Name – Shrinivas Hatyalikar
Div - CS-B
Roll no.- 24
PRN- 12110883

WAP to Perform Prims and Kruskals using adjacency list and adjacency matrix

Prims using Adjacency Matrix:

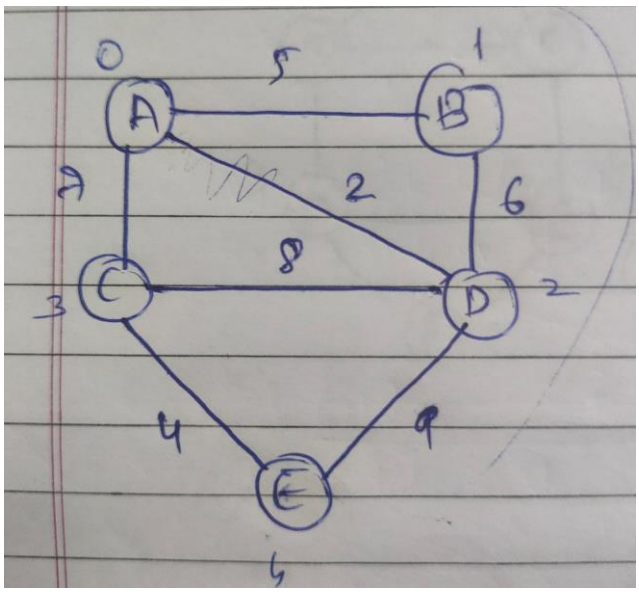
Code:

```
#include<stdio.h>
#include<conio.h>
#define max 9999
// #define v 6
int closevertex(int v,int weight[],int visited[]){
   int min=max;
   int vertex =0;

   for(int i=0;i<v;i++){
      if(visited[i]==0 && weight[i]<min){
         min = weight[i];
         vertex=i;
      }
   }
   return vertex;
}
void prims(int v,int G[][v]){
   int parent[v];
   int weight[v];
   int visited[v];

   for(int i=0;i<v;i++){
      weight[i]=max;
      visited[i]=0;
   }
```

```
    parent[0]=-1;
    weight[0]=0;

    for(int i=0;i<v-1;i++){

        int u = closevertex(v,weight,visited);
        visited[u]=1;

        for(int j=0;j<v;j++){
            /*3 conditions
                1) Edge should be there between u and j
                2) vertex j should not be visited
                3) Edge weight is smaller than current edge weight
            */
            if(G[u][j]!=0 && visited[j]!=1 && G[u][j]<weight[j]){
                weight[j]=G[u][j];
                parent[j]=u;
            }
        }
    }

    for(int i=0;i<v;i++){
        printf("%d %d %d\n",i,parent[i],weight[i]);
    }

}
int main(){
    int v, e;
    printf("Enter the number of vertices & edges: ");
    scanf("%d %d", &v, &e);
    int G[v][v];
    for (int i = 0; i < v; i++) {
        for (int j = 0; j < v; j++) {
            G[i][j] = max;
        }
    }
```

```
    printf("\nEnter the pairs of vertices having edges:\n");
    printf("NOTE: Vertex should start from 0");
    for (int i = 0; i < e; i++) {
        int u, v,w;
        printf("Enter vertex pair: ");
        scanf("%d %d", &u, &v);
        printf("Enter weight: ");
        scanf("%d",&w);
        G[u][v] = w;
        G[v][u] = w;
    }
    printf("\nAdjacency Matrix:\n");
    for (int i = 0; i < v; i++) {
        for (int j = 0; j < v; j++) {
            printf("%d\t", G[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    prims(v,G);
    return 0;
}
```

Input Graph:



Output:

```
Enter the number of vertices & edges: 5 7

Enter the pairs of vertices having edges:
NOTE: Vertex should start from 0
Enter vertex pair: 0 1
Enter weight: 5
Enter vertex pair: 0 3
Enter weight: 7
Enter vertex pair: 0 2
Enter weight: 2
Enter vertex pair: 3 2
Enter weight: 8
Enter vertex pair: 1 2
Enter weight: 6
Enter vertex pair: 3 4
Enter weight: 4
Enter vertex pair: 4 2
Enter weight: 9

Adjacency Matrix:
9999    5       2       7       9999
5       9999    6       9999    9999
2       6       9999    8       9
7       9999    8       9999    4
9999    9999    9       4       9999

0 -1 0
1 0 5
2 0 2
3 0 7
4 3 4
PS C:\Users\sheeh\OneDrive\Desktop\C>
```

Prims using Adjacency List:

Code:

```
//prims algorithm using adjacency list

//minimum spanning tree
//prim's algorithm

#include<stdio.h>
#include<malloc.h>
#define MAX 999

typedef struct graph{
   int vertex, weight;
   struct graph *next;
}graph;

graph *A[10];

void init(graph *A[], int n){
   int i;
   for(i = 0; i<n; i++){
      A[i] = NULL;
   }
}

void create(graph *A[]){
   int v1, v2, weight;
   graph* p;
   char ch;

   do {
      printf("\nEnter the edge: ");
      scanf("%d %d", &v1, &v2);

      printf("\nEnter the weight %d %d: ", v1, v2);
      scanf("%d", &weight);
```

```
        graph* new_node = (graph*)malloc(sizeof(graph));
        new_node->vertex = v2;
        new_node->weight = weight;
        new_node->next = NULL;

        p = A[v1];
        if (p == NULL) {
           A[v1] = new_node;
        } else {
           while (p->next != NULL) {
              p = p->next;
           }
           p->next = new_node;
        }

        new_node = (graph*)malloc(sizeof(graph));
        new_node->vertex = v1;
        new_node->weight = weight;
        new_node->next = NULL;

        p = A[v2];
        if (p == NULL) {
           A[v2] = new_node;
        } else {
           while (p->next != NULL) {
              p = p->next;
           }
           p->next = new_node;
        }

        printf("\nMore Edges (Y/N): ");
        scanf(" %c", &ch);

     } while (ch == 'Y' || ch == 'y');
}
```

```
int closeVertex(int weight[], int visited[], int v){
   int min = MAX, i, index;
   for(i = 0; i < v; i++){
      if(visited[i] == 0 && weight[i] < min){
         min = weight[i];
         index = i;
      }
   }
   return index;
}

void displayMST(int parent[], int weight[], int v){
   for(int i = 0; i < v; i++){
      printf("\n%d = Parent: %d & Weight: %d", i, parent[i], weight[i]);
   }
}


void primAlgo(graph *A[], int v){
   int i, j, parent[v], visited[v], weight[v];
   graph *p;
   for(i = 0; i < v; i++){
      parent[i] = -1;
      visited[i] = 0;
      weight[i] = MAX;
   }
   parent[0] = -1;
   weight[0] = 0;
   for(i = 0; i < v-1; i++){
      int u = closeVertex(weight, visited, v);
      p = A[u];
      visited[u] = 1;
      while(p!= NULL){
         if(visited[p->vertex] == 0 && weight[p->vertex] > p->weight){
            weight[p->vertex] = p->weight;
            parent[p->vertex] = u;
         }
```

```
            p = p->next;
        }
    }

    displayMST(parent,weight, v);
}

int main(){
    int n;
    printf("\nEnter the number of nodes: ");
    scanf("%d", &n);
    init(A, n);
    create(A);

    primAlgo(A, n);
}
```

Output:

```
Enter the number of nodes: 5

Enter the edge: 0 1

Enter the weight 0 1: 5

More Edges (Y/N): y

Enter the edge: 0 2

Enter the weight 0 2: 2

More Edges (Y/N): y

Enter the edge: 1 4

Enter the weight 1 4: 8

More Edges (Y/N): y

Enter the edge: 2 4

Enter the weight 2 4: 5

More Edges (Y/N): y

Enter the edge: 2 3

Enter the weight 2 3: 6

More Edges (Y/N): y
```

```
Enter the edge: 3 4

Enter the weight 3 4: 3

More Edges (Y/N): n

0 = Parent: -1 & Weight: 0
1 = Parent: 0 & Weight: 5
2 = Parent: 0 & Weight: 2
3 = Parent: 4 & Weight: 3
4 = Parent: 2 & Weight: 5
```

Kruskals Using Adjacency Matrix:

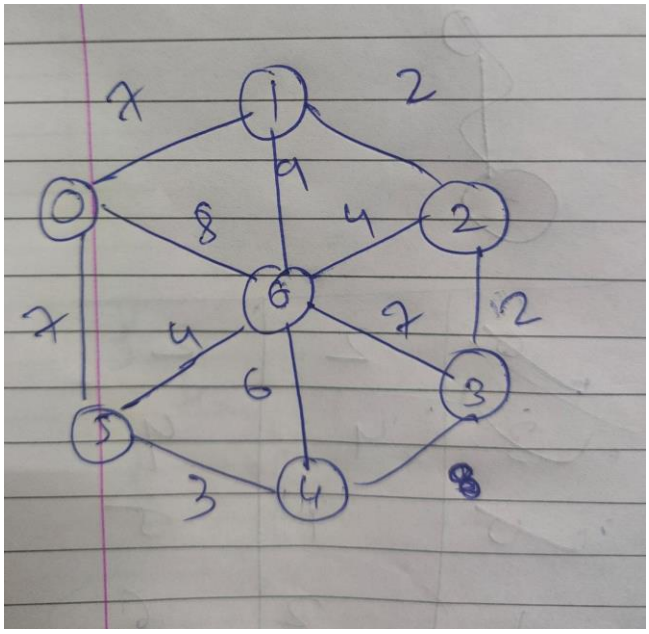
Code:

```
#include <stdio.h>
#include <conio.h>
#define max 999

int parent[100];

void initialize(int n) {
   for (int i = 0; i < n; i++) {
      parent[i] = 0;
   }
}

int findparent(int i){
   while(parent[i]){
      i=parent[i];
```

```
    }
    return i;
}

int ancestor(int u,int v){
    if(u!=v){
        parent[v]=u;
        return 1;
    }
    return 0;
}

void kruskals(int n,int G[][n]){
    int min;
    int ne=1;
    int a,b,v,u,minweight=0;
    initialize(n);
    while(ne<n){
        min = max;
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(G[i][j] < min){
                    min = G[i][j];
                    a=u=i;
                    b=v=j;
                }
            }
        }
        u=findparent(u);
        v=findparent(v);

        if(ancestor(u,v)){
            printf("%d edge (%d,%d) = %d\n",ne++,a,b,min);
            minweight+=min;
        }
        G[a][b] = G[b][a] = max;
    }
```

```
    printf("Minimum Weight = %d",minweight);

}
int main(){
    int v, e;
    printf("Enter the number of vertices & edges: ");
    scanf("%d %d", &v, &e);
    int G[v][v];
    for (int i = 0; i < v; i++) {
        for (int j = 0; j < v; j++) {
            G[i][j] = max;
        }
    }
    printf("\nEnter the pairs of vertices having edges:\n");
    printf("NOTE: Vertex should start from 0\n");
    for (int i = 0; i < e; i++) {
        int u, v,w;
        printf("Enter vertex pair: ");
        scanf("%d %d", &u, &v);
        printf("Enter weight: ");
        scanf("%d",&w);
        G[u][v] = w;
        G[v][u] = w;
    }
    printf("\nAdjacency Matrix:\n");
    for (int i = 0; i < v; i++) {
        for (int j = 0; j < v; j++) {
            printf("%d\t", G[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    kruskals(v,G);
    return 0;
}
```

Input Graph:



Output:

```
s_adj_matrix }
Enter the number of vertices & edges: 7 12

Enter the pairs of vertices having edges:
NOTE: Vertex should start from 0
Enter vertex pair: 0 1
Enter weight: 7
Enter vertex pair: 1 2
Enter weight: 2
Enter vertex pair: 2 3
Enter weight: 2
Enter vertex pair: 5 0
Enter weight: 7
Enter vertex pair: 0 6
Enter weight: 8
Enter vertex pair: 1 6
Enter weight: 9
Enter vertex pair: 2 6
Enter weight: 4
Enter vertex pair: 3 6
Enter weight: 7
Enter vertex pair: 4 6
Enter weight: 6
Enter vertex pair: 5 6
Enter weight: 4
```

```
Adjacency Matrix:
999     7       999     999     999     7       8
7       999     2       999     999     999     9
999     2       999     2       999     999     4
999     999     2       999     8       999     7
999     999     999     8       999     3       6
7       999     999     999     3       999     4
8       9       4       7       6       4       999

1 edge (1,2) = 2
2 edge (2,3) = 2
3 edge (4,5) = 3
4 edge (2,6) = 4
5 edge (5,6) = 4
6 edge (0,1) = 7
Minimum Weight = 22
PS C:\Users\sheeh\OneDrive\Desktop\C>
```

Kruskals Using Adjacency List:

Code:

```
//  Krushkals algorithm

#include<stdio.h>
#include<malloc.h>
#define MAX 999

typedef struct graph{
   int vertex, weight;
   struct graph *next;
}graph;

graph *A[10];

void init(graph *A[], int n){
   int i;
   for(i = 0; i<n; i++){
      A[i] = NULL;
```

```
    }
}

void create(graph *A[]){
    int v1, v2, weight;
    graph* p;
    char ch;

    do {
        printf("\nEnter the edge: ");
        scanf("%d %d", &v1, &v2);

        printf("\nEnter the weight %d %d: ", v1, v2);
        scanf("%d", &weight);

        graph* new_node = (graph*)malloc(sizeof(graph));
        new_node->vertex = v2;
        new_node->weight = weight;
        new_node->next = NULL;

        p = A[v1];
        if (p == NULL) {
            A[v1] = new_node;
        } else {
            while (p->next != NULL) {
                p = p->next;
            }
            p->next = new_node;
        }

        new_node = (graph*)malloc(sizeof(graph));
        new_node->vertex = v1;
        new_node->weight = weight;
        new_node->next = NULL;

        p = A[v2];
        if (p == NULL) {
            A[v2] = new_node;
        } else {
            while (p->next != NULL) {
```

```
            p = p->next;
        }
        p->next = new_node;
    }

    printf("\nMore Edges (Y/N): ");
    scanf(" %c", &ch);

  } while (ch == 'Y' || ch == 'y');
}

int findParent(int i,int parent[]){
  while(parent[i]!=-1){
    i=parent[i];
  }
  return i;
}

int ancestor(int u,int v,int parent[]){
  if(u!=v){
    parent[v]=u;
    return 1;
  }
  return 0;

}

void kruskals(graph *A[],int n){
  int weight=0;
  int parent[n];
  graph *p;
  for(int u = 0; u<n; u++){
    parent[u] = -1;
  }
  int ne=0,i,j, k= 0, y=0;
  int a,b,u,v,min;
  while(ne < n){
    min=MAX;
    for(i=0; i<n; i++){
      p = A[i];
```

```
            while(p!=NULL){
               if(p->weight < min){
                  min = p->weight;
                  a = u = i;
                  b = v = p->vertex;
               }
               p = p->next;
            }
         }

         u=findParent(u,parent);
         v=findParent(v,parent);
         int x=ancestor(a,b,parent);

         if(x){
            printf("%d: Edge:(%d - %d): %d \n",k++,a, b, min);
            weight+=min;
         }
         graph *r, *s;
         r = A[a];
         s = A[b];
         while(r->vertex != b){
            r = r->next;
         }
         while(s->vertex != a){
            s =s->next;
         }
         r->weight = s->weight = MAX;
         ne++;
      }

      printf("\nParent List: ");
      for(i=0;i<n;i++){
            printf("%d  ",parent[i]);
      }

      printf("\nOverall Weight: %d", weight);
   }

   void main(){
```

```
    int n;
    printf("\nEnter the number of nodes: ");
    scanf("%d", &n);
    init(A, n);
    create(A);


    kruskals(A,n);
}
```

Output:

```
Enter the number of nodes: 5

Enter the edge: 0 1

Enter the weight 0 1: 5

More Edges (Y/N): y

Enter the edge: 0 2

Enter the weight 0 2: 2

More Edges (Y/N): y

Enter the edge: 2 3

Enter the weight 2 3: 6

More Edges (Y/N): y

Enter the edge: 2 4

Enter the weight 2 4: 5

Enter the weight 3 4: 3

More Edges (Y/N): n
0: Edge:(0 - 2): 2
1: Edge:(3 - 4): 3
2: Edge:(0 - 1): 5
3: Edge:(2 - 4): 5
4: Edge:(2 - 3): 6

Parent List: -1  0  0  2  2
Overall Weight: 21
```