LAPORAN

TUGAS EKSPORASI MANDIRI

Nama Dosen
Randi Proska Sandra, M.Sc.

-Warshall's and Floyd's Algorithms-

YAZID AQIL ASSALAM
22343014
Informatika
Perancangan dan Analisis Algoritma
202323430047

2024

A. IDENTITAS MAHASISWA

WARSHALL'S AND FLOYD'S ALGORITHMS	
Nama	**YAZID AQIL ASSALAM**
NIM	**22343014**
Prodi	**Informatika**
Nama Dosen	**Randi Proska Sanda, M.Sc.**
Mata Kuliah	**Perancangan dan Analisis Algoritma**
No.Seksi	**202323430047**

B. PENJELASAN PROGRAM / ALGORITMA

Pencarian jalur erat kaitannya dengan jarak, dengan pencarian pada suatu lokasi atau tempat dalam suatu pemetaan dalam pencarian banyak digunakan algoritma heuristik. Salah satu algoritma yang digunakan untuk menentukan jalur terdekat adalah algoritma floyd warshall yang menerapkan fungsi heuristik sehingga akan memberikan hasil pencarian jalur yang paling efektif.

Algoritma Warshall untuk menghitung penutupan transitif dari grafik berarah dan Algoritma Floyd untuk masalah jalur terpendek semua pasangan. Algoritma ini pada dasarnya didasarkan pada ide yang sama yaitu mengeksploitasi hubungan antara suatu masalah dan versinya yang lebih sederhana daripada versi yang lebih kecil.

Algoritma Warshall

- Contoh pengaplikasian Algoritma Warshall yaitu ketika nilai dalam sel spreadsheet diubah, perangkat lunak spreadsheet harus mengetahui semua sel lain yang terpengaruh oleh perubahan tersebut. Jika spreadsheet dimodelkan dengan digraf yang simpulnya mewakili sel spreadsheet dan tepinya menunjukkan ketergantungan sel, penutupan transitif akan memberikan informasi tersebut. Dalam rekayasa perangkat lunak, penutupan transitif dapat digunakan untuk menyelidiki aliran data dan ketergantungan aliran kontrol serta untuk pengujian pewarisan perangkat lunak berorientasi objek. Dalam teknik elektronik, ini digunakan untuk identifikasi redundansi dan pembangkitan pengujian untuk sirkuit digital.

Algoritma Floyd

- Contoh pengaplikasian Algoritma Floyd yaitu jika diberikan graf terhubung berbobot (tidak berarah atau berarah), maka masalah jalur terpendek semua pasangan meminta untuk mencari jarak yaitu, panjang jalur terpendek dari setiap simpul ke semua simpul lainnya. Ini adalah salah satu

dari beberapa variasi masalah yang melibatkan jalur terpendek dalam grafik. Karena penerapannya yang penting pada komunikasi, jaringan transportasi, dan riset operasi, teknologi ini telah dipelajari secara menyeluruh selama bertahun-tahun. Di antara penerapan terbaru dari masalah jalur terpendek semua pasangan adalah menghitung jarak terlebih dahulu untuk perencanaan gerak dalam permainan komputer.

C. PSEUDOCODE

```
// Implementasi Algoritma Floyd
ALGORITMA FLOYD(matriks_tetangga)
    n <- PANJANG(matriks_tetangga)

    // Inisialisasi matriks jarak
    jarak <- matriks_tetangga

    UNTUK k DARI 0 KE n-1 LAKUKAN
        UNTUK i DARI 0 KE n-1 LAKUKAN
            UNTUK j DARI 0 KE n-1 LAKUKAN
                jarak[i][j] <- MINIMUM(jarak[i][j], jarak[i][k] + jarak[k][j])
            SELESAI
        SELESAI
    SELESAI

    KEMBALIKAN jarak
SELESAI ALGORITMA

// Implementasi Algoritma Warshall
ALGORITMA WARSHALL(matriks_tetangga)
    n <- PANJANG(matriks_tetangga)

    // Inisialisasi matriks keberadaan jalur langsung
    jalur_langsung <- matriks_tetangga

    UNTUK k DARI 0 KE n-1 LAKUKAN
        UNTUK i DARI 0 KE n-1 LAKUKAN
            UNTUK j DARI 0 KE n-1 LAKUKAN
                jalur_langsung[i][j]      <-      jalur_langsung[i][j]      ATAU
(jalur_langsung[i][k] DAN jalur_langsung[k][j])
            SELESAI
        SELESAI
    SELESAI

    KEMBALIKAN jalur_langsung
SELESAI ALGORITMA

// Fungsi untuk mencetak matriks
PROSEDUR CETAK_MATRIKS(matriks, nama)
    TULIS(nama)
    UNTUK SETIAP baris DALAM matriks LAKUKAN
        TULIS(baris)
    SELESAI
SELESAI PROSEDUR

// Fungsi untuk mencetak matriks awal
PROSEDUR CETAK_MATRIKS_AWAL(matriks_tetangga)
    TULIS("Matriks Awal:")
    UNTUK SETIAP baris DALAM matriks_tetangga LAKUKAN
        TULIS(baris)
    SELESAI
SELESAI PROSEDUR
```

```
// Contoh penggunaan
matriks_tetangga <- [
    [0, 5, tak hingga, 10],
    [tak hingga, 0, 3, tak hingga],
    [tak hingga, tak hingga, 0, 1],
    [tak hingga, tak hingga, tak hingga, 0]
]

CETAK_MATRIKS_AWAL(matriks_tetangga)

hasil_floyd <- FLOYD(matriks_tetangga)
hasil_warshall <- WARSHALL(matriks_tetangga)

CETAK_MATRIKS(hasil_floyd, "Matriks Jarak (Floyd)")
CETAK_MATRIKS(hasil_warshall, "Matriks Jalur Langsung (Warshall)")
```

D. SOURCE CODE

```python
# Implementasi Algoritma Floyd
def floyd(matriks_tetangga):
    n = len(matriks_tetangga)

    # Inisialisasi matriks jarak
    jarak = [baris[:] for baris in matriks_tetangga]

    for k in range(n):
        for i in range(n):
            for j in range(n):
                jarak[i][j] = min(jarak[i][j], jarak[i][k] + jarak[k][j])

    return jarak

# Implementasi Algoritma Warshall
def warshall(matriks_tetangga):
    n = len(matriks_tetangga)

    # Inisialisasi matriks keberadaan jalur langsung
    jalur_langsung = [baris[:] for baris in matriks_tetangga]

    for k in range(n):
        for i in range(n):
            for j in range(n):
                jalur_langsung[i][j] = jalur_langsung[i][j] or
(jalur_langsung[i][k] and jalur_langsung[k][j])

    return jalur_langsung

# Fungsi untuk mencetak matriks
def cetak_matriks(matriks, nama):
    print(f"{nama}:")
    for baris in matriks:
        print(baris)

# Fungsi untuk mencetak matriks awal
def cetak_matriks_awal(matriks_tetangga):
```

```
    print("\nMatriks Awal:")
    for baris in matriks_tetangga:
        print(baris)

# Contoh penggunaan
matriks_tetangga = [
    [0, 5, float('inf'), 10],
    [float('inf'), 0, 3, float('inf')],
    [float('inf'), float('inf'), 0, 1],
    [float('inf'), float('inf'), float('inf'), 0]
]

cetak_matriks_awal(matriks_tetangga)

hasil_floyd = floyd(matriks_tetangga)
hasil_warshall = warshall(matriks_tetangga)

cetak_matriks(hasil_floyd, "\nMatriks Jarak (Floyd)")
cetak_matriks(hasil_warshall, "\nMatriks Jalur Langsung (Warshall)")
```

E. ANALISIS KEBUTUHAN WAKTU

1. Analisis Operasi/Instruksi
   a. Floyd's Algorithm
      - Dalam algoritma ini, terdapat tiga loop bersarang, yaitu loop untuk k, i, dan j.
      - Di dalam loop terdalam, terdapat operasi penugasan dan operasi perbandingan (min()).
      - Sehingga kompleksitas waktu Floyd's Algorithm adalah O(n^3).

```
# Implementasi Algoritma Floyd
def floyd(matriks_tetangga):
    n = len(matriks_tetangga)

    # Inisialisasi matriks jarak
    jarak = [baris[:] for baris in matriks_tetangga]

    for k in range(n):  # Loop untuk k
        for i in range(n):  # Loop untuk i
            for j in range(n):  # Loop untuk j
                # Operasi penugasan dan perbandingan (min())
                jarak[i][j] = min(jarak[i][j], jarak[i][k] + jarak[k][j])

    return jarak
```

   b. Warshall's Algorithm

- Sama seperti Floyd, Warshall juga memiliki tiga loop bersarang.
- Di dalam loop terdalam, terdapat operasi logika (or) dan operasi perbandingan.

- Sehingga kompleksitas waktu Warshall's Algorithm juga O(n^3).

```
# Implementasi Algoritma Warshall
def warshall(matriks_tetangga):
    n = len(matriks_tetangga)

    # Inisialisasi matriks keberadaan jalur langsung
    jalur_langsung = [baris[:] for baris in matriks_tetangga]

    for k in range(n):  # Loop untuk k
        for i in range(n):  # Loop untuk i
            for j in range(n):  # Loop untuk j
                # Operasi logika (or) dan perbandingan
                jalur_langsung[i][j] = jalur_langsung[i][j] or
(jalur_langsung[i][k] and jalur_langsung[k][j])

    return jalur_langsung
```

2. Analisis Jumlah Operasi Abstrak

a. Floyd's Algorithm

⇨ Jumlah operasi abstrak yang dilakukan dalam algoritma ini adalah sekitar n^3.

b. Warshall's Algorithm

⇨ Jumlah operasi abstrak yang dilakukan dalam algoritma ini juga sekitar n^3.

Dalam kedua algoritma Floyd dan Warshall, kompleksitas waktu diekspresikan sebagai O(n^3) karena keduanya menggunakan tiga loop bersarang yang bergantung pada ukuran masukan n.

- Loop untuk k (Outermost Loop)

⇨ Setiap algoritma memiliki loop luar yang berjalan sebanyak n kali (0 hingga n-1).

- Loop untuk i (Middle Loop)

⇨ Di dalam loop untuk k, terdapat loop untuk i yang juga berjalan sebanyak n kali.

- Loop untuk j (Innermost Loop):

⇨ Di dalam loop untuk i, terdapat loop untuk j yang juga berjalan sebanyak n kali.

Oleh karena itu, jika kita menggabungkan ketiga loop tersebut, kita mendapatkan kompleksitas waktu sekitar n * n * n, yang dapat disederhanakan menjadi O(n^3).

3. Analisis Best-Case, Worst-Case, dan Average-Case
   a. Floyd's Algorithm
      - Best-Case: O(n^3) - terjadi saat matriks tidak berubah (seluruh elemen matriks jarak sudah optimal).
      - Worst-Case: O(n^3) - terjadi saat algoritma selalu memperbarui matriks jarak pada setiap iterasi.
      - Average-Case: O(n^3) - kompleksitas rata-rata tetap O(n^3).
   b. Warshall's Algorithm
      - Best-Case: O(n^3) - terjadi saat matriks tidak berubah (seluruh elemen matriks jalur langsung sudah benar).
      - Worst-Case: O(n^3) - terjadi saat algoritma selalu memperbarui matriks jalur langsung pada setiap iterasi.
      - Average-Case: O(n^3) - kompleksitas rata-rata tetap O(n^3).

   Dengan demikian, keduanya memiliki kompleksitas waktu yang sama, yaitu O(n^3), dan tidak memiliki variasi yang signifikan dalam best-case, worst-case, maupun average-case.

F. REFERENSI

1) Levitin, A. (2012). Introduction to The Design and Analysis of Algorithms. United State of America: Addison-Wesley.
2) Darmadi, Diansyah, T. M., & Handoko, D. (2023, Mei). Penerapan Algoritma Floyd Warshall dengan Menggunakan Euclidean Distance dalam Menentukan Rute Terbaik. Jurnal Ilmu Komputer dan Sistem Informasi, 2, 311 - 321.