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»

“Проектування і аналіз алгоритмів зовнішнього сортування”

Виконав(ла) ІП-12 Волков Вадим Всеволодович 27.10.2022
(шифр, прізвище, ім’я, по батькові)

Перевірив Сопов Олексій Олександрович 27.10.2022
(прізвище, ім’я, по батькові)

Київ 2022

ЗМІСТ

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 відрізнятись початковим станом. Серія повинна містити не менше 20 експериментів для кожного алгоритму. Початковий стан зафіксувати у таблиці експериментів. За проведеними серіями необхідно визначити:

• середню кількість етапів (кроків), які знадобилось для досягнення розв’язку (ітерації);

• середню кількість випадків, коли алгоритм потрапляв в глухий кут (не міг знайти оптимальний розв’язок) – якщо таке можливе;

• середню кількість згенерованих станів під час пошуку;

• середню кількість станів, що зберігаються в пам’яті під час роботи програми.

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

Використані позначення:

• 8-ферзів – Задача про вісім ферзів полягає в такому розміщенні восьми ферзів на шахівниці, що жодна з них не ставить під удар один одного. Тобто, вони не повинні стояти в одній вертикалі, горизонталі чи діагоналі.

• LDFS – Пошук вглиб з обмеженням глибини.

• A* – Пошук A*.

• F2 – кількість пар ферзів, які б’ють один одного без урахування видимості.

№	Задача	АНП	АІП	АЛП	Func
8	8-ферзів	LDFS	A*		F2

3. ВИКОНАННЯ

1. Псевдокод алгоритму

Алгоритм LDFS

```
findPath():
        queensInit = []
        playfield = [[0, 0, 0, ... *8], ...  *8]
        finalPath = []
        solved = false

        addRandomly()

        queens = copy(queensInit)
        for q=0 in queenCount do
                x = queens[q][0]
                y = queens[q][1]
                for i=0 in fieldSize do
                        if not x = i then
                                move(queens, q, i, y, 0, 1)
                        end if
                end for
                for i=0 in fieldSize do
                        if not y = i then
                                move(queens, q, x, i, 0, 2)
                        end if
                end for
                z = y-x
                for i=0 in fieldSize do
                        if not x = i and z+i ≥ 0 and z+i < fieldSize then
                                move(queens, q, i, z+i, 0, 3)
                        end if
                end for
                z = y+x
                for i=0 in fieldSize do
                        if not x = i and z-i ≥ 0 and z-i < fieldSize then
                                move(queens, q, i, z-i, 0, 4)
                        end if
                end for
        end for
        return finalPath
end function

addRandomly():
        add = queenCount
        while add > 0 do
                x = random(0, fieldSize-1)
                y = random(0, fieldSize-1)
                skip = false
                for q=0 in queensInit.length then
                        if queensInit[q][0] = x and queensInit[q][1] = y then
                                skip = true
                        end if
                end for
                if not skip then
                        playfield[y][x] = 1

                        wrong = 0
                        for i=0 in queens2.length do
                                x2 = queens2[i][0]
                                y2 = queens2[i][1]
```

```
                                if x = x2 or y = y2 or x+y = x2+y2 or x-y = x2-
y2 then
                                        queens2[i][2] = queens2[i][2] + 1
                                        wrong = wrong + 1
                                        totalWrong = totalWrong + 1
                                end if
                        end for
                        totalWrong = totalWrong + wrong + 1
                        queensInit.append([x, y, wrong+1])
                        add = add - 1
                end if
        end while
end function

move(queens, queen, x, y, depth, last):
        if depth ≥ maxDepth or solved or playfield[y][x] = 1 then
                return
        end if

        oldx = queens[queen][0]
        oldy = queens[queen][1]
        oldw = queens[queen][2]
        finalPath.append([queen, x, y])
        queens[queen][0] = x
        queens[queen][1] = y
        playfield[y][x] = 1
        playfield[oldy][oldx] = 0

        totalWrong = totalWrong - oldw
        wrong = 0
        for i=0 in queenCount do
                if not i = queen then
                        x2 = queens[i][0]
                        y2 = queens[i][1]
                        if oldx = x2 or oldy = y2 or oldx+oldy = x2+y2 or oldx-
oldy = x2-y2 then
                                queens[i][2] = queens[i][2] - 1
                                totalWrong = totalWrong - 1
                        end if
                        if x = x2 or y = y2 or x+y = x2+y2 or x-y = x2-y2 then
                                queens[i][2] = queens[i][2] + 1
                                wrong = wrong + 1
                                totalWrong = totalWrong + 1
                        end if
                end if
        end for
        queens[queen][2] = wrong+1
        totalWrong += wrong+1
        if totalWrong = queenCount then
                solved = true
                return
        end if

        for q=0 in queenCount then
                x = queens[q][0]
                y = queens[q][1]
                if not q = queen or not last = 1 then
                        for i=0 in fieldSize do
                                if not x = i then
                                        move(queens, q, i, y, depth + 1, 1)
                                end if
                        end for
                end if
                if not q = queen or not last = 2 then
```

```
                        for i=0 in fieldSize do
                                if not y = i then
                                        move(queens, q, x, i, depth + 1, 2)
                                end if
                        end for
                end if
                if not q = queen or not last = 3 then
                        z = y-x
                        for i=0 in fieldSize do
                                if not x = i and z+i ≥ 0 and z+i < fieldSize
then
                                        move(queens, q, i, z+i, depth + 1, 3)
                                end if
                        end for
                end if
                if not q = queen or not last = 4 then
                        z = y+x
                        for i=0 in fieldSize do
                                if not x = i and z-i ≥ 0 and z-i < fieldSize
then
                                        move(queens, q, i, z-i, depth + 1, 4)
                                end if
                        end for
                end if
        end for

        if solved then
                return
        end if

        queens[queen][0] = oldx
        queens[queen][1] = oldy
        queens[queen][2] = oldw
        playfield[y][x] = 0
        playfield[oldy][oldx] = 1
        totalWrong = totalWrong - wrong + 1 + oldw
        for i=0 in queenCount do
                if not i = queen then
                        x2 = queens[i][0]
                        y2 = queens[i][1]
                        if oldx = x2 or oldy = y2 or oldx+oldy = x2+y2 or oldx-
oldy = x2-y2 then
                                queens[i][2] = queens[i][2] + 1
                                totalWrong = totalWrong + 1
                        end if
                        if x = x2 or y = y2 or x+y = x2+y2 or x-y = x2-y2 then
                                queens[i][2] = queens[i][2] - 1
                                totalWrong = totalWrong - 1
                        end if
                end if
        end for
        log.popFromEnd()
end function
```

Алгоритм A*

```
find():
        addRandomly()
        solve()

        states = []
        action = finalState.whatIDid
        finalPath = []
        while action isn't null do
```

```
                finalPath.insertAtStart([action[1], action[2], action[3]])
                action = action[4]
        end while
        return finalPath
end function

class State:
        depth = 0
        hitfield = []
        actions = []
        queens = []
        whatIDid = null

        # Перевірка, чи є цей стан вирішенням
        isSolution():
                hitfield = this.hitfield
                for q=0 to queenCount do
                        x = this.queens[q][0]
                        y = this.queens[q][1]
                        if hitfield[y][x] > 1 then
                                return false
                        end if
                end for
                return true
        end function

        # Генерування повного списку можливих дій
        # та сортування їх за евристичою функцією
        genActions():
                actions = []
                hitfield = this.hitfield
                for q=0 in queenCount do
                        x = this.queens[q][0]
                        y = this.queens[q][1]
                        for i=0 in fieldSize do
                                if not x = i then
                                        actions.append([hitfield[y][i], q, i,
y])
                                end if
                        end for
                        for i=0 in fieldSize do
                                if not y = i then
                                        actions.append([hitfield[i][x], q, x,
i])
                                end if
                        end for
                        z = y-x
                        for i=0 in fieldSize do
                                if not x = i and z+i ≥ 0 and z+i < fieldSize
then
                                        actions.append([hitfield[z+i][i], q, i,
z+i])
                                end if
                        end for
                        z = y+x
                        for i=0 in fieldSize do
                                if not x = i and z-i ≥ 0 and z-i < fieldSize
then
                                        actions.append([hitfield[z-i][i], q, i,
z-i])
                                end if
                        end for
                end for
                actions.sortBy0thElement()
```

```
                this.actions = actions
        end function

        # Створення похідного стану з найкращої дії
        # Видалення себе коли всі дії оброблено
        makeChild(myIndex):
                action = this.actions.popFromStart()
                newState = new State()
                newState.depth = this.depth + 1
                newState.whatIDid = action
                action.append(this.whatIDid)
                newState.queens = copy(this.queens)
                newState.hitfield = copy(this.hitfield)
                addHitfield(newState.hitfield, newState.queens[action[1]][0],
newState.queens[action[1]][1], -1)
                newState.queens[action[1]][0] = action[2]
                newState.queens[action[1]][1] = action[3]
                addHitfield(newState.hitfield, action[2], action[3], 1)

                if newState.isSolution() then
                        finalState = newState
                        return true
                end if

                newState.genActions()
                states.append(newState)
                if(this.actions.length == 0) states.splice(myIndex, 1)
                return false
        end function
end class

# Вирішення
solve():
        forever do
                minIdx = null
                minVal = Infinity
                for i=0 in states.length do
                        state = states[i]
                        val = state.depth + state.actions[0][0]
                        if val < minVal then
                                minVal = val
                                minIdx = i
                        end if
                end for
                if states[minIdx].makeChild(minIdx) then
                        return true
                end if
        end forever
end function

# Заповнення початкового стану
addRandomly():
        newState = new State()
        queens = newState.queens
        hitfield = [[0, 0, 0, ... *8], ... *8]
        newState.hitfield = hitfield
        add = queenCount

        while add > 0 do
                x = random(0, fieldSize-1)
                y = random(0, fieldSize-1)
                skip = false
                for q=0 in queens.length do
                        if queens[q][0] = x && queens[q][1] = y then
```

```
                                skip = true
                        end if
                end for

                if not skip then
                        addHitfield(hitfield, x, y, 1)
                        queens.append([x, y])
                        add = add - 1
                end if
        end while

        newState.genActions()
        states.append(newState)
end function

# Функція, що додає вказане число до всіх клітинок
# по вертикалі, горизонталі та діагоналі від вказаної
# за допомогою x y, у переданому масиві
addHitfield(hitfield, x, y, val):
        hitfield[y][x] += val
        for i=0 in fieldSize do
                if not x = i then
                        hitfield[y][i] = hitfield[y][i] + val
                end if
        end for
        for i=0 in fieldSize do
                if not y = i then
                        hitfield[i][x] = hitfield[i][x] + val
                end if
        end for
        z = y - x
        for i=0 in fieldSize do
                if not x = i and z+i ≥ 0 and z+i < fieldSize then
                        hitfield[z+i][i] = hitfield[z+i][i] + val
                end if
        end for
        z = y + x
        for i=0 in fieldSize do
                if not x = i and z-i ≥ 0 and z-i < fieldSize then
                        hitfield[z-i][i] = hitfield[z-i][i] + val
                end if
        end for
end function
```

2. Програмна реалізація алгоритму

    1. Вихідний код

LDFS html

```
<!doctype html>
<html>
        <head>
                <title>Lab2</title>
                <meta charset="UTF-8">
                <meta name="viewport" content="width=device-width, initial-scale=1.0">
                <link href="style.css" rel="stylesheet">
                <script src="main.js" defer></script>
        </head>
        <body>
                <label for="board">Розмір дошки</label><input type="number" id="board" value="8"
min="2" max="8"><br>
                <label for="queen">Кількість ферзів</label><input type="number" id="queen" value="8"
min="2" max="8"><br>
                <label for="depth">Обмеження глибини</label><input type="number" id="depth"
value="10" min="1" max="99"><br>
```

```
                <label for="delay">Уповільнення</label><input type="number" id="delay" value="0"
min="0" max="10"><br>
                <button id="start">Пуск</button><span id="info"></span><br>
                <canvas width=300 height=300 id="canv"></canvas>
        </body>
</html>
```

LDFS javascript

```
const canvas = document.getElementById("canv");
const ctx = canvas.getContext("2d");
const queen = new Image(); queen.src = "../queen.png";
let queens2 = [];
let playfield = new Array(8).fill(0).map(array => new Array(8).fill(0));
let log = [];
let maxDepth = 6;
let totalWrong = 0;
let queenCount = 8;
let fieldSize = 8;
let delay = 0;
let solved = false;
document.getElementById("start").addEventListener("click", async function() {
        document.getElementById("info").innerText = "";
        queenCount = Math.max(document.getElementById("queen").value | 0, 2);
        fieldSize = Math.max(document.getElementById("board").value | 0, 2);
        maxDepth = Math.max(document.getElementById("depth").value | 0, 1);
        delay = Math.max(document.getElementById("delay").value, 0);
        canvas.width = canvas.height = 44 + fieldSize * 32;
        queens2 = [];
        playfield = new Array(8).fill(0).map(array => new Array(8).fill(0));
        log = [];
        solved = false;
        addRandomly();
        draw(queens2);
        await wait(0.5);
        document.getElementById("info").innerText = "Пошук шляху...";
        await waitFrame();
        let startTime = Date.now();
        let queens = copy(queens2);
        for(let q=0; q<queenCount; q++) {
                let x = queens[q][0];
                let y = queens[q][1];
                for(let i=0; i<fieldSize; i++) {
                        if(x !== i) await move(queens, q, i, y, 0, 1);
                }
                for(let i=0; i<fieldSize; i++) {
                        if(y !== i) await move(queens, q, x, i, 0, 2);
                }
                let z = y-x;
                for(let i=0; i<fieldSize; i++) {
                        if(x !== i && z+i >= 0 && z+i < fieldSize) await
move(queens, q, i, z+i, 0, 3);
                }
                z = y+x;
                for(let i=0; i<fieldSize; i++) {
                        if(x !== i && z-i >= 0 && z-i < fieldSize) await
move(queens, q, i, z-i, 0, 4);
                }
        }
        if(solved) {
                document.getElementById("info").innerText = "Шлях знайдено за "+
((Date.now() - startTime) / 1000)+" секунд";
                await playAnimation();
        } else {
```

```
                document.getElementById("info").innerText = "Шлях не знайдено";
        }
});
function draw(queens) {
        queens = queens.slice().sort((a,b) => a[1]-b[1]);
        ctx.clearRect(0, 0, 300, 300);
        ctx.fillStyle = "black";
        ctx.fillRect(20, 20, fieldSize*32+4, fieldSize*32+4);
        ctx.font = "20px sans-serif";
        const colors = ["white", "black"];
        for(let y=0; y<fieldSize; y++) {
                for(let x=0; x<fieldSize; x++) {
                        ctx.fillStyle = colors[(x+y) % 2];
                        ctx.fillRect(22+x*32, 22+y*32, 32, 32);
                }
        }
        for(let q=0; q<queenCount; q++) {
                let x = queens[q][0];
                let y = queens[q][1];
                ctx.drawImage(queen, 22+x*32, 22+y*32);
                if(delay > 0) {
                        let l = ctx.measureText(queens[q][2]).width/2;
                        ctx.fillStyle = "#000000";
                        ctx.fillText(queens[q][2], 38-l+x*32, 46+y*32);
                        ctx.fillStyle = "#ff0000";
                        ctx.fillText(queens[q][2], 38-l+x*32, 44+y*32);
                }
        }
        ctx.fillStyle = "black";
        let fs32 = fieldSize*32;
        for(let y=0; y<fieldSize; y++) {
                ctx.fillText("12345678"[y], 4, 12+fs32-y*32);
                ctx.fillText("12345678"[y], 31+fs32, 12+fs32-y*32);
                ctx.fillText("ABCDEFGH"[y], 33+y*32, 14);
                ctx.fillText("ABCDEFGH"[y], 33+y*32, 44+fs32);
        }
}
function addRandomly() {
        let add = queenCount;
        while(add > 0) {
                let x = random(0, fieldSize-1);
                let y = random(0, fieldSize-1);
                let skip = false;
                for(let q=0; q<queens2.length; q++) {
                        if(queens2[q][0] === x && queens2[q][1] === y) skip =
true;
                }
                if(skip) continue;
                playfield[y][x] = 1;
                let wrong = 0;
                for(let i=0; i<queens2.length; i++) {
                        let x2 = queens2[i][0];
                        let y2 = queens2[i][1];
                        if(x === x2 || y === y2 || x+y === x2+y2 || x-y === x2-
y2) {
                                queens2[i][2]++;
                                wrong++;
                                totalWrong++;
                        }
                }
                totalWrong += wrong+1;
                queens2.push([x, y, wrong+1]);
                add--;
        }
```

```
}
function random(a, b) {
	return a + Math.floor(Math.random() * (b-a+1));
}
async function move(queens, queen, x, y, depth, last) {
	if(depth >= maxDepth || solved) return;
	if(playfield[y][x] === 1) return;
	let oldx = queens[queen][0];
	let oldy = queens[queen][1];
	let oldw = queens[queen][2];
	log.push([queen, x, y]);
	queens[queen][0] = x;
	queens[queen][1] = y;
	playfield[y][x] = 1;
	playfield[oldy][oldx] = 0;
	totalWrong -= oldw;
	let wrong = 0;
	for(let i=0; i<queenCount; i++) {
		if(i == queen) continue;
		let x2 = queens[i][0];
		let y2 = queens[i][1];
		if(oldx === x2 || oldy === y2 || oldx+oldy === x2+y2 || oldx-
oldy === x2-y2) {
			queens[i][2]--;
			totalWrong--;
		}
		if(x === x2 || y === y2 || x+y === x2+y2 || x-y === x2-y2) {
			queens[i][2]++;
			wrong++;
			totalWrong++;
		}
	}
	queens[queen][2] = wrong+1;
	totalWrong += wrong+1;
	if(totalWrong == queenCount) {
		solved = true;
		return;
	}
	if(delay > 0) {
		draw(queens);
		await wait(delay);
	}
	depth++;
	for(let q=0; q<queenCount; q++) {
		let x = queens[q][0];
		let y = queens[q][1];
		if(q !== queen || last !== 1) {
			for(let i=0; i<fieldSize; i++) {
				if(x !== i) await move(queens, q, i, y, depth,
1);
			}
		}
		if(q !== queen || last !== 2) {
			for(let i=0; i<fieldSize; i++) {
				if(y !== i) await move(queens, q, x, i, depth,
2);
			}
		}
		if(q !== queen || last !== 3) {
			let z = y-x;
			for(let i=0; i<fieldSize; i++) {
				if(x !== i && z+i >= 0 && z+i < fieldSize) await
move(queens, q, i, z+i, depth, 3);
			}
```

```
                }
                if(q !== queen || last !== 4) {
                        let z = y+x;
                        for(let i=0; i<fieldSize; i++) {
                                if(x !== i && z-i >= 0 && z-i < fieldSize) await
move(queens, q, i, z-i, depth, 4);
                        }
                }
        }
        if(solved) return;
        queens[queen][0] = oldx;
        queens[queen][1] = oldy;
        queens[queen][2] = oldw;
        playfield[y][x] = 0;
        playfield[oldy][oldx] = 1;
        totalWrong -= wrong+1;
        totalWrong += oldw;
        for(let i=0; i<queenCount; i++) {
                if(i == queen) continue;
                let x2 = queens[i][0];
                let y2 = queens[i][1];
                if(oldx === x2 || oldy === y2 || oldx+oldy === x2+y2 || oldx-
oldy === x2-y2) {
                        queens[i][2]++;
                        totalWrong++;
                }
                if(x === x2 || y === y2 || x+y === x2+y2 || x-y === x2-y2) {
                        queens[i][2]--;
                        totalWrong--;
                }
        }
        log.pop();
}
function copy(arr) {
        return arr.map(e => e.slice());
}
async function wait(seconds) {
        return new Promise((resolve, reject) => setTimeout(resolve, seconds *
1000));
}
async function waitFrame() {
        return new Promise((resolve, reject) => requestAnimationFrame(resolve));
}
async function playAnimation() {
        console.log("animation");
        for(let step=0; step<log.length; step++) {
                let q = log[step][0];
                let oldx = queens2[q][0];
                let oldy = queens2[q][1];
                let newx = log[step][1];
                let newy = log[step][2];
                for(let t=0; t<=1; t+=0.01) {
                        let t2 = t * t * (1.5 - t) * 2;
                        queens2[q][0] = oldx + (newx - oldx) * t2;
                        queens2[q][1] = oldy + (newy - oldy) * t2 - t * (1-t);
                        draw(queens2);
                        await waitFrame();
                }
        }
}
```

A* html

```
<!doctype html>
<html>
	<head>
		<title>Lab2</title>
		<meta charset="UTF-8">
		<meta name="viewport" content="width=device-width, initial-
scale=1.0">
		<script src="main.js" defer></script>
	</head>
	<body>
		<label for="board">Розмір дошки</label><input type="number"
id="board" value="8" min="2" max="8"><br>
		<label for="queen">Кількість ферзів</label><input type="number"
id="queen" value="8" min="2" max="8"><br>
		<button id="start">Пуск</button><span id="info"></span><br>
		<canvas width=300 height=300 id="canv"></canvas>
	</body>
</html>
```

A* javascript

```
const canvas = document.getElementById("canv");
const ctx = canvas.getContext("2d");
const queen = new Image(); queen.src = "../queen.png";
let states = [];
let finalState = null;
let queenCount = 8;
let fieldSize = 8;
document.getElementById("start").addEventListener("click", async function() {
	document.getElementById("info").innerText = "";
	queenCount = Math.max(document.getElementById("queen").value | 0, 2);
	fieldSize = Math.max(document.getElementById("board").value | 0, 2);
	canvas.width = canvas.height = 44 + fieldSize * 32;
	addRandomly();
	let queensInitial = states[0].queens;
	draw(queensInitial);
	await wait(0.5);
	document.getElementById("info").innerText = "Пошук шляху...";
	let startTime = Date.now();
	await solve();
	document.getElementById("info").innerText = "Шлях знайдено за "+
((Date.now() - startTime) / 1000)+" секунд";
	states = [];
	let action = finalState.whatIDid;
	let log = [];
	while(action) {
		log.unshift([action[1], action[2], action[3]]);
		action = action[4];
	}
	await playAnimation(queensInitial, log);
});
class State {
	depth = 0;
	hitfield = [];
	actions = [];
	queens = [];
	whatIDid = null;
	isSolution() {
		let hitfield = this.hitfield;
		for(let q=0; q<queenCount; q++) {
			let x = this.queens[q][0];
			let y = this.queens[q][1];
			if(hitfield[y*8+x] > 1) return false;
```

```
                }
                return true;
        }
        genActions() {
                let actions = [];
                let hitfield = this.hitfield;
                for(let q=0; q<queenCount; q++) {
                        let x = this.queens[q][0];
                        let y = this.queens[q][1];
                        for(let i=0; i<fieldSize; i++) {
                                if(x !== i) actions.push([hitfield[y*8+i], q, i,
y]);
                        }
                        for(let i=0; i<fieldSize; i++) {
                                if(y !== i) actions.push([hitfield[i*8+x], q, x,
i]);
                        }
                        let z = y-x;
                        for(let i=0; i<fieldSize; i++) {
                                if(x !== i && z+i >= 0 && z+i < fieldSize)
actions.push([hitfield[(z+i)*8+i], q, i, z+i]);
                        }
                        z = y+x;
                        for(let i=0; i<fieldSize; i++) {
                                if(x !== i && z-i >= 0 && z-i < fieldSize)
actions.push([hitfield[(z-i)*8+i], q, i, z-i]);
                        }
                }
                actions.sort((a,b) => a[0]-b[0]);
                this.actions = actions;
        }
        makeChild(myIndex) {
                let action = this.actions.shift();
                let newState = new State();
                newState.depth = this.depth + 1;
                newState.whatIDid = action;
                action.push(this.whatIDid);
                newState.queens = copy(this.queens);
                newState.hitfield = this.hitfield.slice();
                addHitfield(newState.hitfield, newState.queens[action[1]][0],
newState.queens[action[1]][1], -1);
                newState.queens[action[1]][0] = action[2];
                newState.queens[action[1]][1] = action[3];
                addHitfield(newState.hitfield, action[2], action[3], 1);
                if(newState.isSolution()) {
                        console.log(newState);
                        finalState = newState;
                        return true;
                }
                newState.genActions();
                states.push(newState);
                if(this.actions.length === 0) states.splice(myIndex, 1);
        }
}
async function solve() {
        let delay = 0;
        while(true) {
                let minIdx = null;
                let minVal = Infinity;
                for(let i=0; i<states.length; i++) {
                        let state = states[i];
                        let val = state.depth + state.actions[0][0];
                        if(val < minVal) {
                                minVal = val;
```

```
                                minIdx = i;
                        }
                }
                if(states[minIdx].makeChild(minIdx)) return true;
                if(++delay > 1000) {
                        delay = 0;
                        await wait(0.001);
                }
        }
}
function draw(queens) {
        queens = queens.slice().sort((a,b) => a[1]-b[1]);
        ctx.clearRect(0, 0, 300, 300);
        ctx.fillStyle = "black";
        ctx.fillRect(20, 20, fieldSize*32+4, fieldSize*32+4);
        ctx.font = "20px sans-serif";
        const colors = ["white", "black"];
        for(let y=0; y<fieldSize; y++) {
                for(let x=0; x<fieldSize; x++) {
                        ctx.fillStyle = colors[(x+y) % 2];
                        ctx.fillRect(22+x*32, 22+y*32, 32, 32);
                }
        }
        for(let q=0; q<queenCount; q++) {
                let x = queens[q][0];
                let y = queens[q][1];
                ctx.drawImage(queen, 22+x*32, 22+y*32);
        }
        ctx.fillStyle = "black";
        let fs32 = fieldSize*32;
        for(let y=0; y<fieldSize; y++) {
                ctx.fillText("12345678"[y], 4, 12+fs32-y*32);
                ctx.fillText("12345678"[y], 31+fs32, 12+fs32-y*32);
                ctx.fillText("ABCDEFGH"[y], 33+y*32, 14);
                ctx.fillText("ABCDEFGH"[y], 33+y*32, 44+fs32);
        }
}
function addRandomly() {
        let newState = new State();
        let queens = newState.queens;
        let hitfield = newState.hitfield = new Array(8*8).fill(0);
        let add = queenCount;
        while(add > 0) {
                let x = random(0, fieldSize-1);
                let y = random(0, fieldSize-1);
                let skip = false;
                for(let q=0; q<queens.length; q++) {
                        if(queens[q][0] === x && queens[q][1] === y) skip =
true;
                }
                if(skip) continue;
                addHitfield(hitfield, x, y, 1);
                queens.push([x, y]);
                add--;
        }
        draw(queens);
        newState.genActions();
        states.push(newState);
}
function addHitfield(hitfield, x, y, val) {
        hitfield[y*8+x] += val;
        for(let i=0; i<fieldSize; i++) {
                if(x !== i) hitfield[y*8+i] += val;
        }
```

```
        for(let i=0; i<fieldSize; i++) {
                if(y !== i) hitfield[i*8+x] += val;
        }
        let z = y-x;
        for(let i=0; i<fieldSize; i++) {
                if(x !== i && z+i >= 0 && z+i < fieldSize) hitfield[(z+i)*8+i]
+= val;
        }
        z = y+x;
        for(let i=0; i<fieldSize; i++) {
                if(x !== i && z-i >= 0 && z-i < fieldSize) hitfield[(z-i)*8+i]
+= val;
        }
}
function random(a, b) {
        return a + Math.floor(Math.random() * (b-a+1));
}
function copy(arr) {
        return arr.map(e => e.slice());
}
async function wait(seconds) {
        return new Promise((resolve, reject) => setTimeout(resolve, seconds *
1000));
}
async function waitFrame() {
        return new Promise((resolve, reject) => requestAnimationFrame(resolve));
}
async function playAnimation(queens, log) {
        console.log("animation");
        for(let step=0; step<log.length; step++) {
                let q = log[step][0];
                let oldx = queens[q][0];
                let oldy = queens[q][1];
                let newx = log[step][1];
                let newy = log[step][2];
                for(let t=0; t<=1; t+=0.01) {
                        let t2 = t * t * (1.5 - t) * 2;
                        queens[q][0] = oldx + (newx - oldx) * t2;
                        queens[q][1] = oldy + (newy - oldy) * t2 - t * (1-t);
                        draw(queens);
                        await waitFrame();
                }
        }
}
```

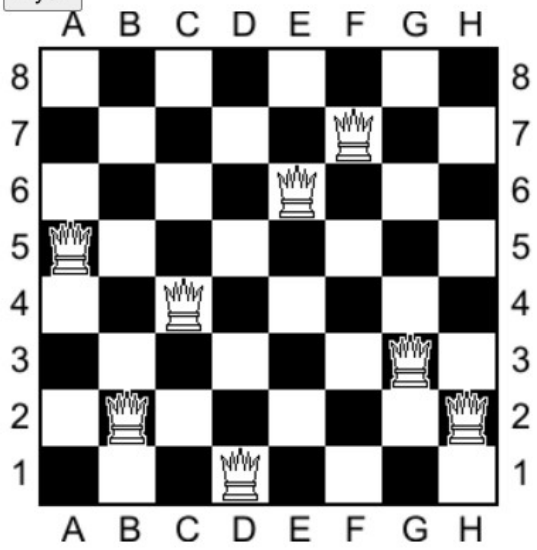
2. Виконання

LDFS з

LDFS до

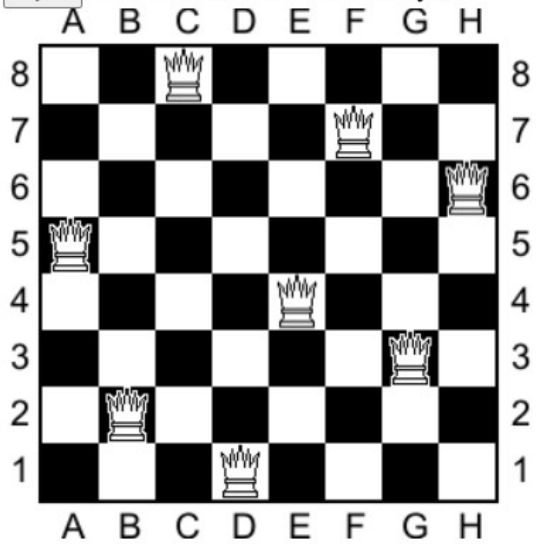
A* від

Розмір дошки 8
Кількість ферзів 8
Пуск



A* до

Розмір дошки 8
Кількість ферзів 8
Пуск Шлях знайдено за 0.048 секунд



3. Дослідження алгоритмів

	LDFS	A*
Повний	ні якщо L<d	так
Час	$O(b^l)$	
Простір	O(bL)	
Оптимальний	ні якщо L>d	
Основна ідея (глобально)	Цей алгоритм це модифікація DFS з доданим лімітом. Алгоритм DFS передбачає якнайшвидше заглиблення до кінцевих вузлів, їх обробку та повернення на 1 рівень вгору тільки коли всі гілки були оброблені.	Цей алгоритм на кожному кроці обирає для обробки вузол, що має найменшу суму пройденого шляху до нього та евристичної функції, що зазвичай є відстанню до цілі.
Основна ідея (тут)	На кожному кроці можна здвинути один з 8ми ферзів на буль-яку допустиму кількість клітинок до краю у одному з 8ми напрямів. Для знаходження шляху відбувається перебір всіх можливих комбінацій ходів на кожному з кроків до знаходження першого вирішення. Кількість кроків дорівнює ліміту.	На кожному кроці можна здвинути один з 8ми ферзів на буль-яку допустиму кількість клітинок до краю у одному з 8ми напрямів. Програма зберігає всі розгорнуті стани і з них на кожному кроці обирає який наступний стан розгорнути обираючи дію з мінімальною сумою кількості видимих ферзів з клітинки та глибини стану.

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

Таблиця 3.1

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пам’яті
Стан 1	391463	383469	178462987637760	1
Стан 2	133086	130364	178462987637760	1
Стан 3	7371714	7221266	178462987637760	1

Стан 4	2648796	2594734	178462987637760	1
Стан 5	1758680	1722784	178462987637760	1
Стан 6	368432	360907	178462987637760	1
Стан 7	1574401	1542265	178462987637760	1
Стан 8	903473	885030	178462987637760	1
Стан 9	848833	831504	178462987637760	1
Стан 10	81334	79669	178462987637760	1
Стан 11	1989270	1948668	178462987637760	1
Стан 12	2285688	2239036	178462987637760	1
Стан 13	491950	481905	178462987637760	1
Стан 14	285406	279576	178462987637760	1
Стан 15	639011	625965	178462987637760	1
Стан 16	248740	243658	178462987637760	1
Стан 17	47921	46938	178462987637760	1
Стан 18	391002	383017	178462987637760	1
Стан 19	188076	184233	178462987637760	1
Стан 20	925029	906146	178462987637760	1

В таблиці 3.1 наведені характеристики оцінювання алгоритму A* задачі 8-ферзів для 20 початкових станів.

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пам’яті
Стан 1	6587	0	168077	3119
Стан 2	6958	0	13013	239
Стан 3	15605	0	207793	3865
Стан 4	11599	0	264481	4930
Стан 5	8857	0	218167	4054
Стан 6	7197	0	178163	3310
Стан 7	13618	0	310622	5790
Стан 8	24230	0	246148	4585
Стан 9	413	0	14035	258
Стан 10	45530	0	16035	295
Стан 11	52619	0	178271	3310
Стан 12	53178	0	18481	340
Стан 13	53593	0	13753	253
Стан 14	62005	0	206292	3834

Стан 15	45502	0	225006	4187
Стан 16	70838	0	11741	216
Стан 17	80033	0	224269	4169
Стан 18	89453	0	225556	4196
Стан 19	154742	0	230082	4281
Стан 20	137219	0	231501	4306

4. ВИСНОВОК

При виконанні лабораторної роботи було ознайомлено з різними алгоритмами пошуку шляху, а також детально розглянуто неінформативний алгоритм пошуку в глибину з обмеженням глибини LDFS та інформативний алгоритм пошуку A*. Для них був спроектований псевдокод, створена імплементація на мові програмування Javascript та були проведені заміри обох алгоритмів. В результаті було виявлено, що випадку проблеми 8 ферзів, алгоритм LDFS, не дивлячись на те, що він іноді може знайти вирішення дуже швидко, через те що він перебирає всі можливі комбінації кроків у багатьох випадках займає занадто довго (у одному з випробувань реалізація алгоритму на мові C не знайшла шлях за 3 години). У порівнянні з ним, A* працює швидше і може знайти шлях за адекватну кількість часу для більшої кількості початкових станів, але в обмін на це витрачає багато оперативної пам’яті. Це відбувається тому, що в саме цьому випадку шлях знаходиться для всіх ферзів одночасно, де відвідана погана для певного ферза клітинка може стати гарною після здвигу якогось з інших ферзів, що унеможливлює відмічання відвіданих клітинок, як це робиться майже завжди при використанні A*. Також приходиться зберігати всі стани з розстановками ферзів окремо. В результаті можна сказати, що імплементації не одного з двох алгоритмів не могли знайти шлях для 8ми ферзів на 8x8 дошці за 1 годину для всіх початкових розстановок у 100% випадках, хоча вирішення завжди існує. Для 6ми ферзів на 6x6 дошці обидва алгоритми працюють добре.

5. КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку 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%.