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-22 Іщенко К. В* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Ахаладзе І.Е.* _____
(прізвище, ім'я, по батькові)

Київ 2023

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: $T = 1000 - k \cdot t$, 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	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

LDFS

```
Function DepthLimitedSearch(problem, start, limit):
    Initialize stackNode and push start node onto it
    While stackNode is not empty:
      currentNode <- top node of stackNode

      If problem's heuristic for currentNode's state is 0:
        Mark currentNode as solution
        Return currentNode

      If currentNode does not have children and its depth is less than limit:
        child <- create children of currentNode
        Push all children onto stackNode

      Else if stackNode size is more than 1:
        Pop stackNode

      Else if stackNode size is 1:
        Pop stackNode

    Return start node
  End Function
```

RBFS(n, B)

1. if n is a goal
2. solution ← n;
3. return n;
4. C ← expand(n)

5. if C is empty, return ∞

6. for each child n_i in C

7. if f(n) < F(n) then $F(n_i) \leftarrow \max(F(n), f(n_i))$

8. else $F(n_i) \leftarrow f(n_i)$

9. (n1, n2) ← bestF(C)

10. while ($F(n_1) \leq B$ and $F(n_1) < \infty$)

11. $F(n_1) \leftarrow$ RBFS(n1, min(B, $F(n_2)$))

12. (n_1, n_2) ← bestF(C)

13. return $F(n_1)$

3.2 Програмна реалізація

3.2.1 Вихідний код

LDFS

```
Node* LDFS::DepthLimitedSearch(ProblemLDFS* problem, Node* start, int limit) {
    stackNode.push(start);
    totalStates++;
    statesInMemory++;

    while (!stackNode.empty()) {
        iterationCount++;
        Node* currentNode = stackNode.top();

        if (problem->heuristics(currentNode->getState()) == 0) {
            currentNode->setIsSolution(true);
            solutionNode = currentNode;
            return currentNode;
        }

        if ((currentNode->getHaveChild() == false) && (currentNode->getLevelOfDeep() < limit)) {
            vector <Node*> child = problem->createChild(currentNode);
```

```
            totalStates += problem->getCounter();
            statesInMemory += problem->getCounter();
            addInStack(child);
            problem->addParentState(currentNode->getState());
            currentNode->setHaveChild(true);
        }
        else if (stackNode.size() > 1) {
            stackNode.pop();
            deadEndCount++;
            statesInMemory--;
            delete currentNode;

        }
        else if (stackNode.size() == 1) {
            stackNode.pop();
        }
    }

    return start;
}
```

RBFS

```
#include <vector>
#include <algorithm>
using namespace std;
int RBFS::RecursiveBestFirstSearch(NodeHeuristic* node, ProblemHeuristic* problem, int B)
{
    iterationCount++;
    bool findBetter = false;
    vector<NodeHeuristic*> successors;
    if (problem->heuristics(node->getState()) == 0) {
        if (!successors.empty()) {
            for (auto* ptr : successors) {
                delete ptr;
            }
        }
        solutionFound = true;
        solutionNode = node;
        return node->getBestF();
```

```
        }
        successors = problem->createChild(node);
        allNode.insert(allNode.end(), successors.begin(), successors.end());
        totalStates += successors.size();
        NodeHeuristic* best = successors[0];
        if (successors.empty()){
            deadEndCount++;
            return INFINITY;
        }
        for (auto child : successors) {
            if (node->getCurrentF() < node->getBestF()) {
                child->setBestF(max(node->getBestF(), child->getCurrentF()));
                findBetter = true;
            }
            else {
                child->setBestF(child->getCurrentF());
            }
        }
        if (findBetter) statesInMemory -= successors.size() - 2;
        statesInMemory += successors.size();
        sortChild(&successors);

        while (successors[0]->getBestF() <= B && successors[0]->getBestF() < INFINITY && !solutionFound) {
            successors[0]->setBestF( RecursiveBestFirstSearch(successors[0], problem, min(B, successors[1]->getBestF())));
            if (!solutionFound) {
                sortChild(&successors);
            }
            else {
                return successors[0]->getBestF();
            }
        }

        return successors[0]->getBestF();
    }
```

3.2.2 Приклади роботи

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

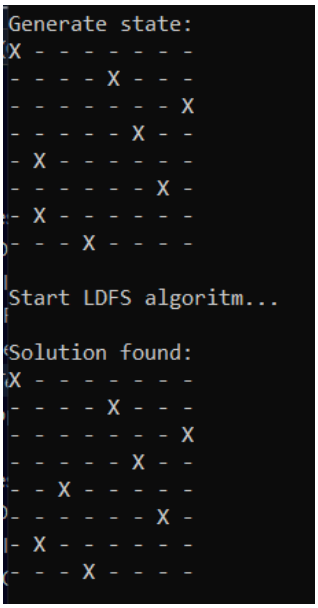

Рисунок 3.1 – Алгоритм LDFS

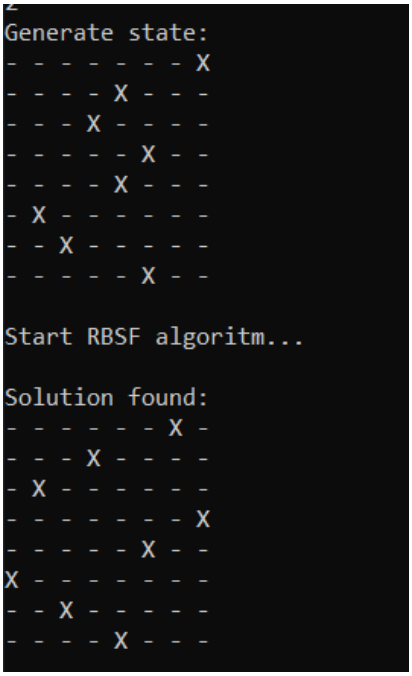

Рисунок 3.2 – Алгоритм RBFS

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 алгоритму LDFS

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пам'яті
Стан 1	23419	22558	10401611	10401411
Стан 2	54753	52740	24318690	24318585
Стан 3	38754	37329	17212692	17212587
Стан 4	24634	23728	10941256	10941151
Стан 5	42131	40582	18712595	18712490
Стан 6	38032	36634	16892014	16891909
Стан 7	58470	56320	25969606	25969501
Стан 8	30618	29492	13599066	13598961
Стан 9	24103	23217	10705411	10705306
Стан 10	27901	26875	12392303	12392198
Стан 11	58272	56130	25881663	25881558
Стан 12	27629	26613	12271494	12271389
Стан 13	48893	47095	21715956	21715851
Стан 14	23791	22916	10566836	10566731
Стан 15	52045	50132	23115925	23115820
Стан 16	35767	34452	15886008	15885903
Стан 17	44995	43341	19984649	19984544
Стан 18	53963	51979	23967810	23967705
Стан 19	29911	28811	13285050	13284945
Стан 20	36149	34820	16055674	16055569

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

Таблиця 3.2 – Характеристики оцінювання алгоритму RBFS

Початкові стани	Ітерації	К-сть гл. кутів	Всього станів	Всього станів у пам’яті
Стан 1	1593	0	42542	40663
Стан 2	5462	0	149777	141344
Стан 3	82418	0	2273627	2166863
Стан 4	31095	0	824430	795105
Стан 5	66097	0	1793516	1711116
Стан 6	17234	0	432778	405663
Стан 7	3188	0	85825	81591
Стан 8	949	0	26096	24565
Стан 9	8942	0	241015	218501
Стан 10	589	0	16012	15627
Стан 11	4853	0	128467	123675
Стан 12	62078	0	1533418	1521257
Стан 13	239151	0	6639965	6316144
Стан 14	12	0	305	305
Стан 15	22838	0	613831	564143
Стан 16	649	0	18092	16567
Стан 17	247	0	43985	41688
Стан 18	654980	0	8095433	785538
Стан 19	7834	0	154756	152432
Стан 20	17	0	546	546

ВИСНОВОК

При виконанні даної лабораторної роботи було розглянуто два алгоритми пошуку у просторі станів та проведено порівняння результатів їх роботи. За кількістю використаної пам’яті та часу роботи, можна зробити висновок, що RBFS працює краще, адже це інформований пошук, в той час як LDFS являє собою перебір з обмеженням глибини. Втім обидва алгоритми можуть працювати довго якщо згенеровано стан для якого важко знайти оптимальний розв’язок.

КРИТЕРІЇ ОЦІНЮВАННЯ

За умови здачі лабораторної роботи до 5.11.2023 включно максимальний бал дорівнює – 5. Після 5.11.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

- псевдокод алгоритму – 10%;
- програмна реалізація алгоритму – 40%;
- робота з гіт – 20%;
- дослідження алгоритмів – 25%;
- висновок – 5%.