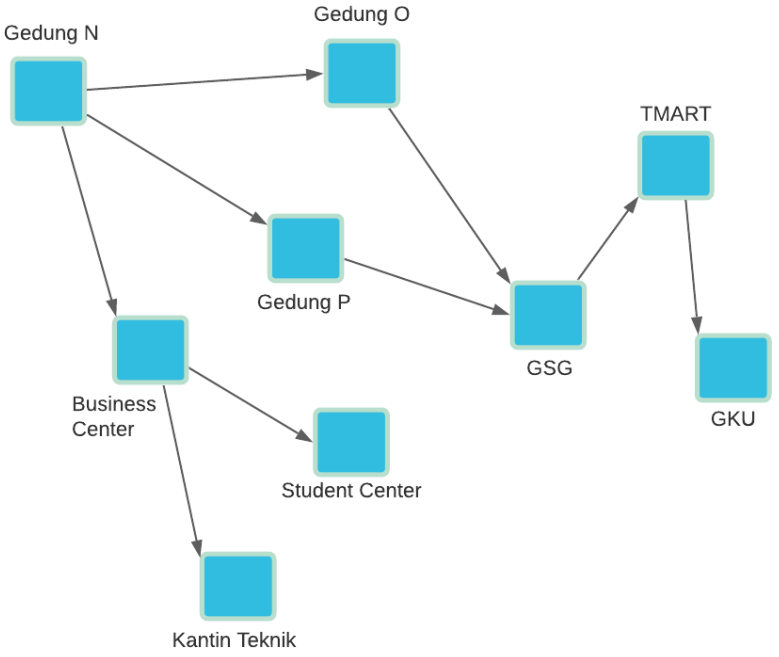
JURNAL

MODUL 1

SEARCHING

A. UNINFORMED SEARCH

1. Diketahui sebuah peta dengan gambar sebagai berikut:



Dengan peta yang ada, tentukan proses perjalanan dari **Gedung N** sampai **GKU** menggunakan DFS dan BFS secara **manual**.

- DFS :
  - Gedung N – Business Center – Kantin Teknik – Student Center
  - Gedung P – GSG – TMART – **GKU**

- BFS :

  Langkah 1 = Business Center – Gedung P – Gedung O

  Langkah 2 = Kantin Teknik – Student Center – GSG

  Langkah 3 = TMART - **GKU**

2. Buatlah sebuah *project* bernama ‘coba DFS’ lalu cobalah program dengan

source code di bawah ini:

```
#inisialisasi nama dan nomer gedung
namanya = {
    1:'Gedung N',
    2:'Business Center',
    3:'Kantin Teknik',
    4:'Student Center',
    5:'Gedung P',
    6:'GSG',
    7:'TMART',
    8:'GKU',
    9:'Gedung O'
}
```

```
#inisialisasi hubungan antar tempat menggunakan dictionary
cost = {
    1: {1: 0, 2: 1, 3: 0, 4: 0, 5: 1, 6: 0, 7: 0, 8: 0, 9: 1},
    2: {1: 1, 2: 0, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0},
    3: {1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0},
    4: {1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0},
    5: {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0},
    6: {1: 0, 2: 0, 3: 0, 4: 0, 5: 1, 6: 0, 7: 1, 8: 0, 9: 1},
    7: {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 1, 9: 0},
    8: {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 1, 8: 0, 9: 0},
    9: {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0},
}

if __name__=="__main__":
    n = 10
    visit = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] #inisialisasi
tempat yang sedang dilewati menggunakan dictionary
    visited = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] #inisialisasi
tempat yang sudah dilewati menggunakan dictionary
    stk = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] #inisialisasi Stack
menggunakan dictionary
    #Print Nomor dan nama Gedung
    for i in range(1, n):
        print(f"{namanya[i]} = {i}")
    #Print hubungan antar gedung
    print('\nHubungan Antar Gedung:')
    for i in range(1, n):
        for j in range(1, n):
            if cost[i][j] == 1:
                print(f'{namanya[i]} ke {namanya[j]}')

    #Input rute
    v = int(input('\nMasukan Initial State: '))
    s = int(input('Masukan Final State: '))
    #Hasil Rute yang dilewati
    print(f'\nRute:\n{namanya[v]}')
    #DFS
    visited[v] = 1
    k = 1
    top = 0
    while(k<n and k!=s):
        for i in range(n-1, 0, -1): # pengecekan rute
```

```
            if cost[v][i] != 0 and visited[i] != 1 and visit[i]
!= 1:
                visit[i] = 1
                stk[top] = i
                top += 1

        top -= 1
        v = stk[top]

        #Menampilkan rute yang dilewati
        print(namanya[v])

        k += 1
        visit[v] = 0
        visited[v] = 1
```

3. *Running* program, masukkan *input* Gedung N (1) sebagai *Initial State* dan GKU (8) sebagai *Final State*. Bandingkan hasilnya dengan perhitungan manual sebelumnya. Apakah sama atau tidak? Jelaskan.

```
s-python.python-2022.2.1924087327\pythonFiles\lib\python\debugpy\launcher' '64413' '--' 'c:\Users\pusak\OneDrive\Documents\Kuliah\Semester 6\Praktikum Kecerdasan Bua
tan\Modul 1\DFS.py'
Gedung N = 1
Business Center = 2
Kantin Teknik = 3
Student Center = 4
Gedung P = 5
GSG = 6
TMART = 7
GKU = 8
Gedung O = 9

Hubungan Antar Gedung:
Gedung N ke Business Center
Gedung N ke Gedung P
Gedung N ke Gedung O
Business Center ke Gedung N
Business Center ke Kantin Teknik
Business Center ke Student Center
Kantin Teknik ke Business Center
Student Center ke Business Center
Gedung P ke Gedung N
Gedung P ke GSG
GSG ke Gedung P
GSG ke TMART
GSG ke Gedung O
TMART ke GSG
TMART ke GKU
GKU ke TMART
Gedung O ke Gedung N
Gedung O ke GSG

Masukan Initial State: 1
Masukan Final State: 8


Business Center
Kantin Teknik
Student Center
Gedung P
GSG
TMART
GKU
```

Hasil ouput dari program sama dengan hasil dari perhitungan manual.

4. Buatlah sebuah project bernama ‘coba BFS’, kemudian cobalah program dengan *source code* di bawah ini:

```
#inisialisasi nama dan nomer gedung
namanya = {
    1:'Gedung N',
    2:'Business Center',
    3:'Gedung P',
    4:'Gedung O',
    5:'Kantin Teknik',
    6:'Student Center',
    7:'GSG',
    8:'TMART',
    9:'GKU'
}
```

```
cost = {
    1: {1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0},
    2: {1: 1, 2: 0, 3: 0, 4: 0, 5: 1, 6: 1, 7: 0, 8: 0, 9: 0},
    5: {1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0},
    6: {1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0},
    3: {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 1, 8: 0, 9: 0},
    7: {1: 0, 2: 0, 3: 1, 4: 1, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0},
    8: {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 1, 8: 0, 9: 1},
    9: {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 1, 9: 0},
    4: {1: 1, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 1, 8: 0, 9: 0},
}
if __name__=="__main__":
    n = 10
    visit = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] #yang sedang
dilewati
    visited = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] #yang sudah
dilewati
    qu = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] #menampung daftar
tunggu queue
    for i in range(1, 10):
        print(f"{namanya[i]} = {i}")
    #hubungan antar gedung
    print('Hubungan Antar Gedung\n')
    for i in range(1, 10):
        for j in range(1, 10):
            if cost[i][j] == 1:
                print(f'{namanya[i]} ke {namanya[j]}')
v = int(input('\n\nMasukan Initial State: '))
    s = int(input('Masukan Final State: '))
#metode pencarian bfs
print(f'\nRUTE:\n{namanya[v]}')
    visited[v] = 1
    k = 1
    front = 0
    rare = 0
```

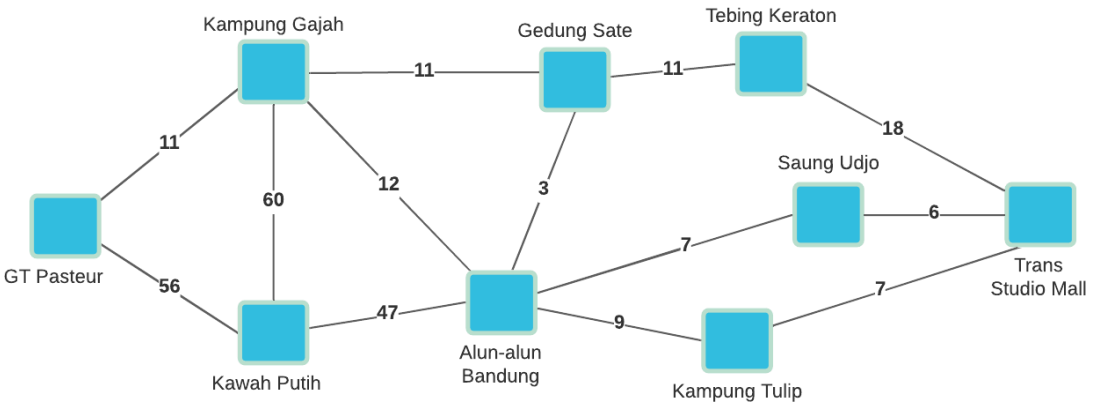
```
while(k<n and k!=s):
        for i in range(1, 10):
            if cost[v][i] != 0 and visited[i]!=1 and visit[i]!=1:
                visit[i] = 1
                rare += 1
                qu[rare] = i

        front += 1
        v = qu[front]
        print(namanya[v])

        k += 1
        visit[v] = 0
        visited[v] = 1
```

5. Running program dibawah ini, masukkan input Gedung N (1) sebagai *Initial State* dan GKU (9) sebagai *Final State*. Bandingkan hasilnya dengan perhitungan manual sebelumnya. Apakah sama atau tidak? Jelaskan.

```
tan\Modul 1\BFS.py'
Gedung N = 1
Business Center = 2
Gedung P = 3
Gedung O = 4
Kantin Teknik = 5
Student Center = 6
GSG = 7
TMART = 8
GKU = 9
Hubungan Antar Gedung

Gedung N ke Business Center
Gedung N ke Gedung P
Gedung N ke Gedung O
Business Center ke Gedung N
Business Center ke Kantin Teknik
Business Center ke Student Center
Gedung P ke Gedung N
Gedung P ke GSG
Gedung O ke Gedung N
Gedung O ke GSG
Kantin Teknik ke Business Center
Student Center ke Business Center
GSG ke Gedung P
GSG ke Gedung O
GSG ke TMART
TMART ke GSG
TMART ke GKU
GKU ke TMART


Masukan Initial State: 1
Masukan Final State: 9

RUTE:
Gedung N
Business Center
Gedung P
Gedung O
Kantin Teknik
Student Center
GSG
TMART
GKU
```

Hasil ouput dari program sama dengan hasil dari perhitungan manual.

B. INFORMED SEARCH



n	GT Pasteur	Kawah Putih	Kampung Gajah	Alun-alun Bandung	Gedung Sate	Kampung Tulip	Saung Udjo	Tebing Keraton	Trans Studio Mall
$h(n)$	274	288	261	169	143	170	82	81	0

1. Berdasarkan peta di atas, carilah rute dari **GT Pasteur** menuju **Trans Studio Mall** dengan menggunakan algoritma *Greedy Best-First Search*.

   ['GT Pasteur: Heuristic (274) Actual(0)', 'Kampung Gajah: Heuristic (261) Actual(11)', 'Gedung Sate: Heuristic (143) Actual(22)', 'Tebing Keraton: Heuristic (81) Actual(33)', 'Trans Studio

   Mall: Heuristic (0) Actual(**51**)']

2. Buatlah *project* bernama “Greedy.py” lalu ketikan *Source code* berikut ini:

```
# Class graph menunjukkan hubungan antar node
class Graph:
    # menginisialisasikan node
    def init (self, graph_dict=None, directed=True):
        self.graph_dict = graph_dict or {}
        self.directed = directed
        if not directed:
            self.make_undirected()
    # membuat graph yang tidak langsung
    def make_undirected(self):
        for a in list(self.graph_dict.keys()):
            for (b, dist) in self.graph_dict[a].items():
                self.graph_dict.setdefault(b, {})[a] = dist
    # menghubungkan dari satu graph ke graph lainnya
    def connect(self, A, B, distance=1):
        self.graph_dict.setdefault(A, {})[B] = distance
        if not self.directed:
```

```
            self.graph_dict.setdefault(B, {})[A] = distance
    # mengecek tetangga dari graph
    def get(self, a, b=None):
        links = self.graph_dict.setdefault(a, {})
        if b is None:
            return links
        else:
            return links.get(b)
    # Return list nodes didalam graph tersebut
    def nodes(self):
        s1 = set([k for k in self.graph_dict.keys()])
        s2 = set([k2 for v in self.graph_dict.values() for k2, v2
in v.items()])
        nodes = s1.union(s2)
        return list(nodes)
# class berikut untuk membuat node
class Node:
    # Inisialisasi class
    def __init__(self, name:str, parent:str):
        self.name = name
        self.parent = parent
        self.g = 0 # Jarak dari start
        self.h = 0 # Jarak ke goal
        self.f = 0 # Total cost
    # Membandingkan node
    def __eq__(self, other):
        return self.name == other.name
    # Sorting node
    def __lt__(self, other):
         return self.f < other.f
    #  Menampilkan
    def __repr__(self):
        return ('({0},{1})'.format(self.position, self.f))
# Algortima greedy
def best_first_search(graph, heuristics, start, end):

    # Membuat list untuk node terbuka dan node tertutup
    open = []
    closed = []
    # membuat start node dan goal node
    start_node = Node(start, None)
    goal_node = Node(end, None)
```

```
    # Menambahkan start node
    open.append(start_node)

    # looping untuk mengecek ketersediaan open list
    while len(open) > 0:
        # Mengurutkan list open dari yang terkecil
        open.sort()
        # menemukan node dengan node cost terkecil
        current_node = open.pop(0)
        # Memindahkan node sekarang ke list closed
        closed.append(current_node)

        # mengecek kondisi goal
        if current_node == goal_node:
            path = []
            while current_node != start_node:
                path.append(current_node.name + ': Heuristic ('+
str(heuristics.get(current_node.name)) + ") Actual(" +
str(current_node.g)+")")
                current_node = current_node.parent
            path.append(start_node.name + ': Heuristic ('+
str(heuristics.get(current_node.name)) + ") Actual(" +
str(start_node.g)+")")
            return path[::-1]
        # Mencari tetangga
        neighbors = graph.get(current_node.name)
        # Loop neighbors
        for key, value in neighbors.items():
            neighbor = Node(key, current_node)

            if(neighbor in closed):
                continue
            # menghitung total cost
            neighbor.g = current_node.g +
graph.get(current_node.name,  neighbor.name)
            neighbor.h = heuristics.get(neighbor.name)
            neighbor.f = neighbor.h
            # mengecek apabila tetangga dalam list open dan
memiliki nilai heuristic yang lebih kecil
            if(add_to_open(open, neighbor) == True):
                open.append(neighbor)
    # Return None,apabila rute tidak ditemukan
```

```
    return None
# mengecek tetangga untuk ditambahkan ke list open atau tidak
def add_to_open(open, neighbor):
    for node in open:
        if (neighbor == node and neighbor.f >= node.f):
            return False
    return True

def main():
    # membuat graph
    graph = Graph()
    # mendeklarasikan actual cost jarak antar kota
    graph.connect('GT Pasteur', 'Kawah Putih', 56)
    graph.connect('GT Pasteur', 'Kampung Gajah', 11)
    graph.connect('Kawah Putih', 'Alun-alun Bandung', 47)
    graph.connect('Kampung Gajah', 'Kawah Putih', 60)
    graph.connect('Kampung Gajah', 'Alun-alun Bandung', 12)
    graph.connect('Kampung Gajah', 'Gedung Sate', 11)
    graph.connect('Alun-alun Bandung', 'Gedung Sate', 3)
    graph.connect('Alun-alun Bandung', 'Kampung Tulip', 9)
    graph.connect('Alun-alun Bandung', 'Saung Udjo', 7)
    graph.connect('Gedung Sate', 'Tebing Keraton', 11)
    graph.connect('Kampung Tulip', 'Trans Studio Mall', 7)
    graph.connect('Saung Udjo', 'Trans Studio Mall', 6)
    graph.connect('Tebing Keraton', 'Trans Studio Mall', 18)

    # membuat graph tidak langsung
    graph.make_undirected()
    # mendeklarasikan nilai heuristic cost antar kota
    heuristics = {}
    heuristics['GT Pasteur'] = 274
    heuristics['Kawah Putih'] = 288
    heuristics['Kampung Gajah'] = 261
    heuristics['Alun-alun Bandung'] = 169
    heuristics['Gedung Sate'] = 143
    heuristics['Kampung Tulip'] = 170
    heuristics['Saung Udjo'] = 82
    heuristics['Tebing Keraton'] = 81
    heuristics['Trans Studio Mall'] = 0

    # Menu
    print("=============Program Greedy Best-First Search
```

```
Algorithm============")
    print("Masukan Asal dan Tujuan: ")
    kota_awal = int(input('Masukkankota awal:'))
    if kota_awal == 1:
      awal = "GT Pasteur"
    elif kota_awal == 2:
      awal = "Kawah Putih"
    elif kota_awal == 3:
      awal = "Kampung Gajah"
    elif kota_awal == 4:
      awal = "Gedung Sate"
    elif kota_awal == 5:
      awal = "Alun-Alun Bandung"
    elif kota_awal == 6:
      awal = "Kampung Tulip"
    elif kota_awal == 7:
      awal = "Saung Udjo"
    elif kota_awal == 8:
      awal = "Tebing Keraton"
    elif kota_awal == 9:
      awal = "Trans Studio Mall"

    kota_tujuan = int(input('Masukkan kota tujuan:'))
    if kota_tujuan == 1:
      tujuan = "GT Pasteur"
    elif kota_tujuan == 2:
      tujuan = "Kawah Putih"
    elif kota_tujuan == 3:
      tujuan = "Kampung Gajah"
    elif kota_tujuan == 4:
      tujuan = "Gedung Sate"
    elif kota_tujuan == 5:
      tujuan = "Alun-alun Bandung"
    elif kota_tujuan == 6:
      tujuan = "Kampung Tulip"
    elif kota_tujuan == 7:
      tujuan = "Saung Udjo"
    elif kota_tujuan == 8:

      tujuan = "Tebing Keraton"
    elif kota_tujuan == 9:
      tujuan = "Trans Studio Mall"
```

```
    print("\nMenampilkan Jalur, Nilai Heuristic dan Actual Cost")
    path = best_first_search(graph, heuristics, awal, tujuan)
    print(path)

if __name__ == "__main__": main()
```

```
rs\pusak\OneDrive\Documents\Kuliah\Semester 6\Praktikum Kecerdasan Buatan\Modul 1\GFS.py'
============Program Greedy Best-First Search Algorithm===========
Masukan Asal dan Tujuan:
1
9

Menampilkan Jalur, Nilai Heuristic dan Actual Cost
['GT Pasteur: Heuristic (274) Actual(0)', 'Kampung Gajah: Heuristic (261) Actual(11)', 'Gedung Sate: Heuristic (143) Actual(22)',
'Tebing Keraton: Heuristic (81) Actual(33)', 'Trans Studio Mall: Heuristic (0) Actual(51)']
```

3. Jelaskan maksud dari *class graph* dan *class node* pada source code
    - Class graph menunjukkan hubungan antar node
    - Class node adalah class untuk membuat node
4. Jelaskan hasil analisa pada *source code* proses algoritma *Greedy Best First-Search*.
    - Algoritma ini bekerja dengan mengecek ketersedian open list atau node dan mengurutkannya dari heuristic yang terkecil, dan memilih node yang memiliki heuristic terkecil.
5. Cobalah untuk memodifikasi nilai *heuristic* pada program. Apakah program tetap dapat menemukan tujuan *vertex*-nya?. Apakah memungkinkan terjadinya *Stuck In Loops* pada *Greedy Best-First Search*? Jelaskan.
    - Ketika nilai heuristic di modifikasi, program akan tetap menemukan tujuan vertexnya. Dalam *Greedy Best-First Search* memungkinkan terjadinya *Stuck In Loops* karena disebabkan oleh karakteristik dari Greedy Search itu sendiri yang mengekspansi.
6. Apakah pencarian rute menggunakan algoritma *Greedy Best-First Search* selalu menghasilkan solusi optimal? Jika tidak, mengapa?.
    - Karena *Greedy Best-First Search* hanya memilih rute dengan heuristic yang tercekil, dan nilai heuristic belum tentu lebih kecil dari actual cost .

C. KESIMPULAN

Apa kesimpulan yang Anda dapatkan setelah mengikuti praktikum ini?

Pada praktikum ini kita dan memahami cara kerja dari algoritma Informed and Uninformed Search, serta dapat memahami perbedaan dari kedua jenis algoritma pencarian tersebut.

D. KRITIK DAN SARAN

a. Laboratorium: Mantap

b. Asisten: Mantap

c. Praktikum: Mantap