**Assignment: 13**

**EXP**: WAP to implement the following scenarios.

1. Graph representation using adjacency matrix and adjacency list.
2. Graph traversal (BFS and DFS), write the path traversed for the non-weighted graph.
3. Implement Minimum Spanning Tree( Prim+Kruskal) for a weighted graph, and implement the same using both the adjacency matrix and list.

**Coding:**

#include<stdio.h>

#include<stdlib.h>

struct AdjListNode

{

    int dest;

    struct AdjListNode\* next;

};

struct AdjList

{

    struct AdjListNode \*head;

};

struct Graph

{

    int V;

    struct AdjList\* array;

};

struct AdjListNode\* newAdjListNode(int dest)

{

    struct AdjListNode\* newNode =

     (struct AdjListNode\*) malloc(sizeof(struct AdjListNode));

    newNode->dest = dest;

    newNode->next = NULL;

    return newNode;

}

struct Graph\* createGraph(int V)

{

    struct Graph\* graph =

        (struct Graph\*) malloc(sizeof(struct Graph));

    graph->V = V;

    graph->array =

      (struct AdjList\*) malloc(V \* sizeof(struct AdjList));

    int i;

    for (i = 0; i < V; ++i)

        graph->array[i].head = NULL;

    return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

    struct AdjListNode\* newNode = newAdjListNode(dest);

    newNode->next = graph->array[src].head;

    graph->array[src].head = newNode;

    newNode = newAdjListNode(src);

    newNode->next = graph->array[dest].head;

    graph->array[dest].head = newNode;

}

void printGraph(struct Graph\* graph)

{

    int v;

    for (v = 0; v < graph->V; ++v)

    {

        struct AdjListNode\* pCrawl = graph->array[v].head;

        printf("\n Adjacency list of vertex %d\n head ", v);

        while (pCrawl)

        {

            printf("-> %d", pCrawl->dest);

            pCrawl = pCrawl->next;

        }

        printf("\n");

    }

}

void BFS(struct Graph\* graph, int s)

{

    int v;

    int visited[graph->V];

    for (v = 0; v < graph->V; v++)

        visited[v] = 0;

    int \*queue = (int \*)malloc(sizeof(int)\*graph->V);

    visited[s] = 1;

    queue[0] = s;

    int i = 0;

    int j = 1;

    while (i < j)

    {

        int u = queue[i];

        printf("%d ", u);

        i++;

        struct AdjListNode\* pCrawl = graph->array[u].head;

        while (pCrawl)

        {

            int v = pCrawl->dest;

            if (!visited[v])

            {

                visited[v] = 1;

                queue[j] = v;

                j++;

            }

            pCrawl = pCrawl->next;

        }

    }

}

void DFS(struct Graph\* graph, int s)

{

    int v;

    int visited[graph->V];

    for (v = 0; v < graph->V; v++)

        visited[v] = 0;

    int \*stack = (int \*)malloc(sizeof(int)\*graph->V);

    visited[s] = 1;

    stack[0] = s;

    int i = 0;

    int j = 1;

    while (i < j)

    {

        int u = stack[i];

        printf("%d ", u);

        i++;

        struct AdjListNode\* pCrawl = graph->array[u].head;

        while (pCrawl)

        {

            int v = pCrawl->dest;

            if (!visited[v])

            {

                visited[v] = 1;

                stack[j] = v;

                j++;

            }

            pCrawl = pCrawl->next;

        }

    }

}

int main()

{

    int V = 5;

    struct Graph\* graph = createGraph(V);

    addEdge(graph, 0, 1);

    addEdge(graph, 0, 4);

    addEdge(graph, 1, 2);

    addEdge(graph, 1, 3);

    addEdge(graph, 1, 4);

    addEdge(graph, 2, 3);

    addEdge(graph, 3, 4);

    printGraph(graph);

    int s = 0;

    printf("\nBFS starting from vertex %d : ", s);

    BFS(graph, s);

    s = 0;

    printf("\nDFS starting from vertex %d : ", s);

    DFS(graph, s);

    return 0;

}

**Output:**

