LAPORAN JOBSHEET 4
INFORMED SEARCH 2

Oleh :

Yulyanggraeni

1731710173

MI 3B

PRODI D III MANAJEMEN INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG
TAHUN 2019

PRATIKUM 1

1. Percobaan 1: Implementasi Algoritma Greedy

1) Untuk memperdalam pengertian algoritma greedy, kita akan mengimplementasikan algoritma yang telah dijelaskan pada bagian sebelumnya ke dalam kode program. Seperti biasa, contoh kode program akan diberikan dalam bahasa pemrograman python. Sebagai langkah awal, tentunya kita terlebih dahulu harus merepresentasikan graph. Pada implementasi yang kita lakukan, graph direpresentasikan dengan menggunakan dictionary di dalam dictionary, seperti berikut:

```
In [1]: DAG = {'A':{'C':4, 'G':9},
               'G':{'E':6},
               'C':{'D':6,'H':12},
               'D':{'E':7},
               'H':{'F':15},
               'E':{'F':8},
               'F':{'B':5}}
```

2) Selanjutnya kita akan membuat fungsi algoritma Greedy sebagai berikut:

```
In [2]: def shortest_path(graph, source, dest):
            result = []
            result.append(source)
            while dest not in result:
                current_node = result[-1]

                local_max = min(graph[current_node].values())
                for node, weight in graph[current_node].items():
                    if weight == local_max:
                        result.append(node)
            return result
```

Pertanyaan

1) Amati output pada percobaan 1, dan Jelaskan bagaimana hasilnya?
   Jawab :

```
In [3]: print ('Shortest path=',shortest_path(DAG, 'A', 'B'))
        Shortest path= ['A', 'C', 'D', 'E', 'F', 'B']
```

2) Jelaskan tahapan-tahapan pada fungsi algoritma Greedy di percobaan 1 langkah ke 2 yang sudah dijelaskan di atas.
   Jawab :
   Potongan code program dibawah ini adalah tahapan untuk mengambil graph, titik awal, dan titik akhir sebagai argumennya

```
In [2]: def shortest_path(graph, source, dest):
```

   Pada algoritma Greedy untuk mendapatkan jarak terpendek yaitu dengan cara mengambil nilai local maximum. Hal ini tentunya kita akan menyimpan jarak terpendek dalam suatu variabel dengan source sebagai isi awal variabelnya. Jarak terpendek tersebut kita simpan dalam sebuah list untuk menyederhanakan proses penambahan nilai

```
In [2]: def shortest_path(graph, source, dest):
            result = []
            result.append(source)
```

Penelusuran graph sendiri akan kita lakukan melalui result, karena variabel ini merepresentasikan seluruh node yang telah kita kunjungi dari keseluruhan graph. Titik awal dari rute tentunya secara otomatis ditandai sebagai node yang telah dikunjungi.dan pada potongan kode tersebut untuk menelusuri graph sampai titik tujuan.

```
    while dest not in result:
        current_node = result[-1]
```

Berikut ada potongan code program untuk mencari nilai local maximum

```
        local_max = min(graph[current_node].values())
```

Selanjutnya ambil node dari local maximum dan ditambahkan ke result

```
        for node, weight in graph[current_node].items():
            if weight == local_max:
                result.append(node)
```

Setelah seluruh graph ditelurusi sampai mendapatkan hasil, kita dapat mengembalikan result ke pemanggil fungsi

```
        for node, weight in graph[current_node].items():
            if weight == local_max:
                result.append(node)
    return result
```

3) Bagaimanakah kompleksitas waktu pada algoritma Greedy? Jelaskan dan beri contoh!
   Jawab :
   Algoritma greedy merupakan algoritma yang besifat heuristik, mencari nilai maksimal sementara dengan harapan akan mendapatkan solusi yang cukup baik. Meskipun tidak selalu mendapatkan solusi terbaik (optimum), algoritma greedy umumnya memiliki kompleksitas waktu yang cukup baik, sehingga algoritma ini sering digunakan untuk kasus yang memerlukan solusi cepat meskipun tidak optimal seperti sistem real-time atau game.

PRATIKUM 2

Percobaan 2: Menyelesaikan Knapsack Problem dengan Algoritma Greedy

Untuk soal knapsack diasumsikan bahwa ada 3 parameter dimana algoritma greedy akan mencari dan menentukan berdasarkan apa dia mengambil nilai terbaik di setiap langkahnya. Parameter tersebut meliputi : bobot, profit, dan profit/bobot. Yang bisa dilihat pada list berikut:

```
item = [[3,4],[4,5],[1,2],[7,5],[6,5],[8,8],[9,11]]
```

pada list item berbentuk 7 x 2. Yang artinya ada 7 baris dan 2 kolom. Baris melambangkan banyaknya nilai. Kolom ke-1 melambangkan bobot, kolom ke-2 melambangkan profit. Kita akan

mencari profit optimum berdasarkan algoritma greedy tadi dengan Kapasitas sistemnya adalah 20.

1) Dengan menggunakan python, kita dapat menyelesaikan masalah knapsack diatas. Buatlah sebuah coding list python, yang di dalam nya terdiri dari penginisialisasian item serta fungsi atau prosedur knapsack. Untuk code program awal tahap ini dengan Python, sebagai berikut:

```
In [1]: from operator import itemgetter, attrgetter
        w = [3,4,1,7,6,8,9]
        p = [4,5,2,5,5,8,11]
        item = [[3,4],[4,5],[1,2],[7,5],[6,5],[8,8],[9,11]]
```

2) Setelah itu buatlah list coding yang berisi fungsi knapsack sebagai berikut:

```
In [2]: i=0
        while i<len(item):
            hasil = item[i][1]/item[i][0]
            item[i].append(hasil)
            i += 1

        data = sorted(item,key=itemgetter(2), reverse = True)

        def knapsack(data,cap,flag):
            total=0
            tres = ""
            if(flag==0):
                dataS = sorted(data,key=itemgetter(flag), reverse = True)
                tres="bobot prioritas : "
            elif(flag==1):
                dataS = sorted(data,key=itemgetter(flag), reverse = True)
                tres="keuntungan prioritas : "
            elif(flag==2):
                dataS = sorted(data,key=itemgetter(flag), reverse = True)
                tres="p prioritas : "
            else:
                return "Error"

            j=0
            hasil=0
            #print("sini")
            cek=0
            weight=0
            while(j<len(dataS)):

                if(cek+dataS[j][0]<=cap):
                    hasil=hasil+dataS[j][1]
                    weight=weight+dataS[j][0]
                    print(dataS[j][0])
                cek=weight
                j+=1;
                #print("here")
            return("Optimal dalam "+str(tres)+str(hasil))

        print(knapsack(item,20,0))
        print(knapsack(item,20,1))
        print(knapsack(item,20,2))
```

Pertanyaan

1) Amati output pada percobaan 2, dan Jelaskan bagaimana hasilnya?
   Jawab :

```
9
8
3
Optimal dalam bobot prioritas : 23
9
8
3
Optimal dalam keuntungan prioritas : 23
1
3
4
9
Optimal dalam p prioritas : 22
```

2) Jelaskan tahapan-tahapan pada fungsi algoritma Greedy di percobaan 2 langkah ke 2 yang sudah dijelaskan di atas.
   Jawab :

   - Langkah pertama yaitu mengidentifikasi himpunan kandidat, yaitu berisi elemenelemen pembentuk solusi.
   - Kemudian mengidentifikasikan himpunan solusi, yaitu berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.
   - Maka akan muncul outputnya

PRATIKUM 3

Percobaan 3: Implementasi Algoritma Dijkstra

Buatlah sebuah class graph, yang di dalam nya terdiri dari beberapa fungsi atau prosedur. Untuk code program awal tahap ini dengan Python, sebagai berikut:

In [4]:
```
from collections import defaultdict
class Graph:
    def __init__(self):
        self.nodes = set()
        self.edges = defaultdict(list)
        self.distances = {}

    def add_node(self, value):
        self.nodes.add(value)

    def add_edge(self, from_node, to_node, distance):
        self.edges[from_node].append(to_node)
        self.edges[to_node].append(from_node)
        self.distances[(from_node, to_node)] = distance

def dijsktra(graph, initial):
    visited = {initial: 0}
    path = {}

    nodes = set(graph.nodes)

    while nodes:
        min_node = None
        for node in nodes:
            if node in visited:
                if min_node is None:
                    min_node = node
                elif visited[node] < visited[min_node]:
                    min_node = node

        if min_node is None:
            break

        nodes.remove(min_node)
        current_weight = visited[min_node]

        for edge in graph.edges[min_node]:
            weight = current_weight + graph.distances[(min_node, edge)]
            if edge not in visited or weight< visited[edge]:
                visited[edge] = weight
                path[edge] = min_node

    return visited, path
```

Pertanyaan

1) Amati output pada percobaan 3, dan Jelaskan bagaimana hasilnya?
   Jawab :

In [5]:
```
g = Graph()

g.add_node('A')
g.add_node('B')
g.add_node('C')
g.add_node('D')
g.add_node('E')
g.add_node('F')

g.add_edge('A','B',7)
g.add_edge('A','C',9)
g.add_edge('A','F',14)
g.add_edge('B','D',15)
g.add_edge('B','A',7)
g.add_edge('B','C',10)
g.add_edge('C','F',2)
g.add_edge('C','A',9)
g.add_edge('C','B',10)
g.add_edge('C','D',11)
g.add_edge('D','E',6)
g.add_edge('D','B',15)
g.add_edge('D','C',11)
g.add_edge('E','D',6)
g.add_edge('E','F',9)
g.add_edge('F','C',2)
g.add_edge('F','E',9)
g.add_edge('F','A',14)
print(dijsktra(g,'A'))
```

```
({'A': 0, 'B': 7, 'C': 9, 'F': 11, 'D': 20, 'E': 20}, {'B': 'A', 'C': 'A', 'F': 'C', 'D': 'C', 'E': 'F'})
```

2) Jelaskan tahapan-tahapan pada fungsi algoritma Dijkstra di percobaan 3 yang sudah dijelaskan di atas.
   Jawab :

   - Tentukan titik mana yang akan menjadi node awal, lalu beri bobot jarak pada node pertama ke node terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap.
   - Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada node awal dan nilai tak hingga terhadap node lain (belum terisi) 2.
   - Set semua node yang belum dilalui dan set node awal sebagai "Node keberangkatan"
   - Dari node keberangkatan, pertimbangkan node tetangga yang belum dilalui dan hitung jaraknya dari titik keberangkatan. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru
   - Saat kita selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai node yang telah dilalui sebagai "Node dilewati". Node yang dilewati tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.
   - Set "Node belum dilewati" dengan jarak terkecil (dari node keberangkatan) sebagai "Node Keberangkatan" selanjutnya dan ulangi langkah nya

3) Bagaimanakah kompleksitas waktu pada algoritma Dijkstra? Jelaskan dan beri contoh!
   Jawab :

   - Pada lagoritma dijkstra, kompleksitas waktu yg dimiliki lebih kecil. Program yang baik akan mempunyai kompleksitas waktu yang kecil sehingga dapat dengan cepat melakukan perhitungan. Algoritma dijkstra dapat dengan mudah dimodifikasi sehingga dapat menampilkan jalur atau path dari suatu simpul ke simpul lainnya dengan jarak terpendek. Hal ini sanagt penting jika graf merepresentasikan sebuah jaringan yang besar dimana kecepatan transfer data antar komputer menjadi suatu nilai penting untuk diperhatikan.