YILDIZ TECHNICAL UNIVERSITY

ELECTRICAL- ELECTRONICS

FACULTY

COMPUTER ENGINEERING

DEPARTMENT

KAMRAN BALAYEV 17011904

Creating Huffman Tree

In this assignment I will show the code of creating Huffman tree. Application has 9 functions included the main function and one struct.

Explanation of functions:

- Print list function:
  - Purpose of this function is listing the frequencies. It has one parameter called node which will be used as reference to the head of the linked list.
- Find the frequency function:
  - This function is finding the frequencies of characters in linked list. It has 3 parameters one of them is freq which is reference to the head of linked list, the second one is input string and the last one is the length of this string.
- Insertion sort function:
  - This function will be used for sorting the frequencies. It has 1 parameter and it is the head of the linked list.
- Sorted Insert function:
  - This function will be used for inserting new node in a list . It has 2 parameters, one of them is double pointer for referencing head of linked list and second one is single pointer referencing new node

- Create node function:
  - This function will be used for creating Huffman tree nodes. Function has 3 parameters one of them is the left part of tree, another one is right part of tree and the final parameter is value of root which is the sum of left and right
- Insert function:
  - This function will be used to insert a new node to the sorted list. First parameter is the head of list and the second parameter is new node.
- Print Huffman function:
  - This function will print the Huffman tree. It has 2 parameters one of them is the head of linked list and the second one is space which will be used for distance between levels.
- Huffman function:
  - This function is creating the Huffman tree. It has one parameter which is the head of linked list.

Detailed explanations and screenshots are available below:

```c
/*
    KAMRAN BALAYEV 17011904

    SOURCES:
        https://www.geeksforgeeks.org/
        https://www.programiz.com/
*/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
 #define COUNT 10 //variable for printing huffman tree
typedef struct Node {
    int data;//frequency of character
    char ch;//character
    struct Node *left, *right, *next;//left, right and next element of list
}node;

 void printList(node *node);
 void findFrequency(node *, char *, int);
 void sortedInsert(node **, node*);
 void insertionSort(node **);
 void hufman(node *freq);
 void insert(node *, node*);
 node * createNode(node*, node*, int);
 void printHufman(node *, int);
```

```
28
29
30  int main()
31  {
32
33      char textArr[1000];//store the string value
34      int inputLength, a = 0;
35      node *freq = (node*)malloc(sizeof(node));//allocate node
36
37      freq->right = NULL;
38      freq->left = NULL;
39
40
41      printf("Please enter the text: ");
42      gets(&textArr);//scan the string from user
43      //assign the length of string to the proper variable
44      inputLength = strlen(textArr);
45      //call the function for calculating frequency
46      findFrequency(freq, textArr, inputLength);
47      //insert the frequencies
48      insertionSort(&freq);
49      freq = freq->next;//point to the right location
50      //print the listed frequencies
51      printList(freq);
52      //call the huffman function in order to create the tree
53      hufman(freq);
54      return 0;
55  }
```

```
56  /*
57      Purpose of this function is listing the frequencies.
58      It has one parameter called node which will be used
59      as  reference to the head of linked list.
60  */
61  void printList(node *node) { ... }
69
70  /*
71      This function is finding the frequencies of characters in linked list.
72      It has 3 parameters one of the is freq which is referance to the head
73      of linked list, second one is input string and the last one is the length
74      of this string.
75  */
76  void findFrequency(node *freq, char *textArr, int inputLength) {
77      int i = 0, j, count;
78      node * head = freq;//this variable will be used for setting head of list operation.
79      //find the frequency of character
80      for (i = 0; i < inputLength; i++) {
81          count = 1;//reset count
82          if (textArr[i]) {
83              for (j = i + 1; j < inputLength; j++) {
84                  //if the character matches with another
85                  if (textArr[i] == textArr[j])
86                  {
87                      count++;//increment of the frequency of proper character
88                      textArr[j] = '\0';//purpose of this operation is not repeating frequency during calculation
89                  };
90              }
```

```
91                  freq->ch = textArr[i];//add the character to the list
92                  freq->data = count;//add the counter of character to the list
93                  freq->next = (node*)malloc(sizeof(node));//allocate node
94                  freq = freq->next;//freq will show the next node of list
95                  freq->left = NULL;
96                  freq->right= NULL;
97              }
98          }
99          freq->next = NULL;//make the end of linked list equal to the NULL
100         freq = head;//assign the header of linked list
101     }
102
103     // function to sort a linked list using insertion sort
104     void insertionSort(node **head)
105     {
106         // Initialize sorted linked list
107         node *sorted = NULL;
108
109         // Traverse the given linked list and insert every
110         // node to sorted
111         node *current = *head;
112         while (current != NULL)
113         {
114             // Store next for next iteration
115             node  *next = current->next;
116
117             // insert current in sorted linked list
118             sortedInsert(&sorted, current);
```

```
119
120             // Update current
121             current = next;
122         }
123
124         // Update head_ref to point to sorted linked list
125         *head = sorted;
126     }
127
128     //function to insert a new node in a list
129     void sortedInsert(node ** head, node* newNode)
130     {
131         node* current;
132         /* Special case for the head end */
133         if (*head == NULL || (*head)->data >= newNode->data)
134         {
135             newNode->next = *head;
136             *head = newNode;
137         }
138         else
139         {
140             /* Locate the node before the point of insertion */
141             current = *head;
142             while (current->next != NULL && current->next->data < newNode->data)
143             {
144                 current = current->next;
145             }
146             newNode->next = current->next;
```

```
147             current->next = newNode;
148         }
149     }
150
151     /*
152         This function will be used to insert new node to the sorted list.
153         First parameter is the head of list and the second parameter is
154         new node.
155     */
156     void insert(node *freq, node * newNode) {
157
158         while (newNode->data > freq->data  && freq->next != NULL)
159         {
160             freq = freq->next;
161         }
162
163         newNode->next = freq->next;
164         freq->next = newNode;
165     }
166     /*
167         This function will be used for creating huffman tree nodes.
168         Function has 3 parametres one of them is the left part of tree, another one is right
169         part of tree and the final parameter is value of root which is the sum of left and right
170     */
171     node *createNode(node* left, node*right, int sum) {
172         node * newNode = (node*)malloc(sizeof(node));//allocate node
173         newNode->right = (node*)malloc(sizeof(node));//allocate node
174         newNode->left = (node*)malloc(sizeof(node));//allocate node
```

```
175
176         newNode->left = left;//assign left value of tree
177         newNode->right = right;//assign right value of tree
178
179         newNode->data = sum;//assign the value of root which is sum of left and right
180         newNode->ch = NULL;//root value has no character
181
182         return newNode;//return the node
183     }
184
185
186     /*
187         This function will print the huffman tree.
188     */
189     void printHufman(node *freq, int space)
190     {
191         // Base case
192         if (freq == NULL)
193             return;
194
195         // Increase distance between levels
196         space += COUNT;
197
198         // Process right child first
199         printHufman(freq->right, space);
200
201         // Print current node after space
202         printf("\n");
```

```
202         printf("\n");
203         for (int i = COUNT; i < space; i++)
204             printf(" ");
205         if (freq->ch == '\0')
206             printf("%d\n", freq->data);
207         else
208             printf("%d %c\n", freq->data, freq->ch);
209
210         // Process left child
211         printHufman(freq->left, space);
212     }
213
214     /*
215         This function is creating the huffman tree.
216         It has one parameter which is the head of
217         linked list
218
219     */
220     void hufman(node *freq) {
221         node * newNode;//this variable will used when we call the createNode function
222         int sum = 0;//this variable will store the sum of left and right child in tree
223         while (freq->next != NULL) {
224             sum = freq->data + freq->next->data;//add left and right child of tree
225             newNode = createNode(freq,freq->next,sum);//create new node and assign it to the proper variable
226             insert(freq, newNode);//insert node to the linked list
227             freq = freq->next->next;//show the proper value in linked list for next operations
228         }
229         printf("\n");
230         printf("\n");
231         printHufman(freq, 0);//print the huffman tree
232     }
```

```
Please enter the text: huffman coding is a data compression algorithm
1 1 1 1 2 2 2 2 2 2 2 3 3 3 4 4 5 6


                                        4 o

                                8

                                                        1 u

                                        2

                                                        1 p

                                        4

                                        2 r

                        16

                                        2 f

                                        4

                                        2 c

                                8

                                        4 i

                28

                                        3 m

                                6

                                        3 n

                        12

                                6

46

                                        3 s

                                5

                                        2 h

                        10

                                5 a

                18

                                        2 t

                                4

                                                1 e

                                        2
```

```
Please enter the text: aaabbbcdee
1 1 2 3 3


                    3 a

          6

                    3 b

10

                              1 c

                    2

                              1 d

          4

                    2 e
```