**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 4 з дисципліни

«Проектування алгоритмів»

„**Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*Гавриленко Я.С*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2020

Зміст

[1 Мета лабораторної роботи 3](#_Toc52291748)

[2 Завдання 4](#_Toc52291749)

[3 Виконання 10](#_Toc52291750)

[3.1 Покроковий алгоритм 10](#_Toc52291751)

[3.2 Програмна реалізація алгоритму 10](#_Toc52291752)

[3.2.1 Вихідний код 10](#_Toc52291753)

[3.2.2 Приклади роботи 10](#_Toc52291754)

[3.3 Тестування алгоритму 11](#_Toc52291755)

[Висновок 12](#_Toc52291756)

[Критерії оцінювання 13](#_Toc52291757)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи розробки метаеврестичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

# Завдання

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім степенем деталізації.

Виконати його програмну реалізацію на будь-якій мові програмування.

Перелік задач наведено у таблиці 2.1.

Перелік алгоритмів і досліджуваних параметрів у таблиці 2.2.

Задача і алгоритм наведені в таблиці 2.3.

Змінюючи параметри алгоритму, визначити кращі вхідні параметри алгоритму. Для цього необхідно:

* обрати критерій зупинки алгоритму (кількість ітерацій або значення ЦФ);
* зафіксувати усі параметри крім одного і змінювати цей параметр, поки не буде досягнуто пікової ефективності;
* після цього параметр фіксується і змінюються інші параметри;
* далі повторюємо процедуру спочатку, з першого зафіксованого параметру;
* зупиняємось коли будуть знайдені оптимальні параметри для даної задачі або встановлена залежність одних параметрів від інших.

Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

|  |  |
| --- | --- |
| **№** | **Задача** |
| 1 | **Задача про рюкзак** (місткість P=500, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 20 (випадкова)). Для заданої множини предметів, кожен з яких має вагу і цінність, визначити яку кількість кожного з предметів слід взяти, так, щоб сумарна вага не перевищувала задану, а сумарна цінність була максимальною.  Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика. |
| 2 | **Задача комівояжера** (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів.  **Розглядається симетричний, асиметричний та змішаний варіанти.**  В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напряму, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.  У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.  У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху.  Застосування:   * доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); * доставка води; * моніторинг об'єктів; * поповнення банкоматів готівкою; * збір співробітників для доставки вахтовим методом. |
| 3 | **Розфарбовування графа** (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають хроматичне число.  Застосування:   * розкладу для освітніх установ; * розкладу в спорті; * планування зустрічей, зборів, інтерв'ю; * розклади транспорту, в тому числі - авіатранспорту; * розкладу для комунальних служб; |
| 4 | **Задача вершинного покриття** (300 вершин, степінь вершини не більше 30, але не менше 2)**.** Вершинне покриття для неорієнтованого графа G = (V, E) - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S.  Задача вершинного покриттяполягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).  На вході: Граф G = (V, E).  Результат: множина C ⊆ V - найменше вершинне покриття графа G.    Застосування:   * розміщення пунктів обслуговування; * призначення екіпажів на транспорт; * проектування інтегральних схем і конвеєрних ліній. |
| 5 | **Задача про кліку** (300 вершин, степінь вершини не більше 30, але не менше 2)**.** Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній.  Задача про кліку існує у двох варіантах: у **задачі розпізнавання** потрібно визначити, чи існує в заданому графі G кліка розміру k, тоді як в **обчислювальному варіанті** потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити).  Застосування:   * біоінформатика; * електротехніка; |
| 6 | **Задача про найкоротший шлях** (300 вершин, відстань між вершинами випадкова від 5 до 150, степінь вершини не більше 10, але не менше 1) - задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.  Важливість задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між точкою відправлення і точкою призначення. Як вершин виступають перехрестя, а дороги є ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший. |

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

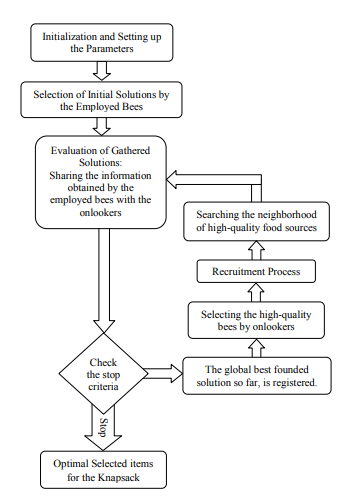
|  |  |
| --- | --- |
| **№** | **Алгоритми і досліджувані параметри** |
| 1 | **Генетичний алгоритм:**   * оператор схрещування (мінімум 3); * мутація (мінімум 2); * оператор локального покращення (мінімум 2). |
| 2 | **Мурашиний алгоритм**:   * α; * β; * ρ; * Lmin; * кількість мурах М і їх типи (елітні, тощо…); * маршрути з однієї чи різних вершин. |
| 3 | **Бджолиний алгоритм:**   * кількість ділянок; * кількість бджіл (фуражирів і розвідників). |

Таблиця 2.3 – Варіанти задач і алгоритмів

|  |  |
| --- | --- |
| **№** | **Задачі і алгоритми** |
| 1 | Задача про рюкзак + Генетичний алгоритм |
| 2 | Задача про рюкзак + Бджолиний алгоритм |
| 3 | Задача комівояжера (асиметрична мережа) + Генетичний алгоритм |
| 4 | Задача комівояжера (симетрична мережа) + Генетичний алгоритм |
| 5 | Задача комівояжера (змішана мережа) + Генетичний алгоритм |
| 6 | Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм |
| 7 | Задача комівояжера (симетрична мережа) + Мурашиний алгоритм |
| 8 | Задача комівояжера (змішана мережа) + Мурашиний алгоритм |
| 9 | Задача вершинного покриття + Генетичний алгоритм |
| 10 | Задача вершинного покриття + Бджолиний алгоритм |
| 11 | Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм |
| 12 | Задача комівояжера (симетрична мережа) + Бджолиний алгоритм |
| 13 | Задача комівояжера (змішана мережа) + Бджолиний алгоритм |
| 14 | Розфарбовування графа + Генетичний алгоритм |
| 15 | Розфарбовування графа + Бджолиний алгоритм |
| 16 | Задача про кліку (задача розпізнавання) + Генетичний алгоритм |
| 17 | Задача про кліку (задача розпізнавання) + Бджолиний алгоритм |
| 18 | Задача про кліку (обчислювальна задача) + Генетичний алгоритм |
| 19 | Задача про кліку (обчислювальна задача) + Бджолиний алгоритм |
| 20 | Задача про найкоротший шлях + Генетичний алгоритм |
| 21 | Задача про найкоротший шлях + Мурашиний алгоритм |
| 22 | Задача про найкоротший шлях + Бджолиний алгоритм |
| 23 | Задача про рюкзак + Генетичний алгоритм |
| 24 | Задача про рюкзак + Бджолиний алгоритм |
| 25 | Задача комівояжера (асиметрична мережа) + Генетичний алгоритм |
| 26 | Задача комівояжера (симетрична мережа) + Генетичний алгоритм |
| 27 | Задача комівояжера (змішана мережа) + Генетичний алгоритм |
| 28 | Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм |
| 29 | Задача комівояжера (симетрична мережа) + Мурашиний алгоритм |
| 30 | Задача комівояжера (змішана мережа) + Мурашиний алгоритм |

# Виконання

## Покроковий алгоритм



## Програмна реалізація алгоритму

### Вихідний код

using System;

using System.Collections.Generic;

namespace laba4

{

class Program

{

// Returns the maximum value that can be put in a knapsack of capacity W

static int knapSack(int capacity, int[] weight, int[] val, int n)

{

// Base Case

if (n == 0 || capacity == 0)

return 0;

// If weight of the nth item is more than Knapsack capacity W, then

// this item cannot be included in the optimal solution

if (weight[n - 1] > capacity)

return knapSack(capacity, weight, val, n - 1);

// Return the maximum of two cases:

// (1) nth item included

// (2) not included

else return Math.Max(

val[n - 1] + knapSack(capacity - weight[n - 1], weight, val, n - 1),

knapSack(capacity, weight, val, n - 1)

);

}

public static void Main(string[] args)

{

FileReader FileReader = FileReader.getInstance();

FileReader.filename = "input.txt";

FileReader.ReadKnapsack();

List<Item> fullItemList = FileReader.ReadItems();

//int[] beesDistribution = { 70, 50, 30, 20, 15, 10, 8, 7, 6, 5, 4, 3, 2};

int[] beesDistribution = { 70, 50, 30, 20, 15, 8, 7, 6, 5, 4, 3, 2, 2, 2, 2, 1, 1, 1, 1 };

BeesColonyAlgorithm algo1 =

new BeesColonyAlgorithm(

parcel: new Knapsack(),

fullItemList: fullItemList,

maxIter: 500,

beesDistribution: beesDistribution,

sourcesToDump: 5,

ph: new KnapsackHeuristics());

Knapsack p1 = algo1.Run();

Console.WriteLine(p1);

}

static void ApproximateResult()

{

int[] val =

new int[] { 67, 52, 15, 14, 46, 56, 82, 62, 61, 154, 45,

145, 112, 27, 101, 53, 112, 175, 142, 41, 99, 143, 144,

33, 20, 20, 21, 8, 89, 127, 162, 160, 32, 113, 109, 38,

94, 77, 21, 23 };

int[] weight =

new int[] { 100, 46, 43, 21, 25, 81, 82, 63, 42, 16, 57, 46,

83, 31, 68, 92, 77, 51, 97, 45, 19, 90, 19, 29, 22, 31,

29, 18, 54, 36, 95, 41, 36, 51, 33, 27, 22, 52, 19, 91 };

int W = 400;

int n = val.Length;

Console.WriteLine(knapSack(W, weight, val, n));

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba4

{

public class KnapsackHeuristics

{

Random random;

public KnapsackHeuristics()

{

random = new Random();

}

public Knapsack GenerateRandomKnapsack(List<Item> fullItemList)

{

Knapsack newKnapsack = new Knapsack();

List<double> ItemsQuality = new List<double>();

List<int> shuffleList = new List<int>();

for (int i = 0; i < fullItemList.Count; i++)

{

shuffleList.Add(i);

}

var rnd = new Random();

shuffleList = shuffleList.OrderBy(n => rnd.Next()).ToList();

foreach (int x in shuffleList)

{

Item itemToPut = fullItemList.ElementAt(x);

if (newKnapsack.HasItem(itemToPut)) continue;

newKnapsack.AddItem(fullItemList.ElementAt(x));

}

return newKnapsack;

}

public List<Knapsack> GenerateMultipleRandomKnapsacks(List<Item> fullItemList, int numOfParcels)

{

List<Knapsack> result = new List<Knapsack>();

while (result.Count!= numOfParcels)

{

Knapsack newParcel = GenerateRandomKnapsack(fullItemList);

if (result.Contains(newParcel))

{

break;

}

result.Add(newParcel);

}

return result;

}

public List<Knapsack> GenerateRandomNeighbours(List<Item> fullItemList, Knapsack Knapsack, int numOfKnapsacks)

{

//rucksacks to return

List<Knapsack> Knapsacks = new List<Knapsack>();

int KnapsackListLength = Knapsack.itemList.Count;

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

{ // the amount of neighbours to generate

Knapsack newKnapsack = new Knapsack();

// return the close laying items

if (KnapsackListLength > 6)

{

for (int i = 0; i < 6; i++)

{

Item itemToPut = Knapsack.itemList.ElementAt(new Random().Next(KnapsackListLength));

if (newKnapsack.HasItem(itemToPut))

{

--i;

continue;

}

newKnapsack.AddItem(itemToPut); // write elements to new list

}

}

List<int> shuffleList = new List<int>();

for (int i = 0; i < fullItemList.Count; i++)

{

shuffleList.Add(i);

}

shuffleList = shuffleList.OrderBy(n => random.Next()).ToList();

foreach(int x in shuffleList)

{

Item itemToPut = fullItemList[x];

if (newKnapsack.HasItem(itemToPut)) continue;

newKnapsack.AddItem(fullItemList[x]);

//add random items to the list

}

// ПРОВЕРКА, ЧТО ПАКЕТЫ НЕ ДУБЛИРУЮТСЯ - УБИЙСТВА ВОКРУГ УМНЫХ

// ДЕЛАЙТЕ ЭТО ПОТОМУ ЧТО ЭТО ЯВНО РАСШИРЯЕТ ИТЕРАЦИИ

/\* for (Knapsack p : Knapsacks) {

if (p.equals(newKnapsack)) {

j--;

continue outer;

}

}

\*/

Knapsacks.Add(newKnapsack);

}

return Knapsacks;

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba4

{

public class FileReader

{

private static FileReader instance;

public static string filename = "input.txt";

public static FileReader getInstance()

{

if (instance == null)

{

instance = new FileReader();

}

return instance;

}

private FileReader() { }

public void ReadKnapsack()

{

StreamReader reader = null;

reader = new StreamReader(filename);

Console.WriteLine("Opened file with name " + filename);

try {

string sCurrentLine;

sCurrentLine = reader.ReadLine();

Knapsack.MAX\_WEIGHT = Convert.ToDouble(sCurrentLine.Split(" ")[0]);

} finally {

if (reader!= null) reader.Close();

}

}

public List<Item> ReadItems()

{

StreamReader br = null;

string filePath = filename;

try

{

List<Item> result = new List<Item>();

string sCurrentLine;

br = new StreamReader(filePath);

List<string> lines = new List<string>();

while ((sCurrentLine = br.ReadLine()) != null)

{

lines.Add(sCurrentLine);

}

for (int i = 1; i < lines.Count; i++)

{

string[] item = lines[i].Split(" ");

string name = item[0];

double weight = Convert.ToDouble(item[1]);

double price = Convert.ToDouble(item[2]);

result.Add(new QuantityItem(name, weight, price));

}

return result;

}

finally

{

if (br != null) br.Close();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba4

{

public class BeesColonyAlgorithm

{

public List<double> bestOfRun = new List<double>();

public List<double> meanOfRun = new List<double>();

public int iterWithBest = 0;

protected KnapsackHeuristics ph;

protected Knapsack originalParcel;

protected List<Item> fullItemList;

int maxIter;

int[] beesDistribution;

int sourcesToDump;

public BeesColonyAlgorithm(Knapsack parcel,

List<Item> fullItemList,

KnapsackHeuristics ph,

int maxIter,

int[] beesDistribution,

int sourcesToDump)

{

this.originalParcel = parcel;

this.fullItemList = fullItemList;

this.ph = ph;

this.maxIter = maxIter;

this.beesDistribution = beesDistribution;

this.sourcesToDump = sourcesToDump;

}

public Knapsack Run()

{

if (sourcesToDump > beesDistribution.Length) throw new Exception("MoreDumpedSourcesThanScouts");

bestOfRun.Clear();

meanOfRun.Clear();

int currentIter;

int numOfScouts = beesDistribution.Length;

List<Knapsack> currentSources = ph.GenerateMultipleRandomKnapsacks(fullItemList, numOfScouts); // generate different rucksacks

for (currentIter = 0; currentIter < maxIter; currentIter++)

{

Console.WriteLine("" + (currentIter + 1) + ": ");

currentSources.Sort(new ReverseKnapsackComparer());

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

{

currentSources[i] = SendBees(currentSources[i], beesDistribution[i]);

}

currentSources.Sort(new KnapsackComparer());

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

{

currentSources[i] = ph.GenerateRandomKnapsack(fullItemList);

}

currentSources.Sort(new ReverseKnapsackComparer());

foreach (var knapsack in currentSources)

{

Console.WriteLine("" + knapsack.CurrentQuality + " ");

}

Console.WriteLine();

bestOfRun.Add(currentSources[0].CurrentQuality);

double sum = 0;

long count = 0;

foreach(Knapsack currentSource in currentSources)

{

double currentQuality = currentSource.CurrentQuality;

sum += currentQuality;

count++;

}

meanOfRun.Add(count > 0 ? sum / (double)count : 0);

}

currentSources.Sort(new ReverseKnapsackComparer());

return currentSources[0];

}

private Knapsack SendBees(Knapsack source, int numOfBeesToSend)

{

List<Knapsack> neighbours = new List<Knapsack>(ph.GenerateRandomNeighbours(fullItemList, source, numOfBeesToSend));

bool seen = false;

Knapsack result = null;

foreach(Knapsack neighbour in neighbours)

{

if (!seen || neighbour.CompareTo(result) > 0)

{

seen = true;

result = neighbour;

}

}

Knapsack best = seen ? result : new Knapsack();

return best.CompareTo(source) > 0 ? best : source;

}

public bool shouldStop()

{

return false;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba4

{

public class QuantityItem : Item

{

public double Quality;

public QuantityItem(string name, double weight, double price)

:base(name, weight, price)

{

this.Quality = price / weight;

}

override public string ToString()

{

return "Item [name=" + Name + ", weight=" + Weight + ", price=" + Price + ", quality=" + Quality + " ]";

}

override public int GetHashCode()

{

return Name.GetHashCode() + Weight.GetHashCode() + Price.GetHashCode() + Quality.GetHashCode();

}

override public bool Equals(object obj)

{

if (!(obj is QuantityItem)) return false;

if (obj == this) return true;

QuantityItem castedObject = (QuantityItem)obj;

return castedObject.Name.Equals(Name) &&

castedObject.Weight == Weight &&

castedObject.Price == Price &&

castedObject.Quality == Quality;

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba4

{

public class Knapsack : IComparable

{

public static double MAX\_WEIGHT;

public double CurrentWeight = 0;

public double CurrentQuality = 0;

public ICollection<Item> itemList;

public Knapsack()

{

itemList = new List<Item>();

}

public Knapsack(ICollection<Item> itemList)

{

this.itemList = itemList;

}

public bool HasItem(Item item)

{

return itemList.Contains(item);

}

/\*\*

\* Adds an item to the Knapsack if it is possible.

\* If the item is already in the Knapsack, method will throw an exception, but will continue to run.

\* The weight of the item should be less or equal than remaining weight in the Knapsack.

\*/

public bool AddItem(Item item)

{

if (itemList.Contains(item)) throw new Exception();

if (CurrentWeight + item.Weight > MAX\_WEIGHT) return false;

itemList.Add(item);

CurrentWeight += item.Weight;

CurrentQuality += item.Price;

return true;

}

public bool RemoveItem(Item item)

{

if (itemList.Remove(item))

{

CurrentWeight -= item.Weight;

CurrentQuality -= item.Price;

return true;

}

return false;

}

public void RemoveItems()

{ //17.122

CurrentWeight = 0;

CurrentQuality = 0;

itemList.Clear();

}

override public string ToString()

{

var result = "Knapsack [MAX\_WEIGHT=" + MAX\_WEIGHT + ", CurrentWeight=" + CurrentWeight + ", CurrentQuality =" + CurrentQuality

+ ", itemList=";

foreach(var item in itemList)

{

result += item.ToString()+"\n";

}

return result += " ]";

}

override public bool Equals(object o)

{

if (this == o) return true;

if (o == null || this.GetType() != o.GetType()) return false;

Knapsack Knapsack = (Knapsack)o;

if (Knapsack.GetHashCode() != GetHashCode()) return false;

HashSet<Item> hs = new HashSet<Item>(Knapsack.itemList);

HashSet<Item> hs2 = new HashSet<Item>(itemList);

return hs.Equals(hs2);

}

override public int GetHashCode()

{

return itemList != null ? itemList.GetHashCode() : 0;

}

#nullable enable

public int CompareTo(object? o)

{

if (o is not Knapsack)

throw new ArgumentException("Passed non-knapsack object to CompareTo method of Knapsack");

return (int)(CurrentQuality - ((Knapsack)o).CurrentQuality);

}

#nullable disable

}

public class KnapsackComparer : IComparer<Knapsack>

{

public int Compare(Knapsack x, Knapsack y) =>

(int)(x.CurrentQuality - y.CurrentQuality);

}

public class ReverseKnapsackComparer : IComparer<Knapsack>

{

public int Compare(Knapsack x, Knapsack y) =>

(int)(y.CurrentQuality - x.CurrentQuality);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba4

{

public class Item

{

public string Name;

public double Weight;

public double Price;

public Item(string name, double weight, double price)

{

this.Name = name;

this.Weight = weight;

this.Price = price;

}

override public bool Equals(object obj)

{

if (!(obj is Item)) return false;

if (obj == this) return true;

Item castedObject = (Item)obj;

return castedObject.Name.Equals(Name) &&

castedObject.Weight == Weight &&

castedObject.Price == Price;

}

override public int GetHashCode()

{

return Name.GetHashCode() + Weight.GetHashCode() + Price.GetHashCode();

}

override public string ToString()

{

return "Item [name=" + Name + ", weight=" + Weight + ", price=" + Price + " ]";

}

}

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

40 бджіл

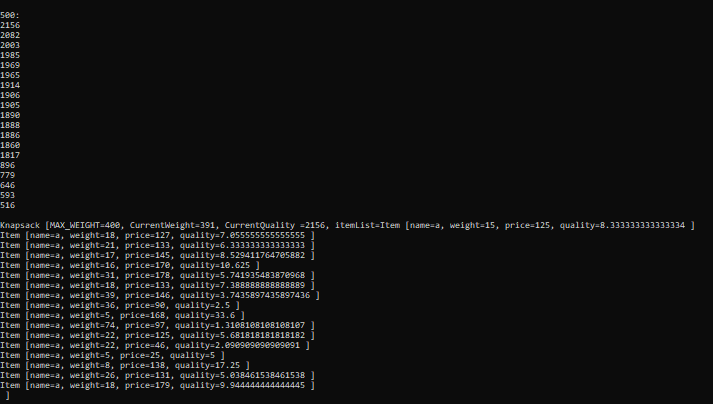


Рисунок 3.1 –

20 бджіл

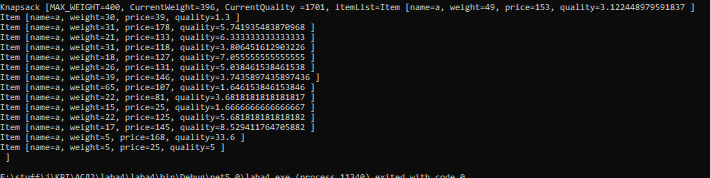


Рисунок 3.2 –

## Тестування алгоритму

|  |  |  |
| --- | --- | --- |
|  | Вага | Вартість |
| 20 бджіл | 1701 | 396 |
| 40 бджіл | 1800 | 398 |
| 60 бджіл | 1843 | 399 |

|  |  |  |
| --- | --- | --- |
|  | Вага | Вартість |
| 10 ділянок | 399 | 1813 |
| 20 ділянок | 399 | 1844 |
| 30 ділянок | 399 | 1837 |

Рисунок 3.3 –

Висновок

В рамках даної лабораторної роботі ми навчилися працювати с метаевристичними алгоритмами та використовувати їх для вирішення задач, знаходити оптимальні параметри. Дослідили стан та кінцеві результати алгоритмів за заданих умов

Github repo: <https://github.com/yan14171/PALAB4>

(при натисканні чомусь не працює, копіювати в браузер)

Критерії оцінювання

При здачі лабораторної роботи до 26.11.2021 включно максимальний бал дорівнює – 5. Після 26.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* покроковий алгоритм – 15%;
* програмна реалізація алгоритму – 50%;
* тестування алгоритму– 30%;
* висновок – 5%.