TUGAS MANDIRI
PERANCANGAN & ANALISIS
ALGORITMA “Warshall’s and Floyd’s Algorithms”

202323430048

Disusun Oleh :
Assariy Ramdhani
(22343038)

Dosen Pengampu :
Randi Proska Sandra, S.Pd., M.Sc

INFORMATIKA (NK)
DEPARTEMEN TEKNIK ELEKTRONIKA
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG

A. PENJELASAN PROGRAM/ALGORITMA

Warshall's and Floyd's Algorithms

dua algoritma yang terkenal: algoritma Warshall untuk menghitung penutup transitif dari sebuah graf berarah dan algoritma Floyd untuk masalah jalur terpendek antara semua pasang simpul. Algoritma-algoritma ini didasarkan pada ide yang sama: memanfaatkan hubungan antara suatu masalah dan versi yang lebih sederhana daripada kecil. Warshall dan Floyd menerbitkan algoritma mereka tanpa menyebutkan pemrograman dinamis. Namun demikian, algoritma-algoritma tersebut jelas memiliki rasa pemrograman dinamis dan telah dianggap sebagai aplikasi dari teknik ini.

- **Warshall's Algorithm**

  bahwa algoritma ini digunakan untuk menghasilkan penutup transitif dari sebuah digraf. Penutup transitif ini memberikan informasi tentang jalur-jalur berarah antara semua pasangan simpul dalam graf, memungkinkan untuk menentukan dengan cepat apakah suatu simpul dapat dicapai dari simpul lainnya.

  Algoritma Warshall memanfaatkan matriks ketetanggaan graf untuk secara bertahap menghasilkan penutup transitif melalui serangkaian matriks boolean. Dengan algoritma ini, kita dapat mengetahui dengan cepat dan efisien jalur-jalur berarah yang dapat dilalui antara simpul-simpul dalam graf.

  Aplikasi dari penutup transitif yang dihasilkan oleh algoritma Warshall sangat luas, mulai dari analisis spreadsheet hingga rekayasa perangkat lunak dan rekayasa elektronik. Algoritma ini memberikan alat yang kuat untuk menyelidiki ketergantungan dan aliran data dalam berbagai jenis sistem.

  Penutup 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 (yaitu, jalur berarah dengan panjang positif) dari simpul ke-i ke simpul ke-j; jika tidak, tij adalah 0. Algoritma Warshall membangun penutup transitif melalui serangkaian matriks boolean berukuran n × n:

  $$R^{(1)}, \ldots, R^{(k-1)}, R^{(k)}, \ldots, R^{(n)}$$

  penjelasan → konteks algoritma Warshall merepresentasikan langkah-langkah dalam membangun penutup transitif dari graf berarah. Setiap matriks dalam urutan tersebut menggambarkan keberadaan jalur-jalur dari simpul ke simpul dengan memperhitungkan jumlah simpul perantara yang semakin bertambah, mulai dari yang tidak memiliki simpul perantara hingga yang mempertimbangkan semua simpul sebagai simpul perantara.

- **Floyd's Algorithm for the All-Pairs Shortest-Paths Problem**

Algoritma Floyd-Warshall adalah algoritma untuk mencari jalur terpendek dari setiap pasangan simpul, berbeda dengan Dijkstra dan Bellman-Ford yang merupakan algoritma untuk mencari jalur terpendek dari satu simpul tertentu. Algoritma ini dapat digunakan untuk graf berarah maupun tidak berarah yang memiliki bobot pada setiap tepinya. Namun, algoritma ini tidak dapat digunakan untuk graf yang memiliki siklus negatif (di mana jumlah bobot tepi dalam suatu siklus adalah negatif). Algoritma ini mengikuti pendekatan Pemrograman Dinamis untuk memeriksa setiap kemungkinan jalur yang melalui setiap simpul yang mungkin untuk menghitung jarak terpendek antara setiap pasangan simpul. Kita dapat menghasilkan matriks jarak dengan algoritma yang sangat mirip dengan algoritma Warshall. Algoritma ini disebut algoritma Floyd setelah salah satu penemu bersamanya, Robert W. Floyd. Algoritma ini dapat diterapkan pada graf terarah atau tidak terarah yang memiliki bobot pada setiap tepinya, asalkan tidak mengandung siklus negatif.

Algoritma Floyd menghitung matriks jarak dari sebuah graf terarah atau tidak terarah yang memiliki bobot pada setiap tepinya dengan n simpul melalui serangkaian matriks berukuran n × n:

$$D^{(0)},\ldots,D^{(k-1)},D^{(k)},\ldots.,D^{(n)}$$

Penjelasan→Urutan matriks $D^{(0)},\ldots,D^{(k-1)}$, $D^{(k)},\ldots.,D^{(n)}$ menggambarkan proses iteratif dalam algoritma Floyd-Warshall untuk mencari jarak terpendek antara setiap pasang simpul dalam sebuah graf berbobot. Pada setiap iterasi, matriks $D^{(k)}$ direvisi dengan mempertimbangkan simpul $D^{(k)}$sebagai simpul perantara, sehingga memperbarui estimasi jarak terpendek antara setiap pasangan simpul. Proses ini dilakukan secara berurutan, dimulai dari pertimbangan simpul yang tidak memiliki simpul perantara hingga mempertimbangkan semua simpul sebagai simpul perantara. Selama iterasi, matriks $D^{(k)}$memperbarui estimasi jarak terpendek berdasarkan jarak terpendek yang diperoleh dari matriks $D^{(k-1)}$ dan informasi yang diperoleh dari simpul $D^{(k)}$ sebagai simpul perantara.

B. PSEUDOCODE

Warshall's Algorithm

```
fungsi Warshall(matriks_adj):
    n = jumlah baris atau kolom dalam matriks_adj

    // Inisialisasi matriks transitif dengan matriks ketetanggaan
awal
    transitif = salin(matriks_adj)

    // Iterasi untuk mencari jalur terpendek antara setiap pasangan
simpul
    untuk k dari 0 hingga n-1:
        untuk i dari 0 hingga n-1:
            untuk j dari 0 hingga n-1:
                // Perbarui matriks transitif untuk mencatat jalur
terpendek
                transitif[i][j] = transitif[i][j] OR (transitif[i][k]
AND transitif[k][j])

    kembalikan transitif
```

Pseudocode di atas menggambarkan algoritma Warshall untuk menemukan jalur terpendek antara setiap pasangan simpul dalam graf berarah atau tak berarah. Berikut adalah penjelasan dari setiap bagian pseudocode:

- **Fungsi Warshall(matriks_adj)**: Ini adalah definisi fungsi yang akan menerima matriks ketetanggaan graf sebagai argumen dan mengembalikan matriks transitif yang merepresentasikan jalur terpendek antara semua pasangan simpul.
- **Inisialisasi Variabel**: Variabel **n** diinisialisasi dengan jumlah baris atau kolom dalam matriks ketetanggaan, yang menunjukkan jumlah simpul dalam graf.
- **Inisialisasi Matriks Transitif**: Matriks transitif diinisialisasi dengan menyalin matriks ketetanggaan awal. Ini dilakukan agar kita dapat memodifikasi matriks transitif tanpa mengganggu matriks ketetanggaan asli.
- **Iterasi untuk Mencari Jalur Terpendek**: Terdapat tiga loop bersarang untuk mencari jalur terpendek antara setiap pasangan simpul. Loop pertama adalah untuk simpul perantara **k**, loop kedua dan ketiga digunakan untuk simpul **i** dan **j** yang akan diuji.

Floyd's Algorithm

```
fungsi Floyd(matriks_adj):
    n = jumlah baris atau kolom dalam matriks_adj

    // Inisialisasi matriks jarak dengan matriks ketetanggaan awal
    jarak = salin(matriks_adj)

    // Iterasi untuk mencari jarak terpendek antara setiap pasangan
simpul
    untuk k dari 0 hingga n-1:
        untuk i dari 0 hingga n-1:
            untuk j dari 0 hingga n-1:
                // Perbarui matriks jarak untuk mencatat jarak
terpendek
                jarak[i][j] = minimum(jarak[i][j], jarak[i][k] +
jarak[k][j])

    kembalikan jarak
```

Pseudocode di atas mengimplementasikan algoritma Floyd untuk mencari jarak terpendek antara semua pasangan simpul dalam sebuah graf berbobot. Prosesnya dapat dijelaskan sebagai berikut:

1. **Inisialisasi**: Algoritma menginisialisasi matriks jarak dengan matriks ketetanggaan awal. Ini berarti setiap entri di matriks jarak awalnya adalah bobot dari tepi yang menghubungkan dua simpul, atau infinity jika tidak ada tepi yang menghubungkan kedua simpul tersebut langsung.

2. **Iterasi**: Algoritma melakukan iterasi melalui semua kemungkinan simpul perantara (k) dari 0 hingga n-1. Di dalam iterasi ini, setiap pasangan simpul (i, j) dipertimbangkan, dan jarak terpendek antara i dan j diperbarui menggunakan simpul perantara k. Jika terdapat jalur yang lebih pendek melalui simpul k, maka jarak terpendek antara i dan j diperbarui dengan nilai tersebut.

3. **Kembalikan**: Setelah proses iterasi selesai, algoritma mengembalikan matriks jarak yang berisi jarak terpendek antara setiap pasangan simpul dalam graf tersebut.

Algoritma Floyd menggunakan pendekatan dynamic programming untuk secara bertahap memperbarui jarak terpendek antara semua pasangan simpul. Hal ini memungkinkan pencarian efisien terhadap solusi tanpa memerlukan pendekatan secara terpisah untuk setiap pasangan simpul.

C. SOURCE CODE

Warshall's Algorithm

Source Code :

```
def warshall(matriks_ketetanggaan):
    n = len(matriks_ketetanggaan)

    # Inisialisasi matriks transitif dengan matriks ketetanggaan awal
    matriks_transitif = [list(row) for row in matriks_ketetanggaan]

     # Iterasi untuk mencari jalur terpendek antara setiap pasangan
simpul
    for k in range(n):
        for i in range(n):
            for j in range(n):
                  # Perbarui matriks transitif untuk mencatat jalur
terpendek
                matriks_transitif[i][j] = matriks_transitif[i][j] or
(matriks_transitif[i][k] and matriks_transitif[k][j])

    return matriks_transitif

graf = [
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1],
    [1, 0, 0, 0]
]

hasil = warshall(graf)
for baris in hasil:
    print(baris)
```

Hasil Output :

```
PS C:\Users\hp> & C:/Users/hp/AppData/Local/Programs/Python/Python312/python.exe
Floyd's Algorithms/Warshall's Algorithm.py"
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 1, 1, 1]
```

Kode di atas merupakan implementasi dalam Python dari algoritma Warshall untuk mencari penutup transitif dari matriks ketetanggaan sebuah graf berarah Fungsi **warshall** menerima matriks ketetanggaan sebagai input dan mengembalikan matriks transitif sebagai output. Matriks transitif diinisialisasi dengan membuat salinan matriks ketetanggaan awal. Algoritma melakukan iterasi melalui semua simpul (k) untuk mencari jalur terpendek antara setiap pasangan simpul (i, j). Pada setiap iterasi, perbarui entri matriks transitif untuk mencatat apakah ada jalur terpendek dari simpul i ke simpul j yang melalui simpul k. Hasil akhir berupa matriks transitif yang berisi informasi tentang jalur terpendek antara setiap pasangan simpul dalam graf.

Floyd's Algorithm

Source Code:

```
def floyd(matriks_adj):
    n = len(matriks_adj)

    # Inisialisasi matriks jarak dengan matriks ketetanggaan awal
    jarak = [row[:] for row in matriks_adj]

     # Iterasi untuk mencari jarak terpendek antara setiap pasangan
simpul
    for k in range(n):
        for i in range(n):
            for j in range(n):
               # Perbarui matriks jarak untuk mencatat jarak terpendek
                    jarak[i][j] = min(jarak[i][j], jarak[i][k] +
jarak[k][j])

    return jarak

graf = [
    [0, 3, float('inf'), 7],
    [8, 0, 2, float('inf')],
    [5, float('inf'), 0, 1],
    [2, float('inf'), float('inf'), 0]
]

# Panggil fungsi Floyd
hasil = floyd(graf)

# Cetak hasil
for baris in hasil:
    print(baris)
```

Hasil Output :

```
PS C:\Users\hp> & C:/Users/hp/AppData/Local/Programs/Python/Python312/python.exe "c
Floyd's Algorithms/Floyd's Algorithm.py"
[0, 3, 5, 6]
[5, 0, 2, 3]
[3, 6, 0, 1]
[2, 5, 7, 0]
PS C:\Users\hp>
```

Fungsi **floyd** di atas mengimplementasikan algoritma Floyd untuk mencari jarak terpendek antara setiap pasangan simpul dalam sebuah graf berbobot. Pada awalnya, matriks jarak diinisialisasi dengan matriks ketetanggaan awal. Kemudian, dilakukan iterasi untuk setiap simpul sebagai simpul perantara, di mana setiap iterasi memperbarui matriks jarak dengan membandingkan jarak yang ada dengan kemungkinan jarak baru melalui simpul perantara tersebut. Setelah iterasi selesai, fungsi mengembalikan matriks jarak yang berisi jarak terpendek antara setiap pasangan simpul. Contoh penggunaan fungsi tersebut pada graf tertentu ditunjukkan pada blok kode di bawahnya, di mana hasilnya dicetak untuk diperiksa.

D. ANALISIS KEBUTUHAN WAKTU

Warshall's Algorithm

Analisis Kebutuhan Waktu Algoritma Warshall's Algorithm:

1. Analisis Berdasarkan Operasi Penugasan dan Aritmatika:
    - Algoritma Warshall menggunakan tiga loop bersarang, yang berarti ada tiga tingkat iterasi. Di dalam setiap iterasi, terdapat operasi penugasan sederhana untuk memperbarui matriks transitif. Jumlah operasi penugasan di dalam loop ini adalah sekitar O(n^3) di mana n adalah jumlah simpul dalam graf. Operasi aritmatika terdiri dari operasi logis (OR dan AND) yang digunakan untuk memperbarui nilai-nilai dalam matriks transitif. Oleh karena itu, kompleksitas waktu dari operasi aritmatika dalam loop tersebut juga sekitar O(n^3),
2. Analisis Berdasarkan Jumlah Operasi Abstrak:
    - Algoritma Warshall melakukan n kali iterasi untuk setiap simpul sebagai simpul perantara. Di dalam setiap iterasi, setiap elemen dalam matriks transitif diperbarui menggunakan operasi logis (OR dan AND), yang berarti terdapat sekitar (n^2),operasi logis dalam setiap iterasi. Oleh karena itu, jumlah operasi abstrak untuk algoritma Warshall adalah sekitar O(n^2).

3. Analisis Kasus best-case (Terbaik) , worst-case (Terburuk) , dan average-case (Rata-Rata):
    - Best-case: Kasus terbaik terjadi ketika graf memiliki sedikit simpul atau sedikit tepi. Dalam kasus terbaik, kompleksitas waktu algoritma Warshall tetap O(n^3), karena jumlah iterasi tidak berubah.

    - Worst-case: Kasus terburuk terjadi ketika graf memiliki banyak simpul dan tepi. Dalam kasus terburuk, kompleksitas waktu algoritma Warshall tetap O(n^3), karena tidak ada skenario di mana jumlah iterasi lebih sedikit.

    - Average-case: Dalam kasus rata-rata, kompleksitas waktu algoritma Warshall juga O(n^3), karena algoritma ini bergantung pada jumlah total simpul dan tepi dalam graf, dan tidak ada variasi yang signifikan dalam jumlah iterasi.

Floyd's Algorithm

Analisis Kebutuhan Waktu Algoritma Floyd's Algorithm:

1. Analisis Berdasarkan Operasi Penugasan dan Aritmatika:
    - Algoritma Floyd menggunakan tiga loop bersarang, yang berarti ada tiga tingkat iterasi. Di dalam setiap iterasi, terdapat operasi penugasan sederhana untuk memperbarui matriks jarak. Jumlah operasi penugasan di dalam loop ini adalah sekitar O(n^3), di mana n adalah jumlah simpul dalam graf. Operasi aritmatika terdiri dari operasi penjumlahan dan pembandingan untuk memperbarui nilai-nilai dalam matriks jarak. Oleh karena itu, kompleksitas waktu dari operasi aritmatika dalam loop tersebut juga sekitar O(n^3).

2. Analisis Berdasarkan Jumlah Operasi Abstrak:

- Algoritma Floyd melakukan n kali iterasi untuk setiap simpul sebagai simpul perantara. Di dalam setiap iterasi, setiap elemen dalam matriks jarak diperbarui menggunakan operasi penjumlahan dan pembandingan. Oleh karena itu, jumlah operasi abstrak untuk algoritma Floyd juga adalah sekitar O(n^3).

3. Analisis Kasus Terbaik, Terburuk, dan Rata-Rata:
   - Kasus Terbaik: Kasus terbaik terjadi ketika graf memiliki sedikit simpul atau sedikit tepi. Dalam kasus terbaik, kompleksitas waktu algoritma Floyd tetap O(n^3), karena jumlah iterasi tidak berubah.
   - Kasus Terburuk: Kasus terburuk terjadi ketika graf memiliki banyak simpul dan tepi. Dalam kasus terburuk, kompleksitas waktu algoritma Floyd tetap O(n^3), karena tidak ada skenario di mana jumlah iterasi lebih sedikit.
   - Kasus Rata-Rata: Dalam kasus rata-rata, kompleksitas waktu algoritma Floyd juga O(n^3), karena algoritma ini bergantung pada jumlah total simpul dan tepi dalam graf, dan tidak ada variasi yang signifikan dalam jumlah iterasi.

E. REFERENSI

Warshall's and Floyd's Algorithms

Floyd-Warshall Algorithm (programiz.com)

Floyd Warshall Algorithm - GeeksforGeeks

BUKU (Introduction to the Design)

F. LAMPIRAN LINK GITHUB

https://github.com/AssariyRamdhani/Warshall-s-and-Floyd-s-Algorithms.git

AssariyRamdhani/Warshall-s-and-Floyd-s-Algorithms (github.com)