**Tugas Eksplorasi Mandiri**

**The Maximum-Flow Problem**

**Perancangan dan Analisis Algoritma (INF1.62.4001)**

**Dosen Pengampu :**

**Randi Proska Sandra, S.Pd, M.Sc**



**Disusun Oleh:**

**Rendi Aigo Brandon**

**23343082**

**INFORMATIKA  
DEPARTEMEN ELEKTRONIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS NEGERI PADANG**

**2025**

**A. RINGKASAN**

Masalah aliran maksimum (Maximum-Flow Problem) adalah masalah optimisasi yang bertujuan untuk menentukan jumlah aliran maksimum yang dapat melewati sebuah jaringan dari sumber (source) ke tujuan (sink) tanpa melebihi kapasitas yang ditentukan pada setiap sisi (edge). Masalah ini banyak digunakan dalam berbagai bidang seperti perencanaan jaringan transportasi, pengelolaan sumber daya, dan komunikasi data. Salah satu algoritma yang umum digunakan untuk menyelesaikan masalah ini adalah algoritma Ford-Fulkerson.

Algoritma ini bekerja dengan mencari jalur augmentasi, yaitu jalur dari sumber ke tujuan yang masih memiliki kapasitas tersisa untuk dilewati aliran tambahan. Setiap kali ditemukan jalur augmentasi, aliran pada jalur tersebut ditingkatkan hingga mencapai kapasitas maksimum jalur tersebut. Proses ini diulangi hingga tidak ada lagi jalur augmentasi yang dapat ditemukan dalam jaringan residual. Jaringan residual adalah representasi dari jaringan asli yang menunjukkan kapasitas tersisa setelah mempertimbangkan aliran yang sudah ada. Dengan kata lain, jaringan residual membantu dalam menentukan apakah masih ada ruang untuk menambah aliran dari sumber ke tujuan.

Dalam implementasinya, algoritma Ford-Fulkerson dapat menggunakan dua metode pencarian, yaitu pencarian lebar-pertama (BFS) atau pencarian dalam-pertama (DFS). BFS umumnya lebih disukai karena lebih mudah memastikan bahwa jalur augmentasi ditemukan secara optimal. Implementasi Ford-Fulkerson yang menggunakan BFS dikenal sebagai algoritma Edmonds-Karp, yang memiliki kompleksitas waktu O(VE²), di mana V adalah jumlah simpul dan E adalah jumlah sisi dalam jaringan. Kompleksitas ini menjadikan algoritma Edmonds-Karp lebih stabil dan dapat diprediksi dibandingkan dengan versi Ford-Fulkerson yang menggunakan DFS.

Penting untuk dicatat bahwa jika kapasitas sisi dinyatakan dalam bilangan real, algoritma Ford-Fulkerson mungkin tidak konvergen. Oleh karena itu, untuk memastikan konvergensi, kapasitas sisi biasanya dinyatakan dalam bilangan bulat. Selain itu, ada beberapa kondisi yang dapat mempengaruhi efisiensi algoritma ini, seperti struktur jaringan dan bagaimana aliran diatur dalam setiap iterasi. Dalam beberapa kasus, algoritma ini bisa berjalan dengan lebih cepat jika jalur augmentasi dipilih dengan strategi tertentu untuk menghindari terlalu banyak iterasi yang tidak efisien.

Algoritma Ford-Fulkerson memiliki aplikasi luas dalam berbagai bidang. Dalam optimisasi jaringan, algoritma ini sering digunakan untuk menemukan kapasitas maksimum dari suatu jaringan transportasi, misalnya dalam pengaturan lalu lintas jalan atau jaringan pipa distribusi air. Dalam bidang komunikasi data, algoritma ini dapat membantu dalam menentukan jalur komunikasi yang paling efisien dalam suatu jaringan komputer. Selain itu, dalam masalah perencanaan sumber daya, algoritma ini dapat digunakan untuk mengalokasikan sumber daya secara optimal di antara berbagai titik permintaan dan suplai.

Secara keseluruhan, algoritma Ford-Fulkerson merupakan salah satu solusi yang paling umum dan efektif untuk menyelesaikan masalah aliran maksimum dalam jaringan. Dengan prinsip dasar yang sederhana tetapi efektif, algoritma ini dapat diimplementasikan dalam berbagai skenario dunia nyata untuk meningkatkan efisiensi dalam distribusi sumber daya, transportasi, dan komunikasi.

**B. PSEUDOCODE**

Fungsi FordFulkerson(graf, sumber, tujuan):

aliran\_maksimum = 0

graf\_residual = salinGraf(graf)

selama adaJalurAugmentasi(graf\_residual, sumber, tujuan, jalur):

aliran\_jalur = kapasitasMinimum(jalur, graf\_residual)

perbaruiKapasitasResidual(jalur, graf\_residual, aliran\_jalur)

aliran\_maksimum = aliran\_maksimum + aliran\_jalur

kembalikan aliran\_maksimum

Fungsi adaJalurAugmentasi(graf, sumber, tujuan, jalur):

inisialisasi struktur data untuk pencarian (misalnya, BFS atau DFS)

tambahkan sumber ke struktur data pencarian

tandai sumber sebagai dikunjungi

selama struktur data pencarian tidak kosong:

simpul = ambil elemen berikutnya dari struktur data pencarian

jika simpul adalah tujuan:

bangun jalur dari sumber ke tujuan

kembalikan benar

untuk setiap tetangga dari simpul:

jika tetangga belum dikunjungi dan kapasitas residual > 0:

tambahkan tetangga ke struktur data pencarian

tandai tetangga sebagai dikunjungi

simpan informasi jalur

kembalikan salah

Fungsi kapasitasMinimum(jalur, graf):

kapasitas\_min = tak hingga

untuk setiap (u, v) dalam jalur:

jika graf[u][v] < kapasitas\_min:

kapasitas\_min = graf[u][v]

kembalikan kapasitas\_min

Fungsi perbaruiKapasitasResidual(jalur, graf, aliran):

untuk setiap (u, v) dalam jalur:

graf[u][v] = graf[u][v] - aliran

graf[v][u] = graf[v][u] + aliran

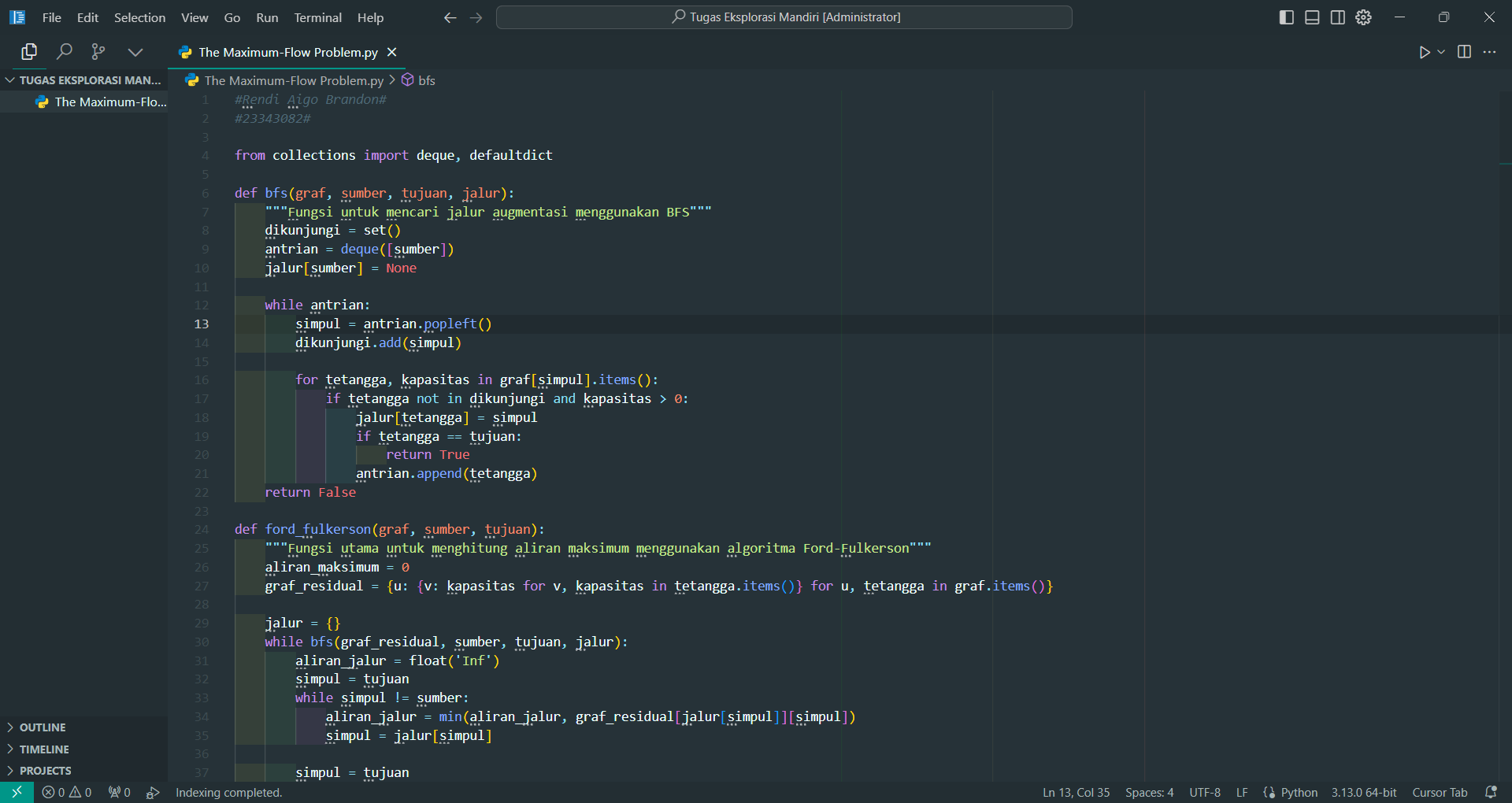
1. **Fungsi FordFulkerson:** Fungsi utama yang menerima parameter graf, sumber, dan tujuan. Fungsi ini menginisialisasi aliran maksimum menjadi 0 dan membuat salinan graf untuk graf residual. Selama masih ada jalur augmentasi yang ditemukan oleh fungsi adaJalurAugmentasi, fungsi akan menghitung kapasitas minimum pada jalur tersebut menggunakan fungsi kapasitasMinimum, memperbarui kapasitas residual dengan fungsi perbaruiKapasitasResidual, dan menambahkan aliran jalur ke aliran maksimum.

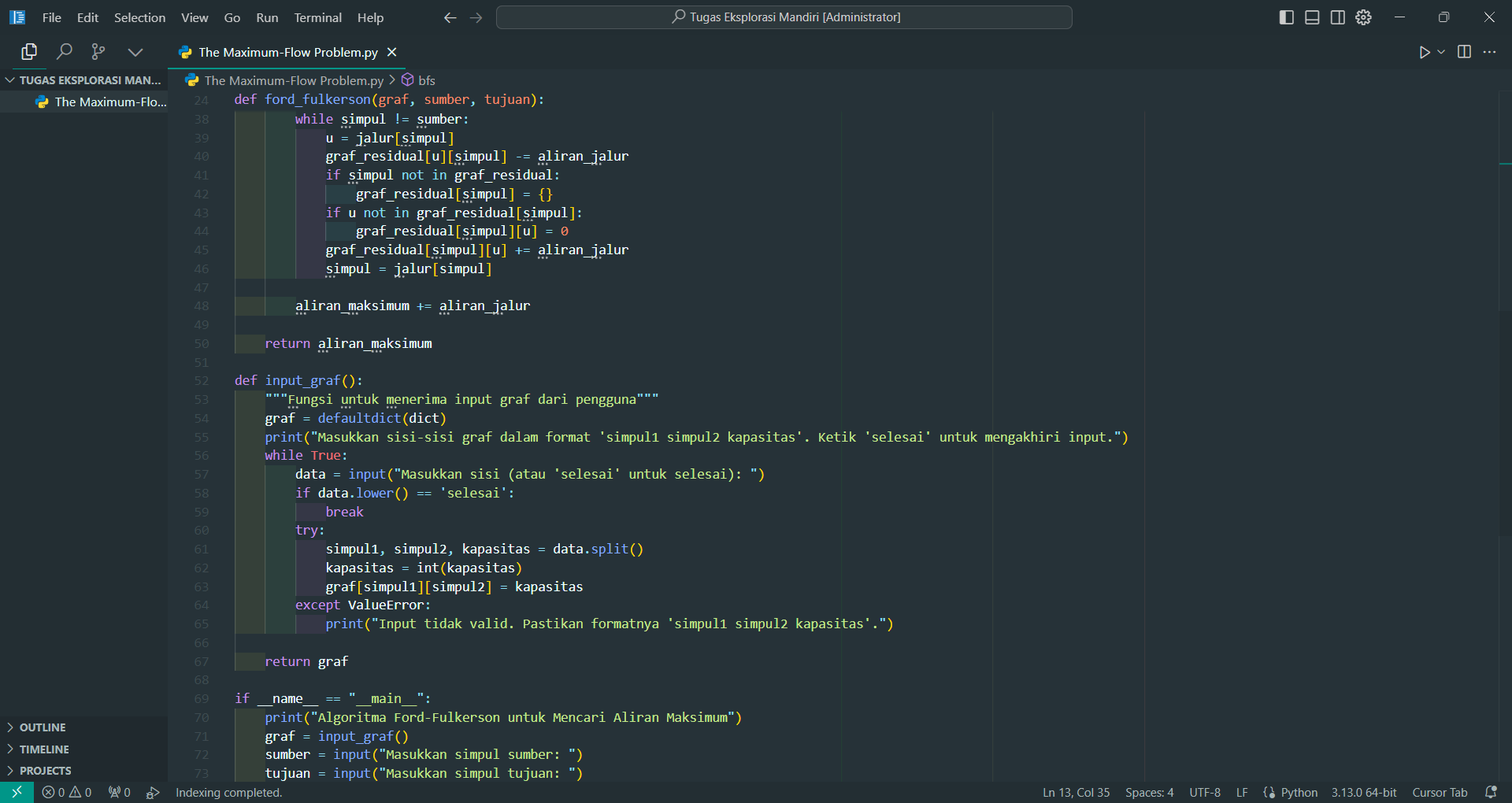
2. **Fungsi adaJalurAugmentasi:** Fungsi ini mencari jalur dari sumber ke tujuan yang masih memiliki kapasitas tersisa menggunakan metode pencarian seperti BFS atau DFS. Jika jalur tersebut ditemukan, fungsi akan membangun jalur dari sumber ke tujuan dan mengembalikan nilai benar. Jika tidak ada jalur yang ditemukan, fungsi mengembalikan nilai salah.

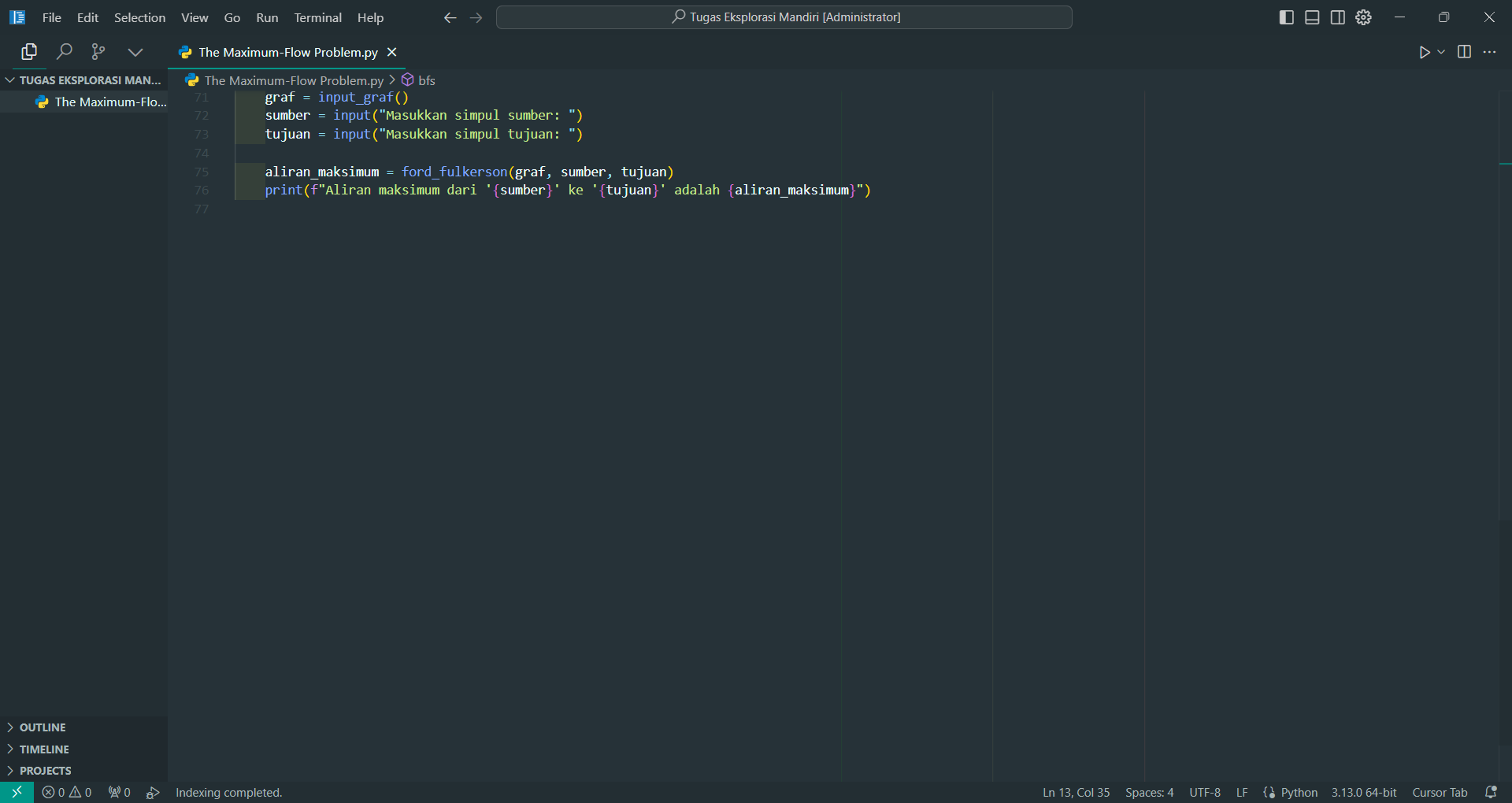
3. **Fungsi kapasitasMinimum:** Fungsi ini menelusuri jalur yang ditemukan dan menentukan kapasitas minimum (bottleneck) di sepanjang jalur tersebut. Kapasitas minimum ini menentukan berapa banyak aliran tambahan yang dapat ditambahkan melalui jalur tersebut.

4. **Fungsi perbaruiKapasitasResidual:** Fungsi ini memperbarui kapasitas residual di sepanjang jalur augmentasi dengan mengurangi kapasitas sisi sesuai dengan aliran dan menambahkan aliran ke sisi balik (reverse edge) di graf residual. Hal ini memungkinkan algoritma untuk mempertimbangkan aliran balik dalam iterasi berikutnya.

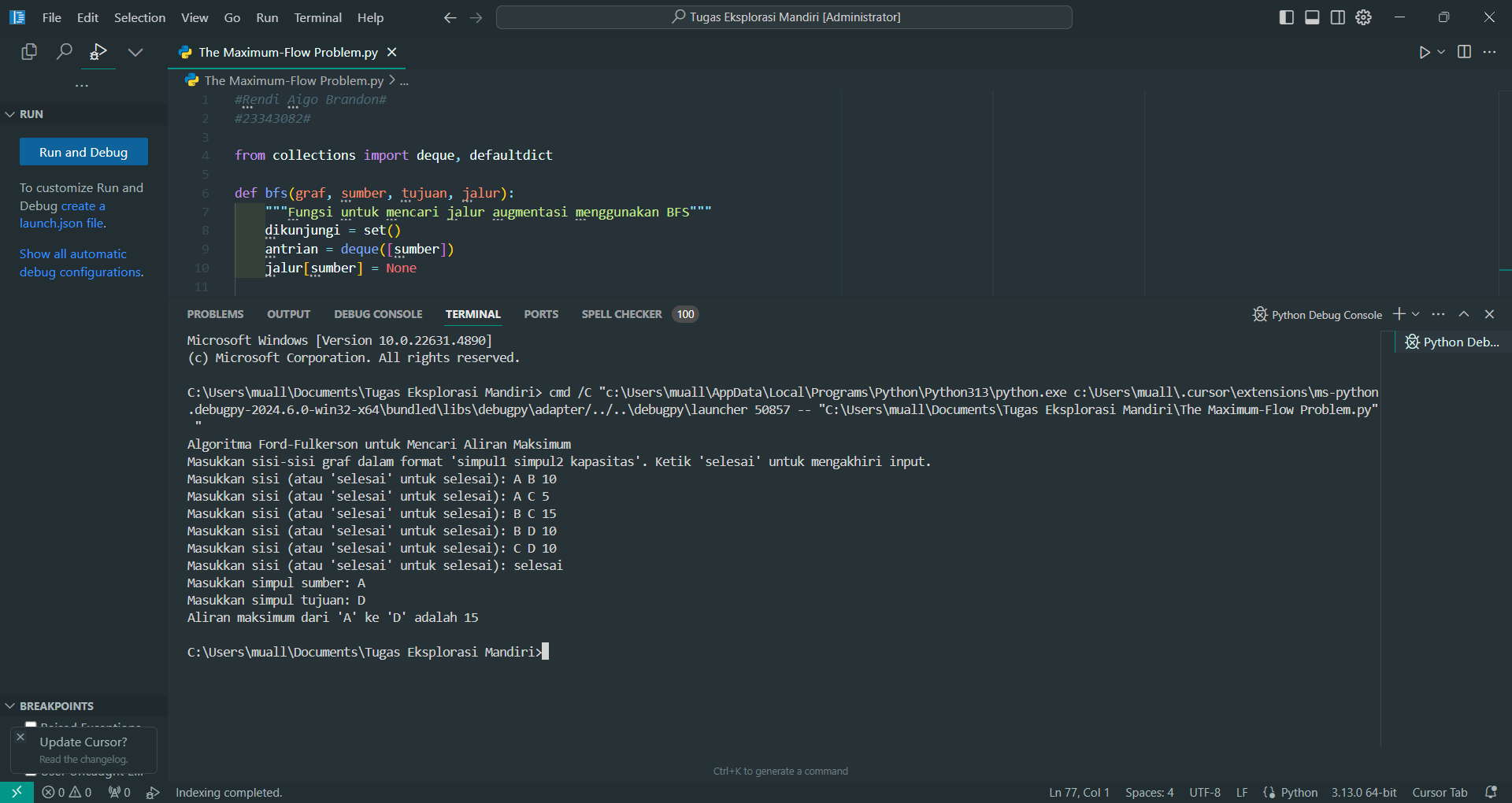
**C. PROGRAM**

****

****

****

**OUTPUT**

****

**E. ANALISIS KEBUTUHAN WAKTU**

**1) Analisis Menyeluruh Berdasarkan Operasi/Instruksi yang Dieksekusi**

Algoritma Ford-Fulkerson dalam program ini terdiri dari dua bagian utama:

* **Mencari jalur augmentasi menggunakan BFS *(bfs()* function)**
* **Memperbarui kapasitas jalur dalam *ford\_fulkerson()* function**

Kita menganalisis kebutuhan waktu berdasarkan operator **penugasan** (=, +=, -=, dll.) dan **aritmatika** (+, -, \*, /, %).

**a. Operasi pada Fungsi BFS**

dikunjungi = set() # O(1)

antrian = deque([sumber]) # O(1)

jalur[sumber] = None # O(1)

* **Penugasan dan inisialisasi set/list:** O(1)

Loop utama BFS:

while antrian: # Looping O(V)

simpul = antrian.popleft() # O(1)

dikunjungi.add(simpul) # O(1)

* while antrian dijalankan sebanyak V kali (V = jumlah simpul).
* popleft() dari deque berjalan dalam O(1).
* add() ke dalam set berjalan dalam O(1).

Iterasi melalui tetangga:

for tetangga, kapasitas in graf[simpul].items(): # O(E) total

if tetangga not in dikunjungi and kapasitas > 0: # O(1)

jalur[tetangga] = simpul # O(1)

antrian.append(tetangga) # O(1)

* Loop berjalan sebanyak E kali dalam seluruh eksekusi BFS (karena memeriksa semua sisi).
* Operasi if dan append berjalan dalam O(1).

**Kompleksitas total BFS:** **O(V + E)**

**b. Operasi pada Fungsi Ford-Fulkerson**

Loop utama:

while bfs(graf\_residual, sumber, tujuan, jalur): # O(f) iterasi

* Ford-Fulkerson dapat mengulang hingga f kali, di mana f adalah aliran maksimum.

Mencari aliran jalur minimum:

aliran\_jalur = float('Inf') # O(1)

simpul = tujuan

while simpul != sumber: # O(V)

aliran\_jalur = min(aliran\_jalur, graf\_residual[jalur[simpul]][simpul]) # O(1)

simpul = jalur[simpul] # O(1)

* Loop berjalan dalam O(V) karena menelusuri jalur augmentasi.

Memperbarui kapasitas:

simpul = tujuan

while simpul != sumber: # O(V)

graf\_residual[u][simpul] -= aliran\_jalur # O(1)

graf\_residual[simpul][u] += aliran\_jalur # O(1)

simpul = jalur[simpul] # O(1)

* Operasi dalam O(V) untuk setiap iterasi.

Kompleksitas total Ford-Fulkerson (tanpa optimasi) adalah: O(E \* f)  
Jika BFS digunakan (Edmonds-Karp), kompleksitas menjadi: O(VE²)

**2) Analisis Berdasarkan Jumlah Operasi Abstrak atau Operasi Khas**

* **Operasi BFS** (pencarian jalur augmentasi) → **O(V + E)**
* **Operasi pembaruan kapasitas jaringan residual** → **O(V)**
* **Jumlah iterasi maksimum** → **O(f)**

Total operasi dalam satu iterasi Ford-Fulkerson:

O(V+E)+O(V)=O(V+E)

Total operasi dalam semua iterasi:

O(f)×O(V+E)=O(E⋅f)

Jika menggunakan BFS (Edmonds-Karp), maka jumlah iterasi maksimal adalah **O(VE)**, sehingga kompleksitas menjadi **O(VE²)**.

**3) Analisis Menggunakan Pendekatan Best-Case, Worst-Case, dan Average-Case**

**a) Best-Case (Kasus Terbaik)**

* Jika setiap jalur augmentasi langsung mencapai **aliran maksimum dalam sedikit iterasi**, maka jumlah iterasi menjadi kecil.
* Kompleksitas terbaik terjadi jika kita hanya butuh satu iterasi BFS.
* **Kompleksitas best-case:** **O(V + E)**

**b) Worst-Case (Kasus Terburuk)**

* Jika peningkatan aliran maksimum sangat kecil setiap iterasi (misalnya, hanya meningkat 1 unit setiap iterasi).
* Jumlah iterasi bisa mencapai **O(f)**.
* Kompleksitas **tanpa BFS**: **O(E \* f)**
* Kompleksitas **dengan BFS** (Edmonds-Karp): **O(VE²)**

**c) Average-Case (Kasus Rata-Rata)**

* Dalam kebanyakan aplikasi nyata, performa Ford-Fulkerson mendekati **O(VE²)** karena jalur augmentasi tidak selalu optimal tetapi juga tidak selalu buruk.

**F, SUMBER REFERENSI**

* **Anany Levitin, *Introduction to the Design & Analysis of Algorithms (3rd Edition)***
* **Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms (3rd Edition)***
* **GeeksForGeeks - Ford-Fulkerson Algorithm for Maximum Flow Problem**[**https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/**](https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/)
* **Brilliant - Ford-Fulkerson Algorithm**[**https://brilliant.org/wiki/ford-fulkerson-algorithm/**](https://brilliant.org/wiki/ford-fulkerson-algorithm/)
* **Stack Overflow - Complexity of Ford-Fulkerson Algorithm**[**https://stackoverflow.com/questions/20036483/complexity-of-the-ford-fulkerson-algorithm**](https://stackoverflow.com/questions/20036483/complexity-of-the-ford-fulkerson-algorithm)
* **OpenAI. (2025). *ChatGPT* (Model GPT-4).**

[**https://chat.openai.com**](https://chat.openai.com)