Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 2-4

Виконав:

студент 5 курсу ФІОТ

групи ІО-з91МП

Гріщенко К. C.

Перевірила:

Русанова О. В.

Київ – 2020

**Мета роботи** – реалізувати перший етап планування шляхом формування черг обчислювальних робіт.

**Вхідні дані –** коректний граф задачі (результат виконання лабораторної роботи №1), три задані алгоритми формування черг із таблиці 1.

**Виконання**

За варіантом обрано 2 алгоритми

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 3 | У порядку спадання критичного по часу шляхів до кінця графа задачі. | Критичний шлях по часу для всіх вершин |
| 15 | 2 | У порядку зростання ваги вершин | Вага вершин |

Сортування і алгоритми призначення реалізовано на серверній частині WEB застосунку.

**У порядку спадання критичного шляху**

Приклад роботи алгоритму для графу

1

1

3

101



Демонстрація результату сортування вершин.

Изображение выглядит как снимок экрана

Автоматически созданное описание

**У порядку зростання ваги вершини**

Приклад роботи алгоритму для графу

1

1

3

101



Демонстрація результату сортування вершин.

Изображение выглядит как снимок экрана

Автоматически созданное описание

**Вихідний код**

using System;

using System.Collections.Generic;

using System.Linq;

*public* interface IJobSorter

{

*IEnumerable*<*Job*> GetJobOrder(*TaskGraph* graph);

}

*public* class CriticalPathByTime : *IJobSorter*

{

*public* CriticalPathByTime()

{

}

*public* *IEnumerable*<*Job*> GetJobOrder(*TaskGraph* graph)

{

var taskOrder = new *List*<*Job*>();

var order = new *SortedList*<long, *Job*>(new *DuplicateKeyComparer*<long>());

var critPath = new *Dictionary*<*Job*, long>();

foreach(var finishTask in graph.FinishTasks)

{

fillCriticalPath(finishTask, 0, critPath);

}

foreach(var kvp in critPath)

{

var job = kvp.Key;

var crit = kvp.Value;

order.Add(crit, job);

}

return order.Select( kvp => kvp.Value).Reverse();

}

*private* void fillCriticalPath(*Job* job, long depth, *Dictionary*<*Job*, long> jobOrder)

{

var addedDepth = job.Weight;

var totalDepth = addedDepth + depth;

foreach(var childJob in job.Parents)

{

fillCriticalPath(childJob.RelatedJob, totalDepth + childJob.Weight, jobOrder);

}

var crit = totalDepth;

if(jobOrder.TryGetValue(job, *out* long critPath))

{

crit = Math.Min(critPath, crit);

}

jobOrder.Add(job, crit);

}

}

*public* class ByWeight : *IJobSorter*

{

*public* *IEnumerable*<*Job*> GetJobOrder(*TaskGraph* graph)

{

var jobs = graph.Select(j => j);

return jobs.OrderBy( j=> j.Weight);

}

}

*public* class DuplicateKeyComparer<TKey>

:

*IComparer*<*TKey*> where *TKey* : *IComparable*

{

#region IComparer<TKey> Members

*public* int Compare(*TKey* x, *TKey* y)

{

int result = x.CompareTo(y);

if (result == 0)

return 1; *// Handle equality as beeing greater*

else

return result;

}

#endregion

}