

TUGAS ANALISIS DEPTH-FIRST SEARCH

Dosen Pengampu:
Randi Proska Sandra, M.Sc.



Disusun oleh

Nama : Isrezal Akbar
NIM : 23343041
Hari : Selasa

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG
2024

A. Ringkasan

1. Pendahuluan

Graf adalah struktur data yang sangat penting dalam ilmu komputer karena digunakan untuk memodelkan hubungan kompleks antar objek, seperti pada jaringan komputer, sistem transportasi, dan analisis hubungan sosial. Setiap graf terdiri dari simpul (vertex) dan sisi (edge) yang menghubungkannya. Salah satu teknik penelusuran graf yang esensial adalah Depth-First Search (DFS). DFS memungkinkan penelusuran yang mendalam, di mana algoritma akan menyusuri satu cabang graf sampai tidak ada lagi simpul yang dapat dikunjungi, baru kemudian kembali ke simpul sebelumnya untuk mengeksplorasi cabang lain. Teknik ini sangat berguna dalam menemukan komponen terhubung, mendeteksi siklus, dan mencari jalur dalam labirin atau aplikasi lainnya.

2. Algoritma Depth-First Search

DFS merupakan metode penelusuran graf yang bekerja dengan cara menyelidiki cabang sedalam mungkin sebelum kembali ke simpul sebelumnya. Algoritma ini dapat diterapkan dengan dua pendekatan: rekursif dan iteratif.

Pada pendekatan rekursif, DFS memulai dari simpul awal dan menandainya sebagai dikunjungi. Kemudian, algoritma secara rekursif memeriksa setiap tetangga dari simpul tersebut. Jika tetangga belum dikunjungi, DFS dipanggil kembali untuk simpul tersebut. Proses ini terus berlangsung hingga semua simpul yang dapat dijangkau dari simpul awal telah dikunjungi. Pendekatan ini memanfaatkan mekanisme tumpukan panggilan (call stack) bawaan bahasa pemrograman untuk menyimpan status penelusuran.

Pada pendekatan iteratif, struktur data stack eksplisit digunakan untuk menyimpan simpul yang akan dikunjungi. Algoritma dimulai dengan memasukkan simpul awal ke dalam stack. Selama stack tidak kosong, simpul diambil (pop) dari stack; jika simpul tersebut belum dikunjungi, maka ditandai dan dicetak, kemudian semua tetangganya ditambahkan ke stack. Untuk menjaga urutan penelusuran yang konsisten dengan pendekatan rekursif, tetangga sering ditambahkan dalam urutan terbalik.

Kedua pendekatan memastikan bahwa setiap simpul hanya dikunjungi satu kali sehingga total waktu eksekusi DFS adalah $O(V + E)$, dengan V adalah jumlah simpul dan E adalah jumlah sisi. DFS banyak diaplikasikan dalam berbagai masalah, mulai dari pencarian jalur optimal hingga analisis struktur graf.

B. Pseudocode

FUNGSI `telusuri_dfs(peta, simpul_awal, sudah_dikunjungi)`

JIKA `sudah_dikunjungi` adalah NULL

 BUAT `sudah_dikunjungi` sebagai SET kosong

TAMBAHKAN simpul_awal ke dalam sudah_dikunjungi

CETAK simpul_awal

UNTUK setiap tetangga dalam peta[simpul_awal]

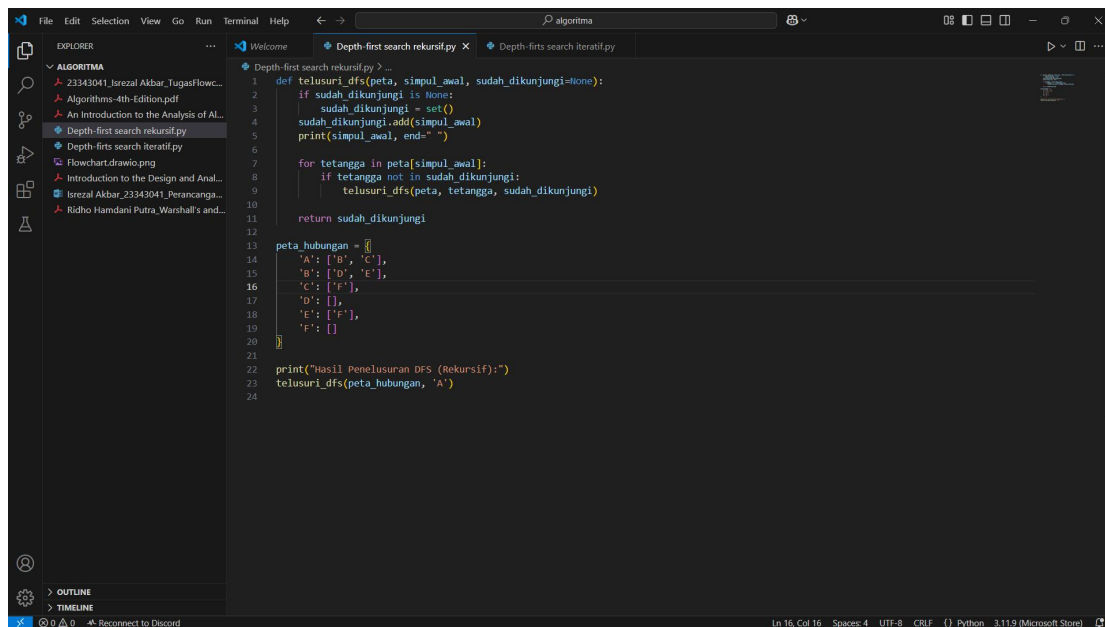
JIKA tetangga BELUM ada dalam sudah_dikunjungi

PANGGIL telusuri_dfs(peta, tetangga, sudah_dikunjungi)

KEMBALIKAN sudah_dikunjungi

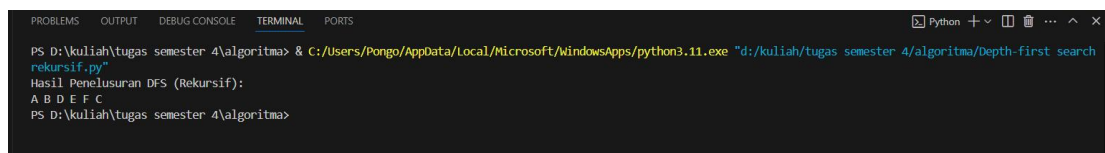
C. Source Code

1. Depth-First Search (Rekursif)



```
1 def telusuri_dfs(peta, simpul_awal, sudah_dikunjungi=None):
2     if sudah_dikunjungi is None:
3         sudah_dikunjungi = set()
4         sudah_dikunjungi.add(simpul_awal)
5         print(simpul_awal, end=" ")
6
7     for tetangga in peta[simpul_awal]:
8         if tetangga not in sudah_dikunjungi:
9             telusuri_dfs(peta, tetangga, sudah_dikunjungi)
10
11     return sudah_dikunjungi
12
13 peta_hubungan = {
14     'A': ['B', 'C'],
15     'B': ['D', 'E'],
16     'C': ['F'],
17     'D': [],
18     'E': ['F'],
19     'F': []
20 }
21
22 print("Hasil Penelusuran DFS (Rekursif):")
23 telusuri_dfs(peta_hubungan, 'A')
24
```

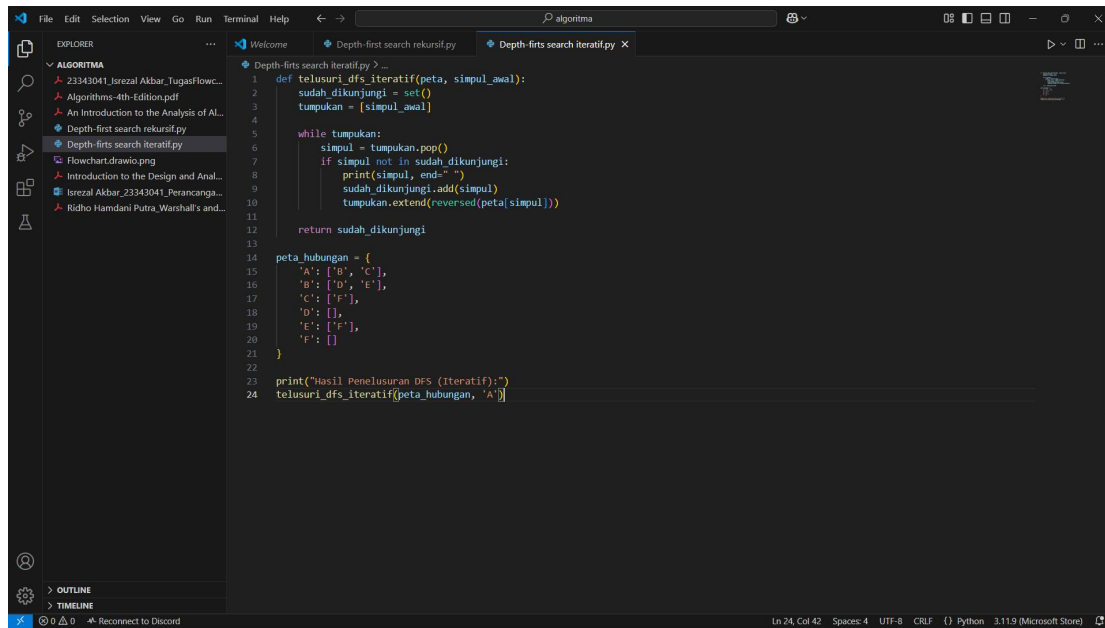
Output:



```
PS D:\kuliahtugas semester 4\algoritma> C:\Users\Pongo\AppData\Local\Microsoft\WindowsApps\python3.11.exe "d:\kuliahtugas semester 4\algoritma\Depth-first search rekursif.py"
Hasil Penelusuran DFS (Rekursif):
A B D E F C
PS D:\kuliahtugas semester 4\algoritma>
```

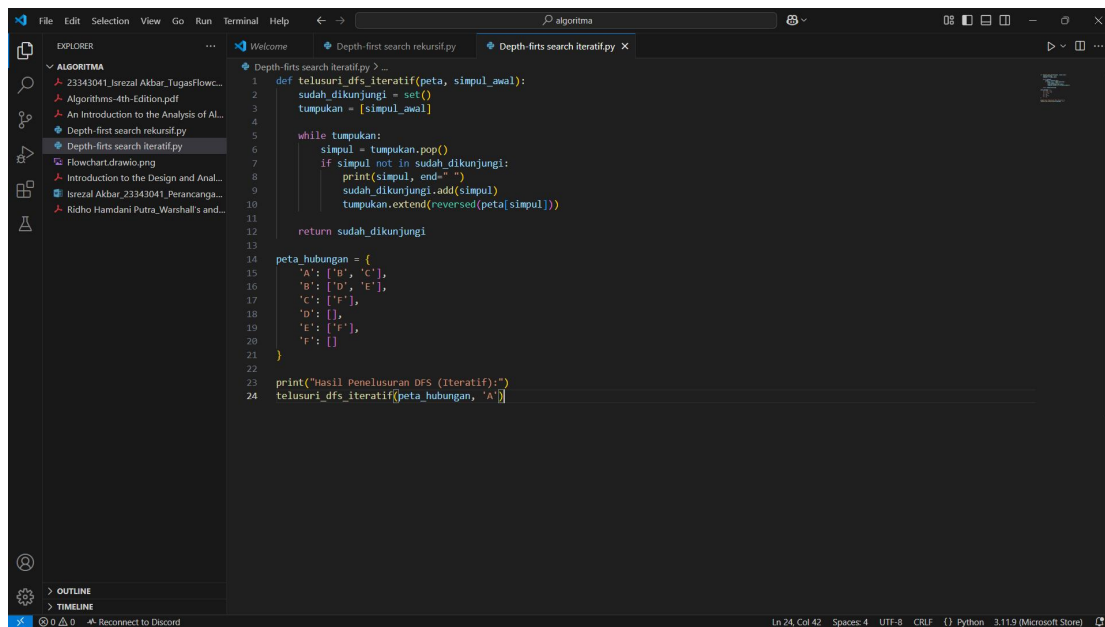
Program diatas menggunakan DFS rekursif, di mana setiap simpul yang dikunjungi dicatat dalam set, lalu algoritma memanggil dirinya sendiri untuk menelusuri tetangga yang belum dikunjungi hingga semua simpul telah dieksplorasi.

2. Depth-First Search (Iteratif)



```
1 def telusuri_dfs_iteratif(peta, simpul_awal):
2     sudah_dikunjungi = set()
3     tumpukan = [simpul_awal]
4
5     while tumpukan:
6         simpul = tumpukan.pop()
7         if simpul not in sudah_dikunjungi:
8             print(simpul, end=" ")
9             sudah_dikunjungi.add(simpul)
10            tumpukan.extend(reversed(peta[simpul]))
11
12    return sudah_dikunjungi
13
14    peta_hubungan = {
15        'A': ['B', 'C'],
16        'B': ['D', 'E'],
17        'C': ['F'],
18        'D': [],
19        'E': ['F'],
20        'F': []
21    }
22
23    print("Hasil Penelusuran DFS (Iteratif):")
24    telusuri_dfs_iteratif(peta_hubungan, 'A')
```

Output:



```
1 def telusuri_dfs_iteratif(peta, simpul_awal):
2     sudah_dikunjungi = set()
3     tumpukan = [simpul_awal]
4
5     while tumpukan:
6         simpul = tumpukan.pop()
7         if simpul not in sudah_dikunjungi:
8             print(simpul, end=" ")
9             sudah_dikunjungi.add(simpul)
10            tumpukan.extend(reversed(peta[simpul]))
11
12    return sudah_dikunjungi
13
14    peta_hubungan = {
15        'A': ['B', 'C'],
16        'B': ['D', 'E'],
17        'C': ['F'],
18        'D': [],
19        'E': ['F'],
20        'F': []
21    }
22
23    print("Hasil Penelusuran DFS (Iteratif):")
24    telusuri_dfs_iteratif(peta_hubungan, 'A')
```

Program diatas menerapkan DFS iteratif dengan tumpukan, menggantikan rekursi untuk menghindari batasan kedalaman panggilan fungsi. Keduanya bekerja dengan prinsip yang sama, yaitu menjelajahi graf secara mendalam sebelum berpindah ke cabang lain.

D. Analisis Kebutuhan Waktu

1. Analisis berdasarkan operator assignment dan aritmatika

Pada implementasi DFS, baik rekursif maupun iteratif, setiap simpul (vertex) dan sisi (edge) diproses melalui serangkaian operasi dasar:

Operasi Assignment:

- Pada DFS rekursif, setiap simpul yang dikunjungi ditandai dengan operasi seperti `visited.add(vertex)`.
- Setiap pemanggilan fungsi (rekursif) dan setiap iterasi pada daftar tetangga mengandung assignment untuk variabel lokal (misalnya, saat iterasi menggunakan `for neighbor in graph[vertex]`).
- Pada DFS iteratif, operasi seperti `stack.pop()` dan `stack.extend(...)` mengandung assignment ke variabel lokal dan struktur data `stack`.

Operasi Aritmatika dan Perbandingan:

- Meskipun DFS tidak banyak melakukan operasi aritmatika murni (seperti penjumlahan atau perkalian), operasi perbandingan muncul saat memeriksa kondisi, misalnya pada `if neighbor not in visited`.
- Operasi ini dilakukan setiap kali sebuah tetangga dicek, sehingga jumlah operasi perbandingan sebanding dengan jumlah sisi (E).

Secara keseluruhan, jika kita misalkan bahwa setiap simpul mengakibatkan c_1 operasi assignment dan setiap sisi mengakibatkan c_2 operasi perbandingan, total instruksi yang dieksekusi adalah sekitar $c_1 \cdot V + c_2 \cdot E$, yang secara asimptotik menjadi $O(V + E)$.

2. Analisis Berdasarkan Jumlah Operasi Abstrak

Dalam pendekatan ini, kita menghitung operasi abstrak utama yang mencakup:

- Pengolahan Simpul: Setiap simpul diproses satu kali (misalnya, penandaan, pencetakan, dan penambahan ke `visited set`).
- Pengolahan Sisi: Setiap sisi diperiksa satu kali ketika algoritma mengiterasi tetangga dari setiap simpul.

Jika kita mendefinisikan biaya konstan untuk setiap operasi pengolahan simpul (c_v) dan tiap operasi pengolahan sisi (c_e), total biaya abstrak adalah:

$$\text{Total Operasi} = c_v \cdot V + c_e \cdot E$$

Di mana V adalah jumlah simpul dan E adalah jumlah sisi. Dengan demikian, jumlah operasi abstrak adalah $O(V + E)$.

3. Analisis Berdasarkan Best-Case, Worst-Case, dan Average-Case

- Best-Case (Kasus Terbaik):

Misalnya, pada graf yang berbentuk pohon (graf sparing) di mana setiap simpul hanya memiliki satu atau dua tetangga, DFS akan mengunjungi setiap simpul tepat satu kali dan jumlah sisi adalah $V-1$. Maka, waktu eksekusi adalah $O(V + (V-1)) = O(V)$.

- Worst-Case (Kasus Terburuk):

Pada graf padat, misalnya graf lengkap di mana hampir setiap simpul terhubung dengan hampir setiap simpul lainnya, jumlah sisi mendekati $O(V^2)$. Meskipun DFS masih hanya mengunjungi setiap simpul satu kali, pemeriksaan kondisi untuk setiap sisi harus dilakukan, sehingga waktu eksekusi bisa mendekati $O(V + V^2) = O(V^2)$.

- Average-Case (Kasus Rata-Rata):

Dalam banyak aplikasi praktis, graf yang dihadapi tidak ekstrem (tidak benar-benar spars atau complete). Biasanya, jumlah sisi adalah proporsional terhadap V (misalnya, $E = O(V)$ atau sedikit lebih besar). Sehingga, pada kasus rata-rata, waktu eksekusi DFS adalah $O(V + E)$ dengan E relatif kecil dibandingkan V^2 .

E. Refrensi

Sedgewick, R. & Wayne, K. (2011). *Algorithms, Fourth Edition*. Pearson Education.

Levitin, A. (2012). *Introduction to the Design and Analysis of Algorithms, 3rd Edition*. Addison-Wesley.

GeeksforGeeks. (2025). Depth First Search (DFS)

<https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

Programiz. (2025). DFS Algorithm in Python <https://www.programiz.com/dsa/depth-first-search>

F. Link Github

<https://github.com/IsrezalAkbar/Perancangan-dan-analisis-algoritma>