**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-15 Розін Олексій Іванович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2022

Зміст

[**1**](#_30j0zll) **Мета лабораторної роботи 3**

[**2**](#_3znysh7) **ЗаВдання 4**

[**3**](#_2et92p0) **Виконання 8**

[3.1](#_tyjcwt) Псевдокод алгоритмів 8

[3.2](#_3dy6vkm) Програмна реалізація 8

[*3.2.1*](#_1t3h5sf) *Вихідний код 8*

[*3.2.2*](#_4d34og8) *Приклади роботи 8*

[3.3](#_2s8eyo1) Дослідження алгоритмів 8

[**Висновок 11**](#_17dp8vu)

[**Критерії оцінювання 12**](#_3rdcrjn)

# 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-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.
* **BFS** – Пошук вшир.
* **A\*** – Пошук А\*.
* **H1** – кількість фішок, які не стоять на своїх місцях.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Задача** | **АНП** | **АІП** | **АЛП** | **Func** |
| 23 | 8-puzzle | BFS | A\* |  | H1 |

# Виконання

## Псевдокод алгоритмів

root – вузол початкового стану пазлу

extendNode() – метод вузла, який створює у вузла атрибут-масив children нових вузлів

isGoal() – метод вузла, який перевіряє чи дорівнює пазл вузла вирішеному пазлу

heuristic(int[] puzzle) – еврестична функція для розрахунку кількісті фішок, які не стоять на своїх місцях.

**Алгоритм BFS**

**Початок**

openList = []

openList.push(root)

goalFound = False

**поки** openList.length > 0 **та** goalFound == False **повторити**

currNode = openList.pop(0)

currNode.extendNode()

**для кожного** child **в** currNode.children **повторити**

**якщо** child.isGoal()

goalFound = True

**все якщо**

openList.push(child)

**все повторити**

**все повторити**

**Кінець**

**Алгоритм А\***

**Початок**

openList = []

closedList = []

openList.push(root)

goalFound = False

**поки** openList.length > 0 **та** goalFound == False **повторити**

currNode = openList.pop(0)

closedList.push(currNode)

currNode.extendNode()

**для кожного** child **в** currNode.children **повторити**

**якщо** child.isGoal()

goalFound = True

**все якщо**

**якщо** child **не входить в** closedList

child.hVal = heuristic(child)

openList.push(child)

**все якщо**

**все повторити**

**відсортувати** openList **по** hVal

**все повторити**

**Кінець**

## Програмна реалізація

### Вихідний код

1. class MyTree:
2. """Клас дерева пошуку"""
3. def \_\_init\_\_(self,root):
4. self.root = root
5. def BreadthFirstSearch(self):
6. pathToSolution = []
7. openList = []
8. iterations = 0
9. stateCount = 0
10. openList.append(self.root)
11. goalFound = False
12. while len(openList) > 0 and not goalFound:
13. iterations += 1
14. currNode = openList.pop(0)
15. currNode.extendNode()
16. stateCount += 4
17. for child in currNode.children:
18. if child.isGoal():
19. pathToSolution = self.pathTrace(child)
20. goalFound = True
22. openList.append(child)
23. return pathToSolution[::-1], iterations, len(openList), stateCount
25. def A(self):
26. pathToSolution = []
27. openList = []
28. closedList = []
29. iterations = 0
30. stateCount = 0
31. openList.append(self.root)
32. goalFound = False
33. while len(openList) > 0 and not goalFound:
34. iterations += 1
35. currNode = openList.pop(0)
36. closedList.append(currNode)
37. currNode.extendNode()
38. stateCount += 4
39. for child in currNode.children:
40. if child.isGoal():
41. pathToSolution = self.pathTrace(child)
42. goalFound = True
43. if not self.contains(closedList,child):
44. child.hVal = self.heuristic(child)
45. openList.append(child)
46. openList.sort(key = lambda x:x.hVal,reverse=False)
47. return pathToSolution[::-1], iterations, (len(openList) + len(closedList)), stateCount
48. def contains(self, nodes, myNode):
49. contains = False
50. for node in nodes:
51. if node.puzzle == myNode.puzzle:
52. contains = True
53. break
54. return contains
55. def heuristic(self,node):
56. goal = [1,2,3,4,5,6,7,8,0]
57. score = 0
58. for i in range(9):
59. if node.puzzle[i] != goal[i]:
60. score += 1
61. return score
62. def pathTrace(self,currentNode):
63. path = []
64. path.append(currentNode.puzzle)
65. while currentNode.parent != None:
66. currentNode = currentNode.parent
67. path.append(currentNode.puzzle)
69. return path

### Приклади роботи

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

Рисунок 3.1 – Алгоритм BFS

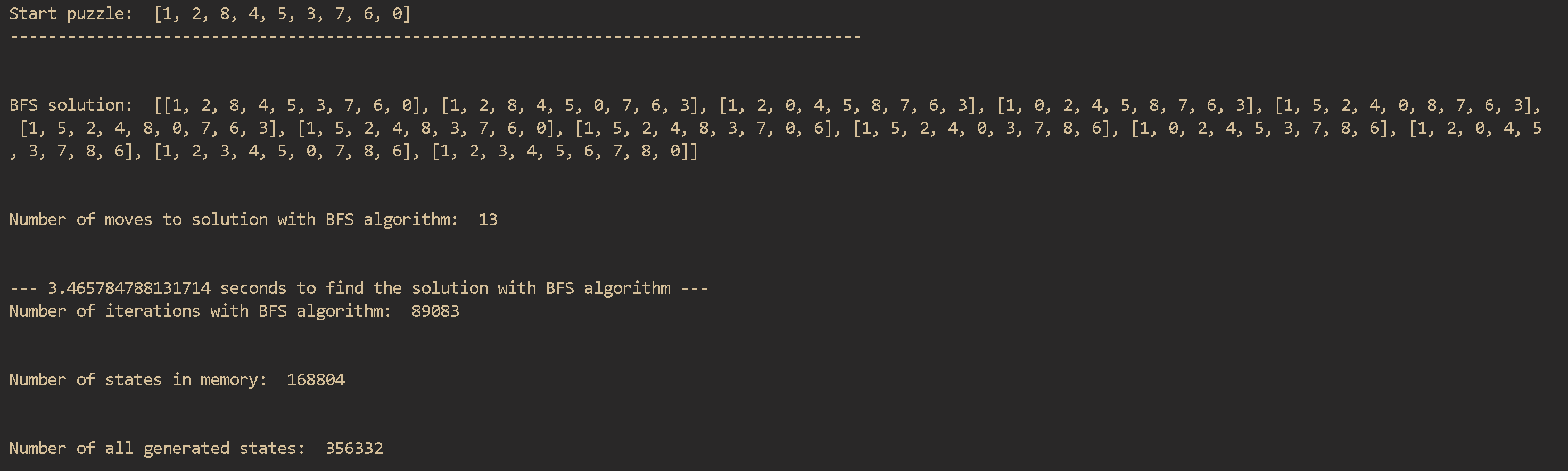
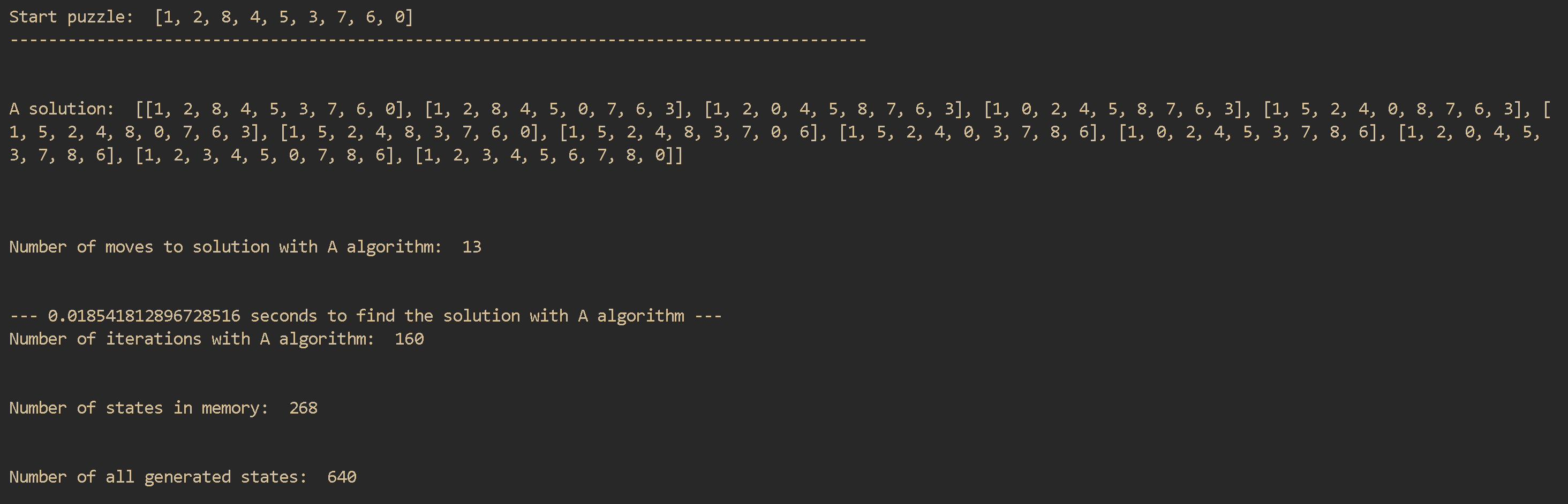


Рисунок 3.2 – Алгоритм A\*



## 1.3 Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму BFS, задачі 8 Puzzle Game для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання BFS

|  |  |  |  |
| --- | --- | --- | --- |
| Початкові стани | Ітерації | Всього станів | Всього станів у пам’яті |
| [1, 2, 3, 4, 5, 8, 0, 6, 7] | 10392 | 41568 | 19614 |
| [1, 2, 3, 5, 8, 6, 7, 4, 0] | 488738 | 1954952 | 902578 |
| [1, 0, 3, 5, 2, 6, 7, 8, 4] | 338941 | 1355764 | 640434 |
| [1, 2, 3, 8, 5, 4, 7, 6, 0] | 490469 | 1961876 | 906040 |
| [1, 2, 3, 8, 7, 6, 5, 4, 0] | 410914 | 1643656 | 746930 |
| [1, 2, 3, 0, 5, 6, 8, 4, 7] | 43929 | 175716 | 83178 |
| [1, 2, 0, 4, 5, 6, 7, 3, 8] | 8347 | 33388 | 15524 |
| [1, 2, 3, 7, 5, 6, 0, 8, 4] | 5410 | 21640 | 9650 |
| [1, 2, 3, 4, 6, 0, 5, 8, 7] | 536876 | 2147504 | 1017774 |
| [1, 3, 6, 4, 5, 2, 7, 8, 0] | 881 | 3524 | 1616 |
| [1, 2, 3, 5, 0, 6, 7, 8, 4] | 132970 | 531880 | 247218 |
| [1, 2, 4, 5, 3, 6, 7, 8, 0] | 611554 | 2446216 | 1148210 |
| [1, 2, 3, 4, 5, 8, 6, 7, 0] | 51352 | 205408 | 93342 |
| [5, 2, 3, 1, 4, 6, 7, 8, 0] | 11035 | 44140 | 20900 |
| [1, 2, 8, 4, 5, 3, 7, 6, 0] | 89083 | 356332 | 168804 |
| [1, 5, 3, 4, 0, 6, 7, 8, 2] | 147897 | 591588 | 277072 |
| [0, 2, 3, 4, 5, 6, 7, 1, 8] | 67816 | 271264 | 126270 |
| [2, 4, 3, 1, 5, 6, 7, 8, 0] | 997 | 3988 | 1848 |
| [0, 1, 3, 4, 5, 6, 7, 8, 2] | 44913 | 179652 | 80464 |
| [6, 2, 3, 1, 5, 4, 7, 8, 0] | 448725 | 1794900 | 822552 |

За результатами тестів 20 станів алгоритмом BFS середні значення такі:

Середня кількість ітерацій: 197061

Середня кількість згенерованих станів: 788247

Середня кількість станів у пам’яті: 366500

В таблиці 3.2 наведені характеристики оцінювання алгоритмуА\*, задачі 8 Puzzle Game для 20 початкових станів.

Таблиця 3.2 – Характеристики оцінювання алгоритму А\*

|  |  |  |  |
| --- | --- | --- | --- |
| Початкові стани | Ітерації | Всього станів | Всього станів у пам’яті |
| [1, 8, 2, 0, 5, 6, 7, 3, 4] | 453 | 1812 | 744 |
| [6, 4, 3, 7, 0, 1, 5, 2, 8] | 1034 | 4136 | 1689 |
| [1, 6, 8, 0, 5, 3, 4, 2, 7] | 271 | 1084 | 450 |
| [6, 0, 3, 1, 4, 8, 7, 5, 2] | 71 | 284 | 123 |
| [4, 7, 1, 8, 6, 3, 2, 0, 5] | 979 | 3916 | 1592 |
| [6, 2, 0, 7, 3, 8, 5, 1, 4] | 306 | 1224 | 497 |
| [7, 5, 1, 6, 2, 0, 4, 3, 8] | 699 | 2796 | 1163 |
| [7, 1, 3, 5, 0, 8, 6, 2, 4] | 247 | 988 | 411 |
| [4, 0, 1, 8, 6, 5, 7, 3, 2] | 122 | 488 | 212 |
| [0, 7, 8, 6, 1, 3, 2, 4, 5] | 1710 | 6840 | 2795 |
| [6, 1, 4, 5, 2, 7, 3, 0, 8] | 1184 | 4736 | 1977 |
| [1, 3, 8, 4, 2, 7, 6, 0, 5] | 243 | 972 | 406 |
| [8, 0, 1, 5, 7, 4, 2, 3, 6] | 710 | 2840 | 1139 |
| [8, 3, 7, 5, 0, 1, 6, 4, 2] | 611 | 2444 | 1011 |
| [7, 8, 1, 2, 5, 0, 3, 4, 6] | 399 | 1596 | 652 |
| [7, 2, 3, 1, 4, 6, 0, 8, 5] | 849 | 3396 | 1380 |
| [0, 7, 6, 8, 5, 3, 1, 2, 4] | 1000 | 4000 | 1669 |
| [3, 8, 1, 7, 0, 4, 2, 6, 5] | 391 | 1564 | 660 |
| [6, 7, 2, 3, 4, 0, 5, 1, 8] | 462 | 1848 | 769 |
| [7, 0, 5, 6, 2, 8, 1, 3, 4] | 555 | 2220 | 916 |

За результатами тестів 20 станів алгоритмом А\* середні значення такі:

Середня кількість ітерацій: 614

Середня кількість згенерованих станів: 2459

Середня кількість станів у пам’яті: 1012

Висновок

При виконанні даної лабораторної роботи було розглянуто та досліджено два алгоритму пошуку рішення задачі 8 Puzzle Game (BFS та A\*). По результатам тестів можна сказати, що неінформативному алгоритму пошуку BFS потрібно набагато більше ітерацій, пам’яті та відповідно часу. Інформативний алгоритм пошуку A\*, який використовує еврестичну функцію, яка рахує кількість фішок, які стоять не на своїх місцях працює набагато швидше (в залежності від стартового пазлу різниця в 5-100+ разів), але в деяких випадках не зважаючи на швидкість алгоритму А\*, він знаходить рішення (наприклад )в 40 ходів, коли алгоритм пошуку BFS працює значно довше, але знаходить рішення (наприклад) в 16 ходів.

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%.