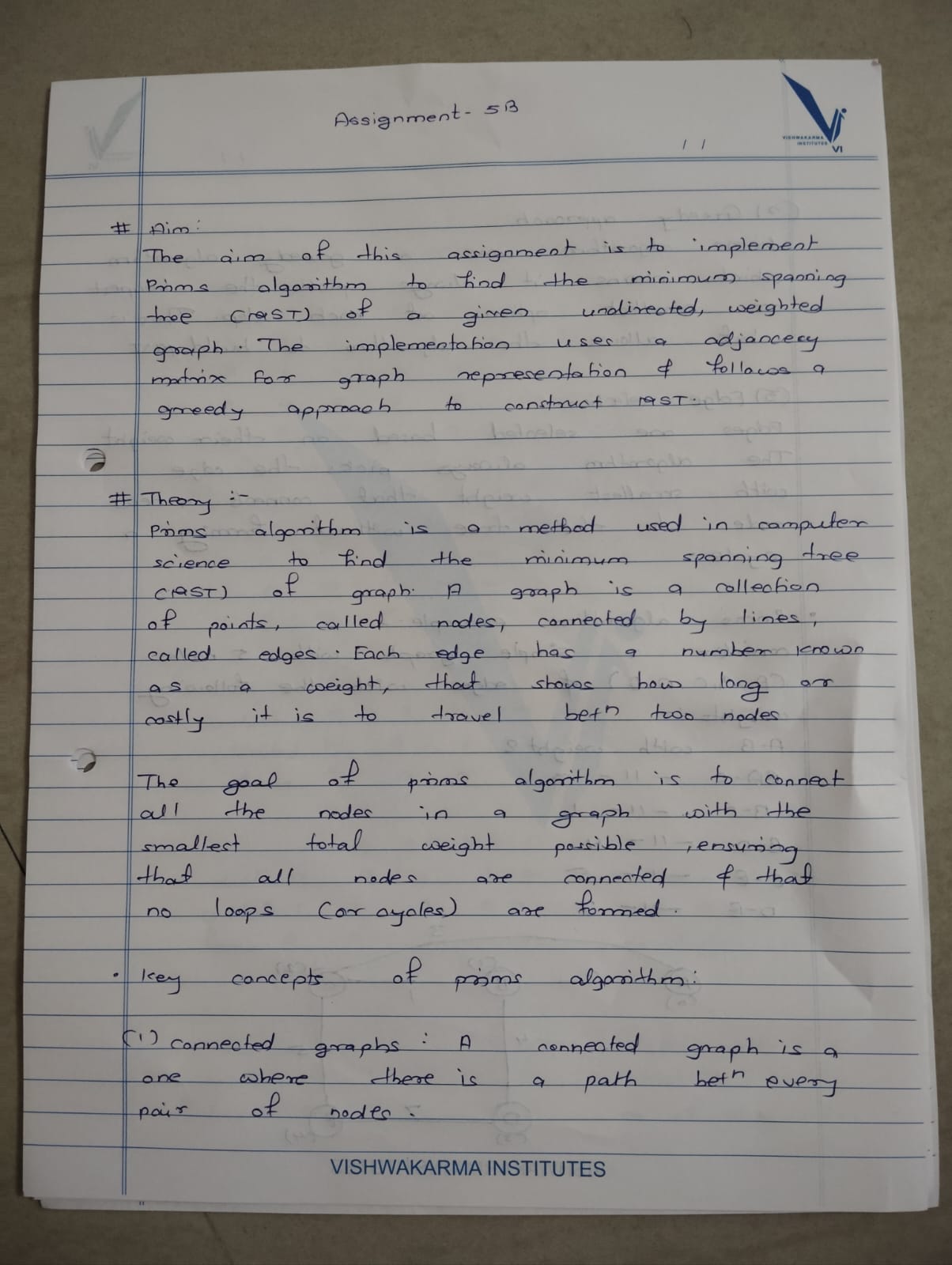
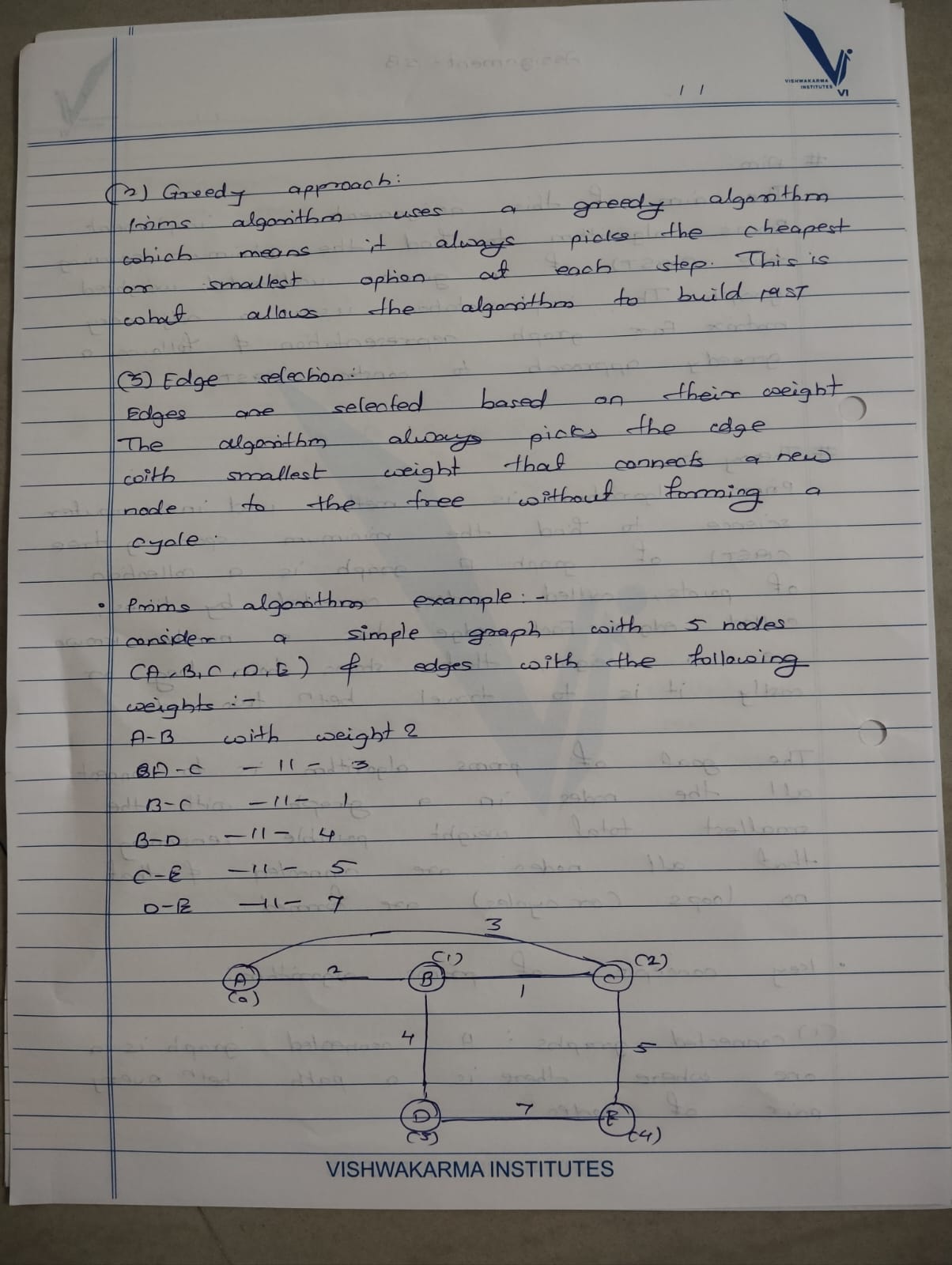
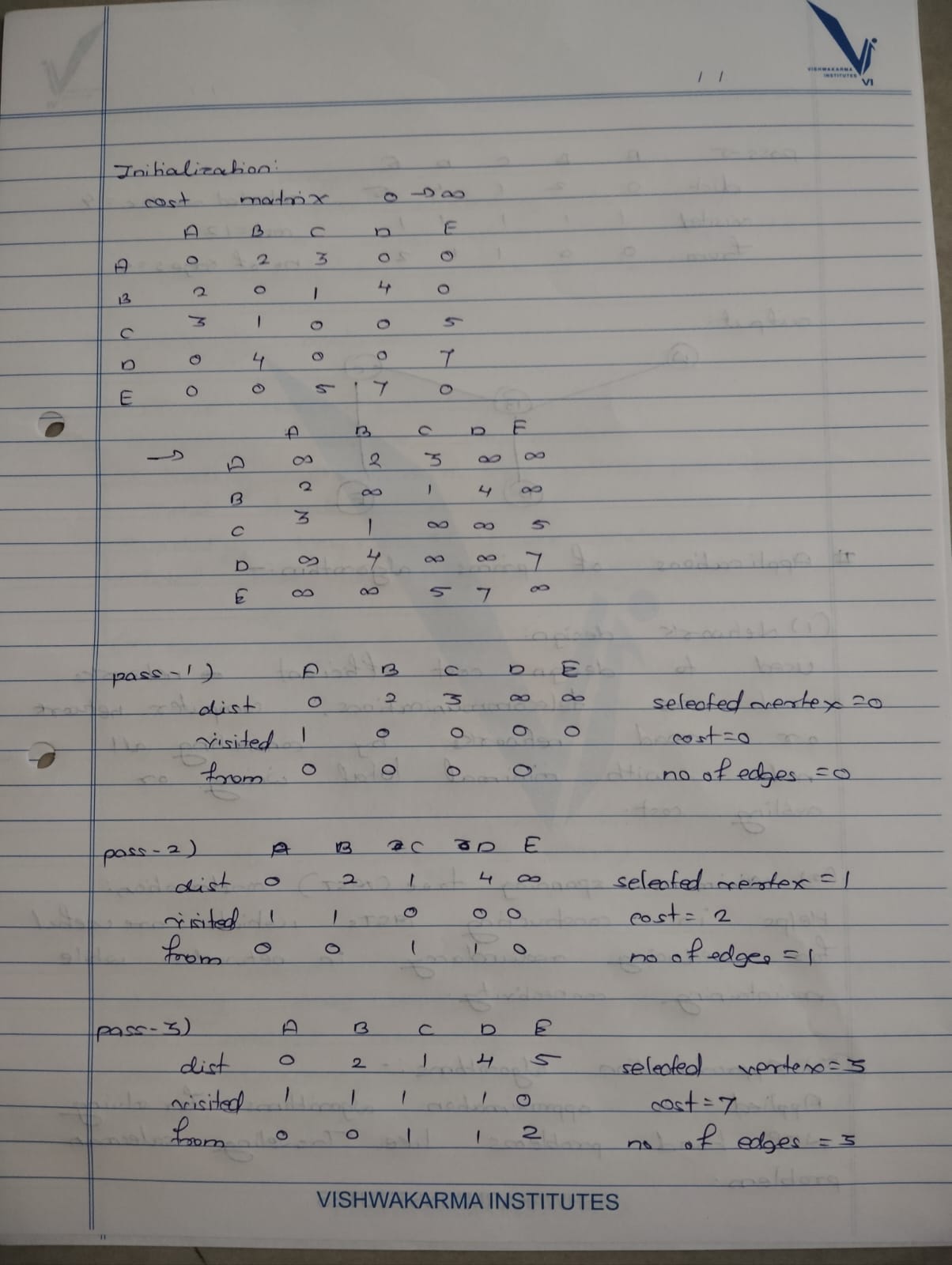
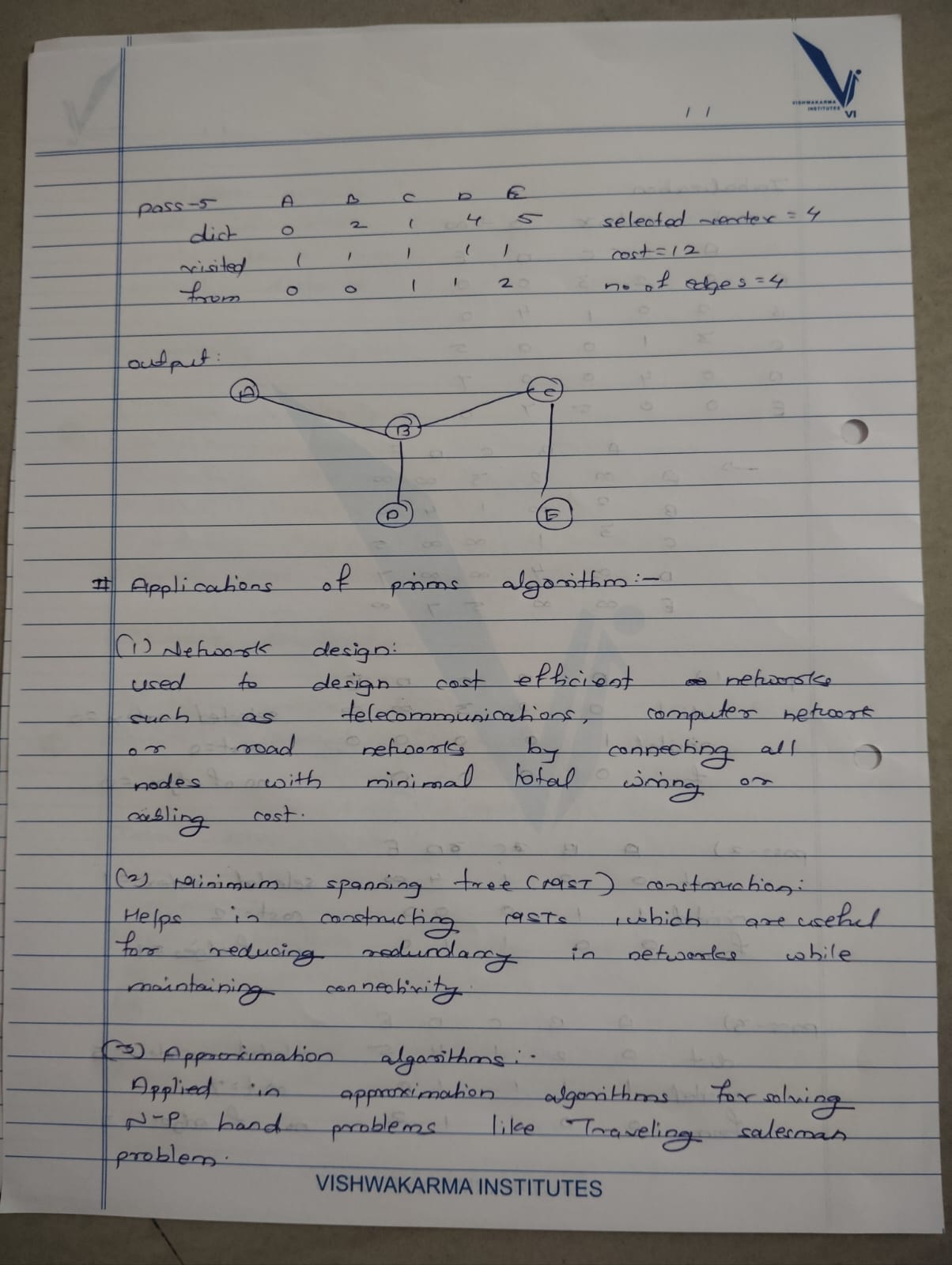
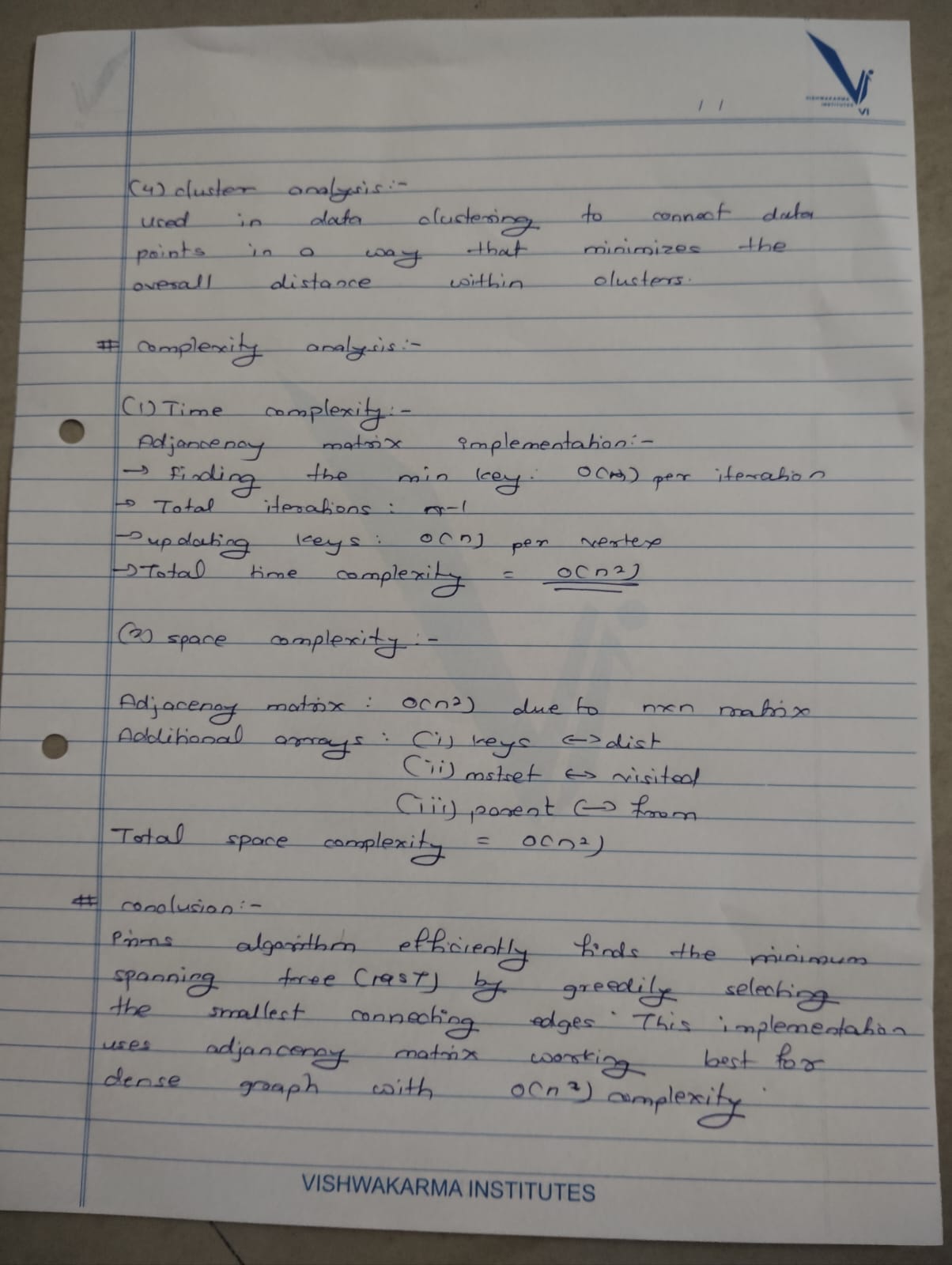
****

****

****

****

****

**Name:** Rohan Arun Nalawade

**Roll No:** 231012

**PRN:** 22310407

**SY IT A1**

**Assignment 5(B)**

**Code Implementation:**

#include <iostream>

#include <algorithm>

#include <climits>

#include <vector>

using namespace std;

#define MAX\_OFFICES 100

struct Edge {

    int u, v, weight;

    Edge(int u, int v, int weight) : u(u), v(v), weight(weight) {}

    bool operator<(const Edge& other) const {

        return weight < other.weight;

    }

};

class DisjointSet {

public:

    int parent[MAX\_OFFICES], rank[MAX\_OFFICES];

    DisjointSet(int n) {

        for (int i = 0; i < n; i++) {

            parent[i] = i;

            rank[i] = 0;

        }

    }

    int find(int x) {

        if (parent[x] != x)

            parent[x] = find(parent[x]);

        return parent[x];

    }

    bool union\_sets(int x, int y) {

        int rootX = find(x);

        int rootY = find(y);

        if (rootX != rootY) {

            if (rank[rootX] > rank[rootY])

                parent[rootY] = rootX;

            else if (rank[rootX] < rank[rootY])

                parent[rootX] = rootY;

            else {

                parent[rootY] = rootX;

                rank[rootX]++;

            }

            return true;

        }

        return false;

    }

};

int kruskal(int n, vector<Edge>& edges) {

    DisjointSet ds(n);

    int totalCost = 0;

    sort(edges.begin(), edges.end());

    int edgeCount = 0;

    for (int i = 0; i < edges.size(); i++) {

        int u = edges[i].u;

        int v = edges[i].v;

        int weight = edges[i].weight;

        if (ds.union\_sets(u, v)) {

            totalCost += weight;

            edgeCount++;

        }

        if (edgeCount == n - 1)

            break;

    }

    return totalCost;

}

int main() {

    int n, m;

    cout << "Enter number of offices (nodes): ";

    cin >> n;

    cout << "Enter number of connections (edges): ";

    cin >> m;

    vector<Edge> edges;

    cout << "Enter the edges (u, v, weight):\n";

    for (int i = 0; i < m; i++) {

        int u, v, weight;

        cin >> u >> v >> weight;

        edges.push\_back(Edge(u, v, weight));

    }

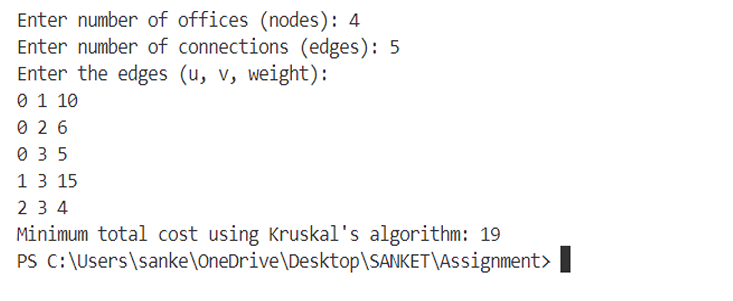
    int minCost = kruskal(n, edges);

    cout << "Minimum total cost using Kruskal's algorithm: " << minCost << endl;

    return 0;

}

**Output:**

****