MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
UNIVERSITATEA TEHNICĂ A MOLDOVEI
FACULTATEA CALCULATOARE INFORMATICĂ ȘI MICROELECTRONICĂ
TEHNOLOGIA INFORMAȚIEI

RAPORT

DISCIPLINA: Analiza și proiectarea algoritmilor (APA)

LUCRAREA DE LABORATOR NR. 3

TEMA: "Algoritmi greedy."

Student: _____ **Raevschi Grigore TI-231**

Coordonator: _____ **Asistent univ. Coșer Cătălin**

Chișinău 2024

Cuprins

Scopul lucrării:

1. Studierea tehnicii greedy.
2. Analiza şi implementarea algoritmilor greedy.

Sarcina de bază:

1. De studiat tehnica greedy de proiectare a algoritmilor.
2. De implementat într-un limbaj de programare algoritmii Kruskal, Prim.
3. De făcut analiza empirică a algoritmilor Kruskal şi Prim.
4. De alcătuit un raport.

1 Introducere:

Metoda de programare Greedy se aplică problemelor de optimizare. Aceasta metoda constă în faptul că se construieşte soluția optimă pas cu pas, la fiecare pas fiind selectat în soluție elementul care pare „cel mai bun/cel mai optim" la momentul respectiv, în speranța că această alegere locală va conduce la optimul global.

Algoritmii Greedy sunt foarte eficienți, dar nu conduc în mod necesar la o soluție optimă. Şi nici nu este posibilă formularea unui criteriu general conform căruia să putem stabili exact dacă metoda Greedy rezolvă sau nu o anumită problemă de optimizare. Din acest motiv, orice algoritm Greedy trebuie însoțit de o demonstrație a corectitudinii sale . Demonstrația faptului că o anumită problemă are proprietatea alegerii Greedy se face de obicei prin inducție matematică.

Metoda Greedy se aplică problemelor pentru care se dă o mulţime A cu n elemente şi pentru care trebuie determinată o submulţime a sa, S cu m elemente, care îndeplinesc anumite condiţii, numite si condiții de optime.

Principiu (strategia) Greedy :

— adaugă succesiv la rezultat elementul care realizează optimul local
— decizie luată pe parcurs nu se mai modifică ulterior

1.1 Algoritm Greedy:

```
GreedyAlgorithm(input):
    sort(input) // Sortează elementele în funcție de o anumită metrică
    solution = [] // Soluția finală

    for each element in input:
        if isFeasible(element, solution): // Verifică dacă elementul poate fi adăugat
            solution.add(element) // Adaugă elementul în soluție

    return solution
```

1.2 Principiul metodei Greedy:

Algoritmii greedy (sau „algoritmi lacomi”) sunt o familie de algoritmi care fac alegeri optime locale în scopul de a găsi o soluție global optimă. Aceștia sunt utilizați în diverse probleme, cum ar fi găsirea arborelui de acoperire minim, problemele de rucsac, programarea intervalelor și multe altele.

2 Arbori parțiali de cost minim: Kruskal, Prim

Atât în algoritmul lui Prim cât și în algoritmul lui Kruskal putem identifica elementele generale specifice metodei Greedy astfel:

- — există o mulțime inițială AA din care se aleg elementele care vor compune soluția (AA este
- — mulțimea muchiilor grafului);
- — fiecărui element al mulțimii AA îi este asociată o valoare numerică (ponderea muchiei);
- — mulțimea inițială se sortează crescător, în ordinea ponderilor asociate muchiilor;
- — din mulțimea AA se aleg, în ordine, primele $mm - 1$ elemente care nu formează cicluri
- — (criteriul de alegere).

3 Algoritmului Kruskal:

În acest raport, vom explora principiile de bază ale algoritmului Kruskal, pașii săi esențiali și aplicabilitatea sa în diverse domenii, cum ar fi rețelele de calculatoare, planificarea infrastructurii și optimizarea resurselor. De asemenea, vom analiza complexitatea temporală a algoritmului și vom compara eficiența acestuia cu alte metode de calculare a arborelui de acoperire minim, precum algoritmul Prim. Această evaluare va evidenția relevanța algoritmului Kruskal în rezolvarea problemelor practice și teoretice din domeniul informaticii și al ingineriei.

Algoritmul Kruskal este utilizat pentru a determina arborele de acoperire minim (MST - Minimum Spanning Tree) al unui graf neorientat. Un arbore de acoperire minim este un subgraf care conține toate vârfurile originale și are suma minimă a greutăților muchiilor. Algoritmul funcționează pe baza principiului de selecție a celor mai ușoare muchii, asigurându-se că nu se formează cicluri.

Algoritmul programului în **ANEXA A**.

3.1 Complexitate

Complexitatea de timp:

- Sortarea muchiilor are o complexitate de $O(Elog\ E)$, unde EE este numărul de muchii. Operațiile de unire și căutare folosind Union-Find au o complexitate aproape constantă, $O(\alpha(V))$, cu αα fiind funcția inversă a lui Ackermann.
- Prin urmare, complexitatea totală a algoritmului Kruskal este $O(Elog\ E)$.

Complexitatea de spațiu:

- Algoritmul necesită spațiu pentru stocarea muchiilor și a structurii de date Union-Find, deci complexitatea de spațiu este $O(V + E)$.

3.2 Avantaje și Dezavantaje

Avantaje:

- **Simplicity**: Algoritmul este intuitiv și ușor de implementat.
- **Eficiență**: Este adesea mai eficient decât alte metode pentru grafuri sparse.

Dezavantaje:

- **Sortarea**: Procesul de sortare a muchiilor poate fi costisitor pentru grafuri cu un număr mare de muchii.
- **Grafuri Dense**: În cazul grafurilor dense, alte algoritmi, cum ar fi Prim, pot avea performanțe mai bune.

4 Algoritmul lui Prim

Algoritmul lui Prim este un alt algoritm fundamental folosit pentru a găsi arborele de acoperire minim (MST - Minimum Spanning Tree) al unui graf neorientat. Spre deosebire de algoritmul Kruskal, care selectează muchii în funcție de greutate, Prim construiește arborele de acoperire minim începând cu un nod și adăugând treptat cele mai ușoare muchii care extind arborele existent.

Algoritmul programului în **ANEXA B**.

4.1 Complexitate

Complexitatea de timp:

- Algoritmul poate fi implementat eficient utilizând o structură de date precum un **min-heap** (tas minim). În acest caz, complexitatea de timp devine $O(Elog\ V)$, unde EE este numărul de muchii și VV este numărul de vârfuri.
- O altă implementare, utilizând matricea de adiacență, are o complexitate de $O(V^2)$.

Complexitatea de spațiu:

- Algoritmul necesită spațiu pentru a stoca graful și pentru structura de date utilizată, deci complexitatea de spațiu este $O(V + E)$.

4.2 Avantaje și Dezavantaje

Avantaje:

- **Eficiență**: Algoritmul lui Prim este adesea mai eficient pentru grafuri dense, unde numărul de muchii este aproape de maximul posibil.

— **Ușurința de Implementare**: Algoritmul este intuitiv și relativ simplu de implementat, în special cu ajutorul structurilor de date moderne.

Dezavantaje:

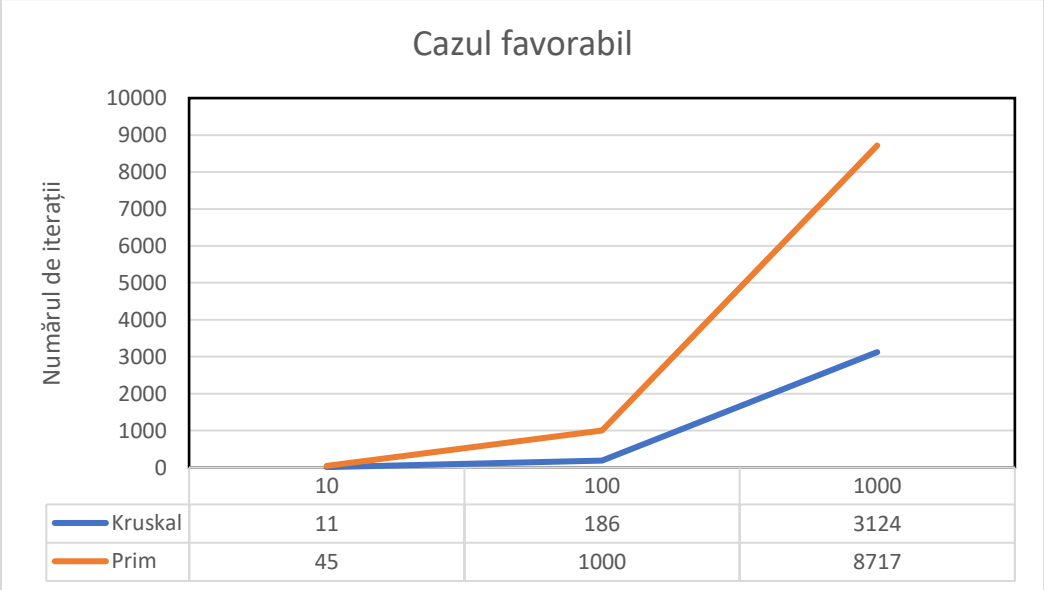
— **Dependent de Graf**: Performanța sa poate scădea în cazul grafurilor sparse, unde Kruskal ar putea fi mai eficient.

— **Necesită Structuri de Date Avansate**: Utilizarea eficientă a min-heap-ului sau a altor structuri de date poate complica implementarea.

5 Analiza empirică

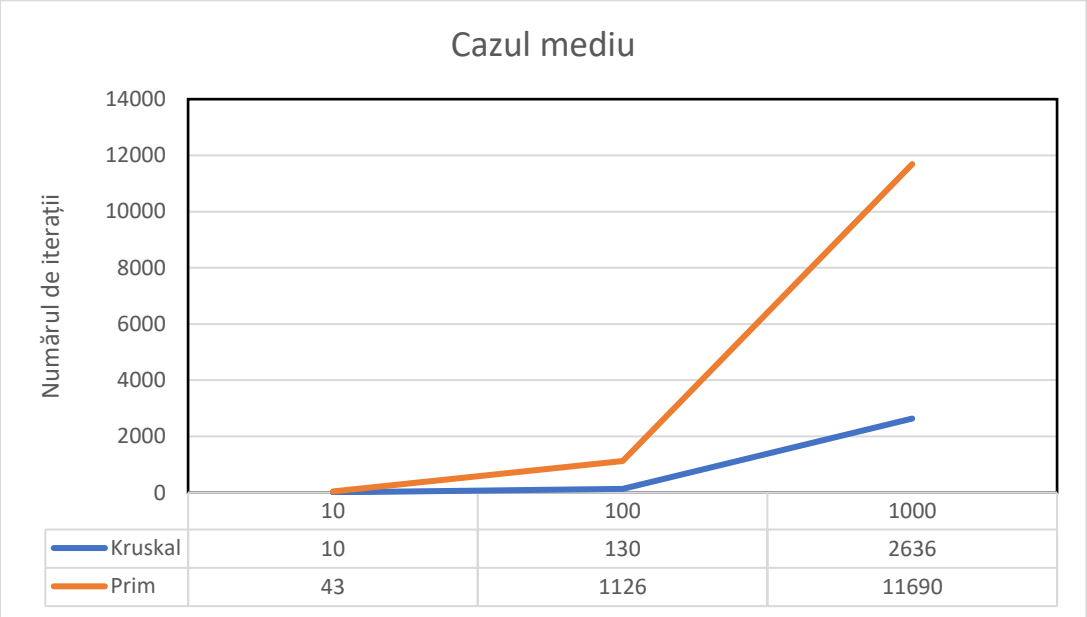
5.1 Cazul favorabil

Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	3124	1,678856
Kruskal	100	186	0,007530
Kruskal	10	11	0,000663
Prim	1000	8717	0,488688
Prim	100	1000	0,013207
Prim	10	45	0,000592



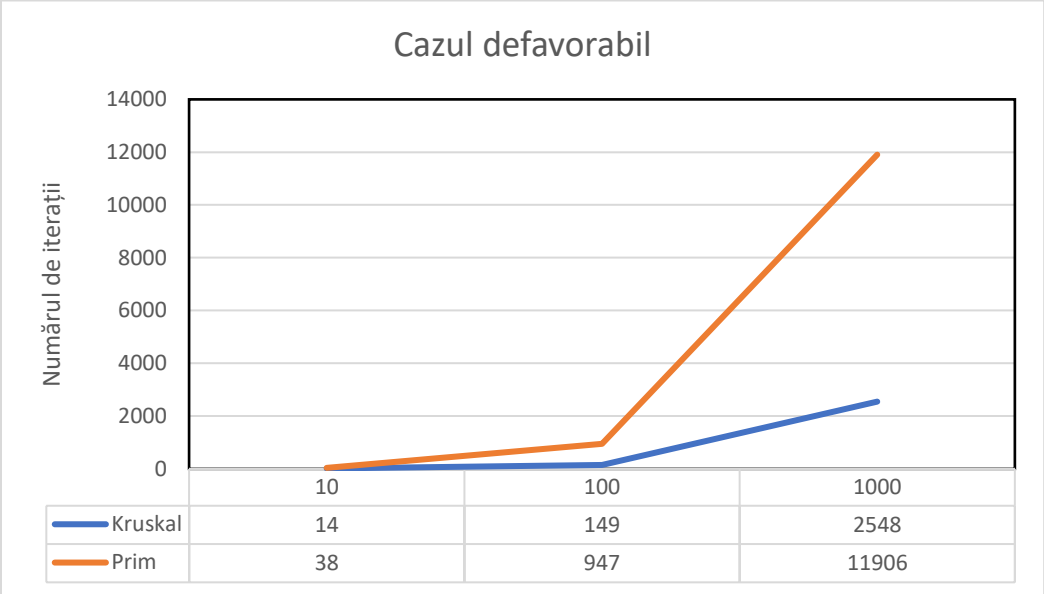
5.2 Cazul mediu

Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	2636	1,601732
Kruskal	100	130	0,006904
Kruskal	10	10	0,000443
Prim	1000	11690	0,765826
Prim	100	1126	0,013765
Prim	10	43	0,000483



5.3 Cazul defavorabil

Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	2548	1,865848
Kruskal	100	149	0,005108
Kruskal	10	14	0,000402
Prim	1000	11906	0,703878
Prim	100	947	0,005161
Prim	10	38	0,000577



6 Codul programului

Codul programului este scris în ANEXA C. Acesta implementează algoritmii Prim și Kruskal pentru generarea și analiza grafurilor, folosind o matrice de adiacență pentru a reprezenta costurile muchiilor. Programul permite utilizatorului să selecteze între diferite cazuri (favorabil, mediu, defavorabil) și să execute algoritmii pe baza grafului generat.

Pentru detalii suplimentare și pentru a vizualiza implementarea completă a codului, consultați **ANEXA C**.

7 Concluzie

În această lucrare, am explorat implementarea algoritmilor de căutare a arborelui de acoperire minimă, în special algoritmii lui Prim și Kruskal, utilizând o matrice de adiacență pentru a reprezenta grafurile. Am analizat diferite scenarii de testare: cazuri favorabile, medii și defavorabile, pentru a evalua performanța acestor algoritmi în diverse condiții.

Algoritmul lui Prim este eficient în gestionarea grafurilor dense, deoarece se concentrează pe extinderea arborelui de acoperire minimă prin adăugarea treptată a celor mai mici costuri disponibile. Această abordare permite o explorare sistematică a grafului, asigurându-se că, la fiecare pas, se face alegerea optimă locală. În contrast, algoritmul lui Kruskal se bazează pe o abordare bazată pe muchii, sortând toate muchiile și adăugându-le în arborele de acoperire minimă, cu condiția ca adăugarea să nu formeze un ciclu. Această diferență de strategii subliniază versatilitatea ambelor metode, fiecare având propriile avantaje în funcție de structura grafului.

Prin testarea acestor algoritmi în diferite condiții de intrare, am observat cum variațiile în structura grafului afectează timpul de execuție și eficiența algoritmilor. În cazurile favorabile, am observat o performanță optimă, cu timpi de execuție reduși, în timp ce în cazurile defavorabile, timpii de execuție au crescut semnificativ, evidențiind impactul complexității grafurilor asupra eficienței algoritmice.

De asemenea, am implementat strategii de optimizare, cum ar fi path compression în algoritmul lui Kruskal, care contribuie la reducerea timpului de execuție prin minimizarea adâncimii arborilor de seturi disjuncte. Acest tip de optimizare este esențial în contextul grafurilor mari, unde numărul de noduri și muchii poate deveni semnificativ, afectând astfel performanța globală a algoritmilor.

În concluzie, atât algoritmul lui Prim, cât și cel al lui Kruskal sunt instrumente fundamentale în teoria grafurilor, având aplicații variate în domenii precum rețelele de calculatoare, planificarea resurselor și analiza datelor. Studiul acestor algoritmi nu doar că ne ajută să înțelegem conceptele fundamentale ale structurilor de date și ale algoritmilor, dar ne oferă și o bază solidă pentru a aborda probleme mai complexe în informatică. În urma experimentelor realizate, putem concluziona că alegerea algoritmului adecvat depinde de natura specifică a problemei, precum și de structura datelor implicate. Această lucrare deschide, de asemenea, calea pentru cercetări viitoare, în special în direcția optimizării algoritmilor și a explorării altor tehnici de căutare a arborelui de acoperire minimă.

8 Bibliografie

1. „Algoritmi pentru determinare a arborelui parţial de cost minim.” https://www.mateinfo.net/muscalua/parcarbori.pdf
2. „Arbori parțiali de cost minim: Kruskal, Prim.” https://www.studocu.com/ro/document/academia-de-studii-economice-din-bucuresti/algoritmi-si-tehnici-de-programare/s13-arbori-partiali-de-cost-minim-kruskal-si-prim/26218183
3. „Curs 13 - Tehnici de programare.” https://www.cs.ubbcluj.ro/~istvanc/fp/curs/Curs13%20-%20Greedy,%20Programare%20dinamica.pdf
4. Kruskal’s Minimum Spanning Tree (MST) Algorithm https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/
5. Kruskal Algorithm Java https://www.javatpoint.com/kruskal-algorithm-java
6. Time and Space Complexity Analysis of Prim's Algorithm https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-prims-algorithm/
7. Prim's algorithm Java https://www.javatpoint.com/prims-algorithm-java

ANEXA A: Pseudocod pentru Algoritmul Kruskal

```
Kruskal(G):
    // G este graful, reprezentat printr-o listă de muchii
    MST = setul de muchii ale arborelui de acoperire minim
    edges = sort(G.edges, key=weight) // Sortează muchiile în funcţie de greutate
    parent = [] // Array pentru părinţi
    rank = []   // Array pentru ranguri
    for each vertex v in G:
        parent[v] = v   // Fiecare nod este iniţial un set separat
        rank[v] = 0     // Rang iniţial este 0
    for each edge (u, v) in edges:
        if Find(parent, u) != Find(parent, v): // Verifică dacă u şi v sunt în acelaşi
set
            MST.addEdge(u, v)              // Adaugă muchia în MST
            Union(parent, rank, u, v)      // Unifică seturile
    return MST
Find(parent, v):
    if parent[v] != v:
        parent[v] = Find(parent, parent[v]) // Căutare cu compresie
    return parent[v]
Union(parent, rank, u, v):
    rootU = Find(parent, u)
    rootV = Find(parent, v)


    if rootU != rootV:
        if rank[rootU] > rank[rootV]:
            parent[rootV] = rootU
        else if rank[rootU] < rank[rootV]:
            parent[rootU] = rootV
        else:
            parent[rootV] = rootU
            rank[rootU] += 1 // Creşte rangul
```

ANEXA B: Pseudocod pentru Algoritmul lui Prim

```
Prim(G, start):
    // G este graful, reprezentat printr-o matrice de adiacență
    // start este nodul de început
    V = numărul de vârfuri în G
    MST = setul de muchii ale arborelui de acoperire minim
    cost = 0
    // Inițializare
    for each vertex v in G:
        key[v] = ∞      // Costul minim pentru a ajunge la v
        parent[v] = NULL // Părintele lui v în MST
    key[start] = 0       // Costul pentru nodul de start este 0
    minHeap = MinHeap()  // Heap minim pentru a selecta nodul cu costul minim
    minHeap.insert(start)

    while not minHeap.isEmpty():
        u = minHeap.extractMin() // Extrage nodul cu costul minim
        // Adaugă muchia (parent[u], u) în MST
        if parent[u] is not NULL:
            MST.addEdge(parent[u], u)
            cost += weight(parent[u], u) // Adaugă greutatea muchiei la cost total
        // Actualizează costurile pentru nodurile adiacente
        for each vertex v adjacent to u:
            if v is not in MST and weight(u, v) < key[v]:
                key[v] = weight(u, v) // Actualizează costul
                parent[v] = u              // Actualizează părintele
                minHeap.decreaseKey(v, key[v]) // Actualizează heap-ul
    return MST, cost
```

ANEXA C: Codul programului

```
import java.util.Scanner;
import java.util.Random;

public class GraphAlgorithm {
    static final int MAX = 10000;
    static final int INF = 10000;
    static int[][] MS = new int[MAX][MAX];
    static int[][] MV = new int[MAX][MAX];
    static int[][] MVprim = new int[MAX][MAX];
    static int[][] MVkruskal = new int[MAX][MAX];
    static int n;
    static int count1 = 0, count2 = 0;
    static int[] parent = new int[MAX];
    static int[] visited = new int[MAX];

    public static void reset() {
        count1 = 0;
        count2 = 0;
        for (int i = 0; i < n; i++) {
            parent[i] = 0;
            visited[i] = 0;
            for (int j = 0; j < n; j++) {
                MVprim[i][j] = MV[i][j];
                MVkruskal[i][j] = MV[i][j];
            }
        }
    }

    public static void nrVirfDefavorabil() {
        System.out.print("Numarul de virfuri: ");
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        Random rand = new Random();
```

```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (j > i) {
                    MS[i][j] = rand.nextInt(10000);
                } else if (i > j) {
                    MS[i][j] = MS[j][i];
                } else {
                    MS[i][j] = 0;
                }
            }
        }
    }

    public static void costurileDefavorabil() {
        System.out.println("Costurile muchiilor:");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (MS[i][j] != 0 && i != j) {
                    MV[i][j] = MS[i][j];
                } else {
                    MV[i][j] = INF;
                }
            }
        }
        reset();
    }

    public static void nrVirfFavorabil() {
        System.out.print("Numarul de virfuri: ");
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (i != j) {
```

```
                MS[i][j] = rand.nextInt(1000) + 1;
            } else {
                MS[i][j] = 0;
            }
        }
    }
}

public static void costurileFavorabil() {
    System.out.println("Costurile muchiilor:");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (MS[i][j] != 0 && i != j) {
                MV[i][j] = MS[i][j];
            } else {
                MV[i][j] = INF;
            }
        }
    }
    reset();
}

public static void nrVirfMediu() {
    System.out.print("Numarul de virfuri: ");
    Scanner sc = new Scanner(System.in);
    n = sc.nextInt();
    Random rand = new Random();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (j > i) {
                MS[i][j] = rand.nextInt(1000) + 1;
                MS[j][i] = MS[i][j];
            } else if (i == j) {
                MS[i][j] = 0;
            }
```

```
        }
    }
}

public static void costurileMediu() {
    System.out.println("Costurile muchiilor:");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (MS[i][j] != 0 && i != j) {
                System.out.println((i + 1) + " -> " + (j + 1) + " : " + MS[i][j]);
                MV[i][j] = MS[i][j];
            } else {
                MV[i][j] = INF;
            }
        }
    }
    reset();
}

public static int find(int i) {
    if (parent[i] == 0) return i;
    return parent[i] = find(parent[i]);
}

public static boolean union(int i, int j) {
    if (i != j) {
        parent[j] = i;
        return true;
    }
    return false;
}

public static void prim() {
    int a = -1, b = -1, u = -1, v = -1, ne = 1, min;
    visited[0] = 1;
```

```
        System.out.println("Algoritmul lui Prim:\n");
        long startTime = System.nanoTime();

        while (ne < n) {
            count1++;
            min = INF;
            for (int i = 0; i < n; i++) {
                if (visited[i] == 1) {
                    for (int j = 0; j < n; j++) {
                        if (MVprim[i][j] < min && visited[j] == 0) {
                            count1++;
                            min = MVprim[i][j];
                            a = i;
                            b = j;
                        }
                    }
                }
            }

            if (visited[b] == 0) {
                System.out.println("Muchia " + (ne++) + ": " + (a + 1) + " -> " + (b +
1) + ", costul = " + min);
                visited[b] = 1;
            }
            MVprim[a][b] = MVprim[b][a] = INF;
        }

        long endTime = System.nanoTime();
        double executionTime = (endTime - startTime) / 1_000_000_000.0;
        System.out.println("\nNr. Iteratii: " + count1);
        System.out.printf("Timpul de executie: %.6f secunde\n", executionTime);
    }


    public static void kruskal() {
```

```
        int a = -1, b = -1, u = -1, v = -1, ne = 1, min;
        System.out.println("Algoritmul Kruskal:\n");
        long startTime = System.nanoTime();

        while (ne < n) {
            count2++;
            min = INF;

            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (MVkruskal[i][j] < min) {
                        count1++;
                        min = MVkruskal[i][j];
                        a = i;
                        b = j;
                    }
                }
            }

            u = find(a);
            v = find(b);
            if (union(u, v)) {
                System.out.println("Muchia " + (ne++) + ": " + (a + 1) + " -> " + (b +
1) + ", costul = " + min);
            }
            MVkruskal[a][b] = MVkruskal[b][a] = INF;
        }

        long endTime = System.nanoTime();
        double executionTime = (endTime - startTime) / 1_000_000_000.0;
        System.out.println("\nNr. Iteratii: " + count2);
        System.out.printf("Timpul de executie: %.6f secunde\n", executionTime);
    }
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Random rand = new Random();
    int chooseMenu;
    while (true) {
        System.out.println("1. Cazul favorabil.");
        System.out.println("2. Cazul mediu.");
        System.out.println("3. Cazul defavorabil.");
        System.out.println("0. Stop Program.");
        System.out.print("Raspuns: ");
        chooseMenu = sc.nextInt();

        switch (chooseMenu) {
            case 1 -> {
                nrVirfFavorabil();
                costurileFavorabil();
                prim();
                kruskal();
            }
            case 2 -> {
                nrVirfMediu();
                costurileMediu();
                prim();
                kruskal();
            }
            case 3 -> {
                nrVirfDefavorabil();
                costurileDefavorabil();
                prim();
                kruskal();
            }
            case 0 -> System.exit(0);
            default -> System.out.println("ERROR");
```

```
                }
            }
        }
}
```