**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

**«Неінформативний, інформативний та локальний пошук»**

„ **Проектування і аналіз алгоритмів зовнішнього сортування**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-13 Замковий Д. В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О. О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc81070688)

[2 ЗаВдання 4](#_Toc81070689)

[3 Виконання 9](#_Toc81070690)

[3.1 Псевдокод алгоритмів 9](#_Toc81070691)

[3.2 Програмна реалізація 10](#_Toc81070692)

[3.2.1 Вихідний код 10](#_Toc81070693)

[3.2.2 Приклади роботи 16](#_Toc81070694)

[3.3 Дослідження алгоритмів 17](#_Toc81070695)

[Висновок 19](#_Toc81070696)

[Критерії оцінювання 20](#_Toc81070697)

# 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 А\*.
* **RBFS** – Рекурсивний пошук за першим найкращим співпадінням.
* **F1** – кількість пар ферзів, які б’ють один одного з урахуванням видимості (ферзь А може стояти на одній лінії з ферзем В, проте між ними стоїть ферзь С; тому А не б’є В).
* **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 Т 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

## Псевдокод алгоритмів

## Програмна реалізація

### Вихідний код

const SIZE: usize = 8;

const LIMIT: usize = 8;

enum RBFSRes {

    Ok ([usize; SIZE]),

    Err,

}

enum LDFSRes {

    Success ([usize; SIZE], usize, usize),

    Cutoff (usize, usize),

    DeadEnd (usize, usize),

}

struct State

{

    board: [usize; SIZE],

    child\_state: Vec<State>,

}

impl State {

    fn new (arr: [usize; SIZE]) -> State {

        return State {

            board: arr,

            child\_state: Vec::new(),

        };

    }

    fn expand\_state(&mut self) {

        for i in 0..SIZE {

            let row = self.board[i];

            for j in 0..SIZE {

                let mut new\_state = State::new(self.board);

                if j != row {

                    new\_state.board[i] = j;

                    self.child\_state.push(new\_state);

                }

            }

        }

    }

    fn copy (&mut self) -> State {

        let mut new\_state = State::new(self.board);

        return new\_state;

    }

}

fn arr\_out (arr: [usize; SIZE]) {

    for i in 0..SIZE {

        let mut out = "".to\_string();

        for j in 0..SIZE {

            if j == arr[i] {

                out += &"[Q]";

            } else {

                out += &"[ ]";

            }

        }

        println!("{}", out);

    }

}

fn main()

{

    // let input: [usize; SIZE] = [3, 5, 7, 1, 6, 0, 2, 4];

    let input: [usize; SIZE] = [1, 3, 6, 4, 1, 4, 6, 6];

    println!("Input:");

    arr\_out(input);

    println!("LDFS:");

    match ldfs(input, 0, 0, 8, 0, 0)

    {

        LDFSRes::Success (k, \_, \_) => arr\_out(k),

        LDFSRes::DeadEnd (c, t) => println!("Dead end\nCount of dead\_ends - {}\nTotal states - {}", c, t),

        LDFSRes::Cutoff (c, t) => println!("Not found at this depth\nCount of dead\_ends - {}\nTotal states - {}", c, t),

    };

    println!("RBFS:");

    match rbfs(&mut State::new(input), 64, 0) {

        RBFSRes::Ok(e) => arr\_out(e),

        RBFSRes::Err => println!("Error"),

    }

}

fn f2(arr: [usize; SIZE]) -> usize {

    let mut result: usize = 0;

    for i in 1..SIZE {

                for j in 0..SIZE {

                    if arr[i] + i == arr[j] + j {

                        result += 1;

                    }

                    if abs(arr[i], i) == abs(arr[j], j) {

                        result += 1;

                    }

                    if arr[i] == arr[j] {

                        result += 1;

                    }

                }

            }

            return result;

}

fn min(arr: Vec<usize>) -> (usize, usize) {

    let mut key = 0;

    let mut min = arr[key];

    for i in 1..arr.len() {

        if arr[i] < min {

            min = arr[i];

            key = i;

        }

    }

    return (min, key);

}

fn rbfs (state: &mut State, f\_limit: usize, depth: usize) -> RBFSRes

{

    if check\_state(state.board) {

        return RBFSRes::Ok(state.board);

    }

    if depth >= LIMIT {

        return RBFSRes::Err;

    }

    state.expand\_state();

    let mut f: Vec<usize> = Vec::new();

    for i in 0..state.child\_state.len() {

        f.push(f2(state.child\_state[i].board))

    }

    loop {

        let best\_value: usize;

        let best\_index: usize;

        (best\_value, best\_index) = min(f.clone());

        let mut best\_state = state.child\_state[best\_index].copy();

        if best\_value > f\_limit {

            return RBFSRes::Err;

        }

        state.child\_state.remove(best\_index);

        f.remove(best\_index);

        let alt = min(f.clone());

        if let RBFSRes::Ok(k) = rbfs(&mut best\_state, f\_limit.min(alt.0), depth + 1) {

            return RBFSRes::Ok(k);

        }

    }

}

fn ldfs (arr: [usize; SIZE], curent\_line: usize, depth: usize, max\_depth: usize, count\_of\_dead\_ends: usize, total\_states: usize) -> LDFSRes

{

    let mut count: usize = count\_of\_dead\_ends;

    let mut total: usize = total\_states + 1;

    if depth == max\_depth || curent\_line > SIZE - 1 {

        if check\_state(arr) {

            return LDFSRes::Success (arr, count, total);

        } else {

            count += 1;

            if depth == max\_depth {

                return LDFSRes::DeadEnd (count, total)

            } else {

                return LDFSRes::Cutoff (count, total)

            }

        }

    }

    match ldfs(arr, curent\_line + 1, depth, max\_depth, count, total) {

        LDFSRes::Success (k, c, t) => {

            return LDFSRes::Success (k, c, t);

        },

        LDFSRes::Cutoff (c, t) => {

            count = c;

            total = t;

        },

        LDFSRes::DeadEnd (c, t) => {

            count = c;

            total = t;

        },

    }

    for i in 0..SIZE {

        if i != arr[curent\_line] {

            let mut new\_arr = arr;

            new\_arr[curent\_line] = i;

            match ldfs(new\_arr, curent\_line + 1, depth + 1, max\_depth, count, total) {

                LDFSRes::Success (k, c, t) => {

                    return LDFSRes::Success (k, c, t);

                },

                LDFSRes::Cutoff (c, t) => {

                    count = c;

                    total = t;

                },

                LDFSRes::DeadEnd (c, t) => {

                    count = c;

                    total = t;

                },

            }

        }

    }

    return LDFSRes::Cutoff (count, total);

}

fn check\_state (arr: [usize; SIZE]) -> bool

{

    for \_i in 0..SIZE {

        if find(arr) {

            return false;

        }

    }

    for i in 0..SIZE-1 {

        for j in i+1..SIZE {

            if abs(arr[j], arr[i]) == abs(j, i) {

                return false;

            }

        }

    }

    return true;

}

fn abs (int1: usize, int2: usize) -> usize

{

    if int2 > int1 {

        return int2 - int1;

    } else {

        return int1 - int2;

    }

}

fn find (arr: [usize; SIZE]) -> bool

{

    for i in 0..SIZE-1 {

        for j in i+1..SIZE {

            if arr[i] == arr[j] {

                return true;

            }

        }

    }

    return false;

}

### Приклади роботи

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

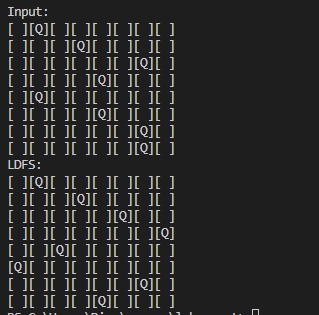


Рисунок 3.1 – Алгоритм LDFS

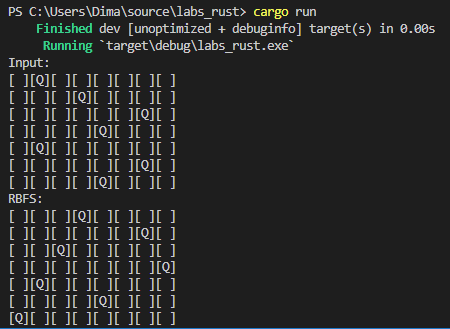


Рисунок 3.2 – Алгоритм RBFS

## Дослідження алгоритмів

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

Таблиця 3.1 – Характеристики оцінювання алгоритму пошуку з обмеженням глибини.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пам’яті | |
| [3,2,7,7,0,7,3,3] | 742853 | 649994 | 742860 | 8 | |
| [6,7,2,4,3,0,6,7] | 335536 | 293589 | 335543 | 7 | |
| [0,1,2,1,6,6,6,6] | 1485706 | 1299988 | 1485712 | 5 | |
| [0,4,1,5,2,7,3,2] | 262684 | 229843 | 262691 | 5 | |
| [2,0,2,3,7,3,0,0] | 243605 | 213149 | 243610 | 7 | |
| [3,4,3,4,0,6,1,0] | 482773 | 422421 | 482779 | 5 | |
| [4,2,5,6,5,6,0,1] | 39187 | 34283 | 39193 | 7 | |
| [4,3,0,5,2,5,3,4] | 416247 | 364211 | 416254 | 8 | |
| [0,5,2,7,5,1,2,3] | 262684 | 229843 | 262689 | 8 | |
| [3,0,3,4,2,7,4,1] | 184285 | 161245 | 184291 | 7 | |
| [3,2,4,3,0,4,1,2] | 333415 | 291733 | 333421 | 6 | |
| [1,5,6,6,4,4,6,0] | 40340 | 35292 | 40346 | 7 | |
| [6,0,0,1,7,0,2,5] | 111426 | 97439 | 111432 | 8 | |
| [7,1,4,2,5,7,4,1] | 1146 | 997 | 1152 | 6 | |
| [3,5,7,1,0,3,4,2] | 3623 | 3164 | 3628 | 8 | |
| [7,3,1,3,6,5,1,4] | 55211 | 48304 | 55216 | 7 |
| [4,3,4,2,1,2,6,6] | 477047 | 417411 | 477053 | 7 | |
| [3,2,5,4,2,6,4,2] | 820816 | 755709 | 820823 | 6 | |
| [4,7,7,0,4,6,5,1] | 152022 | 133014 | 152029 | 7 | |
| [4,1,6,1,3,7,1,7] | 177540 | 155343 | 177546 | 6 | |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| [5,6,6,2,6,6,4,2] | 6 | 0 | 336 | 65 |
| [4,7,2,4,1,6,3,4] | 9 | 0 | 504 | 64 |
| [4,4,4,7,2,2,0,7] | 3438 | 3037 | 22456 | 65 |
| [1,1,4,4,1,3,6,6] | 159 | 117 | 2352 | 63 |
| [5,5,5,7,7,3,2,4] | 4826 | 4482 | 19264 | 61 |
| [7,2,0,3,7,7,1,0] | 34 | 19 | 840 | 65 |
| [7,3,1,4,3,5,7,6] | 106 | 75 | 1736 | 61 |
| [3,1,1,0,7,6,3,6] | 14 | 3 | 616 | 65 |
| [0,7,4,0,5,1,5,0] | 862 | 675 | 10472 | 64 |
| [5,7,5,5,2,3,4,7] | 95 | 74 | 1176 | 65 |
| [6,5,2,6,7,6,4,7] | 6 | 0 | 336 | 63 |
| [1,3,3,1,5,6,2,4] | 5 | 0 | 280 | 63 |
| [4,5,7,7,2,1,6,4] | 1514 | 1320 | 10864 | 61 |
| [7,1,5,5,1,1,4,0] | 6378 | 5693 | 38360 | 65 |
| [4,1,1,6,0,4,7,4] | 4 | 0 | 224 | 60 |
| [3,2,2,3,1,2,6,1] | 60 | 40 | 1120 | 63 |
| [1,0,5,3,7,4,5,1] | 1209 | 1064 | 8120 | 64 |
| [1,1,1,4,3,5,1,0] | 56 | 37 | 1064 | 60 |
| [2,1,3,7,5,4,0,4] | 6 | 0 | 336 | 61 |
| [5,6,7,7,6,3,0,6] | 310 | 266 | 2464 | 64 |

Висновок

При виконанні даної лабораторної роботи було розглянуто алгоритми неінформативного, інформативного та локального пошуку. Було досліджено переваги та недоліки кожного з методів, а також набуто практичних навичок проектування алгоритмів під час реалізації алгоритмів RBFS та LDFS для задачі 8 ферзів.

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%.