МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Паралельні та розподілені обчислення

ЛАБОРАТОРНА РОБОТА №6
«Алгоритм Дейкстри»

Виконала:
студентка групи ПМі-31
Дудчак Валентина Юріївна

Львів 2024

Тема: Розпаралелення алгоритму Дейкстри

Мета: Написати програми розв’язування алготму Дейкстри (послідовний та паралельний алгоритми). Для зваженого графа G(V,F), використовуючи алгоритм Дейкстри, знайти найкоротший шлях між заданою вершиною a та усіма іншими.

Хід роботи:

Для задачі я подаю граф у вигляді словника, де кожна вершина має словник зі значеннями кожної вершини, з якою вона з’днана, та вагою ребра між ними. Вершин, між якими відсутні ребра, та петель немає в словнику.

Послідовний алгоритм

Послідовний алгоритм Дейкстри реалізовано у методі SequentialDijkstraAlgorithm.

У ньому обчислюється найкоротший шлях між усіма парами вершин за допомогою алгоритму Дейкстри та зберігається у словник. Використано також об’єкт класу Stopwatch для засікання часу:

```
public static Stopwatch SequentialDijkstraAlgorithm(Dictionary<int, Dictionary<int, int>> graph, int nodeA)
{
  Stopwatch stopWatch = new Stopwatch();
  stopWatch.Start();

  var distances = new Dictionary<int, int>();
  var set = new Dictionary<int, bool>();
  int vertexCount = graph.Count;

  for (int i = 0; i < vertexCount; i++)
  {
    distances[i] = int.MaxValue;
    set[i] = false;
  }
  distances[nodeA] = 0;

  for (int i = 0; i < vertexCount - 1; i++)
  {
    int minDist = int.MaxValue;
    int curNode = -1;

    foreach (var node in distances)
    {
      if (!set[node.Key] && node.Value <= minDist)
      {
        minDist = node.Value;
        curNode = node.Key;
      }
    }
    if (curNode != -1)
    {
      set[curNode] = true;
      foreach (var edge in graph[curNode])
      {
        int updDist = distances[curNode] + edge.Value;
        if (updDist < distances[edge.Key])
        {
          distances[edge.Key] = updDist;
        }
      }
    }
  }

  stopWatch.Stop();
  // PrintDistances(distances, nodeA);
  return stopWatch;
}
```

Крім цього, я перевірила, чи результати обчислюються правильно:

```
Vertex 0: (Vertex 4, weight 40) (Vertex 2, weight 47) (Vertex 1, weight 34)
Vertex 1: (Vertex 2, weight 4) (Vertex 3, weight 83) (Vertex 4, weight 70) (Vertex 0, weight 71)
Vertex 2: (Vertex 4, weight 57) (Vertex 3, weight 22) (Vertex 0, weight 62)
Vertex 3: (Vertex 4, weight 1)
Vertex 4: (Vertex 1, weight 10) (Vertex 0, weight 32) (Vertex 3, weight 85) (Vertex 2, weight 30)
```

```
Distances from 1:
Vertex 0: 59
Vertex 1: 0
Vertex 2: 4
Vertex 3: 26
Vertex 4: 27
```

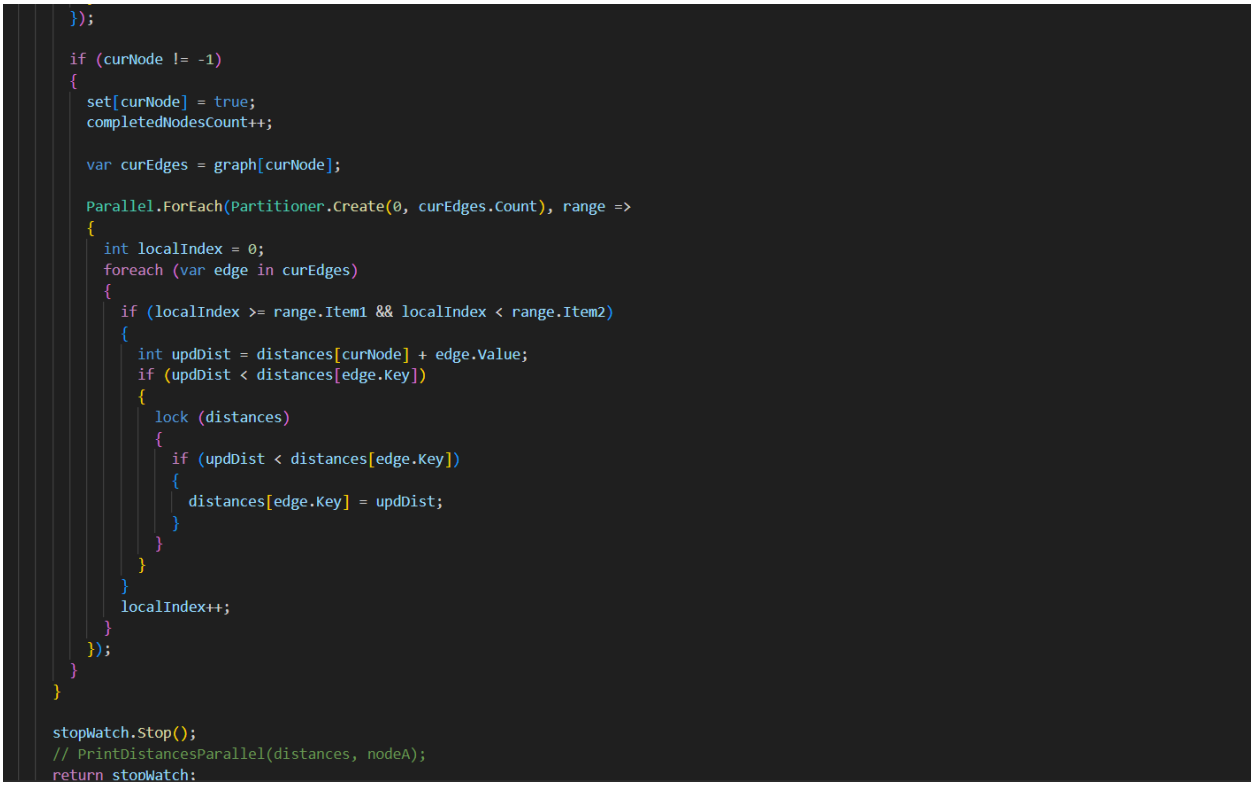
```
Distances from 1:
Vertex 0: 59
Vertex 1: 0
Vertex 2: 4
Vertex 3: 26
Vertex 4: 27
```

Паралельний алгоритм

Паралельний алгоритм Дейкстри реалізовано у методі ParallelDjikstraAlgorithm.

Для розпаралелення використовується конструкція Parallel.ForEach. Вона ділить вершини на частини для кожного потоку (за допомогою Partitioner.Create). Так як потоки залежать одне від одного, при оновленні відстані між вершинами використовується блокування (lock)

```
1 reference
public static Stopwatch ParallelDijkstraAlgorithm(Dictionary<int, Dictionary<int, int>> graph, int nodeA, int maxThreadCount)
{
  Stopwatch stopWatch = new Stopwatch();
  stopWatch.Start();

  var distances = new ConcurrentDictionary<int, int>();
  var set = new ConcurrentDictionary<int, bool>();
  int vertexCount = graph.Count;

  for (int i = 0; i < vertexCount; i++)
  {
    distances[i] = int.MaxValue;
    set[i] = false;
  }
  distances[nodeA] = 0;

  int completedNodesCount = 0;

  while (completedNodesCount < vertexCount - 1)
  {
    int minDist = int.MaxValue;
    int curNode = -1;

    Parallel.ForEach(Partitioner.Create(0, vertexCount, vertexCount / maxThreadCount), range =>
    {
      for (int i = range.Item1; i < range.Item2; i++)
      {
        if (!set[i] && distances[i] <= minDist)
        {
          lock (set)
          {
            if (!set[i] && distances[i] <= minDist)
            {
              minDist = distances[i];
              curNode = i;
            }
          }
```



```
    });

    if (curNode != -1)
    {
      set[curNode] = true;
      completedNodesCount++;

      var curEdges = graph[curNode];

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

  stopWatch.Stop();
  // PrintDistancesParallel(distances, nodeA);
  return stopWatch;
```

Метод також працює правильно (видно у результатах вище)

Аналіз результатів

На малих розмірах графа розпаралелення неефективне:

```
n = 100, a = 1
Sequential time: 0 ms
Parallel time: 25 ms, threads: 10
Acceleration: 0.011872588743917334
Efficiency: 0.0011872588743917333
```

```
n = 1000, a = 1
Sequential time: 26 ms
Parallel time: 125 ms, threads: 10
Acceleration: 0.20858295712808073
Efficiency: 0.020858295712808072
```

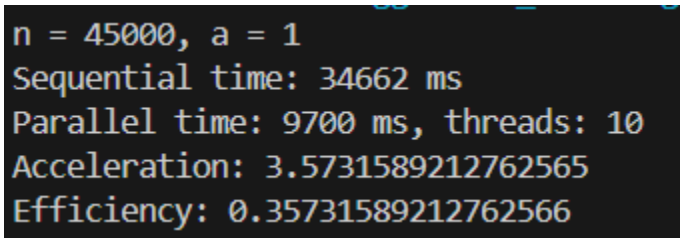
На більших розмірностях розпаралелення ефективніше:

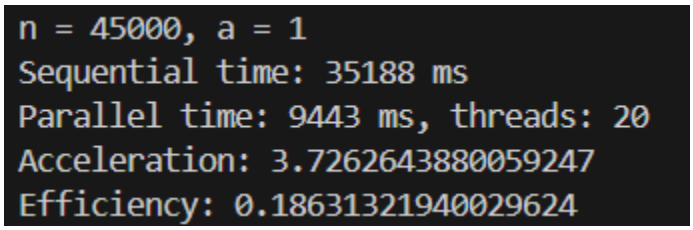
```
n = 10000, a = 1
Sequential time: 1941 ms
Parallel time: 2254 ms, threads: 10
Acceleration: 0.8608521789704541
Efficiency: 0.0860852178970454
```

```
n = 25000, a = 1
Sequential time: 11603 ms
Parallel time: 11667 ms, threads: 5
Acceleration: 0.9945088127563614
Efficiency: 0.1989017625512723
```

```
n = 45000, a = 1
Sequential time: 34981 ms
Parallel time: 36046 ms, threads: 3
Acceleration: 0.9704485529512852
Efficiency: 0.3234828509837617
```

Також, на великих розмірах даних ефективність розпаралелення збільшується при зростанні кількості потоків:





Висновок:

Виконуючи лабораторну роботу, я реалізувала послідовний та паралельний варіанти алгоритму Дейкстри для пошуку найкоротших шляхів у зваженому графі. Порівняння часу виконання показало, що паралельний алгоритм є ефективнішим на великих розмірностях.