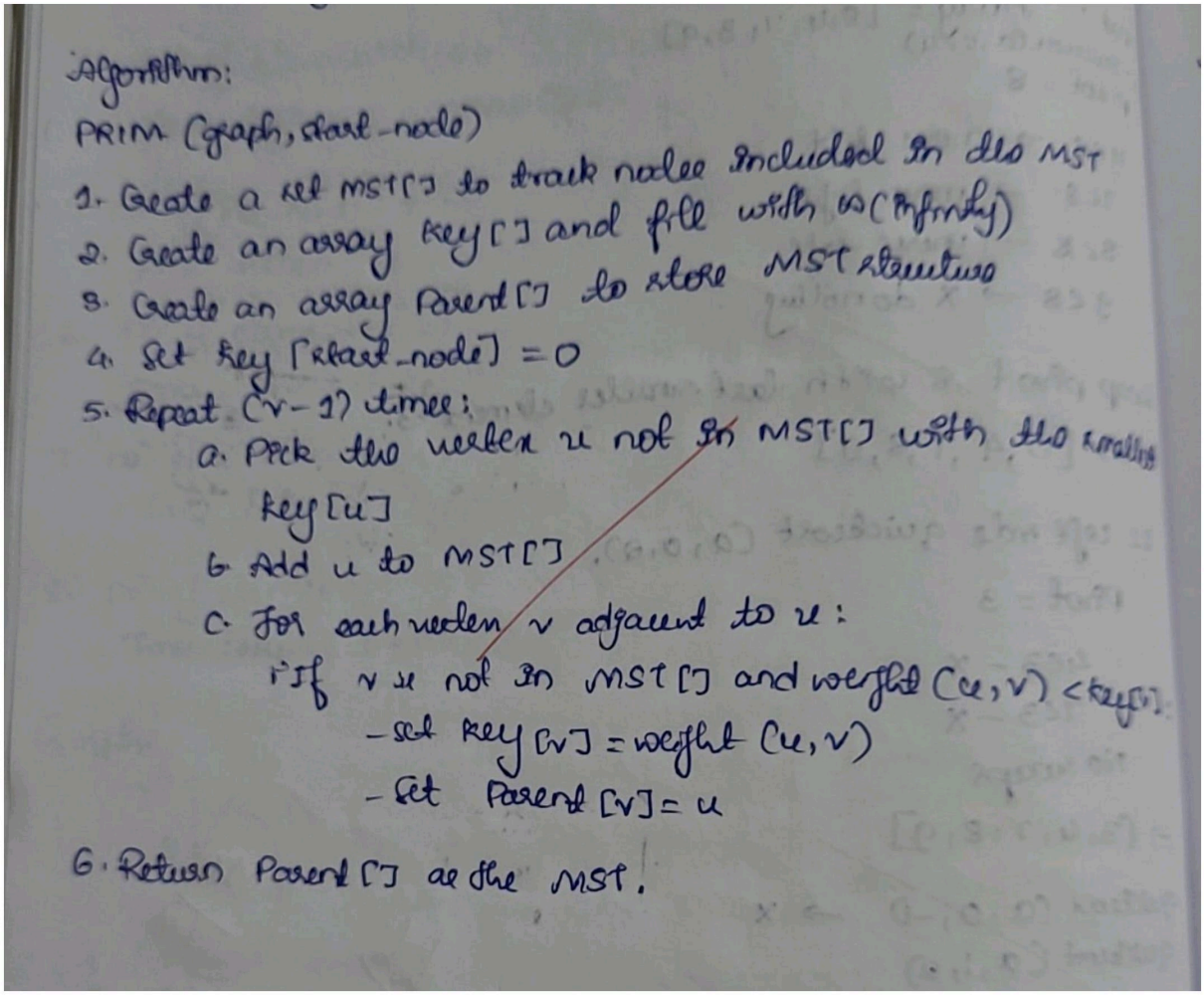
PROGRAM 3

3.1 Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

ALGORITHM



CODE

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

#define MAX 100

int findMinKey(int key[], int mstSet[], int n) {
    int min = INT_MAX, minIndex;
```

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

    return minIndex;
}

void primMST(int graph[MAX][MAX], int n) {
    int parent[MAX]; // stores MST
    int key[MAX];    // used to pick minimum weight edge
    int mstSet[MAX]; // included in MST

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

    key[0] = 0;      // Start from first vertex
    parent[0] = -1;  // First node is root

    for (int count = 0; count < n - 1; count++) {
        int u = findMinKey(key, mstSet, n);
        mstSet[u] = 1;

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

```
            }
        }
    }

    int totalCost = 0;
    printf("\nEdge \tWeight\n");
    for (int i = 1; i < n; i++) {
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
        totalCost += graph[i][parent[i]];
    }
    printf("\nTotal Cost of MST = %d\n", totalCost);
}

int main() {
    int n;
    int graph[MAX][MAX];

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

    printf("Enter the adjacency matrix (use 0 for no edge):\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    primMST(graph, n);

    return 0;
}
```

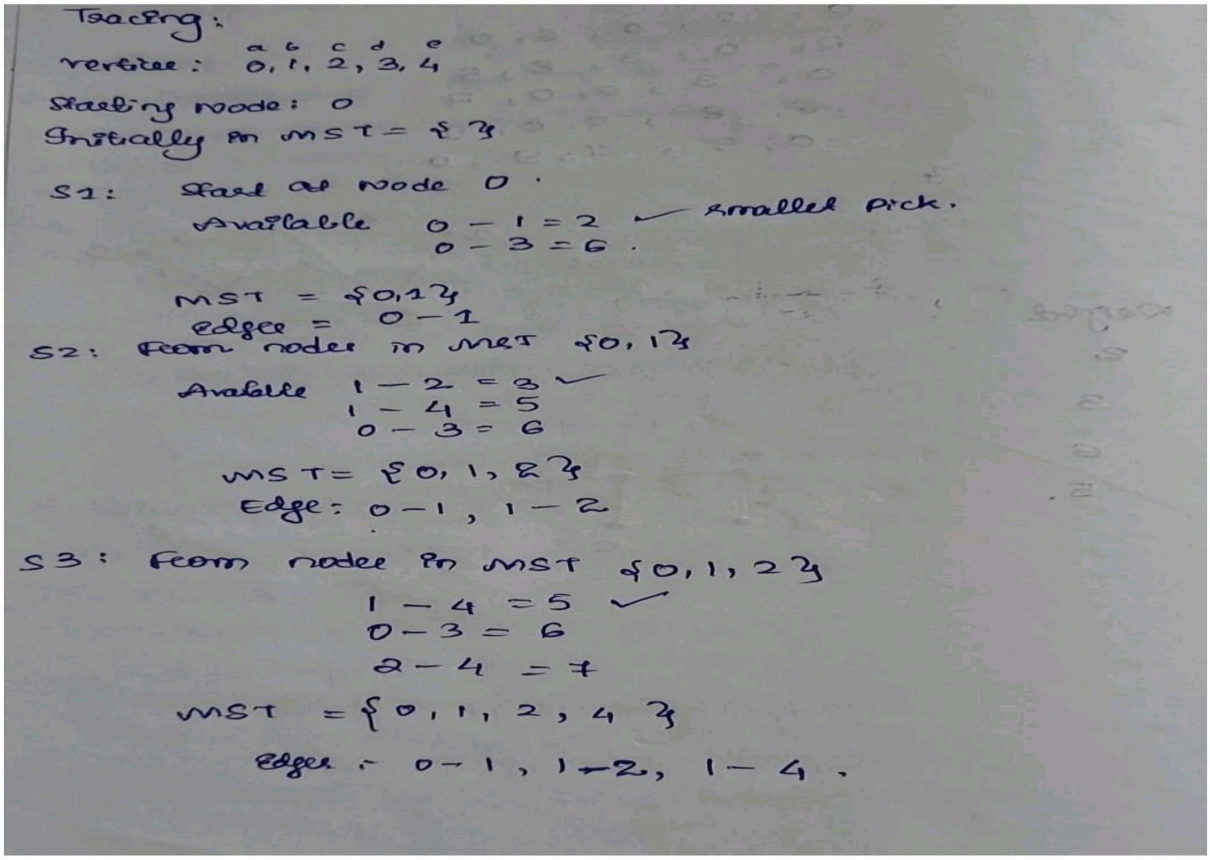
OUTPUT

```
Enter number of vertices: 5
Enter the adjacency matrix (use 0 for no edge):
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0

Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5

Total Cost of MST = 16
```

TRACING



S4:

From MST nodes {0, 1, 2, 4}

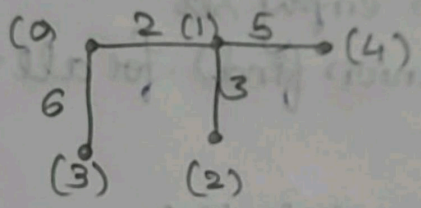
0 - 3 = 6 ✓

3 - 4 = 9

Pick 0-3

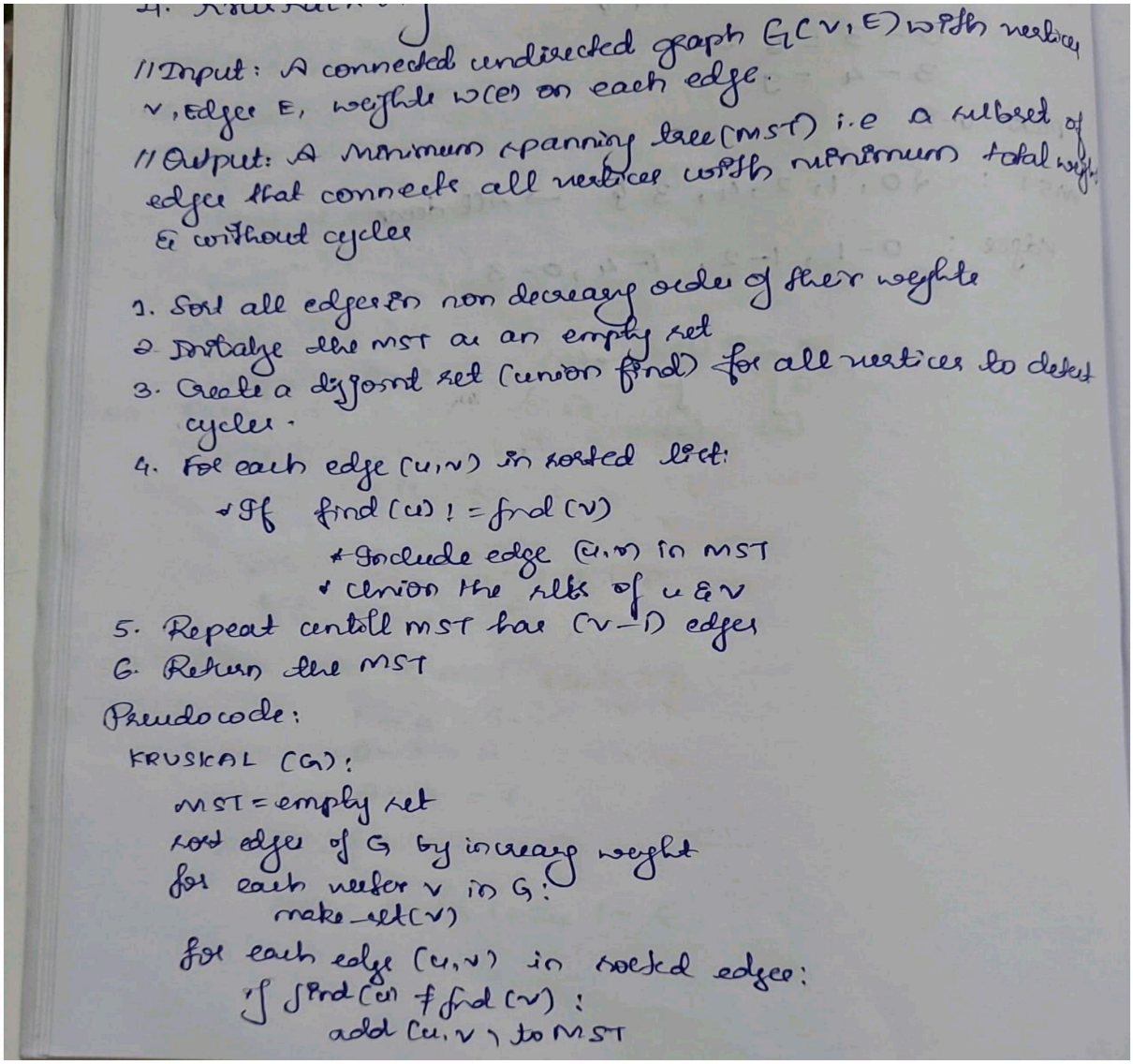
MST : {0, 1, 2, 4, 3} → All done

Edges : 0-1, 1-2, 1-4, 0-3



3.2 Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

ALGORITHM

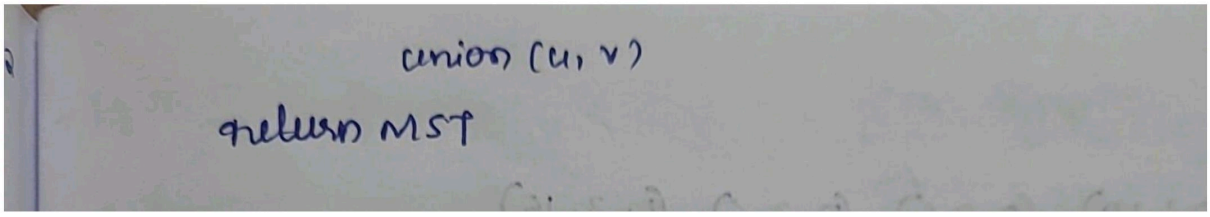


// Input: A connected undirected graph G(V, E) with vertices V, Edges E, weight w(e) on each edge.
// Output: A minimum spanning tree (MST) i.e a subset of edges that connects all vertices with minimum total weight & without cycles

1. Sort all edges in non decreasing order of their weights
2. Initialize the MST as an empty set
3. Create a disjoint set (union find) for all vertices to detect cycles.
4. For each edge (u, v) in sorted list:
   - If find(u) != find(v)
     - Include edge (u, v) in MST
     - Union the sets of u & v
5. Repeat until MST has (V − 1) edges
6. Return the MST

Pseudocode:

```
KRUSKAL (G):
   MST = empty set
   sort edges of G by increasing weight
   for each vertex v in G:
       make_set(v)
   for each edge (u, v) in sorted edges:
       if find(u) ≠ find(v):
           add (u, v) to MST
```



union (u, v)

return MST

CODE

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

#define MAX 100

// Structure to represent an edge
struct Edge {
    int src, dest, weight;
};

// Structure to represent a graph
struct Graph {
    int V, E;
    struct Edge edge[MAX];
};

// Find parent of a node (with path compression)
int find(int parent[], int i) {
    if (parent[i] != i)
        parent[i] = find(parent, parent[i]);
    return parent[i];
}

// Union of two sets
void Union(int parent[], int x, int y) {
    parent[x] = y;
}

// Comparator for sorting edges by weight
```

```
int compare(const void* a, const void* b) {
    struct Edge* e1 = (struct Edge*)a;
    struct Edge* e2 = (struct Edge*)b;
    return e1->weight - e2->weight;
}

// Kruskal's algorithm
void KruskalMST(struct Graph* graph) {
    int V = graph->V;
    struct Edge result[MAX];
    int parent[MAX];
    int e = 0; // count of edges in MST
    int i = 0;

    // Initially each node is its own parent
    for (int v = 0; v < V; v++)
        parent[v] = v;

    // Sort edges by weight
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), compare);

    while (e < V - 1 && i < graph->E) {
        struct Edge next = graph->edge[i++];
        int x = find(parent, next.src);
        int y = find(parent, next.dest);

        if (x != y) {
            result[e++] = next;
            Union(parent, x, y);
        }
```

```
    }

    int totalCost = 0;
    printf("\nEdge \tWeight\n");
    for (i = 0; i < e; ++i) {
        printf("%d - %d \t%d\n", result[i].src, result[i].dest, result[i].weight);
        totalCost += result[i].weight;
    }
    printf("\nTotal Cost of MST = %d\n", totalCost);
}

int main() {
    struct Graph graph;
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &graph.V, &graph.E);

    printf("Enter each edge as: src dest weight\n");
    for (int i = 0; i < graph.E; i++) {
        scanf("%d %d %d", &graph.edge[i].src, &graph.edge[i].dest, &graph.edge[i].weight);
    }

    KruskalMST(&graph);

    return 0;
}
```

OUTPUT

```
Enter number of vertices and edges: 4 5
Enter each edge as: src dest weight
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4

Edge    Weight
2 - 3   4
0 - 3   5
0 - 1   10

Total Cost of MST = 19
```

TRACING