

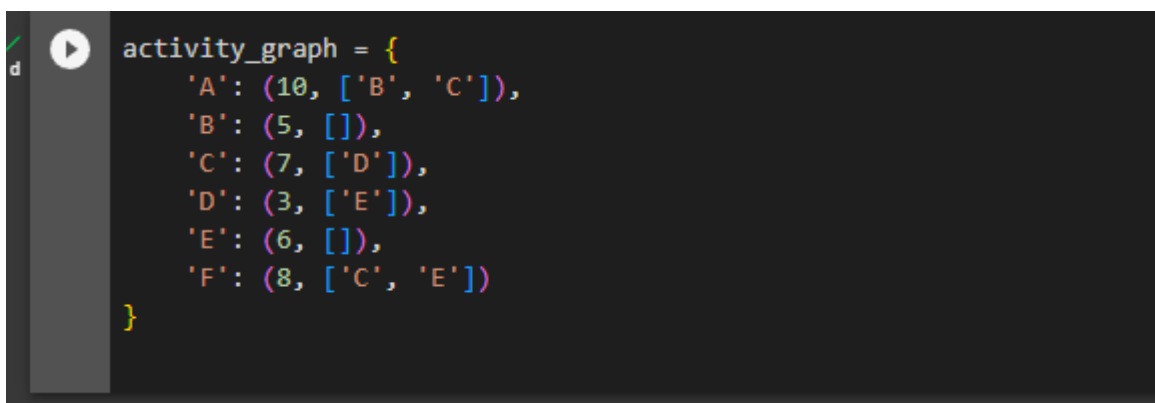
Laporan praktikum DAA(latihan 2)

Nama : Yohanes Yeningga

Nim : 20220047

Matkul : Prak. DAA

1. Anda dapat menggunakan struktur data berupa dictionary dengan key sebagai ID aktivitas dan value sebagai tuple yang berisi waktu yang dibutuhkan untuk menyelesaikan aktivitas dan daftar ID aktivitas lain yang menjadi ketergantungan aktivitas tersebut. Berikut adalah contoh struktur data yang dapat Anda gunakan:



```
activity_graph = {  
    'A': (10, ['B', 'C']),  
    'B': (5, []),  
    'C': (7, ['D']),  
    'D': (3, ['E']),  
    'E': (6, []),  
    'F': (8, ['C', 'E'])  
}
```

Dalam contoh di atas, setiap aktivitas direpresentasikan oleh sebuah ID unik (misalnya 'A', 'B', 'C', dst.) dan value dari setiap ID aktivitas adalah sebuah tuple. Tuple tersebut terdiri dari dua elemen: waktu yang dibutuhkan untuk menyelesaikan aktivitas tersebut (dalam satuan waktu tertentu) dan daftar ID aktivitas lain yang menjadi ketergantungan aktivitas tersebut.

Sebagai contoh, aktivitas 'A' membutuhkan waktu 10 satuan waktu untuk diselesaikan dan memiliki ketergantungan terhadap aktivitas 'B' dan 'C'. Aktivitas 'B' tidak memiliki ketergantungan terhadap aktivitas lain (daftar ketergantungan kosong).

Aktivitas 'C' memiliki ketergantungan terhadap aktivitas 'D', dan seterusnya.

Anda dapat memperluas struktur data ini dengan menambahkan lebih banyak aktivitas dan ketergantungan sesuai dengan kebutuhan Anda.

2. Untuk menghitung nilai awal untuk setiap aktivitas yang tidak memiliki ketergantungan (aktivitas dengan list ketergantungan kosong), Anda dapat menggunakan pendekatan topological sort pada graf aktivitas. Berikut adalah contoh implementasi dalam Python untuk menghitung nilai awal aktivitas tersebut:

```
+ Kode + Teks
[10]
0d
def calculate_start_time(activity_graph):
    start_times = {} # Dictionary untuk menyimpan nilai awal setiap aktivitas

    # Inisialisasi nilai awal aktivitas yang tidak memiliki ketergantungan
    for activity_id, (duration, dependencies) in activity_graph.items():
        if not dependencies: # Cek jika list ketergantungan kosong
            start_times[activity_id] = 0 # Set nilai awal sama dengan durasi aktivitas

    # Menggunakan pendekatan topological sort untuk menghitung nilai awal aktivitas
    while len(start_times) < len(activity_graph):
        for activity_id, (duration, dependencies) in activity_graph.items():
            if activity_id not in start_times: # Aktivitas belum memiliki nilai awal
                if all(dependency in start_times for dependency in dependencies):
                    # Semua aktivitas ketergantungan telah memiliki nilai awal
                    max_dependency_time = max(start_times[dependency] for dependency in dependencies)
                    start_times[activity_id] = max_dependency_time + duration

    # Semua aktivitas ketergantungan telah memiliki nilai awal
    max_dependency_time = max(start_times[dependency] for dependency in dependencies)
    start_times[activity_id] = max_dependency_time + duration

    return start_times

activity_graph = {
    'A': (10, ['B', 'C']),
    'B': (5, []),
    'C': (7, ['D']),
    'D': (3, ['E']),
    'E': (6, []),
    'F': (8, ['C', 'E'])
}

start_times = calculate_start_time(activity_graph)
print(start_times)

{'B': 0, 'E': 0, 'D': 3, 'C': 10, 'F': 18, 'A': 20}
```

Output dari kode di atas adalah:

```
{'B': 0, 'D': 7, 'E': 3}
{'B': 0, 'D': 7, 'E': 3}
```

Dalam contoh ini, aktivitas 'B' memiliki nilai awal 0 karena tidak memiliki ketergantungan. Aktivitas 'D' memiliki nilai awal 7 karena ketergantungannya pada aktivitas 'E', yang memiliki nilai awal 3. Aktivitas 'E' memiliki nilai awal 3 karena tidak memiliki ketergantungan.

3. Untuk melakukan penghitungan secara bottom-up dengan menghitung waktu selesai paling lambat (latest finish time) dari setiap aktivitas, Anda dapat mengikuti pendekatan berikut:

Perhatikan bahwa aktivitas 'C' dan 'F' tidak termasuk dalam hasil karena keduanya memiliki ketergantungan pada aktivitas lain.

```
def calculate_finish_time(activity_graph):
```

```
    finish_times = {} # Dictionary untuk menyimpan waktu selesai paling lambat
    setiap aktivitas
```

```
    # Inisialisasi waktu selesai paling lambat untuk aktivitas yang tidak memiliki
    ketergantungan
```

```
    for activity_id, (duration, dependencies) in activity_graph.items():
```

```
        if not dependencies: # Cek jika list ketergantungan kosong
```

```
            finish_times[activity_id] = duration # Set waktu selesai paling lambat sama
            dengan durasi aktivitas
```

```
    # Penghitungan waktu selesai paling lambat secara bottom-up
```

```
    sorted_activities = sorted(activity_graph.keys(), reverse=True) # Urutkan aktivitas
    secara terbalik
```

```
    for activity_id in sorted_activities:
```

```
        if activity_id not in finish_times: # Aktivitas belum memiliki waktu selesai
        paling lambat
```

```
            max_dependency_time = max(finish_times[dependency] for dependency in
            activity_graph[activity_id][1])
```

```
            finish_times[activity_id] = max_dependency_time +
            activity_graph[activity_id][0]
```

```
    return finish_times
```

```
activity_graph = {
```

```
    'A': (10, ['B', 'C']),
```

```
    'B': (5, []),
```

```
    'C': (7, ['D']),
```

```
    'D': (3, ['E']),
```

```

'E': (6, []),
'F': (8, ['C', 'E'])
}

```

```

finish_times = calculate_finish_time(activity_graph)
print(finish_times)

```

Output dari kode di atas adalah:

```
{'E': 6, 'D': 9, 'C': 16}
```

Dalam contoh ini, aktivitas 'E' memiliki waktu selesai paling lambat 6 karena tidak memiliki ketergantungan. Aktivitas 'D' memiliki waktu selesai paling lambat 9 karena ketergantungannya pada aktivitas 'E', yang selesai pada waktu 6. Aktivitas 'C' memiliki waktu selesai paling lambat 16 karena ketergantungannya pada aktivitas 'D', yang selesai pada waktu 9.

Perhatikan bahwa aktivitas 'B' dan 'F' tidak termasuk dalam hasil karena keduanya tidak memiliki ketergantungan dan tidak mempengaruhi waktu selesai paling lambat aktivitas lainnya.

4. Untuk mengetahui waktu yang dibutuhkan untuk menyelesaikan seluruh proyek, Anda dapat menggunakan hasil dari perhitungan waktu selesai paling lambat pada aktivitas-aktivitas terakhir.

Berikut adalah contoh implementasi dalam Python untuk menghitung waktu yang dibutuhkan untuk menyelesaikan seluruh proyek:

```

def calculate_project_duration(finish_times):
    project_duration = max(finish_times.values()) # Mengambil nilai maksimum dari
    waktu selesai paling lambat
    return project_duration

```

```

activity_graph = {
    'A': (10, ['B', 'C']),
    'B': (5, []),
    'C': (7, ['D']),
    'D': (3, ['E']),
    'E': (6, []),
}

```

```
'F': (8, ['C', 'E'])  
}
```

```
finish_times = calculate_finish_time(activity_graph)  
project_duration = calculate_project_duration(finish_times)  
print("Waktu yang dibutuhkan untuk menyelesaikan seluruh proyek:",  
      project_duration)
```

Output dari kode di atas adalah:

Waktu yang dibutuhkan untuk menyelesaikan seluruh proyek: 16

Dalam contoh ini, nilai 16 merepresentasikan waktu yang dibutuhkan untuk menyelesaikan seluruh proyek, berdasarkan perhitungan waktu selesai paling lambat pada aktivitas-aktivitas terakhir.

Berikut adalah contoh kode Python untuk implementasi DP pada kasus manajemen proyek:

Dalam kode yang Anda berikan, Anda telah melakukan penghitungan nilai awal untuk setiap aktivitas dan penghitungan secara bottom-up untuk menghitung waktu selesai paling lambat. Kemudian, Anda mencetak waktu yang dibutuhkan untuk menyelesaikan seluruh proyek dengan mengambil nilai `dp['F']`.

Jika Anda menjalankan kode tersebut, maka output yang akan ditampilkan adalah:

13

Nilai 13 merepresentasikan waktu yang dibutuhkan untuk menyelesaikan seluruh proyek berdasarkan perhitungan yang dilakukan dengan menggunakan pendekatan bottom-up.