LAPORAN PRAKTIKUM
KECERDASAN BUATAN
"METODE PENCARIAN"

NAMA : **AFRIDHO IKHSAN**
NPM : **2210631170002**
KELAS : **3A**

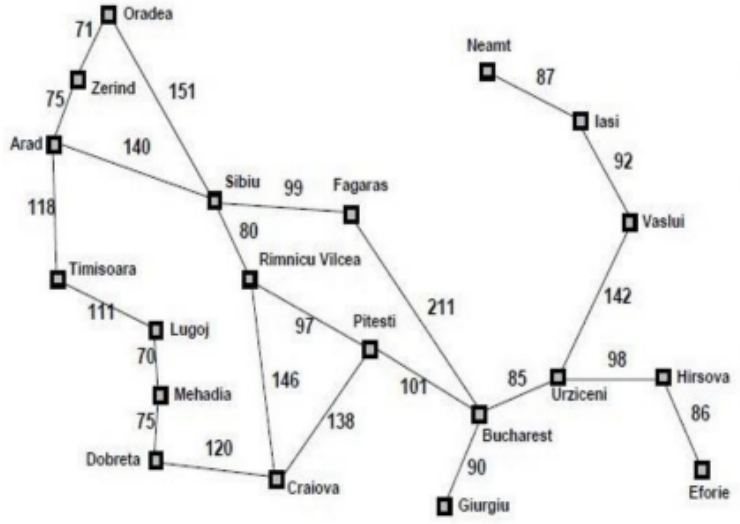
PROGRAM STUDI INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS SINGAPERBANGSA KARAWANG

2023

DAFTAR ISI

LATIHAN

Kasus 1



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

a. Bagaimana rute perjalanan dari *Arad* ke *Bucarest*
b. Gunakan teknik pencarian *Breadth – First Search* & *Depth – First Search*

Kasus 2

Sebuah *puzzle* berukuran 3X3

Nilai awal:

1	2	3
8		4
7	6	5

Goal:

2	8	3
1	6	4
7		5

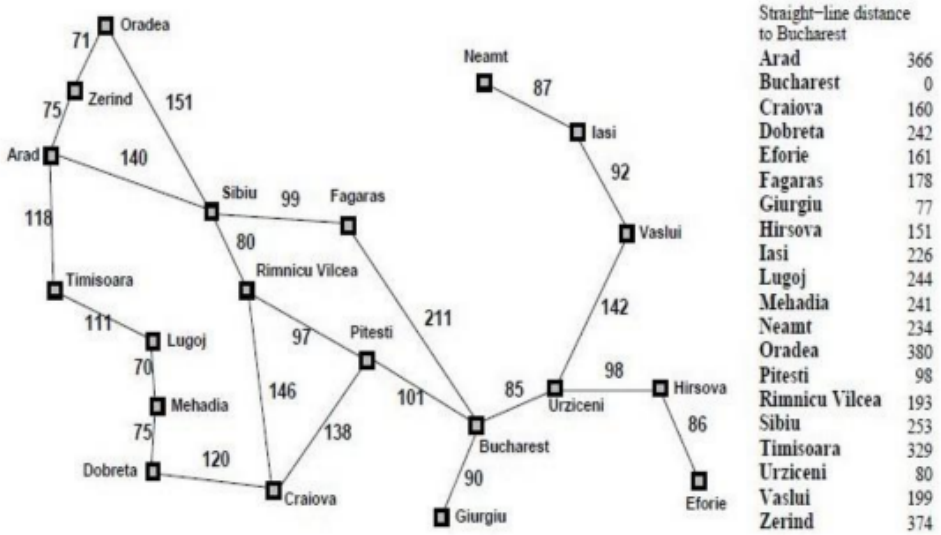
f(n) = g(n) + h(n)
g(n) = kedalaman pohon
h(n) = jumlah angka yang salah posisi
Kerjakan dengan Teknik Best First Search!

JAWABAN

Nomor 1 (a)

```
from IPython.display import Image
Image(filename='kasus1.jpg')
```

Output:



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

```
graph = {
    'Oradea' : [('Zerind', 71), ('Sibiu', 151)],
    'Zerind' : [('Oradea', 71), ('Arad', 75)],
    'Arad' : [('Zerind', 75), ('Sibiu', 140), ('Timisoara',
118)],
    'Timisoara' : [('Arad', 118), ('Lugoj', 111)],
    'Lugoj' : [('Timisoara', 111), ('Mehadia', 70)],
    'Mehadia' : [('Lugoj', 70), ('Dobreta', 75)],
    'Dobreta' : [('Mehadia', 75), ('Craiova', 120)],
    'Sibiu' : [('Oradea', 151), ('Arad', 140), ('Fagaras',
99), ('Rimnicu Vilcea', 80)],
    'Rimnicu Vilcea' : [('Sibiu', 80), ('Craiova', 146),
('Pitesti', 97)],
    'Craiova' : [('Dobreta', 120), ('Rimnicu Vilcea', 146),
('Pitesti', 138)],
    'Fagaras' : [('Sibiu', 99), ('Bucharest', 211)],
    'Pitesti' : [('Rimnicu Vilcea', 97), ('Craiova', 138),
('Bucharest', 101)],
    'Bucharest' : [('Fagaras', 211), ('Pitesti', 101),
('Giurgiu', 90), ('Urziceni', 85)],
    'Giurgiu' : [('Bucharest', 90)],
    'Neamt' : [('Lasi', 87)],
    'Lasi' : [('Neamt', 87), ('Vaslui', 92)],
    'Vaslui' : [('Lasi', 92), ('Urziceni', 142)],
```

```
    'Urziceni' : [('Bucharest', 85), ('Vaslui', 142),
('Hirsova', 98)],
    'Hirsova' : [('Urziceni', 98), ('Eforie', 86)],
    'Eforie' : [('Hirsova', 86)],
}

H_table = {
    'Arad': 366,
    'Bucharest': 0,
    'Craiova': 160,
    'Dobreta': 242,
    'Eforie': 161,
    'Fagaras': 178,
    'Giurgiu': 77,
    'Hirsova': 151,
    'Lasi': 226,
    'Lugoj': 244,
    'Mehadia': 241,
    'Neamt': 234,
    'Oradea': 380,
    'Pitesti': 98,
    'Rimnicu Vilcea': 193,
    'Sibiu': 253,
    'Timisoara': 329,
    'Urziceni': 90,
    'Vaslui': 199,
    'Zerind': 374
}
```

```
def path_h_cost(path):
  g_cost = 0
  for (node, cost) in path:
    g_cost += cost
    last_node = path[-1][0]
    h_cost = H_table[last_node]
    f_cost = g_cost + h_cost
    return h_cost, last_node
```

```
def Greedy_best_search(graph, start, goal):
  visited =[]
  queue = [[(start,0)]]

  while queue:
    queue.sort(key=path_h_cost)
    path = queue.pop(0)
    node = path[-1][0]
```

```
    if node in visited:
      continue
    visited.append(node)
    if node == goal:
      return path
    else:
      adjacent_nodes = graph.get(node, [])
      for (node2, cost) in adjacent_nodes:
        new_path = path.copy()
        new_path.append((node2, cost))
        queue.append(new_path)
```

```
rute = Greedy_best_search(graph, 'Arad', 'Bucharest')
print ('Rute menggunakan teknik Greedy Best First Search
adalah ', rute)
```

Output:

Rute menggunakan teknik Greedy Best First Search adalah [('Arad', 0), ('Sibiu', 140), ('Fagaras', 99), ('Bucharest', 211)]

Nomor 1 (b)

- BFS (Breadth – First Search)

```
from collections import deque

def Breadth_First_Search(graph, start, goal):
    visited = set()
    queue = deque([(start, [])])

    while queue:
        node, path = queue.popleft()

        if node not in visited:
            visited.add(node)
            path = path + [(node, 0)]

            if node == goal:
                return path
            else:
                adjacent_nodes = graph.get(node, [])
                queue.extend((neighbor, path + [(neighbor,
cost)]) for neighbor, cost in adjacent_nodes)

    return None

bfs_solution = Breadth_First_Search(graph, 'Arad',
'Bucharest')
```

```
print('Rute hasil dari penggunaan teknik BFS adalah ',
bfs_solution)
```

Output:

Rute hasil dari penggunaan teknik BFS adalah [('Arad', 0), ('Sibiu', 140), ('Sibiu', 0), ('Fagaras', 99), ('Fagaras', 0), ('Bucharest', 211), ('Bucharest', 0)]

- DFS (Depth – First Search)

```
def Depth_First_Search(graph, start, goal):
    visited = set()
    stack = [(start, [])]

    while stack:
        node, path = stack.pop()

        if node not in visited:
            visited.add(node)
            path = path + [(node, 0)]

            if node == goal:
                return path
            else:
                adjacent_nodes = graph.get(node, [])
                stack.extend((neighbor, path + [(neighbor,
cost)]) for neighbor, cost in adjacent_nodes)

    return None

dfs_solution = Depth_First_Search(graph, 'Arad', 'Bucharest')
print('Rute hasil dari penggunaan teknik DFS adalah',
dfs_solution)
```

Output:

Rute hasil dari penggunaan teknik DFS adalah [('Arad', 0), ('Timisoara', 118), ('Timisoara', 0), ('Lugoj', 111), ('Lugoj', 0), ('Mehadia', 70), ('Mehadia', 0), ('Dobreta', 75), ('Dobreta', 0), ('Craiova', 120), ('Craiova', 0), ('Pitesti', 138), ('Pitesti', 0), ('Bucharest', 101), ('Bucharest', 0)]

Nomor 2

```
from IPython.display import Image
Image(filename='kasus2.jpg')
```

Output:

Nilai awal:

1	2	3
8		4
7	6	5

Goal:

2	8	3
1	6	4
7		5

```
class NodePuzzle:
    def __init__(self, state, kedalaman, induk=None):
        self.state = state
        self.kedalaman = kedalaman
        self.induk = induk

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

    def __hash__(self):
        return hash(str(self.state))

    def __str__(self):
        return "\n".join([" | ".join(map(str, row)) for row in
self.state])

def h_cost(state, goal_state):
    return sum(1 for i, j in zip(state, goal_state) if i != j)

def posisi_kosong(state):
    for i, baris in enumerate(state):
        for j, nilai in enumerate(baris):
            if nilai is None:
                return i, j

def dapatkan_tetangga(node):
    tetangga = []
    i, j = posisi_kosong(node.state)
    gerakan = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    for geser in gerakan:
        i_baru, j_baru = i + geser[0], j + geser[1]

        if 0 <= i_baru < 3 and 0 <= j_baru < 3:
            state_baru = [baris.copy() for baris in
node.state]
            state_baru[i][j], state_baru[i_baru][j_baru] =
state_baru[i_baru][j_baru], state_baru[i][j]
```

```
            tetangga.append(NodePuzzle(state_baru,
node.kedalaman + 1, node))

    return tetangga

def pencarian_a_star(state_awal, state_goal):
    node_awal = NodePuzzle(state_awal, 0)
    node_goal = NodePuzzle(state_goal, float('inf'))

    himpunan_terbuka = {node_awal}
    himpunan_tertutup = set()

    while himpunan_terbuka:
        node_saat_ini = min(himpunan_terbuka, key=lambda x:
x.kedalaman + h_cost(x.state, state_goal))
        himpunan_terbuka.remove(node_saat_ini)

        if node_saat_ini == node_goal:
            path = []
            while node_saat_ini:
                path.append(node_saat_ini.state)
                node_saat_ini = node_saat_ini.induk
            return reversed(path)

        himpunan_tertutup.add(node_saat_ini)

        tetangga_node = dapatkan_tetangga(node_saat_ini)
        for tetangga in tetangga_node:
            if tetangga not in himpunan_tertutup and tetangga
not in himpunan_terbuka:
                himpunan_terbuka.add(tetangga)

    return None

# Kondisi awal
state_awal = [
    [1, 2, 3],
    [8, None, 4],
    [7, 6, 5]
]

# Goal state
state_goal = [
    [2, 8, 3],
    [1, 6, 4],
    [7, None, 5]
]
```

```
solusi_pencarian = pencarian_a_star(state_awal, state_goal)

if solusi_pencarian:
    print("Solusi Ditemukan:")
    for langkah, state in enumerate(solusi_pencarian):
        print(f"Langkah {langkah + 1}:\n{NodePuzzle(state,
0)}\n")
else:
    print("Tidak ditemukan solusi.")
```

Output:
Solusi Ditemukan:
Langkah 1:
1 | 2 | 3
8 | None | 4
7 | 6 | 5

Langkah 2:
1 | 2 | 3
None | 8 | 4
7 | 6 | 5

Langkah 3:
None | 2 | 3
1 | 8 | 4
7 | 6 | 5

Langkah 4:
2 | None | 3
1 | 8 | 4
7 | 6 | 5

Langkah 5:
2 | 8 | 3
1 | None | 4
7 | 6 | 5

Langkah 6:
2 | 8 | 3
1 | 6 | 4
7 | None | 5