BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to Mumbai University)
Department Of Computer Engineering

Name	Manish Shashikant Jadhav
UID	2023301005
Subject	Design and Analysis of Algorithms (DAA)
Experiment No.	6
Aim	To implement Greedy Approach (Prim's Algorithm and Dijkstra's Algorithm)
Code:	

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

void primsAlgorithm(int **graph, int vertices) {
    int parent[vertices];
    int key[vertices];
    int mstSet[vertices];

    for (int i = 0; i < vertices; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < vertices - 1; count++) {
        int minKey = INT_MAX, minIndex;

        for (int v = 0; v < vertices; v++) {
            if (mstSet[v] == 0 && key[v] < minKey) {
                minKey = key[v];
                minIndex = v;
            }
        }

        mstSet[minIndex] = 1;
```

```
        for (int v = 0; v < vertices; v++) {
            if (graph[minIndex][v] && mstSet[v] == 0 &&
graph[minIndex][v] < key[v]) {
                parent[v] = minIndex;
                key[v] = graph[minIndex][v];
            }
        }
    }

    printf("Edge \tWeight\n");
    for (int i = 1; i < vertices; i++) {
        printf("%d - %d \t%d \n", parent[i], i,
graph[i][parent[i]]);
    }
}

void dijkstraAlgorithm(int **graph, int vertices, int src) {
    int dist[vertices];
    int visited[vertices];

    for (int i = 0; i < vertices; i++) {
        dist[i] = INT_MAX;
        visited[i] = 0;
    }

    dist[src] = 0;

    for (int count = 0; count < vertices - 1; count++) {
        int minDist = INT_MAX, minIndex;

        for (int v = 0; v < vertices; v++) {
            if (!visited[v] && dist[v] <= minDist) {
                minDist = dist[v];
                minIndex = v;
            }
        }
```

```
        visited[minIndex] = 1;

        for (int v = 0; v < vertices; v++) {
            if (!visited[v] && graph[minIndex][v] &&
dist[minIndex] != INT_MAX &&
                dist[minIndex] + graph[minIndex][v] < dist[v])
{
                dist[v] = dist[minIndex] + graph[minIndex][v];
            }
        }
    }

    printf("Vertex \tDistance from Source\n");
    for (int i = 0; i < vertices; i++) {
        printf("%d \t%d\n", i, dist[i]);
    }
}

int main() {
    int choice, vertices, src;

    printf("Enter the number of vertices: ");
    scanf("%d", &vertices);

    int **graph = (int **)malloc(vertices * sizeof(int *));
    for (int i = 0; i < vertices; i++) {
        graph[i] = (int *)malloc(vertices * sizeof(int));
    }

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
```

```
    do {
        printf("\nChoose an option:\n");
        printf("1. Prim's Algorithm\n");
        printf("2. Dijkstra's Algorithm\n");
        printf("3. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                primsAlgorithm(graph, vertices);
                break;
            case 2:
                printf("Enter the source vertex for Dijkstra's
Algorithm: ");
                scanf("%d", &src);
                if (src >= 0 && src < vertices) {
                    dijkstraAlgorithm(graph, vertices, src);
                } else {
                    printf("Invalid source vertex.\n");
                }
                break;
            case 3:
                printf("Execution Completed\n");
                break;
            default:
                printf("Invalid choice. Please enter
again.\n");
        }
    } while (choice != 3);

    for (int i = 0; i < vertices; i++) {
        free(graph[i]);
    }
    free(graph);

    return 0;
}
```

Output

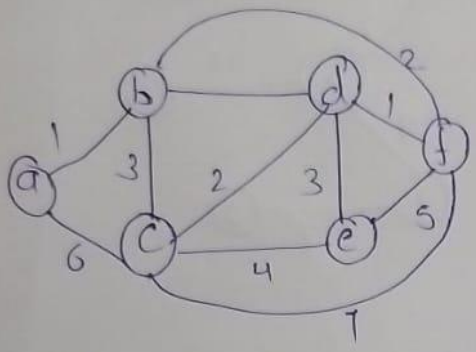
```
PS D:\Manish\SPIT> cd 'd:\Manish\SPIT\4th SEM\DAA\Exp6\output'
PS D:\Manish\SPIT\4th SEM\DAA\Exp6\output> & .\'greedy.exe'
Enter the number of vertices: 4
Enter the adjacency matrix:
0 1 3 0
1 0 1 4
3 1 0 2
0 4 2 0

Choose an option:
1. Prim's Algorithm
2. Dijkstra's Algorithm
3. Exit
1
Edge    Weight
0 - 1   1
1 - 2   1
2 - 3   2

Choose an option:
1. Prim's Algorithm
2. Dijkstra's Algorithm
3. Exit
2
Enter the source vertex for Dijkstra's Algorithm: 1
Vertex  Distance from Source
0       1
1       0
2       1
3       3

Choose an option:
1. Prim's Algorithm
2. Dijkstra's Algorithm
3. Exit
3
Execution Completed
PS D:\Manish\SPIT\4th SEM\DAA\Exp6\output>
```

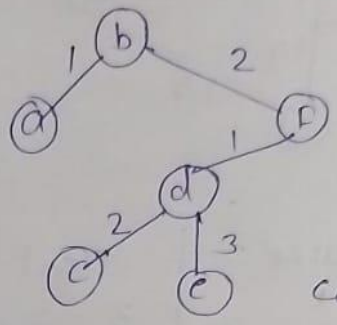
BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to Mumbai University)
Department Of Computer Engineering

Pseudo Code

* Prims Algorithm

Start 'a'.

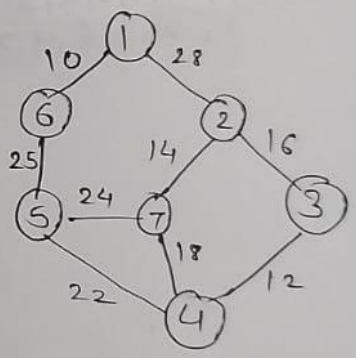




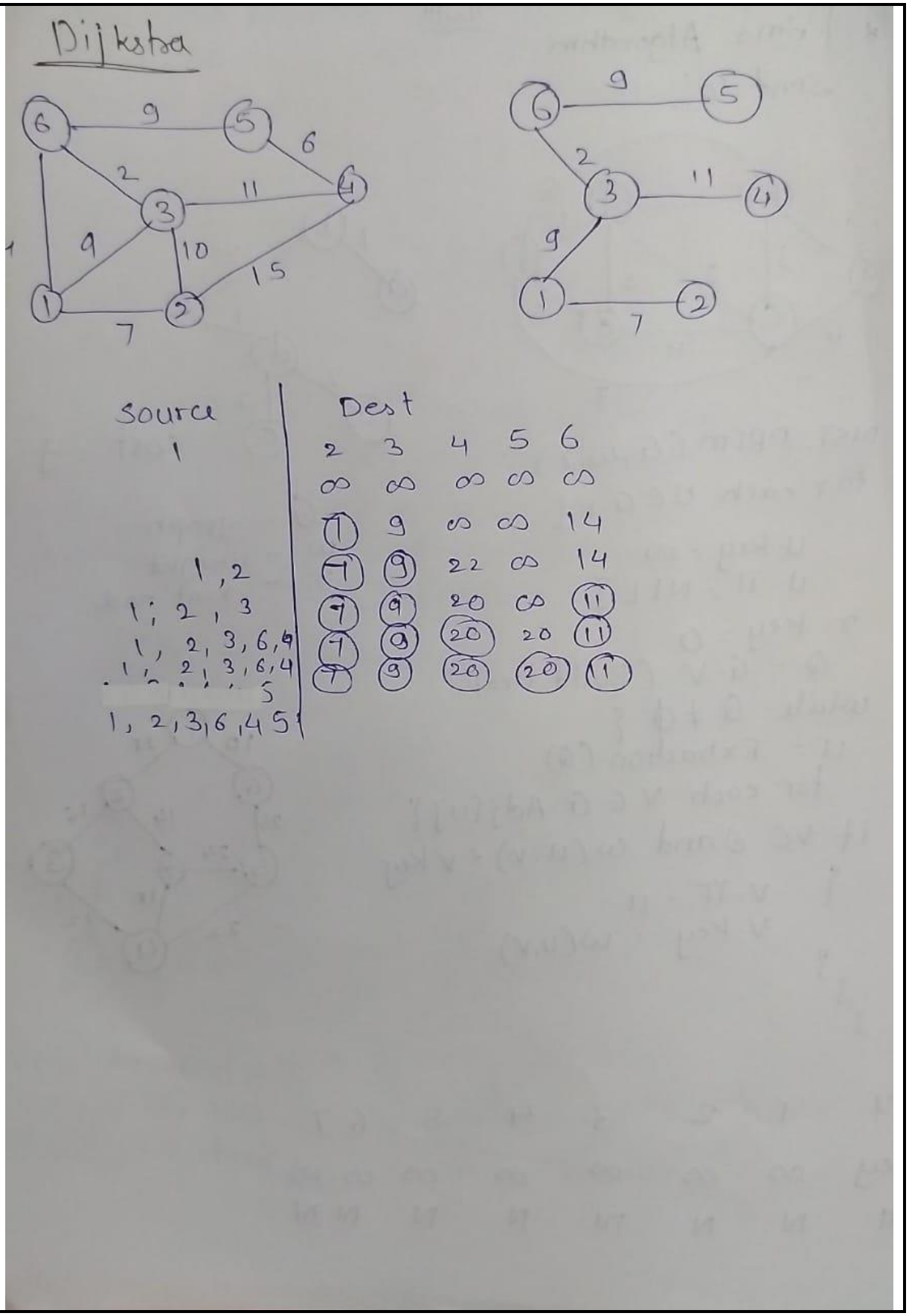
cost = 9

```
MST_PRIM (G, ω, r)
for each u ∈ G.V {
    u.key = ∞
    u.π = NIL }
r.key = 0
Q = G.V (Build Heap)
while Q ≠ ∅ {
  u = Extraction (Q)
  for each v ∈ G.Adj[u] {
if v ∈ Q and ω(u,v) < v.key
  { v.π = u
    v.key = ω(u,v)
  }
}
}
```

G – graph
ω – weight
r – root node



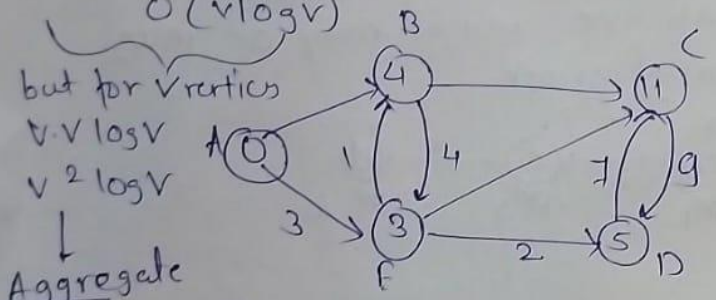
u	1	2	3	4	5	6	7
key	∞	∞	∞	∞	∞	∞	∞
π	N	N	N	N	N	N	N


Dijkstra
6
9
5
6
2
11
4
3
9
10
15
1
2
7
9
5
6
2
11
4
3
9
1
2
7
Source
1
Dest
2 3 4 5 6
∞ ∞ ∞ ∞ ∞
1 9 ∞ ∞ 14
1 9 22 ∞ 14
1 9 20 ∞ 11
1 9 20 20 11
1 9 20 20 11
1, 2
1, 2, 3
1, 2, 3, 6, 4
1, 2, 3, 6, 4
5
1, 2, 3, 6, 4 5

DIJKSTRA (G, w, s)
Initialize single source (G, s) → O(V)
s = ϕ O(1)
Q = G.V O(V)

while Q ≠ ϕ
  u = extract min(Q) O(log V)
  S = S ∪ {u}
1
O(V log V)
  for each vertex v ∈ G . Adj [u]
    Relax (u, v, w)
    ↓
    Decrease key
    O(V log V)

but for V vertices
V . V log V
V² log V
↓
Aggregate
E log V



S = {A, E, B, D, C}

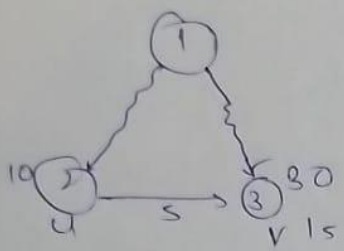
∴ TC = O(V) + O(1) + O(V) + O(log V) + O(V log V) + O(E log V)
= O((V + E) log V)

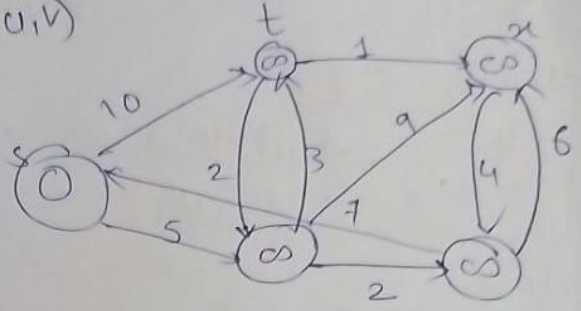
Dijkstra's Algorithm

Relaxation

if $d(v) > d(u) + w(u,v)$

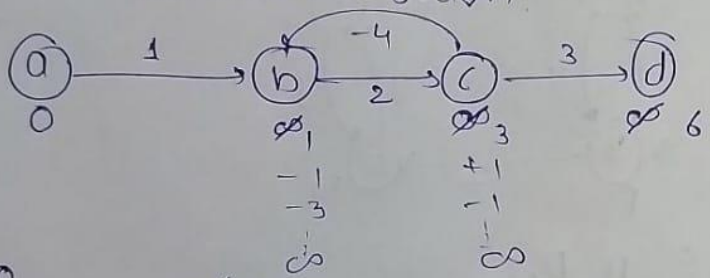
then $d(v) = d(u) + w(u,v)$

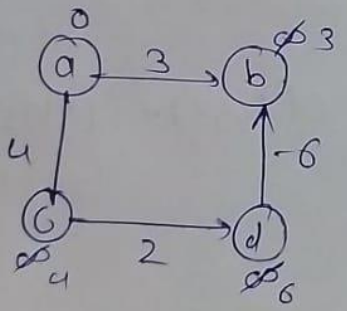




A graph with -ve weight cycle will always result in incorrect weight graph.

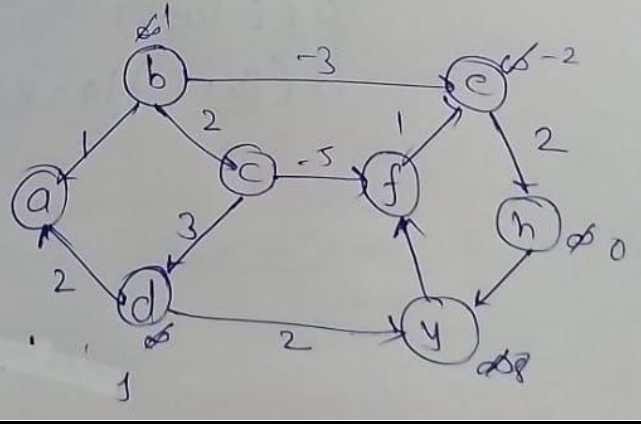
But a graph with -ve weight may or may not result in incorrect graph





Complete correct path

1) Even if the node is deleted still check by



Conclusion	Hence, by completing this experiment I came to know about implementation of Prims and Dijkestra algorithm.