**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)

*Спаських Микола група ІТ04*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2020

Зміст

[1 Мета лабораторної роботи 3](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291748)

[2 Завдання 4](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291749)

[3 Виконання 10](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291750)

[3.1 Покроковий алгоритм 10](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291751)

[3.2 Програмна реалізація алгоритму 10](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291752)

[3.2.1 Вихідний код 10](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291753)

[3.2.2 Приклади роботи 10](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291754)

[3.3 Тестування алгоритму 11](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291755)

[Висновок 12](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_Toc52291756)

[Критерії оцінювання 13](file:///C:\Users\user\Documents\учебники%20и%20задания\2%20курс\теорія%20алгоритмів\лаби\lr4.ukr.docx#_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). |

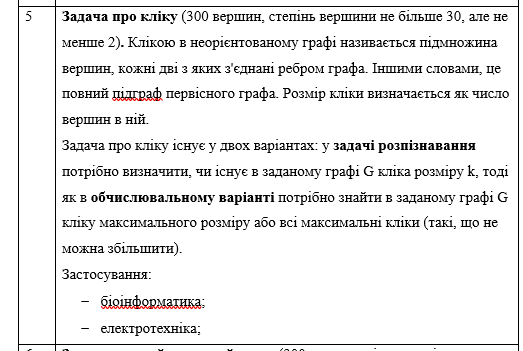
Таблиця 2.3 – Варіанти задач і алгоритмів

|  |  |
| --- | --- |
| **№** | **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

В мене варіант 18:





## 3. Покроковий алгоритм

## 

Тільки в мене там максимальне значення шукається в турнірній селекції і рекорд максимальний.

## Програмна реалізація алгоритму

### Вихідний код

using System;

using System.Collections.Generic;

using System.IO;

using System.Text.Json;

namespace lab4

{

class graf

{

public int n = 300; // kilkist vershin

public int m\_min = 2; //stepin vershini

public int m\_max = 30;

public int[,] mas; // matrisa sumijnosti

public int p = 10; // rozmir populatsii

public int p1; // pershiy predok

public int p2; // drugiy predok

public List<int> stepin\_uzlov;

public List <int[]> spisok\_hromosom;

public int[] kilkist\_vershin\_v\_hromosomah;

public List<int[]> shilds; // spisok nashadkiv

int[] bestshild;

int rekord;

public graf()

{

mas = new int[n, n];

stepin\_uzlov = new List<int>();

for (int i = 0; i < n; i++) stepin\_uzlov.Add(0);

//gen\_mas();

load();

spisok\_hromosom = new List<int[]>();

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

{

int[] mas = new int[n];

spisok\_hromosom.Add(mas);

}

//new int[p, n];

kilkist\_vershin\_v\_hromosomah = new int[p];

gen\_spisok\_hromosom();

shilds = new List<int[]>();

rekord = 1;

}

public void gen\_spisok\_hromosom()

{

Random ran = new Random();

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

{

for (int j = 0; j < n; j++) spisok\_hromosom[i][j] = 0;

spisok\_hromosom[i][ran.Next(n)] = 1;

kilkist\_vershin\_v\_hromosomah[i] = 1;

}

}

public void gen\_mas()

{

Random rnd = new Random();

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

{

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

{

mas[i, j] = 0;

}

int s1 = 0;

int s2 = 0;

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

if (mas[t, i] == 1) s1++;

for (int t = i + 1; t < n; t++) if (stepin\_uzlov[t]>= m\_max) s2++;

int k\_sv = m\_max - s1;

if (k\_sv > n - i - 1 - s2) k\_sv = n - i - 1 - s2;

if (k\_sv >= m\_min) k\_sv = rnd.Next(m\_min, k\_sv);

for (int k = 0; k < k\_sv; k++)

{

int r = rnd.Next(i + 1, n);

if (mas[i, r] == 0 && stepin\_uzlov[i] < m\_max && stepin\_uzlov[r] < m\_max)

{

mas[i, r] = 1;

stepin\_uzlov[i]++;

stepin\_uzlov[r]++;

}

else k--;

}

}

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

{

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

{

mas[j, i] = mas[i, j];

}

}

for (int i = 0; i < n; i++) // rahuemo stepeni uzliv

{

int s = 0;

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

{

if (mas[i, j] == 1) s++;

}

stepin\_uzlov[i] = s;

}

}

public void vivod()

{

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

{

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

{

Console.Write($"{ mas[i, j]}");

}

Console.WriteLine();

}

}

public void vivod\_hromosom()

{

Console.WriteLine("spisok hromosom:");

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

{

Console.Write($"hromosoma {i} mae {kilkist\_vershin\_v\_hromosomah[i]} vershin: ");

for (int j = 0; j < n; j++) Console.Write( spisok\_hromosom[i][j] + " ");

Console.WriteLine("\n");

}

}

public void vibir\_batkiv\_turnir(int k)

{

Random rnd = new Random();

List<int> vibrani = new List<int>(); //turnirniy spisok

int t;

for (int i=0; i<k; i++) // zapovniaem k kandidatami

{

t = rnd.Next(p);

if (vibrani.Contains(t)) i--;

else vibrani.Add(t);

}

//Console.WriteLine("spisok turniru ");

//for (int i = 0; i < k; i++) Console.Write(vibrani[i] + " ");

//Console.WriteLine();

int max1 = 0;

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

{

if (max1 < kilkist\_vershin\_v\_hromosomah[vibrani[i]]) max1 = vibrani[i];

}

int max2 = 0;

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

{

if (max2 < kilkist\_vershin\_v\_hromosomah[vibrani[i]] && vibrani[i]!=max1) max2 = vibrani[i];

}

p1 = max1;

p2 = max2;

}

public int f(int[] hromosoma)

{

int rez = 0;

for (int i = 0; i < hromosoma.Length; i++)

if (hromosoma[i] == 1) rez++;

return rez;

}

public bool chi\_e\_klika(int[] hromosoma)

{

bool rez = true;

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

{

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

{

if ((i != j) && (hromosoma[i] == 1) && hromosoma[j] == 1 && mas[i, j] == 0)

{

rez = false;

return rez;

}

}

}

return rez;

}

public void skreshuvania1()

{

int[] shild1 = new int[n];

int[] shild2 = new int[n];

int t = n / 2;

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

{

if (i < t)

{

shild1[i] = spisok\_hromosom[p1][i];

shild2[i] = spisok\_hromosom[p2][i];

}

else

{

shild1[i] = spisok\_hromosom[p2][i];

shild2[i] = spisok\_hromosom[p1][i];

}

}

if (chi\_e\_klika(shild1)) shilds.Add(shild1);

if (chi\_e\_klika(shild2)) shilds.Add(shild2);

shilds.Sort((x, y) => f(y) - f(x));

if(shilds.Count!=0) bestshild = shilds[0];

shilds.Clear();

}

public void skreshuvania2()

{

//int[] shild1 = new int[n];

int[] shild2 = new int[n];

int[] shild3 = new int[n];

int[] shild4 = new int[n];

int[] shild5 = new int[n];

int[] shild6 = new int[n];

int[] shild7 = new int[n];

//int[] shild8 = new int[n];

int t1 = n / 3;

int t2 = 2 \* t1;

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

{

if (i < t1)

{

//shild1[i] = spisok\_hromosom[p1, i];

shild2[i] = spisok\_hromosom[p1][i];

shild3[i] = spisok\_hromosom[p1][i];

shild4[i] = spisok\_hromosom[p1][i];

shild5[i] = spisok\_hromosom[p2][i];

shild6[i] = spisok\_hromosom[p2][i];

shild7[i] = spisok\_hromosom[p2][i];

//shild8[i] = spisok\_hromosom[p2, i];

}

else if (i<t2)

{

//shild1[i] = spisok\_hromosom[p1, i];

shild2[i] = spisok\_hromosom[p1][i];

shild3[i] = spisok\_hromosom[p2][i];

shild4[i] = spisok\_hromosom[p2][i];

shild5[i] = spisok\_hromosom[p1][i];

shild6[i] = spisok\_hromosom[p1][i];

shild7[i] = spisok\_hromosom[p2][i];

//shild8[i] = spisok\_hromosom[p2, i];

}

else

{

//shild1[i] = spisok\_hromosom[p1, i];

shild2[i] = spisok\_hromosom[p2][i];

shild3[i] = spisok\_hromosom[p1][i];

shild4[i] = spisok\_hromosom[p2][i];

shild5[i] = spisok\_hromosom[p1][i];

shild6[i] = spisok\_hromosom[p2][i];

shild7[i] = spisok\_hromosom[p1][i];

//shild8[i] = spisok\_hromosom[p2, i];

}

}

if (chi\_e\_klika(shild2)) shilds.Add(shild2);

if (chi\_e\_klika(shild3)) shilds.Add(shild3);

if (chi\_e\_klika(shild4)) shilds.Add(shild4);

if (chi\_e\_klika(shild5)) shilds.Add(shild5);

if (chi\_e\_klika(shild6)) shilds.Add(shild6);

if (chi\_e\_klika(shild7)) shilds.Add(shild7);

if (shilds.Count != 0)

{

shilds.Sort((x, y) => f(y) - f(x));

bestshild = shilds[0];

}

else bestshild = null;

shilds.Clear();

}

public void skreshuvania3()

{

//int[] shild1 = new int[n];

int[] shild2 = new int[n];

int[] shild3 = new int[n];

int[] shild4 = new int[n];

int[] shild5 = new int[n];

int[] shild6 = new int[n];

int[] shild7 = new int[n];

int[] shild8 = new int[n];

int[] shild9 = new int[n];

int[] shild10 = new int[n];

int[] shild11 = new int[n];

int[] shild12 = new int[n];

int[] shild13 = new int[n];

int[] shild14 = new int[n];

int[] shild15 = new int[n];

//int[] shild16 = new int[n];

int t1 = n / 4;

int t2 = 2 \* t1;

int t3 = 3 \* t1;

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

{

if (i < t1)

{

//shild1[i] = spisok\_hromosom[p1, i];

shild2[i] = spisok\_hromosom[p1][i];

shild3[i] = spisok\_hromosom[p1][i];

shild4[i] = spisok\_hromosom[p1][i];

shild5[i] = spisok\_hromosom[p1][i];

shild6[i] = spisok\_hromosom[p1][i];

shild7[i] = spisok\_hromosom[p1][i];

shild8[i] = spisok\_hromosom[p1][i];

shild9[i] = spisok\_hromosom[p2][i];

shild10[i] = spisok\_hromosom[p2][i];

shild11[i] = spisok\_hromosom[p2][i];

shild12[i] = spisok\_hromosom[p2][i];

shild13[i] = spisok\_hromosom[p2][i];

shild14[i] = spisok\_hromosom[p2][i];

shild15[i] = spisok\_hromosom[p2][i];

//shild16[i] = spisok\_hromosom[p2, i];

}

else if (i < t2)

{

//shild1[i] = spisok\_hromosom[p1, i];

shild2[i] = spisok\_hromosom[p1][i];

shild3[i] = spisok\_hromosom[p1][i];

shild4[i] = spisok\_hromosom[p1][i];

shild5[i] = spisok\_hromosom[p2][i];

shild6[i] = spisok\_hromosom[p2][i];

shild7[i] = spisok\_hromosom[p2][i];

shild8[i] = spisok\_hromosom[p2][i];

shild9[i] = spisok\_hromosom[p1][i];

shild10[i] = spisok\_hromosom[p1][i];

shild11[i] = spisok\_hromosom[p1][i];

shild12[i] = spisok\_hromosom[p1][i];

shild13[i] = spisok\_hromosom[p2][i];

shild14[i] = spisok\_hromosom[p2][i];

shild15[i] = spisok\_hromosom[p2][i];

//shild16[i] = spisok\_hromosom[p2, i];

}

else if (i < t3)

{

//shild1[i] = spisok\_hromosom[p1, i];

shild2[i] = spisok\_hromosom[p1][i];

shild3[i] = spisok\_hromosom[p2][i];

shild4[i] = spisok\_hromosom[p2][i];

shild5[i] = spisok\_hromosom[p1][i];

shild6[i] = spisok\_hromosom[p1][i];

shild7[i] = spisok\_hromosom[p2][i];

shild8[i] = spisok\_hromosom[p2][i];

shild9[i] = spisok\_hromosom[p1][i];

shild10[i] = spisok\_hromosom[p1][i];

shild11[i] = spisok\_hromosom[p2][i];

shild12[i] = spisok\_hromosom[p2][i];

shild13[i] = spisok\_hromosom[p1][i];

shild14[i] = spisok\_hromosom[p1][i];

shild15[i] = spisok\_hromosom[p2][i];

//shild16[i] = spisok\_hromosom[p2, i];

}

else

{

//shild1[i] = spisok\_hromosom[p1, i];

shild2[i] = spisok\_hromosom[p2][i];

shild3[i] = spisok\_hromosom[p1][i];

shild4[i] = spisok\_hromosom[p2][i];

shild5[i] = spisok\_hromosom[p1][i];

shild6[i] = spisok\_hromosom[p2][i];

shild7[i] = spisok\_hromosom[p1][i];

shild8[i] = spisok\_hromosom[p2][i];

shild9[i] = spisok\_hromosom[p1][i];

shild10[i] = spisok\_hromosom[p2][i];

shild11[i] = spisok\_hromosom[p1][i];

shild12[i] = spisok\_hromosom[p2][i];

shild13[i] = spisok\_hromosom[p1][i];

shild14[i] = spisok\_hromosom[p2][i];

shild15[i] = spisok\_hromosom[p1][i];

//shild16[i] = spisok\_hromosom[p2, i];

}

}

if (chi\_e\_klika(shild2)) shilds.Add(shild2);

if (chi\_e\_klika(shild3)) shilds.Add(shild3);

if (chi\_e\_klika(shild4)) shilds.Add(shild4);

if (chi\_e\_klika(shild5)) shilds.Add(shild5);

if (chi\_e\_klika(shild6)) shilds.Add(shild6);

if (chi\_e\_klika(shild7)) shilds.Add(shild7);

if (chi\_e\_klika(shild8)) shilds.Add(shild8);

if (chi\_e\_klika(shild9)) shilds.Add(shild9);

if (chi\_e\_klika(shild10)) shilds.Add(shild10);

if (chi\_e\_klika(shild11)) shilds.Add(shild11);

if (chi\_e\_klika(shild12)) shilds.Add(shild12);

if (chi\_e\_klika(shild13)) shilds.Add(shild13);

if (chi\_e\_klika(shild14)) shilds.Add(shild14);

if (chi\_e\_klika(shild15)) shilds.Add(shild15);

if (shilds.Count != 0)

{

shilds.Sort((x, y) => f(y) - f(x));

bestshild = shilds[0];

}

else bestshild = null;

shilds.Clear();

}

public void vivid\_nashadkiv()

{

if (shilds.Count!=0) Console.WriteLine("nashadki: ");

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

{

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

{

Console.Write(shilds[i][j] + " ");

}

Console.WriteLine("\n");

}

Console.WriteLine($"bestshild: {rekord} vuzliv: ");

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

{

Console.Write(bestshild[j] + " ");

}

}

public void mutasia(int p) // p - vidsotok mutantiv

{

if (bestshild != null)

for (int t = 0; t < 20; t++)

{

Random ran = new Random();

if (ran.Next(101) <= p)

{

int i = ran.Next(n);

int[] shild1 = new int[n];

for (int j = 0; j < n; j++) shild1[j] = bestshild[j];

shild1[i] = ran.Next(2);

if (chi\_e\_klika(shild1)) bestshild = shild1;

}

}

//{

// int[] shild1 = new int[n];

// for (int j = 0; j < n; j++) shild1[j] = bestshild[j];

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

// {

// if (shild1[j] == 1)

// {

// shild1[j] = 0;

// bestshild = shild1;

// return;

// }

// }

//}

}

public void lokalne\_pokrashenia(int k\_pokrashen)

{

if (bestshild != null)

{

for (int t=0; t<k\_pokrashen; t++)

{

int[] shild1 = new int[n];

for (int i = 0; i < n; i++) shild1[i] = bestshild[i];

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

{

if (shild1[i] == 0)

{

shild1[i] = 1;

if (chi\_e\_klika(shild1))

{

bestshild = shild1;

break;

}

else shild1[i] = 0;

}

}

}

//if (k\_pokrashen==1)

//{

// int[] shild1 = new int[n];

// for (int i = 0; i < n; i++) shild1[i] = bestshild[i];

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

// {

// if (shild1[i] == 0)

// {

// shild1[i] = 1;

// if (chi\_e\_klika(shild1))

// {

// bestshild = shild1;

// return;

// }

// else shild1[i] = 0;

// }

// }

//}

//else if (k\_pokrashen==2)

//{

// int[] shild1 = new int[n];

// for (int i = 0; i < n; i++) shild1[i] = bestshild[i];

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

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

// {

// if (shild1[i] == 0 && shild1[j] == 0 && i != j)

// {

// shild1[i] = 1;

// shild1[j] = 1;

// if (chi\_e\_klika(shild1))

// {

// bestshild = shild1;

// return;

// }

// else

// {

// shild1[i] = 0;

// shild1[j] = 0;

// }

// }

// }

//}

}

}

public void zmina\_rekordu()

{

if (rekord < f(bestshild)) rekord = f(bestshild);

}

public void dodavania\_nashadka\_do\_populatsii()

{

spisok\_hromosom.Add(bestshild);

spisok\_hromosom.Sort((x, y) => f(y) - f(x));

//Console.WriteLine($"vidaleno hromosomu z {f(spisok\_hromosom[spisok\_hromosom.Count - 1])} uzlami");

spisok\_hromosom.RemoveAt(spisok\_hromosom.Count - 1);

}

public void evolutsia(int k)

{

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

{

vibir\_batkiv\_turnir(5);

skreshuvania1();

mutasia(80);

lokalne\_pokrashenia(3);

zmina\_rekordu();

dodavania\_nashadka\_do\_populatsii();

}

}

public void save()

{

string fname = "data.txt";

using (StreamWriter sw = new StreamWriter(fname, false, System.Text.Encoding.Default))

{

string s = "";

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

{

for (int j = 0; j < n; j++) s += mas[i, j] + " ";

}

sw.WriteLine(s);

//sw.WriteLine(JsonSerializer.Serialize<int[,]>(mas));

}

}

public void load()

{

string fname = "data.txt";

string s = "";

using (StreamReader sr = new StreamReader(fname))

{

int k = 0;

s = sr.ReadToEnd();

string[] dataS = s.Split(" ");

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

{

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

{

mas[i, j] = Int32.Parse(dataS[k]);

k++;

}

}

//s = sr.ReadToEnd();

//mas = JsonSerializer.Deserialize<int[,]>(s);

}

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

graf g = new graf();

g.vivod();

//g.save();

//g.load();

//Console.WriteLine("load.save");

//g.vivod();

g.evolutsia(300);

g.vivid\_nashadkiv();

}

}

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 –

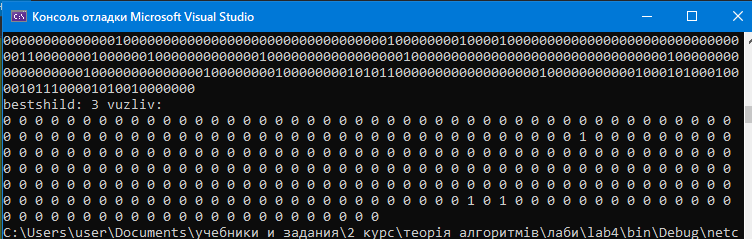
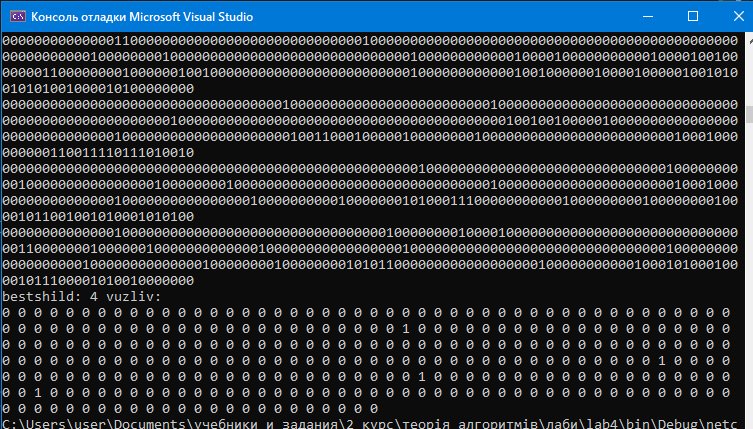
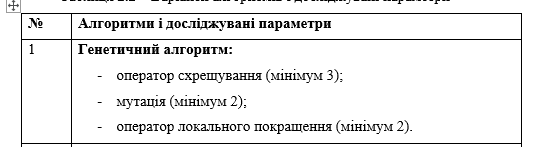
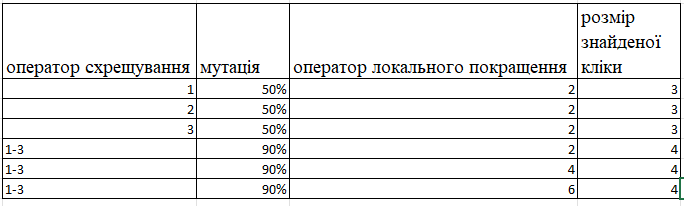


Рисунок 3.2 –



## Тестування алгоритму на графі з 300 вершинами і 2-30 степеннювершини.





Від оператора схрещування результат практично не залежав, схоже там був локальний максимум 3, як збільшив відсоток мутації знайшло краще рішення з максимумом 4.

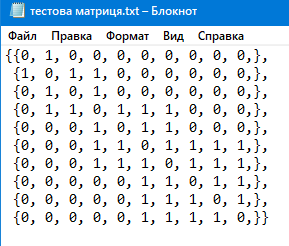
За порадою в чаті, створив невеликий граф з 4 локальними максимумами(2,3,4,5) і побачив, що алгоритм дійсно працює ефективно і зміг оптимально підібрати параметри. (найкраще рішення найшвидше знаходиться триточковим оператором схрещування з ймовірністю мутації 90% при оператору локального покращення 5 (хоча від нього майже не залежало)). Якщо задати малий відсоток мутації, то пошук не виходить за межі локального максимуму. 

Рисунок 3.3 –

Висновок

В рамках даної лабораторної роботи номер 4 я дослідив генетичний алгоритм в задачі про пошук максимальної кліки. І підібрав параметри, щоб алгоритм працював оптимально і знаходив не лише локальний максимум. Викрутить мутацію на максимум і отримаємо не локальні розв’язки.

Ссилка на гідхаб з лабами: https://github.com/kolya-sp/alg\_2