**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 Шоман Данило*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов Олексій Олександрович*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc125014109)

[2 ЗаВдання 4](#_Toc125014110)

[3 Виконання 8](#_Toc125014111)

[3.1 Псевдокод алгоритмів 8](#_Toc125014112)

[3.2 Програмна реалізація 9](#_Toc125014113)

[3.2.1 Вихідний код 9](#_Toc125014114)

[3.2.2 Приклади роботи 14](#_Toc125014115)

[3.3 Дослідження алгоритмів 14](#_Toc125014116)

[Висновок 23](#_Toc125014117)

[Критерії оцінювання 24](#_Toc125014118)

# 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 BFS

**function BFS\_search**(initial\_state) **do**

start\_node **=** Puzzle(state=initial\_state, parent=None, action=None, path\_cost=0)

**if** (node is goal) **do**

**return** start\_node, None

**end if**

q = **Queue**

**add** start\_node **to** q

explored = []

**while not** (q **is empty**)

node = **get element from** q

**add** (state **of** node) **to end of explored**

children = **generate child of** node

instances\_in\_memory **= len of** explored

**for** (child **in** children) **do**

**if (**state **of** child **not in** explored**)**

**if (**child is goal**)**

**return** actions to achieve the goal **for** child

**add** child **to** q

**end if**

**end if**

**end for**

**end while**

**end function**

3.1.2 RBFS

**function** **recursive\_best\_first\_search** (initial\_state) **do**

node = **RBFS\_search**(Puzzle(state=initial\_state, parent=None, action=None, path\_cost=0, needs\_hueristic=True), f\_limit=maxsize)

node = node[0]

**return** actions to achieve the goal

**end function**

**function RBFS\_search**(node, f\_limit) **do**

successors = []

**if** (node is goal) **do**

**return** node, None

**end if**

children = **generate child of** node

**if not** (**len** children):

**return** None, maxsize

**end if**

count = -1

**for** (child **in** children) **do**

count += 1

 **add** (evaluation\_function **of** child, count, child) **to end of** \_ successors

**end for**

instances\_in\_memory **= len of** successors

**while** (**len** **of** successors):

**sort** successors

best\_node = successors[0][2]

**if** (evaluation\_function **of** best\_node > f\_limit) **do**

**return** None, evaluation\_function **of** best\_node

**end if**

 alternative = successors[1][0]

result, best\_node.evaluation\_function = **RBFS\_search**(best\_node,

min **min**(f\_limit, alternative))

 successors[0] = (best\_node.evaluation\_function, successors[0][1], best\_node)

**if** (result != None) **do**

**break**

**end if**

**end while**

**return** result, None

**end function**

## Програмна реалізація

### Вихідний код

**Main.py**

from time import time  
from BFS import breadth\_first\_search  
from RBFS import recursive\_best\_first\_search  
from puzzle import Puzzle  
  
  
state=[[1, 3, 5,  
 0, 4, 2,  
 7, 8, 6],  
  
 [5, 1, 3,  
 4, 2, 6,  
 0, 7, 8],  
  
 [2, 3, 5,  
 1, 8, 0,  
 4, 7, 6],  
  
 [1, 5, 2,  
 7, 4, 8,  
 0, 6, 3],  
  
 [4, 1, 0,  
 7, 6, 3,  
 5, 2, 8],  
  
 [0, 3, 6,  
 1, 8, 5,  
 4, 7, 2],  
  
 [3, 5, 8,  
 1, 6, 2,  
 4, 0, 7],  
  
 [1, 3, 8,  
 6, 5, 0,  
 4, 2, 7],  
  
 [2, 0, 8,  
 4, 1, 3,  
 7, 6, 5],  
  
 [6, 4, 2,  
 7, 1, 8,  
 0, 3, 5],  
  
 [4, 1, 5,  
 7, 8, 2,  
 0, 6, 3],  
  
 [3, 1, 6,  
 2, 8, 0,  
 4, 5, 7],  
  
 [5, 2, 4,  
 1, 0, 7,  
 8, 6, 3],  
  
 [1, 3, 8,  
 5, 2, 0,  
 4, 6, 7],  
  
 [3, 5, 6,  
 4, 0, 8,  
 2, 1, 7],  
  
 [4, 5, 2,  
 3, 8, 1,  
 0, 7, 6],  
  
 [1, 8, 2,  
 7, 0, 3,  
 6, 4, 5],  
  
 [7, 0, 3,  
 2, 1, 8,  
 6, 4, 5],  
  
 [4, 2, 5,  
 8, 7, 1,  
 6, 0, 3],  
  
 [1, 2, 3,  
 4, 5, 6,  
 7, 8, 0],  
 ]  
  
for i in range(0,20):  
 try:  
 Puzzle.num\_of\_instances=0  
 Puzzle.instances\_in\_memory = 1  
 t0=time()  
 bfs=breadth\_first\_search(state[i])  
 t1=time()-t0  
 print('BFS:', bfs)  
 print('states:',Puzzle.num\_of\_instances)  
 print('states in memory:', Puzzle.instances\_in\_memory)  
 print('time:',t1)  
 print()  
 except:  
 print("Time is out")  
 print('states:', Puzzle.num\_of\_instances)  
 print('states in memory:', Puzzle.instances\_in\_memory)  
 print('time:', t1)  
 print()  
 try:  
 Puzzle.num\_of\_instances = 0  
 Puzzle.instances\_in\_memory = 1  
 t0 = time()  
 RBFS = recursive\_best\_first\_search(state[i])  
 t1 = time() - t0  
 print('RBFS:',RBFS)  
 print('states:', Puzzle.num\_of\_instances)  
 print('states in memory:', Puzzle.instances\_in\_memory)  
 print('time:', t1)  
 print()  
 except:  
 print("Time is out")  
 print('states:', Puzzle.num\_of\_instances)  
 print('states in memory:', Puzzle.instances\_in\_memory)  
 print('time:', t1)  
 print()  
  
 print('------------------------------------------')

**Puzzle.py**

class Puzzle:  
 goal\_state = [1,2,3,4,5,6,7,8,0]  
 heuristic = None  
 evaluation\_function = None  
 needs\_hueristic = False  
 num\_of\_instances = 0  
 instances\_in\_memory = 1  
 def \_\_init\_\_(self,state,parent,action,path\_cost,needs\_hueristic=False):  
 self.parent=parent  
 self.state=state  
 self.action=action  
 if parent:  
 self.path\_cost = parent.path\_cost + path\_cost  
 else:  
 self.path\_cost = path\_cost  
 if needs\_hueristic:  
 self.needs\_hueristic = True  
 self.generate\_heuristic()  
 self.evaluation\_function=self.heuristic+self.path\_cost  
 Puzzle.num\_of\_instances+=1  
  
 def \_\_str\_\_(self):  
 return str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.state[6:9])  
  
 def generate\_heuristic(self):  
 self.heuristic=0  
 for num in range(1,9):  
 distance=abs(self.state.index(num) - self.goal\_state.index(num))  
 i=int(distance/3)  
 j=int(distance%3)  
 self.heuristic=self.heuristic+i+j  
  
 def goal\_test(self):  
 if self.state == self.goal\_state:  
 return True  
 return False  
  
 @staticmethod  
 def find\_legal\_actions(i,j):  
 legal\_action = ['U', 'D', 'L', 'R']  
 if i == 0:  
 legal\_action.remove('U')  
 elif i == 2:  
 legal\_action.remove('D')  
 if j == 0:  
 legal\_action.remove('L')  
 elif j == 2:  
 legal\_action.remove('R')  
 return legal\_action  
  
 def generate\_child(self):  
 children=[]  
 x = self.state.index(0)  
 i = int(x / 3)  
 j = int(x % 3)  
 legal\_actions=self.find\_legal\_actions(i,j)  
  
 for action in legal\_actions:  
 new\_state = self.state.copy()  
 if action == 'U':  
 new\_state[x], new\_state[x-3] = new\_state[x-3], new\_state[x]  
 elif action == 'D':  
 new\_state[x], new\_state[x+3] = new\_state[x+3], new\_state[x]  
 elif action == 'L':  
 new\_state[x], new\_state[x-1] = new\_state[x-1], new\_state[x]  
 elif action == 'R':  
 new\_state[x], new\_state[x+1] = new\_state[x+1], new\_state[x]  
 children.append(Puzzle(new\_state,self,action,1,self.needs\_hueristic))  
 return children  
  
 def find\_solution(self):  
 solution = []  
 solution.append(self.action)  
 path = self  
 while path.parent != None:  
 path = path.parent  
 solution.append(path.action)  
 solution = solution[:-1]  
 solution.reverse()  
 return solution

**BFS.py**

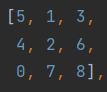
from queue import Queue  
from puzzle import Puzzle  
from func\_timeout import func\_timeout  
  
def rbfs(initial\_state):  
 return func\_timeout(1800, breadth\_first\_search, args=[initial\_state])  
  
def breadth\_first\_search(initial\_state):  
 start\_node = Puzzle(initial\_state, None, None, 0)  
 if start\_node.goal\_test():  
 return start\_node.find\_solution()  
 q = Queue()  
 q.put(start\_node)  
 explored=[]  
 while not(q.empty()):  
 node=q.get()  
 explored.append(node.state)  
 children=node.generate\_child()  
 Puzzle.instances\_in\_memory = len(explored)  
 for child in children:  
 if child.state not in explored:  
 if child.goal\_test():  
 return child.find\_solution()  
 q.put(child)  
 return

**RBFS.py**

from puzzle import Puzzle  
from sys import maxsize  
from func\_timeout import func\_timeout  
  
def rbfs(initial\_state):  
 return func\_timeout(1800, recursive\_best\_first\_search, args=[initial\_state])  
  
def recursive\_best\_first\_search(initial\_state):  
 node=RBFS\_search(Puzzle(state=initial\_state, parent=None, action=None, path\_cost=0, needs\_hueristic=True), f\_limit=maxsize)  
 node=node[0]  
 return node.find\_solution()  
  
def RBFS\_search(node,f\_limit):  
 successors=[]  
 result = None  
 if node.goal\_test():  
 return node,None  
 children=node.generate\_child()  
 if not len(children):  
 return None, maxsize  
 count=-1  
 for child in children:  
 count+=1  
 successors.append((child.evaluation\_function,count,child))  
 Puzzle.instances\_in\_memory = len(successors)  
 while len(successors):  
 successors.sort()  
 best\_node=successors[0][2]  
 if best\_node.evaluation\_function > f\_limit:  
 return None, best\_node.evaluation\_function  
 alternative=successors[1][0]  
 result,best\_node.evaluation\_function=RBFS\_search(best\_node,min(f\_limit,alternative))  
 successors[0]=(best\_node.evaluation\_function,successors[0][1],best\_node)  
 if result!=None:  
 break  
 return result,None

### Приклади роботи

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



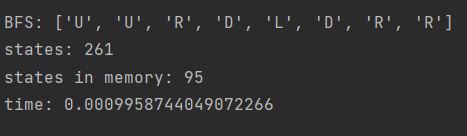


Рисунок 3.1 – Алгоритм BFS

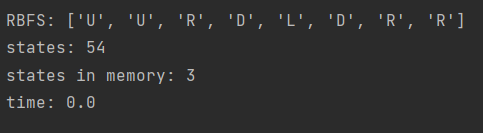


Рисунок 3.2 – Алгоритм RBFS

## Дослідження алгоритмів

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

Таблиця 3.1 – Характеристики оцінювання BFS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| Стан 1 | ['R', 'R', 'U', 'L', 'D', 'R', 'D'] | 0 | 282 | 100 |
| Стан 2 | ['U', 'U', 'R', 'D', 'L', 'D', 'R', 'R'] | 0 | 261 | 95 |
| Cтан 3 | ['U', 'L', 'L', 'D', 'D', 'R', 'U', 'R', 'D'] | 0 | 573 | 213 |
| Стан 4 | ['U', 'R', 'R', 'D', 'L', 'U', 'U', 'R', 'D', 'D'] | 0 | 976 | 360 |
| Стан 5 | ['D', 'L', 'D', 'L', 'U', 'U', 'R', 'D', 'D', 'R'] | 0 | 898 | 334 |
| Стан 6 | ['D', 'D', 'R', 'U', 'R', 'D', 'L', 'U', 'R', 'U', 'L', 'D', 'D', 'R'] | 0 | 5905 | 2205 |
| Стан 7 | ['U', 'U', 'L', 'D', 'R', 'R', 'U', 'L', 'D', 'D', 'R', 'U', 'L', 'L', 'D', 'R', 'R'] | 0 | 28257 | 10583 |
| Стан 8 | ['L', 'D', 'R', 'U', 'U', 'L', 'D', 'L', 'D', 'R', 'R', 'U', 'L', 'D', 'R'] | 0 | 15370 | 5560 |
| Стан 9 | ['R', 'D', 'D', 'L', 'L', 'U', 'R', 'U', 'L', 'D', 'D', 'R', 'U', 'R', 'D'] | 0 | 15766 | 5692 |
| Стан 10 | ['U', 'U', 'R', 'D', 'D', 'R', 'U', 'L', 'L', 'U', 'R', 'R', 'D', 'L', 'D', 'R'] | 0 | 15124 | 5666 |
| Стан 11 | ['U', 'U', 'R', 'R', 'D', 'D', 'L', 'U', 'U', 'R', 'D', 'D'] | 0 | 2194 | 814 |
| Стан 12 | ['L', 'U', 'L', 'D', 'R', 'D', 'R', 'U', 'U', 'L', 'L', 'D', 'D', 'R', 'R'] | 0 | 14020 | 5110 |
| Стан 13 | ['R', 'U', 'L', 'L', 'D', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'U', 'R', 'D', 'D'] | 0 | 29007 | 10471 |
| Стан 14 | ['U', 'L', 'D', 'D', 'R', 'U', 'L', 'L', 'D', 'R', 'R'] | 0 | 1452 | 542 |
| Стан 15 | ['D', 'L', 'U', 'R', 'U', 'L', 'D', 'R', 'D', 'R', 'U', 'U', 'L', 'D', 'L', 'D', 'R', 'R'] | 0 | 57349 | 21169 |
| Стан 16 | ['R', 'U', 'R', 'U', 'L', 'D', 'L', 'U', 'R', 'D', 'R', 'U', 'L', 'D', 'R', 'D'] | 0 | 21676 | 7850 |
| Стан 17 | ['D', 'L', 'U', 'R', 'U', 'R', 'D', 'D', 'L', 'U', 'R', 'D'] | 0 | 3206 | 1188 |
| Стан 18 | ['D', 'L', 'U', 'R', 'D', 'D', 'L', 'U', 'R', 'D', 'R', 'U', 'L', 'D', 'R'] | 0 | 11437 | 4249 |
| Стан 19 | ['L', 'U', 'R', 'R', 'U', 'L', 'D', 'R', 'D', 'L', 'L', 'U', 'U', 'R', 'R', 'D', 'D'] | 0 | 38994 | 14162 |
| Стан 20 | [] | 0 | 1 | 1 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму BFS, задачі 8-puzzle для 20 початкових станів.

Таблиця 3.3 – Характеристики оцінювання RBFS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пом’яті |
| Стан 1 | ['R', 'R', 'U', 'L', 'D', 'R', 'D'] | 0 | 23 | 3 |
| Стан 2 | ['U', 'U', 'R', 'D', 'L', 'D', 'R', 'R'] | 0 | 54 | 3 |
| Cтан 3 | ['U', 'L', 'L', 'D', 'D', 'R', 'U', 'R', 'D'] | 0 | 26 | 3 |
| Стан 4 | ['U', 'R', 'R', 'D', 'L', 'U', 'U', 'R', 'D', 'D'] | 0 | 38 | 3 |
| Стан 5 | ['D', 'L', 'D', 'L', 'U', 'U', 'R', 'D', 'D', 'R'] | 0 | 30 | 3 |
| Стан 6 | ['D', 'D', 'R', 'U', 'R', 'D', 'L', 'U', 'R', 'U', 'L', 'D', 'D', 'R'] | 0 | 3778 | 3 |
| Стан 7 | ['U', 'U', 'L', 'D', 'R', 'R', 'U', 'L', 'D', 'D', 'R', 'U', 'L', 'L', 'D', 'R', 'R'] | 0 | 716 | 3 |
| Стан 8 | ['L', 'D', 'R', 'U', 'U', 'L', 'D', 'L', 'D', 'R', 'R', 'U', 'L', 'D', 'R'] | 0 | 255 | 3 |
| Стан 9 | ['R', 'D', 'D', 'L', 'L', 'U', 'R', 'U', 'L', 'D', 'D', 'R', 'U', 'R', 'D'] | 0 | 2805 | 3 |
| Стан 10 | ['U', 'U', 'R', 'D', 'D', 'R', 'U', 'L', 'L', 'U', 'R', 'R', 'D', 'L', 'D', 'R'] | 0 | 52 | 3 |
| Стан 11 | ['U', 'U', 'R', 'R', 'D', 'D', 'L', 'U', 'U', 'R', 'D', 'D'] | 0 | 33 | 3 |
| Стан 12 | ['L', 'U', 'L', 'D', 'R', 'D', 'R', 'U', 'U', 'L', 'L', 'D', 'D', 'R', 'R'] | 0 | 303 | 3 |
| Стан 13 | ['R', 'U', 'L', 'L', 'D', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'U', 'R', 'D', 'D'] | 0 | 7696 | 3 |
| Стан 14 | ['U', 'L', 'D', 'D', 'R', 'U', 'L', 'L', 'D', 'R', 'R'] | 0 | 33 | 3 |
| Стан 15 | ['D', 'L', 'U', 'R', 'U', 'L', 'D', 'R', 'D', 'R', 'U', 'U', 'L', 'D', 'L', 'D', 'R', 'R'] | 0 | 845 | 3 |
| Стан 16 | ['R', 'U', 'R', 'U', 'L', 'D', 'L', 'U', 'R', 'D', 'R', 'U', 'L', 'D', 'R', 'D'] | 0 | 8138 | 3 |
| Стан 17 | ['D', 'L', 'U', 'R', 'U', 'R', 'D', 'D', 'L', 'U', 'R', 'D'] | 0 | 78 | 3 |
| Стан 18 | ['D', 'L', 'U', 'R', 'D', 'D', 'L', 'U', 'R', 'D', 'R', 'U', 'L', 'D', 'R'] | 0 | 453 | 3 |
| Стан 19 | ['L', 'U', 'R', 'R', 'U', 'L', 'D', 'R', 'D', 'L', 'L', 'U', 'U', 'R', 'R', 'D', 'D'] | 0 | 3286 | 3 |
| Стан 20 | [] | 0 | 1 | 1 |

Висновок

При виконанні даної лабораторної роботи було розглянуто алгоритми неінформативного (BFS), інформативного та локального пошуку (RBFS), було проведено порівняльний аналіз ефективності використання алгоритмів. «RBFS» виявився ефективнішим за «BFS» для заданої задачі «8-puzzle» у всіх заданих початкових станах. Кількість станів, та кількість станів, збережених у пам’яті усюди булі менше у RBFS. Також додатково було проведено тестування часу роботи алгоритмів і виявилось, що у найгірших випадках RBFS мав час виконання до 1 секунд, а BFS аж до 13 секунд.

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%.