**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-12 Скорик Р.О.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc81070688)

[2 ЗаВдання 4](#_Toc81070689)

[3 Виконання 8](#_Toc81070690)

[3.1 Псевдокод алгоритмів 8](#_Toc81070691)

[3.2 Програмна реалізація 8](#_Toc81070692)

[3.2.1 Вихідний код 8](#_Toc81070693)

[3.2.2 Приклади роботи 8](#_Toc81070694)

[3.3 Дослідження алгоритмів 8](#_Toc81070695)

[Висновок 11](#_Toc81070696)

[Критерії оцінювання 12](#_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 |

Варіант 22

# Виконання

## Псевдокод алгоритмів

**LDFS**

**function** LDFS(problem) **returns** result **or** failure

RecLDFS(problem, startNode[proplem], limit)

**function** RecLDFS (problem, node, limit) **returns** result **or** failure/cutoff

cutoff\_occured = False

**if** problem.sloved(state[node]) **then return** node

**if** depth[node] = limit **then return** cutoff

successors = Expand(node, problem)

**for each** i **in** successors **do**

result = RecLDFS(problem, best,)

**if** result = cutoff **then** cutoff\_occured = True

**else if** result != failure **then return** result

**if** cutoff\_occurred

then **return** cutoff

**else** **return** failure

**RBFS**

**function** RBFS(problem) retur**ns** result **or** failure

RecRBFS(problem, startNode[proplem], inf)

**function** RecRBFS(problem, node, f\_limit**) returns** result **or** failure, f\_limit

**if** problem.sloved(state[node]) **then return** node

successors = Expand(node, problem)

**if** isEmpty(successors) **then return** failure, inf

**for each** i **in** successors **do**

f[i] = max(g(i) + h(i), f[node])

**repeat**

best = min\_f(successors)

**if** f[best] > f\_limit **then return** failure, f[best]

alternative = second\_min\_f(f[successors])

result, f[best] = RecRBFS(problem, best, min(f\_limit, alternative))

**if** result != failure **then return** result

## Програмна реалізація

### Вихідний код

**LDFS**

#include <iostream>  
#include <vector>

void LDFS(Node\* start, int limit, bool &failure, Node\*&res) {  
 bool cutoff = false;  
 failure = true;  
 RecDFS(start, limit, cutoff, failure, res);  
 if(cutoff == true)  
 std::cout<<"memory or time etention"<<std::endl;  
}  
  
void RecDFS(Node \*cur, int limit, bool &cutoff, bool &failure, Node\*& res) {  
  
  
  
 int row[] = { -1, 0, 1, 0 };  
 int col[] = { 0, -1, 0, 1 };  
  
 cutoff = false;  
 if(isSolution(\*cur))  
 {  
 res= cur;  
 failure = false;  
 }  
 else if(cur->getLevel() == limit)  
 cutoff = true;  
 else  
 for(int i=0; i<4 && failure; ++i)  
 {  
 int x\_tmp= cur->getX() + row[i];  
 int y\_tmp= cur->getY() + col[i];  
 if(isPossible(x\_tmp, y\_tmp, cur->getParent()))  
 {  
 Node\* next = new Node(cur, x\_tmp, y\_tmp);  
 RecDFS(next, limit, cutoff, failure, res);  
 }  
  
 }  
}  
  
bool isPossible(int x, int y, Node\* per) {  
 if(per == nullptr)  
 return (x>=0 && x<=2)&&(y>=0 && y<=2);  
 else  
 return (x>=0 && x<=2)&&(y>=0 && y<=2) && x!=per->getX() && y!=per->getY();  
}

**RBFS**

void RBFS(NodeWithPrice \*start, bool &failure, NodeWithPrice \*&res) {  
 bool cutoff = false;  
 failure = true;  
 int f\_limit = 10000;  
 RecBFS(start, f\_limit, failure, res);  
}  
  
void RecBFS(NodeWithPrice \*cur, int& f\_limit, bool &failure, NodeWithPrice \*&res) {  
  
 int row[] = { -1, 0, 1, 0 };  
 int col[] = { 0, -1, 0, 1 };  
  
 if(isSolution(\*cur))  
 {  
 res= cur;  
 failure = false;  
 return;  
 }  
 std::priority\_queue<NodeWithPrice\*, std::vector<NodeWithPrice\*>, LessThanByF> successors;  
 for(int i=0; i<4 && failure; ++i)  
 {  
 int x\_tmp= cur->getX() + row[i];  
 int y\_tmp= cur->getY() + col[i];  
 if(isPossible(x\_tmp, y\_tmp, cur->getParent()))  
 {  
 NodeWithPrice\* tmp = new NodeWithPrice(cur, x\_tmp, y\_tmp);  
 successors.push(tmp);  
 }  
  
 }  
 while (failure && !successors.empty()){  
 NodeWithPrice\* best = successors.top();  
 if (best->getF() > f\_limit){  
 f\_limit = best->getF();  
 clearQueue(successors);  
 return;  
 }  
 successors.pop();  
  
 int f\_lim\_tmp;  
 if (!successors.empty())  
 f\_lim\_tmp =std::min(f\_limit, successors.top()->getF());  
 else  
 f\_lim\_tmp =f\_limit;  
  
 RecBFS(best, f\_lim\_tmp, failure, res);  
 best->setF(f\_lim\_tmp);  
 successors.push(best);  
 }  
  
  
}  
  
void clearQueue( std::priority\_queue<NodeWithPrice\*, std::vector<NodeWithPrice\*>, LessThanByF> &tmp) {  
 while (!tmp.empty()){  
 delete tmp.top();  
 tmp.pop();  
 }  
}

bool isPossible(int x, int y, NodeWithPrice\* per) {  
 if(per == nullptr)  
 return (x>=0 && x<=2)&&(y>=0 && y<=2);  
 else  
 return (x>=0 && x<=2)&&(y>=0 && y<=2) && x!=per->getX() && y!=per->getY();  
}

### Приклади роботи

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

Рисунок 3.1 – Алгоритм LDFS

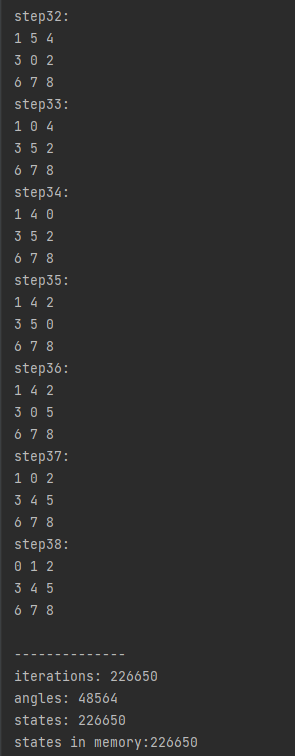
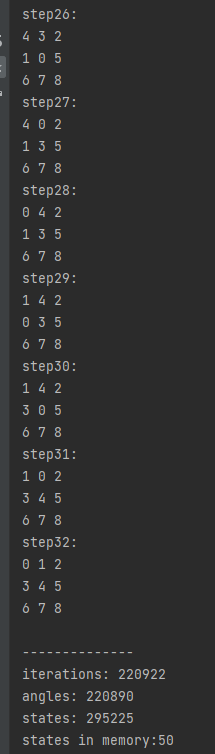


Рисунок 3.2 – Алгоритм RBFS



## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму LDFS, задачі 8-puzzle 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 |
| 6 3 7  4 1 5  2 8 0 | 477751 | 143320 | 477751 | 477751 |
| 8 7 4  2 6 5  1 3 0 | 404846 | 121448 | 404846 | 404846 |
| 5 4 8  2 0 6  1 7 3 | 814543 | 222144 | 814543 | 814543 |
| 5 6 0  3 8 4  7 1 2 | 93621 | 28080 | 93621 | 93621 |
| 1 6 0  7 5 2  4 3 8 | 614559 | 184360 | 614559 | 614559 |
| 8 5 7  2 1 6  0 4 3 | 569540 | 170856 | 569540 | 569540 |
| 5 8 0  1 6 7  2 3 4 | 3418201 | 1025456 | 3418201 | 3418201 |
| 6 0 3  1 4 7  8 2 5 | 2692925 | 616624 | 2692925 | 2692925 |
| 5 8 4  7 0 1  3 2 6 | 1032042 | 281460 | 1032042 | 1032042 |
| 7 3 4  5 2 0  1 8 6 | 184719 | 69260 | 184719 | 184719 |
| 8 4 5  2 7 6  0 3 1 | 923545 | 277056 | 923545 | 923545 |
| 3 1 2  4 7 8  5 6 0 | 10184 | 3048 | 10184 | 10184 |
| 2 6 5  3 7 1  0 8 4 | 1552021 | 465600 | 1552021 | 1552021 |
| 4 2 0  6 1 3  8 7 5 | 605352 | 181600 | 605352 | 605352 |
| 1 5 2  3 6 8  4 7 0 | 854 | 248 | 854 | 854 |
| 7 1 4  2 6 8  0 3 5 | 1256822 | 377040 | 1256822 | 1256822 |
| 1 8 5  6 2 7  3 4 0 | 2240340 | 672096 | 2240340 | 2240340 |
| 4 5 8  1 2 0  3 7 6 | 642717 | 241012 | 642717 | 642717 |
| 8 7 2  6 0 4  1 3 5 | 252546 | 68868 | 252546 | 252546 |
| 5 4 7  3 1 2  8 6 0 | 377486 | 113240 | 377486 | 377486 |
| 4 6 3  2 1 5  7 8 0 | 864977 | 259488 | 864977 | 864977 |
| Середнє | 906171 | 262966,9 | 906171 | 906171 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму RBFS, задачі 8-puzzle 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 |
| 1 8 3  2 0 4  5 7 6 | 21447666 | 21447627 | 28656016 | 61 |
| 8 2 3  4 0 7  1 6 5 | 228768 | 228737 | 306662 | 49 |
| 0 8 4  5 1 2  3 7 6 | 1096272 | 1096239 | 1473004 | 50 |
| 1 8 7  6 0 2  3 5 4 | 722497 | 722466 | 990284 | 49 |
| 0 3 7  6 1 5  8 4 2 | 43227 | 43198 | 58076 | 44 |
| 8 6 5  2 7 3  0 4 1 | 111787 | 111754 | 148695 | 50 |
| 7 3 8  4 0 1  6 5 2 | 6395688 | 6395653 | 8588123 | 55 |
| 3 2 5  4 7 1  6 0 8 | 11 | 3 | 18 | 14 |
| 7 3 6  8 2 5  0 4 1 | 9838775 | 9838734 | 13105757 | 62 |
| 7 3 0  8 2 6  5 4 1 | 683295 | 683258 | 916702 | 56 |
| 4 6 5  0 1 3  7 2 8 | 73156 | 73126 | 98874 | 46 |
| 5 4 7  1 2 8  6 0 3 | 332213 | 332181 | 449398 | 50 |
| 5 6 8  3 7 1  0 2 4 | 1565340 | 1565303 | 2106004 | 56 |
| 8 1 4  0 7 2  6 5 3 | 5792076 | 5792040 | 7812648 | 56 |
| 7 3 5  2 8 0  6 4 1 | 6976084 | 6976048 | 9449844 | 56 |
| 0 4 5  2 6 8  1 3 7 | 350384 | 350351 | 470297 | 50 |
| 7 2 6  1 0 8  5 3 4 | 1385331 | 1385296 | 1852181 | 55 |
| 5 6 1  0 8 7  4 3 2 | 2038055 | 2038019 | 2711887 | 56 |
| 0 6 1  3 7 2  5 4 8 | 106082 | 106053 | 142299 | 44 |
| 2 1 8  4 0 7  6 3 5 | 83380 | 83353 | 112014 | 43 |
| 3 7 0  5 2 8  6 1 4 | 476529 | 476496 | 651485 | 50 |
| Середнє | 2845077 | 2845045 | 3814298 | 50,09524 |

Висновок

При виконанні даної лабораторної роботи було розглянуто і порівняно 2 алгоритми пошуку деревом на задачі 8-puzzle. Перший - неінформативний LDFS пошук, другий – інформативний RBFS. Розглянувши властивості бачимо, що у першого алгоритму є ряд недоліків: він не гарантує знаходження оптимального розв’яку, якщо кількість кроків у ньому не дорівнює глубині пошуку,а також не знайде розв’язку взагалі, якщо глубина пошуку недостатня для оптимального розв’язку; він зберігає у пам’ятті усі створені ноди; через неігформативну природу, він не відсікає заздалегідь неоптимальні стани. Натомість другий алгоритм цих недоліків позбавлений. На практиці, за глубини пошуку в 40 LDFS майже гарантовано вирішує цю задачу, хоч не оптимально, роблячи це за порівнянний з RBFS час(хоч і програє в середньому), але використовує при цьому кратно більше пам’яті. Основним недоліком RBFS ж є те, що він використовує замало пам’яті: непотрбні ноди постійно видаляються, ожнак це призводить до значної кількості рекрсивних повернень, збільшуючи кількість ітерацій алгоритму. За використання більшох кількості пам’яті, пошук був би значно кращим за критерієм часу. Отож, неінформативний алгоритм був би кращим лише на достатньо потужній машині за умови, що достатньо мати неоптимальний розв’язок та створити хорошу еврестичну функцію для задачі неможливо, що буває вкрай рідко і для складних задач. Інформативний RBFS ідеально підходть для мінімзації затрат пам’яті, тож актуальний колм це вузьке місце системи.