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 № 1 з дисципліни
«ПІІС»

«Неінформативний, інформативний та локальний пошук»

Виконав(ла) ІТ-02 Тригуб Діана ІТ-02. _____
(шифр, прізвище, ім'я, по батькові)

Перевірив _____ _____
(прізвище, ім'я, по батькові)

Київ 2021

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – розглянути та дослідити алгоритми неінформативного, інформативного та локального пошуку. Провести порівняльний аналіз ефективності використання алгоритмів.

ВИКОНАННЯ

1.1 Lee Search

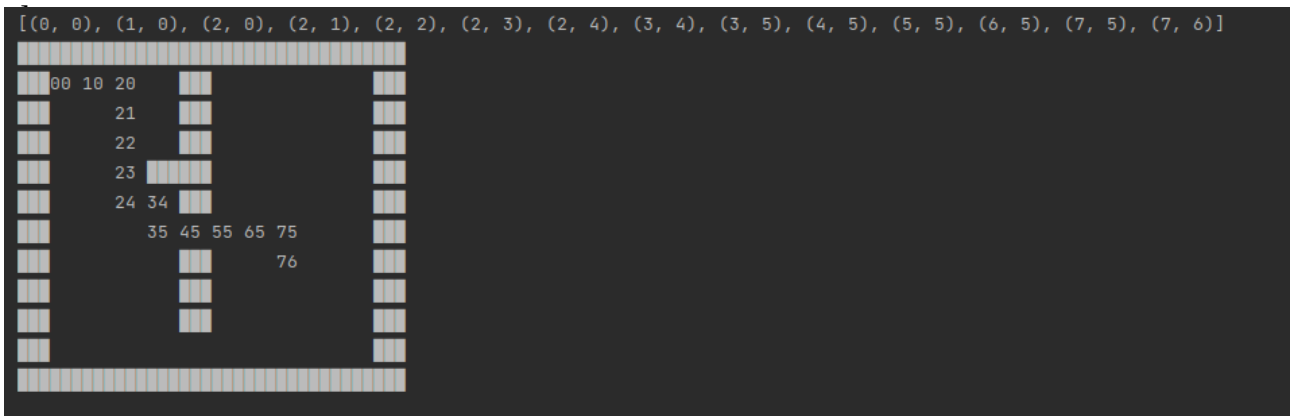
Код алгоритму:

```
from collections import deque


class Cell:
    def __init__(self, x: int, y: int):
        self.x = x
        self.y = y


class Node:
    def __init__(self, pt: Cell, dist: int, parent):
        self.pt = pt  # coordinates
        self.dist = dist  # distance from the source
        self.parent = parent


def is_valid(row: int, col: int):
    return (row >= 0) and (row < rows) and (col >= 0) and (col < cols)


def lee(mat, src: Cell, dest: Cell):
    if mat[src.x][src.y] != 1 or mat[dest.y][dest.x] != 1:
        print(mat[src.x][src.y])
        print(mat[dest.x][dest.y])
        return -1
```

```
visited = [[False for i in range(cols)]
          for j in range(rows)]

# Mark the source cell as visited
visited[src.x][src.y] = True

# Create a queue for BFS
q = deque()

s = Node(src, 0, None)
q.append(s)

while q:

    curr = q.popleft()

    pt = curr.pt
    if pt.x == dest.x and pt.y == dest.y:
        return curr

    for i in [(0, -1), (0, 1), (-1, 0), (1, 0)]:
        row = pt.y + i[0]
        col = pt.x + i[1]

        # Enqueue valid adjacent cell that is not visited
        if (is_valid(row, col) and
                mat[row][col] == 1 and
                not visited[row][col]):
            visited[row][col] = True
            Adjcell = Node(Cell(col, row),
```

```
                    curr.dist + 1, curr)
            q.append(Adjcell)

    return -1


def draw_maze(matrix, path):
    wall = b'\xdb'.decode('cp437')
    space = ' '
    n = 3
    print(wall * n * (len(matrix[0]) + 2))
    for i in range(len(matrix)):
        print(wall * n, end='')
        for j in range(len(matrix[0])):
            if matrix[i][j] == 0:
                print(wall * n, end='')
            else:
                if (j, i) in path:
                    print(f'{j}{i} ', end='')
                else:
                    print(space * n, end='')
        print(wall * n)

    print(wall * n * (len(matrix[0]) + 2))

mat = [[1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 0, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
```

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


rows = len(mat)
cols = len(mat[0])


source = Cell(0, 0)
dest = Cell(7, 6)

path_node = lee(mat, source, dest)


if path_node.dist != -1:
    print("Length of the Shortest Path is", path_node.dist)
    path = []
    while path_node:
        path.append((path_node.pt.x, path_node.pt.y))
        path_node = path_node.parent
    path.reverse()
    print(path)
    draw_maze(mat, path)
```



Алгоритм є неінформативним, однак в ситуації, коли під рукою більше нічого нема є досить дієвим, та зі своєю задачею справляється. Алгоритм “під капотом” оснований на BFS, що дозволяє нам стверджувати, що знайдений нами шлях є оптимальним. Однак 3 іншого боку, така система не може працювати на зваженому графі.

1.2 A Star

Код алгоритму:

```
class Node():


    def __init__(self, parent=None, state=None):
        self.g = 0
        self.h = 0
        self.f = 0
        self.parent = parent
        self.state = state


    def __eq__(self, other):
        return self.state == other.state


    def count_g(self, node):
        if self.state[0] == self.parent.state[0] or self.state[1] == self.parent.state[1]:
            self.g = node.g + 10
        else:
            self.g = node.g + 14


    def count_h(self, end):
        self.h = (abs(self.state[0] - end.state[0]) + abs(self.state[1] - end.state[1])) *
10


    def count_f(self):
        self.f = self.g + self.h



def A(maze, start, end):
```

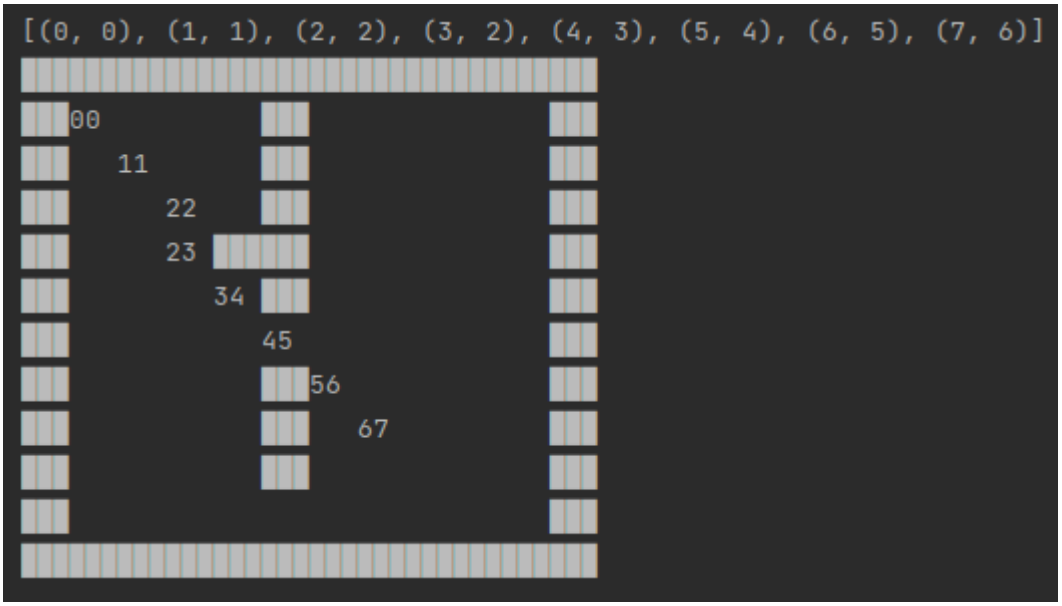
```
    open_list = []
    closed_list = []

    open_list.append(start)

    while open_list:

        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index

        open_list.pop(current_index)
        closed_list.append(current_node)

        if current_node == end:
            path = []
            current = current_node
            while current is not None:
                path.append(current.state)
                current = current.parent
            return path[::-1] # Return reversed path

        # Generate children
        children = []
        for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]:
# Adjacent squares

            # Get node position
            node_position = (current_node.state[0] + new_position[0],
current_node.state[1] + new_position[1])

            # Make sure within range
            if node_position[0] > (len(maze) - 1) or node_position[0] < 0 or
node_position[1] > (len(maze[len(maze)-1]) -1) or node_position[1] < 0:
                continue
```

```
            # Make sure walkable terrain
            if maze[node_position[0]][node_position[1]] != 1:
                continue

            # Create new node
            new_node = Node(current_node, node_position)

            # Append
            children.append(new_node)

        # Loop through children
        for child in children:

            # Child is on the closed list
            for closed_child in closed_list:
                if child == closed_child:
                    continue

            # Create the f, g, and h values
            child.count_g(current_node)
            child.count_h(end)
            child.count_f()

            # Child is already in the open list
            for open_node in open_list:
                if child == open_node and child.g > open_node.g:
                    continue

            # Add the child to the open list
            open_list.append(child)


def draw_maze(matrix, path):
    wall = b'\xdb'.decode('cp437')
    space = ' '
    n = 3
    print(wall * n * (len(matrix[0]) + 2))
```

```
    for i in range(len(matrix)):
        print(wall * n, end='')
        for j in range(len(matrix[0])):
            if matrix[i][j] == 0:
                print(wall * n, end='')
            else:
                if (i, j) in path:
                    print(f'{j}{i} ', end='')
                else:
                    print(space * n, end='')
        print(wall * n)

    print(wall * n * (len(matrix[0]) + 2))



def main(begin, end):
    mat = [[1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 0, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

    start = Node(None, begin)
    end_ = Node(None, end)

    path = A(mat, start, end_)
    print(path)

    draw_maze(mat, path)



if __name__ == '__main__':
```

```
main((0, 0), (7, 6))
```

Результат:



A*

базується все на тому ж BFS, однак з одним дуже суттєвим покращенням. В A Star наступна в ітерації вершина обирається не случайно, а опираючись на оцінку цієї вершини (оцінка складається з евристики та відстанні до цієї вершини). Таким чином на кожному кроці ми обираємо найбільш вирогідно правильний крок, що суттєво прискорює знаходження шляху.

ВИСНОВОК

Висновок щодо кожного алгоритму окремо я напиала в кінці підрозділів про кожний алгоритм.