**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 № 2 з дисципліни

«Проектування алгоритмів»

**«Неінформативний, інформативний та локальний пошук»**

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-14 Радзівіло Валерія Артемівна*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc81070688)

[2 ЗаВдання 4](#_Toc81070689)

[3 Виконання 7](#_Toc81070690)

[3.1 Псевдокод алгоритмів 7](#_Toc81070691)

[3.2 Програмна реалізація 9](#_Toc81070692)

[3.2.1 Вихідний код 9](#_Toc81070693)

[3.2.2 Приклади роботи 39](#_Toc81070694)

[3.3 Дослідження алгоритмів 40](#_Toc81070695)

[Висновок 45](#_Toc81070696)

[Критерії оцінювання 46](#_Toc81070697)

# Мета лабораторної роботи

Мета роботи – розглянути та дослідити алгоритми неінформативного, інформативного та локального пошуку. Провести порівняльний аналіз ефективності використання алгоритмів.

# ЗаВдання

Записати алгоритм розв’язання задачі у вигляді псевдокоду, відповідно до варіанту (таблиця 2.1).

Реалізувати програму, яка розв’язує поставлену задачу згідно варіанту (таблиця 2.1) за допомогою алгоритму неінформативного пошуку **АНП**, алгоритму інформативного пошуку **АІП,** що використовує задану евристичну функцію Func, або алгоритму локального пошуку **АЛП та бектрекінгу,** що використовує задану евристичну функцію Func.

Програму реалізувати на довільній мові програмування.

**Увага!** Алгоритм неінформативного пошуку **АНП,** реалізовується за принципом «AS IS», тобто так, як є, без додаткових модифікацій (таких як перевірка циклів, наприклад).

Провести серію експериментів для вивчення ефективності роботи алгоритмів. Кожний експеримент повинен відрізнятись початковим станом. Серія повинна містити не менше 20 експериментів для кожного алгоритму. Початковий стан зафіксувати у таблиці експериментів. За проведеними серіями необхідно визначити:

* середню кількість етапів (кроків), які знадобилось для досягнення розв’язку (ітерації);
* середню кількість випадків, коли алгоритм потрапляв в глухий кут (не міг знайти оптимальний розв’язок) – якщо таке можливе;
* середню кількість згенерованих станів під час пошуку;
* середню кількість станів, що зберігаються в пам’яті під час роботи програми.

Передбачити можливість обмеження виконання програми за часом (30 хвилин) та використання пам’яті (1 Гб).

**Використані позначення:**

* **8-ферзів** – Задача про вісім ферзів полягає в такому розміщенні восьми ферзів на шахівниці, що жодна з них не ставить під удар один одного. Тобто, вони не повинні стояти в одній вертикалі, горизонталі чи діагоналі.
* **8-puzzle** – гра, що складається з 8 однакових квадратних пластинок з нанесеними числами від 1 до 8. Пластинки поміщаються в квадратну коробку, довжина сторони якої в три рази більша довжини сторони пластинок, відповідно в коробці залишається незаповненим одне квадратне поле. Мета гри – переміщаючи пластинки по коробці досягти впорядковування їх по номерах, бажано зробивши якомога менше переміщень.
* **Лабіринт** – задача пошуку шляху у довільному лабіринті від початкової точки до кінцевої з можливими випадками відсутності шляху. Структура лабіринту зчитується з файлу, або генерується програмою.
* **LDFS** – Пошук вглиб з обмеженням глибини.
* **BFS** – Пошук вшир.
* **IDS** – Пошук вглиб з ітеративним заглибленням.
* **A\*** – Пошук А\*.
* **RBFS** – Рекурсивний пошук за першим найкращим співпадінням.
* **F1** – кількість пар ферзів, які б’ють один одного з урахуванням видимості (ферзь А може стояти на одній лінії з ферзем В, проте між ними стоїть ферзь С; тому А не б’є В).
* **F2** – кількість пар ферзів, які б’ють один одного без урахування видимості.
* **H1** – кількість фішок, які не стоять на своїх місцях.
* **H2** – Манхетенська відстань.
* **H3** – Евклідова відстань.
* **COLOR** – Задача розфарбування карти самостійно обраної країни, не менше 20 регіонів (областей). Необхідно розфарбувати карту не більше ніж у 4 різні кольори. Мається на увазі приписування кожному регіону власного кольору так, щоб кольори сусідніх регіонів відрізнялись. Використовувати евристичну функцію, яка повертає кількість пар суміжних вузлів, що мають однаковий колір (тобто кількість конфліктів). Реалізувати алгоритм пошуку із поверненнями (backtracking) для розв’язання поставленої задачі. Для підвищення швидкодії роботи алгоритму використати евристичну функцію, а початковим станом вважати випадкову вершину.
* **HILL –** Пошук зі сходженням на вершину з використанням із використанням руху вбік (на 100 кроків) та випадковим перезапуском (кількість необхідних разів запуску визначити самостійно).
* **ANNEAL** – Локальний пошук із симуляцією відпалу. Робоча характеристика – залежність температури Т від часу роботи алгоритму t. Можна розглядати лінійну залежність: T = 1000 - k∙t, де k – змінний коефіцієнт.
* **BEAM** – Локальний променевий пошук. Робоча характеристика – кількість променів k. Експерименти проводи із кількістю променів від 2 до 21.
* **MRV** – евристика мінімальної кількості значень;
* **DGR** – ступенева евристика.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Задача** | **АНП** | **АІП** | **АЛП** | **Func** |
| 14 | 8-ферзів | BFS | RBFS |  | F2 |

# Виконання

## Псевдокод алгоритмів

1. BFS

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

gr.addEdge(0, initials.get(k));

// fill graph with initial values

// if the first value is 7 0 or 1, place 2 (there are no graphs with these starting values

if (initials.get(0)==7||initials.get(0)==0||initials.get(0)==1)

gr.get\_adj(0).removeFirst();

gr.get\_adj(0).addFirst(2);

// go through graph duplicating values if some options are possible

graph\_count = go\_through\_graph(gr, n, queen\_number, graph\_count, should\_place, 11,possible\_position,true );

// check if we found the answer

if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11)

System.out.println("CONGRATS");

gr.get\_adj(graph\_count).add(9);

return;

// analyzed graphs that are not completely filled with values

analyze\_graph(gr,graph\_count,n,initial\_task);

if the value is found

BFS(int s)//s – key value for easier search

{

boolean[] visited = new boolean[V]; // mark all searched routes as visited

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

if(ArrWork.check\_in\_arr(adj[i],s)) // find if the key value is in array

return adj[i];

else

visited[i] = true;

return null;

}

if (result != null)

print result as board

1. RBFS

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

gr.addEdge(0, initials.get(k));

// fill graph with initial values

// if the first value is 7 0 or 1, place 2 (there are no graphs with these starting values

if (initials.get(0)==7||initials.get(0)==0||initials.get(0)==1)

gr.get\_adj(0).removeFirst();

gr.get\_adj(0).addFirst(2);

// go through graph duplicating values if some options are possible

graph\_count = go\_through\_graph(gr, n, queen\_number, graph\_count, should\_place, 11,possible\_position,true );

// check if we found the answer

if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11)

System.out.println("CONGRATS");

gr.get\_adj(graph\_count).add(9);

return;

// analyzed graphs that are not completely filled with values

analyze\_graph(gr,graph\_count,n,initial\_task);

if the value is found

RBFS(int s)//s – key value for easier search

{

int answer = 0;

if(adj[i].size()!=n+1 && !ArrWork.check\_in\_arr(adj[i],s)) // find if the key value is in array

visited.add(i, true);

i++;

adj[answer] = RBFS(s,visited,i,n); // recursion

else

return adj[i];

return adj[answer];

}

if (result != null)

print result as board

1. F2

int pairs=0;

//check diagonals

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

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

if(j!=i && Math.abs(array[i]-array[j])==Math.abs(i-j))

pairs++;

//check verticals

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

int tmp =0;

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

if(array[i]==array[j]&& i!=j)

tmp++;

if (tmp>1)

pairs+=tmp;

if(tmp==1)

pairs++;

## Програмна реалізація

### Вихідний код

Main.java:

import ArrWork.\*;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.*assertArrayEquals*;  
  
public class Main {  
 public static void main(String args[]) {  
 int n =8;  
 int [] not\_rand\_arr = new int[]{7, 5, 5, 2, 5, 1, 5, 3};  
 int [] result = make\_n\_queens.*make\_n\_queens*(n,true ,not\_rand\_arr );  
  
 System.*out*.println("Result: ");  
 ArrWork.*create\_matr*(result,n);  
 }  
  
  
 @Test  
 void testValueFromMethod()  
 {  
 int n=8;  
 int []expected = {3, 7, 0, 2, 5, 1, 6, 4};  
  
 int []result = make\_n\_queens.*make\_n\_queens*(n,false, new int[]{3,6,2,3,0,5,6,7});  
 *assertArrayEquals*(expected,result ,"Test Check");  
 System.*out*.println("Expected: ");  
 ArrWork.*create\_matr*(expected,n);  
 System.*out*.println("Result: ");  
 ArrWork.*create\_matr*(result,n);  
 }  
  
 @Test  
 void ifBoardCorrect()  
 {  
 int n=8;  
 int []expected = {2,5,7,0,3,6,4,1};  
  
 int []result = make\_n\_queens.*make\_n\_queens*(n,false, new int[]{2,5,7,0,3,6,4,1});  
 *assertArrayEquals*(expected,result ,"Test Check");  
 System.*out*.println("Expected: ");  
 ArrWork.*create\_matr*(expected,n);  
 System.*out*.println("Result: ");  
 ArrWork.*create\_matr*(result,n);  
 }  
  
}

make\_n\_queens.java:

package ArrWork;  
  
public class make\_n\_queens {  
 public static int[] make\_n\_queens(int n, boolean isRand, int[] notrand)  
 {  
 int [ ] initial\_placement = new int[n];  
 if(isRand)  
 initial\_placement = ArrWork.*make\_rand\_int\_arr*(n);  
 else  
 initial\_placement =notrand.clone();  
  
 System.*out*.println("Initial task: ");  
 ArrWork.*create\_matr*(initial\_placement,n);  
 int k =0;  
 if(n<8)  
 k =n;  
 else  
 k=n-2;  
 Graph gr = new Graph((int) Math.*pow*(n,k));  
 int pairs = F2.*F2*(initial\_placement,n);  
 System.*out*.println("Pairs F2: ");  
 System.*out*.println(pairs);  
 Searches.*fill\_graph*(gr,n,initial\_placement);  
 int[] result = Searches.*find\_result*(gr,n);  
 pairs = F2.*F2*(result,n);  
 System.*out*.println("Pairs F2 after replacement: ");  
 System.*out*.println(pairs);  
 return result;  
 }  
}

Searches.java:

package ArrWork;  
  
import java.util.\*;  
  
public class Searches {  
  
 public static int fill\_possible\_options(ArrayList<Integer> pos, int n, int column, int[] already\_placed, int queen\_number) {  
  
 ArrayList<Integer>pos\_before\_check = new ArrayList<>();  
 for (int k = 0; k < n; k++) {  
 if (k != column && k + 1 != column && k - 1 != column && !(ArrWork.*check\_in\_int\_arr*(already\_placed, k))) {  
 pos\_before\_check.add(k);  
  
 }  
 }  
 for (Integer integer : pos\_before\_check) {  
 if (ArrWork.*place*(queen\_number, integer, already\_placed))  
 pos.add(integer);  
 }  
 return pos.size();  
  
  
 }  
  
 public static void fill\_graph(Graph gr, int n, int[] initial\_task) {  
 int[] solution = new int[n];  
 List<Integer> initials = ArrWork.*array\_to\_list*(initial\_task);  
 int queen\_number = 0;  
 int[] already\_placed = new int[n];  
 int graph\_count = 0;  
 int counter\_check = 0;  
 System.*out*.println("Add initial: ");  
 for (int k = 0; k < n; k++) {  
 gr.addEdge(0, initials.get(k));  
 }  
 gr.print\_graph();  
 graph\_count++;  
 if (initials.get(0)==7||initials.get(0)==0||initials.get(0)==1) {  
 gr.get\_adj(0).removeFirst();  
 gr.get\_adj(0).addFirst(2);  
 }  
  
  
  
 // Stage 1  
  
 LinkedList<Integer> clone = gr.get\_adj(0);  
  
 System.*out*.println("Stage 1: ");  
 if (gr.get\_adj(0).get(0) == n - 1 || gr.get\_adj(0).get(0) == 0) {  
 gr.addEdge(1, 1);  
 for (int i = 1; i < n; i++)  
 gr.addEdge(1, gr.get\_adj(0).get(i));  
 graph\_count++;  
 }  
  
 int[] clone\_arr = ArrWork.*linked\_to\_array*(clone);  
 int[] clone\_arr\_copy = clone\_arr.clone();  
 int[] should\_place = clone\_arr.clone();  
 counter\_check = 1;  
 should\_place = clone\_arr.clone();  
 clone\_arr = clone\_arr\_copy.clone();  
 queen\_number = 0;  
 int possible\_amount = 0;  
 int possible\_position =0;  
 graph\_count = *go\_through\_graph*(gr, n, queen\_number, graph\_count, should\_place, 11,possible\_position,true );  
  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 gr.get\_adj(graph\_count).add(9);  
 return;  
 }  
  
  
 System.*out*.println("ANALYZING: ");  
 *analyze\_graph*(gr,graph\_count,n,initial\_task);  
  
  
  
 }  
  
 private static int go\_through\_graph(Graph gr, int n, int queen\_number, int graph\_count, int[] clone\_arr,  
 int forbidden, int current\_possibles\_pos, boolean expand\_with\_possibles) {  
 int[] already\_placed;  
 int pos\_amount = 0;  
 already\_placed = new int[n];  
 int additional\_graphs = 0;  
 for (int i = 0; i < n; i++) {  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count));  
 pos\_amount = *insert\_in\_graph*(gr, clone\_arr[i], queen\_number, already\_placed, graph\_count, n, forbidden, current\_possibles\_pos);  
 queen\_number++;  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count));  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 gr.get\_adj(graph\_count).add(9);  
 i = n;  
 return graph\_count+additional\_graphs;  
 }  
  
  
 int additional\_pos\_amount =0;  
 if(pos\_amount>1&& gr.get\_adj(graph\_count).getLast() != 11&& expand\_with\_possibles) {  
  
  
 int[] new\_clone = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count));  
  
 if (gr.get\_adj(graph\_count).getLast() != 11) {  
 for (int k = 1; k < pos\_amount; k++) {  
 int queen\_number\_new = 0;  
 additional\_graphs++;  
  
 for (int j = 0; j < new\_clone.length-1; j++) {  
 gr.addEdge(graph\_count+additional\_graphs,new\_clone[j]);  
 queen\_number\_new++;  
 }  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count+additional\_graphs));  
 *insert\_in\_graph*(gr, new\_clone[new\_clone.length-1], queen\_number\_new, already\_placed, graph\_count + additional\_graphs, n, new\_clone[new\_clone.length-1], k);  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 gr.get\_adj(graph\_count).add(9);  
 k=pos\_amount;  
 return graph\_count+additional\_graphs;  
 }  
 ArrayList<Integer>add\_pos = new ArrayList<>();  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count+additional\_graphs));  
 additional\_pos\_amount = *fill\_possible\_options*(add\_pos,n,already\_placed[already\_placed.length-1],already\_placed,queen\_number\_new);  
  
 gr.print\_graph();  
  
 if(additional\_pos\_amount>0) {  
 int[] new\_add\_clone = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count+additional\_graphs));  
 for (int z = 1; z < additional\_pos\_amount; z++) {  
 int queen\_add\_number\_new = 0;  
 additional\_graphs++;  
  
 for (int j = 0; j < new\_add\_clone.length - 1; j++) {  
 gr.addEdge(graph\_count + additional\_graphs, new\_add\_clone[j]);  
 queen\_add\_number\_new++;  
 }  
 gr.addEdge(graph\_count+additional\_graphs,new\_add\_clone[new\_add\_clone.length-1]);  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count + additional\_graphs));  
 *insert\_in\_graph*(gr, new\_add\_clone[new\_add\_clone.length - 1], queen\_add\_number\_new, already\_placed, graph\_count + additional\_graphs, n, new\_add\_clone[new\_add\_clone.length - 1], z);  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 gr.get\_adj(graph\_count).add(9);  
 z = additional\_pos\_amount;  
 k=pos\_amount;  
 return graph\_count+additional\_graphs;  
 }  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count + additional\_graphs));  
  
 gr.print\_graph();  
 }  
 }  
  
 }  
 }  
 }  
  
 if (gr.get\_adj(graph\_count).getLast() == 11)  
 i = n;  
  
  
 }  
  
 gr.print\_graph();  
  
 return graph\_count+additional\_graphs;  
 }  
 public static boolean check\_adj(Graph gr, int position)  
 {  
 int queen\_number = 0;  
 int check = 0;  
 int[] already\_placed = new int[gr.get\_adj(position).size()];  
 for(int i =0; i< gr.get\_adj(position).size();i++)  
 {  
 if(ArrWork.*place*(queen\_number,gr.get\_adj(position).get(i),already\_placed))  
 check++;  
 queen\_number++;  
  
 already\_placed[i] = gr.get\_adj(position).get(i);  
 }  
 if(check==gr.get\_adj(position).size())  
 return true;  
 return false;  
 }  
  
 private static void analyze\_graph(Graph gr, int graph\_count, int n, int[]initial\_task)  
 {  
  
 for(int i =2; i<graph\_count;i++) {  
  
 if(gr.get\_adj(i).size()<n && *check\_adj*(gr,i)) {  
 int queen\_number = gr.get\_adj(i).size();  
 int []clone = ArrWork.*linked\_to\_array*(gr.get\_adj(i));  
 for(int j =gr.get\_adj(i).size()-1; j <n-1;j++) {  
 clone = ArrWork.*addtoArr*(1,clone);  
 }  
  
 int possible\_amounts = *go\_through\_graph*(gr,n,queen\_number,i,clone,11,0,false);  
 if (possible\_amounts == 0)  
 gr.addEdge(i, 11);  
 }  
  
  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(i).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 gr.get\_adj(i).add(9);  
 i = graph\_count;  
 return;  
 }  
  
 if (gr.BFS(9)!=null)  
 return;  
 }  
 }  
  
  
  
  
  
 static int insert\_in\_graph(Graph gr, int insert\_value, int queen\_number, int[] already\_placed,  
 int variant\_start, int n, int forbidden, int possible\_cur\_position) {  
 ArrayList<Integer> possible\_options = new ArrayList<>();  
 int possibles\_amount=0;  
 if(already\_placed.length>0)  
 possibles\_amount= *fill\_possible\_options*(possible\_options, n, already\_placed[already\_placed.length - 1], already\_placed, queen\_number);;  
 if (ArrWork.*place*(queen\_number, insert\_value, already\_placed) && insert\_value!=forbidden) {  
 gr.addEdge(variant\_start, insert\_value);  
  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(variant\_start));  
 } else {  
 possibles\_amount = *fill\_possible\_options*(possible\_options, n, already\_placed[already\_placed.length - 1], already\_placed, queen\_number);  
  
 if (possible\_options.size() > 0) {  
 if (ArrWork.*place*(queen\_number, possible\_options.get(possible\_cur\_position), already\_placed)) {  
 gr.addEdge(variant\_start, possible\_options.get(possible\_cur\_position));  
 } else {  
 while (possible\_options.size() > 0 && !(ArrWork.*place*(queen\_number, possible\_options.get(possible\_cur\_position), already\_placed))) {  
 possible\_options.remove(possible\_cur\_position);  
 if (possible\_options.size() > 0) {  
 if (ArrWork.*place*(queen\_number, possible\_options.get(possible\_cur\_position), already\_placed)) {  
 gr.addEdge(variant\_start, possible\_options.get(possible\_cur\_position));  
 break;  
 }  
 }  
  
 }  
 }  
 if (possible\_options.size() == 0)  
 gr.addEdge(variant\_start, 11);  
  
  
 }  
 if (possible\_options.size() == 0)  
 gr.addEdge(variant\_start, 11);  
  
 }  
 return possibles\_amount;  
 }  
  
 public static int[] find\_result(Graph gr, int n) {  
 LinkedList<Integer> solution = gr.BFS(9);  
 if (solution != null) {  
 Object[] result = solution.toArray();  
 int[] result\_to\_print = new int[n];  
 for (int i = 0; i < n; i++)  
 result\_to\_print[i] = (int) result[i];  
 return result\_to\_print;  
 }  
 return null;  
 }  
  
  
}

ArrWork.java:

package ArrWork;  
  
import java.util.\*;  
import java.util.stream.IntStream;  
  
public class ArrWork {  
  
 public static String[][] create\_matr(int[] arr, int n)  
 {  
 String[][] matr = new String[n][n];  
 for(int i =0; i <n;i++)  
 {  
 for (int j = 0; j <n; j++)  
 {  
 if(arr[i]==j)  
 System.*out*.print(" Q ");  
 else{  
 System.*out*.print(" # ");  
 }  
  
 }  
 System.*out*.println();  
 }  
 return matr;  
 }  
  
  
 public static boolean place(int queen\_number, int column\_number, int[] columns\_placed)  
 {  
 for(int j =0; j<queen\_number;j++)  
 {  
 if(columns\_placed[j]==column\_number || Math.*abs*(columns\_placed[j]-column\_number)==Math.*abs*(j-queen\_number))  
  
 {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 public static boolean check\_in\_arr(LinkedList<Integer> arr, int toCheckValue)  
 {  
 return arr.contains(toCheckValue);  
 }  
  
  
 public static boolean check\_in\_int\_arr(int[] arr, int toCheckValue)  
 {  
 boolean test  
 = IntStream.*of*(arr)  
 .anyMatch(x -> x == toCheckValue);  
 return test;  
 }  
 public static int[] linked\_to\_array(LinkedList<Integer> solution)  
 {  
 Object[] result = solution.toArray();  
 int[] result\_to\_print = new int[solution.size()];  
 for (int i = 0; i < solution.size(); i++)  
 result\_to\_print[i] = (int) result[i];  
  
 return result\_to\_print;  
 }  
  
 public static List<Integer> array\_to\_list(int []arr)  
 {  
 List<Integer> answer = new ArrayList<>();  
 for (int j : arr) {  
 answer.add(j);  
 }  
  
 return answer;  
 }  
  
  
  
 public static int[] make\_rand\_int\_arr (int n)  
 {  
 Random random = new Random();  
 int[]result = new int[n];  
 for(int i = 0; i <n; i++)  
 result[i]=random.nextInt(n);  
 return result;  
 }  
  
 public static int[] addtoArr(int amount\_to\_add, int arr[])  
 {  
 int[] new\_arr = new int[arr.length+amount\_to\_add];  
 for(int i =0; i< new\_arr.length; i++)  
 {  
 if(i<arr.length)  
 new\_arr[i]=arr[i];  
 else  
 new\_arr[i] = 1;  
 }  
 return new\_arr;  
 }  
  
  
}

Graph.java:

package ArrWork;  
  
import java.util.LinkedList;  
  
public class Graph {  
 private int V; // No. of vertices  
 private LinkedList<Integer> adj[]; //Adjacency Lists  
  
 public int graph\_size() {  
 int size\_gr=0;  
  
 for (int i = 0; i < V; i++) {  
 if (!adj[i].isEmpty())  
 size\_gr=i;  
 else  
 i=V;  
 }  
 return size\_gr;  
 }  
  
  
 public LinkedList<Integer> get\_adj(int V)  
 {  
 return adj[V];  
 }  
  
  
 public void print\_graph()  
 {  
 for(int i =0; i< V;i++) {  
 if(!adj[i].isEmpty())  
 System.*out*.print(adj[i]);  
 else  
 i = V;  
 }  
 System.*out*.println();  
 }  
  
 // Constructor  
 public Graph(int v) {  
 V = v;  
 adj = new LinkedList[v];  
 for (int i = 0; i < v; ++i)  
 adj[i] = new LinkedList();  
 }  
  
 // Function to add an edge into the graph  
 void addEdge(int v, int w) {  
 adj[v].add(w);  
 }  
  
 // prints BFS traversal from a given source s  
 LinkedList<Integer> BFS(int s) {  
 boolean[] visited = new boolean[V];  
 for(int i =0; i< V;i++)  
 {  
 if(ArrWork.*check\_in\_arr*(adj[i],s))  
 {  
 return adj[i];  
 }  
 else  
 visited[i] = true;  
 }  
 return null;  
 }  
}

2. RBFS:

Main.java:

import ArrWork.\*;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.*assertArrayEquals*;  
  
public class Main {  
 public static void main(String args[]) {  
 int n =8;  
 int [] not\_rand\_arr = new int[]{7, 5, 0, 6, 6, 3, 7, 6};  
 int [] result = make\_n\_queens.*make\_n\_queens*(n,true ,not\_rand\_arr );  
  
 System.*out*.println("Result: ");  
 ArrWork.*create\_matr*(result,n);  
  
 }  
  
  
 @Test  
 void testValueFromMethod()  
 {  
 int n=8;  
 int []expected = {3, 7, 0, 2, 5, 1, 6, 4};  
  
 int []result = make\_n\_queens.*make\_n\_queens*(n,false, new int[]{3,6,2,3,0,5,6,7});  
 *assertArrayEquals*(expected,result ,"Test Check");  
 System.*out*.println("Expected: ");  
 ArrWork.*create\_matr*(expected,n);  
 System.*out*.println("Result: ");  
 ArrWork.*create\_matr*(result,n);  
 }  
  
 @Test  
 void ifBoardCorrect()  
 {  
 int n=8;  
 int []expected = {2,5,7,0,3,6,4,1};  
  
 int []result = make\_n\_queens.*make\_n\_queens*(n,false, new int[]{2,5,7,0,3,6,4,1});  
 *assertArrayEquals*(expected,result ,"Test Check");  
 System.*out*.println("Expected: ");  
 ArrWork.*create\_matr*(expected,n);  
 System.*out*.println("Result: ");  
 ArrWork.*create\_matr*(result,n);  
 }  
  
}

make\_n\_queens.java:

package ArrWork;  
  
public class make\_n\_queens {  
 public static int[] make\_n\_queens(int n, boolean isRand, int[] notrand)  
 {  
 int [ ] initial\_placement = new int[n];  
 if(isRand)  
 initial\_placement = ArrWork.*make\_rand\_int\_arr*(n);  
 else  
 initial\_placement =notrand.clone();  
  
 System.*out*.println("Initial task: ");  
 ArrWork.*create\_matr*(initial\_placement,n);  
 int k =0;  
 if(n<8)  
 k =n;  
 else  
 k=n-2;  
 Graph gr = new Graph((int) Math.*pow*(n,k));  
 int pairs = F2.*F2*(initial\_placement,n);  
 System.*out*.println("Pairs F2 before replacement: ");  
 System.*out*.println(pairs);  
 Searches.*fill\_graph*(gr,n,initial\_placement);  
 int[] result = Searches.*find\_result*(gr,n);  
 pairs = F2.*F2*(result,n);  
 System.*out*.println("Pairs F2 after replacement: ");  
 System.*out*.println(pairs);  
 return result;  
 }  
}

ArrWork.java:

package ArrWork;  
  
import java.util.\*;  
import java.util.stream.IntStream;  
  
public class ArrWork {  
  
 public static String[][] create\_matr(int[] arr, int n)  
 {  
 String[][] matr = new String[n][n];  
 for(int i =0; i <n;i++)  
 {  
 for (int j = 0; j <n; j++)  
 {  
 if(arr[i]==j)  
 System.*out*.print(" Q ");  
 else{  
 System.*out*.print(" # ");  
 }  
  
 }  
 System.*out*.println();  
 }  
 return matr;  
 }  
  
  
 public static boolean place(int queen\_number, int column\_number, int[] columns\_placed)  
 {  
 for(int j =0; j<queen\_number;j++)  
 {  
 if(columns\_placed[j]==column\_number || Math.*abs*(columns\_placed[j]-column\_number)==Math.*abs*(j-queen\_number))  
  
 {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 public static boolean check\_in\_arr(LinkedList<Integer> arr, int toCheckValue)  
 {  
 return arr.contains(toCheckValue);  
 }  
  
  
 public static boolean check\_in\_int\_arr(int[] arr, int toCheckValue)  
 {  
 boolean test  
 = IntStream.*of*(arr)  
 .anyMatch(x -> x == toCheckValue);  
 return test;  
 }  
 public static int[] linked\_to\_array(LinkedList<Integer> solution)  
 {  
 Object[] result = solution.toArray();  
 int[] result\_to\_print = new int[solution.size()];  
 for (int i = 0; i < solution.size(); i++)  
 result\_to\_print[i] = (int) result[i];  
  
 return result\_to\_print;  
 }  
  
 public static List<Integer> array\_to\_list(int []arr)  
 {  
 List<Integer> answer = new ArrayList<>();  
 for (int j : arr) {  
 answer.add(j);  
 }  
  
 return answer;  
 }  
  
  
 public static int[] make\_rand\_int\_arr (int n)  
 {  
 Random random = new Random();  
 int[]result = new int[n];  
 for(int i = 0; i <n; i++)  
 result[i]=random.nextInt(n);  
 return result;  
 }  
  
 public static int[] addtoArr(int amount\_to\_add, int arr[])  
 {  
 int[] new\_arr = new int[arr.length+amount\_to\_add];  
 for(int i =0; i< new\_arr.length; i++)  
 {  
 if(i<arr.length)  
 new\_arr[i]=arr[i];  
 else  
 new\_arr[i] = 1;  
 }  
 return new\_arr;  
 }  
  
}

Graph.java:

package ArrWork;  
  
import java.util.\*;  
import java.util.LinkedList;  
  
public class Graph {  
 private int V; // No. of vertices  
 private LinkedList<Integer> adj[]; //Adjacency Lists  
  
 public int graph\_size() {  
 int size\_gr=0;  
  
 for (int i = 0; i < V; i++) {  
 if (!adj[i].isEmpty())  
 size\_gr=i;  
 else  
 i=V;  
 }  
 return size\_gr;  
 }  
  
  
  
 public LinkedList<Integer> get\_adj(int V)  
 {  
 return adj[V];  
 }  
  
  
 public void print\_graph()  
 {  
 for(int i =0; i< V;i++) {  
 if(!adj[i].isEmpty())  
 System.*out*.print(adj[i]);  
 else  
 i = V;  
 }  
 System.*out*.println();  
 }  
  
 // Constructor  
 public Graph(int v) {  
 V = v;  
 adj = new LinkedList[v];  
 for (int i = 0; i < v; ++i)  
 adj[i] = new LinkedList();  
 }  
  
 // Function to add an edge into the graph  
 void addEdge(int v, int w) {  
 adj[v].add(w);  
 }  
  
 // prints BFS traversal from a given source s  
 LinkedList<Integer> RBFS(int s, ArrayList<Boolean>visited, int i, int n) {  
 int answer = 0;  
 if(adj[i].size()!=n+1 && !ArrWork.*check\_in\_arr*(adj[i],s))  
 {  
 visited.add(i, true);  
 i++;  
 adj[answer] = RBFS(s,visited,i,n);  
 }  
 else {  
 return adj[i];  
 }  
  
 return adj[answer];  
 }  
}

Searches.java:

package ArrWork;  
  
import java.util.\*;  
  
public class Searches {  
  
 public static int fill\_possible\_options(ArrayList<Integer> pos, int n, int column, int[] already\_placed, int queen\_number) {  
 pos.clear();  
 ArrayList<Integer>pos\_before\_check = new ArrayList<>();  
 for (int k = 0; k < n; k++) {  
 if (k != column && k + 1 != column && k - 1 != column && !(ArrWork.*check\_in\_int\_arr*(already\_placed, k))) {  
 pos\_before\_check.add(k);  
  
 }  
 }  
 for (Integer integer : pos\_before\_check) {  
 if (ArrWork.*place*(queen\_number, integer, already\_placed))  
 pos.add(integer);  
 }  
 return pos.size();  
  
  
 }  
  
 public static void fill\_graph(Graph gr, int n, int[] initial\_task) {  
 int[] solution = new int[n];  
 List<Integer> initials = ArrWork.*array\_to\_list*(initial\_task);  
 int queen\_number = 0;  
 int[] already\_placed = new int[n];  
 int graph\_count = 0;  
 int counter\_check = 0;  
 System.*out*.println("Add initial: ");  
 for (int k = 0; k < n; k++) {  
 gr.addEdge(0, initials.get(k));  
 }  
 gr.print\_graph();  
 graph\_count++;  
 if (initials.get(0)==7||initials.get(0)==0||initials.get(0)==1) {  
 gr.get\_adj(0).removeFirst();  
 gr.get\_adj(0).addFirst(2);  
 }  
  
  
  
 // Stage 1  
  
 LinkedList<Integer> clone = gr.get\_adj(0);  
  
 System.*out*.println("Stage 1: ");  
 if (gr.get\_adj(0).get(0) == n - 1 || gr.get\_adj(0).get(0) == 0) {  
 gr.addEdge(1, 1);  
 for (int i = 1; i < n; i++)  
 gr.addEdge(1, gr.get\_adj(0).get(i));  
 graph\_count++;  
 }  
  
 int[] clone\_arr = ArrWork.*linked\_to\_array*(clone);  
 int[] clone\_arr\_copy = clone\_arr.clone();  
 int[] should\_place = clone\_arr.clone();  
 counter\_check = 1;  
 should\_place = clone\_arr.clone();  
 clone\_arr = clone\_arr\_copy.clone();  
 queen\_number = 0;  
 int possible\_amount = 0;  
 int possible\_position =0;  
 graph\_count = *go\_through\_graph*(gr, n, queen\_number, graph\_count, should\_place, 11,possible\_position,true );  
  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 gr.get\_adj(graph\_count).add(9);  
  
 System.*out*.println(gr.get\_adj(graph\_count));  
 return;  
 }  
  
  
 System.*out*.println("ANALYZING: ");  
 *analyze\_graph*(gr,graph\_count,n,initial\_task);  
  
  
  
 }  
  
 private static int go\_through\_graph(Graph gr, int n, int queen\_number, int graph\_count, int[] clone\_arr,  
 int forbidden, int current\_possibles\_pos, boolean expand\_with\_possibles) {  
 int[] already\_placed;  
 int pos\_amount = 0;  
 already\_placed = new int[n];  
 int additional\_graphs = 0;  
 for (int i = 0; i < n; i++) {  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count));  
 pos\_amount = *insert\_in\_graph*(gr, clone\_arr[i], queen\_number, already\_placed, graph\_count, n, forbidden, current\_possibles\_pos);  
 queen\_number++;  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count));  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 System.*out*.println(gr.get\_adj(graph\_count));  
 gr.get\_adj(graph\_count).add(9);  
 i = n;  
 return 101;  
 }  
  
  
 int additional\_pos\_amount =0;  
 if(pos\_amount>1&& gr.get\_adj(graph\_count).getLast() != 11&& expand\_with\_possibles) {  
  
  
 int[] new\_clone = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count));  
  
 if (gr.get\_adj(graph\_count).getLast() != 11) {  
 for (int k = 1; k < pos\_amount; k++) {  
 int queen\_number\_new = 0;  
 additional\_graphs++;  
  
 for (int j = 0; j < new\_clone.length-1; j++) {  
 gr.addEdge(graph\_count+additional\_graphs,new\_clone[j]);  
 queen\_number\_new++;  
 }  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count+additional\_graphs));  
 *insert\_in\_graph*(gr, new\_clone[new\_clone.length-1], queen\_number\_new, already\_placed, graph\_count + additional\_graphs, n, new\_clone[new\_clone.length-1], k);  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 System.*out*.println(gr.get\_adj(graph\_count));  
 gr.get\_adj(graph\_count).add(9);  
 k=pos\_amount;  
 return 101;  
 }  
 ArrayList<Integer>add\_pos = new ArrayList<>();  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count+additional\_graphs));  
 additional\_pos\_amount = *fill\_possible\_options*(add\_pos,n,already\_placed[already\_placed.length-1],already\_placed,queen\_number\_new);  
  
 gr.print\_graph();  
  
 if(additional\_pos\_amount>0) {  
 int[] new\_add\_clone = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count+additional\_graphs));  
 for (int z = 1; z < additional\_pos\_amount; z++) {  
 int queen\_add\_number\_new = 0;  
 additional\_graphs++;  
  
 for (int j = 0; j < new\_add\_clone.length - 1; j++) {  
 gr.addEdge(graph\_count + additional\_graphs, new\_add\_clone[j]);  
 queen\_add\_number\_new++;  
 }  
 gr.addEdge(graph\_count+additional\_graphs,new\_add\_clone[new\_add\_clone.length-1]);  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count + additional\_graphs));  
 *insert\_in\_graph*(gr, new\_add\_clone[new\_add\_clone.length - 1], queen\_add\_number\_new, already\_placed, graph\_count + additional\_graphs, n, new\_add\_clone[new\_add\_clone.length - 1], z);  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(graph\_count).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 gr.get\_adj(graph\_count).add(9);  
 System.*out*.println(gr.get\_adj(graph\_count));  
 z = additional\_pos\_amount;  
 k=pos\_amount;  
 return 101;  
 }  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(graph\_count + additional\_graphs));  
  
 gr.print\_graph();  
 }  
 }  
  
 }  
 }  
 }  
  
 if (gr.get\_adj(graph\_count).getLast() == 11)  
 i = n;  
  
  
 }  
  
 gr.print\_graph();  
  
 return graph\_count+additional\_graphs;  
 }  
  
 public static boolean check\_adj(Graph gr, int position)  
 {  
 int queen\_number = 0;  
 int check = 0;  
 int[] already\_placed = new int[gr.get\_adj(position).size()];  
 for(int i =0; i< gr.get\_adj(position).size();i++)  
 {  
 if(ArrWork.*place*(queen\_number,gr.get\_adj(position).get(i),already\_placed))  
 check++;  
 queen\_number++;  
  
 already\_placed[i] = gr.get\_adj(position).get(i);  
 }  
 if(check==gr.get\_adj(position).size())  
 return true;  
 return false;  
 }  
  
 private static void analyze\_graph(Graph gr, int graph\_count, int n, int[]initial\_task)  
 {  
  
 for(int i =2; i<graph\_count+1;i++) {  
  
 if (gr.get\_adj(i).size() < n && *check\_adj*(gr, i)) {  
 int queen\_number = gr.get\_adj(i).size();  
 int[] clone = ArrWork.*linked\_to\_array*(gr.get\_adj(i));  
 for (int j = gr.get\_adj(i).size() - 1; j < n - 1; j++) {  
 clone = ArrWork.*addtoArr*(1, clone);  
 }  
  
 int possible\_amounts = *go\_through\_graph*(gr, n, queen\_number, i, clone, 11, 0, false);  
 if (possible\_amounts == 101)  
 i = graph\_count;  
 }  
 else{  
 gr.get\_adj(i).add(11);  
 }  
  
  
 if (gr.get\_adj(graph\_count).size() > n - 1 && gr.get\_adj(i).getLast() != 11) {  
 System.*out*.println("CONGRATS");  
 gr.get\_adj(i).add(9);  
 System.*out*.println(gr.get\_adj(graph\_count));  
  
 i = graph\_count;  
 return;  
 }  
  
  
 }  
 }  
  
  
  
  
  
 static int insert\_in\_graph(Graph gr, int insert\_value, int queen\_number, int[] already\_placed,  
 int variant\_start, int n, int forbidden, int possible\_cur\_position) {  
 ArrayList<Integer> possible\_options = new ArrayList<>();  
 int possibles\_amount=0;  
 if(already\_placed.length>0)  
 possibles\_amount= *fill\_possible\_options*(possible\_options, n, already\_placed[already\_placed.length - 1], already\_placed, queen\_number);;  
 if (ArrWork.*place*(queen\_number, insert\_value, already\_placed) && insert\_value!=forbidden) {  
 gr.addEdge(variant\_start, insert\_value);  
  
 already\_placed = ArrWork.*linked\_to\_array*(gr.get\_adj(variant\_start));  
 } else {  
 possibles\_amount = *fill\_possible\_options*(possible\_options, n, already\_placed[already\_placed.length - 1], already\_placed, queen\_number);  
  
 if (possible\_options.size() > 0) {  
 if (ArrWork.*place*(queen\_number, possible\_options.get(possible\_cur\_position), already\_placed)) {  
 gr.addEdge(variant\_start, possible\_options.get(possible\_cur\_position));  
 } else {  
 while (possible\_options.size() > 0 && !(ArrWork.*place*(queen\_number, possible\_options.get(possible\_cur\_position), already\_placed))) {  
 possible\_options.remove(possible\_cur\_position);  
 if (possible\_options.size() > 0) {  
 if (ArrWork.*place*(queen\_number, possible\_options.get(possible\_cur\_position), already\_placed)) {  
 gr.addEdge(variant\_start, possible\_options.get(possible\_cur\_position));  
 break;  
 }  
 }  
  
 }  
 }  
 if (possible\_options.size() == 0)  
 gr.addEdge(variant\_start, 11);  
  
  
 }  
 if (possible\_options.size() == 0)  
 gr.addEdge(variant\_start, 11);  
  
 }  
 return possibles\_amount;  
 }  
  
 public static int[] find\_result(Graph gr, int n) {  
 ArrayList<Boolean>visited = new ArrayList<>();  
 int j =0;  
 LinkedList<Integer> solution = gr.RBFS(9, visited, j,n);  
 if (solution != null) {  
 Object[] result = solution.toArray();  
 int[] result\_to\_print = new int[n];  
 for (int i = 0; i < n; i++)  
 result\_to\_print[i] = (int) result[i];  
 return result\_to\_print;  
 }  
 return null;  
 }  
  
  
}

1. F2.java:

package ArrWork;  
  
public class F2 {  
 public static int F2(int[]array, int n)  
 {  
 int pairs=0;  
 for (int i=0; i<n;i++)  
 {  
 for(int j =i+1; j<n;j++)  
 if(j!=i && Math.*abs*(array[i]-array[j])==Math.*abs*(i-j))  
 pairs++;  
  
 }  
  
 for(int i =0; i<n;i++)  
 {  
 int tmp =0;  
 for(int j =i+1; j<n; j++)  
 if(array[i]==array[j]&& i!=j)  
 tmp++;  
 if (tmp>1){  
 pairs+=tmp;  
 }  
 if(tmp==1)  
 pairs++;  
 }  
  
  
 return pairs;  
 }  
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для різних алгоритмів пошуку.

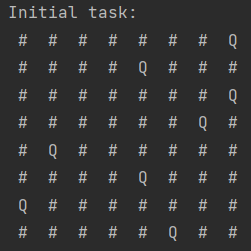
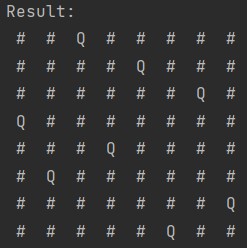
 

Рисунок 3.1 – Алгоритм BFS

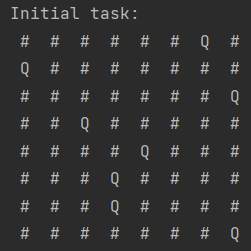
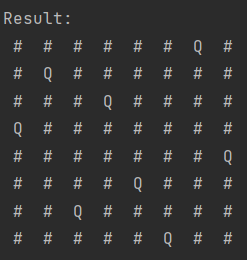
 

Рисунок 3.2 – Алгоритм RBFS

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму BFS, задачі 8 queens для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму BFS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пам’яті |
|  | 2 |  | 28 | 28 |
|  | 4 |  | 48 | 48 |
|  | 2 |  | 30 | 30 |
|  | 3 |  | 38 | 38 |
|  | 2 |  | 39 | 39 |
|  | 16 |  | 40 | 40 |
|  | 7 |  | 39 | 39 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму RBFS, задачі 8 queens для 20 початкових станів.

Таблиця 3.3 – Характеристики оцінювання алгоритму RBFS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пам’яті |
|  | 8 |  | 21 | 21 |
|  | 11 |  | 22 | 22 |
|  | 3 |  | 26 | 26 |
|  | 3 |  | 22 | 22 |
|  | 19 |  | 21 | 21 |
|  | 3 |  | 23 | 23 |
|  | 4 |  | 18 | 18 |

Висновок

* При виконанні даної лабораторної роботи було розглянуто та досліджено алгоритми неінформативного, інформативного та локального пошуку. В часності BFS та RBFS для логічної задачі 8 ферзів. Задача полягає у розташуванні 8 ферзів на полі так, щоб ні один не бив іншого. Проведено порівняльний аналіз ефективності використання алгоритмів. Обидва алгоритми показали правильні варіанти розв’язання задачі, перевірені як вручну, так і використовуючи функцію F2. F2 – кількість пар ферзів, які б’ють один одного без урахування видимості. Усі алгоритми указані в звіті.

Критерії оцінювання

За умови здачі лабораторної роботи до 23.10.2022 включно максимальний бал дорівнює – 5. Після 23.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 10%;
* програмна реалізація алгоритму – 60%;
* дослідження алгоритмів – 25%;
* висновок – 5%.