**Міністерство освіти України**

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Факультет інформатики та обчислювальної техніки**

**ЗВІТ**

до лабораторної роботи № 8

на тему: “Програмна реалізація алгоритму для вирішення задачі мінімізації сумарного зваженого відхилення відносно директивних термінів”

з дисципліни “Математичні методи оптимізації”

Виконала:

Студентка групи ІП-71

Каспрук Анастасія Андріївна

Київ 2020

**Опис реалізації**

Реалізація побудована на базі лекційного матеріалу.

План виконання задач представляє собою клас **Plan** . У своїй структурі він містить колекцію **Tasks**, в якій послідовно розміщені задачі для виконання, а також величину штрафу **CurrentFine** для поточного плану.

Задачі представлені об’єктами класу **Task** . Задача у свої структурі містить наступну інформацію:

* **Number** - номер завдання;
* **Duration** - тривалість виконання;
* **Term** - директивний строк виконання;
* **InitialFineForEarlier** - штраф за виконання раніше директивного строку, що задається на початку;
* **InitialFineForLater** - штраф за виконання пізніше директивного строку, що задається на початку;
* **StartMoment** - момент початку виконання завдання в тому плані, у якому на поточний момент знаходиться об’єкт завдання;
* **CurrentFine** – поточний штраф за завдання в тому плані, у якому на поточний момент знаходиться об’єкт завдання.

Алгоритм реалізований у методі **GetBestPlan** класу **Divider** . Клас **Divider** містить 3 приватних поля:

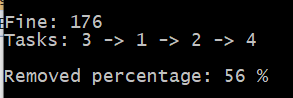
* **\_tasks** – колекція зі всіма вхідними задачами;
* **\_currentPlans** – колекція зі всіма можливими поточними планами виконання;
* **\_record** – поточний рекорд.

На початку виконання алгоритму кожна задача по одній розподіляється по планам і дані плани додаються в колекцію **\_currentPlans** .

На кожному кроці виконання алгоритму з колекції **\_currentPlans** відбирається план з найменшим штрафом **bestPlan** і від нього відбувається розгалуження. Отримані в результаті розгалуження плани поміщаються у колекцію **\_currentPlans** , а **bestPlan** видаляється з **\_currentPlans** . Також з **\_currentPlans** видаляються усі плани, для яких значення штрафу більше за значення штрафу рекорду **\_record** . Так відбувається до тих пір, поки у колекції **\_currentPlans** не залишаться лише ті плани, які включають у себе всі завдання та для яких значення штрафу буде рівним значенню штрафу рекорду.

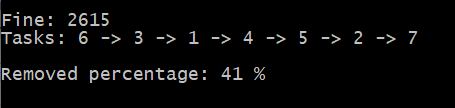
**Приклад виконання**

Завдання варіанту 11:



Штраф складає 176 одиниць. План виконання завдань: 3 -> 1 -> 2 -> 4. Відсоток розв’язків, які відсіклися процедурою тесту: 56 %.

Розподіл для рандомно згенерованих 7 завдань:



Штраф складає 2615 одиниць. План виконання завдань: 6 -> 3 -> 1 -> 4 -> 5 -> 2 -> 7. Відсоток розв’язків, які відсіклися процедурою тесту: 41 %.

**Програмний код**

1. Файл Task.cs.

using System;

using System.Text;

namespace BranchesAndBoundaries

{

/// <summary>

/// Завдання

/// </summary>

public class Task

{

/// <summary>

/// Номер завдання

/// </summary>

public int Number { get; }

/// <summary>

/// Тривалість виконання

/// </summary>

public int Duration { get; }

/// <summary>

/// Директивний строк виконання

/// </summary>

public int Term { get; }

/// <summary>

/// Штраф за виконання раніше директивного строку

/// </summary>

public int InitialFineForEarlier { get; }

/// <summary>

/// Штраф за виконання пізніше директивного строку

/// </summary>

public int InitialFineForLater { get; }

/// <summary>

/// Момент початку виконання завдання

/// </summary>

public int StartMoment { get; set; }

/// <summary>

/// Поточний штраф

/// </summary>

public int CurrentFine

{

get

{

int deviation = Term - (Duration + StartMoment);

if (deviation > 0)

return deviation \* InitialFineForEarlier;

return Math.Abs(deviation) \* InitialFineForLater;

}

}

public Task(int number,

int duration,

int term,

int initialFineForEarlier,

int initialFineForLater)

{

Number = number;

Duration = duration;

Term = term;

InitialFineForEarlier = initialFineForEarlier;

InitialFineForLater = initialFineForLater;

}

public Task Clone()

{

return this.MemberwiseClone() as Task;

}

public override string ToString()

{

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.AppendLine($"Number: {Number}");

stringBuilder.AppendLine($"Duration: {Duration}");

stringBuilder.AppendLine($"Term: {Term}");

stringBuilder.AppendLine($"Fine for earlier: {InitialFineForEarlier}");

stringBuilder.AppendLine($"Fine for later: {InitialFineForLater}");

stringBuilder.AppendLine($"Start moment: {StartMoment}");

stringBuilder.AppendLine($"Curent fine: {CurrentFine}");

return stringBuilder.ToString();

}

}

}

1. Файл Plan.cs.

using System.Collections.Generic;

using System.Linq;

namespace BranchesAndBoundaries

{

/// <summary>

/// План

/// </summary>

public class Plan

{

/// <summary>

/// Завдання, які входять до плану

/// </summary>

public LinkedList<Task> Tasks { get; }

/// <summary>

/// Поточний штраф для плану

/// </summary>

public int CurrentFine => Tasks.Sum(t => t.CurrentFine);

public Plan(Task appendedTask, LinkedList<Task> currentPlan = null)

{

if (currentPlan == null)

currentPlan = new LinkedList<Task>();

if(appendedTask != null)

currentPlan.AddLast(appendedTask);

LinkedListNode<Task> currentTask = currentPlan.First;

int currentDuration = 0;

while(currentTask.Next != null)

{

currentDuration += currentTask.Value.Duration;

currentTask.Next.Value.StartMoment = currentDuration;

currentTask = currentTask.Next;

}

Tasks = currentPlan;

}

public Plan Clone()

{

LinkedList<Task> cloneTaskList = new LinkedList<Task>();

foreach(Task task in Tasks)

{

cloneTaskList.AddLast(task.Clone());

}

return new Plan(null, cloneTaskList);

}

public override string ToString()

{

return $"Fine: {CurrentFine}\n" +

$"Tasks: {string.Join(" -> ", Tasks.Select(t => t.Number))}";

}

}

}

1. Файл Divider.cs.

using System.Collections.Generic;

using System.Linq;

namespace BranchesAndBoundaries

{

public class Divider

{

private IReadOnlyList<Task> \_tasks;

private List<Plan> \_currentPlans;

private Plan \_record;

public Divider(List<Task> tasks)

{

\_tasks = tasks;

\_currentPlans = new List<Plan>();

}

public (Plan, float) GetBestPlan()

{

foreach(Task task in \_tasks)

{

\_currentPlans.Add(new Plan(task.Clone()));

}

Plan bestPlan = null;

List<int> bestPlanTaskNumbers = null;

int allPlansCount = \_currentPlans.Count;

int removedPlansCount = 0;

int currentMinFine;

List<Plan> recordCandidates;

while (!\_currentPlans.All(p => p.Tasks.Count == \_tasks.Count && p.CurrentFine == \_record.CurrentFine))

{

currentMinFine = \_currentPlans.Min(p => p.CurrentFine);

recordCandidates = \_currentPlans.Where(plan => plan.Tasks.Count == \_tasks.Count).ToList();

\_record = recordCandidates.FirstOrDefault(c => c.CurrentFine == recordCandidates.Min(rc => rc.CurrentFine));

if (\_record != null)

{

removedPlansCount += \_currentPlans.RemoveAll(p => p.CurrentFine > \_record.CurrentFine ||

(p.CurrentFine == \_record.CurrentFine && p.Tasks.Count < \_tasks.Count));

}

bestPlan = \_currentPlans.FirstOrDefault(

plan => plan.CurrentFine == currentMinFine && plan.Tasks.Count < \_tasks.Count);

if (bestPlan == null)

continue;

bestPlanTaskNumbers = bestPlan.Tasks.Select(t => t.Number).ToList();

foreach (Task task in \_tasks.Where(t => !bestPlanTaskNumbers.Any(n => n == t.Number)))

{

\_currentPlans.Add(new Plan(task.Clone(), bestPlan.Clone().Tasks));

allPlansCount++;

}

if(\_currentPlans.Count != 1)

\_currentPlans.Remove(bestPlan);

}

return (\_currentPlans.First(), removedPlansCount \* 100 / allPlansCount);

}

}

}

1. Файл Program.cs.

using System;

using System.Collections.Generic;

namespace BranchesAndBoundaries

{

class Program

{

static void Main(string[] args)

{

var random = new Random();

int randomTasksCount = random.Next(5, 8);

List<Task> initialTasks = new List<Task>(randomTasksCount);

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

{

initialTasks.Add(

new Task(

number: i,

duration: random.Next(5, 25),

term: random.Next(15, 50),

initialFineForEarlier: random.Next(3, 20),

initialFineForLater: random.Next(3, 20)));

}

//List<Task> initialTasks = new List<Task>

//{

// new Task(

// number: 1,

// duration: 12,

// term: 26,

// initialFineForEarlier: 9,

// initialFineForLater: 1),

// new Task(

// number: 2,

// duration: 9,

// term: 33,

// initialFineForEarlier: 8,

// initialFineForLater: 4),

// new Task(

// number: 3,

// duration: 8,

// term: 26,

// initialFineForEarlier: 5,

// initialFineForLater: 3),

// new Task(

// number: 4,

// duration: 8,

// term: 37,

// initialFineForEarlier: 4,

// initialFineForLater: 9)

//};

var divider = new Divider(initialTasks);

(Plan bestPlan, float removedPercentage) = divider.GetBestPlan();

Console.WriteLine();

Console.WriteLine(bestPlan.ToString());

Console.WriteLine();

Console.WriteLine($"Removed percentage: {removedPercentage} %");

}

}

}