Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

JHANVI PAREKH

60009210033

CSE(Data Science)

Experiment 3

(Greedy Algorithm)

Aim: Implementation of Prims's & Kruskal's method.

Prim's algorithm:

Theory:

Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph?

Algorithm:

Step 1:

- Randomly choose any vertex.
- The vertex connecting to the edge having least weight is usually selected.

Step 2:

- Find all the edges that connect the tree to new vertices.
- Find the least weight edge among those edges and include it in the existing tree.
- If including that edge creates a cycle, then reject that edge and look for the next least weight edge.

Step 3:

- Keep repeating step-02 until all the vertices are included and Minimum Spanning Tree (MST) is obtained.

Example:

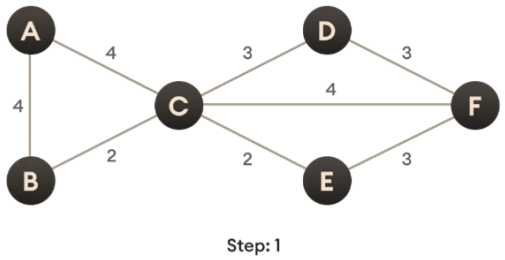
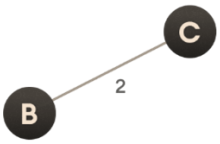


Figure 1. Start with a weighted graph

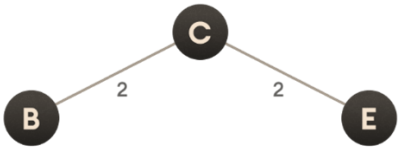
Step: 2

Figure 2.Choose a vertex



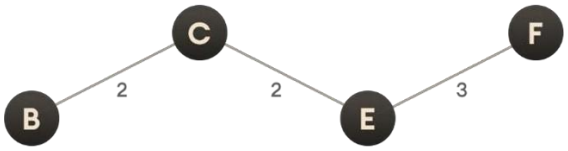
Step: 3

Figure 3.Choose the shortest edge from this vertex and add it



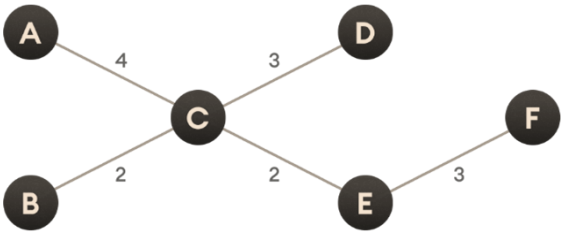
Step: 4

Figure 4.Choose the nearest vertex not yet in the solution



Step: 5

Figure 5.Choose the nearest edge not yet in the solution, if there are multiple choices, choose one at random



Step: 6

Figure 6.Repeat until you have a spanning tree

Complexity:

The time complexity of Prim's algorithm is O(E log V).

Kruskal's algorithm:

Theory:

Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

- form a tree that includes every vertex
- has the minimum sum of weights among all the trees that can be formed from the graph?

Algorithm:

Step 1:

- Sort all the edges from low weight to high weight.

Step 2:

- Take the edge with the lowest weight and use it to connect the vertices of graph.
- If adding an edge creates a cycle, then reject that edge and go for the next least weight edge.

Step 3:

- Keep adding edges until all the vertices are connected and a Minimum Spanning Tree (MST) is obtained.

Example:

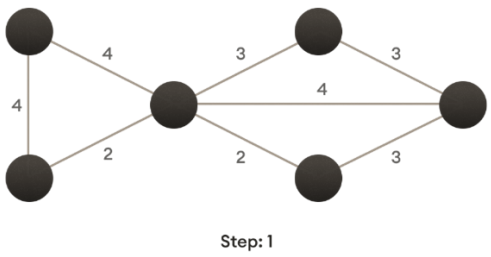


Figure 7. Start with a weighted graph

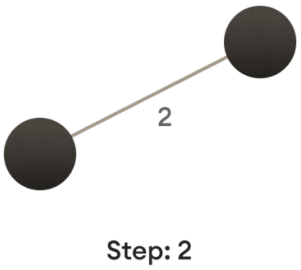


Figure 8. Choose the edge with the least weight, if there are more than 1, choose anyone

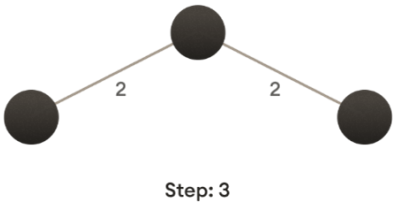


Figure 9. Choose the next shortest edge and add it

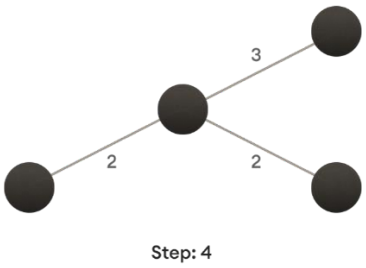


Figure 10. Choose the next shortest edge that doesn't create a cycle and add it

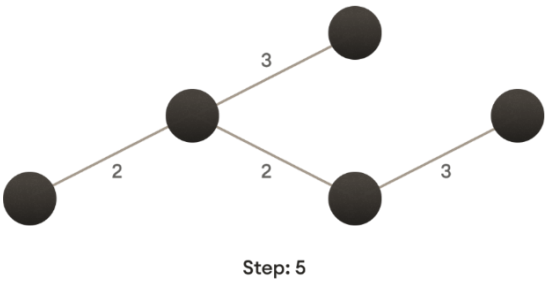


Figure 11. Choose the next shortest edge that doesn't create a cycle and add it

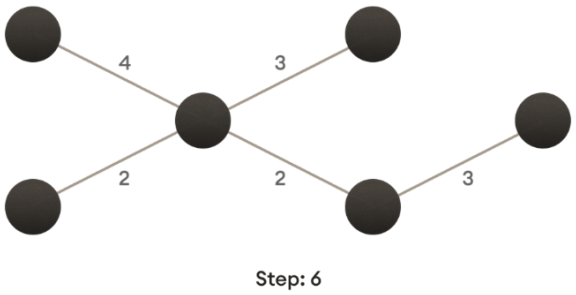


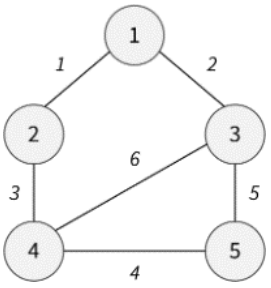
Figure 12.Repeat until you have a spanning tree

Complexity:

The time complexity Of Kruskal's Algorithm is: O(E log E).

Lab Assignment:

Write a C program to implement Prim's and Kruskal's Algorithm using greedy algorithm for the following graph.



Department of Computer Science and Engineering (Data Science)

PRIM'S ALGORITHM

CODE:

main.c

Run

```c
// Prim's Algorithm in C

#include<stdio.h>
#include<stdbool.h>

#define INF 9999999

// number of vertices in graph
#define V 5

// create a 2d array of size 5x5
//for adjacency matrix to represent graph
int G[V][V] = {
  {0, 1, 2, 0, 0 },
  {1, 0, 0, 3, 0},
  {2, 0, 0, 6, 5},
  {0, 3, 6, 0, 4},
  {0, 0, 5, 4, 0}};

int main() {
  int no_edge, MST_Cost=0;  // number of edge

  // create a array to track selected vertex
  // selected will become true otherwise false
  int selected[V];

  // set selected false initially
  memset(selected, false, sizeof(selected));

  // set number of edge to 0
  no_edge = 0;

  // the number of egde in minimum spanning tree will be
  // always less than (V -1), where V is number of vertices in
  //graph

  // choose 0th vertex and make it true
```

Department of Computer Science and Engineering (Data Science)

main.c

Run

```
// always less than (V -1), where V is number of vertices in
//graph

// choose 0th vertex and make it true
selected[0] = true;

int x;  //  row number
int y;  //  col number

// print for edge and weight
printf("Edge : Weight\n");

while (no_edge < V - 1) {
  //For every vertex in the set S, find the all adjacent vertices
  // , calculate the distance from the vertex selected at step 1.
  // if the vertex is already in the set S, discard it otherwise
  //choose another vertex nearest to selected vertex  at step 1.

  int min = INF;
  x = 0;
  y = 0;

  for (int i = 0; i < V; i++) {
    if (selected[i]) {
      for (int j = 0; j < V; j++) {
        if (!selected[j] && G[i][j]) {  // not in selected and there is an edge
          if (min > G[i][j]) {
            min = G[i][j];
            x = i;
            y = j;
          }
        }
      }
    }
  }
  printf("%d - %d : %d\n", x+1, y+1, G[x][y]);
```

Department of Computer Science and Engineering (Data Science)

main.c

Run

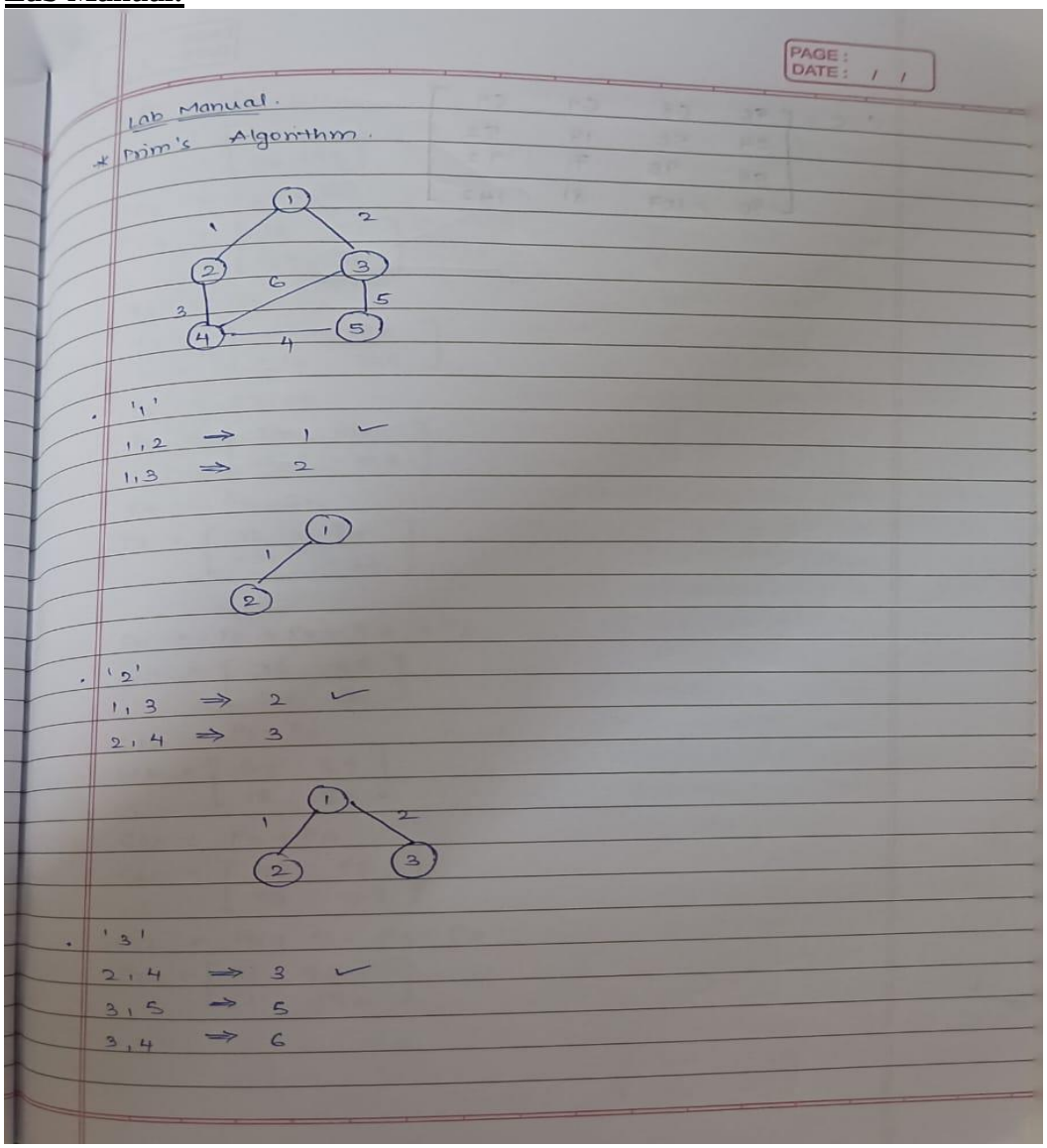
```
int x;  //  row number
int y;  //  col number

// print for edge and weight
printf("Edge : Weight\n");

while (no_edge < V - 1) {
  //For every vertex in the set S, find the all adjacent vertices
  // , calculate the distance from the vertex selected at step 1.
  // if the vertex is already in the set S, discard it otherwise
  //choose another vertex nearest to selected vertex  at step 1.

  int min = INF;
  x = 0;
  y = 0;

  for (int i = 0; i < V; i++) {
    if (selected[i]) {
      for (int j = 0; j < V; j++) {
        if (!selected[j] && G[i][j]) {  // not in selected and there is an edge
          if (min > G[i][j]) {
            min = G[i][j];
            x = i;
            y = j;
          }
        }
      }
    }
  }
  printf("%d - %d : %d\n", x+1, y+1, G[x][y]);
 MST_Cost+=G[x][y];
  selected[y] = true;
  no_edge++;
}
  printf("MST Cost= %d\n",MST_Cost);
return 0;
}
```

Department of Computer Science and Engineering (Data Science)

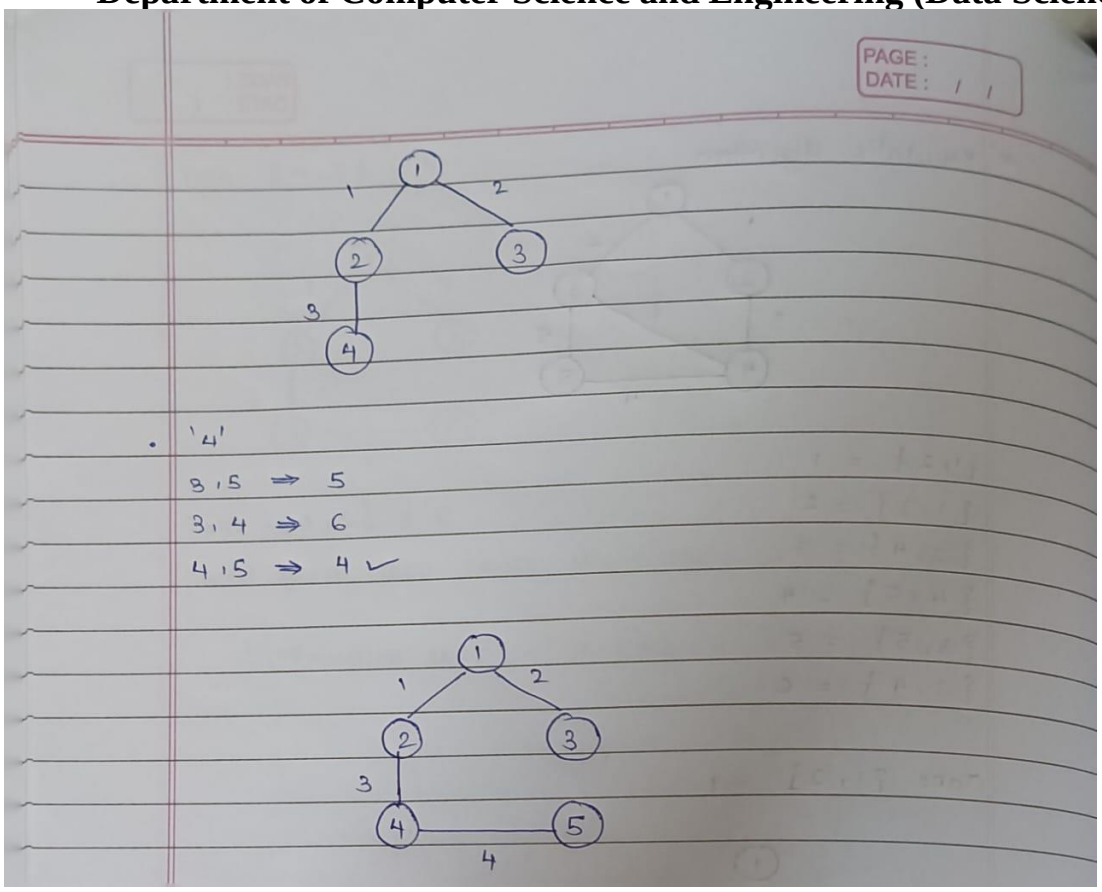
OUTPUT:

```
Output

/tmp/3DTqbYvknm.o
Edge : Weight
1 - 2 : 1
1 - 3 : 2
2 - 4 : 3
4 - 5 : 4
MST Cost= 10
```

Lab Manual:



Department of Computer Science and Engineering (Data Science)



Department of Computer Science and Engineering (Data Science)

KRUSKAL'S ALGORITHM

CODE:

main.c

Run

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_EDGES 1000
#define MAX_VERTICES 100

typedef struct edge {
    int src;
    int dest;
    int weight;
} Edge;

int parent[MAX_VERTICES];

void make_set(int x) {
    parent[x] = x;
}

int find_set(int x) {
    if (parent[x] != x) {
        parent[x] = find_set(parent[x]);
    }
    return parent[x];
}

void union_sets(int x, int y) {
    int px = find_set(x);
    int py = find_set(y);
    parent[px] = py;
}

int compare_edges(const void* a, const void* b) {
    Edge* ea = (Edge*)a;
    Edge* eb = (Edge*)b;
```

Department of Computer Science and Engineering (Data Science)

main.c

Run

```
34      Edge* eb = (Edge*)b;
35      return ea->weight - eb->weight;
36  }
37
38  void kruskal(Edge edges[], int num_edges, int num_vertices) {
39      int i, j, MST_Cost=0;
40      Edge e;
41      qsort(edges, num_edges, sizeof(Edge), compare_edges);
42      for (i = 0; i < num_vertices; i++) {
43          make_set(i);
44      }
45      for (i = 0; i < num_edges; i++) {
46          e = edges[i];
47          if (find_set(e.src) != find_set(e.dest)) {
48              union_sets(e.src, e.dest);
49              printf("(%d, %d) -> %d\n", e.src, e.dest, e.weight);
50              MST_Cost+=e.weight;
51          }
52      }
53      printf("MST Cost= %d\n",MST_Cost);
54  }
55
56  int main() {
57      int num_vertices, num_edges, i;
58      Edge edges[MAX_EDGES];
59      printf("Enter the number of vertices: ");
60      scanf("%d", &num_vertices);
61      printf("Enter the number of edges: ");
62      scanf("%d", &num_edges);
63      printf("Enter the edges as source, destination, weight:\n");
64      for (i = 0; i < num_edges; i++) {
65          printf("source- ");
66          scanf("%d", &edges[i].src);
67          printf("destination- ");
```

Department of Computer Science and Engineering (Data Science)

main.c

Run

```
    qsort(edges, num_edges, sizeof(Edge), compare_edges);
    for (i = 0; i < num_vertices; i++) {
        make_set(i);
    }
    for (i = 0; i < num_edges; i++) {
        e = edges[i];
        if (find_set(e.src) != find_set(e.dest)) {
            union_sets(e.src, e.dest);
            printf("(%d, %d) -> %d\n", e.src, e.dest, e.weight);
            MST_Cost+=e.weight;
        }
    }
    printf("MST Cost= %d\n",MST_Cost);
}

int main() {
    int num_vertices, num_edges, i;
    Edge edges[MAX_EDGES];
    printf("Enter the number of vertices: ");
    scanf("%d", &num_vertices);
    printf("Enter the number of edges: ");
    scanf("%d", &num_edges);
    printf("Enter the edges as source, destination, weight:\n");
    for (i = 0; i < num_edges; i++) {
        printf("source- ");
        scanf("%d", &edges[i].src);
        printf("destination- ");
        scanf("%d",&edges[i].dest);
        printf("Weight- ");
        scanf("%d",&edges[i].weight);
    }
    kruskal(edges, num_edges, num_vertices);
    return 0;
}
```

OUTPUT:

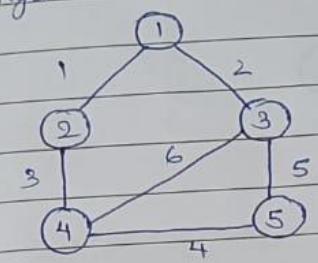
```
Output

/tmp/1p16BYcxab.o
Enter the number of vertices: 5
Enter the number of edges: 6
Enter the edges as source, destination, weight:
source- 1
destination- 2
Weight- 1
source- 1
destination- 3
Weight- 2
source- 2
destination- 4
Weight- 3
source- 3
destination- 5
Weight- 5
source- 3
destination- 4
Weight- 6
source- 4
destination- 5
Weight- 4
(1, 2) -> 1
(1, 3) -> 2
(2, 4) -> 3
(4, 5) -> 4
MST Cost= 10
```

Department of Computer Science and Engineering (Data Science)

LAB MANUAL:

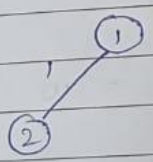
PAGE :
DATE : / /

* Kruskal's Algorithm.

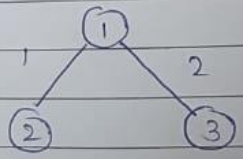


{1, 2} = 1
{1, 3} = 2
{2, 4} = 3
{4, 5} = 4
{3, 5} = 5
{3, 4} = 6

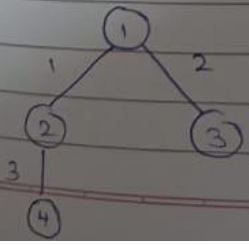
Take {1, 2} = 1



Take {1, 3} = 2



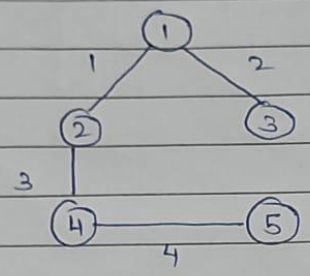
Take {2, 4} = 3



Department of Computer Science and Engineering (Data Science)

PAGE :
DATE : / /

Take {4, 5} = 4



Take {3, 5} = 5

will form loop hence discarded.

∴ Minimum cost = 1 + 2 + 3 + 4 = 10.