|  |  |  |  |
| --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  **Кафедра програмних систем і технологій**  Дисципліна  **«Структури даних, аналіз і алгоритми комп'ютерної обробки інформації»**  **Лабораторна робота №1**  **«Динамічне програмування»** | | | |
| **Виконав**: | Самойленко Олександр Васильович | **Перевірив**: | Бичков Олексій Сергійович |
| **Група** | ІПЗ-21 | **Дата перевірки** |  |
| **Форма навчання** | денна | **Оцінка** |  |
| **Спеціальність** | 121 |
| 2022 | | | |

# Завдання:

1. Написати програму для розв'язання задачі про рюкзак (додаткове заняття)

* Розв'яжіть задачу про рюкзак для випадку, коли є необмежена кількість предметів кожного типу
* Розв'яжіть завдання про рюкзак для випадку, коли предмети можна брати не повністю (не золоті злитки, а золотий пісок). Цікаво зробити для не однакових одиниць, наприклад, об’єм в м3 та вага в кг.
* Розв'яжіть змішане завдання про рюкзак - частину предметів можна брати тільки повністю, а решту - можна і не повністю

2. Написати програму розв'язання задачі про дужки

Задано рядок, що складається з дійсних чисел, що розділені арифметичними операціями. Потрібно розставити в рядку дужки таким чином, щоб значення отриманого виразу було максимальним. Вхідні дані - рядок, вихідні дані - рядок та результат арифметичного виразу.

3. Оптимальна тріангуляція.

* Нехай вартістю трикутника вважається його площа. Написати програму знаходження оптимальної тріангуляції.
* Нехай необхідно мінімізувати сумарну довжину проведених діагоналей. Написати програму знаходження оптимальної тріангуляції.

# Код програми:

# №1

public abstract class Item  
{  
 public double Price { get; set; }  
 public double Weight { get; set; }  
  
 public string Tag { get; set; }  
  
 public double GetPriceToWeight()  
 {  
 var ptw = Price / Weight;  
 return ptw;  
 }  
 public abstract bool TryGetItemValue(double availableSpace, out ItemValue itemValue);  
}

public class ItemValue  
{  
 public Item ParentItem { get; set; }  
 public double ResultWeight { get; set; }  
 public double ResultPrice { get; set; }  
}

public class UndividedItem : Item  
{  
 public override bool TryGetItemValue(double availableSpace, out ItemValue itemValue)  
 {  
 if (availableSpace >= Weight)  
 {  
 itemValue = new ItemValue()  
 {  
 ResultPrice = Price,  
 ResultWeight = Weight,  
 ParentItem = this  
 };  
 return true;  
 }  
  
 itemValue = null;  
 return false;  
 }  
}

public class DividedItem : Item  
{  
 public override bool TryGetItemValue(double availableSpace, out ItemValue itemValue)  
 {  
 if (availableSpace >= Weight)  
 {  
 itemValue = new ItemValue()  
 {  
 ResultPrice = Price,  
 ResultWeight = Weight,  
 ParentItem = this  
 };  
 return true;  
 }  
  
 itemValue = new ItemValue()  
 {  
 ResultPrice = availableSpace / Weight \* Price,  
 ResultWeight = availableSpace,  
 ParentItem = this  
 };  
 return true;  
 }  
}

public static class Task1  
{  
 private static List<UndividedItem> GetItems(string line)  
 {  
 var result = new List<UndividedItem>();  
 var pairs = line.Split(',', StringSplitOptions.TrimEntries);  
 foreach (var pair in pairs)  
 {  
 var values = pair.Replace(",", "").Split(' ', StringSplitOptions.TrimEntries);  
 int.TryParse(values[0], out var price);  
 int.TryParse(values[1], out var weight);  
 var item = new UndividedItem()  
 {  
 Price = price,  
 Weight = weight  
 };  
 result.Add(item);  
 }  
  
 return result;  
 }  
  
 private static void InitiateTable(List<List<int>> array, int width, int height)  
 {  
 array.Clear();  
 for (int i = 0; i < height; i++)  
 {  
 var l = new List<int>();  
 for (int j = 0; j < width; j++)  
 l.Add(0);  
 array.Add(l);  
 }  
 }  
   
 public static void ForUndivided()  
 {  
 var totalWeight = InputOutput.Input("Please, enter backpack capacity");  
 Console.WriteLine("Input prices and weights like: p1 w1, p2 w2, p3 w3");  
 var items = GetItems(Console.ReadLine()!);  
 items.Insert(0, new UndividedItem(){Price = 0, Weight = 0});  
 var worthMatrix = new List<List<int>>();  
 InitiateTable(worthMatrix, totalWeight + 1, items.Count +1);  
 for(int i = 1; i <= items.Count - 1; i++) // s -номер поточного предмета  
 {  
 for(int n = 0; n <= totalWeight; n++) // n -вага поточного рюкзака  
 {  
 worthMatrix[i][n] = worthMatrix[i - 1][n]; //Припускаємо, що s не беремо  
 if (items[i].Weight <= n   
 && worthMatrix[i][n] < items[i].Price + worthMatrix[i - 1][n-(int)items[i].Weight])  
 {  
 worthMatrix[i][n] = worthMatrix[i - 1][n - (int)items[i].Weight] + (int)items[i].Price;  
 }   
 }  
 }  
 Console.WriteLine("Items to take:");  
 Print(worthMatrix, items, items.Count - 1, totalWeight);  
 }  
   
 private static void Print(List<List<int>> A, List<UndividedItem> items, int s, int n)  
 {  
 if (A[s][n]==0)   
 return;   
 if (A[s-1][n] == A[s][n])  
 Print(A, items, s - 1,n);   
 else  
 {  
 Print(A, items, s-1,n - (int)items[s].Weight);  
 Console.WriteLine(s);   
 }  
 }  
   
 private static List<Item> GetAnyItems(string line)  
 {  
 var result = new List<Item>();  
 var pairs = line.Split(',', StringSplitOptions.TrimEntries);  
 var iterator = 1;  
 foreach (var pair in pairs)  
 {  
 var values = pair.Replace(",", "").Split(' ', StringSplitOptions.TrimEntries);  
 if (values.Length <= 2 || values[2] == "0" || values[2] == "f")  
 {  
 double.TryParse(values[0], out var undividedPrice);  
 double.TryParse(values[1], out var undividedWeight);  
 var undividedItem = new UndividedItem()  
 {  
 Price = (int)undividedPrice,  
 Weight = (int)undividedWeight,  
 Tag = iterator.ToString()  
 };  
 result.Add(undividedItem);  
 }  
 else  
 {  
 double.TryParse(values[0], out var dividedPrice);  
 double.TryParse(values[1], out var dividedWeight);  
 var dividedItem = new DividedItem()  
 {  
 Price = (int)dividedPrice,  
 Weight = (int)dividedWeight,  
 Tag = iterator.ToString()  
 };  
 result.Add(dividedItem);  
 }  
  
 iterator++;  
 }  
  
 return result;  
 }  
   
 public static void ForDivided()  
 {  
 var totalWeight = InputOutput.InputDouble("Please, enter backpack capacity");  
 Console.WriteLine("Input prices and weights like: p1 w1 1, p2 w2 0, p3 w3 0 | Where 0 for undivided item and 1 for divided");  
 var items = GetAnyItems(Console.ReadLine()!)  
 .OrderByDescending(item => item.GetPriceToWeight())  
 .ToList();  
 var resultValues = new List<ItemValue>();  
 var capacityLeft = totalWeight;  
 foreach (var item in items)  
 {  
 if (!item.TryGetItemValue(capacityLeft, out var itemValue))  
 {  
 continue;  
 }  
 resultValues.Add(itemValue);  
 capacityLeft -= itemValue.ResultWeight;  
 if (capacityLeft == 0)  
 {  
 break;  
 }  
 }  
  
 foreach (var res in resultValues)  
 {  
 Console.WriteLine(res.ParentItem is UndividedItem  
 ? $"Undivided number {res.ParentItem.Tag}, price: {res.ResultPrice}, weight: {res.ResultWeight}"  
 : $"Divided number {res.ParentItem.Tag}, price {res.ResultPrice}, weight: {res.ResultWeight} " +  
 $"in proportion: {res.ResultPrice / (res.ParentItem as DividedItem)!.Price}");  
 }  
 }  
}

# №2

public class Operand  
{  
 public double Value { get; set; }  
 public string Predecessor { get; set; }  
  
 public static Operand operator +(Operand left, Operand right)  
 {  
 return new Operand()  
 {  
 Value = left.Value + right.Value,  
 Predecessor = $"({left.Predecessor} + {right.Predecessor})"  
 };  
 }  
   
 public static Operand operator -(Operand left, Operand right)  
 {  
 return new Operand()  
 {  
 Value = left.Value - right.Value,  
 Predecessor = $"({left.Predecessor} - {right.Predecessor})"  
 };  
 }  
   
 public static Operand operator \*(Operand left, Operand right)  
 {  
 return new Operand()  
 {  
 Value = left.Value \* right.Value,  
 Predecessor = $"({left.Predecessor} \* {right.Predecessor})"  
 };  
 }  
   
 public static Operand operator /(Operand left, Operand right)  
 {  
 if (right.Value == 0)  
 throw new DivideByZeroException();  
 return new Operand()  
 {  
 Value = left.Value / right.Value,  
 Predecessor = $"({left.Predecessor} / {right.Predecessor})"  
 };  
 }  
}

public static class Task2  
{  
 public static void Run()  
 {  
 Console.WriteLine("Please, enter an expression:");  
 var expression = Console.ReadLine()!;  
 var maxOperand = GetAllPossibleValues(expression)  
 .OrderByDescending(operand => operand.Value)  
 .First();  
 Console.WriteLine($"{maxOperand.Predecessor} = {maxOperand.Value}");  
 }  
   
 private static List<Operand> GetAllPossibleValues(string input)   
 {  
 var result = new List<Operand>();  
 var hasDivisionByZeroHappened = false;  
   
 for (int i = 0; i < input.Length; i++)   
 {  
 if (!input[i].IsOperator()) {  
 continue;  
 }  
  
 try  
 {  
 var leftPossibleValues = GetAllPossibleValues(input.Substring(0, i));  
 var rightPossibleValues = GetAllPossibleValues(input.Substring(i + 1));  
  
 foreach (var left in leftPossibleValues)  
 {  
 foreach (var right in rightPossibleValues)  
 {  
 var val = Evaluate(left, right, input[i]);  
 result.Add(val);  
 }  
 }  
 }  
 catch (DivideByZeroException)  
 {  
 hasDivisionByZeroHappened = true;  
 }  
 }  
   
 if (!result.Any() && !hasDivisionByZeroHappened)   
 {  
 result.Add(new Operand()  
 {  
 Value = double.Parse(input),   
 Predecessor = input  
 });  
 }  
 return result;  
 }  
  
 private static Operand Evaluate(Operand left, Operand right, char op)  
 {  
 switch (op)  
 {  
 case '+':  
 return left + right;  
 case '-':  
 return left - right;  
 case '\*':  
 return left \* right;  
 case '/':  
 return left / right;  
 }  
 throw new InvalidOperationException();  
 }  
  
 private static bool IsOperator(this char op)   
 {  
 return op is '+' or '-' or '\*' or '/';  
 }  
}

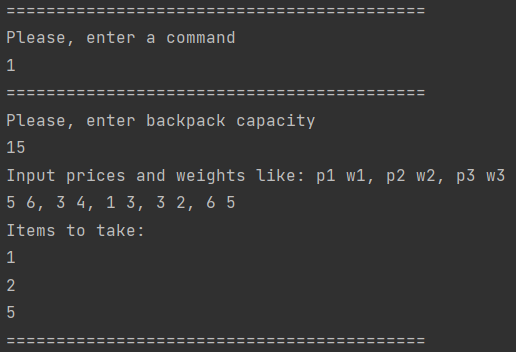
# №3

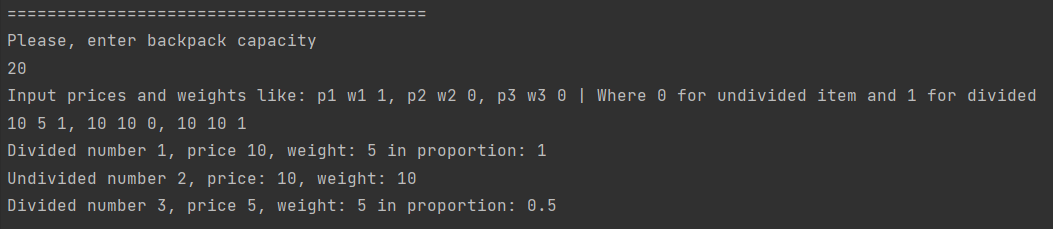
public class Vertex  
{  
 public double X { get; set; }  
 public double Y { get; set; }  
  
 public double DistanceTo(Vertex anotherVertex)  
 {  
 var distance = Math.Sqrt(Math.Pow(X - anotherVertex.X, 2) + Math.Pow(Y - anotherVertex.Y, 2));  
 return distance;  
 }  
}

public static class Task3  
{  
 public static List<Vertex> GetVertices(string line)  
 {  
 var result = new List<Vertex>();  
 var pairs = line.Split(',', StringSplitOptions.TrimEntries);  
 foreach (var pair in pairs)  
 {  
 var values = pair.Replace(",", "").Split(' ', StringSplitOptions.TrimEntries);  
 double.TryParse(values[0], out var x);  
 double.TryParse(values[1], out var y);  
 var vertex = new Vertex()  
 {  
 X = x,  
 Y = y  
 };  
 result.Add(vertex);  
 }  
  
 return result;  
 }  
   
 private static void InitiateTable(List<List<(double, int)>> array, int width, int height)  
 {  
 array.Clear();  
 for (int i = 0; i < height; i++)  
 {  
 var l = new List<(double, int)>();  
 for (int j = 0; j < width; j++)  
 {  
 if(i > j)  
 l.Add((Double.MinValue, -1));  
 else  
 l.Add((0, -1));  
 }   
 array.Add(l);  
 }  
 }  
   
 public static void Run(Func<Vertex, Vertex, Vertex, double> worthFunc)  
 {  
 Console.WriteLine("Input vertices coords: x1 y1, x2 y2, x3 y3...");  
 var vertices = GetVertices(Console.ReadLine()!);  
 var worthMatrix = new List<List<(double Value, int K)>>();  
 InitiateTable(worthMatrix, vertices.Count, vertices.Count);  
 for (int i = 0; i < vertices.Count; i++)  
 {  
 for (int j = i + 1; j < vertices.Count; j++)  
 {  
 var minWorth = (double.MaxValue, -1);  
 for (int k = i + 1; k < j; k++)  
 {  
 var funcResult = worthFunc(vertices[i], vertices[j], vertices[k]);  
 var currentWorth = worthMatrix[i][k].Value  
 + worthMatrix[k + 1][j].Value   
 + funcResult;  
 if (currentWorth < minWorth.MaxValue && currentWorth != 0)  
 {  
 minWorth.MaxValue = currentWorth;  
 minWorth.Item2 = k;  
 }  
 }  
   
 worthMatrix[i][j] = Math.Abs(minWorth.MaxValue - double.MaxValue) < 0.0003 ? (0, -1) : minWorth;   
 }  
 }  
   
 Console.WriteLine("Optimal triangles:");  
 int outI = 0;  
 int outJ = vertices.Count - 1;  
 while (outJ - outI > 1)  
 {  
 var worth = worthMatrix[outI][outJ];  
 Console.WriteLine($"{vertices[outI].X} {vertices[outI].Y}\t" +  
 $"{vertices[outJ].X} {vertices[outJ].Y}\t" +  
 $"{vertices[worth.K].X} {vertices[worth.K].Y}\t{worth.Value}");  
 outI = worth.K;  
 }  
 }  
  
 public static double GetAreaWorth(Vertex v1, Vertex v2, Vertex v3)  
 {  
 double a = v1.DistanceTo(v2);  
 double b = v2.DistanceTo(v3);  
 double c = v3.DistanceTo(v1);  
 double p = (a + b + c) / 2;  
 return Math.Sqrt(p \* (p-a) \* (p-b) \* (p-c));  
 }  
  
 public static double GetDiagonalLengthWorth(Vertex v1, Vertex v2, Vertex v3)  
 {  
 double a = v1.DistanceTo(v3);  
 double b = v2.DistanceTo(v3);  
 return a + b;  
 }  
}

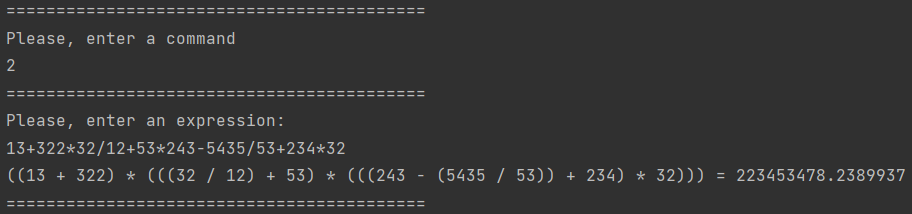
# Результат виконання:

# №1

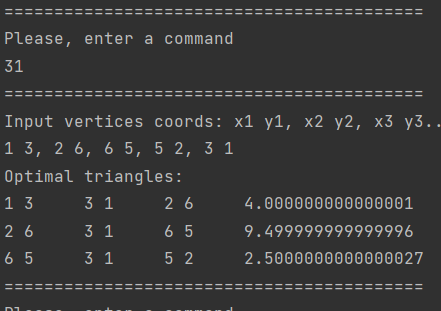


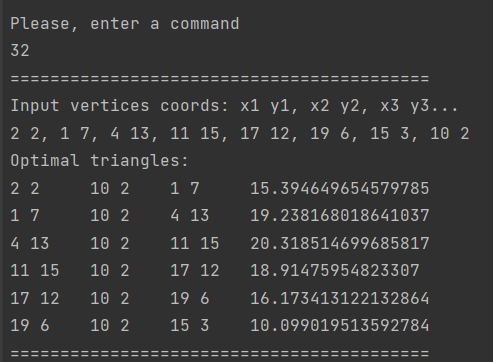


# №2



# №3





# Висновки:

Було виконано завдання лабораторної роботи номер 1, побудовано алгоритми розв’язання задачі про рюкзак, обчислення найбільшого можливого значення виразу та оптимальної тріангуляції за площею та загальною довжиною проведених діагоналей.