

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Лабораторна робота №7
з курсу «Паралельні та розподілені обчислення»

Виконав:
Гуменюк Станіслав
Група Пмі-33с

Оцінка ____
Перевірив:
Пасічник
Т.В.

2024

Завдання:

Для зваженого зв'язного неорієнтованого графа G , використовуючи алгоритм Прима, з довільно заданої вершини a побудувати мінімальне кісткове дерево. Для різної розмірності графів та довільного вузла a порахувати час виконання програми без потоків та при заданих k потоках розпаралелення

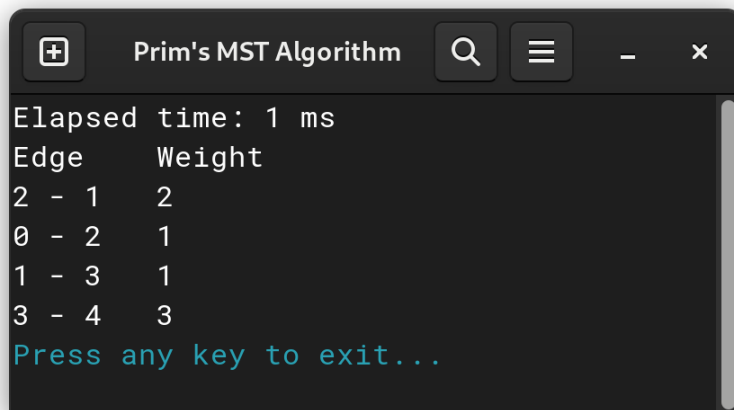
Програмна реалізація:

Програма написана на пакеті .NET з використанням мови C#. В меню є вибір функціоналу



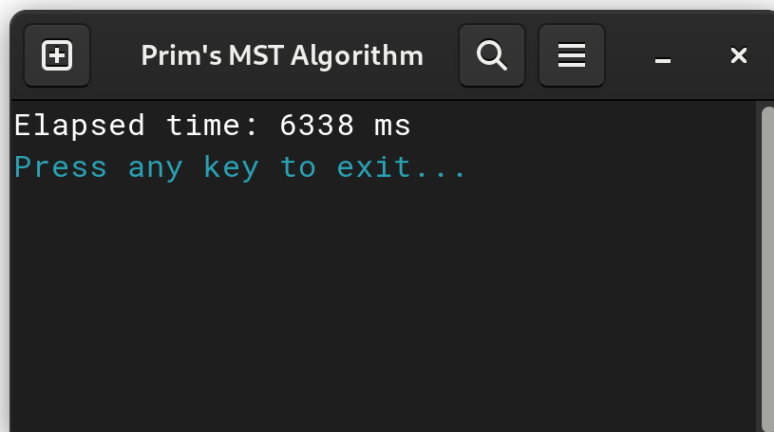
```
Prim's MST Algorithm
One thread on small
One thread on large
Multi threads on small
Multi threads on large
Difference
-> Best efficiency
Exit
```

Один потік на малому графі:



```
Prim's MST Algorithm
Elapsed time: 1 ms
Edge    Weight
2 - 1    2
0 - 2    1
1 - 3    1
3 - 4    3
Press any key to exit...
```

Один потік на великому графі (20000):



```
Prim's MST Algorithm
Elapsed time: 6338 ms
Press any key to exit...
```

Декілька потоків на малому графі:

```
Prim's MST Algorithm
Enter the number of threads:
5
Elapsed time: 5 ms
Edge    Weight
2 - 1    2
0 - 2    1
1 - 3    1
3 - 4    3
Press any key to exit...
```

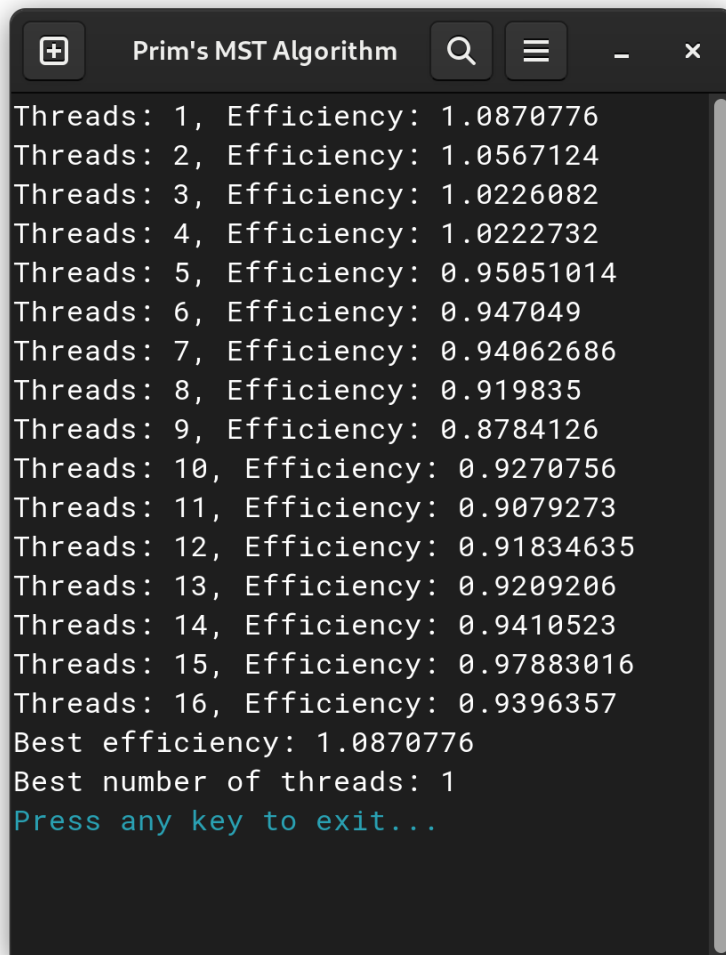
Декілька потоків на великому графі:

```
Prim's MST Algorithm
Enter the number of threads:
15
Elapsed time: 6528 ms
Press any key to exit...
```

Різниця між одним потоком і багатьма

```
Prim's MST Algorithm
Enter the number of threads:
15
Single-threaded: 6093 ms
Multi-threaded: 6702 ms
Difference: 0.9091316
Press any key to exit...
```

Різниця на різних кількостях потоків:



```
Prim's MST Algorithm
Threads: 1, Efficiency: 1.0870776
Threads: 2, Efficiency: 1.0567124
Threads: 3, Efficiency: 1.0226082
Threads: 4, Efficiency: 1.0222732
Threads: 5, Efficiency: 0.95051014
Threads: 6, Efficiency: 0.947049
Threads: 7, Efficiency: 0.94062686
Threads: 8, Efficiency: 0.919835
Threads: 9, Efficiency: 0.8784126
Threads: 10, Efficiency: 0.9270756
Threads: 11, Efficiency: 0.9079273
Threads: 12, Efficiency: 0.91834635
Threads: 13, Efficiency: 0.9209206
Threads: 14, Efficiency: 0.9410523
Threads: 15, Efficiency: 0.97883016
Threads: 16, Efficiency: 0.9396357
Best efficiency: 1.0870776
Best number of threads: 1
Press any key to exit...
```

Код послідовного алгоритму:

```
public static (Dictionary<int, int>, Dictionary<int, int>) PrimAlgorithm(
    Dictionary<int, Dictionary<int, int>> graph,
    int startVertex,
    out long elapsedMilliseconds)
{
    var stopwatch = Stopwatch.StartNew();
    var parent = new Dictionary<int, int>();
    var key = new Dictionary<int, int>();
    var mstSet = new Dictionary<int, bool>();

    foreach (var vertex in graph.Keys)
    {
        key[vertex] = int.MaxValue;
        mstSet[vertex] = false;
    }

    key[startVertex] = 0;
    parent[startVertex] = -1;

    for (int count = 0; count < graph.Count - 1; count++)
    {
        int u = MinKey(key, mstSet);
        if (u == -1 || !graph.ContainsKey(u)) continue;

        mstSet[u] = true;
        foreach (var v in graph[u].Keys)
        {
            if (!mstSet[v] && graph[u][v] < key[v])
            {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    stopwatch.Stop();
    elapsedMilliseconds = stopwatch.ElapsedMilliseconds;

    return (parent, key);
}
```

Код параллельного алгоритму:

```
var stopwatch = Stopwatch.StartNew();
var parent = new Dictionary<int, int>();
var key = new Dictionary<int, int>();
var mstSet = new Dictionary<int, bool>();
var lockObject = new object();

foreach (var vertex in graph.Keys)
{
    key[vertex] = int.MaxValue;
    mstSet[vertex] = false;
}
key[startVertex] = 0;
parent[startVertex] = -1;

var vertices = graph.Keys.Where(v => v != startVertex).ToList();
var chunkSize = (int)Math.Ceiling(vertices.Count / (double)k);
var resetEvents = new ManualResetEvent[k];
```

```
for (int i = 0; i < k; i++)
{
    resetEvents[i] = new ManualResetEvent(false);
    int threadIndex = i;

    new Thread(() =>
    {
        int start = threadIndex * chunkSize;
        int end = Math.Min(start + chunkSize, vertices.Count);

        for (int count = 0; count < end - start; count++)
        {
            int u;
            lock (lockObject)
            {
                u = MinKey(key, mstSet);
                if (u != -1 && graph.ContainsKey(u))
                    mstSet[u] = true;
            }

            if (u == -1 || !graph.ContainsKey(u)) continue;
        }
    })
```

```
        foreach (var v in graph[u].Keys)
        {
            if (!mstSet[v] && graph[u][v] < key[v])
            {
                lock (lockObject)
                {
                    if (!mstSet[v] && graph[u][v] < key[v])
                    {
                        parent[v] = u;
                        key[v] = graph[u][v];
                    }
                }
            }
        }
    }
}
```

```
resetEvents[threadIndex].Set();
}).Start();
```

```
WaitHandle.WaitAll(resetEvents);
```

```
// диспоз ManualResetEvent
foreach (var evt in resetEvents)
{
    evt.Dispose();
}
```

```
stopwatch.Stop();
elapsedMilliseconds = stopwatch.ElapsedMilliseconds;
return (parent, key);
```

```
private static int MinKey(Dictionary<int, int> key, Dictionary<int, bool> mstSet)
{
    int min = int.MaxValue;
    int minIndex = -1;

    foreach (var vertex in key.Keys)
    {
        if (!mstSet[vertex] && key[vertex] < min)
        {
            min = key[vertex];
            minIndex = vertex;
        }
    }

    return minIndex;
}
```

Висновок: Загалом розпаралелення алгоритму Прима має свій результат, проте конкретно на моїй машині процесор не є дуже потужним в багато потокових обчисленнях, але є потужним в однопотоці, тому результати показали інше