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 Кондрацька Соня _____
(шифр, прізвище, ім'я, по батькові)

Перевірив Головченко М.М. _____
(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.
- **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(initial_state):

```
q = Queue()
q.push(initial_state)
while ( !q.empty() ):
    if time >= 1800:
        return False   curr_
    state = q.pop()
    if curr_state.is_solution == True:
        return curr_state
    curr_depth = curr_state.get_depth()
    if curr_depth != n:
        for i from 0 to n:
            new_state = State(curr_state, curr_depth)
            new_state.move(curr_depth, i)
            q.push(new_state)
```

A*(initial_state):

```
q = PriorityQueue()
q.push(initial_state)
while ( !q.empty() ):
    if time >= 1800:
        return False
    curr_state = q.pop()
    if curr_state.is_solution == True:
        return curr_state
    curr_depth = curr_state.get_depth()
```

```
        if curr_depth != n:
            for i from 0 to n:
                new_state = State(curr_state, curr_depth)
                new_state.move(curr_depth, i)
                q.push(new_state)
```

is_solution():

```
    for i from 0 to n - 1:
        for j from i +1 to n :
            dist = j –i
            if queens[i] == queens[j] | | queens[i] == queens[j] – dist | |
            queens[i] == queens[j] + dist:
                return False  return True
```

F1():

```
    num_of _pairs = 0
    for i from 0 to n - 1:
        is_visible_row = True
        is_visible_d1 = True
        is_visible_d2 = True
        for j from i +1 to n :
            dist = j –i
            if queens[i] == queens[j] && is_visible_row == True:
                num_of_pairs += 1
                is_visible_row = False
            if queens[i] == queens[j] – dist && is_visible_d1== True:
                num_of_pairs += 1
                is_visible_d1 = False
            if queens[i] == queens[j] + dist && is_visible_d2 == True:
                num_of_pairs += 1
                is_visible_d2 = False
```

return num_of_pairs

priority(): return F1() + depth

3.2 Програмна реалізація

3.2.1 Вихідний код

```
        def BFS_solution(self, initial_state):
start_time = time()
q = Queue()
q.put(initial_state)


iterations = 0
total_states = 1
states_in_mem = 1


while not q.empty():


    if time() - start_time >= 1800:
        print('Time limit!')
        return False


    iterations += 1
    curr_state = q.get()
    states_in_mem -= 1


    if curr_state.is_solution():
        return curr_state, iterations, total_states, states_in_mem, time() - start_time


    curr_depth = curr_state.get_depth()
    if curr_depth != self.__n:
        for i in range(self.__n):
            new_state = State(curr_state.get_queens(), curr_depth + 1)
            new_state.move(curr_depth, i)  # queens[row] = col
            q.put(new_state)
            total_states += 1
            states_in_mem += 1
```

```
        def A_star_solution(self, initial_state):
start_time = time()
q = PriorityQueue()
q.put(initial_state)


iterations = 0
total_states = 1
states_in_mem = 1


while not q.empty():


    if time() - start_time >= 1800:
        return False


    iterations += 1
    curr_state = q.get()
    states_in_mem -= 1


    if curr_state.is_solution():
        return curr_state, iterations, total_states, states_in_mem, time() - start_time


    curr_depth = curr_state.get_depth()
    if curr_depth != self.__n:
        for i in range(self.__n):
            new_state = State(curr_state.get_queens(), curr_depth + 1)
            new_state.move(curr_depth, i)  # queens[row] = col
            q.put(new_state)
            total_states += 1
            states_in_mem += 1
```

```
        def __f1_heuristic(self):
num_of_pairs = 0


for i in range(len(self._queens) - 1):
    is_visible_row = True   # visibility in row
    is_visible_d1 = True    # visibility in left diagonal
    is_visible_d2 = True    # visibility in left diagonal
```

```
        for j in range(i + 1, len(self._queens)):
            dist = j - i
            if self._queens[i] == self._queens[j] and is_visible_row:
                num_of_pairs += 1
                is_visible_row = False
            if self._queens[i] == self._queens[j] - dist and is_visible_d1:
                num_of_pairs += 1
                is_visible_d1 = False
            if self._queens[i] == self._queens[j] + dist and is_visible_d2:
                num_of_pairs += 1
                is_visible_d2 = False
    return num_of_pairs


def __priority(self):
    return self.__f1_heuristic() + self._depth
```

3.2.2 Приклади роботи

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

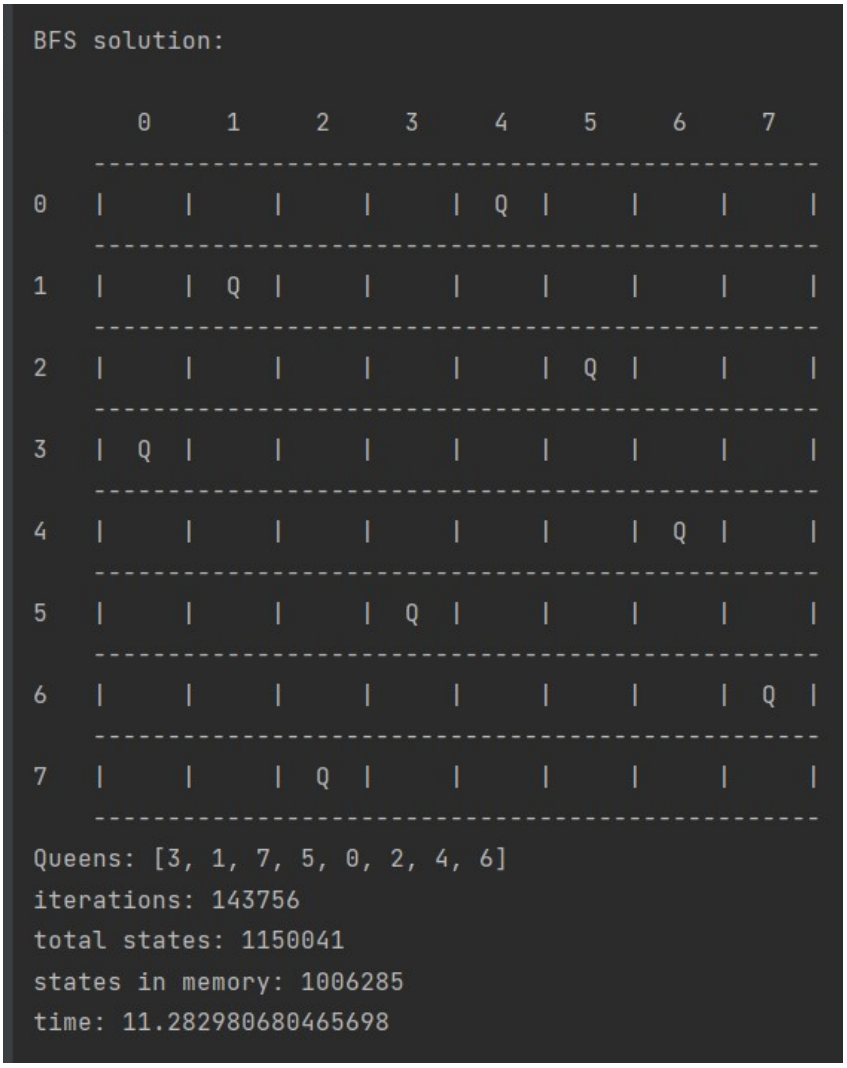

Рисунок 3.1 – Алгоритм BFS для 8ми ферзів

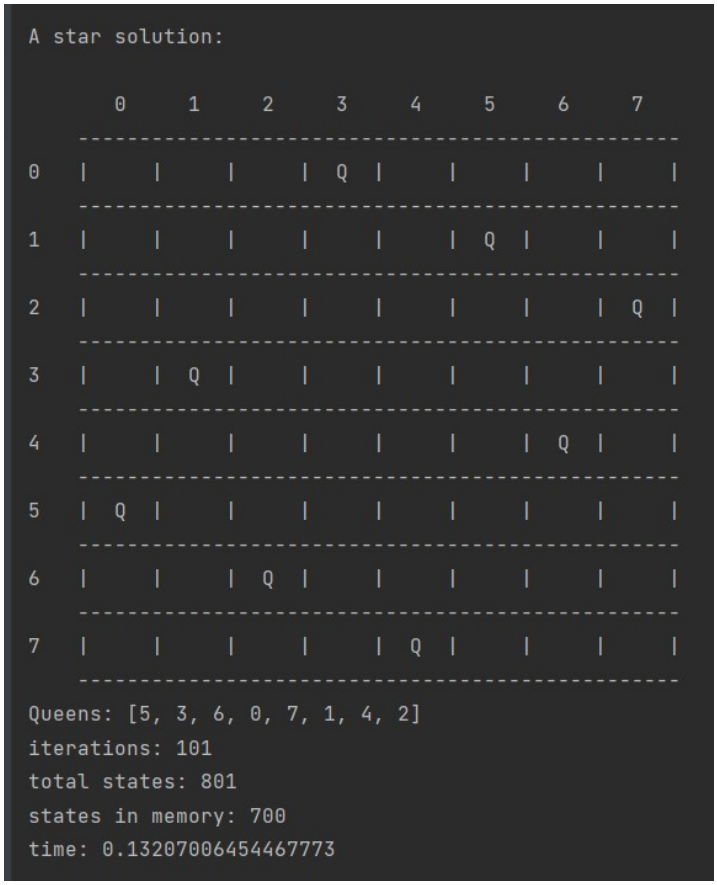

Рисунок 3.2 – Алгоритм A*

3.3 Дослідження алгоритмів

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

Таблиця 3.1 – Характеристики оцінювання BFS

Початкові стани	Ітерації	Всього станів	Всього станів у пом’яті
[1,1,1,1,1,1,1,1]	1000496	8003961	7003465
[0,0,0,0,0,0,0,0]	962984	7703865	6740881
[2,2,2,2,2,2,2,2]	511510	4092073	3580563
[3,3,3,3,3,3,3,3]	462075	3696593	3234518
[4,4,4,4,4,4,4,4]	493539	3948305	3454766
[5,5,5,5,5,5,5,5]	850965	6807713	5956748
[6,6,6,6,6,6,6,6]	1139084	9112665	7973581
[7,7,7,7,7,7,7,7]	1299852	10398809	9098957
[0,1,2,3,4,5,6,7]	1299852	10398809	9098957
[7,6,5,4,3,2,1,0]	962984	7703865	6740881
[0,0,0,0,1,1,1,1]	1000496	8003961	7003465
[0,0,1,1,2,2,3,3]	462075	3696593	3234518
[4,4,5,5,6,6,7,7]	1299852	10398809	9098957
[2,2,2,2,3,3,3,3]	462075	3696593	3234518
[4,4,4,4,5,5,5,5]	850965	6807713	5956748
[6,6,6,6,7,7,7,7]	1299852	10398809	9098957
[0,0,0,0,7,7,7,7]	1299852	10398809	9098957
[0,7,1,6,2,5,3,4]	493539	3948305	3454766
[1,1,1,1,1,3,4,5]	850965	6807713	5956748
[5,4,3,1,1,1,1,1]	1000496	8003961	7003465

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

Таблиця 3.3 – Характеристики оцінювання A*

Початкові стани	Ітерації	Всього станів	Всього станів у пом'яті
[1,1,1,1,1,1,1,1]	283	2209	1926
[0,0,0,0,0,0,0,0]	79	625	546
[2,2,2,2,2,2,2,2]	129	1009	880
[3,3,3,3,3,3,3,3]	385	2937	2552
[4,4,4,4,4,4,4,4]	237	1769	1532
[5,5,5,5,5,5,5,5]	279	2057	1778
[6,6,6,6,6,6,6,6]	198	1553	1355
[7,7,7,7,7,7,7,7]	522	4089	3567
[0,1,2,3,4,5,6,7]	1626	10529	8903
[7,6,5,4,3,2,1,0]	769	5561	4792
[0,0,0,0,1,1,1,1]	30	233	203
[0,0,1,1,2,2,3,3]	169	1281	1112
[4,4,5,5,6,6,7,7]	208	1585	1377
[2,2,2,2,3,3,3,3]	290	2161	1871
[4,4,4,4,5,5,5,5]	110	873	763
[6,6,6,6,7,7,7,7]	46	361	315
[0,0,0,0,7,7,7,7]	84	665	581
[0,7,1,6,2,5,3,4]	800	6393	5593
[1,1,1,1,1,3,4,5]	466	3657	3191
[5,4,3,1,1,1,1,1]	116	921	805

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто один алгоритм неінформативного пошуку (BFS) та один алгоритм інформативного пошуку (A*). Проведено аналіз ефективності роботи цих алгоритмів для 20ти станів задачі 8ми ферзів з використанням евристичої функції 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)).

Як видно з таблиць характеристик оцінювання, іформативний пошук виявився в декілька разів швидшим при будя-яких вхідних даних.

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%.