

Name : Bhushan Sharad Tejankar

Roll no. : I30

reg_no. : 2020BIT030

```

1 // Bhushan Sharad Tejankar
2 // 2020BIT030
3 // DAA Practical 10
4 #include <limits.h>
5 #include <stdio.h>
6 #define V 9
7 int minDistance(int dist[], bool sptSet[]){
8     int min = INT_MAX, min_index;
9     for (int v = 0; v < V; v++)
10         if (sptSet[v] == false && dist[v] <= min)
11             min = dist[v], min_index = v;
12     return min_index;
13 }
14 int printSolution(int dist[], int n){
15     printf("Vertex Distance from Source\n");
16     for (int i = 0; i < V; i++)
17         printf("%d \t\t %d\n", i, dist[i]);
18 }
19 void dijkstra(int graph[V][V], int src){
20 int main(){
21     int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
22                          { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
23                          { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
24                          { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
25                          { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
26                          { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
27                          { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
28                          { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
29                          { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
30     dijkstra(graph, 0);
31     return 0;
32 }

```

/tmp/uUTdUB50BW.o

Vertex Distance from Source

0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

main.cpp



Run

Output

Clear

```
1 // Bhushan Sharad Tejankar
2 // 2020BIT030
3 #include <cstdlib>
4 #include <iostream>
5 using namespace std;
6 #define MAX_TREE_HT 100
7 struct MinHeapNode {
8     char data;
9     unsigned freq;
10    struct MinHeapNode *left, *right;
11 };
12 struct MinHeap {
13     unsigned size;
14     unsigned capacity;
15     struct MinHeapNode** array;
16 };
17 struct MinHeapNode* newNode(char data, unsigned freq){
18     struct MinHeapNode* temp = (struct MinHeapNode*)malloc(
19         sizeof(struct MinHeapNode));
20     temp->left = temp->right = NULL;
21     temp->data = data;
22     temp->freq = freq;
23     return temp;
24 }
25 struct MinHeap* createMinHeap(unsigned capacity){
26     struct MinHeap* minHeap
27         = (struct MinHeap*)malloc(sizeof(struct MinHeap));
28     minHeap->size = 0;
29     minHeap->capacity = capacity;
30     minHeap->array = (struct MinHeapNode**)malloc(
31         minHeap->capacity * sizeof(struct MinHeapNode));
32     return minHeap;
```

/tmp/uUTdUB50BW.o

```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

main.cpp



Run

Output

Clear

```
33 }
34 void swapMinHeapNode(struct MinHeapNode** a,
35                      struct MinHeapNode** b){
36     struct MinHeapNode* t = *a;
37     *a = *b;
38     *b = t;
39 }
40 void minHeapify(struct MinHeap* minHeap, int idx){
41     int smallest = idx;
42     int left = 2 * idx + 1;
43     int right = 2 * idx + 2;
44     if (left < minHeap->size
45         && minHeap->array[left]->freq
46         < minHeap->array[smallest]->freq)
47         smallest = left;
48     if (right < minHeap->size
49         && minHeap->array[right]->freq
50         < minHeap->array[smallest]->freq)
51         smallest = right;
52     if (smallest != idx) {
53         swapMinHeapNode(&minHeap->array[smallest],
54                        &minHeap->array[idx]);
55         minHeapify(minHeap, smallest);
56     }
57 }
58 int isSizeOne(struct MinHeap* minHeap){
59     return (minHeap->size == 1);
60 }
61 struct MinHeapNode* extractMin(struct MinHeap* minHeap){
62     struct MinHeapNode* temp = minHeap->array[0];
63     minHeap->array[0] = minHeap->array[minHeap->size - 1];
64     --minHeap->size;
```

/tmp/uUTdUB50BW.o

```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

main.cpp



Run

Output

Clear

```
65     minHeapify(minHeap, 0);
66     return temp;
67 }
68 void insertMinHeap(struct MinHeap* minHeap,
69     struct MinHeapNode* minHeapNode){
70     ++minHeap->size;
71     int i = minHeap->size - 1;
72     while (i
73         && minHeapNode->freq
74         < minHeap->array[(i - 1) / 2]->freq) {
75         minHeap->array[i] = minHeap->array[(i - 1) / 2];
76         i = (i - 1) / 2;
77     }
78     minHeap->array[i] = minHeapNode;
79 }
80 void buildMinHeap(struct MinHeap* minHeap){
81     int n = minHeap->size - 1;
82     int i;
83     for (i = (n - 1) / 2; i >= 0; --i)
84         minHeapify(minHeap, i);
85 }
86 void printArr(int arr[], int n){
87     int i;
88     for (i = 0; i < n; ++i)
89         cout << arr[i];
90     cout << "\n";
91 }
92 int isLeaf(struct MinHeapNode* root){
93     return !(root->left) && !(root->right);
94 }
95 struct MinHeap* createAndBuildMinHeap(char data[],
96     int freq[], int size){
```

/tmp/uUTdUB50BW.o

```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

main.cpp



Run

Output

Clear

```
97 struct MinHeap* minHeap = createMinHeap(size);
98 for (int i = 0; i < size; ++i)
99     minHeap->array[i] = newNode(data[i], freq[i]);
100 minHeap->size = size;
101 buildMinHeap(minHeap);
102 return minHeap;
103 }
104 struct MinHeapNode* buildHuffmanTree(char data[],
105                                     int freq[], int size){
106     struct MinHeapNode *left, *right, *top;
107     struct MinHeap* minHeap
108         = createAndBuildMinHeap(data, freq, size);
109     while (!isSizeOne(minHeap)) {
110         left = extractMin(minHeap);
111         right = extractMin(minHeap);
112         top = newNode('$', left->freq + right->freq);
113         top->left = left;
114         top->right = right;
115         insertMinHeap(minHeap, top);
116     }
117     return extractMin(minHeap);
118 }
119 void printCodes(struct MinHeapNode* root, int arr[],
120               int top)
121 {
122     if (root->left) {
123         arr[top] = 0;
124         printCodes(root->left, arr, top + 1);
125     }
126     if (root->right) {
127         arr[top] = 1;
128         printCodes(root->right, arr, top + 1);
129     }
130 }
```

/tmp/uUTdUB50BW.o

```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

```
129     if (isLeaf(root)) {
130         cout << root->data << ": ";
131         printArr(arr, top);
132     }
133 }
134 void HuffmanCodes(char data[], int freq[], int size){
135     struct MinHeapNode* root
136         = buildHuffmanTree(data, freq, size);
137     int arr[MAX_TREE_HT], top = 0;
138     printCodes(root, arr, top);
139 }
140 int main(){
141     char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
142     int freq[] = { 5, 9, 12, 13, 16, 45 };
143     int size = sizeof(arr) / sizeof(arr[0]);
144     HuffmanCodes(arr, freq, size);
145     return 0;
146 }
147
```


THANK
You!