TUGAS 1

ANALISIS & DESAIN ALGORITMA

“Warshall’s and Floyd’s Algorithms”

OLEH :

Chindy Rahmawati
22343089

Dosen Pengampu :
Randi Proska Sandra, M.Sc

PRODI INFORMATIKA
JURUSAN ELEKTRONIKA
FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG
2024

Algoritma Warshall dan algoritma Floyd memiliki tujuan dan metode yang berbeda, tetapi keduanya didasarkan pada ide yang sama: memanfaatkan hubungan antara suatu masalah dan versi yang lebih sederhana daripada yang lebih kecil. Meskipun Warshall dan Floyd tidak secara eksplisit menyebutkan dynamic programming, algoritma-algoritma ini memiliki ciri khas dynamic programming dan telah diakui sebagai aplikasi dari teknik ini.

Warshall's Algorithm

Algoritma Warshall adalah penutupan transitif dari sebuah graf berarah dengan n simpul dapat didefinisikan sebagai matriks boolean n × n T = {tij} di mana elemen dalam baris ke-i dan kolom ke-j adalah 1 jika terdapat jalur nontrivial.

1. **Tujuan Algoritma Warshall**:
    - Algoritma ini bertujuan untuk menemukan *transitive closure* dari sebuah graf berarah.
    - *Transitive closure* menggambarkan hubungan transitif antara semua pasangan vertex dalam graf. Jika ada jalur dari vertex (i) ke (j), maka *transitive closure* akan mencatat bahwa (i) dapat mencapai (j).
2. **Metode Algoritma Warshall**:
    - Graf dinyatakan dalam bentuk matriks boolean, di mana setiap elemen (m[i][j]) menunjukkan apakah ada jalur dari vertex (i) ke (j).
    - Jika ada jalur dari (i) ke (j), maka (m[i][j] = 1); jika tidak, (m[i][j] = 0).
3. **Proses Algoritma Warshall**:
    - Algoritma ini melakukan iterasi melalui semua pasangan vertex.
    - Untuk setiap pasangan (i) dan (j), jika ada jalur tidak langsung dari (i) ke (j) melalui vertex lain (misalnya (k)), maka (m[i][j]) diperbarui menjadi 1.
    - Dengan kata lain, algoritma ini memperbarui matriks berdasarkan keberadaan jalur tidak langsung.
4. **Contoh**:
    - Misalkan kita memiliki graf berarah dengan 4 vertex: A, B, C, dan D.
    - Jika ada jalur dari A ke B, B ke C, dan C ke D, maka *transitive closure* akan mencatat bahwa ada jalur dari A ke D melalui B dan C.

Floyd Algorithm

Algoritma Floyd-Warshall, yang dinamai setelah penciptanya Robert Floyd dan Stephen Warshall, adalah algoritma fundamental dalam ilmu komputer dan teori graf. Algoritma ini

digunakan untuk menemukan *shortest paths* (jalur terpendek) antara semua pasangan node dalam sebuah graf berbobot. Berikut adalah penjelasan lebih lanjut:

1. **Tujuan Algoritma Floyd-Warshall**:
    - Algoritma ini bertujuan untuk menemukan *shortest paths* antara semua pasangan node dalam sebuah graf berbobot.
    - Dapat mengatasi graf dengan bobot tepi positif maupun negatif, sehingga menjadi alat yang serbaguna untuk memecahkan berbagai masalah jaringan dan konektivitas.
2. **Cara Kerja Algoritma Floyd-Warshall**:
    - Algoritma ini dimulai dari satu titik awal dan secara bertahap menggunakan teknik *relaksasi* untuk menemukan jalur terpendek antara titik awal tersebut dan semua titik lain dalam graf.
    - Algoritma ini membandingkan banyak jalur yang mungkin melalui graf antara setiap pasangan vertex.
    - Algoritma ini dijamin menemukan semua jalur terpendek, bahkan jika ada tepi dengan bobot negatif dalam graf.
3. **Pseudo-Kode Algoritma Floyd-Warshall**:
    - Inisialisasi matriks solusi sama dengan matriks graf input.
    - Kemudian perbarui matriks solusi dengan mempertimbangkan semua vertex sebagai vertex perantara.
    - Ketika memilih vertex nomor (k) sebagai vertex perantara, kita sudah mempertimbangkan vertex {0, 1, 2, … k-1} sebagai vertex perantara.
    - Untuk setiap pasangan (i, j) dari vertex sumber dan tujuan, ada dua kemungkinan:
        - (k) bukan vertex perantara dalam jalur terpendek dari (i) ke (j). Kita mempertahankan nilai (dist[i][j]) seperti adanya.
        - (k) adalah vertex perantara dalam jalur terpendek dari (i) ke (j). Kita memperbarui nilai (dist[i][j]) sebagai (dist[i][k] + dist[k][j]), jika (dist[i][j] > dist[i][k] + dist[k][j]).
4. **Aplikasi Dunia Nyata Algoritma Floyd-Warshall**:
    - Digunakan dalam pemodelan jaringan, perencanaan rute, dan optimisasi jaringan.
    - Dapat diterapkan dalam permainan video untuk menghitung jalur terpendek antara karakter atau objek dalam lingkungan permainan.
    - Berguna dalam analisis transportasi dan logistik.

PSEUDOCODE WARSHALL'S AND FLOYD ALGORITHM

Berikut adalah pseudocode untuk **Algoritma Warshall** dan **Algoritma Floyd-Warshall**:

1. **Pseudocode Algoritma Warshall**:

```
Warshall(G):
    Input: G (Graph)
    Output: d (matriks ketetanggaan jarak antara semua pasangan vertex)

    Inisialisasi matriks d dengan nilai tak hingga (inf)
    Untuk setiap vertex v dalam G:
        d[v][v] = 0
    Untuk setiap edge (u, v) dalam G:
        d[u][v] = cost(u, v)  # cost(u, v) adalah bobot edge (u, v)

    Untuk setiap vertex k dalam G:
        Untuk setiap pasangan vertex (i, j) dalam G:
            d[i][j] = min(d[i][j], d[i][k] + d[k][j])

    Kembalikan matriks d
```

2. **Pseudocode Algoritma Floyd-Warshall**:

```
FloydWarshall(G):
    Input: G (Graph)
    Output: d (matriks ketetanggaan jarak antara semua pasangan vertex)

    Inisialisasi matriks d dengan nilai tak hingga (inf)
    Untuk setiap vertex v dalam G:
        d[v][v] = 0
    Untuk setiap edge (u, v) dalam G:
        d[u][v] = cost(u, v)  # cost(u, v) adalah bobot edge (u, v)

    Untuk setiap vertex k dalam G:
        Untuk setiap pasangan vertex (i, j) dalam G:
            d[i][j] = min(d[i][j], d[i][k] + d[k][j])

    Kembalikan matriks d
```

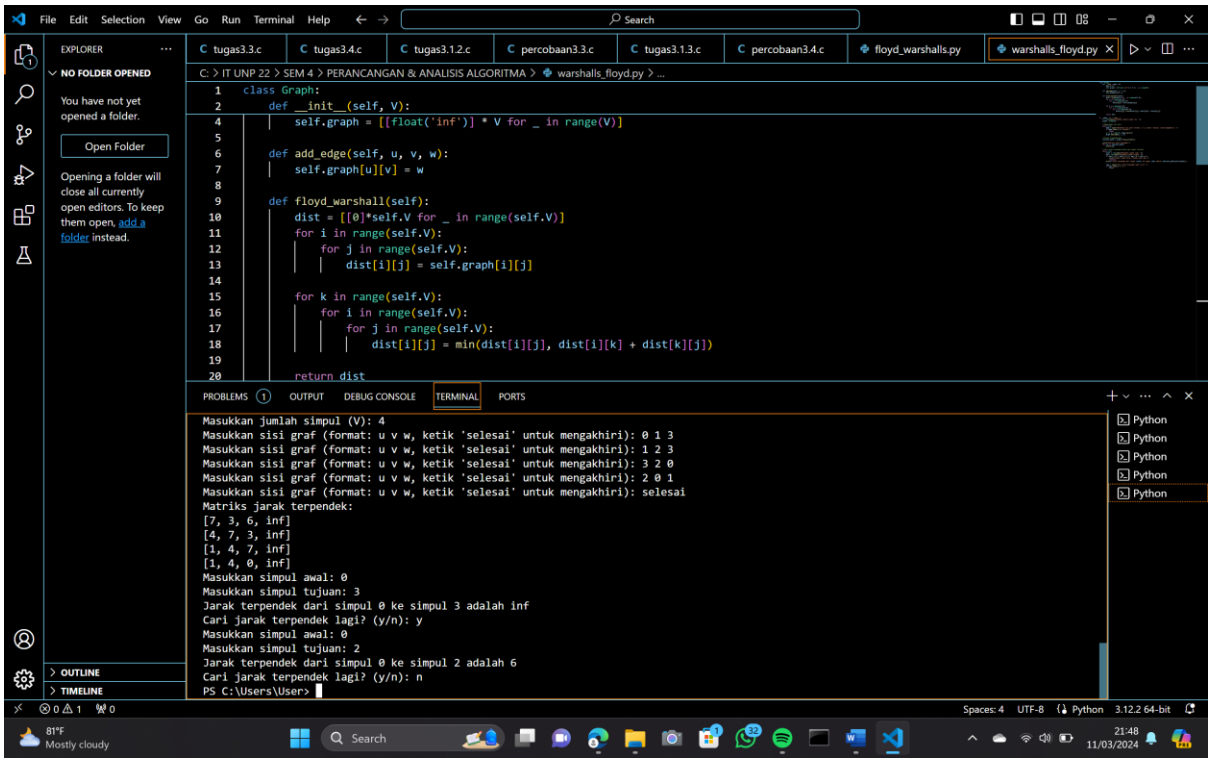
PROGRAM

SOURCE CODE :

```
class Graph:
    def __init__(self, V):
        self.V = V
        self.graph = [[float('inf')] * V for _ in range(V)]

    def add_edge(self, u, v, w):
        self.graph[u][v] = w

    def floyd_warshall(self):
        dist = [[0]*self.V for _ in range(self.V)]
        for i in range(self.V):
            for j in range(self.V):
```

```
                dist[i][j] = self.graph[i][j]

        for k in range(self.V):
            for i in range(self.V):
                for j in range(self.V):
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

        return dist

if __name__ == "__main__":
    V = int(input("Masukkan jumlah simpul (V): "))
    graph = Graph(V)

    # Memasukkan sisi graf
    while True:
        edge = input("Masukkan sisi graf (format: u v w, ketik 'selesai'
untuk mengakhiri): ")
        if edge.lower() == 'selesai':
            break
        u, v, w = map(int, edge.split())
        graph.add_edge(u, v, w)

    # Proses Floyd-Warshall
    shortest_paths = graph.floyd_warshall()

    print("Matriks jarak terpendek:")
    for row in shortest_paths:
        print(row)

    # Cari jarak terpendek antara dua simpul tertentu
    while True:
        start = int(input("Masukkan simpul awal: "))
        end = int(input("Masukkan simpul tujuan: "))
        if start < 0 or start >= V or end < 0 or end >= V:
            print("Simpul tidak valid. Silakan coba lagi.")
            continue
        print(f"Jarak terpendek dari simpul {start} ke simpul {end} adalah
{shortest_paths[start][end]}")

        lagi = input("Cari jarak terpendek lagi? (y/n): ")
        if lagi.lower() != 'y':
```

break



ANALISIS

1. Analisis menyeluruh :

1. **Operator Penugasan (=):** Operator ini digunakan untuk menginisialisasi nilai-nilai awal dalam matriks graf dan matriks jarak terpendek. Misalnya, `self.graph[u][v] = w` digunakan untuk menugaskan bobot w ke edge yang menghubungkan simpul u dan v.

2. **Operator Aritmatika (+, min()):** Operator aritmatika ini digunakan untuk menghitung jarak terpendek antara dua simpul. Pada baris `dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])`, operator `min()` digunakan untuk memilih nilai terkecil antara jarak langsung dari i ke j dan jarak melalui simpul k. Ini digunakan untuk memperbarui nilai jarak terpendek antara i dan j.

3. **Operator Perulangan (for)**: Perulangan digunakan untuk mengiterasi melalui semua simpul dalam algoritma Floyd-Warshall. Terdapat tiga perulangan bersarang yang digunakan untuk memperbarui matriks jarak terpendek. Perulangan luar digunakan untuk menentukan simpul potensial sebagai titik tengah jalur terpendek, sedangkan perulangan dalam digunakan untuk memeriksa semua pasangan simpul yang mungkin.

4. **Input dan Output:** Perintah `input()` digunakan untuk meminta input pengguna, seperti jumlah simpul dan sisi graf. Perintah `print()` digunakan untuk mencetak matriks jarak terpendek dan jarak terpendek antara dua simpul tertentu.

Dengan demikian, program secara efisien menggunakan operator penugasan dan operator aritmatika untuk menginisialisasi, memperbarui, dan mengakses matriks graf dan matriks jarak terpendek, sambil menggunakan perulangan untuk mengiterasi melalui simpul-simpul dan pasangan simpul dalam algoritma Floyd-Warshall.

2. Analisis berdasarkan jumlah operasi abstrak atau operasi khas

1. **Perbandingan (min()):** Operasi perbandingan ini digunakan untuk membandingkan dua nilai untuk menentukan nilai terkecil. Pada setiap iterasi dari algoritma Floyd-Warshall, kita harus membandingkan setiap nilai dalam matriks jarak terpendek dengan jumlah simpul lainnya. Dalam total, ada sekitar $O(n^3)$ perbandingan yang dilakukan, di mana *n* adalah jumlah simpul.
2. **Penugasan Nilai:** Setiap kali kita menemukan nilai yang lebih kecil saat memperbarui matriks jarak terpendek, kita melakukan operasi penugasan untuk mengupdate nilai tersebut. Dalam total, ada sekitar$O(n^3)$penugasan nilai yang dilakukan.
3. **Perulangan (for):** Terdapat tiga perulangan bersarang yang digunakan untuk mengiterasi melalui setiap pasangan simpul dalam algoritma Floyd-Warshall. Setiap perulangan berjalan *V* kali, sehingga totalnya adalah $O(n^3)$ iterasi.

Jadi, secara keseluruhan, kompleksitas waktu algoritma Floyd-Warshall adalah $O(n^3)$, di mana *n* adalah jumlah simpul dalam graf. Jumlah operasi abstrak atau operasi khas yang dominan dalam algoritma ini sekitar $O(n^3)$, terutama didominasi oleh perbandingan dan penugasan nilai dalam matriks jarak terpendek.

3. Analisis menggunakan pendekatan best-case (kasus terbaik), worst-case (kasus terburuk), dan average-case (kasus rata-rata)

1. **Best-case (Kasus Terbaik):**

- **Deskripsi**:Kasus terbaik terjadi ketika graf memiliki sedikit simpul atau ketika graf memiliki struktur yang sangat sederhana sehingga tidak memerlukan banyak perbandingan atau penugasan nilai saat menjalankan algoritma Floyd-Warshall.
- **Kompleksitas Waktu:** Dalam kasus terbaik, kompleksitas waktu algoritma Floyd-Warshall masih \(O(V^3) \), di mana \(V \) adalah jumlah simpul dalam graf. Ini karena algoritma harus melakukan perulangan sebanyak \(V^3 \) kali untuk mengupdate matriks jarak terpendek, bahkan jika tidak banyak perubahan yang terjadi pada matriks selama iterasi.

2. **Worst-case (Kasus Terburuk):**

- **Deskripsi**: Kasus terburuk terjadi ketika graf memiliki banyak simpul atau ketika graf memiliki struktur yang sangat rumit sehingga memerlukan banyak perbandingan dan penugasan nilai saat menjalankan algoritma Floyd-Warshall.
- **Kompleksitas Waktu**: Dalam kasus terburuk, kompleksitas waktu algoritma Floyd-Warshall masih \(O(V^3) \), di mana \(V \) adalah jumlah simpul dalam graf. Dalam kasus terburuk, algoritma harus melakukan perulangan sebanyak \(V^3 \) kali, dan jumlah perbandingan dan penugasan nilai juga mencapai titik maksimum.

3. **Average-case (Kasus Rata-rata):**

- **Deskripsi**: Kasus rata-rata adalah kasus di mana kita mengasumsikan distribusi yang seragam dari input graf, yang artinya kita mempertimbangkan semua kemungkinan struktur dan ukuran graf.
- **Kompleksitas Waktu:** Secara umum, algoritma Floyd-Warshall memiliki kompleksitas waktu $O(n_3)$, dalam kasus rata-rata, karena sebagian besar pengujian memerlukan perulangan sebanyak $O(n_3)$, kali. Namun, dalam beberapa kasus rata-rata, algoritma mungkin sedikit lebih cepat atau lambat tergantung pada struktur dan ukuran graf yang spesifik. Tetapi secara keseluruhan, kompleksitas waktu cenderung berada pada tingkat $O(n_3)$,.

REFERENSI

Introduction to the Design & Analysis of Algorithms 3rd Edition karya Anany Levitin

Floyd-Warshall Algorithm (programiz.com)

What is Floyd-Warshall Algorithm? - A Comprehensive Handbook (upperinc.com)

Floyd Warshall Algorithm - GeeksforGeeks

Floyd-Warshall algorithm implementation in Python · GitHub

LINK GITHUB

chyraa/floydandwarshall: This is a program written in Python language about the Floyd and Warshall algorithms. (github.com)CH