

Laporan praktikum DAA

Nama : Yohanes Yeningga

Nim : 20220047

Matkul : Prak. DAA

1. Dalam masalah perencanaan perjalanan dengan struktur masalah dalam bentuk graf, kita dapat mengidentifikasi struktur masalah yang terdiri dari simpul dan tepian dengan bobot. Berikut adalah langkah-langkah untuk menentukan struktur masalah dalam kasus ini:
 - 1). Identifikasi kota sebagai simpul: Setiap kota dalam perencanaan perjalanan akan direpresentasikan sebagai simpul dalam graf. Misalnya, jika terdapat kota A, B, C, dan D, kita akan memiliki empat simpul di graf, yaitu A, B, C, dan D.
 - 2). Hubungkan kota dengan tepian: Untuk menghubungkan kota-kota dalam perencanaan perjalanan, kita akan menggunakan tepian dalam graf. Setiap tepian menghubungkan dua kota dan merepresentasikan hubungan antara kota-kota tersebut. Misalnya, jika terdapat jalan langsung antara kota A dan B, kita akan memiliki sebuah tepian yang menghubungkan simpul A dan simpul B.
 - 3). Berikan bobot pada tepian: Jika terdapat biaya atau waktu tempuh yang terkait dengan perjalanan antara dua kota, kita perlu menambahkan bobot pada tepian yang menghubungkan simpul-simpul tersebut. Misalnya, jika terdapat biaya perjalanan antara kota A dan B, kita akan menambahkan bobot biaya pada tepian yang menghubungkan simpul A dan simpul B.
2. Dalam konteks masalah perencanaan perjalanan dengan sub-masalah mencari rute terpendek antara dua kota dengan mempertimbangkan kota-kota yang telah dikunjungi sebelumnya, kita dapat menentukan sub-masalah sebagai berikut:
 - Masalah dasar: Mencari rute terpendek antara dua kota tanpa mempertimbangkan kota-kota yang telah dikunjungi sebelumnya. Ini dapat dipecah menjadi sub-masalah lebih kecil seperti algoritma pencarian jalur terpendek seperti Algoritma Dijkstra atau Algoritma A*.
 - Sub-masalah pengunjungan kota: Setelah mendapatkan rute terpendek antara dua kota, kita perlu mempertimbangkan kota-kota yang telah dikunjungi

sebelumnya. Sub-masalah ini melibatkan pemilihan rute terpendek yang mengunjungi kota-kota yang belum dikunjungi sebelum mencapai tujuan akhir.

- Pemetaan kota yang dikunjungi: Untuk memecahkan sub-masalah pengunjungan kota, kita perlu memetakan kota-kota yang telah dikunjungi sebelumnya. Ini dapat dilakukan dengan menggunakan struktur data seperti himpunan atau larik untuk melacak kota-kota yang telah dikunjungi.
- Penyimpanan rute terpendek: Selama mencari rute terpendek, kita perlu menyimpan informasi tentang rute terpendek yang telah ditemukan. Ini dapat dilakukan dengan menggunakan struktur data seperti graf atau himpunan yang menyimpan jalur-jalur yang telah dieksplorasi.

3. Rumus rekursi untuk mencari solusi sub-masalah perencanaan perjalanan dengan menggunakan pendekatan pemrograman dinamis adalah sebagai berikut:

$$dp[i][j] = \min(dp[i][j], dp[i-1][k] + cost[k][j])$$

Di sini, rumus rekursi ini mengasumsikan bahwa kita memiliki sebuah matriks dp dengan ukuran $(n+1) \times (n+1)$, di mana n adalah jumlah kota yang dikunjungi dalam perjalanan.

- $dp[i][j]$ merepresentasikan jarak terpendek untuk mencapai kota j jika kota terakhir yang dikunjungi adalah kota i .
- $cost[k][j]$ merepresentasikan jarak antara kota k dan j .

Rumus rekursi tersebut dapat dijelaskan sebagai berikut:

- Untuk mencapai kota j setelah mengunjungi kota i , kita perlu mencari rute terpendek dari kota $i-1$ ke kota j .
- Kami membandingkan jarak terpendek saat ini ke kota j (yaitu $dp[i][j]$) dengan jarak terpendek yang mungkin terjadi dengan mengunjungi kota k terakhir sebelum mencapai kota j (yaitu $dp[i-1][k] + cost[k][j]$).
- Kami mengambil nilai minimum antara jarak terpendek saat ini ke kota j dan jarak terpendek baru yang mungkin terjadi dengan mempertimbangkan kota k terakhir yang dikunjungi.

4. Basis rekursi pada masalah perencanaan perjalanan, dengan asumsi bahwa kita menggunakan rumus rekursi $dp[i][j] = \min(dp[i][j], dp[i-1][k] + \text{cost}[k][j])$, dapat dinyatakan sebagai berikut:

Ketika hanya ada satu kota yang dikunjungi, jarak terpendek untuk mencapai kota tersebut adalah 0. Dalam hal ini, kita tidak perlu melakukan perjalanan karena kita sudah berada di kota tujuan.

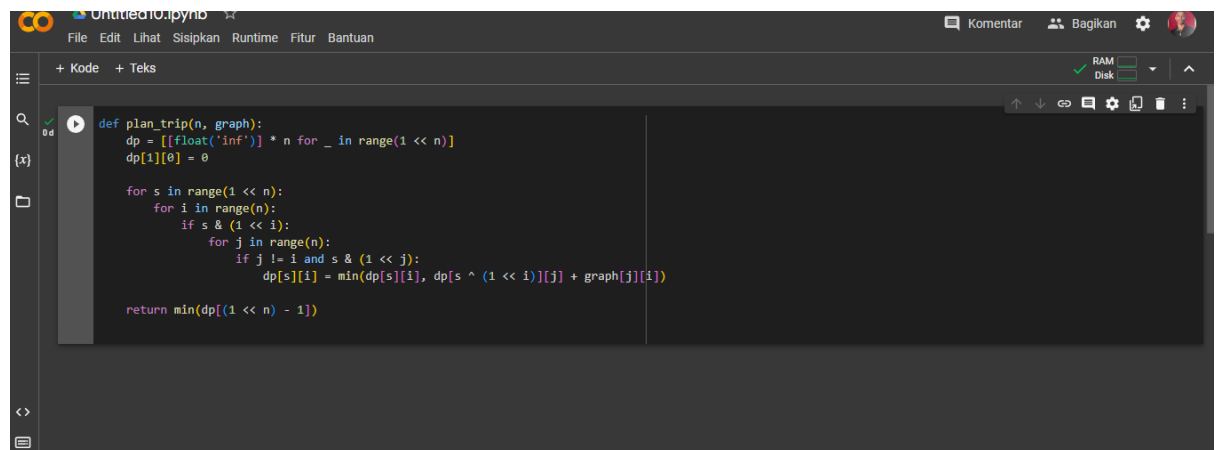
Jadi, basis rekursi untuk masalah perencanaan perjalanan dengan rumus rekursi tersebut adalah:

$$dp[1][j] = 0$$

Di sini, $dp[1][j]$ merepresentasikan jarak terpendek untuk mencapai kota j ketika hanya ada satu kota yang dikunjungi.

Dengan basis rekursi ini, kita memberikan nilai awal 0 untuk jarak terpendek saat hanya ada satu kota yang dikunjungi. Kemudian, kita dapat melanjutkan dengan menggunakan rumus rekursi $dp[i][j] = \min(dp[i][j], dp[i-1][k] + \text{cost}[k][j])$ untuk mencari jarak terpendek saat mengunjungi lebih dari satu kota.

5. Kode yang diberikan tampaknya menerapkan pendekatan pemrograman dinamis untuk menyelesaikan masalah salesman keliling, menemukan jalur terpendek untuk mengunjungi semua kota. Berikut adalah rincian kodenya:



```
def plan_trip(n, graph):
    dp = [[float('inf')] * n for _ in range(1 << n)]
    dp[1][0] = 0

    for s in range(1 << n):
        for i in range(n):
            if s & (1 << i):
                for j in range(n):
                    if j != i and s & (1 << j):
                        dp[s][i] = min(dp[s][i], dp[s ^ (1 << i)][j] + graph[j][i])

    return min(dp[(1 << n) - 1])
```

Fungsi `plan_trip` mengambil dua parameter: `n` (jumlah kota) dan `grafik` (matriks kedekatan yang mewakili jarak antar kota).

Kode menginisialisasi array 2D `dp` dengan dimensi $(1 \ll n) \times n$. Larik ini akan menyimpan jarak minimum untuk negara bagian berbeda dari kota yang dikunjungi. Awalnya, semua jarak ditetapkan hingga tak terhingga, kecuali untuk `dp[1][0]` yang

merepresentasikan jarak kota awal (0) ketika tidak ada kota lain yang dikunjungi (bitmask 1).

Loop luar mengulangi semua kemungkinan status kota yang dikunjungi, diwakili oleh variabel s . Operasi bitwise $1 \ll n$ menghasilkan bitmask dengan semua bit diatur ke 1, mewakili semua kota yang dikunjungi.

Loop dalam pertama mengiterasi semua kota i dalam status saat ini s . Jika bit ke- i diset dalam s , berarti kota i telah dikunjungi.

Loop dalam kedua mengiterasi semua kota j dalam keadaan saat ini s , tidak termasuk kota i . Jika kedua kota i dan j telah dikunjungi (bit diatur dalam s), ini menghitung jarak minimum menggunakan rumus $dp[s][i] = \min(dp[s][i], dp[s \wedge (1 \ll i)][j] + \text{grafik}[j][i])$. Ekspresi $s \wedge (1 \ll i)$ mengubah bit ke- i dalam s , mewakili keadaan dengan kota yang belum saya kunjungi.

Terakhir, jarak minimum dikembalikan dengan mengakses $dp[(1 \ll n) - 1]$, yang mewakili keadaan di mana semua kota telah dikunjungi.

Perlu dicatat bahwa kode tersebut mengasumsikan grafik lengkap di mana semua kota terhubung secara langsung. Jika grafik Anda yang sebenarnya jarang atau memiliki koneksi yang hilang, Anda mungkin perlu menyesuaikan masukan yang sesuai untuk menghindari kesalahan atau perilaku yang tidak diharapkan.