

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, Гуменюк О.В.

(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	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>12</i>
3.3	ДОСЛІДЖЕННЯ АЛГОРИТМІВ.....	12
	ВИСНОВОК	25
	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 **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 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

3.1.1 Псевдокод евристичної функції F1

```
Function calculateConflictNum(placements: Int[][]): Int
    conflictNum: Int = 0
    xDuplicates: Int = кількість повторень координат рядків
    yDuplicates: Int = кількість повторень координат стовпців
    conflictNum = xDuplicates + yDuplicates
    directionalConflicts: Int[][][]
    for i = 1 to QUEEN_NUM do
        checkDirection(placements, i, 1, 1, directionalConflicts)
        checkDirection(placements, i, -1, 1, directionalConflicts)
        checkDirection(placements, i, 1, -1, directionalConflicts)
        checkDirection(placements, i, -1, -1, directionalConflicts)
    end for
    conflictNum = conflictNum + directionalConflicts.length
    return conflictNum
end

Function checkDirection(placements: Int[][], i: Int, xIncrement: Int,
    yIncrement: Int, directionalConflicts: Int[][][])
    queenCoords = placements[i]
    for i = 1 to QUEEN_NUM do
        coords = {queenCoords[1] + i*xIncrement,
            queenCoords[2] + i*yIncrement}
        if (placements.contains(coords)) do
            if ( !directionalConflicts.contains({queenCoords, coords}
                and !directionalConflicts.contains({coords, queenCoords}))
            do
                directionalConflicts.add({coords, queenCoords})
            end if
        end if
    end for
end
```


3.1.2 Псевдокод алгоритму LDFS

```
Function recursiveLDFS(problemNode: ProblemNode): ProblemNode or NULL
    if (calculateConflictNum(problemNode.placements) == 0) do
        return placements
    end if
    else if (problemNode.depth == QUEEN_NUM) do
        return NULL
    end else if
    else do
        children = problemNode.generateChildren()
        for child in children do
            result = recursiveLDFS(child)
            if (result != NULL) do
                return result
            end if
        end for
    end else
    return NULL
end
```

3.1.3 Псевдокод алгоритму A*

```
Function AStar(initialNode: ProblemNode): ProblemNode or NULL
    open: orderedArray(comparator(depth + conflictNum))
    closed: orderedArray(comparator(depth + conflictNum))
    open.insert(initialNode)
    while (open.length != 0) do
        current = open.extractMin()
        if (calculateConflictNum(current.placements) == 0) do
            return placements
        end if
        closed.insert(current)
        children = current.generateChildren()
        for child in children do
            if (!closed.binarySearchContains(child)) do
                open.insert(child)
            end if
        end for
    end while
    return NULL
end
```

3.2 Програмна реалізація

3.2.1 Вихідний код

```
fun calculateConflictNum() {
    val xValues = placements.map { it.first }
    val yValues = placements.map { it.second }
    conflictNum = getDuplicatesNum(xValues) + getDuplicatesNum(yValues)

    val directionalConflicts = ArrayList<Pair<Pair<Int, Int>, Pair<Int,
Int>>>>(0)

    for (i in 0 until QUEEN_NUM) {
        checkDirection(i, 1, 1, directionalConflicts)
        checkDirection(i, -1, 1, directionalConflicts)
        checkDirection(i, 1, -1, directionalConflicts)
        checkDirection(i, -1, -1, directionalConflicts)
    }
    conflictNum += directionalConflicts.size
}

private fun getDuplicatesNum(arr: List<Int>): Int {
    return arr.size - arr.distinct().count()
}

private fun checkDirection(queenNum: Int, xIncrement: Int, yIncrement:
Int, conflicts: ArrayList<Pair<Pair<Int, Int>, Pair<Int, Int>>>>){
    val (x, y) = placements[queenNum]
    for (i in 1 until QUEEN_NUM){
        val coords = Pair(x + xIncrement*i, y + yIncrement*i)
        if (placements.contains(coords)){
            addIfNotMatch(placements[queenNum], coords, conflicts)
            break
        }
    }
}

fun findSolutionLDFS(initialBoard: Board): Board?{
    return recursiveLDFS(initialBoard)
}

fun recursiveLDFS(board: Board): Board?{
    if (board.conflictNum == 0) return board
```

```

else if (board.depth == QUEEN_NUM) return null
else{
    val children = board.generateChildren()
    for (child in children){
        val result = recursiveLDFS(child)
        if (result != null) return result
    }
}
return null
}

fun findSolutionAStar(initialBoard: Board): Board?{
    val open = BoardOrderedArray()
    val closed = BoardOrderedArray()
    var current: Board

    open.insert(initialBoard)
    while (open.isNotEmpty()){
        current = open.extractMin()
        if (current.conflictNum == 0) {
            return current
        }
        closed.insert(current)
        val children = ArrayList(current.generateChildren().filter
{!closed.isPresent(it)})
        open.append(children)
    }
    return null
}

```

3.2.2 Приклади роботи

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

```
You may enter queens coordinates one by one or multiple at once, seperated by commas (ex. 1 2, 4 5 or 1, 2), or random
Enter problem difficulty (easy/hard): easy
Enter queen(s) coordinates (1-8) or random: random
[(1, 6), (2, 5), (3, 3), (4, 1), (5, 3), (6, 5), (7, 6), (8, 1)]
  1  2  3  4  5  6  7  8
1  _|_|_|_|_|_|_|_|#|_|_|_|
2  _|_|_|_|_|_|_|_|_|_|_|_|_|
3  _|_|_|_|_|_|_|_|_|_|_|_|_|
4  _|#|_|_|_|_|_|_|_|_|_|_|_|
5  _|_|_|_|_|_|_|_|_|_|_|_|_|
6  _|_|_|_|_|_|_|_|_|_|_|_|_|
7  _|_|_|_|_|_|_|_|_|_|_|_|_|
8  _|#|_|_|_|_|_|_|_|_|_|_|_|
Placement input is finished!
Enter algorithm type (a*/ldfs): ldfs
Running time of the program is 19.048 seconds
Success!
  1  2  3  4  5  6  7  8
1  _|_|_|_|_|_|_|_|_|_|_|_|_|
2  _|_|_|_|_|_|_|_|_|_|_|_|_|
3  _|_|_|_|_|_|_|_|_|_|_|_|_|
4  _|#|_|_|_|_|_|_|_|_|_|_|_|
5  _|_|_|_|_|_|_|_|_|_|_|_|_|
6  _|_|_|_|_|_|_|_|_|_|_|_|_|
7  _|_|_|_|_|_|_|_|_|_|_|_|_|
8  _|_|_|_|_|_|_|_|_|_|_|_|_|
Found solution in 8 steps! Steps:
```

Рисунок 3.1 – Алгоритм LDFS

```
You may enter queens coordinates one by one or multiple at once, seperated by commas (ex. 1 2, 4 5 or 1, 2), or random
Enter problem difficulty (easy/hard): easy
Enter queen(s) coordinates (1-8) or random: random
[(1, 1), (2, 3), (3, 2), (4, 7), (5, 2), (6, 5), (7, 5), (8, 2)]
  1  2  3  4  5  6  7  8
1  _|#|_|_|_|_|_|_|_|_|_|_|_|
2  _|_|_|_|_|_|_|_|_|_|_|_|_|
3  _|_|_|_|_|_|_|_|_|_|_|_|_|
4  _|_|_|_|_|_|_|_|_|_|_|_|_|
5  _|_|_|_|_|_|_|_|_|_|_|_|_|
6  _|_|_|_|_|_|_|_|_|_|_|_|_|
7  _|_|_|_|_|_|_|_|_|_|_|_|_|
8  _|_|_|_|_|_|_|_|_|_|_|_|_|
Placement input is finished!
Enter algorithm type (a*/ldfs): a*
Running time of the program is 1.028 seconds
Success!
  1  2  3  4  5  6  7  8
1  _|_|_|_|_|_|_|_|_|_|_|_|_|
2  _|_|_|_|_|_|_|_|_|_|_|_|_|
3  _|#|_|_|_|_|_|_|_|_|_|_|_|
4  _|_|_|_|_|_|_|_|_|_|_|_|_|
5  _|_|_|_|_|_|_|_|_|_|_|_|_|
6  _|_|_|_|_|_|_|_|_|_|_|_|_|
7  _|_|_|_|_|_|_|_|_|_|_|_|_|
8  _|_|_|_|_|_|_|_|_|_|_|_|_|
Found solution in 5 steps! Steps:
```

Рисунок 3.2 – Алгоритм A*

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 глухих кутів	Всього станів	Всього станів у пам'яті
<div><div><div>12345678</div><div>1 _ _ _ _ _ _ _ _ _ # _ </div><div>2 _ _ _ _ # _ _ _ _ _ _ _ </div><div>3 _ _ _ _ _ _ # _ _ _ _ _ _ </div><div>4 _ _ _ _ _ _ _ _ _ _ # _ _ </div><div>5 _ _ _ _ _ _ _ _ _ _ _ # _ </div><div>6 _ _ _ _ # _ _ _ _ _ _ _ _ </div><div>7 _ _ _ _ _ _ _ _ _ _ _ _ # </div><div>8 _ _ _ _ _ _ _ _ # _ _ _ _ </div></div></div> <td>8</td> <td>31 013 230</td> <td>31 581 704</td> <td>441</td>	8	31 013 230	31 581 704	441
<div><div><div>12345678</div><div>1 _ _ # _ _ _ _ _ _ _ _ _ </div><div>2 _ _ # _ _ _ _ _ _ _ _ _ </div><div>3 _ _ # _ _ _ _ _ _ _ _ _ </div><div>4 _ _ _ _ # _ _ _ _ _ _ _ _ </div><div>5 _ _ # _ _ _ _ _ _ _ _ _ </div><div>6 _ _ # _ _ _ _ _ _ _ _ _ </div><div>7 _ _ # _ _ _ _ _ _ _ _ _ </div><div>8 _ _ _ _ _ _ _ # _ _ _ _ _ </div></div></div> <td>8</td> <td>22 052 121</td> <td>22 456 336</td> <td>441</td>	8	22 052 121	22 456 336	441
<div><div><div>12345678</div><div>1 _ _ _ _ _ _ _ _ _ _ _ # </div><div>2 _ _ _ _ _ _ _ _ _ _ _ # _ </div><div>3 _ _ _ _ _ _ # _ _ _ _ _ _ _ </div><div>4 _ _ _ _ _ _ # _ _ _ _ _ _ _ </div><div>5 _ _ _ _ _ _ # _ _ _ _ _ _ _ </div><div>6 _ _ _ _ _ _ # _ _ _ _ _ _ _ </div><div>7 _ _ _ _ _ _ _ _ _ _ _ _ # </div><div>8 _ _ _ _ _ _ _ _ _ _ _ _ _ </div></div></div> <td>8</td> <td>24 553 885</td> <td>24 978 520</td> <td>441</td>	8	24 553 885	24 978 520	441

Продовження таблиці 3.1

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
	8	40 919 793	41 712 328	441
	8	11 829 384	12 046 216	441
	8	215 348	219 296	441

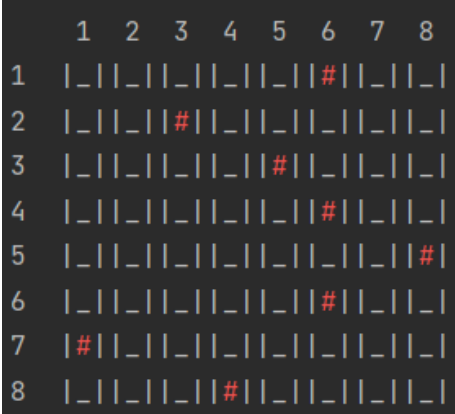
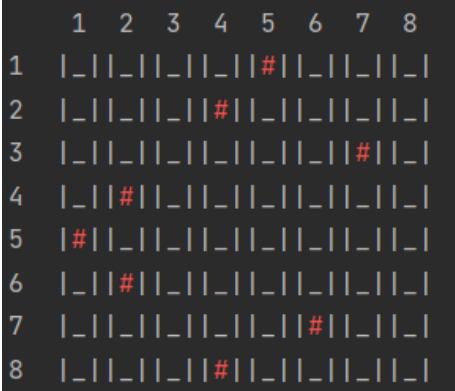
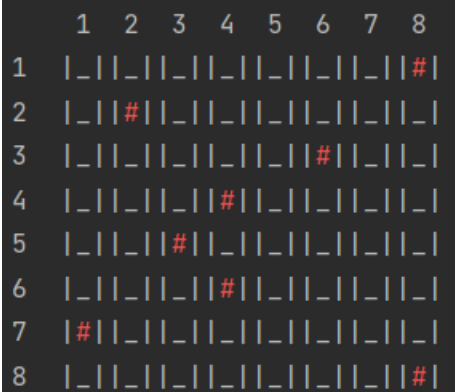
Продовження таблиці 3.1

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
<div> <div>1 2 3 4 5 6 7 8</div> <div>1 _ _ _ _ _ _ _ # _ _ _ _ _ </div> <div>2 _ _ _ _ _ _ _ # _ _ _ _ _ </div> <div>3 # _ _ _ _ _ _ _ _ _ _ _ </div> <div>4 _ _ _ _ _ _ _ _ _ _ _ _ # </div> <div>5 _ _ _ _ # _ _ _ _ _ _ _ _ </div> <div>6 # _ _ _ _ _ _ _ _ _ _ _ </div> <div>7 _ _ # _ _ _ _ _ _ _ _ _ _ </div> <div>8 _ _ _ _ _ _ _ _ # _ _ _ _ _ </div> </div>	8	1 325	1 456	441
<div> <div>1 2 3 4 5 6 7 8</div> <div>1 _ _ _ _ _ _ _ # _ _ _ _ _ </div> <div>2 _ _ _ _ _ _ _ _ _ _ _ # _ </div> <div>3 _ _ # _ _ _ _ _ _ _ _ _ _ </div> <div>4 # _ _ _ _ _ _ _ _ _ _ _ _ </div> <div>5 _ _ _ _ _ _ _ _ # _ _ _ _ _ </div> <div>6 _ _ _ _ _ _ _ _ # _ _ _ _ _ </div> <div>7 _ _ _ _ # _ _ _ _ _ _ _ _ _ </div> <div>8 _ _ _ _ _ _ _ _ # _ _ _ _ _ </div> </div>	8	14 676 814	14 945 840	441
<div> <div>1 2 3 4 5 6 7 8</div> <div>1 _ _ _ _ _ _ _ _ _ _ _ # _ _ _ </div> <div>2 _ _ _ _ # _ _ _ _ _ _ _ _ _ </div> <div>3 _ _ _ _ _ _ _ _ # _ _ _ _ _ _ </div> <div>4 _ _ _ _ _ _ _ _ _ _ _ _ # _ _ _ </div> <div>5 _ _ _ _ _ _ _ _ _ # _ _ _ _ _ _ </div> <div>6 _ _ _ _ _ _ _ _ _ _ _ _ _ # _ _ </div> <div>7 _ _ _ # _ _ _ _ _ _ _ _ _ _ _ _ </div> <div>8 _ _ _ _ _ _ _ _ _ # _ _ _ _ _ _ </div> </div>	8	7 123 447	7 254 020	441

Продовження таблиці 3.1

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
	8	732 493	745 920	441
	8	211 664	215 544	441
	8	11 676 286	11 890 312	441
	8	110 291 055	112 312 684	441

Продовження таблиці 3.1

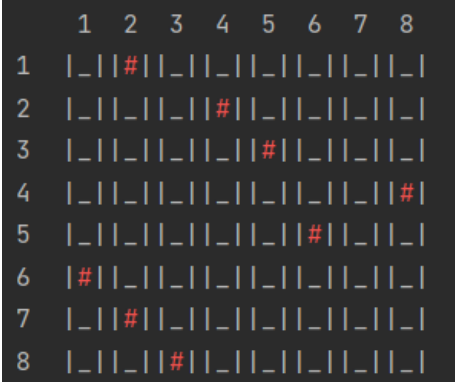
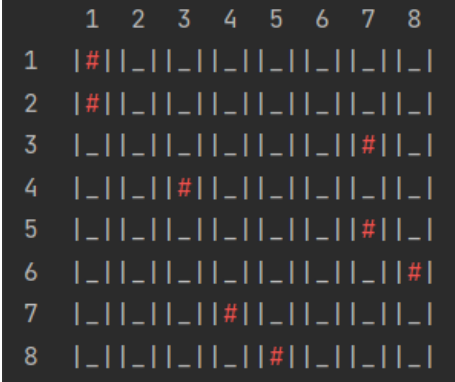
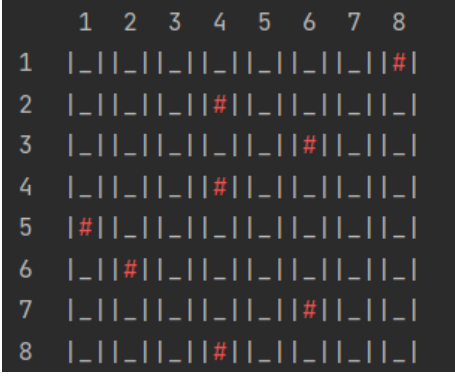
Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
	8	763 563	777 560	441
	8	1 823 754	1 857 184	441
	8	33 293 091	33 903 352	441

Продовження таблиці 3.1

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
	8	96 730	98 504	441
	8	30 793 815	31 358 264	441
	8	770	784	441
	8	732 713	746 144	441

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

Таблиця 3.2 – Характеристики оцінювання A^*

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
	4	0	896	896
	4	0	6 608	6 608
	6	0	1 792	1 792

Продовження таблиці 3.2

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
<pre> 1 2 3 4 5 6 7 8 1 _ _ # _ _ _ _ _ _ _ _ 2 _ _ _ _ _ _ _ # _ _ _ _ 3 _ _ _ _ _ # _ _ _ _ _ _ _ 4 _ _ _ _ _ _ _ _ _ _ # _ _ 5 _ _ _ _ _ _ _ _ _ _ _ # _ 6 _ _ # _ _ _ _ _ _ _ _ _ _ 7 _ _ # _ _ _ _ _ _ _ _ _ _ 8 _ _ _ _ _ _ _ _ _ _ _ _ # </pre>	5	0	11 704	11 704
<pre> 1 2 3 4 5 6 7 8 1 _ _ _ _ _ _ _ _ # _ _ _ _ 2 _ _ _ _ _ _ _ _ _ _ _ _ # 3 _ _ _ _ _ _ _ _ _ _ _ _ # 4 _ _ _ _ # _ _ _ _ _ _ _ _ _ 5 _ _ _ _ _ _ # _ _ _ _ _ _ _ 6 _ _ _ _ _ # _ _ _ _ _ _ _ _ 7 _ _ _ _ _ _ _ _ _ _ _ _ # 8 _ _ _ _ _ _ # _ _ _ _ _ _ _ </pre>	4	0	280	280
<pre> 1 2 3 4 5 6 7 8 1 _ _ _ _ _ _ _ _ _ _ # _ _ 2 _ _ _ _ _ _ _ _ # _ _ _ _ _ 3 # _ _ _ _ _ _ _ _ _ _ _ _ _ 4 _ _ _ _ _ _ _ _ _ _ _ _ _ # 5 _ _ _ _ _ _ _ _ _ _ _ _ _ # 6 _ _ _ _ _ _ # _ _ _ _ _ _ _ _ 7 _ _ # _ _ _ _ _ _ _ _ _ _ _ 8 _ _ _ _ _ _ # _ _ _ _ _ _ _ _ </pre>	4	0	896	896
<pre> 1 2 3 4 5 6 7 8 1 _ _ _ _ _ _ _ _ # _ _ _ _ _ 2 _ _ _ _ # _ _ _ _ _ _ _ _ _ 3 _ _ _ _ _ _ _ _ # _ _ _ _ _ 4 _ _ _ _ _ _ _ _ _ _ _ _ _ # 5 _ _ _ _ _ # _ _ _ _ _ _ _ _ _ 6 _ _ # _ _ _ _ _ _ _ _ _ _ _ 7 # _ _ _ _ _ _ _ _ _ _ _ _ _ 8 _ _ _ _ _ _ _ _ _ _ _ _ _ # </pre>	5	0	29 680	29 680

Продовження таблиці 3.2

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
<div><div><div>1 2 3 4 5 6 7 8</div><div>1 # _ _ _ _ _ _ _ _ _ _ _ </div><div>2 _ _ _ _ _ _ # _ _ _ _ _ _ </div><div>3 _ _ _ _ _ _ _ _ _ _ # _ _ _ </div><div>4 _ _ _ # _ _ _ _ _ _ _ _ _ _ </div><div>5 _ _ _ _ _ _ _ _ _ _ _ _ _ # </div><div>6 _ _ _ _ _ _ _ _ _ _ # _ _ _ _ </div><div>7 _ _ _ _ _ _ _ # _ _ _ _ _ _ _ </div><div>8 _ _ _ # _ _ _ _ _ _ _ _ _ _ </div></div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></</div>				

Продовження таблиці 3.2

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
	5	0	6 832	6 832
	6	0	12 992	12 992
	5	0	20 384	20 384
	4	0	1 792	1 792

Продовження таблиці 3.2

Початкові стани	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
	5	0	3 528	3 528
	4	0	3 696	3 696
	4	0	2 464	2 464

В таблиці 3.3 наведені середні значення характеристик оцінювання алгоритмів LDFS і A*, які були визначені серіями із 20 дослідів (таблиці 3.1-3.2).

Таблиця 3.3 – Середні значення характеристик оцінювання алгоритмів
LDFS і A*

Назва алгоритму	Ітерації	К-сть глухих кутів	Всього станів	Всього станів у пам'яті
LDFS	8	17 140 906	17 455 098	441
A*	4	0	5 709	5 709

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто алгоритм неінформативного пошуку (АНП) – Limited Depth First Search (LDFS), а також алгоритм інформативного пошуку (АПІ) – A^* , з використанням евристичної функції – кількості пар ферзів, які б'ють один одного з урахуванням видимості (функція F1). Алгоритми були розглянуті на прикладі задачі про 8 ферзів. В результаті виконання лабораторної роботи я отримав практичні навички роботи з цими алгоритмами, а саме записав псевдокод алгоритмів, виконав їх програмну реалізацію, а також провів дослідження їх роботи.

Програмна реалізація і псевдокод алгоритму LDFS виконувалась «AS IS», тобто без додаткових модифікацій алгоритму. LDFS працює аналогічно класичному алгоритму DFS (пошук в глибину), але однією важливою відмінністю – з обмеженням максимальної глибини. Тобто вузол з максимально допустимою глибиною не розгортається. Спочатку була спроба тестувати роботу цього алгоритму на «повній задачі» 8-ферзів (ферзі розміщуються довільним чином), однак алгоритм виявився занадто повільним для цієї задачі. Тому для тестування роботи LDFS було вирішено використати спрощений варіант задачі, в якому ферзі розміщені по своїх рядках, і можуть рухатись виключно по ним. Також для алгоритму було обрано максимально глибину 8, тобто кількість ферзів, оскільки будь-яку задачу можна вирішити, якщо поставити кожен з 8 ферзів на правильну клітинку у своєму рядку. Важливо зазначити, що LDFS все одно вирішує деякі «спрощені» задачі достатньо довго. Наприклад, під час одного з тестувань для вирішення задачі LDFS згенерував більше чим 100 000 станів, що зайняло приблизно 26 хвилин.

Програмна реалізація і псевдокод алгоритму A^* виконувались з однієї суттєвою модифікацією: списки closed та open були реалізовані як відсортовані масиви, що забезпечило швидку роботу потрібних операцій (isPresent, extractMin, insert, append). Таким чином A^* може вирішувати у задовільний час спрощену і повну задачу про 8 ферзів, та, навіть, більш загальну задачу про n ферзів, де $n < 15$.

Тестування роботи алгоритмів LDFS і A^* проводилось на спрощеному варіанті задачі про 8 ферзів, щоб забезпечити можливість їх порівняння в однакових умовах. У результаті тестування я зробив висновок, що алгоритм A^* генерує набагато менше станів і працює швидше чим алгоритм LDFS: A^* у середньому генерує 5709 станів, а LDFS – 17455098. Це зумовлено тим, що LDFS – це алгоритм АНП, тобто такий, що просто перебирає вузли, а A^* – це АІП, тобто такий, що використовує додаткову характеристики, щоб обрати наступний вузол для розгортання. Також завдяки використанній цієї характеристики, A^* знаходить кращі розв'язки, які вимагають менше кроків для виконання: A^* у середньому знаходить розв'язок з 4 кроками, а LDFS – з 8 (що дорівнює встановленій максимальній глибині).

У всіх дослідженнях алгоритм A^* ні разу не заходив у «глухий кут», тобто не розгортав вузли без дітей. З іншого боку, кількість глухих кутів у LDFS приблизно дорівнює $T - \frac{T}{b}$, де T – це загальна кількість згенерованих станів, а b – кількість нащадків у вузла. Це зумовлено тим, що майже всі вузли, які розгортає алгоритм LDFS, знаходяться на максимальній глибині, що означає, що вони не мають нащадків.

Єдина перевага LDFS перед A^* – це використання пам'яті: A^* у середньому зберігає 5709, а LDFS – 441. Просторова складність LDFS дорівнює $O(b * L)$, де b – кількість нащадків у вузла, а L – максимальна глибина пошуку. З іншого боку, простора складність A^* дорівнює $O(b^d)$, де d – це глибина найбільш поверхневого рішення, оскільки A^* зберігає кожен розгорнутий вузол у масиві closed, а всю периферію у масиві open.

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%.