**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-13 Нещерет В.О.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О. О.*

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 9](#_Toc81070692)

[3.2.1 Вихідний код 9](#_Toc81070693)

[3.2.2 Приклади роботи 12](#_Toc81070694)

[3.3 Дослідження алгоритмів 13](#_Toc81070695)

[Висновок 21](#_Toc81070696)

[Критерії оцінювання 22](#_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

**LDFS**

Функція RecursiveSearch(node, limit)

ЯКЩО node.state == goalState

ПОВЕРНУТИ node

ВСЕ ЯКЩО

ЯКЩО node.depth >= limit

ПОВЕРНУТИ Cutoff

ВСЕ ЯКЩО

Successors = node.expand()

ЦИКЛ для і в successors

Res = RecursiveSearch(i, limit)

ЯКЩО res != Cutoff && res != Fail

ПОВЕРНУТИ res

ВСЕ ЯКЩО

КІНЕЦЬ ЦИКЛА

ПОВЕРНУТИ Fail

**A\***

queue = PriorityQueue()

closed = List()

queue.add(initialNode)

ПОКИ queue має елементи

currentNode = queue.remove()

ЯКЩО node.state == goalState

ПОВЕРНУТИ node

ВСЕ ЯКЩО

Successors = node.expand()

ЦИКЛ для і в successors

ЯКЩО i не присутнє в closed

queue.add(i)

ВСЕ ЯКЩО

КІНЕЦЬ ЦИКЛА

КІНЕЦЬ ПОКИ

ПОВЕРНУТИ Fail

## Програмна реалізація

### Вихідний код

**LDFS**

import java.lang.Integer.max

class LDFS(val startState: State) {

private val goal = State(

listOf(

mutableListOf(0, 1, 2),

mutableListOf(3, 4, 5),

mutableListOf(6, 7, 8),

)

)

var iterations = 0

var totalStateCounter = 0

var memoryStateCounter = 1

var maxMemoryStateCounter = memoryStateCounter

var deadEndCounter = 0

fun search(limit: Int, startTime: Long): Result {

val root = Node(startState, 0)

return recursiveSearch(root, limit, startTime)

}

fun recursiveSearch(node: Node, limit: Int, startTime : Long): Result {

iterations++

memoryStateCounter -= 1

maxMemoryStateCounter = max(maxMemoryStateCounter, memoryStateCounter)

if (node.state.sameState(goal)) return Result(node, ResultType.SOLUTION)

if (node.depth >= limit || !Statistic().isEnoughMemory() || !Statistic().isEnoughTime(startTime)) {

deadEndCounter++

return Result(node, ResultType.CUTOFF)

}

val successors = node.expand()

memoryStateCounter += successors.size

totalStateCounter += successors.size

for (i in successors) {

val res = recursiveSearch(i, limit, startTime)

if (res.type == ResultType.SOLUTION) return res

}

return Result(node, ResultType.FAIL)

}

}

**A\***

import java.util.\*

class AStar(val startState: State) {

private val goal = State(

listOf(

mutableListOf(0, 1, 2),

mutableListOf(3, 4, 5),

mutableListOf(6, 7, 8),

)

)

var iterations = 0

var totalStateCounter = 1

var isNewStates = false

var deadEndCounter = 0

fun search(startTime : Long): Result {

val root = Node(startState, 0)

val h: Comparator<Node> = compareBy { it.value }

val queue = PriorityQueue(h)

val closedList = mutableListOf<Node>()

queue.add(root)

while (queue.isNotEmpty()) {

isNewStates = false

iterations++

val currentNode = queue.remove()

closedList.add(currentNode)

if (currentNode.state.sameState(goal)) return Result(currentNode, ResultType.SOLUTION)

if (!Statistic().isEnoughMemory() || !Statistic().isEnoughTime(startTime)) {

deadEndCounter++

return Result(currentNode, ResultType.CUTOFF)

}

val successors = currentNode.expand()

for (i in successors) {

var flag = true

for (j in closedList) {

if (j.state.sameState(i.state)) {

flag = false

break

}

}

if (flag) {

queue.add(i)

totalStateCounter++

isNewStates = true

}

}

if (!isNewStates) deadEndCounter++

}

return Result(root, ResultType.FAIL)

}

fun printStats() {

println("Iterations: $iterations\n" +

"Dead ends: $deadEndCounter\n" +

"Total states: $totalStateCounter\n" +

"Max states in memory: $totalStateCounter")

}

}

### Приклади роботи

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

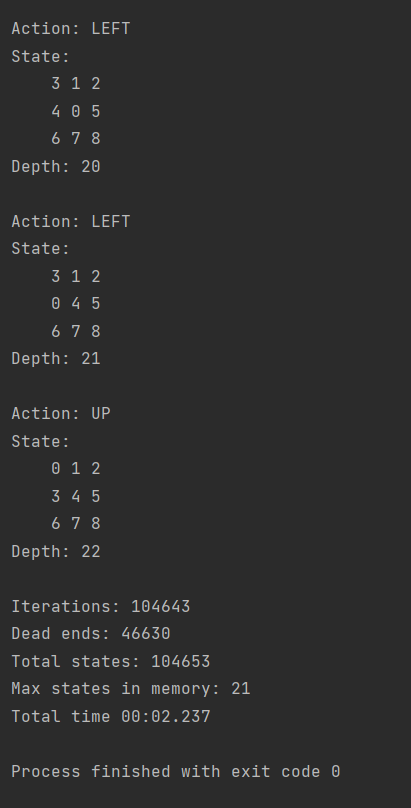


Рисунок 3.1 – Алгоритм LDFS

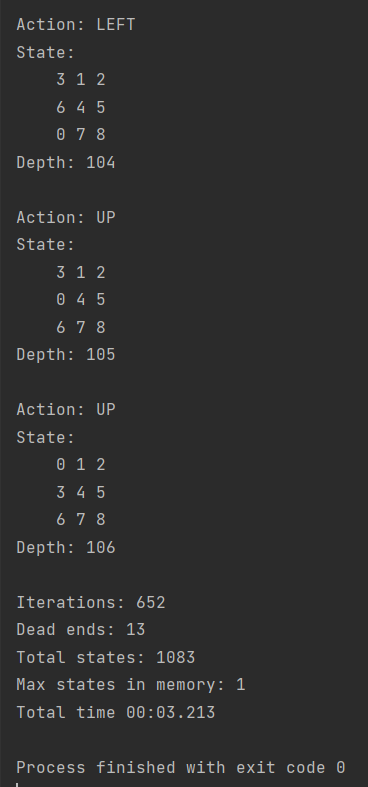


Рисунок 3.2 – Алгоритм A\*

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму LDFS для 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 пом’яті |
| **Стан 1**  3 4 7  2 0 6  1 8 5  Limit = 22 | 325 682 | 145 085 | 325 695 | 23 |
| **Стан 2**  1 0 5  3 2 6  8 7 4  Limit = 22 | 226 902 | 90 650 | 226 911 | 23 |
| **Стан 3**  6 3 5  8 4 7  1 2 0  Limit =22 | 636 907 | 282 616 | 636 906 | 22 |
| **Стан 4**  0 1 2  4 5 7  8 3 6  Limit = 22 | 103 391 | 46 085 | 103 399 | 19 |
| **Стан 5**  0 3 8  7 4 2  1 5 6  Limit = 22 | 636 907 | 282 616 | 636 906 | 21 |
| **Стан 6**  1 0 5  7 8 6  2 3 4  Limit = 22 | 672 816 | 269 335 | 672 822 | 23 |
| **Стан 7**  4 1 2  5 7 8  0 3 6  Limit = 22 | 34 284 | 15 163 | 34 290 | 19 |
| **Стан 8**  7 0 8  5 3 4  6 1 2  Limit = 22 | 708 585 | 283 640 | 708 584 | 23 |
| **Стан 9**  1 0 5  4 6 2  7 3 8  Limit = 22 | 131 708 | 52 610 | 131 715 | 21 |
| **Стан 10**  8 0 7  5 1 4  6 3 2  Limit = 22 | 708 585 | 283 640 | 708 584 | 23 |
| **Стан 11**  5 7 0  2 8 4  1 3 6  Limit = 22 | 174 466 | 77 558 | 174 472 | 21 |
| **Стан 12**  7 6 0  4 2 8  5 1 3  Limit = 22 | 636 907 | 282 616 | 636 906 | 21 |
| **Стан 13**  6 0 3  4 5 2  8 1 7  Limit = 22 | 708 585 | 283 640 | 708 584 | 23 |
| **Стан 14**  3 5 2  0 7 8  4 6 1  Limit = 22 | 182 248 | 72 654 | 182 260 | 21 |
| **Стан 15**  2 4 5  0 6 7  3 1 8  Limit = 22 | 557 483 | 223 311 | 557 494 | 22 |
| **Стан 16**  4 3 5  6 7 8  2 0 1  Limit = 22 | 217 842 | 87 594 | 217 852 | 23 |
| **Стан 17**  8 5 2  6 1 7  4 0 3  Limit = 22 | 708 585 | 283 640 | 708 584 | 23 |
| **Стан 18**  3 8 1  4 0 5  6 7 2  Limit = 22 | 24 203 | 10 666 | 24 212 | 21 |
| **Стан 19**  2 0 1  3 5 6  8 7 4  Limit = 22 | 708 585 | 283640 | 708 584 | 23 |
| **Стан 20**  5 6 4  2 7 8  1 0 3  Limit = 22 | 113 459 | 45 549 | 113 466 | 22 |

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

Таблиця 3.2 – Характеристики оцінювання алгоритму А\*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| **Стан 1**  3 0 1  2 7 5  4 6 8 | 80 | 2 | 137 | 137 |
| **Стан 2**  3 2 4  6 0 5  8 7 1 | 284 | 6 | 478 | 478 |
| **Стан 3**  8 0 2  6 5 3  4 7 1 | 1475 | 19 | 2454 | 2454 |
| **Стан 4**  8 4 1  6 7 5  2 3 0 | 838 | 16 | 1380 | 1380 |
| **Стан 5**  6 3 1  8 7 0  5 2 4 | 284 | 3 | 471 | 471 |
| **Стан 6**  7 1 8  0 4 3  2 5 6 | 662 | 10 | 1096 | 1096 |
| **Стан 7**  7 1 8  6 5 2  3 4 0 | 449 | 11 | 744 | 744 |
| **Стан 8**  4 6 3  8 5 7  2 1 0 | 332 | 3 | 558 | 558 |
| **Стан 9**  7 6 8  1 0 4  5 2 3 | 949 | 15 | 1558 | 1558 |
| **Стан 10**  0 2 1  4 6 3  8 5 7 | 280 | 7 | 469 | 469 |
| **Стан 11**  1 6 3  8 3 4  2 5 0 | 914 | 25 | 1486 | 1486 |
| **Стан 12**  0 1 2  4 6 3  8 7 5 | 202 | 4 | 341 | 341 |
| **Стан 13**  6 5 4  7 8 1  2 3 0 | 74 | 2 | 131 | 131 |
| **Стан 14**  0 6 4  1 3 2  7 5 8 | 24 | 0 | 44 | 44 |
| **Стан 15**  6 3 8  4 5 7  0 1 2 | 391 | 7 | 670 | 670 |
| **Стан 16**  2 4 1  7 5 6  0 3 8 | 885 | 26 | 1439 | 1439 |
| **Стан 17**  0 3 1  4 7 6  8 2 5 | 1545 | 30 | 2561 | 2561 |
| **Стан 18**  6 0 3  8 4 1  2 5 7 | 960 | 23 | 1599 | 1599 |
| **Стан 19**  8 0 1  5 2 4  6 3 7 | 1213 | 23 | 2002 | 2002 |
| **Стан 20**  0 2 1  6 4 5  7 8 3 | 46 | 1 | 80 | 80 |

Висновок

При виконанні даної лабораторної роботи було розглянуто інформовані та неінформовані алгоритми пошуку, виконано програмну реалізацію алгоритмів LDFS та A\*. При порівнянні алгоритмів явно помітно, що інформований пошук набагато швидше знаходить рішення та використовує менше пам'яті. Проте, він не гарантує знаходження оптимального рішення на відміну від LDFS.

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%.