Experiment No.5

Title:-Minimum Cost Spanning Tree of a given undirected graph using Prim's Algorithm and Kruskal's algorithm and compare.

Problem:- Road and Transportation Networks

Designing road systems, railway lines, or pipeline networks connecting multiple locations with minimum construction or maintenance cost

Programm:-
```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_TREE_HT 100

// A Huffman tree node

typedef struct MinHeapNode {

    char data;          // character

    unsigned freq;      // frequency of character

    struct MinHeapNode *left, *right;

} MinHeapNode;

// Min-Heap: collection of min-heap (or Huffman tree) nodes

typedef struct MinHeap {

    unsigned size;

    unsigned capacity;

    MinHeapNode** array;

} MinHeap;

// Create a new node

MinHeapNode* newNode(char data, unsigned freq) {

    MinHeapNode* temp = (MinHeapNode*) malloc(sizeof(MinHeapNode));
```

```
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}
// Create a min-heap of given capacity
MinHeap* createMinHeap(unsigned capacity) {
    MinHeap* minHeap = (MinHeap*) malloc(sizeof(MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (MinHeapNode**) malloc(minHeap->capacity * sizeof(MinHeapNode*));
    return minHeap;
}
// Swap two nodes
void swapMinHeapNode(MinHeapNode** a, MinHeapNode** b) {
    MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}
// Heapify at given index
void minHeapify(MinHeap* minHeap, int idx) {
    int smallest = idx;
    int left = 2*idx + 1;
    int right = 2*idx + 2;
    if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq)
        smallest = left;
```

```
    if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq)
        smallest = right;
    if (smallest != idx) {
        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}
// Check if size is 1
int isSizeOne(MinHeap* minHeap) {
    return (minHeap->size == 1);
}
// Extract minimum value node
MinHeapNode* extractMin(MinHeap* minHeap) {
    MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    minHeap->size--;
    minHeapify(minHeap, 0);
    return temp;
}
// Insert a new node
void insertMinHeap(MinHeap* minHeap, MinHeapNode* minHeapNode) {
    minHeap->size++;
    int i = minHeap->size - 1;
    while (i && minHeapNode->freq < minHeap->array[(i - 1)/2]->freq) {
        minHeap->array[i] = minHeap->array[(i - 1)/2];
        i = (i - 1)/2;
```

```
}
   minHeap->array[i] = minHeapNode;
}
// Build min-heap
void buildMinHeap(MinHeap* minHeap) {
   int n = minHeap->size - 1;
   for (int i = (n-1)/2; i >=0; i--)
      minHeapify(minHeap, i);
}
// Create and build min-heap from characters and frequencies
MinHeap* createAndBuildMinHeap(char data[], int freq[], int size) {
   MinHeap* minHeap = createMinHeap(size);
   for (int i=0; i<size; ++i)
      minHeap->array[i] = newNode(data[i], freq[i]);
   minHeap->size = size;
   buildMinHeap(minHeap);
   return minHeap;
}
// Build Huffman Tree
MinHeapNode* buildHuffmanTree(char data[], int freq[], int size) {
   MinHeapNode *left, *right, *top;
   MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);

   while (!isSizeOne(minHeap)) {
      left = extractMin(minHeap);
      right = extractMin(minHeap);
```

```
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;

        insertMinHeap(minHeap, top);
    }

    return extractMin(minHeap);
}

// Print Huffman codes
void printCodes(MinHeapNode* root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    if (!(root->left) && !(root->right)) {
        printf("%c: ", root->data);
        for (int i=0; i<top; i++)
            printf("%d", arr[i]);
```

```
        printf("\n");
    }
}
// Huffman encoding function
void HuffmanCodes(char data[], int freq[], int size) {
    MinHeapNode* root = buildHuffmanTree(data, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    printf("Huffman Codes:\n");
    printCodes(root, arr, top);
}
// Main function
int main() {
    char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int freq[] = {5, 9, 12, 13, 16, 45};
    int size = sizeof(arr)/sizeof(arr[0]);

    HuffmanCodes(arr, freq, size);
    return 0;
}
```

Output:-

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                    Code

PS C:\Users\shiva> cd "c:\c language\" ; if ($?) { gcc DAAappl_1.c -o DAAappl_1 } ; if ($?) { .\DAAappl_1 }
Huffman Codes:
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
PS C:\c language>
```

Real-World Applications of MST

1. Network Design
    - Telecommunications: Designing telephone, fiber optic, or internet networks to connect multiple cities with minimal wiring/cabling cost.
    - Computer Networks: Laying out LAN/WAN connections efficiently.
    - MST ensures all nodes are connected with minimum total cost.
2. Electrical Grid Optimization
    - Planning electrical distribution networks (power lines) to connect multiple substations while minimizing construction cost.
3. Road and Transportation Networks
    - Designing road systems, railway lines, or pipeline networks connecting multiple locations with minimum construction or maintenance cost.
4. Water Supply Networks
    - Connecting water sources to cities efficiently, minimizing pipe length and cost.
5. Cluster Analysis in Data Mining
    - Used in hierarchical clustering to connect data points in a cluster with minimal distance (cost) between them.
6. Computer Graphics
    - Generating efficient meshes or networks in rendering and 3D modeling by connecting points (vertices) with minimal edge lengths.
7. Urban Planning
    - Efficiently connecting different buildings, facilities, or utility points (like streetlights, sensors, or traffic lights) with minimal wiring or cabling.