data type char) that stores the letter or number assigned to the Edge. Each Edge also

has two Node pointers, one to its child Node and one to its parent Node.

Edge Class:

```
//Edge class: represents the edge connecting two nodes.
class Edge {
    public:
        Edge() {
            //Sets initial edge value to null byte.
            letter = '0';
        }
        ~Edge() {}
        //Letter referenced by edge.
        char letter;
        //Edge child  node.
        Node* c;
        //Edge parent node.
        Node* p;
};
```

Once the Edge and Node classes were completed, the Trie class was created. As mentioned above, each instance of the Trie is initialized with a root Node pointer. Additionally, the Trie utilizes a value member (of data type int) that iterates as each string is inserted into the Trie, and is used to provide the query value (qv) for each end Node in a string sequence. The Trie class contains two member functions, insert and search.

Trie Class (Data Members):

```
//Trie class: combination of nodes and edges.
class trie {
    private:
    //Root node of trie.
    Node* root;
    //Iterator used to assign values to nodes.
    int value;
    public:
        trie() {
            root = new Node();
            value = 0;
        }
        ~trie() {
            delete root;
        }
```

The insert function is used to insert a string into the Trie. It accepts a passed string parameter and returns nothing. The insert function uses a Node pointer, a, which is initially set to the root Node of the Trie. The function sets an integer value, len, equal to the size of the string. The function then iterates through the string position by position. The function first checks if the value at the current string position is a number or a letter and then determines the decimal value of that number or letter. Once the decimal value is determined, the function checks if the Edge for that number or letter already exists. If it exists, a is iterated to the child Node of that Edge. If it does not exist, a new Edge is created in the aforementioned decimal value position in the Nodes Edge pointers array. Next, a child Node is created for that Edge. The Edge is assigned the letter or number in the current position and the child Node is assigned the current substring. Then, a is set as the parent node of the Edge. Once those steps are complete, a is iterated to the newly created child Node. Once the iteration of the string is complete, the last Node created is assigned a query value using the Trie member value.

Insert Function:

```
77          //Inserts a string into the trie.
78          void insert(string s) {
79              Node *a = root;
80              int len = s.size();
81              for(int i = 0; i < len; i++) {
82                  int d = 0;
83                  //Checks if character at current string position is digit.
84                  //If so, assigns it in array based on its value.
85                  if(isdigit(s[i])) {
86                      d = s[i] - '0' + 1;
87                  }
88                  //Checks if character at current string position is char.
89                  //If so, assigns it in array based on its decimal value.
90                  else if(isalpha(s[i])) {
91                      d = s[i] - 'a' + 11;
92                  }
93                  if(! a->edges[d]) {
94                      //Creates new edge for the position.
95                      a->edges[d] = new Edge();
96                      //Creates new child node for the edge.
97                      a->edges[d]->c = new Node();
98                      //Assigns the current letter to the edge.
99                      a->edges[d]->letter = s[i];
100                     //Assigns the current substring to the child node.
101                     a->edges[d]->c->assign(s.substr(0,i+1));
102                     //Assigns the current node as the parent node of new edge.
103                     a->edges[d]->p = a;
104                 }
105                 //Iterates current node to newly created node.
106                 a = a->edges[d]->c;
107             }
108             //Increase count iterator by one.
109             value++;
110             //Assigns an integer value to last node in string sequence.
111             a->qv = value;
112         }
```

The search function was created to test if a given string is present in the Trie. It utilizes similar logic flow seen in the insert function. The search function accepts a passed string parameter and returns a Boolean response. The search function determines if the string is present based on the assigned query value (qv) of the last node in the string sequence. Since each Node's query value is initially set to zero, anything not equal to zero returns true as a tested string. If the Edge containing the next letter of the passed string doesn't exist, or if the Node query value is equal to zero, the tested string returns false.

Search Function:

```
        //Searches trie for a given string.
        bool search(string s) {
            Node *a = root;
            int len = s.size();
            for(int i = 0; i < len; i++) {
                int d = 0;
                //Checks if character at current string position is digit.
                //If so, assigns it in array based on its value.
                if(isdigit(s[i])) {
                    d = s[i] - '0' + 1;
                }
                //Checks if character at current string position is char.
                //If so, assigns it in array based on its decimal value.
                else if(isalpha(s[i])) {
                    d = s[i] - 'a' + 11;
                }
                //Checks if edge exists. If so, iterates to next node.
                if(a->edges[d]) {
                    a = a->edges[d]->c;
                }
                else {
                    cout << s << " not found." << endl;
                    return false;
                }
            }
            //If nodes query value is not zero, it means it is the end node
            //in the sequence for the string.
            if(a->qv != 0) {
                cout << s << " present in Trie." << endl;
                cout << "Value: " << a->qv << endl;
                return true;
            }
            else {
                cout << s << " not found." << endl;
                return false;
            }
        }
};
```

*Note: The); marks the end of the Trie class.

A trieBuild function was implemented separate from the Trie class to construct a Trie based on a given file of strings. The void function accepts two parameters, a Trie, which is passed by reference and a string, which represents the name of the file. The function creates a file stream using the passed file name. It first checks if the file exists, and if it doesn't exits the program. The function interprets each line as a string and iterates through the file, inserting each string into the Trie.

TrieBuild Function:

```
153  //Builds trie using a given file.
154  void trieBuild(trie & Trie, string fileName) {
155      ifstream file (fileName);
156      //Checks if file exists.
157      if(!file) {
158          cout << "Unable to locate file." << endl;
159          exit(1);
160      }
161      string line;
162      //Continues until end of file.
163      while(getline(file, line)) {
164          if(line.empty()) {
165              continue;
166          }
167          //Calls insert method from trie class.
168          Trie.insert(line);
169      }
170      file.close();
171  }
```

Once all the necessary functionality for the program was built, it had to be adequately tested. Using the given dictionary.txt file, three strings were chosen for testing against the search function contained in the Trie class. The strings selected were the first string, “aaa”, the last string, “zycher”, and a random string in the middle “pr01”. Confirming that the Trie contains these strings indicates that the Trie was built appropriately. In order to verify the proper query values, the positions of the strings in the dictionary.txt file were found prior to testing. Additionally, a random string not contained in the file, “dogbone”, was used for testing.

Determining String File Positions:

```
[Ace:Project 1 A$ grep -n aaa dictionary.txt
1:aaa
2:aaas
8469:epmi_ciso_aaa
[Ace:Project 1 A$ grep -n zycher dictionary.txt
28102:zycher
[Ace:Project 1 A$ grep -n pr01 dictionary.txt
1228:apr01
10619:go2ntswpr01scecom
19247:pr01
Ace:Project 1 A$
```

Current Main Function:

```
173  int main() {
174      trie Trie;
175      trieBuild(Trie, "dictionary.txt");
176      cout << "Trie build complete." << endl;
177      Trie.search("aaa");
178      cout << endl;
179      Trie.search("dogbone");
180      cout << endl;
181      Trie.search("zycher");
182      cout << endl;
183      Trie.search("pr01");
184  }
```

Output:

```
[Ace:Project 1 A$ g++ trie.cpp -o 1
[Ace:Project 1 A$ ./1
Trie build complete.
aaa present in Trie.
Value: 1

dogbone not found.

zycher present in Trie.
Value: 28102

pr01 present in Trie.
Value: 19247
Ace:Project 1 A$
```

Based on the output, it is confirmed that Trie was built and is functioning as intended.

2) The next step was implementing the auto complete search query functionality. In order to accomplish this, a function was created with the purpose of returning all possible strings that could complete a given prefix. The void function trieFind was developed to accept a Trie parameter passed by reference and a string parameter. The function uses a Node pointer, a, which is initially set to the root Node of the Trie. Similar to methods in the Trie class, the function iterates through the length of the prefix string, traversing child Nodes, if the corresponding Edge exists. If the Edge associated with a letter in the prefix doesn't exist, it indicates that there are no entries in the Trie that complete the prefix. If no entries exist for the prefix, a message is printed and the function is exited. Once the end of the prefix is reached, the Node pointer, a, will point to the Node that holds the prefix data. The function will then check if that Node is a valid search query and if so, will print the prefix as part of the auto complete results. Lastly, the function utilizes a loop to iterate through all possible Edges of the Node and recursively calls itself, ensuring that all possible auto complete results are displayed.

TrieFind Function:

```
void trieFind(trie & Trie, string prefix) {
    Node* a = Trie.root;
    int len = prefix.size();
    for(int i = 0; i < len; i++) {
            int d = 0;
            //Checks if character at current string position is digit.
            //If so, assigns it in array based on its value.
            if(isdigit(prefix[i])) {
                d = prefix[i] - '0' + 1;
            }
            //Checks if character at current string position is char.
            //If so, assigns it in array based on its decimal value.
            else if(isalpha(prefix[i])) {
                d = prefix[i] - 'a' + 11;
            }
            //Checks if edge exists. If so, iterates to next node.
            if(a->edges[d]) {
                a = a->edges[d]->c;
            }
            //If edge doesn't exist, it exits function.
            else if(! a->edges[d]) {
                cout << "No entries found for prefix: " << prefix << endl;
                return;
            }
    }
    //Checks if prefix is a valid search query.
    if(a->qv != 0) {
        cout << prefix << endl;
    }
    //Iterates through current node edges.
    for(int j = 0; j < 37; j++) {
        if(a->edges[j]) {
            //Utilizes recursive function call.
            trieFind(Trie, a->edges[j]->c->data);
        }
    }
}
```

The last function created was the CLI function, which accepted the parameter of a Trie passed by reference and provided no returns. The CLI function was developed to provide the interface for a user to input prefixes and search the Trie for auto complete results. The function initializes two string instances, word and response. Word is used to reference the user provided prefix and response is used to represent user input on whether to conduct another search or not. The function prompts a user for a prefix and then calls the trieFind function using that prefix. Once the results are provided, the function asks the user if he or she wants to search again. If a user indicates yes, then the function is recursively called. Otherwise, the program ends.

CLI Function:

```
212 void CLI(trie & Trie) {
213     //Prefix to be searched.
214     string word;
215     //User response for another search.
216     string response;
217     //Prompts user for prefix.
218     cout << "Please enter prefix to search for: " << endl;
219     cin >> word;
220     cout << endl;
221     //Calls trieFind and prints results.
222     cout << "Results: " << endl;
223     trieFind(Trie, word);
224     cout << endl;
225     //Asks user for another search.
226     cout << "Search again? Enter y for yes." << endl;
227     cin >> response;
228     if(response == "y" || response == "Y") {
229         cout << endl;
230         CLI(Trie);
231     }
232 }
```

Final Main Function:

```
234 int main() {
235     //Trie initialized.
236     trie Trie;
237     //Trie built using dictionary.txt file.
238     trieBuild(Trie, "dictionary.txt");
239     //Perform search on Trie.
240     CLI(Trie);
241 }
```

Sample Outputs:

```
Please enter prefix to search for:
bart

Results:
bart
bartender
barter
barth
barthel
bartholomew
bartiromo
bartlett
barton

Search again? Enter y for yes.
n
```

```
[Ace:Project 1 A$ ./1
Please enter prefix to search for:
powj

Results:
No entries found for prefix: powj

Search again? Enter y for yes.
y

Please enter prefix to search for:
aa

Results:
aaa
aaas
aactive
aadvantage
aaker
aap
aapg
aaron
aarp
aas
aau

Search again? Enter y for yes.
n
```

```
[Ace:Project 1 A$ ./1
Please enter prefix to search for:
src

Results:
src
src3d
srcspacergif
srcttpwwwfunjetcomemailimagesspacergif

Search again? Enter y for yes.
Y

Please enter prefix to search for:
yankedod

Results:
No entries found for prefix: yankedod

Search again? Enter y for yes.
n
```