5МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

Кафедра дискретного аналізу та інтелектуальних систем

ПАРАЛЕЛЬНІ ТА РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ
ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №6
“АЛГОРИТМ ДЕЙКСТРИ”

Роботу виконав:
Сенюк Віталій Васильович
Студент групи ПМІ-33с

Перевірив:
Доц. Пасічник Т.В.
кафедри програмування
Львівського національного
Університету імені Івана Франка

Львів – 2023

Тема: алгоритм Дейкстри.

Мета роботи: Розпаралелити виконання алгоритму Дейкстри.

Завдання

Для зваженого графа G(V,F), де V={a0, a1,…an} – множина вершин (n – велике число), а Fмножина ребер між вершинами, використовуючи алгоритм Дейкстри, знайти найкоротший шлях між заданою вершиною a та усіма іншими.

Для різної розмірності графів та довільного вузла a порахувати час виконання програми без потоків та при заданих k потоках розпаралелення.

Виконання роботи

1. Створюю клас для роботи з графом у виді матриці.

У даному випадку прийшлось створити вдосконалений клас графу. Тому, щоб показати роботу алогоритму потрібно було створювати великі графи у памяті, що призводило до помилки OUT OF MEMORY.

```
public class Graph
{
    private Dictionary<int, Dictionary<int, int>> adjacencyList;

    1 reference
    public Graph(int numberOfVertices)
    {
        Random random = new Random();
        adjacencyList = new Dictionary<int, Dictionary<int, int>>();

        for (int i = 0; i < numberOfVertices; i++)
        {
            adjacencyList[i] = new Dictionary<int, int>();
            int numberOfEdges = random.Next(1, 11); // Each vertex has 1 to 10 edges

            for (int j = 0; j < numberOfEdges; j++)
            {
                int randomVertex = random.Next(numberOfVertices);
                if (randomVertex != i && !adjacencyList[i].ContainsKey(randomVertex))
                {
                    int weight = random.Next(1, 101); // Edge weight between 1 and 100
                    adjacencyList[i][randomVertex] = weight;
                }
            }
        }
    }

    1 reference
    public Dictionary<int, Dictionary<int, int>> GetAdjacencyList()
    {
        return adjacencyList;
    }
```

2. Створюю алгоритм для лінійного обчислення алгоритму Дейкстри.

```
public static Dictionary<int, int> Dijkstra(Dictionary<int, Dictionary<int, int>> adjacencyList
            , int startVertex)
{
    var distances = new Dictionary<int, int>();
    var unvisited = new Dictionary<int, bool>();
    var numberOfVertices = adjacencyList.Count;
    for (int i = 0; i < numberOfVertices; i++)
    {
        distances[i] = int.MaxValue;
        unvisited[i] = true;
    }
    distances[startVertex] = 0;

    for (int i = 0; i < numberOfVertices - 1; i++)
    {
        int minDistance = int.MaxValue;
        int currentVertex = -1;

        foreach (var vertex in distances)
        {
            if (unvisited[vertex.Key] && vertex.Value <= minDistance)
            {
                minDistance = vertex.Value;
                currentVertex = vertex.Key;
            }
        }
        if (currentVertex != -1)
        {
            unvisited[currentVertex] = false;
            foreach (var edge in adjacencyList[currentVertex])
            {
                int newDistance = distances[currentVertex] + edge.Value;
                if (newDistance < distances[edge.Key])
                {
                    distances[edge.Key] = newDistance;
                }
            }
        }
    }
    return distances;
}
```

3. Створюю паралельний алгоритм.

Ця оновлена паралельна реалізація намагається забезпечити точні результати, використовуючи блокування для синхронізації всіх оновлень, пов’язаних із відстанню та статусом відвідування вершин. Паралелізм походить від поділу пошуку вершини з найкоротшою відстанню та одночасного оновлення її сусідів.

```
public static ConcurrentDictionary<int, int> ParallelDijkstra(Dictionary<int, Dictionary<int, int>> adjacencyList
            , int startVertex, int maxDegreeOfParallelism)
{
    var distances = new ConcurrentDictionary<int, int>();
    var unvisited = new ConcurrentDictionary<int, bool>();

    for (int i = 0; i < adjacencyList.Count; i++)
    {
        distances[i] = int.MaxValue;
        unvisited[i] = true;
    }
    distances[startVertex] = 0;

    int processedVertices = 0;

    while (processedVertices < adjacencyList.Count - 1)
    {
        int minDistance = int.MaxValue;
        int currentVertex = -1;

        Parallel.ForEach(Partitioner.Create(0, adjacencyList.Count, adjacencyList.Count / maxDegreeOfParallelism), range =>
        {
            for (int i = range.Item1; i < range.Item2; i++)
            {
                if (unvisited[i] && distances[i] <= minDistance)
                {
                    lock (unvisited)
                    {
                        if (unvisited[i] && distances[i] <= minDistance)
                        {
                            minDistance = distances[i];
                            currentVertex = i;
                        }
                    }
                }
            }
        });

        if (currentVertex != -1)
        {
            unvisited[currentVertex] = false;
            processedVertices++;

            var currentEdges = adjacencyList[currentVertex];

            Parallel.ForEach(Partitioner.Create(0, currentEdges.Count), range =>
            {
                int index = 0;
                foreach (var edge in currentEdges)
                {
                    if (index >= range.Item1 && index < range.Item2)
                    {
                        int newDistance = distances[currentVertex] + edge.Value;
                        if (newDistance < distances[edge.Key])
                        {
                            lock (distances)
                            {
                                if (newDistance < distances[edge.Key])
                                {
                                    distances[edge.Key] = newDistance;
                                }
                            }
                        }
                    }
                    index++;
                }
            });
        }
    }

    return distances;
}
```

4. Правильність роботи алгоритму.

```
Iterational 'a' (0), time:00:00:00.0043403
To vertex 0: 0
To vertex 1: 62
To vertex 2: 27
To vertex 3: 80
To vertex 4: 96
To vertex 5: 53
To vertex 0: 0
To vertex 1: 62
To vertex 2: 27
To vertex 3: 80
To vertex 4: 96
To vertex 5: 53
Parallel t=2 'a' (0) time: 00:00:00.0317347
To vertex 0: 0
To vertex 1: 62
To vertex 2: 27
To vertex 3: 80
To vertex 4: 96
To vertex 5: 53
```

Як можемо бачити, два алгоритми демонструють однаковий результат.

5. Час та параметри на малому графі розміром 100 вершин.

```
GraphSize 100
Iterational 'a' (0), time:00:00:00.0022548
Parallel t=2 'a' (0) time: 00:00:00.0260259 acceleration: 0,08663677336806797 efficiency: 0,04331838668403398
Parallel t=4 'a' (0) time: 00:00:00.0027077 acceleration: 0,8327362706355947 efficiency: 0,20818406765889869
Parallel t=8 'a' (0) time: 00:00:00.0024781 acceleration: 0,9098906420241314 efficiency: 0,11373633025301642
```

Як бачимо, на малих даних паралельний алгоритм не ефективний.

6. Час та параметри на середньому графі розміром 2500 вершин.

```
GraphSize 2500
Iterational 'a' (0), time:00:00:00.1736462
Parallel t=2 'a' (0) time: 00:00:00.1799082 acceleration: 0,9651933597245707 efficiency: 0,4825966798622853
Parallel t=4 'a' (0) time: 00:00:00.1043184 acceleration: 1,6645788278961333 efficiency: 0,41614470697403333
Parallel t=8 'a' (0) time: 00:00:00.0859051 acceleration: 2,0213724214278312 efficiency: 0,2526715526784789
```

Як бачимо, прискорення відчувається набагато більше. Прискорення в даному випадку вже суттєво відчувається, а використовуючи 8 потоків, прискорення виросло більше ніж в двічі. Ефективність не найкраща.

7. Час та параметри на великому графі розміром 62500 вершин.

```
GraphSize 62500
Iterational 'a' (0), time:00:01:38.2073763
Parallel t=2 'a' (0) time: 00:01:00.5216183 acceleration: 1,6226825894376324 efficiency: 0,8113412947188162
Parallel t=4 'a' (0) time: 00:00:48.9392735 acceleration: 2,0067191291672932 efficiency: 0,5016797822918233
Parallel t=8 'a' (0) time: 00:00:40.4826094 acceleration: 2,4259151708733477 efficiency: 0,30323939635916847
```

У випадку з великим графом, прискорення та ефективність трішки покращуються порівняно з середнім графом.

Висновки

У результаті виконання роботи:

1. Спромігся розпаралелити алгоритм Дейкстри, хоча даний алгоритм не підходить для цього, так як має спільний ресурс, який потрібно синхронізувати між потокам.

Список використаних джерел

1. Вільний простір інтернету.