**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-11 Трикош Іван*

**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 |

# Виконання

## Псевдокод алгоритмів

**Пошук з поверненням (backtracking)**

Graph - граф (матриця суміжності)

edge - вершина, яка розглядається

edges – список вершин, які вже зафарбовані, та їх кольорів

colors – кольори, в які можна зафарбувати вершину

COLORS – кольори для фарбування

function Backtracking(Graph, edge, edges, colors, COLORS):

for each i in colors do:

if edges.Length == Graph.Length:

break

end if

edges[edge] = i

edge, colours = mrv(Graph, edges, COLORS)

if edge == -1:

break

end if

edges = Backtracking(Graph, edge, edges, colours, COLORS)

end for each

return edges

end functions Backtracking

function mrv(Graph, edges, COLORS):

numbers = {}

for i = 0 to Graph.Length - 1 do:

if !(i in edges):

numbers[i] = COLORS

for j = 0 to Graph.Length - 1 do:

if Graph[i][j] == 1:

if j in edges:

if edges[j] in numbers[i]:

numbers[i].delete(edges[j])

end if

end if

end if

end for

end if

end for

min = COLORS.Length + 1

index = -1

for each i in numbers do:

if len(numbers[i]) < min:

min = numbers[i].Length

index = i

end if

end for each

if min == 0:

return -1, []

end if

if min == COLORS.Length + 1:

return -1, []

end if

return index, numbers[index]

end function mrv

**Сходження до вершини з рухом убік**

Graph - граф (матриця суміжності)

edge - вершина, яка розглядається

edges – список кольорів, у які зафарбовані вершини

colors – кольори для фарбування

function Hill(Graph, edge, edges, colors):

flag = True

flags = []

for i = 0 to edges.Length - 1 do:

flags[i] = True

end for

while flag == True:

if flags[edge] == True:

for each color in colors do:

if check(Graph, edge, edges, color):

edges[edge] = color

flags[edge] = False

break

end if

end for

if flags[edge] == True:

flag = False

end if

else:

temp = edge

for i = 0 to edges.Length - 1 do:

if Graph[edge][i] == 1:

if flags[i] == True:

edge = i

break

end if

end if

end for

if temp == edge:

flag = False

end if

end if

end while

return edges

end function Hill

function check(Graph, edge, edges, color):

for i = 0 to Graph[edge].Length - 1 do:

if Graph[edge][i] == 1:

if color == edges[i]:

return False

end if

end if

end for

return True

end function check

function side\_movement(Graph, edge, edges, colors):

for i = 0 to 99 do:

colours = []

for each color in colors do:

if check(Graph, edge, edges, color) == True:

colours += [color]

end if

end for each

if colours.Length > 0:

flag = True

for each i in colours:

temp = edges

temp[edge] = i

if number\_of\_colors(temp) < number\_of\_colors(edges):

edges[edge] = i

flag = False

break

end if

end for each

if flag == True:

colour=randomchoice(colours)/\*Випадковий вибір\*/

edges[edge] = colour

if number\_of\_colors(edges) <= 4:

break

Edges = []

for i = 0 to edges.Length - 1 do:

if Graph[edge][i] == 1:

Edges += [i]

end if

end for

edge = randomchoice(Edges) /\* Випадковий вибір \*/

return edges

end function side\_movement

function number\_of\_colors(edges):

colors = []

for each i in edges:

if !(i in colors):

colors += [i]

end if

end for each

return colors.Length

end function number\_of\_colors

## Програмна реалізація

### Вихідний код

**main.py**

import Functional

def main():

if int(input("Choose algorithm (0 - Hill, 1 - Backtracking): ")) == 0:

test = Functional.Algorithm\_Hill()

test.drow\_graph(test.EDGES)

test.set\_random\_Start\_edge()

test.start()

test.drow\_graph(test.Result)

else:

test = Functional.Algorithm\_Backtracking()

test.set\_random\_Start\_edge()

test.start()

test.drow\_graph(test.Result)

return

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Functional.py**

import random

import networkx as nx

import matplotlib.pyplot as plt

class General:

"""Клас із загальними полями та методами"""

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

"""Конструктор"""

self.GRAPH = [

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0],

[0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1],

[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1],

[0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1],

[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0],

[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],

[1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],

[0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]]

self.EDGES = ["red", "blue", "yellow", "green", "silver", "gray", "maroon", "olive", "lime", "aqua",

"teal", "navy", "fuchsia", "purple", "indianred", "khaki", "salmon", "peachpuff", "lightyellow",

"chocolate", "firebrick", "azure", "coral", "orange", "tan"]

self.COLORS = ["red", "blue", "yellow", "green"]

self.POSITION = {1: (600, 550), 2: (346, 281), 3: (196, 94), 4: (608, 310), 5: (747, 314), 6: (348, 144), 7: (101, 302),

8: (668, 387), 9: (175, 289),

10: (442, 188), 11: (487, 307), 12: (800, 228), 13: (150, 215), 14: (486, 383), 15: (421, 421),

16: (560, 210), 17: (276, 122), 18: (565, 104),

19: (228, 234), 20: (665, 200), 21: (560, 430), 22: (285, 235), 23: (450, 250), 24: (237, 332),

25: (490, 84)}

self.Start\_edge = self.Edge = 0

self.Result = None

return

def drow\_graph(self, edges):

"""Малювання графу"""

Graph = nx.Graph()

plt.gca().invert\_yaxis()

Graph.add\_nodes\_from(self.POSITION.keys())

for i in range(len(self.GRAPH)):

for j in range(i):

if self.GRAPH[i][j]:

Graph.add\_edge(i + 1, j + 1)

for n, p in self.POSITION.items():

Graph.nodes[n]['pos'] = p

nx.draw(Graph, node\_color=edges, with\_labels=True, pos=self.POSITION)

plt.show()

return

def set\_random\_Start\_edge(self):

"""Початкова вершина (початковий стан)"""

self.Start\_edge = self.Edge = random.randint(0, len(self.GRAPH) - 1)

return self.Start\_edge

def set\_Start\_edge(self, edge):

"""Початкова вершина (початковий стан)"""

self.Start\_edge = self.Edge = random.randint(0, len(self.GRAPH) - 1)

return self.Start\_edge

def number\_of\_colors(self, edges):

"""Кількість кольорів"""

colors = []

for i in edges:

if not i in colors:

colors += [i]

return len(colors)

class Algorithm\_Hill(General):

"""Алгоритм сходження до вершини"""

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

"""Конструктор"""

super().\_\_init\_\_()

random.shuffle(self.EDGES)

return

def check(self, index, edges, color):

"""Перевірка фарбування"""

for i in range(len(self.GRAPH[index])):

if self.GRAPH[index][i] == 1:

if color == edges[i]:

return False

return True

def Hill(self):

"""Сходження до вершини"""

edges = self.EDGES.copy()

flag = True

flags = [True for i in range(len(edges))]

while flag:

if flags[self.Edge]:

for color in self.COLORS:

if self.check(self.Edge, edges, color):

edges[self.Edge] = color

flags[self.Edge] = False

break

if flags[self.Edge]:

flag = False

else:

temp = self.Edge

for i in range(len(edges)):

if self.GRAPH[self.Edge][i] == 1:

if flags[i]:

self.Edge = i

break

if temp == self.Edge:

flag = False

return edges

def side\_movement(self, edges):

"""Рух убік"""

for i in range(100):

colours = []

for color in self.COLORS:

if self.check(self.Edge, edges, color):

colours += [color]

if len(colours) > 0:

flag = True

for i in colours:

temp = edges.copy()

temp[self.Edge] = i

if self.number\_of\_colors(temp) < self.number\_of\_colors(edges):

edges[self.Edge] = i

flag = False

break

if flag:

colour = random.choice(colours)

edges[self.Edge] = colour

if self.number\_of\_colors(edges) <= 4:

break

Edges = []

for i in range(len(edges)):

if self.GRAPH[self.Edge][i] == 1:

Edges += [i]

self.Edge = random.choice(Edges)

return edges

def start(self):

"""Запуск алгоритму"""

self.Result = self.EDGES.copy()

for i in range(10):

edges = self.Hill()

if self.number\_of\_colors(edges) > 4:

for j in range(5):

edges = self.side\_movement(edges.copy())

if self.number\_of\_colors(edges) < self.number\_of\_colors(self.Result):

self.Result = edges

if self.number\_of\_colors(edges) <= 4:

break

if self.number\_of\_colors(edges) <= 4:

self.Result = edges

break

else:

break

self.set\_random\_Start\_edge()

return self.Result

class Algorithm\_Backtracking(General):

"""Алгоритм з пошуку поверненням (backtracking)"""

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

"""Конструктор"""

super().\_\_init\_\_()

return

def from\_dict\_to\_list(self, result):

"""Конвертація із словника у список"""

res = []

for i in range(len(self.GRAPH)):

if i in result:

res += [result[i]]

else:

for color in self.EDGES:

if not color in res:

res += [color]

break

return res

def mrv(self, edges):

"""Визначення фарбування вузла"""

numbers = {}

for i in range(len(self.GRAPH)):

if not i in edges:

numbers[i] = self.COLORS.copy()

for j in range(len(self.GRAPH)):

if self.GRAPH[i][j]:

if j in edges:

if edges[j] in numbers[i]:

numbers[i].remove(edges[j])

min = len(self.COLORS) + 1

index = None

for i in numbers:

if len(numbers[i]) < min:

min = len(numbers[i])

index = i

if min == 0 or min == 5:

return -1, []

return index, numbers[index]

def Backtracking(self, edge, edges, colors):

"""Пошук з поверненням"""

for i in colors:

if len(edges) == len(self.GRAPH):

self.Result = edges

break

edges[edge] = i

edge, colours = self.mrv(edges)

if edge == -1:

if len(self.Result) < len(edges):

self.Result = edges

break

edges = self.Backtracking(edge, edges, colours)

return edges

def start(self):

"""Запуск алгоритму"""

self.Result = {}

result = self.Backtracking(self.Start\_edge, {}, self.COLORS.copy())

self.Result = self.from\_dict\_to\_list(result)

return self.Result

### Приклади роботи

Обрана країна – *Україна*.

На рисунках 3.1-3.3 показані приклади роботи програми для різних алгоритмів пошуку.

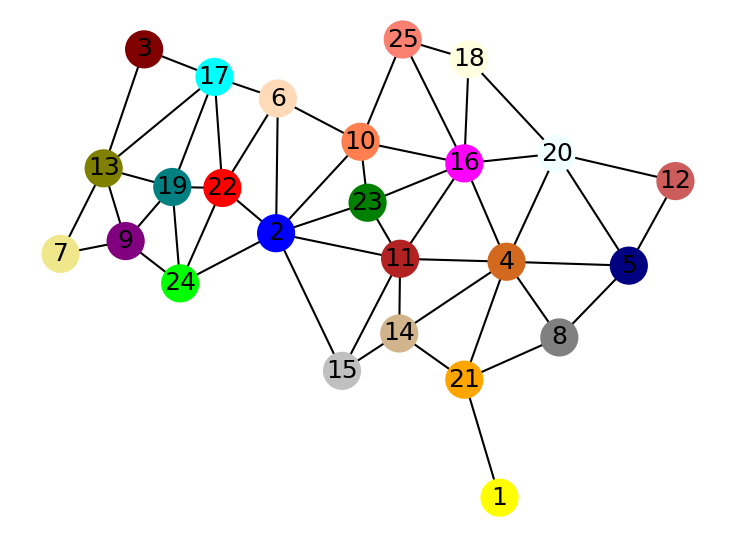


Рисунок 3.1 – Алгоритм сходження до вершини (початковий стан)

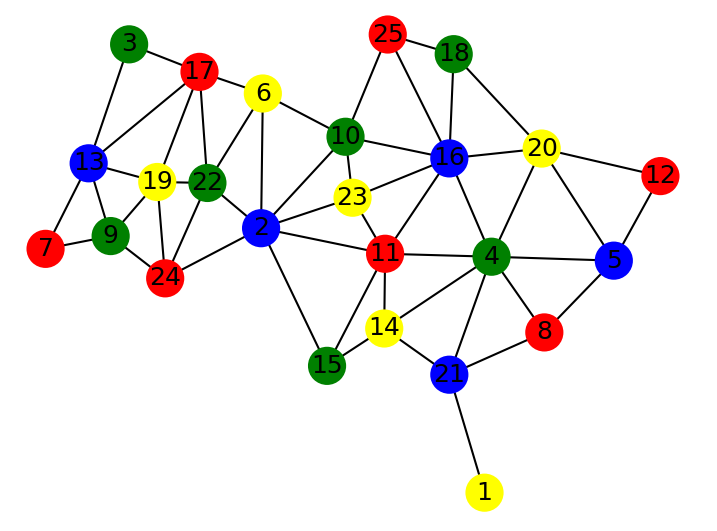


Рисунок 3.2 – Алгоритм сходження до вершини (результат)

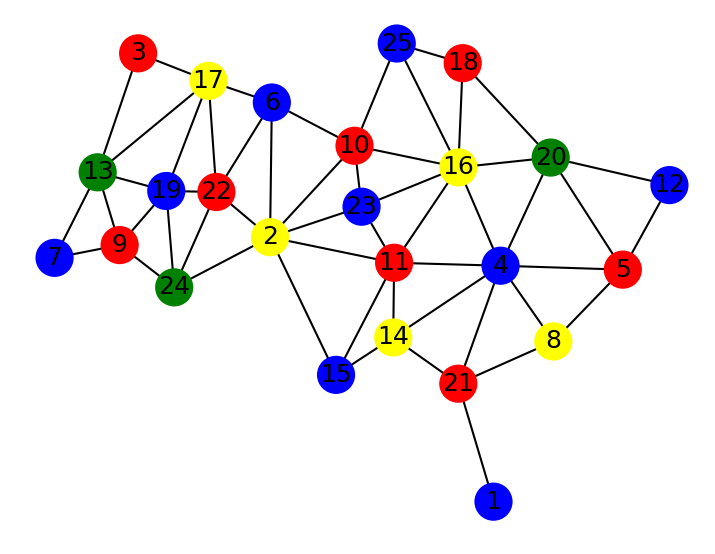


Рисунок 3.3 – Алгоритм пошуку з поверненням (backtracking)

## Дослідження алгоритмів

В таблиці 3.1 наведені характеристики оцінювання алгоритму сходження до вершини, задачі фарбування карти України для 20 початкових станів.

Таблиця 3.1 – Характеристики оцінювання алгоритму сходження до вершини

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пам’яті |
| Стан 1 | 27268 | 3 | 299 | 1 |
| Стан 2 | 19781 | 3 | 299 | 1 |
| Стан 3 | 20817 | 3 | 222 | 1 |
| Стан 4 | 24888 | 3 | 280 | 1 |
| Стан 5 | 46322 | 5 | 509 | 1 |
| Стан 6 | 10788 | 2 | 124 | 1 |
| Стан 7 | 8712 | 1 | 104 | 1 |
| Стан 8 | 5056 | 1 | 63 | 1 |
| Стан 9 | 36758 | 4 | 411 | 1 |
| Стан 10 | 21630 | 3 | 244 | 1 |
| Стан 11 | 24045 | 3 | 274 | 1 |
| Стан 12 | 21295 | 3 | 241 | 1 |
| Стан 13 | 12521 | 2 | 139 | 1 |
| Стан 14 | 18214 | 2 | 209 | 1 |
| Стан 15 | 45261 | 5 | 506 | 1 |
| Стан 16 | 7058 | 1 | 87 | 1 |
| Стан 17 | 15585 | 2 | 172 | 1 |
| Стан 18 | 13619 | 2 | 154 | 1 |
| Стан 19 | 13837 | 2 | 157 | 1 |
| Стан 20 | 16823 | 2 | 178 | 1 |

В таблиці 3.2 наведені характеристики оцінювання алгоритму пошуку з поверненням (backtracking), задачі фарбування карти України для 20 початкових станів.

Таблиця 3.2 – Характеристики оцінювання алгоритму пошуку з поверненням (backtracking)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Початкові стани | Ітерації | К-сть гл. кутів | Всього станів | Всього станів у пам’яті |
| Стан 1 | 8471 | 1 | 46 | 25 |
| Стан 2 | 8470 | 1 | 45 | 25 |
| Стан 3 | 8471 | 1 | 46 | 25 |
| Стан 4 | 8471 | 1 | 46 | 25 |
| Стан 5 | 8471 | 1 | 46 | 25 |
| Стан 6 | 8470 | 1 | 45 | 25 |
| Стан 7 | 8470 | 1 | 45 | 25 |
| Стан 8 | 8471 | 1 | 46 | 25 |
| Стан 9 | 8470 | 1 | 45 | 25 |
| Стан 10 | 8470 | 1 | 45 | 25 |
| Стан 11 | 8471 | 1 | 46 | 25 |
| Стан 12 | 8470 | 1 | 45 | 25 |
| Стан 13 | 8471 | 1 | 46 | 25 |
| Стан 14 | 8471 | 1 | 46 | 25 |
| Стан 15 | 8471 | 1 | 46 | 25 |
| Стан 16 | 8471 | 1 | 46 | 25 |
| Стан 17 | 8471 | 1 | 46 | 25 |
| Стан 18 | 8470 | 1 | 45 | 25 |
| Стан 19 | 8470 | 1 | 45 | 25 |
| Стан 20 | 8471 | 1 | 46 | 25 |

Висновок

При виконанні даної лабораторної роботи було розглянуто алгоритми локального та неінформативного пошуку, такі як сходження до вершини з рухом убік та випадковим перезапуском і пошук з поверненням (backtracking), використовуючи евристику мінімальної кількості значень. Для написання алгоритмів я обрав мову Python.

Для розв’язання задачі я обрав Україну (25 областей). Початковий стан – випадкова вершина.

За таблицями 3.1 та 3.2 можна побачити, що пошук з поверненням є ефективнішим за пошук зі сходженням до вершини, проте він використовує більше пам’яті.

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%.