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 Ткач Владичлав* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Ахаладзе І. Е.* _____
(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 Вихідний код 10

3.2.2 Приклади роботи 11

3.3 Дослідження алгоритмів 14

ВИСНОВОК 20

КРИТЕРІЇ ОЦІНЮВАННЯ 21

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.

- **8-puzzle** – гра, що складається з 8 однакових квадратних пластинок з нанесеними числами від 1 до 8. Пластинки поміщаються в квадратну коробку, довжина сторони якої в три рази більша довжини сторони пластинок, відповідно в коробці залишається незаповненим одне квадратне поле. Мета гри – переміщаючи пластинки по коробці досягти впорядковування їх по номерах, бажано зробивши якомога менше переміщень.
- **Лабіринт** – задача пошуку шляху у довільному лабіринті від початкової точки до кінцевої з можливими випадками відсутності шляху. Структура лабіринту зчитується з файлу, або генерується програмою.
- **LDFS** – Пошук вглиб з обмеженням глибини.
- **BFS** – Пошук вшир.
- **IDS** – Пошук вглиб з ітеративним заглибленням.
- **A*** – Пошук A*.
- **RBFS** – Рекурсивний пошук за першим найкращим співпадінням.
- **F1** – кількість пар ферзів, які б’ють один одного з урахуванням видимості (ферзь A може стояти на одній лінії з ферзем B, проте між ними стоїть ферзь C; тому A не б’є B).
- **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·t, де k – змінний коефіцієнт.
- **BEAM** – Локальний променевий пошук. Робоча характеристика – кількість променів k. Експерименти проводи із кількістю променів від 2 до 21.
- **MRV** – евристика мінімальної кількості значень;
- **DGR** – ступенева евристика.

Таблиця 2.1 – Варіанти алгоритмів

№	Задача	АНП	АІП	АЛП	Func
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

BFS

```
Function FindSolutoion(problem)
    Queue.Enqueue(problem)
    while usedMemory < 1024:
        currentState = prioretyQueue.Dequeue()
        if currentState is Solved:
            return currentState
        end if

        for i = 1, i < 5:
            child = stepManager.MakeStep(curentState, Step(i))
            if child exists:
                prioretyQueue.Enqueue(child)
            end if
        end for
    end while
```

A*

```
Function FindSolutoion(problem)
    prioretyQueue.Enqueue(problem, statePtiorety.CulcStatePriorety(problem))
    while usedMemory < 1024:
        currentState = prioretyQueue.Dequeue()
        if currentState is Solved:
            return currentState
        end if

        for i = 1, i < 5:
            child = stepManager.MakeStep(curentState, Step(i))
            if child exists:
                prioretyQueue.Enqueue(child,
            statePtiorety.CulcStatePriorety(child))
            end if
        end for
    end while
```

3.2 Програмна реалізація

3.2.1 Вихідний код

BFS

```
public Node? FindSolution(Node? problem)
{
    StepManager stepManager = new StepManager();
    PazzleChecker pazzleChecker = new PazzleChecker();

    Queue<Node?> queue = new Queue<Node?>();
    queue.Enqueue(problem);
    while (GC.GetTotalMemory(false) / 1048576 < 1024)
    {
        Node? currentState = queue.Dequeue();
        if (pazzleChecker.IsSolved(currentState.State))
            return currentState;

        for (int i = 1; i < 5; i++)
        {
            Node? child = stepManager.MakeStep(currentState, (Step)i);
            if (child != null)
            {
                queue.Enqueue(child);
            }
        }
    }
    return null;
}
```

```
public Node? MakeStep(Node? node, Step step)
{
    StepMaker stepMaker = new StepMaker();
    int sizeX = node.State.GetLength(1) - 1;
    int sizeY = node.State.GetLength(0) - 1;
    switch (step)
    {
        case Step.Up when node.ForbiddenStep != Step.Up && node.Zero.X != 0:
            return stepMaker.NewStep(node, -1, 0, Step.Down);
        case Step.Down when node.ForbiddenStep != Step.Down && node.Zero.X != sizeX:
            return stepMaker.NewStep(node, 1, 0, Step.Up);
        case Step.Left when node.ForbiddenStep != Step.Left && node.Zero.Y != 0:
            return stepMaker.NewStep(node, 0, -1, Step.Right);
        case Step.Right when node.ForbiddenStep != Step.Right && node.Zero.Y != sizeY:
            return stepMaker.NewStep(node, 0, 1, Step.Left);
    }
    return null;
}
```

```
public enum Step
{
    None,
    Up,
    Down,
    Left,
    Right
}
```

```
public Node? NewStep(Node? node, int deltaX, int deltaY, Step forbiddenStep)
{
    int[,] newState = (int[,])node.State.Clone() ;
    newState[node.Zero.X, node.Zero.Y] = newState[node.Zero.X + deltaX, node.Zero.Y +
deltaY];
    newState[node.Zero.X + deltaX, node.Zero.Y + deltaY] = 0;

    Node? newNode = new Node(newState, forbiddenStep, node.Zero.X + deltaX, node.Zero.Y
+ deltaY, node.Level + 1);
    newNode.Predecessor = node;
    return newNode;
}
```

A*

```
public Node? FindSolution(Node? problem)
{
    StatePriorety statePriorety = new StatePriorety();
    StepManager stepManager = new StepManager();
    PazzleChecker pazzleChecker = new PazzleChecker();

    PriorityQueue<Node?, int> priorityQueue = new PriorityQueue<Node?, int>();
    priorityQueue.Enqueue(problem, statePriorety.CulcStatePrioretty(problem));
    while (GC.GetTotalMemory(false) / 1048576 < 1024)
    {
        Node? currentState = priorityQueue.Dequeue();
        if (pazzleChecker.IsSolved(currentState.State))
            return currentState;

        for (int i = 1; i < 5; i++)
        {
            Node? child = stepManager.MakeStep(currentState, (Step)i);
            if (child != null)
            {
                priorityQueue.Enqueue(child, statePriorety.CulcStatePrioretty(child));
            }
        }
    }
    return null;
}
```

3.2.2 Приклади роботи

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

Рисунок 3.1 – Алгоритм BFS

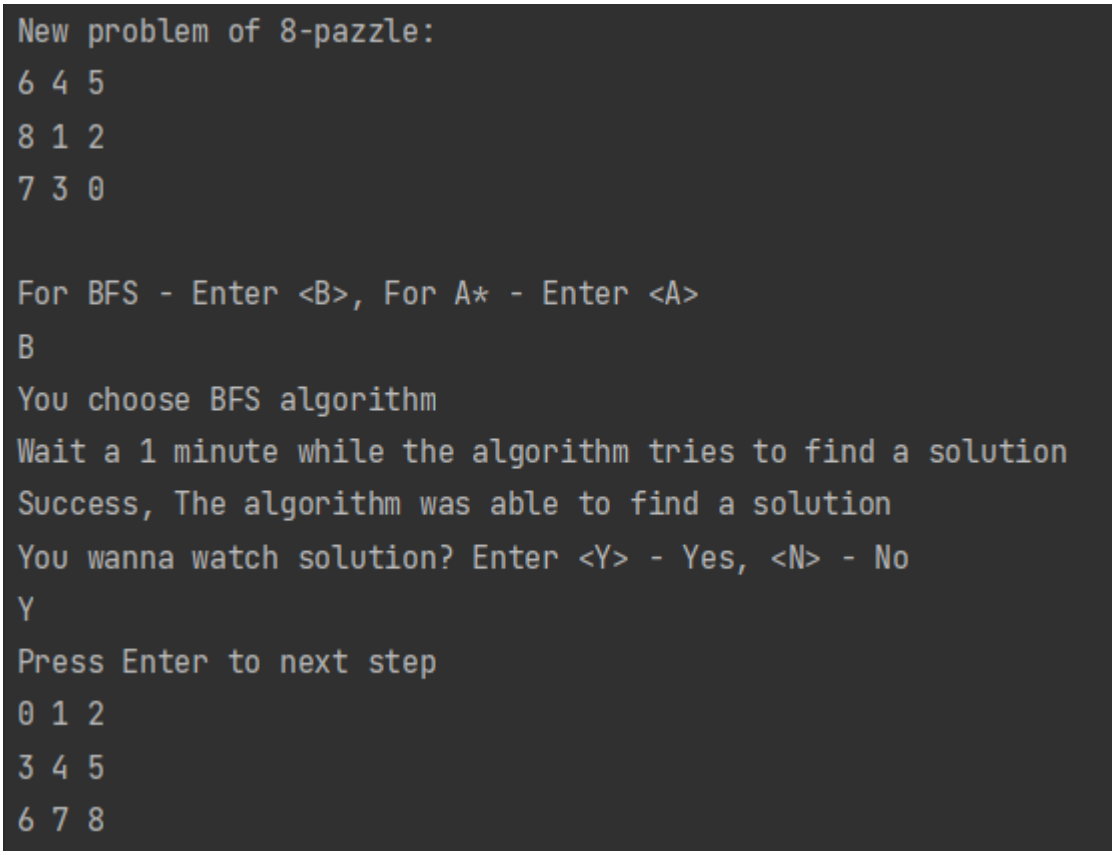

Рисунок 3.2 – Алгоритм A*

```
New problem of 8-pazzle:
6 3 7
4 5 8
2 0 1

For BFS - Enter <B>, For A* - Enter <A>
A
You choose A* algorithm
Wait a 1 minute while the algorithm tries to find a solution
Success, The algorithm was able to find a solution
You wanna watch solution? Enter <Y> - Yes, <N> - No
Y
Press Enter to next step
0 1 2
3 4 5
6 7 8
```

3.3 Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму BFS задачі 8-Pazzle для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму BFS

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пом’яті
4 8 5 3 1 6 7 0 2	373000	0	631296	631296
6 5 1 8 2 3 7 0 4	913768	0	1581198	1581198
3 4 8 1 5 2 6 7 0	373777	0	665563	665563
5 8 4 3 2 0 6 7 1	271638	0	477142	477142
3 7 6 4 5 0 1 8 2	789128	0	1397990	1397990
8 0 2 1 6 4 7 3 5	119920	0	202788	202788
4 0 8 5 7 3 2 6 1	38183004	0	6366128	6366128

1 7 8 0 3 5 6 4 2	403139	0	675771	675771
2 3 0 1 8 6 5 7 4	1725946	0	2904828	2904828
1 5 3 6 2 0 8 4 7	29807	0	52341	52341
7 0 4 1 5 3 6 8 2	104300	0	179450	179450
3 4 5 0 6 8 2 1 7	94419	0	165605	165605
6 2 3 5 4 8 1 7 0	1151458	0	2049350	2049350
0 4 3 8 1 5 2 7 6	3598562	0	6348858	6348858
6 5 1 4 8 3 7 0 2	106288	0	182466	182466
2 1 0 4 3 7 5 6 8	568874	0	961240	961240

0 3 5 8 2 7 1 4 6	61886	0	104940	104940
5 1 2 3 0 8 8 4 6	63456	0	111118	111118
3 5 7 2 0 8 6 4 1	274351	0	460051	460051
4 6 3 2 8 7 5 0 1	2850308	0	4917616	4917616

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

Таблиця 3.2 – Характеристики оцінювання алгоритму A*

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пом’яті
3 4 0 5 2 7 8 6 1	3501	0	5953	5953
2 0 5 8 3 6 7 1 4	811	0	1372	1372
5 7 0 2 6 8 3 4 1	3361	0	5530	5530
7 4 5 2 8 3 0 6 1	2324	0	3961	3961
5 8 2 7 0 1 4 3 6	8111	0	13408	13408
2 7 0 6 5 3 8 1 4	265	0	443	443
7 2 8 5 6 1 3 0 4	2384	0	3996	3996
8 7 3 4 2 6 1 5 0	749	0	1251	1251

2 5 8 4 6 1 7 0 3	641	0	1090	1090
3 4 7 2 0 6 8 5 1	11145	0	18698	18698
8 3 2 4 0 1 5 7 6	1879	0	3116	3116
8 2 0 6 4 3 5 1 7	749	0	1257	1257
5 4 6 3 0 8 7 2 1	566	0	954	954
5 2 4 3 7 8 1 6 0	1337	0	2244	2244
4 1 6 8 3 5 0 2 7	12535	0	20990	20990
7 1 2 5 8 4 3 6 0	818	0	1394	1394
8 1 0 6 7 3 4 5 2	4324	0	7172	7172

7 0 2 4 6 8 1 5 3	167	0	283	283
7 8 5 6 4 0 2 1 3	7720	0	13018	13018
3 7 1 0 8 5 4 2 6	135	0	227	227

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто та досліджено алгоритм неінформативного пошуку BFS та алгоритм інформативного пошуку A* (для визначення пріоритетності використовувалась - Манхетенська відстань) для задачі 8-pazzl. У результаті було створено програмне забезпечення для відображення роботи алгоритмів із можливістю перегляду рішення – покроковий перегляд кожної перестановки, за умови знаходження рішення відповідним алгоритмом, враховуючи обмеження використаної пам’яті у 1Гб.

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%.