

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-15 Лазюта О. С.
(шифр, прізвище, ім'я, по батькові)

Перевірив

Головченко М.М.
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ	7
3.1	ПСЕВДОКОД АЛГОРИТМІВ.....	7
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ	10
3.2.1	<i>Вихідний код.....</i>	<i>10</i>
3.2.2	<i>Приклади роботи.....</i>	<i>14</i>
3.3	ДОСЛІДЖЕННЯ АЛГОРИТМІВ.....	16
	ВИСНОВОК	26
	КРИТЕРІЇ ОЦІНЮВАННЯ	27

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 **АП**, 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 (1Гб).

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, щоскладається 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 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
15	8-ферзів	IDS	A*		F1

3.1 Псевдокод алгоритмів

1.IDS

Procedure IDS() *declare* int totalDepth: 8 **for each** limit less than totalDepth + 1 **do**

limit++

if solved == true **then**

return

end if

DFSRecursive(limit)

end for**end Procedure****Procedure** DFSRecursive(*declare* int limit, *declare* int depth) **if** depth == limit or solved == true **then** **if** solved == false **then** **if** CheckOnGoalBoard(queensPosition) == true **then**

solved : true

end if **end if**

return

end if **for each** i less than nQueensSize **do**

i++

if solved == true **then**

return

end if **else**

queens[depth] : (newQueens[depth] + i) % 8

DFSRecursive(limit, depth + 1)

end else **end for****end Procedure**

2.A*

Procedure AStarFunc (*declare* int[] initialBoard) *declare* PriorityQueue<Element> openQueue *declare* List<Element.BoardPositions> closed

```

    Element current
    EnqueueInitialBoard(intitialBoard, openQueue)
    while openQueue.Length != false do
        current = openQueue.Dequeue()
        if FindAttackPairs(current.BoardPositions) == 0 then
            return current.BoardPosition
        closed.Add(current.BoardPosition)
        posterity = GeneratePosterity(current)
        for each posterior in posterity do
            if closed.Contains(posterior.BoardPosition) then
                openQueue.Enqueue(Element, FindAttackPairs +
depth)
            end if
        end for each
    end while
    return null
end Procedure

```

3.Heuristic function F1

```

Procedure FindAttackPairs(int[] queensBoard)
    declare int attacked : 0
    declare int k : 0;
    declare bool horDimension1 : false, horDimension2 : false,
        diagDimension1 : false, diagDimension2 : false,
        diagDimension3 : false, diagDimension4 : false
    for each element in queensBoard do
        for i less than nQueensSize do
            i++
            if k==i then
                continue
            end if
            declare diff : Abs(i - k)
            if Abs(queensBoard[i] - el) == diff
                and element - queensBoard > 0 and
                and k - i > 0 and diagDimension1 == false
then
                attacked++
                diagDimension1 : true

```



```

    end if
    else if Abs(queensBoard[i] - el) == diff
        and element - queensBoard < 0 and
        and k - i > 0 and diagDimension2 == false
then
        attacked++
        diagDimension2 : true
    end if
    else if Abs(queensBoard[i] - el) == diff
        and element - queensBoard > 0 and
        and k - i < 0 and diagDimension3 == false
then
        attacked++
        diagDimension3 : true
    end if
    else if Abs(queensBoard[i] - el) == diff
        and element - queensBoard < 0 and
        and k - i < 0 and diagDimension == false then
        attacked++
        diagDimension4 : true
    end if
    if element == queensBoard[i] and k - i > 0 and
horDimension1 == false
        attacked++
        horizonDimension1 : true
    end if
    else if element == queensBoard[i] and k - i < 0 and
horDimension2 == false
        attacked++
        horizonDimension2 : true
    end if
end for
horDimension1 : false
horDimension2 : false
diagDimension1 : false
diagDimension2 : false
diagDimension3 : false
diagDimension4 : false
k++

```

end for each
return attacked / 2
end Procedure

3.2 Програмна реалізація

3.2.1 Вихідний код

Файл Program.cs:

```
int[] queens = new int[] { 0, 5, 2, 1, 5, 2, 3, 7 };

const int nQueensSize = 8;

int iterations = 0;
int deadEnds = 0;
int states = 0;
int statesInMemory = 1;

int[,] SetBoard(int[] queensPos)
{
    int[,] emptyBoard = new int[8,8]
    {
        { 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 0, 0, 0 }
    };

    for(int i = 0; i < 8; i++)
    {
        emptyBoard[queensPos[i], i] = 1;
    }

    return emptyBoard;
}

int[] newQueens = queens.Take(8).ToArray();
bool solved = false;

void WriteScreen(int[,] board)
{
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
            Console.Write(board[i, j] + " ");
        Console.WriteLine("\n");
    }
}

void IDS()
{
    int totalDepth = 8;
    for(int limit = 1; limit < totalDepth + 1; limit++)
    {
        if (solved) return;
    }
}
```

```

        newQueens = queens.Take(8).ToArray();
        DFSRecursive(limit);
    }
}

void DFSRecursive(int limit = nQueensSize, int depth = 0)
{
    if(depth == limit || solved)
    {
        if (depth == limit) deadEnds++;
        if(!solved)
            if (CheckOnGoalBoard(newQueens))
                solved = true;
        return;
    }

    for (int i = 0; i < nQueensSize; i++)
    {
        states++;
        iterations++;
        if (solved) return;
        newQueens[depth] = (newQueens[depth] + i) % nQueensSize;
        DFSRecursive(limit, depth + 1);
    }
}

bool CheckOnGoalBoard(int[] queens)
{
    int k = 0;
    foreach(var el in queens)
    {
        for(int i = 0; i < nQueensSize; i++)
        {
            if (k == i) continue;           // перевірка на однаковий елемент

            int diff = Math.Abs(i - k);
            if (Math.Abs(queens[i] - el) == diff) // перевірка по діагоналям
                return false;

            if (el == queens[i])           // перевірка по горизонталям
                return false;
        }
        k++;
    }
    return true;
}

Main();

void Main()
{
    Console.WriteLine("IDS algorithm");
    Console.WriteLine("initial board:");
    WriteScreen(SetBoard(queens));
    IDS();
    Console.WriteLine();
    Console.WriteLine("solved board:");
    WriteScreen(SetBoard(newQueens));
    Console.WriteLine($"iterations : {iterations}");
    Console.WriteLine($"deadEnds : {deadEnds}");
    Console.WriteLine($"states : {states}");
    Console.WriteLine($"states in memory : {newQueens.Length}");
    //AStar astar = new AStar();
    //astar.Main();
}

```

}

Кінець файлу Program.cs

Файл Astar.cs :

```
class AStar
{
    const int nQueensSize = 8;
    static int[] queens = new int[] { 1, 7, 7, 3, 3, 3, 5, 5 };

    int iterations = 0;
    int deadEnds = 0;
    int states = 0;

    int[,] SetBoard(int[] queensPos)
    {
        int[,] emptyBoard = new int[nQueensSize, nQueensSize]
        {
            { 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0 },
            { 0, 0, 0, 0, 0, 0, 0, 0 }
        };

        for (int i = 0; i < nQueensSize; i++)
        {
            emptyBoard[queensPos[i], i] = 1;
        }

        return emptyBoard;
    }

    void WriteScreen(int[,] board)
    {
        for (int i = 0; i < nQueensSize; i++)
        {
            for (int j = 0; j < nQueensSize; j++)
            {
                Console.Write(board[i, j] + " ");
            }
            Console.WriteLine();
        }
    }

    (int[], int) AStarFunc(int[] initialBoard)
    {
        PriorityQueue<(int, int, int[]), int> openQueue = new();
        List<string> closed = new();
        (int, int, int[]) current;
        EnqueueInitialBoard(initialBoard, openQueue); // enqueue children of 0 depth
        while(openQueue.Count != 0)
        {
            current = openQueue.Dequeue(); // extracting min element
            if (FindAttackPairs(current.Item3) == 0)
            {
                return (current.Item3, closed.Count + openQueue.Count + 1); // checking if problem is solved
            }
            closed.Add(String.Join("", current.Item3));
            if (GeneratePosterity(current) == null) continue; // checking if elements can generate children
            (int, int, int[][]) posterity = GeneratePosterity(current).Take(8).ToArray();
            foreach (var posterior in posterity)
            {

```

```

        iterations++;
        if (!closed.Contains(String.Join("", posterior.Item3)))
        {
            openQueue.Enqueue(posterior, FindAttackPairs(posterior.Item3) + posterior.Item1); // enqueue children if they
not already in queue
            states++;
        }
    }

    return (null, 0);
}

void EnqueueInitialBoard(int[] initialBoard, PriorityQueue<(int, int, int[]), int> openQueue)
{
    for (int i = 0; i < 8; i++)
    {
        states++;
        int[] boardState = initialBoard.Take(8).ToArray();
        boardState[0] = (initialBoard[0] + i) % nQueensSize;
        openQueue.Enqueue((0, (initialBoard[0] + i) % nQueensSize, boardState), FindAttackPairs(boardState));
    }
}

(int, int, int[][]) GeneratePosterity((int, int, int[]) current)
{
    if (current.Item1 == 7)
        return null;
    (int, int, int[][]) result = new (int, int, int[])[nQueensSize];
    for (int i = 0; i < 8; i++)
    {
        int[] boardState = current.Item3.Take(8).ToArray();
        boardState[current.Item1 + 1] = (current.Item2 + i) % nQueensSize;
        result[i] = (current.Item1 + 1, (current.Item2 + i) % nQueensSize, boardState);
    }
    return result;
}

int FindAttackPairs(int[] queens)
{
    int attacked = 0;
    int k = 0;
    bool horizonDimension1=false, horizonDimension2=false, diagDimension1=false,
        diagDimension2=false, diagDimension3=false, diagDimension4=false;
    foreach (var el in queens)
    {
        for (int i = 0; i < nQueensSize; i++)
        {
            if (k == i) continue;

            int diff = Math.Abs(i - k);
            if (Math.Abs(queens[i] - el) == diff && el - queens[i] > 0 && k - i > 0 && !diagDimension1)
            {
                attacked++;
                diagDimension1 = true;
            }

            else if (Math.Abs(queens[i] - el) == diff && el - queens[i] < 0 && k - i > 0 && !diagDimension2)
            {
                attacked++;
                diagDimension2 = true;
            }

            else if (Math.Abs(queens[i] - el) == diff && el - queens[i] > 0 && k - i < 0 && !diagDimension3)
            {
                attacked++;
            }
        }
    }
}

```

```

        diagDimension3 = true;
    }

    else if (Math.Abs(queens[i] - el) == diff && el - queens[i] < 0 && k - i < 0 && !diagDimension4)
    {
        attacked++;
        diagDimension4 = true;
    }

    if (el == queens[i] && k - i > 0 && !horizonDimension1)
    {
        attacked++;
        horizonDimension1 = true;
    }
    else if (el == queens[i] && k - i < 0 && !horizonDimension2)
    {
        attacked++;
        horizonDimension2 = true;
    }

    }
    horizonDimension1 = false;
    horizonDimension2 = false;
    diagDimension1 = false;
    diagDimension2 = false;
    diagDimension3 = false;
    diagDimension4 = false;
    k++;
}
return attacked / 2;
}

public void Main()
{
    Console.WriteLine("A* algorithm");
    Console.WriteLine("initial board:");
    WriteScreen(SetBoard(queens));
    Console.WriteLine();
    Console.WriteLine("solved board:");
    (int[], int) result = AStarFunc(queens);
    WriteScreen(SetBoard(result.Item1));
    Console.WriteLine($"iterations : {iterations}");
    Console.WriteLine($"deadEnds : {deadEnds}");
    Console.WriteLine($"states : {states}");
    Console.WriteLine($"statesInMemory : {result.Item2}");
}
}

```

3.2.2 Приклади роботи

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

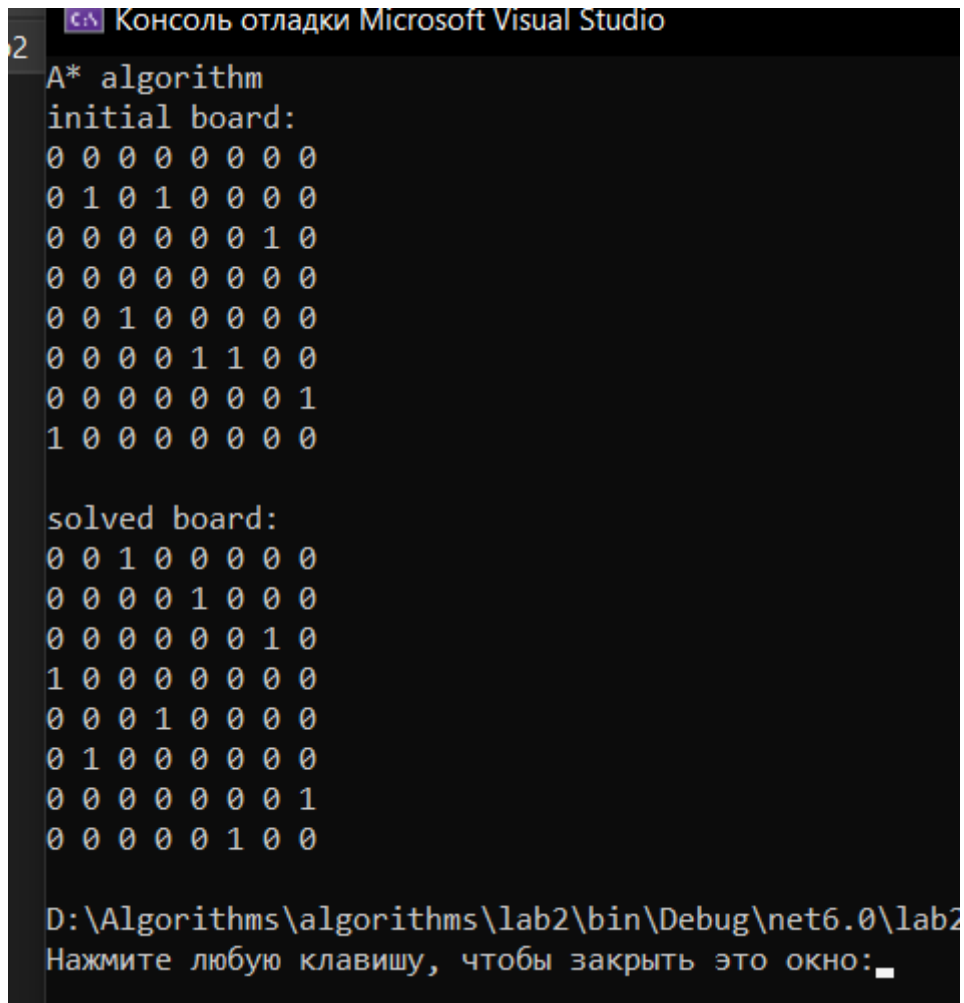
```
Консоль отладки Microsoft Visual Studio

IDS algorithm
initial board:
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1

solved board:
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1

D:\Algorithms\algorithms\lab2\bin\Debug\net6.0\lab2.exe
Нажмите любую клавишу, чтобы закрыть это окно: _
```

Рисунок 3.1 – Алгоритм IDS



```
2  C:\> Консоль отладки Microsoft Visual Studio
A* algorithm
initial board:
0 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0

solved board:
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0

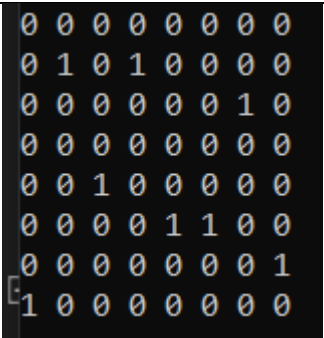
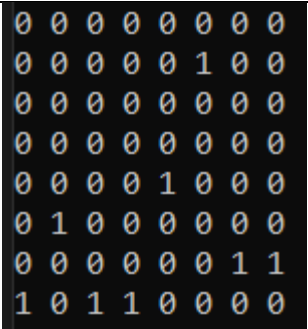
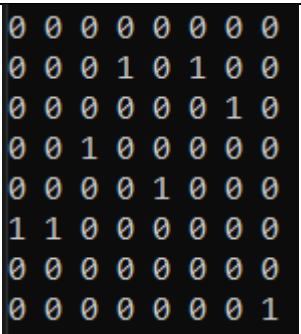
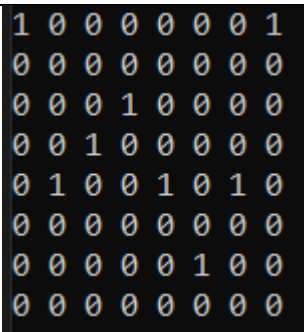
D:\Algorithms\algorithms\lab2\bin\Debug\net6.0\lab2
Нажмите любую клавишу, чтобы закрыть это окно: _
```

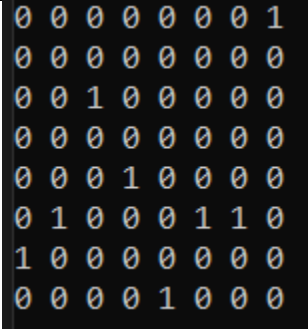
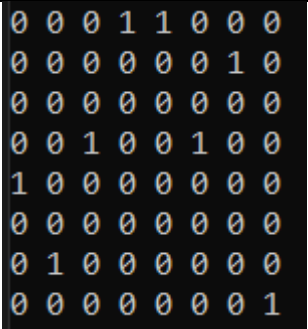
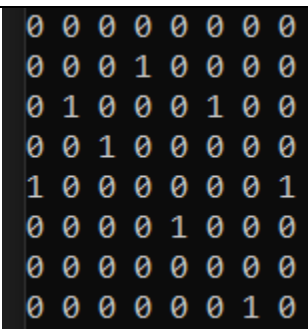
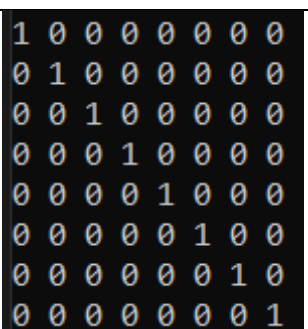
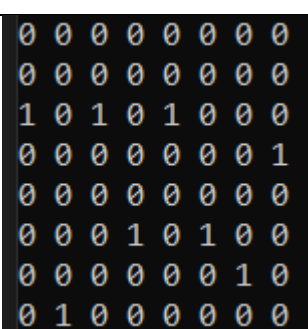
Рисунок 3.2 – Алгоритм A*

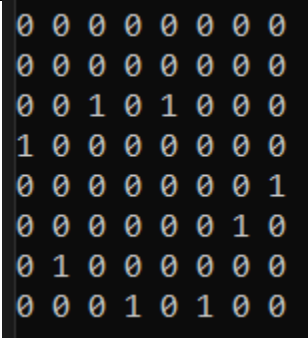
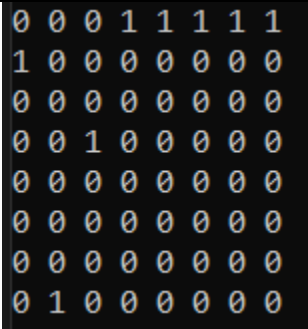
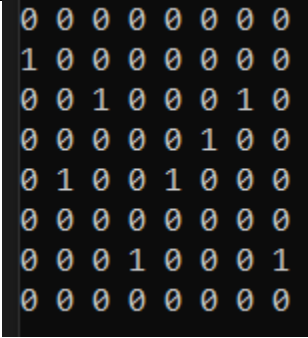
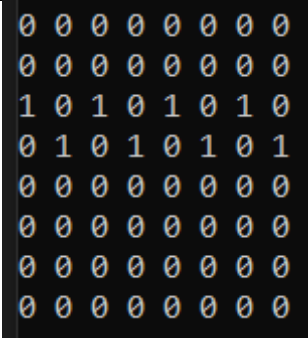
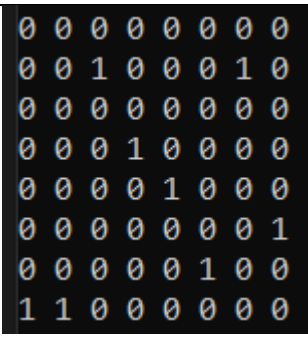
3.3 Дослідження алгоритмів

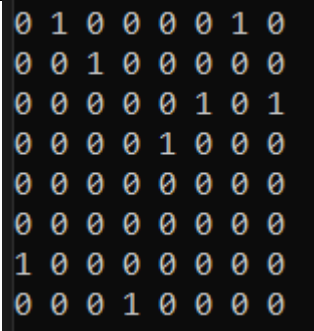
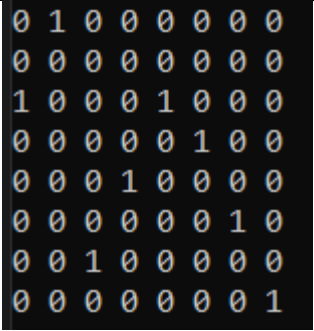
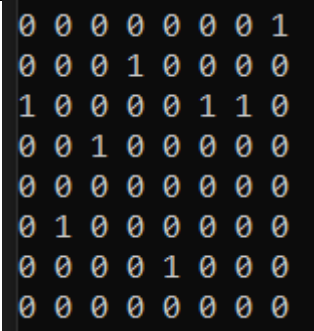
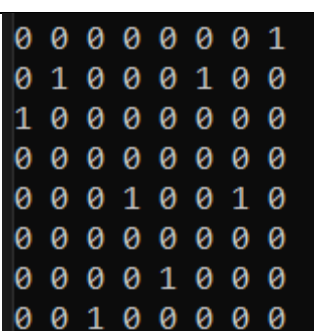
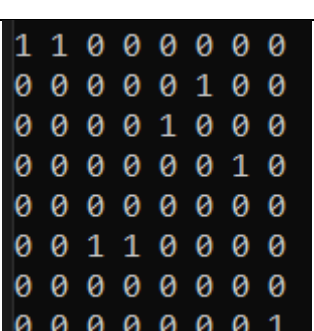
В таблиці 3.1 наведені характеристики оцінювання алгоритму IDS, задачі 8 ферзів для 20 початкових станів.

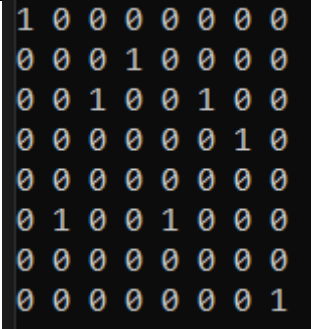
Таблиця 3.1 – Характеристики оцінювання IDS

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пом'яті
	309424	270737	309424	8
	1287439	1126507	1287439	8
	184391	161342	184391	8
	146973	128599	146973	8

	156003	136500	156003	8
	1270	1109	1270	8
	181353	158682	181353	8
	1187219	1038816	1187219	8
	58396	51094	58396	8

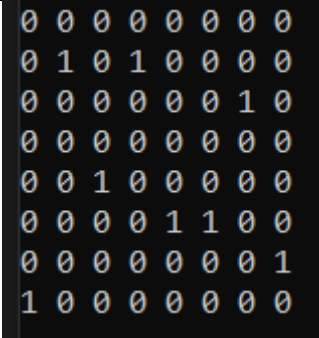
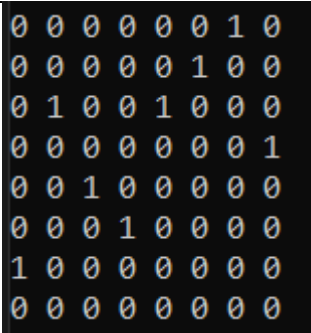
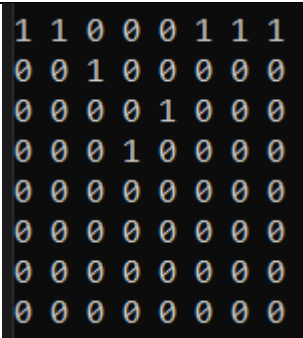
	399924	349930	399924	8
	701211	613557	701211	8
	199849	174865	199849	8
	518753	453906	518753	8
	18507	16193	18507	8

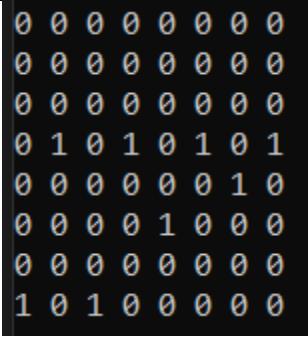
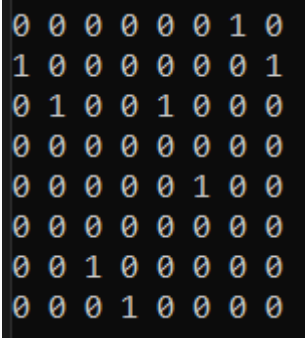
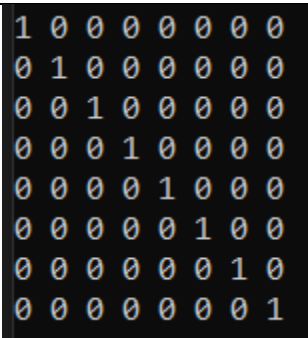
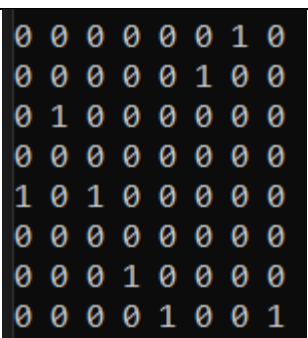
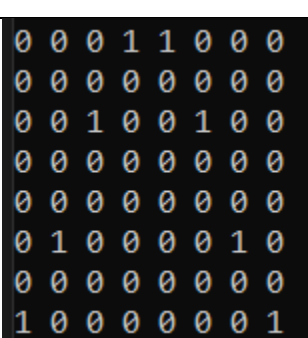
	258118	225856	258118	8
	806790	705938	806790	8
	342498	299681	342498	8
	114429	100124	114429	8
	328338	287299	328338	8

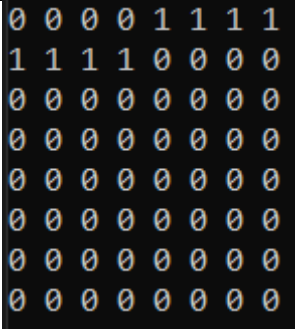
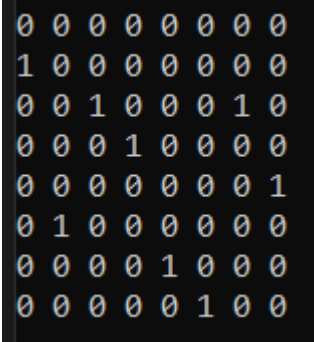
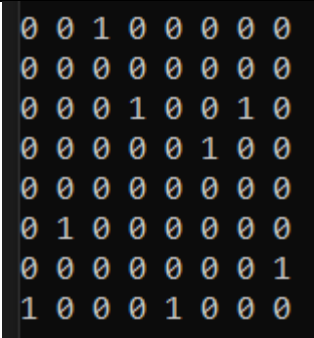
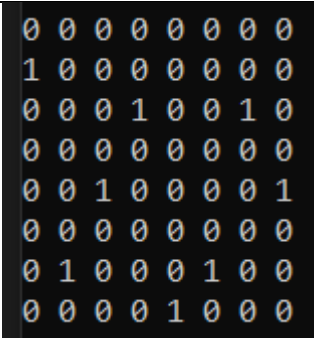
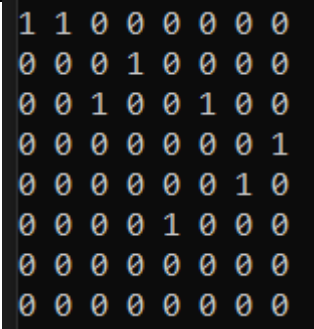
 <pre> 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 </pre>	336014	294012	336014	8
--	--------	--------	--------	---

В таблиці 3.2 наведені характеристики оцінювання алгоритму A*, задачі 8 ферзів для 20 початкових станів.

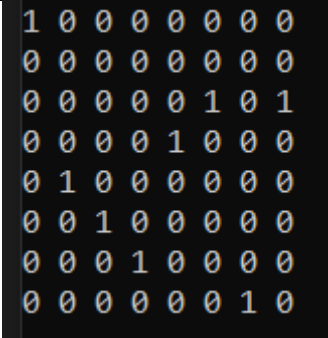
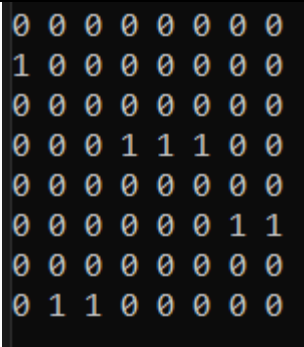
Таблиця 3.3 – Характеристики оцінювання A*

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пом'яті
 <pre> 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 </pre>	1456	0	1282	1282
 <pre> 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </pre>	10048	0	8800	8800
 <pre> 1 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 </pre>	4360	0	3823	3823

	4336	0	3802	3802
	3360	0	2948	2948
	21968	0	19230	19230
	680	0	603	603
	9776	0	8562	8562

	6208	0	5440	5440
	9936	0	8702	8702
	6888	0	6035	6035
	776	0	687	687
	704	0	624	624

<pre> 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 </pre>	5488	0	4810	4810
<pre> 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 </pre>	7000	0	6133	6133
<pre> 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 </pre>	2880	0	2528	2528
<pre> 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 </pre>	1136	0	1002	1002
<pre> 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 </pre>	728	0	645	645

	1496	0	1317	1317
	312	0	281	281

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто та досліджено алгоритми неінформативного та інформативного пошуку для розв'язання задачі 8 ферзів. А саме алгоритми IDS і A*. IDS був реалізований за принципом «AS IS» а A* використовував задану за варіантом евристичну функцію 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).

Було записано псевдокод функцій IDS, A* та евристичної функції F1 та проведено дослідження щодо ефективності роботи даних алгоритмів.

Було використано 20 початкових станів для кожного випадку, для яких було пораховано кількість ітерацій, глухих кутів, всього станів та станів у пам'яті.

Отже, підрахувавши середню кількість даних для кожного алгоритму, маємо:

Алгоритм	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пам'яті
IDS	376 845	329 737	376 845	8
A*	4 977	0	4 363	4 363

Проаналізувавши вищенаведені дані, бачимо що алгоритм A* не має глухих кутів та виконує в рази менше ітерацій, що свідчить про те що він є набагато оптимальнішим за IDS алгоритм.

Отже, підсумувавши вищезазначені дослідження, алгоритми інформативного пошуку є значно ефективнішими за алгоритми неінформативного пошуку, оскільки вони використовують евристичні функції оцінки шляху.

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%.