

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)

ІП-13 Жмайло Д. О.
(шифр, прізвище, ім'я, по батькові)

Перевірив

Сопов О. О.
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ.....	8
3.1	ПСЕВДОКОД АЛГОРИТМІВ.....	8
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ.....	10
3.2.1	<i>Вихідний код.....</i>	<i>10</i>
3.2.2	<i>Приклади роботи</i>	<i>21</i>
3.3	ДОСЛІДЖЕННЯ АЛГОРИТМІВ	23
	ВИСНОВОК	25
	КРИТЕРІЇ ОЦІНЮВАННЯ	26

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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 **АПІ**, 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 відрізнятися 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 впорядкування 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 A*.
- **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 T від часу роботи алгоритму t . Можна розглядати лінійну залежність: $T = 1000 - k \cdot t$, 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	АП	АЛП	Func
1	Лабіринт	LDFS	A*		H2
2	Лабіринт	LDFS	RBFS		H3
3	Лабіринт	BFS	A*		H2
4	Лабіринт	BFS	RBFS		H3
5	Лабіринт	IDS	A*		H2
6	Лабіринт	IDS	RBFS		H3
7	8-ферзів	LDFS	A*		F1
8	8-ферзів	LDFS	A*		F2
9	8-ферзів	LDFS	RBFS		F1
10	8-ферзів	LDFS	RBFS		F2
11	8-ферзів	BFS	A*		F1
12	8-ферзів	BFS	A*		F2
13	8-ферзів	BFS	RBFS		F1
14	8-ферзів	BFS	RBFS		F2
15	8-ферзів	IDS	A*		F1

16	8-ферзів	IDS	A*		F2
17	8-ферзів	IDS	RBFS		F1
18	Лабіринт	LDFS	A*		H3
19	8-puzzle	LDFS	A*		H1
20	8-puzzle	LDFS	A*		H2
21	8-puzzle	LDFS	RBFS		H1
22	8-puzzle	LDFS	RBFS		H2
23	8-puzzle	BFS	A*		H1
24	8-puzzle	BFS	A*		H2
25	8-puzzle	BFS	RBFS		H1
26	8-puzzle	BFS	RBFS		H2
27	Лабіринт	BFS	A*		H3
28	8-puzzle	IDS	A*		H2
29	8-puzzle	IDS	RBFS		H1
30	8-puzzle	IDS	RBFS		H2
31	COLOR			HILL	MRV
32	COLOR			ANNEAL	MRV
33	COLOR			BEAM	MRV
34	COLOR			HILL	DGR
35	COLOR			ANNEAL	DGR
36	COLOR			BEAM	DGR

3 ВИКОНАННЯ

3.1 Псевдокод алгоритмів

АЛГОРИТМ LDFS

ФУНКЦІЯ DepthLimitedSearch (node, limit)

 ПОВЕРНУТИ RecursiveDLS (node, limit)

КІНЕЦЬ ФУНКЦІЇ

ФУНКЦІЯ RecursiveDLS (node, limit)

 ІНІАЛІЗУВАТИ bool cutoffOccured зі значенням false

 ЯКЩО CountConflicts у node дорівнює 0 ТО

 ПОВЕРНУТИ node

 ВСЕ ЯКЩО

 ІНАКШЕ ЯКЩО node дорівнює limit ТО

 ПОВЕРНУТИ node := null cutOff := true

 КІНЕЦЬ ІНАКШЕ ЯКЩО

 Expand node.successors

 ДЛЯ ВСІХ successor В node.successors

 result := RecursiveDLS (successor, limit)

 ЯКЩО result.GetCutOff() := true ТО

 cutoffOccured := true

 КІНЕЦЬ ЯКЩО

 ІНАКШЕ ЯКЩО result != null && result.GetNode() != null ТО

 ПОВЕРНУТИ result

 КІНЕЦЬ ІНАКШЕ

 КІНЕЦЬ ДЛЯ

 ЯКЩО cutoffOccured := true ТО

 ПОВЕРНУТИ node := null cutOff := true

 КІНЕЦЬ ЯКЩО

 ПОВЕРНУТИ node := null isFailed := true

КІНЕЦЬ ФУНКЦІЇ

АЛГОРИТМ RBFS

ФУНКЦІЯ RecursiveBestFirstSearch (node, limit)

ІНІЦІАЛІЗУВАТИ depth := 0

ПОВЕРНУТИ RecursiveBFS (node, limit, depth)

КІНЕЦЬ ФУНКЦІЇ

ФУНКЦІЯ RecursiveBFS(node, limit, depth)

ІНІЦІАЛІЗУВАТИ MAXDEPTH := 16 * (node.state.GetSize() *
node.state.GetSize() - 1);

ЯКЩО CountConflicts у node дорівнює 0 ТО

ПОВЕРНУТИ node

ВСЕ ЯКЩО

ІНАКШЕ ЯКЩО depth > MAXDEPTH ТО

ПОВЕРНУТИ node := null cutOff := true

КІНЕЦЬ ІНАКШЕ ЯКЩО

Expand node.successors

ЯКЩО Count у node.successors дорівнює 0 ТО

ПОВЕРНУТИ node := null isFailed := true

ВСЕ ЯКЩО

ІНІЦІАЛІЗУВАТИ цілочисельний список f

ДЛЯ ВСІХ successor В node.successors

ДОДАТИ в список нове f := successor.state.CountConflicts()

КІНЕЦЬ ДЛЯ

ПОВТОРИТИ

```

ІНІЦІАЛІЗУВАТИ змінну bestValue := f.Min()
ІНІЦІАЛІЗУВАТИ змінну bestIndex = f.IndexOf(bestValue)
ІНІЦІАЛІЗУВАТИ змінну bestNode = node.successors[bestI]
ЯКЩО bestValue > limitТО
    ПОВЕРНУТИ node := null cutOff := true
ВСЕ ЯКЩО
    f.Remove(bestValue);
    node.successors.Remove(bestNode)
    ІНІЦІАЛІЗУВАТИ змінну alternative = f.Min()
    limit = Math.Min(limit, alternative);
    ПОВЕРНУТИ RecursiveBFS (bestNode, limit, depth + 1)

ЯКЩО result != null && result.GetNode() != null ТО
    ПОВЕРНУТИ result
КІНЕЦЬ ЯКЩО
КІНЕЦЬ ПОВТОРИТИ
ПОВЕРНУТИ node := null isFailed := true
КІНЕЦЬ ФУНКЦІЇ

```

3.2 Програмна реалізація

3.2.1 Вихідний код

Chessboard.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2PA
{
    internal class ChessBoard
    {
        private byte[] table;
        private byte size;
    }
}

```

```

public ChessBoard(string message, int minLimit, int maxLimit)
{
    size = InputByte(message, minLimit, maxLimit);
    table = InputQueens();
}
public ChessBoard(byte parentSize, byte[] parentTable)
{
    size = parentSize;
    table = new byte[(int)size];
    Array.Copy(parentTable, table, (int)size);
}
public byte[] GetArray() { return table; }
public byte GetSize() { return size; }
private byte[] InputQueens()
{
    byte[] table = new byte[size];
    Random random = new Random();
    for (int i = 0; i < size; i++)
        table[i] = Convert.ToByte(random.Next(0, size));
    return table;
}

public byte GetQueenCol (int row){
    return table[row];
}
public void MoveQueen(int row, byte newCol)
{
    if (newCol < 0 || newCol > size-1 || row < 0 || row > size-1)
    {
        Console.WriteLine("Wrong position.");
        return;
    }
    table[row] = newCol;
}

public int CountConflicts()
{
    int count = 0;
    count += VerticalConflictsCount();

    for (int col = 0; col < size - 1; col++)
    {
        count += MainDiagonalCount(0, col);
        count += SecondaryDiagonalCount(size - 1, col);
    }

    for (int row = 1; row < size - 1; row++)
    {
        count += MainDiagonalCount(row, 0);
        count += SecondaryDiagonalCount(row, 0);
    }

    return count;
}

// Vertical conflicts; horizontal conflicts == 0
private int VerticalConflictsCount()
{
    int count = 0;
    for (int i = 0; i < size - 1; i++)
    {
        int colCount = 0;
        byte temp = table[i];
        for (int j = i + 1; j < size; j++)
        {

```

```

        if (temp == table[j])
            colCount++;
    }
    if (colCount > 1)
        colCount--;
    count += colCount;
}
return count;
}

private int MainDiagonalCount(int row, int col)
{
    int count = 0;

    for (int i = row; i < size; i++)
    {
        if (table[i] == col)
            count++;
        col++;
    }
    if (count > 0)
        count--;
    return count;
}

private int SecondaryDiagonalCount(int row, int col)
{
    int count = 0;

    for (int i = row; i >= 0; i--)
    {
        if (table[i] == col)
            count++;
        col++;
    }
    if (count > 0)
        count--;
    return count;
}

private static byte InputByte(string message, int minLimit, int maxLimit)
{
    Console.WriteLine($"Enter {message} ({minLimit} <= N <= {maxLimit}): ");
    while (true)
    {
        string input = Console.ReadLine();
        if (byte.TryParse(input, out byte number) && number >= minLimit &&
number <= maxLimit)
            return number;
        else
            Console.WriteLine("Wrong input. Try again: ");
    }
}

public void ShowBoard()
{
    Console.WriteLine("\nYour table is:");
    ShowArray();
    Console.WriteLine(" ");
    for (int c = 0; c < size; c++)
        Console.WriteLine($" {c%10} ");
    Console.WriteLine();

    for (int i = 0; i < size; i++)
    {
        Console.WriteLine(i%10 + " ");
    }
}

```

```

        for (int j = 0; j < size; j++)
        {
            if (table[i] == j)
                Console.Write("[Q]");
            else
                Console.Write("[_]");
        }
        Console.WriteLine();
    }
}

public void ShowArray()
{
    Console.Write("{ ");
    for (int i = 0; i < table.Length; i++)
    {
        if (i == table.Length - 1)
            Console.Write(table[i] + " ");
        else
            Console.Write(table[i] + "; ");
    }
    Console.WriteLine("}");
}
}
}

```

LDFS.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2PA
{
    internal class LDFS : SearchAssistant
    {
        public LDFS(Node node, int limit)
        {
            statsAssistant = new StatsAssistant();
            statsAssistant.stopwatch.Start();

            result = DepthLimitedSearch(node, limit);
            statsAssistant.stopwatch.Stop();

            statsAssistant._statesInMemory =
statsAssistant._statesInMemoryHashSet.Count;
            statsAssistant._totalStates = statsAssistant._totalStatesHashSet.Count;
        }
        private Result DepthLimitedSearch(Node node, int limit)
        {
            return RecursiveDLS(node, limit);
        }

        private Result RecursiveDLS(Node node, int limit)
        {
            statsAssistant._iterations += 1;
            bool cutoffOccured = false;

            if (statsAssistant.stopwatch.ElapsedMilliseconds >= 30 * 60 * 1000)
            {
                statsAssistant._deadEnds += 1;
                return new Result(node, false, false, true);
            }
        }
    }
}

```

```

        if (node.state.CountConflicts() == 0)
            return new Result(node, false, false, false);

        else if (node.depth == limit)
        {
            statsAssistant._deadEnds += 1;
            return new Result(null, false, true, false);
        }

        Node.Expand(node);

        foreach (var successor in node.successors)
        {
            statsAssistant._totalStatesHashSet.Add(successor.GetState());

            Result result = RecursiveDLS(successor, limit);

            if (result.GetTimeIsUp())
                return result;

            if (result.GetCutoff())
                cutoffOccured = true;

            if (result != null && result.GetNode() != null)
            {
                foreach (var s in node.successors)
                    statsAssistant._statesInMemoryHashSet.Add(s.GetState());
                return result;
            }
        }

        if (cutoffOccured)
            return new Result(null, false, true, false);

        else
            return new Result(null, true, false, false);
    }
}

```

Node.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2PA
{
    internal class Node
    {
        private Node parent;

        public ChessBoard state;
        public int depth;
        public List<Node> successors;

        public int GetDepth()
        {
            return depth;
        }

        public ChessBoard GetState()
    }
}

```

```

    {
        return state;
    }
    public List<Node> GetSuccessors()
    {
        return successors;
    }
    public Node(ChessBoard InitialState)
    {
        parent = null;
        state = InitialState;
        depth = 0;
    }

    public Node(ref Node parentNode, int queenRow, int dest)
    {
        parent = parentNode;
        depth = parentNode.depth + 1;
        state = new ChessBoard(parentNode.state.GetSize(),
        parentNode.state.GetArray());
        state.MoveQueen(queenRow, (byte)dest);
    }

    public static void Expand(Node node)
    {
        int index = 0;
        node.successors = new List<Node>();

        for (int row = 0; row < node.state.GetSize(); row++)
        {
            for (int newCol = 0; newCol < node.state.GetSize(); newCol++)
            {
                if (newCol != node.state.GetQueenCol(row))
                {
                    node.successors.Add(new Node(ref node, row, newCol));
                }
            }
        }
    }
}

```

Program.cs:

```

using System;

namespace Lab2PA
{
    class Program
    {
        static void Main()
        {
            while (true)
            {
                ChessBoard startBoard = new ChessBoard("size of the table", 4, 25);
                startBoard.ShowBoard();
                Console.WriteLine($"There are {startBoard.CountConflicts()} conflicts
on a board");

                Node root = new Node(startBoard);
                SearchAssistant searchAssistant =
SearchAssistant.ChooseAlgorithm(root, (int)startBoard.GetSize());
                Result result = searchAssistant.GetResult();
                StatsAssistant statsAssistant = searchAssistant.GetStats();
            }
        }
    }
}

```

```

        Console.WriteLine($"Time taken:
{statsAssistant.stopwatch.ElapsedMilliseconds} ms");
        statsAssistant.ViewStats();

        if (!result.GetCutoff() && !result.GetFailed())
        {
            Console.WriteLine("Solution was found succesfully");
            ChessBoard endBoard = searchAssistant.GetBoard();
            endBoard.ShowBoard();
            Console.WriteLine($"There are {endBoard.CountConflicts()}
conflicts on a board");
        }
        else if (result.GetCutoff())
        {
            Console.WriteLine("The search was cut off");
        }
        else
        {
            Console.WriteLine("The search was failed");
        }
    }
}
}
}

```

RBFS.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2PA
{
    internal class RBFS : SearchAssistant
    {
        public RBFS(Node node)
        {
            statsAssistant = new StatsAssistant();
            statsAssistant.stopwatch.Start();

            result = RecursiveBestFirstSearch(node);
            statsAssistant.stopwatch.Stop();

            statsAssistant._statesInMemory =
statsAssistant._statesInMemoryHashSet.Count;
            statsAssistant._totalStates = statsAssistant._totalStatesHashSet.Count;
        }

        private Result RecursiveBestFirstSearch(Node node)
        {
            int limit = int.MaxValue;
            int depthOfRecursion = 0;
            return RecursiveBFS(node, limit, depthOfRecursion);
        }

        private Result RecursiveBFS(Node node, int limit, int depth)
        {
            statsAssistant._iterations += 1;

            int MAXDEPTH = 16 * (node.state.GetSize() * node.state.GetSize() - 1);
            // to avoid StackOverflow exception

```



```

if (statsAssistant.stopwatch.ElapsedMilliseconds >= 30 * 60 * 1000)
{
    statsAssistant._deadEnds += 1;
    return new Result(null, false, false, true);
}

if (node.state.CountConflicts() == 0)
    return new Result(node, false, false, false);

if (depth > MAXDEPTH)
{
    statsAssistant._deadEnds += 1;
    return new Result(null, false, true, false);
}

Node.Expand(node);
if (node.successors.Count == 0)
{
    statsAssistant._deadEnds += 1;
    return new Result(null, true, false, false);
}

List<int> f = new List<int>();
foreach (var successor in node.successors)
{
    statsAssistant._totalStatesHashSet.Add(successor.GetState());
    f.Add(successor.state.CountConflicts());
    //f.Add(Math.Max(s.state.CountConflicts() + s.depth,
s.state.CountConflicts())); (for test)
}

while (true)
{
    int bestValue = f.Min();
    int bestIndex = f.IndexOf(bestValue);
    Node bestNode = node.successors[bestIndex];

    if (bestValue > limit)
        return new Result(null, true, false, false);

    f.Remove(bestValue);
    node.successors.Remove(bestNode);

    int alternative = f.Min();
    limit = Math.Min(limit, alternative);

    Result result = RecursiveBFS(bestNode, limit, depth + 1);

    if (result.GetTimeIsUp())
        return result;

    if (result != null && result.GetNode() != null)
    {
        foreach (var s in node.successors)
            statsAssistant._statesInMemoryHashSet.Add(s.GetState());
        return result;
    }

    return new Result(null, true, false, false);
}
}
}
}

```

Result.cs:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2PA
{
    internal class Result
    {
        private Node _node;
        public Node GetNode()
        {
            return _node;
        }
        private bool _isFailed;
        public bool GetFailed() { return _isFailed; }
        private bool _cutOff;
        public bool GetCutOff() { return _cutOff; }
        private bool _timeIsUp;
        public bool GetTimeIsUp() { return _timeIsUp; }
        public Result(Node node, bool isFailed, bool cutOff, bool timeIsUp)
        {
            _node = node;
            _cutOff = cutOff;
            _isFailed = isFailed;
            _timeIsUp = timeIsUp;
        }
    }
}
```

SearchAssistant.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2PA
{
    abstract class SearchAssistant
    {
        protected Result result;
        public Result GetResult()
        {
            return result;
        }

        protected StatsAssistant statsAssistant;
        public StatsAssistant GetStats()
        {
            return statsAssistant;
        }

        public static SearchAssistant ChooseAlgorithm(Node node, int limit)
        {
            Console.WriteLine("Choose algorithm: (0 - LDFS; 1 - RBFS) ");
            while (true)
            {
                string input = Console.ReadLine();
                if (int.TryParse(input, out int number) && number == 0)
                {
                    return new LDFS(node, limit);
                }
                else if (int.TryParse(input, out int number) && number == 1)
                {
                    return new RBFS(node, limit);
                }
            }
        }
    }
}
```

```

        {
            return new LDFS(node, limit);
        }
        else if(number == 1)
        {
            return new RBFS(node);
        }
        else
        {
            Console.WriteLine("Wrong input. Try again: ");
        }
    }
}
public ChessBoard GetBoard()
{
    return result.GetNode().GetState();
}
}
}

```

StatsAssistant.cs:

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab2PA
{
    internal class StatsAssistant
    {
        public long _iterations { get; set; }

        public long _totalStates { get; set; }

        public long _statesInMemory { get; set; }

        public int _deadEnds { get; set; }
        public Stopwatch stopwatch;

        public HashSet<ChessBoard> _totalStatesHashSet { get; set; }
        public HashSet<ChessBoard> _statesInMemoryHashSet { get; set; }

        public StatsAssistant()
        {
            _iterations = 0;
            _totalStates = 0;
            _statesInMemory = 0;
            _deadEnds = 0;
            stopwatch = new Stopwatch();

            _totalStatesHashSet = new HashSet<ChessBoard>();
            _statesInMemoryHashSet = new HashSet<ChessBoard>();
        }

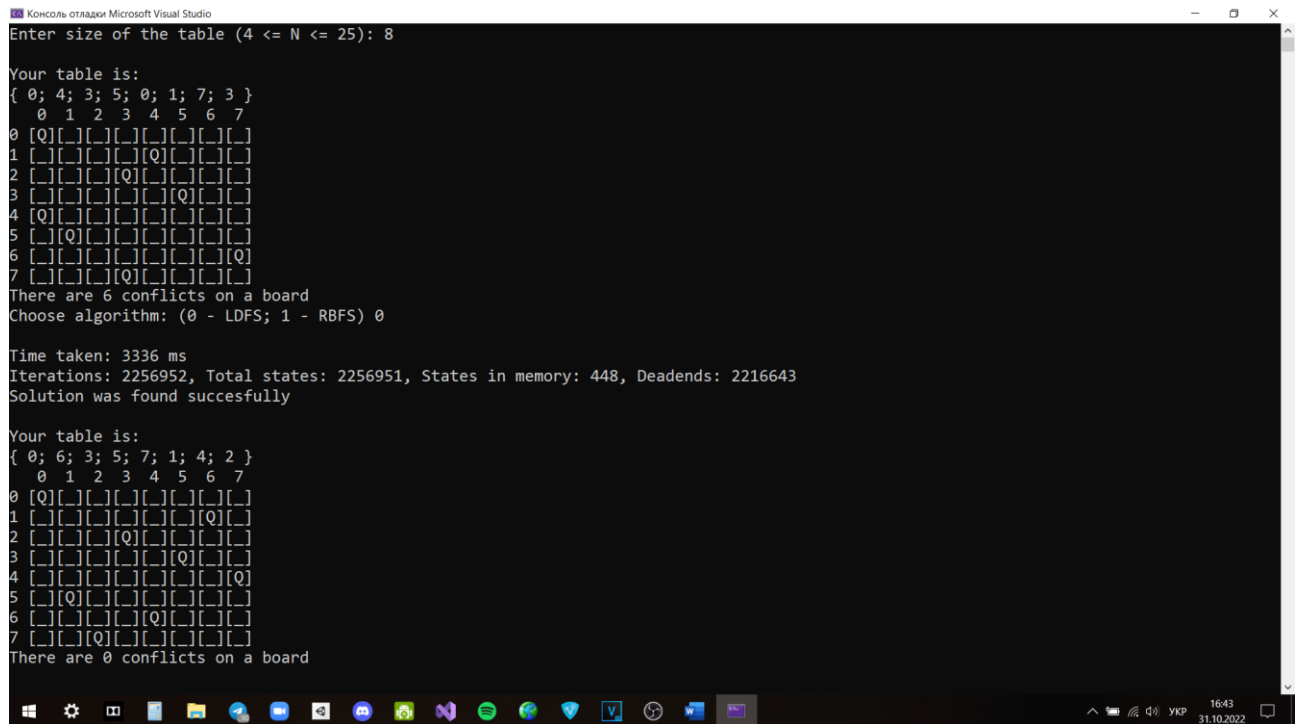
        public void ViewStats()
        {
            Console.WriteLine($"Iterations: {_iterations} , Deadends: {_deadEnds} ,
Total states: {_totalStates} , " +
                $" States in memory: {_statesInMemory} ");
        }
    }
}

```


3.2.2 Приклади роботи

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

Рисунок 3.1 – Алгоритм LDFS



```
Консоль отладки Microsoft Visual Studio
Enter size of the table (4 <= N <= 25): 8

Your table is:
{ 0; 4; 3; 5; 0; 1; 7; 3 }
  0 1 2 3 4 5 6 7
0 [Q][ ][ ][ ][ ][ ][ ][ ]
1 [ ][ ][ ][ ][Q][ ][ ][ ]
2 [ ][ ][ ][Q][ ][ ][ ][ ]
3 [ ][ ][ ][ ][ ][Q][ ][ ]
4 [Q][ ][ ][ ][ ][ ][ ][ ]
5 [ ][Q][ ][ ][ ][ ][ ][ ]
6 [ ][ ][ ][ ][ ][ ][Q][ ]
7 [ ][ ][ ][Q][ ][ ][ ][ ]

There are 6 conflicts on a board
Choose algorithm: (0 - LDFS; 1 - RBFS) 0

Time taken: 3336 ms
Iterations: 2256952, Total states: 2256951, States in memory: 448, Deadends: 2216643
Solution was found succesfully

Your table is:
{ 0; 6; 3; 5; 7; 1; 4; 2 }
  0 1 2 3 4 5 6 7
0 [Q][ ][ ][ ][ ][ ][ ][ ]
1 [ ][ ][ ][ ][ ][Q][ ][ ]
2 [ ][ ][ ][Q][ ][ ][ ][ ]
3 [ ][ ][ ][ ][ ][Q][ ][ ]
4 [ ][ ][ ][ ][ ][ ][Q][ ]
5 [ ][Q][ ][ ][ ][ ][ ][ ]
6 [ ][ ][ ][ ][Q][ ][ ][ ]
7 [ ][ ][Q][ ][ ][ ][ ][ ]

There are 0 conflicts on a board
```

Рисунок 3.2 – Алгоритм RBFS

```
Выбрать Консоль отладки Microsoft Visual Studio
Enter size of the table (4 <= N <= 25): 8

Your table is:
{ 5; 1; 7; 0; 7; 7; 4; 3 }
  0 1 2 3 4 5 6 7
0 [_][_][_][_][Q][_][_]
1 [_][Q][_][_][_][_][_]
2 [_][_][_][_][_][_][Q]
3 [Q][_][_][_][_][_][_]
4 [_][_][_][_][_][_][Q]
5 [_][_][_][_][_][_][Q]
6 [_][_][_][_][Q][_][_]
7 [_][_][Q][_][_][_][_]

There are 4 conflicts on a board
Choose algorithm: (0 - LDFS; 1 - RBFS) 1

Time taken: 18 ms
Iterations: 4, Total states: 168, States in memory: 165, Deadends: 0
Solution was found succesfully

Your table is:
{ 5; 1; 6; 0; 3; 7; 4; 2 }
  0 1 2 3 4 5 6 7
0 [_][_][_][_][Q][_][_]
1 [_][Q][_][_][_][_][_]
2 [_][_][_][_][_][_][Q]
3 [Q][_][_][_][_][_][_]
4 [_][_][_][Q][_][_][_]
5 [_][_][_][_][_][_][Q]
6 [_][_][_][_][Q][_][_]
7 [_][_][Q][_][_][_][_]

There are 0 conflicts on a board
```

3.3 Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму LDFS задачі 8 ферзів для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання LDFS

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пам'яті
{ 3; 6; 7; 5; 0; 5; 4; 2 }	56264	55252	56263	448
{ 6; 5; 5; 3; 4; 6; 3; 7 }	13659050	13415132	13659049	448
{ 6; 2; 6; 1; 0; 2; 0; 5 }	813425	798893	813424	448
{ 3; 6; 2; 3; 4; 4; 7; 4 }	4581242	4499428	4581241	448
{ 1; 3; 6; 7; 1; 2; 5; 2 }	39388	38677	39387	448
{ 2; 0; 6; 0; 5; 4; 3; 3 }	739354	726144	739353	448
{ 7; 5; 3; 3; 5; 3; 2; 4 }	2588161	2541937	2588160	448
{ 0; 7; 4; 4; 0; 7; 1; 3 }	433456	425709	433455	448
{ 0; 6; 1; 3; 2; 4; 7; 0 }	618307	607259	618306	448
{ 4; 3; 4; 7; 4; 2; 4; 1 }	381849	375023	381848	448
{ 3; 7; 7; 5; 3; 2; 6; 5 }	12231316	12012893	12231315	448
{ 0; 4; 1; 3; 1; 0; 3; 1 }	12247053	12028349	12247052	448
{ 7; 4; 5; 6; 2; 7; 7; 7 }	3467164	3405244	3467163	448
{ 1; 0; 5; 0; 5; 4; 1; 3 }	926937	910377	926936	448
{ 2; 5; 7; 2; 0; 4; 7; 3 }	488019	479298	488018	448
{ 4; 6; 2; 4; 0; 3; 7; 7 }	3128703	3072827	3128702	448
{ 4; 4; 6; 7; 3; 7; 3; 3 }	3846101	3777414	3846100	448
{ 6; 0; 6; 4; 6; 3; 3; 2 }	289485	284309	289484	448
{ 5; 7; 2; 6; 2; 4; 3; 3 }	2409524	2366490	2409523	448
{ 2; 5; 4; 2; 7; 6; 7; 3 }	12774124	12546008	12774123	448

Середня кількість ітерацій – 3785946,1

Середня кількість глухих кутів – 3718333,15

Середня кількість згенерованих станів – 3785945,1

Середня кількість станів у пам'яті – 448.0

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

Таблиця 3.2 – Характеристики оцінювання RBFS

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пам'яті
{ 5; 1; 3; 7; 4; 0; 1; 2 }	9	0	448	440
{ 2; 5; 2; 5; 3; 3; 6; 1 }	6	0	280	275
{ 4; 4; 3; 4; 7; 7; 6; 0 }	6	0	280	275
{ 0; 4; 2; 6; 2; 5; 6; 4 }	7	0	336	330
{ 0; 3; 4; 0; 1; 7; 3; 6 }	7	0	336	330
{ 7; 3; 3; 0; 5; 7; 4; 5 }	6	0	280	275
{ 6; 6; 2; 6; 1; 4; 3; 1 }	10	0	504	495
{ 0; 7; 7; 7; 5; 0; 2; 5 }	5	0	224	220
{ 0; 3; 0; 2; 0; 7; 7; 1 }	6	0	280	275
{ 2; 7; 7; 6; 3; 3; 3; 3 }	6	0	280	275
{ 2; 4; 7; 0; 5; 5; 6; 4 }	128	1	7112	0
{ 0; 6; 1; 3; 5; 1; 4; 0 }	8	0	392	385
{ 1; 3; 3; 7; 7; 1; 3; 7 }	6	0	280	275
{ 7; 7; 5; 7; 3; 5; 4; 1 }	6	0	280	275
{ 5; 1; 3; 5; 7; 5; 6; 7 }	7	0	336	330
{ 6; 3; 1; 7; 5; 7; 5; 6 }	4	0	168	165
{ 3; 3; 3; 0; 3; 3; 6; 7 }	9	0	448	440
{ 1; 5; 2; 7; 4; 5; 2; 7 }	128	1	7112	0
{ 2; 5; 7; 5; 3; 6; 6; 3 }	4	0	168	165
{ 7; 1; 2; 4; 7; 5; 3; 0 }	128	1	7112	0

Середня кількість ітерацій (при успішному виконанні) – 6,6

Частота зустрічання глухих кутів – 0,15

Середня кількість згенерованих станів (при успішному виконанні) – 312,9

Середня кількість станів у пам'яті(при успішному виконанні) 307,4

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто два алгоритми пошуку: АНП Limited Depth-First Search та Recursive Best First Search з евристичною функцією F1.

Було реалізовано обидва алгоритми, розроблено псевдокод. Обидва алгоритми були протестовані на 20 випадкових початкових станах кожен. З отриманих даних можна зробити такі висновки:

LDFS не є повним алгоритмом та до того ж часто перевищує усі припустимі границі пам'яті через рекурсивний принцип роботи, так і не знайшовши результату. Порівняно з RBFS виглядає менш оптимальним, але більш надійним.

RBFS же зі свого боку працює дуже швидко, проходить малу кількість ітерацій, проте іноді створює завелику глибину рекурсії, що призводить до виникнення повідомлення про помилку: «Stack Overflow», через що довелося його обмежити в глибині, що іноді призводить до передчасного завершення пошуку.

Виконавши реалізацію обох алгоритмів та протестувавши їх, хочу зробити висновок, що евристичні методи пошуку є більш оптимальними, а клас задач, які вони можуть виконувати – достатньо великий, тому оцінка найбільш вигідних вузлів дає набагато швидший результат, ніж їх простий перебір у глибину.

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%.