Practical No. 10

Name: Vijiyant Tanaji Shejwalkar

Reg No: 2020BIT057

1. Dijkstra's algorithm

```
#include <limits.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[]) {
  int min = INT_MAX, min_index;
  for (int v = 0; v < V; v++)
  if (sptSet[v] == false && dist[v] <= min)
    min = dist[v], min_index = v;
  return min_index;
}
int printSolution(int dist[], int n) {
  printf("Vertex Distance from Source\n");
  for (int i = 0; i < V; i++)
    printf("%d \t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src) {
  int dist[V];
  bool sptSet[V];
  for (int i = 0; i < V; i++)
    dist[i] = INT_MAX, sptSet[i] = false;
    dist[src] = 0;
  for (int count = 0; count < V - 1; count++) {
```

```
    int u = minDistance(dist, sptSet);

    sptSet[u] = true;

    for (int v = 0; v < V; v++)

      if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) dist[v] =
dist[u] + graph[u][v];

  }

  printSolution(dist, V);

}

int main() {

  int graph[V][V] = { { 0, 6, 0, 0, 0, 0, 0, 8, 0 },

    { 6, 0, 8, 0, 0, 0, 0, 13, 0 },

    { 0, 8, 0, 7, 0, 6, 0, 0, 2 },

    { 0, 0, 7, 0, 9, 14, 0, 0, 0 },

    { 0, 0, 0, 9, 0, 10, 0, 0, 0 },

    { 0, 0, 6, 14, 10, 0, 2, 0, 0 },

    { 0, 0, 0, 0, 0, 2, 0, 1, 6 },

    { 8, 13, 0, 0, 0, 0, 1, 0, 7 },

    { 0, 0, 2, 0, 0, 0, 6, 7, 0 }

  };

  dijkstra(graph, 0);

  return 0;

}
```

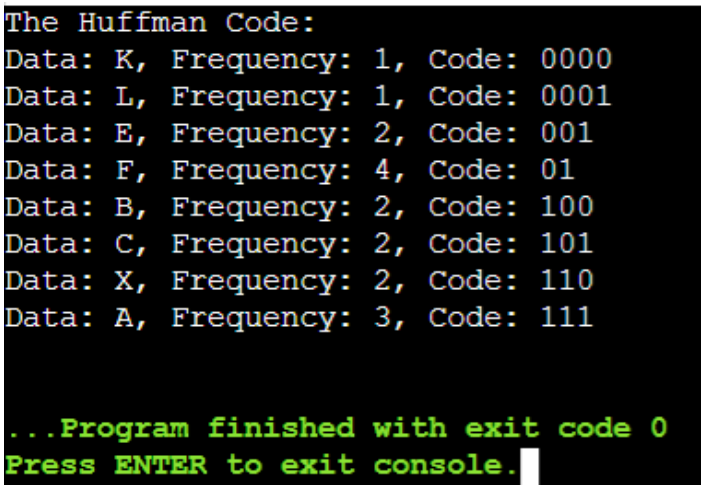
```
Vertex Distance from Source
0         0
1         6
2         14
3         21
4         21
5         11
6         9
7         8
8         15


...Program finished with exit code 0
Press ENTER to exit console.
```

2. Huffman Coding

```
#include<iostream>
#include<queue>
#include<string>
using namespace std;
struct node{
  int freq;
  char data;
  const node *child0, *child1;
  node(char d, int f = -1){ //assign values in the node
    data = d;
    freq = f;
    child0 = NULL;
    child1 = NULL;
  }
  node(const node *c0, const node *c1){
    data = 0;
    freq = c0->freq + c1->freq;
    child0=c0;
    child1=c1;
  }
  bool operator<( const node &a ) const { //< operator performs to find priority in queue
    return freq >a.freq;
  }
  void traverse(string code = "")const{
    if(child0!=NULL){
      child0->traverse(code+'0'); //add 0 with the code as left child
      child1->traverse(code+'1'); //add 1 with the code as right child
    }else{
```

```
      cout << "Data: " << data<< ", Frequency: "<<freq << ", Code: " << code<<endl;
    }
  }
};
void huffmanCoding(string str){
  priority_queue<node> qu;
  int frequency[256];
  for(int i = 0; i<256; i++)
    frequency[i] = 0; //clear all frequency
  for(int i = 0; i<str.size(); i++){
    frequency[int(str[i])]++; //increase frequency
  }
  for(int i = 0; i<256; i++){
    if(frequency[i]){
      qu.push(node(i, frequency[i]));
    }
  }
  while(qu.size() >1){
    node *c0 = new node(qu.top()); //get left child and remove from queue
    qu.pop();
    node *c1 = new node(qu.top()); //get right child and remove from queue
    qu.pop();
    qu.push(node(c0, c1)); //add freq of two child and add again in the queue
  }
  cout << "The Huffman Code: "<<endl;
  qu.top().traverse(); //traverse the tree to get code
}
main(){
  string str = "ACCEBFFFFAAXXBLKE"; //arbitray string to get frequency
  huffmanCoding(str);
}
```


The Huffman Code:
Data: K, Frequency: 1, Code: 0000
Data: L, Frequency: 1, Code: 0001
Data: E, Frequency: 2, Code: 001
Data: F, Frequency: 4, Code: 01
Data: B, Frequency: 2, Code: 100
Data: C, Frequency: 2, Code: 101
Data: X, Frequency: 2, Code: 110
Data: A, Frequency: 3, Code: 111
...Program finished with exit code 0
Press ENTER to exit console.