MODUL PRAKTIKUM KECERDASAN BUATAN

PERTEMUAN 2 – METODE PENCARIAN

Tools : Jupyter notebook

Bahasa Pemrograman : Python

Hal penting dalam menentukan keberhasilan sistem cerdas adalah kesuksesan dalam pencarian. Pencarian merupakan suatu proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan (*state space*). Ruang keadaan merupakan suatu ruang yang berisi semua keadaan yang mungkin. Untuk mengukur perfomansi metode pencarian, terdapat empat kriteria yang dapat digunakan:

a. ***Completeness*** : apakah metode tersebut menjamin penemuan solusi jika solusinya memang ada?
b. ***Time complexity*** : berapa lama waktu yang diperlukan?
c. ***Space complexity*** : berapa banyak memori yang diperlukan?
d. ***Optimality*** : apakah metode tersebut menjamin menemukan solusi yang terbaik jika terdapat beberapa solusi berbeda?

Teknik Pencarian

1. *Blind Search* (Pecarian Buta)

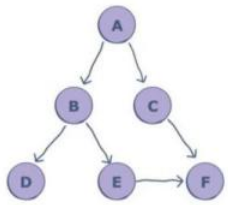
Tidak ada informasi awal yang digunakan dalam proses pencarian

1. Pencarian melebar pertama **(*Breadth – First Search*)**

   Semua node pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi node-node pada level n+1. Pencarian dimulai dari node akar terus ke level 1 dari kiri ke kanan, kemudian berpindah ke level berikutnya dari kiri ke kanan hingga solusi ditemukan.

```
In [19]: from IPython.display import Image
         Image(filename='graph.jpg')
```

Out[19]:



In [2]:
```
graph = {
  'A' : ['B','C'],
  'B' : ['D', 'E'],
  'C' : ['F'],
  'D' : [],
  'E' : ['F'],
  'F' : []
}

visited = []  # List untuk melacak node yang dikunjungi.
queue = []    # Inisialisasi antrian

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    s = queue.pop(0)
    print (s, end = " ")

    for neighbour in graph[s]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

# Driver Code
bfs(visited, graph, 'A')
```

```
A B C D E F
```

2. Pencarian mendalam pertama (***Depth – First Search***)

   Pencarian dilakukan pada suatu simpul dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada simpul sebelah kanan dan simpul yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan.

In [3]:
```
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = set() # Tetapkan untuk melacak node yang dikunjungi.

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
dfs(visited, graph, 'A')
```

```
A
B
D
E
F
C
```

2. *Heuristic Search* (Pencarian Terbimbing)

Heuristik adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian. Fungsi heuristik digunakan untuk mengevaluasi keadaan-keadaan problema individual dan menentukan seberapa jauh hal tersebut dapat digunakan untuk mendapatkan solusi yang diinginkan. Metode ini menggunakan suatu fungsi yang menghitung estimasi biaya dari suatu simpul tertentu menuju ke simpul tujuan, disebut fungsi heuristik.

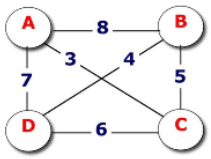
Adanya informasi awal yang digunakan dalam proses pencarian

1. Pendakian Bukit **(*Hill Climbing*)**

   Proses pengujian yang dilakukan dengan menggunakan fungsi heuristik. Pembangkitan keadaan berikutnya sangat tergantung pada feedback dari prosedur pengetesan. Tes yang berupa fungsi heuristik akan menunjukkan seberapa baik nilai terkaan yang diambil terhadap keadaan-keadaan lain yang mungkin. Metode ini mengeksplorasi keadaan secara depth-first search dengan mencari path yang bertujuan menurunkan cost untuk ke goal path.

In [1]:
```
from IPython.display import Image
Image(filename='hill.png')
```

Out[1]:



In [2]:
```
AB = 8
BA = 8
BC = 5
CB = 5
CD = 6
DC = 6
BD = 4
DB = 4
AD = 7
DA = 7
AC = 3
CA = 3
```

In [3]:
```
p1 = AB+BC+CD
p1
```

Out[3]: 19

In [4]:
```
p2 = BA+AC+CD
p3 = AC+CB+BD
p4 = AB+BD+DC
p5 = DB+BC+CA
p6 = AD+DC+CB
p7 = CB+BA+AD

p2, p3, p4, p5, p6, p7
```

Out[4]: (17, 12, 18, 12, 18, 20)

In [5]:
```
q1 = AB+BC+CD
q2 = BC+CA+AD
q3 = BA+AD+DC
q4 = DA+AC+CB
q5 = BD+DC+CA
q6 = CA+AB+BD

q1, q2, q3, q4, q5, q6
```

Out[5]: (19, 15, 21, 15, 13, 15)

In [6]:
```
r1 = CB+AB+BD
r2 = BA+AC+CD
r3 = BC+CD+DA
r4 = DC+CA+AB
r5 = BD+DA+AC
r6 = AC+CB+BD

r1, r2, r3, r4, r5, r6
```

Out[6]: (17, 17, 18, 17, 14, 12)

In [7]:
```
s1 = DB+BA+AC
s2 = BA+AD+DC
s3 = BD+DC+CA
s4 = CD+DA+AB
s5 = BC+CA+AD
s6 = AD+DB+BC

s1, s2, s3, s4, s5, s6
```

Out[7]: (15, 21, 13, 21, 15, 16)

```
In [8]: t1 = DB+BC+CA
        t2 = BC+CD+DA
        t3 = BD+DA+AC
        t4 = AD+DC+CB
        t5 = CB+BA+AD
        t6 = AD+DC+CB

        t1, t2, t3, t4, t5, t6
Out[8]: (12, 18, 14, 18, 20, 18)
```

```
In [13]: u1 = BD+DC+CA
         u2 = DC+CB+BA
         u3 = DB+BA+AC
         u4 = AC+CD+DB
         u5 = DA+AC+CB
         u6 = CB+BD+DA

         u1, u2, u3, u4, u5, u6
Out[13]: (13, 19, 15, 13, 15, 16)
```

2. Pencarian Terbaik Pertama **(*Best First Search*)**

   Best-First Search merupakan sebuah metode yang membangkitkan simpul dari simpul sebelumnya. *Best-First search* memilih simpul baru yang memiliki biaya terkecil diantara semua leaf nodes (simpul-simpul pada level terdalam) yang pernah dibangkitkan. Penentuan simpul terbaik dilakukan dengan menggunakan sebuah fungsi yang disebut fungsi evaluasi f(n). Fungsi evaluasi best-first search dapat berupa biaya perkiraan dari suatu simpul menuju ke *goal* atau gabungan antara biaya sebenarnya dan biaya perkiraan tersebut

   a. *Greedy Best First Search*

   Greedy Best-First adalah algoritma best-first yang paling sederhana dengan hanya memperhitungkan biaya perkiraan (estimated cost) saja, yakni

   f(n) = h(n).

   Biaya yangsebenarnya (actual cost) tidak diperhitungkan. Dengan hanya memperhitungkan biaya perkiraanyang belum tentu kebenarannya, maka algoritma ini menjadi tidak optimal.

   b. A*

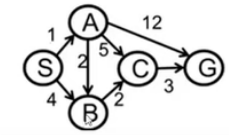
   A* adalah algoritma *best-first search* yang menggabungkan *Uniform Cost Search* dan *Greedy Best-First Search*. Biaya yang diperhitungkan didapat dari biaya sebenarnya ditambah dengan biaya perkiraan. Dalam notasi matematika dituliskan sebagai

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

   Dengan perhitungan biaya seperti ini, algoritma A* adalah *complete* dan *optimal*.

In [1]:
```
from IPython.display import Image
Image(filename='greedy.png')
```

Out[1]:



Node	H
S	7
A	6
B	4
C	2
G	0

In [2]:
```
graph = {

    'S': [('A', 1), ('B', 4)],
    'A': [('B', 2), ('C', 5), ('G', 12)],
    'B': [('C', 2)],
    'C': [('G', 3)]
    }

H_table = {
    'S':7,
    'A':6,
    'B':4,
    'C':2,
    'G':0
}
```

In [8]:
```
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
```

In [9]:
```
path = [('S',0),('A',1),('C',5)]

print(path_h_cost(path))
```

```
(2, 'C')
```

In [10]:
```
path = [('S',0),('A',1),('B',2)]

print(path_h_cost(path))
```

```
(4, 'B')
```

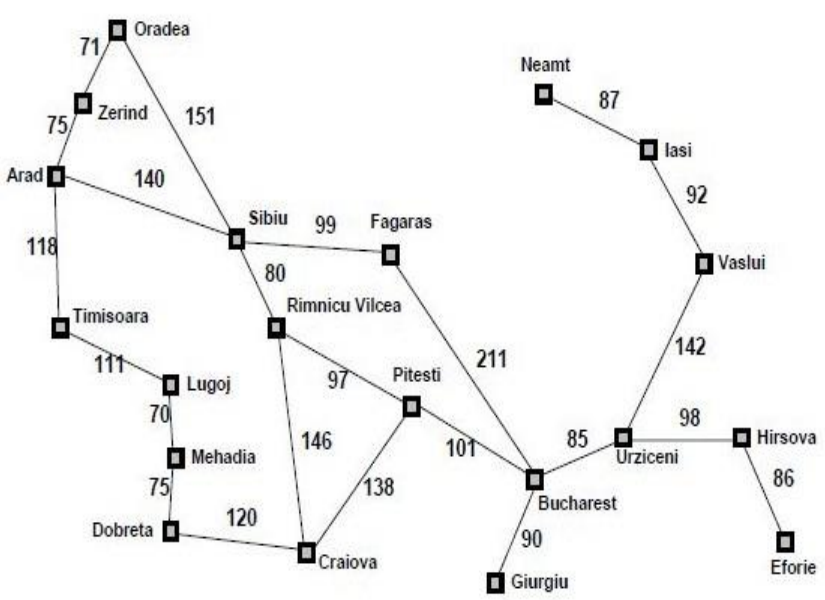
In [15]:
```
def Greedy_best_search(graph, start, goal):
    visited =[]
    queue = [[(start,0)]]
    while queue:
        queue.sort(key=path_h_cost)
        path = queue.pop(0)
        node = path[-1][0]
        if node in visited:
            continue
        visited.append(node)
        if node == 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)
```

In [16]:
```
solution = Greedy_best_search(graph, 'S', 'G')
print ('Solution is', solution)
print ('Cost of Solution is', path_h_cost(solution)[0])
```

```
Solution is [('S', 0), ('B', 4), ('C', 2), ('G', 3)]
Cost of Solution is 0
```

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*!