Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Высшая школа искусственного интеллекта**

**Лабораторная работа 6**

Обработка графов

по дисциплине «Параллельное программирование»

Выполнил

студент гр.3530203/80102 Л.Д. Челищева

<*подпись*>

Руководитель К.А. Туральчук

<*подпись*>

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

# Постановка задачи

**Вариант 2: Построение минимального остовного дерева (алгоритм Прима)**

Задачи:

1. Реализовать последовательный алгоритм.
2. Реализовать параллельный алгоритм.
3. Выполнить анализ эффективности при разном объеме данных и при разной плотности графов (разреженный граф, плотный граф). Построить графики зависимости ускорения алгоритма от объема данных.

# Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число вершин | Плотный граф | | Разреженный граф | |
| Последовательный | Параллельный | Последовательный | Параллельный |
| 200 | 7,631 | 44,575 | 66,984 | 61,111 |
| 500 | 124,057 | 113,603 | 353,552 | 116,074 |
| 1000 | 969,561 | 482,542 | 2640,378 | 741,044 |
| 2000 | 8468,977 | 4402,029 | 27999,145 | 7844,749 |

В таблице время в мс.

Тестирование выполнялось при 4 потоках. Граф хранился в виде таблицы смежности.

# Полный код программы

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Threading;

namespace lab6

{

class Program

{

static int[][] graph; //Исходный граф

static int n = 200; //Число вершин

static int maxw = 100; //Максимальный вес ребра

static List<int> V = new List<int>(); //Массив вершин вне дерева

static List<int> Vt = new List<int>(); //Массви вершин в дерева

static List<string> Result; //Ребра дерева

static double ResultSum; //Вес дерева

static int trcount = 4; // Число потоков

static Thread[] threads;

static (int, int, int)[] ThreadResult;

static AutoResetEvent[] Startevs;

static AutoResetEvent[] Finevs;

static bool stop;

static void Main(string[] args)

{

Console.WriteLine("Вариант 2. Алгоритм Прима\n10 итераций");

RandDenseGraph();//Плотный граф

//ShowArr();

Stopwatch stopWatch = new Stopwatch();

double stime = 0;

double ptime = 0;

for (int i=0; i<11; i++)

{

stopWatch.Start();

SerialPrim();

//ShowResult();

stopWatch.Stop();

if (i>0)

stime += stopWatch.Elapsed.TotalMilliseconds;

stopWatch.Reset();

stopWatch.Start();

ParallelPrim();

//ShowResult();

stopWatch.Stop();

if (i > 0)

ptime += stopWatch.Elapsed.TotalMilliseconds;

stopWatch.Reset();

}

Console.WriteLine("При {0} элементах последовательный подсчет в плотном графе занимет {1} мс", n, stime / 10);

Console.WriteLine("При {0} элементах и {1} потоках параллельный подсчет в плотном графе занимет {2} мс", n, trcount, ptime/10);

RandSparseGraph();//Разреженный граф

//ShowArr();

stime = 0;

ptime = 0;

for (int i = 0; i < 11; i++)

{

stopWatch.Start();

SerialPrim();

//ShowResult();

stopWatch.Stop();

if (i > 0)

stime += stopWatch.Elapsed.TotalMilliseconds;

stopWatch.Reset();

stopWatch.Start();

ParallelPrim();

//ShowResult();

stopWatch.Stop();

if (i > 0)

ptime += stopWatch.Elapsed.TotalMilliseconds;

}

Console.WriteLine("При {0} элементах последовательный подсчет в разреженном графе занимет {1}", n, stime / 10);

Console.WriteLine("При {0} элементах и {1} потоках параллельный подсчет в разреженном графе занимет {2}", n, trcount, ptime / 10);

Console.ReadKey();

}

static void SerialPrim()

{

ResultSum = 0;

Result = new List<string>();

V = new List<int>();

Vt = new List<int>();

int min;

int start = -1;

int fin = -1;

Vt.Add(0);

for (int i = 1; i < n; i++)

{

V.Add(i);

}

while (V.Count > 0)

{

min = maxw + 1;

foreach (int i in Vt)

{

foreach (int j in V)

{

if (graph[i][j] <= maxw)

if (graph[i][j] < min)

{

min = graph[i][j];

start = i;

fin = j;

}

}

}

if (maxw == min) throw new Exception();

ResultSum += min;

Result.Add("(" + start + "-" + fin + ")");

Vt.Add(fin);

V.Remove(fin);

}

}

static void ParallelPrim()

{

threads = new Thread[trcount];

ThreadResult = new (int, int, int)[trcount]; //(min, start, fin)

Startevs = new AutoResetEvent[trcount];

Finevs = new AutoResetEvent[trcount];

stop = false;

for (int i = 0; i < trcount; i++)

{

threads[i] = new Thread(ThreadWork);

Startevs[i] = new AutoResetEvent(false);

Finevs[i] = new AutoResetEvent(false);

}

ResultSum = 0;

Result = new List<string>();

V = new List<int>();

Vt = new List<int>();

int min;

int start = 0;

int fin = 0;

for (int i = 0; i < trcount; i++)

{

threads[i].Start(i);

}

Vt.Add(0);

for (int i = 1; i < n; i++)

{

V.Add(i);

}

while (V.Count > 0)

{

min = maxw + 1;

for (int i = 0; i < trcount; i++)

{

Startevs[i].Set();

}

for (int i = 0; i < trcount; i++)

{

Finevs[i].WaitOne();

}

for (int i = 0; i < trcount; i++)

{

if (ThreadResult[i].Item1 < min)

{

min = ThreadResult[i].Item1;

start = ThreadResult[i].Item2;

fin = ThreadResult[i].Item3;

}

}

if (maxw == min) throw new Exception();

ResultSum += min;

Result.Add("(" + start + "-" + fin + ")");

Vt.Add(fin);

V.Remove(fin);

}

stop = true;

for (int i = 0; i < trcount; i++)

{

Startevs[i].Set();

threads[i].Join();

}

}

static void ThreadWork(object o)

{

int num = (int)o;

int first;

int last;

int min;

int start;

int fin;

while (!stop)

{

Startevs[num].WaitOne();

if (!stop)

{

first = (int)Math.Ceiling((double)Vt.Count \* num / trcount);

last = (int)Math.Ceiling((double)(num + 1) \* Vt.Count / trcount) >= Vt.Count ? Vt.Count : (int)Math.Ceiling((double)(num + 1) \* Vt.Count / trcount);

min = maxw + 1;

start = -1;

fin = -1;

for (int i = first; i < last; i++)

foreach (int j in V)

{

if (graph[Vt[i]][j] <= maxw)

if (graph[Vt[i]][j] < min)

{

min = graph[Vt[i]][j];

start = Vt[i];

fin = j;

}

}

ThreadResult[num] = (min, start, fin);

}

Finevs[num].Set();

}

}

static void RandDenseGraph()

{

Random random = new Random(DateTime.Now.Millisecond);

graph = new int[n][];

for (int i = 0; i < n; i++)

{

graph[i] = new int[n];

}

for (int i = 0; i < n; i++)

{

for (int j = i + 1; j < n; j++)

{

graph[i][j] = random.Next(1, maxw);

graph[j][i] = graph[i][j];

}

graph[i][i] = maxw + 1;

}

}

static void RandSparseGraph()

{

Random random = new Random(DateTime.Now.Millisecond);

Random coin = new Random(DateTime.Now.Millisecond);

int count;

int max;

graph = new int[n][];

for (int i = 0; i < n; i++)

{

graph[i] = new int[n];

}

for (int i = 0; i < n; i++)

{

count = 0;

max = (n - i) / 2;

for (int j = i + 1; j < n - 1; j++)

{

if (coin.Next(10) % 2 == 1 && count <= max)

{

graph[i][j] = random.Next(1, maxw);

graph[j][i] = graph[i][j];

count++;

}

else

{

graph[i][j] = maxw + 1;

graph[j][i] = maxw + 1;

}

}

if (count == 0 || coin.Next(10) % 2 == 1)

{

graph[i][n - 1] = random.Next(1, maxw);

graph[n - 1][i] = graph[i][n - 1];

}

else

{

graph[i][n - 1] = maxw + 1;

graph[n - 1][i] = maxw + 1;

}

graph[i][i] = maxw + 1;

}

}

static void ShowArr()

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

Console.Write(graph[j][i] + "\t");

}

Console.WriteLine();

}

}

static void ShowResult()

{

Console.WriteLine("Дерево весом {0}:", ResultSum);

/\*foreach (string s in Result)

{

Console.WriteLine(s);

}\*/

}

}

}

# Выводы

В ходе лабораторной работы были рассмотрены последовательная и параллельная реализации алгоритма Прима, при этом в последней распараллеливался этап определения минимального не включенного в дерево ребра.

Согласно результатам тестирования, параллельный алгоритм начал показывать значительные преимущества от 500 ребер.