**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»**

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-12 Орищенко Я.О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070688)

[2 ЗаВдання 4](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070689)

[3 Виконання 8](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070690)

[3.1 Псевдокод алгоритмів 8](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070691)

*3.1.1 Псевдокод алгоритму LDFS..........................................................8*

*3.1.2 Псевдокод алгоритму A\*...............................................................8*

[3.2 Програмна реалізація *9*](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070692)

[3.2.1 Вихідний код 9](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070693)

[3.2.2 Приклади роботи 14](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070694)

[3.3 Дослідження алгоритмів 15](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070695)

[Висновок 17](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_Toc81070696)

[Критерії оцінювання 18](file:///C:\Users\Ярослав\Desktop\КПІ\Курс%202\Проєктування%20алгоритмів\lr2_2022.ukr.docx#_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

3.1.1. Псевдокод алгоритму **LDFS**

**function** depth\_limited\_search():

**return** recursive\_dls(root)

**function** recursive\_dls(node):

**if** node.puzzle.solved() **then**:

**return** node.puzzle

**if** node.level == 22 **then**:

r**eturn** None

**for each** move **in** node.moves() **do**:

result = recursive\_dls(Node(move()))

**if** result != None **then**:

**return** result

**return** None

3.1.2. Псевдокод алгоритму **A\***

**function** a\_star(puzzle):

node = Node(puzzle)

queue = PriorityQueue()

queue.put(node.f(), node)

**while** queue **do**:

node = queue.get()

**if** node.puzzle.solved() **then**:

return node.puzzle

**for** move **in** node.moves() **do**:

child = Node(move())

queue.put(child.f(), child)

## Програмна реалізація

### Вихідний код

**main.py**

from puzzle import Puzzle

from solution import Solution

import time

def solve(method, board)

puzzle = Puzzle(board)

print(your puzzle)

puzzle.pprint()

solution = Solution(puzzle)

start = time.time()

if method == a\_star

path = solution.a\_star()

elif method == dls

path = solution.dls()

else

print(wrong method!)

return

end = time.time()

if path is not None

for node in path

node.puzzle.pprint()

print(finding solution time , end - start)

return path

def test(num, board)

for i in range(num)

print(- 30 + n)

puzzle = Puzzle(board)

puzzle.shuffle\_board()

solution = Solution(puzzle)

puzzle.pprint()

print(nA solution )

start = time.time()

res1 = solution.a\_star()

print(time , time.time() - start)

print(nDLS solution )

start = time.time()

res2 = solution.dls()

print(time , time.time() - start)

board = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

res = solve(a\_star, board)

**puzzle.py**

class Puzzle:

def \_\_init\_\_(self, board):

self.board = board

self.goal = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

def solved(self):

return self.board == self.goal

def find\_moves(self):

i = -1

j = -1

moves = []

for x in range(3):

for y in range(3):

if self.board[x][y] == 0:

i = x

j = y

break

directs = {'R': (i, j - 1),

'L': (i, j + 1),

'U': (i + 1, j),

'D': (i - 1, j)}

for action, (i1, j1) in directs.items():

if 0 <= i1 < 3 and 0 <= j1 < 3:

move = self.create\_move((i, j), (i1, j1))

moves.append(move)

return moves

def create\_move(self, at, to):

copy = self.copy()

i, j = at

i1, j1 = to

copy.board[i][j], copy.board[i1][j1] = copy.board[i1][j1], copy.board[i][j]

return copy

def copy(self):

board = []

for row in self.board:

board.append([x for x in row])

return Puzzle(board)

def h(self):

not\_placed = 0

for i in range(3):

for j in range(3):

if self.board[i][j] != self.goal[i][j] and self.board[i][j] != 0:

not\_placed += 1

return not\_placed

def pprint(self):

for i in range(3):

print(self.board[i])

print()

**node.py**

from puzzle import Puzzle

class Node:

def \_\_init\_\_(self, puzzle: Puzzle, parent=None):

self.puzzle = puzzle

self.parent = parent

if self.parent is not None:

self.level = parent.level + 1

else:

self.level = 0

def f(self):

return self.level + self.h()

def h(self):

return self.puzzle.h()

def find\_parents(self):

node = self

p = []

while node:

p.append(node)

node = node.parent

return reversed(p)

def solved(self):

return self.puzzle.solved()

def \_\_lt\_\_(self, other):

return self.f() < other.f()

def \_\_gt\_\_(self, other):

return self.f() > other.f()

def moves(self):

return self.puzzle.find\_moves()

**solotion.py**

from puzzle import Puzzle

from node import Node

from queue import PriorityQueue

class Solution:

def \_\_init\_\_(self, puzzle: Puzzle):

self.puzzle = puzzle

def a\_star(self):

node = Node(self.puzzle)

queue = PriorityQueue()

queue.put((node.f(), node))

while queue:

node = queue.get()[1]

if node.solved():

print("SOLUTION FOUND")

return node.find\_parents()

for move in node.moves():

child = Node(move, node)

queue.put((child.f(), child))

return None

def dls(self):

return self.recursive\_dls(Node(self.puzzle))

def recursive\_dls(self, node: Node):

if node.solved():

print("SOLUTION FOUND")

print("solution level in tree: ", node.level)

return node.puzzle

if node.level == 22:

return None

for move in node.moves():

result = self.recursive\_dls(Node(move, node))

if result is not None:

return result

return None

### Приклади роботи

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

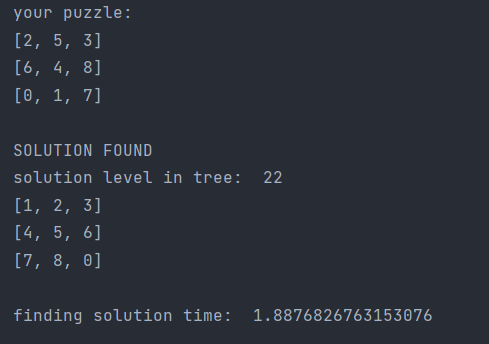


Рисунок 3.1 – Алгоритм LDFS

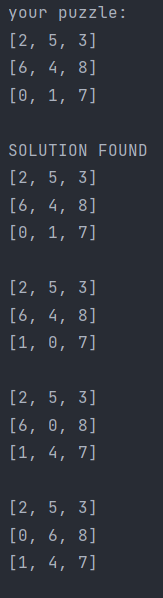
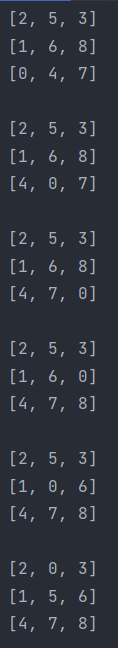
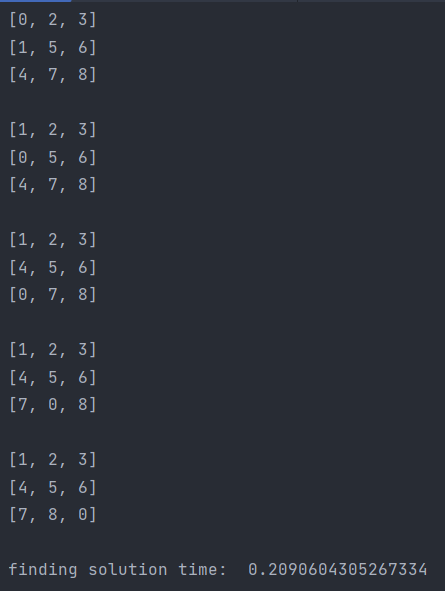
  

Рисунок 3.2 – Алгоритм A\*

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму LDFS 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 |
| Стан 1 | 82871 | 54373 | 82870 | 21 |
| Стан 2 | 23841 | 15634 | 23840 | 22 |
| Стан 3 | 12423747 | 7906012 | 12423740 | 22 |
| Стан 4 | 15083390 | 9594402 | 15095466 | 21 |
| Стан 5 | 1029582 | 649628 | 1044879 | 21 |
| Стан 6 | 841744 | 551698 | 843683 | 22 |
| Стан 7 | 181862 | 115634 | 182103 | 21 |
| Стан 8 | 115530 | 72994 | 117102 | 21 |
| Стан 9 | 5982800 | 3788386 | 6036482 | 21 |
| Стан 10 | 9720 | 5020 | 13698 | 22 |
| Стан 11 | 126252 | 82669 | 126722 | 22 |
| Стан 12 | 91850494 | 59949733 | 92772387 | 22 |
| Стан 13 | 1405080 | 893874 | 1405807 | 21 |
| Стан 14 | 175247 | 111324 | 175800 | 21 |
| Стан 15 | 1483697 | 941132 | 1493189 | 21 |
| Стан 16 | 2205994 | 1444186 | 2215005 | 22 |
| Стан 17 | 30420 | 19360 | 32079 | 22 |
| Стан 18 | 135078 | 88207 | 136306 | 22 |
| Стан 19 | 21130385 | 13440588 | 21145713 | 21 |
| Стан 20 | 1938521 | 1232482 | 1941564 | 21 |

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 пам’яті |
| Стан 1 | 650 | 0 | 1841 | 1841 |
| Стан 2 | 494 | 0 | 1393 | 1393 |
| Стан 3 | 35009 | 0 | 100390 | 100390 |
| Стан 4 | 121267 | 0 | 352374 | 352374 |
| Стан 5 | 6456 | 0 | 18533 | 18533 |
| Стан 6 | 8723 | 0 | 24021 | 24021 |
| Стан 7 | 135 | 0 | 377 | 377 |
| Стан 8 | 808 | 0 | 2381 | 2381 |
| Стан 9 | 29608 | 0 | 83291 | 83291 |
| Стан 10 | 2051 | 0 | 6030 | 6030 |
| Стан 11 | 263 | 0 | 734 | 734 |
| Стан 12 | 498506 | 0 | 1420400 | 1420400 |
| Стан 13 | 406 | 0 | 1134 | 1134 |
| Стан 14 | 4760 | 0 | 14253 | 14253 |
| Стан 15 | 5314 | 0 | 14326 | 14326 |
| Стан 16 | 1777 | 0 | 4932 | 4932 |
| Стан 17 | 901 | 0 | 2561 | 2561 |
| Стан 18 | 651 | 0 | 1880 | 1880 |
| Стан 19 | 9447 | 0 | 24776 | 24776 |
| Стан 20 | 1749 | 0 | 4793 | 4793 |

Висновок

При виконанні даної лабораторної роботи було розглянуто принципи роботи алгоритмів неінформативного, інформативного а також локального пошуку. На практиці було розроблено і протестовано алгоритм неінформативного пошуку LDFS (Пошук вглиб з обмеженням глибини) і алгоритм інформативного пошуку A\*, на прикладі розв’язку задачі 8-puzzle.

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%.