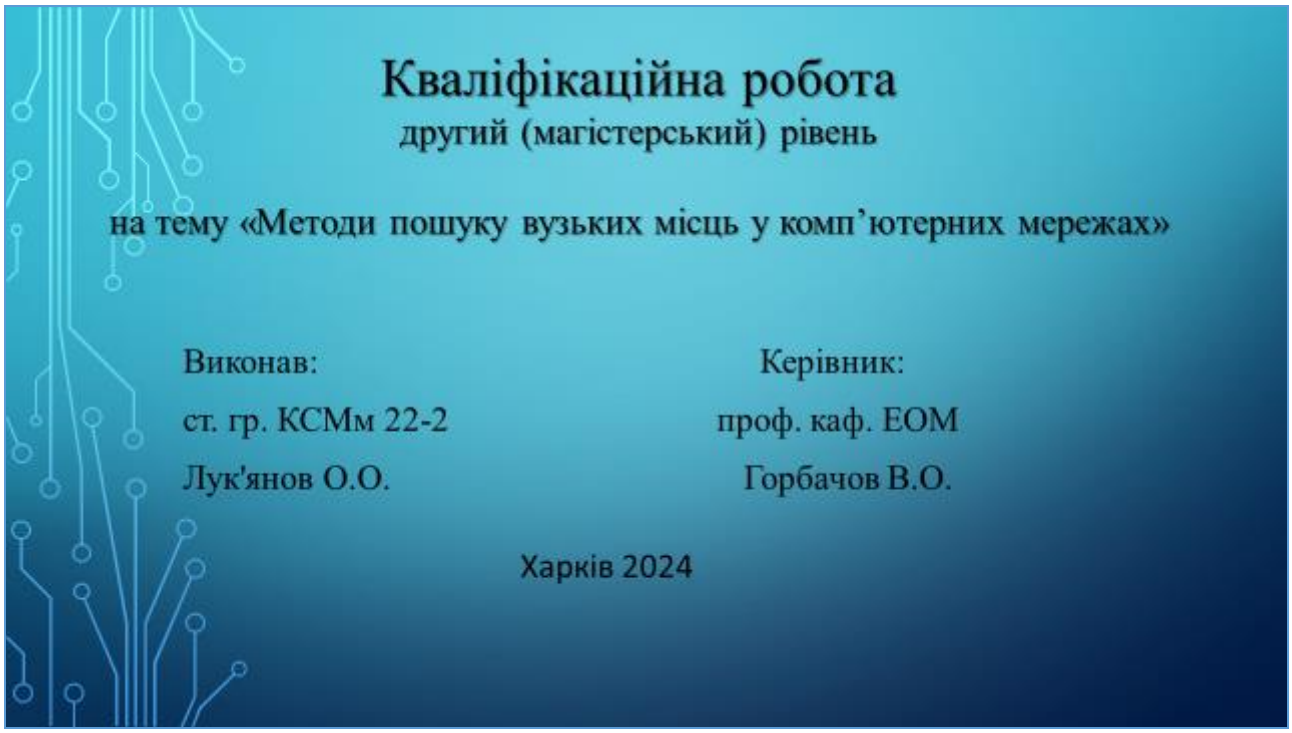
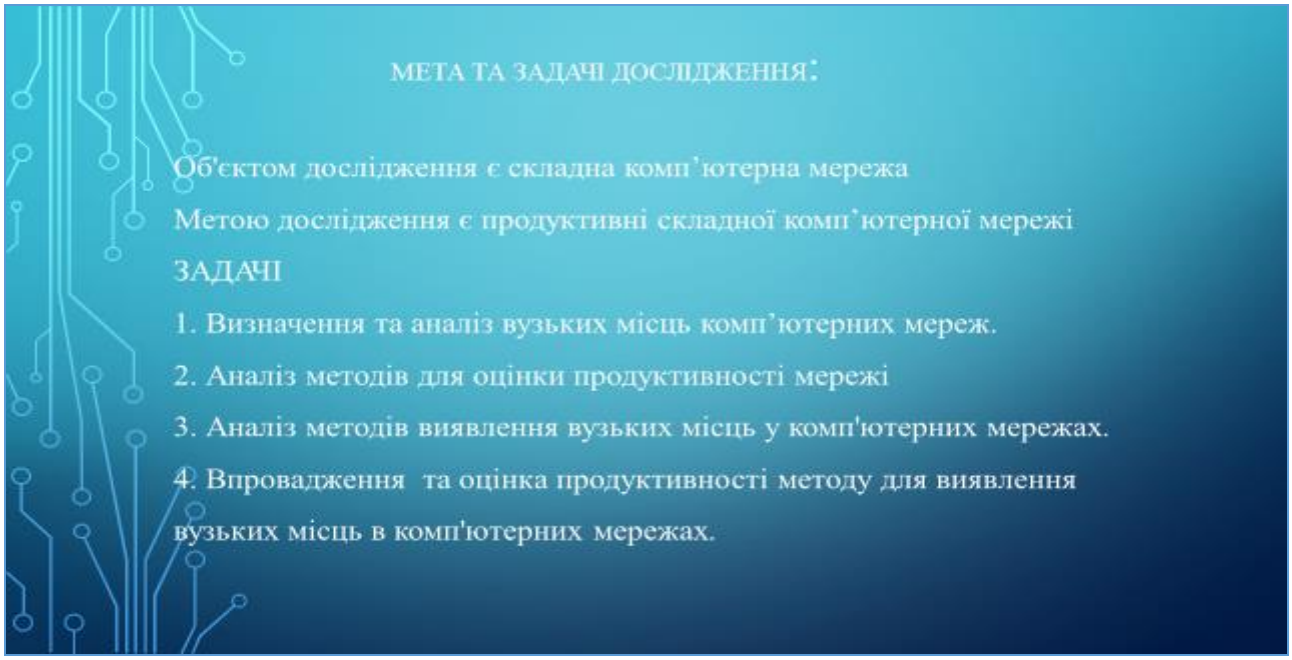
ДОДАТОК А

Графічний матеріал кваліфікаційної роботи


Кваліфікаційна робота
другий (магістерський) рівень
на тему «Методи пошуку вузьких місць у комп'ютерних мережах»
Виконав:
ст. гр. КСМм 22-2
Лук'янов О.О.
Керівник:
проф. каф. ЕОМ
Горбачов В.О.
Харків 2024


МЕТА ТА ЗАДАЧІ ДОСЛІДЖЕННЯ:
Об'єктом дослідження є складна комп'ютерна мережа
Метою дослідження є продуктивні складної комп'ютерної мережі
ЗАДАЧІ
1. Визначення та аналіз вузьких місць комп'ютерних мереж.
2. Аналіз методів для оцінки продуктивності мережі
3. Аналіз методів виявлення вузьких місць у комп'ютерних мережах.
4. Впровадження та оцінка продуктивності методу для виявлення вузьких місць в комп'ютерних мережах.

ВУЗЬКІ МІСЦЯ МЕРЕЖІ ТА ПРИЧИНИ ЇХ ВИНИКНЕННЯ

Вузькі місця це деякі ресурси або обладнання, які значно обмежують продуктивність системи

Обмежена пропускна здатність

Проблеми часто виникають через обмежену пропускну здатність. Можливо, ваше використання перевищило пропускну здатність, виділену вашим постачальником послуг Інтернету (ISP). Виправте це, оновивши план.

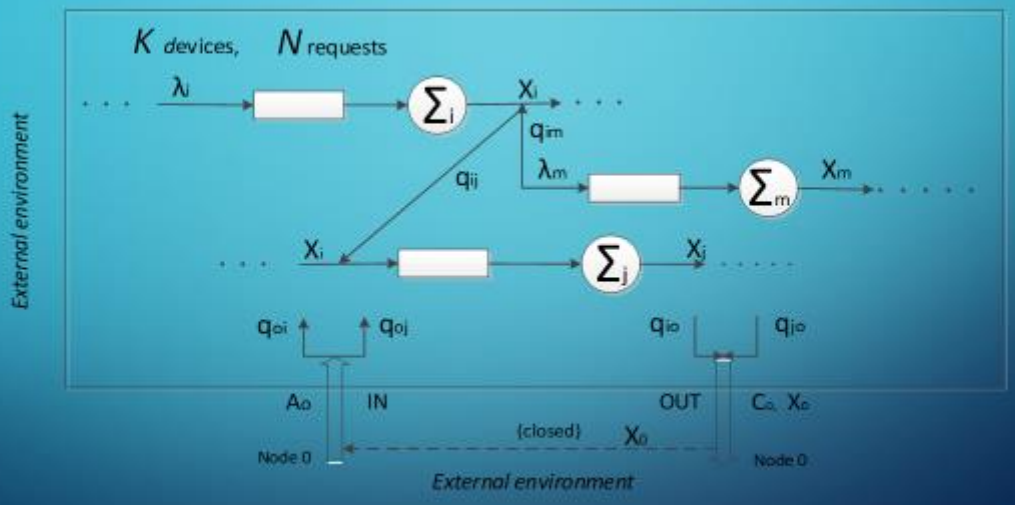
Вихід з ладу пристрою

Іноді наявність несправного пристрою може спричинити вузьке місце мережі через надсилання підробленого трафіку. Якщо на певній робочій станції сповільнюється мережа, перемикання на інший порт може вирішити проблему.

Перевантаження пристрою (коефіцієнт використання)

Концентрація мережі також може спричинити вузьке місце мережі. Якщо до певного сегменту підключено занадто багато пристроїв, це може сповільнити доступ. Вирішіть це, зменшивши кількість перемикань між серверами та робочими станціями. 3

СТРУКТУРНА МОДЕЛЬ КОМП'ЮТЕРНОЇ МЕРЕЖІ



4

ЗАКОН ЗБЕРЕЖЕННЯ (БАЛАНСУ) ПОТОКІВ

Кількість заявок, які надійшли до деякого вузла на протязі тривалого періоду часу T, дорівнює кількості заявок, які залишили цей вузол за час T. Коли потоки в мережі збалансовані , Xi можна розглядати як продуктивність вузла i. Гіпотеза про баланс потоків в мережі може бути представлена у вигляді рівняння:

$$A_j = C_j,\ j = 0,\ldots K$$

Ця гіпотеза може бути використана в сталому режимі і в тому випадку, коли відношення (Aj - Cj) / Cj є незначним.

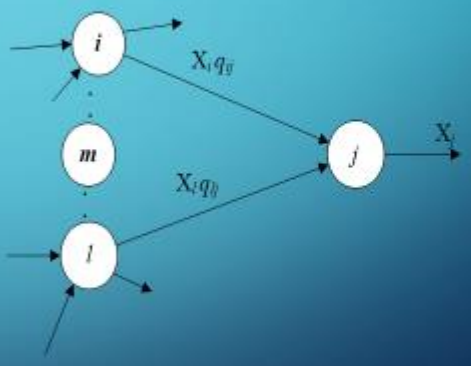
5

РІВНЯННЯ БАЛАНСУ ПОТОКІВ (FLOW BALANCE)

Рівняння балансу потоків

$$X_j = \sum_{i=0}^{K} X_i q_{ij},\ \ j = \overline{0, K}$$

Графічна інтерпретація рівнянь X_j



6

РІВНЯННЯ БАЛАНСУ ПОТОКІВ ЯК СИСТЕМА ЛІНІЙНИХ РІВНЯНЬ

$$
\begin{aligned}
X_0 &= X_0 q_{00} + X_1 q_{10} + \ldots + X_k q_{k0}, \\
X_1 &= X_0 q_{01} + X_1 q_{11} + \ldots + X_k q_{k1}, \\
&\cdot \\
&\cdot \\
X_k &= X_0 q_{0k} + X_1 q_{1k} + \ldots + X_k q_{kk},
\end{aligned}
$$

Система рівнянь балансу потоків являє собою значну цінність, тому що показує взаємозв'язок між структурою мережі, представленої матрицею $|q_{ij}|$ і продуктивністю вузлів X_i

7

ПРОГРАМА ВПРОВАДЖЕННЯ ТА ОЦІНКИ ПРОДУКТИВНОСТІ МЕТОДУ ДЛЯ ВИЯВЛЕННЯ ВУЗЬКИХ МІСЦЬ В КОМПЮТЕРНИХ МЕРЕЖАХ

Мова програмування Java.

Для заданої мережі

1. Знайти вузькі міста між двома вибраними вузлами.
2. Знайти самий короткий шлях між двома вибраними вузлами
3. Знайти вузькі міста на вибраній множені вузлів.

8

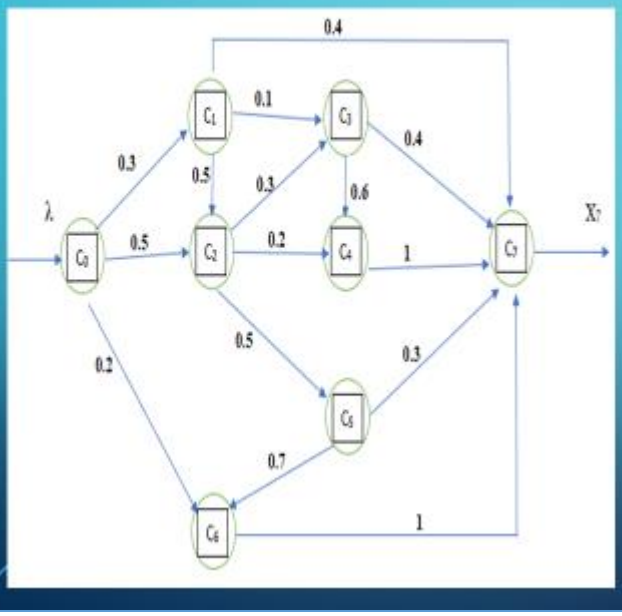
КРОКИ ВПРОВАДЖЕННЯ ПОШУКОВОГО СЦЕНАРІЮ

1. Створення систем лінійних рівнянь балансу потоку.
2. Обчислення продуктивностей Xi.
3. Обчислення коефіцієнтів використання Ui = Si* Xi.
4. Пошук вузьких міст відповідно до задач.
   1. Знайти вузькі міста між двома вибраними вузлами.
   2. Знайти самий короткий шлях між двома вибраними вузлами
   3. Знайти вузькі міста на вибраній множені вузлів.

9

РОЗРОБКА ПРОЕКТУ. ПОСТАНОВКА ЗАДАЧІ

Приклад мережі



Система рівнянь балансу потоків

If we apply this mathematical relation on our system,

We get the eight equations below:

X0 = λ = 35
X1 = 0.3*X0
X2 = 0.5*X0+ 0.5*X1
X3 = 0.1*X1 + 0.3*X2
X4 = 0.2*X2 + 0.6*X3
X5 = 0.5*X2
X6 = 0.2*X0 + 0.7*X5
X7 = 0.4*X1 + 0.4*X3 + 1*X4 + 0.3*X5 + 1*X6

10

РОЗРОБКА ПРОЕКТУ. РЕЗУЛЬТАТ

```
**********Calculate the Throughput Xi**********
enter arrival rate λ
35
enter number of equations
8
enter number of unknowns
8
Enter the elements of the matrix
0 0.3 0.5 0 0 0 0.2 0
0 0 0.5 0.1 0 0 0 0.4
0 0 0 0.3 0.2 0.5 0 0
0 0 0 0 0.6 0 0 0.4
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0.7 0.3
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
X0 = 35.0000
X1 = 10.5000
X2 = 22.7500
X3 = 7.8750
X4 = 9.2750
X5 = 11.3750
X6 = 14.9625
X7 = 35.0000
**********Find Bottleneck node**********
enter Si values:
0.015 0.045 0.03 0.04 0.05 0.027 0.04 0.01
node1:
1
node2:
5
The bottleneck node is: 2
**********Find the reliable path**********
source node:
0
destination node:
5
The distance from node 0 to node 5 is: 1.2075 | Shortest Path: 0 2 5
**********Find the bottleneck set**********
enter the utilizaton values:
0.4 0.6 0.60 0.3 0.72 0.8
The bottleneck nodes are: 0.6 0.68 0.72
```

11

РОЗРОБКА ПРОЕКТУ. РЕЗУЛЬТАТ

Коефіцієнти використання вузл
Ui = Si * Xi

Для критичного вузла
U ≥ 0,6

Ui = Si * Xi 0 = 0.525
U1 = 0.4725
U2 = 0.6825
U3 = 0.315
U4 = 0.46375
U5 = 0.307
U6 = 0.5985
U7 = 0.35

12

ВИСНОВКИ

У ході даної роботи було проаналізовано метод для оцінки продуктивності мережі з використанням рівнянь балансу потоків..

Впроваджено такий метод для виявлення вузьких місць і розрахунку, прогнозування показників продуктивності в комп'ютерних мережах.

Результати цієї роботи можуть бути використані для прогнозування показників продуктивності в комп’ютерних мережах, перевірки достовірності інформації (або перевірки на несуперечливість), управління мережами в реальному масштабі часу.

13

Результати були представлені на конференції і в доповідях

ДОДАТОК Б

Програмний код

```
import java.util.*;
class ShortestPath {
    private static final double EPS = 1e-10;
    int n,m,b_Index = -1, lambda =0;
    double A[][], C[][];
    double B[],x[],S[], U[];
    double minimumDistance(double dist[], Boolean visited[]) {
        // Initialize min value
        double min = Integer.MAX_VALUE, min_index = -1;
        for (int v = 0; v < n; v++)
            if (visited[v] == false && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }

        return min_index;
    }
    void print(double dist[], int src, int n, int[] parents) {
        if (dist[n] != Integer.MAX_VALUE) {
            System.out.print("The distance from node " + src + "
to node " + n + " is: " + dist[n] + " | Shortest Path: ");
            printPath(n, parents);
        } else {
            System.out.println("Sorry! There is no path from " +
src + " to " + n);
        }
    }
    private static void printPath(int currentNode, int[]
parents) {
        if (currentNode == -1) {
            return;
        }
        printPath(parents[currentNode], parents);
        System.out.print(currentNode + " ");
    }
    void dijkstra_Algorithm(double graph[][], int src, int dest)
{
        double dist[] = new double[n];
        int[] parents = new int[n];
        parents[src] = -1;
        Boolean visited[] = new Boolean[n];
        for (int i = 0; i < n; i++) {
            dist[i] = Integer.MAX_VALUE;
            visited[i] = false;
        }
        dist[src] = 0;
```

```
        for (int count = 0; count < n - 1; count++) {
            double u = minimumDistance(dist, visited);
            visited[(int) u] = true;
            for (int v = 0; v < n; v++)
                if (!visited[v] && graph[(int) u][v] != 0 &&
                        dist[(int) u] != Integer.MAX_VALUE &&
                        dist[(int) u] + graph[(int) u][v] <
dist[v]) {
                    parents[v] = (int) u;
                    dist[v] = dist[(int) u] + graph[(int) u][v];
                }
        }
        print(dist, src, dest, parents);
    }
    public double[] solve(double[][] A, double[] B) {
        for (int p = 0; p < n; p++) {
            int max = p;
            for (int i = p + 1; i < m; i++) {
                if (Math.abs(A[i][p]) > Math.abs(A[max][p])) {
                    max = i;
                }
            }
            double[] temp = A[p];
            A[p] = A[max];
            A[max] = temp;
            double t = B[p];
            B[p] = B[max];
            B[max] = t;
            if (Math.abs(A[p][p]) <= EPS) {
                throw new RuntimeException("The Matrix is
singular or nearly singular");
            }
            for (int i = p + 1; i < m; i++) {
                double alpha = A[i][p] / A[p][p];
                B[i] -= alpha * B[p];
                for (int j = p; j < n; j++) {
                    A[i][j] -= alpha * A[p][j];
                }
            }
        }
        x = new double[n];
        for (int i = n - 1; i >= 0; i--) {
            double sum = 0.0;
            for (int j = i + 1; j < n; j++) {
                sum += A[i][j] * x[j];
            }
            x[i] = (B[i] - sum) / A[i][i];
        }
        for (int i = 0; i < n; i++)
            System.out.println("X" + i + " = " +String.
format("%.4f", x[i]));

        return x;
```

```
    }
    public void readMatrixByUser() {
        int i, j;
        Scanner in = null;
        try {
            in = new Scanner(System.in);
            System.out.println("enter arrival rate λ");
            lambda = in.nextInt();
            System.out.println("enter number of equations");
            m = in.nextInt();
            System.out.println("enter number of unknowns");
            n = in.nextInt();
            // Declare the matrix
            A = new double[m][n];
            C = new double[m][n];

            // Read the matrix values
            System.out.println("Enter the elements of the
matrix");
            for (i = 0; i < m; i++) {
                for (j = 0; j < n; j++) {
                    A[j][i] = in.nextDouble();
                    C[i][j] = A[j][i];
                    if (i == j) {
                        if (j == 0)
                            A[j][i] = 1.0;
                        else
                            A[j][i] = -1.0;
                    }
                }
            }
            B = new double[n];
            B[0] = lambda;
            for (int k = 1; k < n; k++)
                B[k] = 0;
        } catch (Exception e) {
        }
    }
    public int findBottleneck(int node1 , int node2 ,
ArrayList<Double>list)
    {
        S = new double[n];
        U = new double[n];
        for(int i=0;i<list.size();i++)
        {
            S[i] = list.get(i);
            U[i] = S[i]*x[i];
        }

        for(int j=node1;j<=node2;j++)
        {
            if(U[j]>=0.6 && U[j]<=0.72) {
                b_Index = j;
```

```
            }
        }
        return b_Index;
    }
    public void subSetBottlenecks(List<Double> util)
    {
        System.out.print("The bottleneck nodes are: ");
        for(int i=0;i<util.size();i++)
            if(util.get(i)>=0.6 && util.get(i)<=0.72)
                System.out.print(util.get(i)+" ");

        System.out.println();
    }
    public double[][] setMatrix(double[][] graph)
    {
        double[][] new_graph = new
double[graph.length][graph.length];
        for(int i=0;i<graph.length;i++)
            for(int j=0;j<graph[i].length;j++)
                if(graph[i][j] != 0)
                    new_graph[i][j] = U[i];

            return new_graph;
    }
    // Driver code
    public static void main(String[] args) {
        ShortestPath g = new ShortestPath();
        Scanner in = new Scanner(System.in);
        ArrayList<Double> input = new ArrayList<Double>();
        ArrayList<Double> list = new ArrayList<Double>();
        System.out.println("*************Calculate the
Throughput Xi************");
        g.readMatrixByUser();
        g.solve(g.A, g.B);
        System.out.println("*************Find Bottleneck
node**************");
        System.out.println("enter Si values: ");
        String str = in.nextLine();
        String[] s = str.split(" ");
        for (int i = 0; i < s.length; i++)
            list.add(Double.parseDouble(s[i]));
        System.out.println("node1: ");
        int n1 = in.nextInt();
        System.out.println("node2: ");
        int n2 = in.nextInt();
        int h= g.findBottleneck(n1,n2,list);
        if(h == -1)
            System.out.println("There is no bottleneck between
the two selected nodes");
        else
            System.out.println("The bottleneck node is: "+h);
        System.out.println("*****************Find the reliable
path**************");
```

```
        System.out.println("source node: ");
        int src = in.nextInt();
        System.out.println("destination node: ");
        int dest = in.nextInt();
        double[][] graph = g.setMatrix(g.C);
        g.dijkstra_Algorithm(graph, src, dest);
        System.out.println();
        System.out.println("***********Find the bottleneck
set*****************");
        System.out.println("enter the utilizaton values: ");
        in.nextLine();
        String string = in.nextLine();
        String[] num = string.split(" ");
        for (int i = 0; i < num.length; i++)
            input.add(Double.parseDouble(num[i]));
        g.subSetBottlenecks(input);  }}
```