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-15Мочалов Дмитро Юрійович* _____
(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 9

3.2.1 Вихідний код 9

3.2.2 Приклади роботи 13

3.3 Дослідження алгоритмів 14

ВИСНОВОК 22

КРИТЕРІЇ ОЦІНЮВАННЯ 23

МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – розглянути та дослідити алгоритми неінформативного, інформативного та локального пошуку. Провести порівняльний аналіз ефективності використання алгоритмів.

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

ВИКОНАННЯ

1.1 Псевдокод алгоритмів

LDFS

```
Function Depth-Limited-Search(State,limit){
  return Recursive-DLS(State,limit)
}
Functions Recursive-DLS(State,limit){
  if State equal GoalState:
    return State
  end if
  if State.Depth > limit:
    return cutoff
  end if
  cutoff_occured = false
  generate_children_for_current_state(State)
  foreach child in children:
    result = Recursive-DLS(child,limit)
    if result = cutoff:
      cutoff_occured = true
    else if result != failure:
      return result
    end if
  end foreach
  if cutoff_occured:
    return cutoff
  else
    return failure
  end if
}
```

AStar

```
Function AStar(State){
  OpenSet = []
  ClosedSet = []
  OpenSet.Append(State)
  while OpenSet is not empty:
    current = remove and return state from openset with lowest F
    ClosedSet.Append(current)
    if current equal GoalState:
     return current
    end if
    generate_children_for_current_state(current)
    foreach child in clidren:
      if child is not in ClosedSet:
        OpenSet.Append(child)
      end if
    end foreach

  // if not found
  return failure
}
```

1.2 Програмна реалізація

1.2.1 Вихідний код

LDFS

```
public static State? Solve(State state, int limit)
{
    State? result = RecursiveLDFS(state, limit);
    if (result != null)
    {
        return result;
    }
    return null;
}

private static State? RecursiveLDFS(State state, int limit)
{
    if (state.Board.isEqual(FunctionsAndConstants.goalState))
    {
        return state;
    }
    if (state.SearchDepth > limit)
    {
        return null;
    }


    var children = FunctionsAndConstants.GenerateChildren(state);
    foreach (var child in children)
    {
        // ResultLDFS result = RecursiveLDFS(child, limit);
        State result = RecursiveLDFS(child, limit);
        if (result != null)
        {
            return result;
        }
    }
    return null;
}
```

AStar

```
public static State Solve(State state)
{
    DateTime data1 = DateTime.Now;
    var OpenList = new PriorityQueue<State, int>();
    var ClosedList = new List<State>();
    OpenList.Enqueue(state,state.F);
    while (OpenList.Count != 0)
    {
        var current = OpenList.Dequeue();
        ClosedList.Add(current);
        if (current.Board.isEqual(FunctionsAndConstants.goalState))
        {
            return current;
        }

        var children = FunctionsAndConstants.GenerateChildren(current);
        foreach (var child in children)
        {
            if (!ClosedList.Contains(child))
            {
                OpenList.Enqueue(child,child.F);
            }
        }
        long memoryUsed =
Process.GetCurrentProcess().PrivateMemorySize64/1000000000;
        if (memoryUsed > 1)
        {
            Console.WriteLine("Memory used is out of available");
            return null;
        }
        DateTime data2 = DateTime.Now;
        if ((data2 - data1).Minutes > 30)
        {
            Console.WriteLine("Timeout");
            return null;
        }
    }

    return null;
}
```

```
public static List<State> GenerateChildren(State state)
{
    (int x, int y) = state.Board.IndexOfBlank();
    var children = new List<State>();

    var rightState = state.MoveBlankToRight(x, y);
    if (rightState != null)
    {
        rightState.LastMove = "right";
        rightState.Parent = state;
        rightState.SearchDepth++;
        rightState.F = rightState.GetF();
        children.Add(rightState);
    }

    var leftState = state.MoveBlankToLeft(x, y);
    if (leftState != null)
    {
        leftState.LastMove = "left";
        leftState.Parent = state;
        leftState.SearchDepth++;
        leftState.F = leftState.GetF();
        children.Add(leftState);
    }

    var downState = state.MoveBlankToDown(x, y);
    if (downState != null)
    {
        downState.LastMove = "down";
        downState.Parent = state;
        downState.SearchDepth++;
        downState.F = downState.GetF();
        children.Add(downState);
    }

    var upState = state.MoveBlankToUp(x, y);
    if (upState != null)
    {
        upState.LastMove = "up";
        upState.Parent = state;
        upState.SearchDepth++;
        upState.F = upState.GetF();
        children.Add(upState);
    }
```

```
    return children;
}
```

1.2.2 Приклади роботи

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

Рисунок 3.1 – Алгоритм LDFS

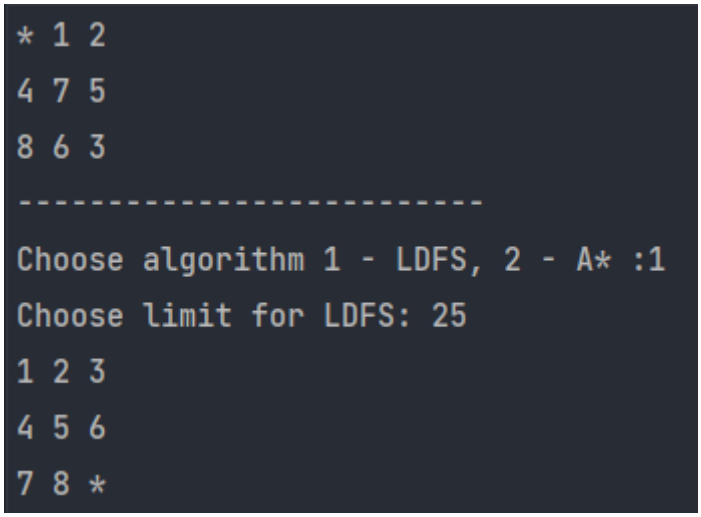
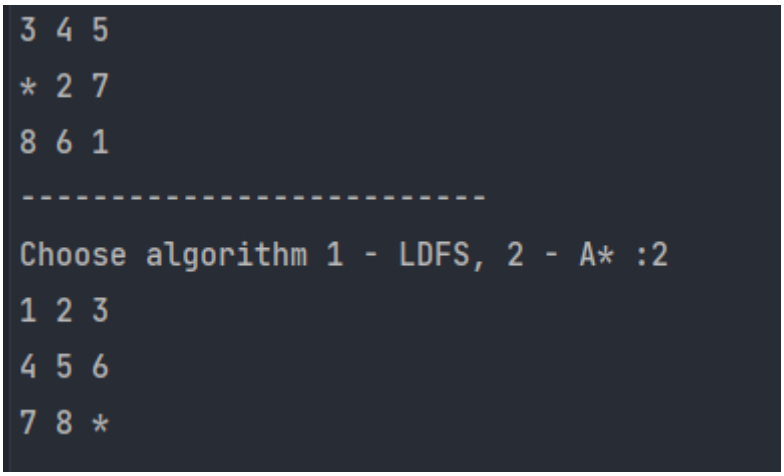


Рисунок 3.2 – Алгоритм AStar



1.3 Дослідження алгоритмів

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

Таблиця 3.1 – Характеристики оцінювання LDFS

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пом’яті
Стан 1 4 6 2 1 7 3 5 * 8 Limit = 22	128431	100066	128442	21
Стан 2 5 1 4 7 * 8 2 3 6 Limit = 22	851941	759104	851941	0
Стан 3 7 6 8 2 4 5 * 3 1 Limit = 22	636907	565232	636907	0
Стан 4 8 4 6 * 2 3 5 7 1 Limit = 22	708585	568304	708585	0
Стан 5 * 2 6 8 3 7 4 1 5 Limit = 22	636907	565232	636907	0
Стан 6	851941	759104	851941	0

8 4 7 1 * 3 5 2 6 Limit = 22				
Стан 7 5 1 6 * 7 4 2 3 8 Limit = 22	708585	568304	708585	0
Стан 8 5 6 8 2 7 * 3 4 1 Limit = 22	708585	568304	708585	0
Стан 9 * 5 4 7 1 3 2 6 8 Limit = 22	636907	565232	636907	0
Стан 10 2 7 * 1 6 8 3 5 4 Limit = 22	636907	565232	636907	0
Стан 11 4 2 5 * 6 1 3 8 7 Limit = 22	708585	568304	708585	0

Стан 12 8 6 1 * 2 3 7 5 4 Limit = 22	708585	568304	708585	0
Стан 13 7 6 * 3 2 4 8 5 1 Limit = 22	636907	565232	636907	0
Стан 14 1 8 6 4 3 7 * 5 2 Limit = 25	1708661	1360934	1708676	24
Стан 15 6 7 * 5 2 8 1 3 4 Limit = 25	148718	121382	148718	24
Стан 16 3 6 8 5 4 1 * 2 7 Limit = 25	752799	605080	752807	22
Стан 17 2 1 7 8 6 3 4 5 *	1369802	1090300	1369812	20

Limit = 25				
Стан 18 6 4 5 7 1 2 3 * 8 Limit = 25	1231551	1096356	1231565	25
Стан 19 5 1 4 8 7 * 6 2 3 Limit = 25	3825553	3399584	3825553	0
Стан 20 2 7 * 6 5 3 4 1 8	3188643	2539448	3188643	0

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

Таблиця 3.3 – Характеристики оцінювання A*

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пом’яті
Стан 1 1 4 * 8 5 3 7 2 6	2617	0	4555	18
Стан 2 6 2 3 8 4 5 * 1 7	39878	0	68319	0
Стан 3 6 4 8 * 5 1 7 3 2	28834	0	50116	0
Стан 4 7 4 6 3 8 1 2 * 5	28833	0	50652	0
Стан 5 * 3 1 7 4 2 8 6 5	1336	0	2317	18
Стан 6 5 4 1 3 2 6 * 7 8	545	0	938	16
Стан 7 3 6 8	664	0	1156	19

* 7 4 2 1 5				
Стан 8 1 5 6 7 8 3 4 * 2	16662	0	29032	23
Стан 9 * 4 8 2 5 1 6 3 7	38168	0	66624	0
Стан 10 1 4 2 7 6 5 3 * 8	6585	0	11431	21
Стан 11 2 7 8 * 6 5 3 1 4	28958	0	50780	0
Стан 12 5 4 8 3 2 7 1 * 6	904	0	1579	18
Стан 13 5 4 2 7 3 8 * 6 1	38733	0	66292	0
Стан 14 4 2 8 7 5 1	28472	0	49125	0

3 * 6				
Стан 15 3 1 7 2 6 * 5 8 4	5371	0	9224	23
Стан 16 6 3 5 7 4 2 1 8 *	6239	0	10895	22
Стан 17 4 2 7 * 3 8 6 5 1	34906	0	60684	0
Стан 18 1 * 2 7 8 5 3 4 6	312	0	537	17
Стан 19 6 * 8 3 1 7 5 2 4	1473	0	2455	21
Стан 20 1 2 7 4 5 3 6 8 *	4989	0	8636	22

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто алгоритми неінформативного пошуку “Пошук в глибину з обмеженням глибини” та алгоритм інформативного пошуку A*. Як можна побачити з експериментів пошук в глибину з обмеженням не завжди може дати результат із-за самого обмеження в глибині.Також оскільки A* не розглядає всі розвітлення, а лише найкращі за екристикою, тому в ній немає глухих кутів, а також меньше ітерацій та станів.

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%.