**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

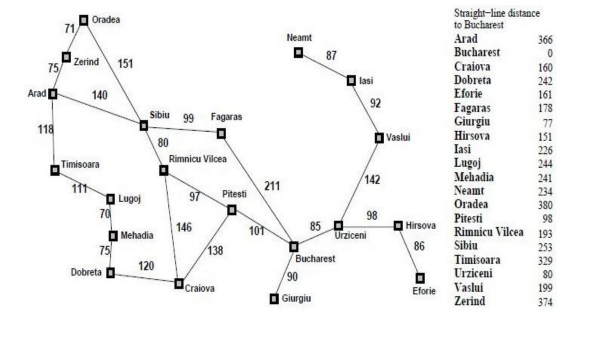
[DAFTAR ISI i](#_Toc160649625)

[LATIHAN 2](#_Toc160649626)

[JAWABAN 3](#_Toc160649627)

# LATIHAN

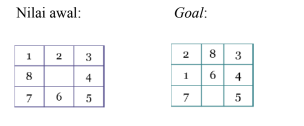
***Kasus 1***

******

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

***Kasus 2***

Sebuah *puzzle* berukuran 3X3

**

f(n) = g(n) + h(n)

g(n) = kedalaman pohon

h(n) = jumlah angka yang salah posisi

Kerjakan dengan Teknik Best First Search!

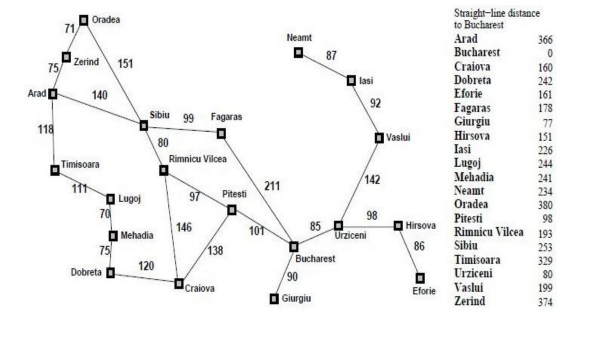
# JAWABAN

**Nomor 1 (a)**

from IPython.display import Image

Image(filename='kasus1.jpg')

Output:



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:

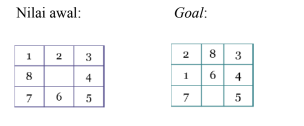
Rute hasil dari penggunaan teknik DFS adalah [('Arad', 0), ('Timisoara', 118), ('Timisoara', 0), ('Lugoj', 111), ('Lugoj', 0), ('Mehadia', 70), ('Mehadia', 0), ('Dobreta', 75), ('Dobreta', 0), ('Craiova', 120), ('Craiova', 0), ('Pitesti', 138), ('Pitesti', 0), ('Bucharest', 101), ('Bucharest', 0)]

**Nomor 2**

from IPython.display import Image

Image(filename='kasus2.jpg')

Output:



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