MODUL PERKULIAHAN

TFC251 – Praktikum Struktur Data

Graph Dalam Python

Penyusun Modul	:	Suamanda Ika Novichasari, M.Kom
Minggu/Pertemuan	:	10
Sub-CPMK/Tujuan Pembelajaran	:	1. Mahasiswa mampu menerapkan konsep graph pada bahasa pemrograman python
Pokok Bahasan	:	1. Representasi dari Struktur Data Graph 2. Transpose Graph 3. BFS dan DFS 4. Siklus Graf 5. Minimum Spanning Tree

Program Studi Teknologi Informasi (S1)
Fakultas Teknik
Universitas Tidar
Tahun 2024

Materi 8
GRAPH DALAM PYTHON

Petunjuk Praktikum :

- Cobalah semua contoh penyelesaian kasus dengan kode program yang terdapat pada semua sub bab dalam modul ini menggunakan google colab.
- Dokumentasikan kegiatan dalam bentuk laporan studi kasus sesuai template laporan praktikum yang telah ditentukan.

Struktur Data Graf adalah kumpulan simpul yang terhubung oleh tepi. Digunakan untuk merepresentasikan hubungan antara entitas yang berbeda. Algoritma graf adalah metode yang digunakan untuk memanipulasi dan menganalisis graf, memecahkan berbagai masalah seperti menemukan lintasan terpendek atau mendeteksi siklus.

7.1 Representasi Struktur Data Graf

Graf adalah struktur data non-linear yang terdiri dari simpul dan tepi. Simpul kadang-kadang juga disebut sebagai simpul dan tepi adalah garis atau lengkungan yang menghubungkan dua simpul dalam graf. Lebih formal sebuah Graf terdiri dari kumpulan simpul (V) dan kumpulan tepi (E). Graf tersebut dilambangkan dengan G(V, E).

Komponen Graf:

- Simpul: Simpul adalah unit dasar dari graf. Kadang-kadang, simpul juga dikenal sebagai simpul atau simpul. Setiap simpul/simpul dapat diberi label atau tidak diberi label.
- Tepi: Tepi digambar atau digunakan untuk menghubungkan dua simpul graf. Ini bisa berupa pasangan terurut dari simpul dalam graf terarah. Tepi dapat menghubungkan dua simpul dalam cara apa pun. Tidak ada aturan. Kadang-

kadang, tepi juga dikenal sebagai lengkungan. Setiap tepi dapat diberi label/tanpa label.

Berikut adalah operasi dasar pada graf:

- Penyisipan Simpul/Tepi dalam graf - Memasukkan simpul ke dalam graf.
- Penghapusan Simpul/Tepi dalam graf - Menghapus simpul dari graf.
- Pencarian pada Graf - Mencari entitas dalam graf.
- Traversal Graf - Melintasi semua simpul dalam graf.

Kelebihan Graf	Kekurangan Graf
• Graf dapat digunakan untuk memodelkan dan menganalisis sistem dan hubungan yang kompleks. • Graf bermanfaat untuk memvisualisasikan dan memahami data. • Algoritma graf secara luas digunakan dalam ilmu komputer dan bidang lainnya, seperti analisis jaringan sosial, logistik, dan transportasi. • Graf dapat digunakan untuk merepresentasikan berbagai jenis data, termasuk jaringan sosial, jaringan jalan, dan internet	• Graf yang besar dapat sulit untuk divisualisasikan dan dianalisis. • Algoritma graf dapat memakan banyak sumber daya komputasi, terutama untuk graf yang besar. • Interpretasi hasil graf dapat subjektif dan mungkin memerlukan pengetahuan spesifik domain. • Graf dapat rentan terhadap noise dan outliers, yang dapat memengaruhi akurasi hasil analisis.

Perbedaan Kunci Antara Graf dan Pohon

- Siklus: Graf dapat mengandung siklus, sedangkan pohon tidak bisa.
- Konektivitas: Graf bisa tidak terhubung (misalnya, memiliki beberapa komponen), sedangkan pohon selalu terhubung.
- Hirarki: Pohon memiliki struktur hirarkis, dengan satu titik ditetapkan sebagai akar. Graf tidak memiliki struktur hirarkis ini.
- Aplikasi: Graf digunakan dalam berbagai aplikasi, seperti jaringan sosial, jaringan transportasi, dan ilmu komputer. Pohon sering digunakan dalam struktur data hirarkis, seperti sistem file dan dokumen XML.

Implementasi graf dalam python dapat dilakukan dengan 2 cara yang umum yaitu :

7.1.1 Adjacency Matrix

Adjacency Matrix:

```python
class Graph:
    def __init__(self, num_vertices):
        self.num_vertices = num_vertices
        self.matrix = [[0] * num_vertices for _ in range(num_vertices)]

    def add_edge(self, u, v):
        self.matrix[u][v] = 1
        self.matrix[v][u] = 1

    def display(self):
        for row in self.matrix:
            print(row)


# Contoh penggunaan
g = Graph(5)
g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 3)
g.add_edge(2, 4)

print("Adjacency Matrix:")
g.display()
```

```
Adjacency Matrix:
[0, 1, 1, 0, 0]
[1, 0, 0, 1, 0]
[1, 0, 0, 0, 1]
[0, 1, 0, 0, 0]
[0, 0, 1, 0, 0]
```

- Kelas Graph didefinisikan untuk merepresentasikan graf. Constructor __init__ menerima parameter num_vertices yang menentukan jumlah total titik dalam graf. Kelas ini memiliki dua atribut:
    - num_vertices: menyimpan jumlah total titik dalam graf.
    - matrix: menyimpan matriks ketetanggaan untuk graf. Matriks ini berukuran num_vertices x num_vertices dan diinisialisasi dengan nilai 0.
- Metode add_edge(self, u, v) digunakan untuk menambahkan tepi antara dua titik u dan v. Dalam matriks ketetanggaan, jika ada tepi antara titik u dan v, maka nilai di posisi matrix[u][v] dan matrix[v][u] diatur menjadi 1.
- Metode display(self) digunakan untuk mencetak matriks ketetanggaan ke layar. Ini melintasi setiap baris dalam matriks dan mencetak nilai-nilai di setiap baris.
- Di bagian bawah program, contoh penggunaan kelas Graph diberikan:
    - Objek g dibuat dari kelas Graph dengan 5 titik (Graph(5)).
    - Beberapa tepi ditambahkan ke graf menggunakan metode add_edge.
    - Pencetakan ke layar menggunakan metode display.

7.1.2 Adjacency List

```python
class Graph:
    def __init__(self):
        self.adj_list = {}

    def add_edge(self, u, v):
        if u not in self.adj_list:
            self.adj_list[u] = []
        if v not in self.adj_list:
            self.adj_list[v] = []
        self.adj_list[u].append(v)
        self.adj_list[v].append(u)

    def display(self):
        for vertex in self.adj_list:
            print(vertex, "->", " -> ".join(map(str, self.adj_list[vertex])))


# Contoh penggunaan
g = Graph()
g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 3)
g.add_edge(2, 4)

print("Adjacency List:")
g.display()
```

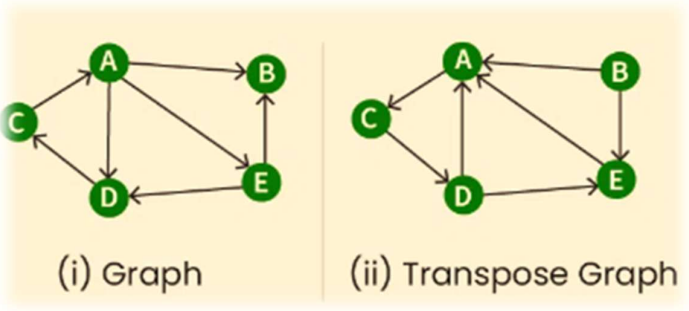
```
Adjacency List:
0 -> 1 -> 2
1 -> 0 -> 3
2 -> 0 -> 4
3 -> 1
4 -> 2
```

- Kelas Graph didefinisikan untuk merepresentasikan graf. Constructor __init__ tidak menerima parameter apa pun. Kelas ini memiliki satu atribut:
    - adj_list: sebuah kamus yang digunakan untuk menyimpan daftar ketetanggaan untuk setiap titik dalam graf. Kunci kamus adalah nomor titik, dan nilai-nilai kamus adalah daftar titik-titik terhubung.
- Metode add_edge(self, u, v) digunakan untuk menambahkan tepi antara dua titik u dan v ke daftar ketetanggaan. Jika titik u atau v belum ada dalam daftar, titik tersebut ditambahkan ke daftar. Kemudian, kedua titik tersebut ditambahkan satu sama lain ke dalam daftar ketetanggaan mereka.
- Metode display(self) digunakan untuk mencetak daftar ketetanggaan ke layar. Metode ini melintasi setiap titik dalam adj_list dan mencetak nomor titik beserta daftar titik-titik terhubungnya.
- Di bagian bawah program, contoh penggunaan kelas Graph diberikan:
    - Objek g dibuat dari kelas Graph.
    - Beberapa tepi ditambahkan ke graf menggunakan metode add_edge.
    - Akhirnya, daftar ketetanggaan untuk graf dicetak ke layar menggunakan metode display.

7.2 Transpose Graf

Transpose dari sebuah graf berarah G adalah graf berarah lain pada himpunan titik yang sama dengan semua tepi dibalik dibandingkan dengan orientasi tepi yang sesuai di G. Artinya, jika G mengandung tepi (u, v) maka konvers/transposisi/balik dari G mengandung tepi (v, u) dan sebaliknya. Diberikan sebuah graf (diwakili sebagai daftar ketetanggaan), kita perlu menemukan graf lain yang merupakan transpose dari graf yang diberikan.

Contoh:

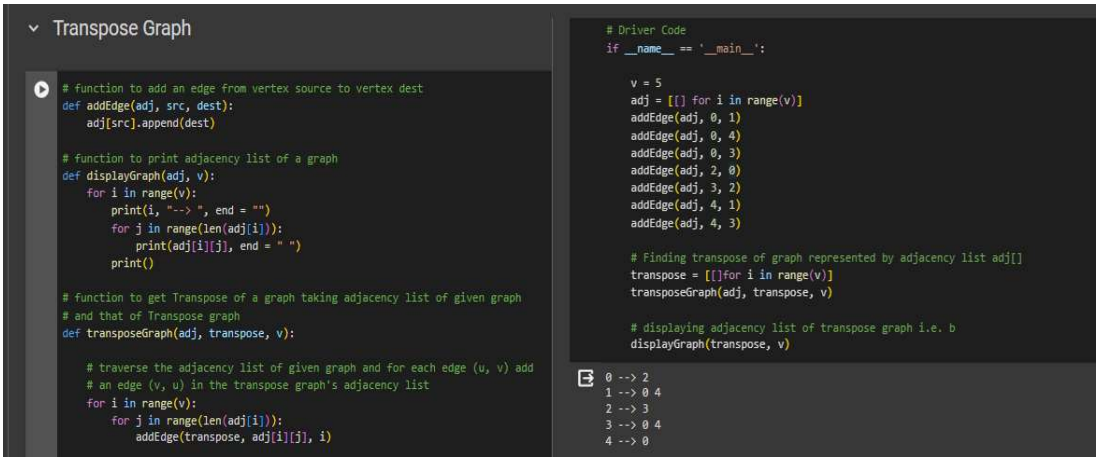


Keterangan :

- Input: gambar (i) adalah graf masukan.
- Output: gambar (ii) adalah graf transpose dari graf yang diberikan.

Kita menelusuri daftar ketetanggaan dan saat kita menemukan sebuah titik v dalam daftar ketetanggaan dari titik u yang menunjukkan sebuah tepi dari u ke v di graf utama, kita hanya menambahkan sebuah tepi dari v ke u di graf transpose yaitu menambahkan u ke daftar ketetanggaan dari titik v dari graf baru. Dengan begitu menelusuri daftar semua titik dari graf utama kita bisa mendapatkan graf transpose. Dengan demikian kompleksitas waktu total algoritma adalah O(V+E) dimana V adalah jumlah titik dari graf dan E adalah jumlah tepi dari graf.

Catatan: Lebih mudah untuk mendapatkan transpose dari sebuah graf yang disimpan dalam format matriks ketetanggaan.



Kompleksitas Waktu:

- Fungsi addEdge memiliki kompleksitas waktu O(1), karena hanya menambahkan elemen ke dalam vektor.
- Kompleksitas waktu fungsi displayGraph adalah O(V + E), di mana V adalah jumlah titik dan E adalah jumlah tepi, karena perlu melintasi daftar ketetanggaan setiap titik dan mencetak titik-titik terhubung.
- Kompleksitas waktu fungsi transposeGraph juga O(V + E), di mana V adalah jumlah titik dan E adalah jumlah tepi, karena perlu melintasi daftar ketetanggaan setiap titik dan menambahkan tepi-titik yang sesuai ke daftar ketetanggaan graf transpose.
- Oleh karena itu, kompleksitas waktu keseluruhan dari program adalah O(V + E).

Kompleksitas Ruang:

- Dalam hal kompleksitas ruang, program menggunakan dua larik vektor untuk merepresentasikan graf asli dan graf transposenya, masing-masing dengan ukuran V (jumlah titik). Selain itu, program menggunakan jumlah ruang konstan untuk menyimpan variabel integer dan struktur data sementara. Oleh karena itu, kompleksitas ruang dari program adalah O(V).
- Perlu dicatat bahwa kompleksitas ruang dari program bisa lebih besar jika graf masukan memiliki jumlah tepi yang besar, karena hal ini akan membutuhkan lebih banyak memori untuk menyimpan daftar ketetanggaan.

7.3 *Breadth First Search* (BFS) dan *Depth First Search* (DFT)

Breadth First Search (BFS) adalah teknik berbasis titik untuk mencari jalur terpendek dalam graf. Ini menggunakan struktur data Antrian yang mengikuti konsep

first in first out. Dalam BFS, satu titik dipilih pada satu waktu ketika dikunjungi dan ditandai, kemudian tetangganya dikunjungi dan disimpan dalam antrian. Metode ini lebih lambat daripada DFS.

Depth First Search (DFT) adalah teknik berbasis tepi. Ini menggunakan struktur data Tumpukan dan melakukan dua tahap, pertama-tama titik yang dikunjungi didorong ke dalam tumpukan, dan kedua jika tidak ada titik lain maka titik yang dikunjungi didorong keluar dari tumpukan.

Perbedaan Antara BFS dan DFS:

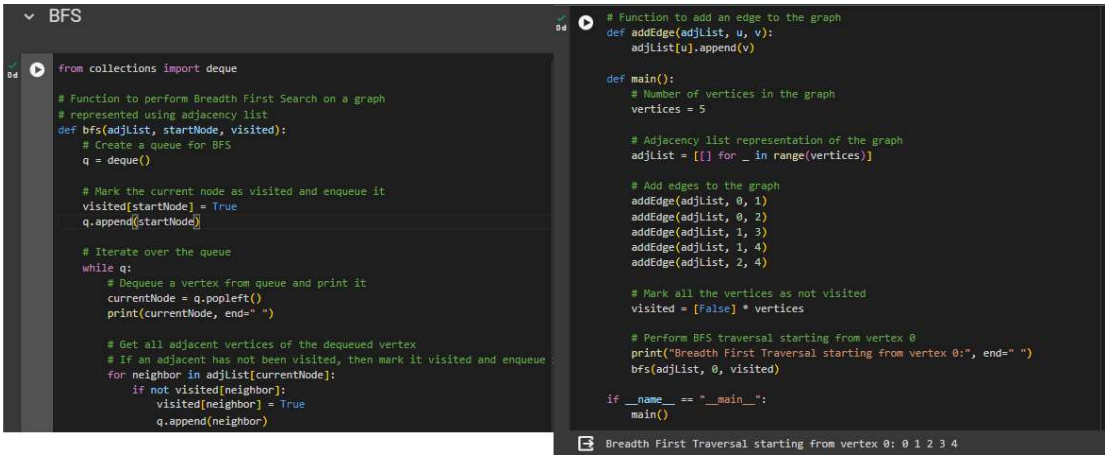
- Parameter: BFS singkatan dari Breadth First Search (Pencarian Lebar), sedangkan DFS singkatan dari Depth First Search (Pencarian Kedalaman).
- Struktur Data: BFS menggunakan struktur data Antrian (Queue) untuk mencari jalur terpendek, sementara DFS menggunakan struktur data Tumpukan (Stack).
- Definisi: BFS adalah pendekatan penelusuran di mana kita pertama-tama berjalan melalui semua node pada level yang sama sebelum pindah ke level berikutnya. DFS juga merupakan pendekatan penelusuran di mana penelusuran dimulai dari node akar dan berlanjut melalui node sejauh mungkin hingga mencapai node tanpa node yang belum dikunjungi di sekitarnya.
- Perbedaan Konseptual: BFS membangun pohon level demi level, sementara DFS membangun pohon sub-pohon demi sub-pohon.
- Pendekatan yang Digunakan: BFS bekerja berdasarkan konsep FIFO (First In First Out), sedangkan DFS bekerja berdasarkan konsep LIFO (Last In First Out).
- Cocok untuk: BFS lebih cocok untuk mencari simpul yang lebih dekat dengan sumber yang diberikan, sementara DFS lebih cocok ketika ada solusi jauh dari sumber.
- Aplikasi: BFS digunakan dalam berbagai aplikasi seperti graf bipartit, jalur terpendek, dll. Sedangkan DFS digunakan dalam berbagai aplikasi seperti graf asiklik dan menemukan komponen terhubung kuat.

7.3.1 Breadth First Search (BFS)

Algoritma BFS adalah sebagai berikut :

- Inisialisasi: Masukkan node awal ke dalam sebuah antrian dan tandai sebagai telah dikunjungi.
- Eksplorasi: Selama antrian tidak kosong:
- Ambil node dari antrian dan kunjungi (misalnya, cetak nilainya).

- Untuk setiap tetangga yang belum dikunjungi dari node yang diambil dari antrian:
- Masukkan tetangga ke dalam antrian.
- Tandai tetangga sebagai telah dikunjungi.



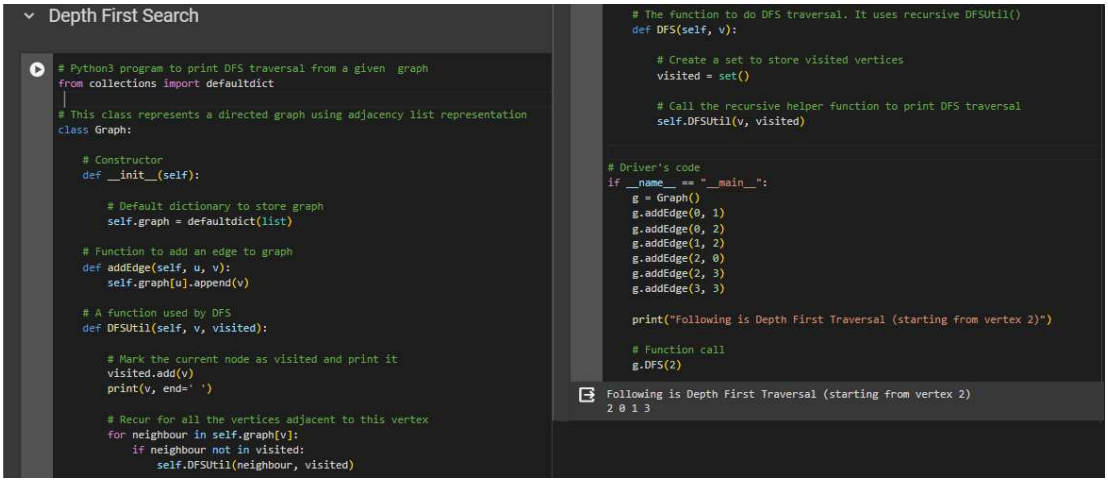
- Impor modul deque dari pustaka collections. deque digunakan di sini untuk mengimplementasikan antrian (queue), yang diperlukan untuk BFS.
- Didefinisikan sebuah fungsi bfs(adjList, startNode, visited) yang akan melakukan penelusuran BFS pada graf. Parameter yang diterima adalah:
    - adjList: representasi daftar ke tetangga dari graf.
    - startNode: simpul awal dari BFS.
    - visited: himpunan yang digunakan untuk melacak simpul-simpul yang telah dikunjungi.
- Dalam fungsi bfs, sebuah antrian (deque) dibuat untuk menyimpan simpul yang akan dikunjungi selama BFS.
- Simpul awal (startNode) ditandai sebagai telah dikunjungi dan dimasukkan ke dalam antrian.
- Selama antrian tidak kosong, program akan mengambil simpul terdepan dari antrian dan mencetaknya.
- Untuk setiap tetangga dari simpul yang diambil dari antrian, jika tetangga tersebut belum dikunjungi, maka akan ditandai sebagai telah dikunjungi dan dimasukkan ke dalam antrian.
- Didefinisikan fungsi addEdge(adjList, u, v) untuk menambahkan tepian ke graf.
- Di dalam fungsi main, dibuat sebuah daftar kosong adjList yang akan menyimpan representasi daftar ke tetangga dari graf.
- Beberapa tepian ditambahkan ke graf menggunakan fungsi addEdge.

- Semua simpul ditandai sebagai belum dikunjungi.
- Panggilan fungsi bfs(adjList, 0, visited) dilakukan untuk memulai penelusuran BFS dari simpul 0 dalam graf.
- Hasil penelusuran BFS dicetak setelah selesai.

7.3.2 *Depth First Search* (DFS)

Berikut adalah algoritma DFS secara umum:

- Mulai dari sebuah simpul awal.
- Tandai simpul awal tersebut sebagai "dikunjungi".
- Untuk setiap simpul tetangga yang belum dikunjungi dari simpul saat ini, lakukan langkah-langkah berikut:
  a. Kunjungi simpul tetangga tersebut.
  b. Rekursif, lakukan DFS pada simpul tetangga tersebut.
- Ulangi langkah-langkah 3 sampai tidak ada simpul tetangga yang belum dikunjungi.
- Jika masih ada simpul yang belum dikunjungi di graf, pilih simpul tersebut sebagai simpul awal dan ulangi langkah-langkah 2-4.
- Selesai.

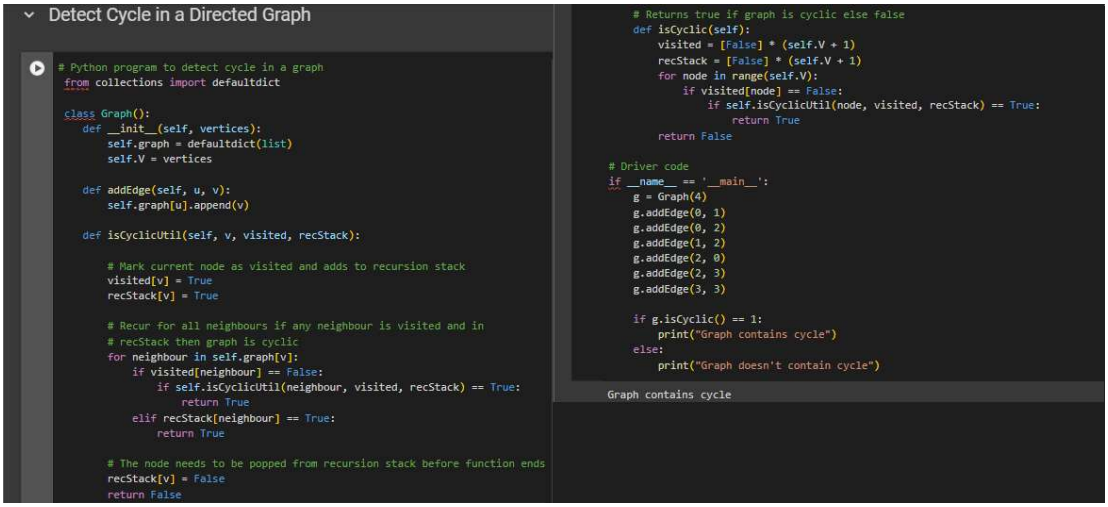


- Import modul defaultdict dari pustaka collections. Modul ini digunakan untuk membuat kamus (dictionary) yang secara otomatis menginisialisasi setiap kunci baru ke nilai default yang disediakan.

- Didefinisikan kelas Graph yang mewakili sebuah graf berarah menggunakan representasi daftar ke tetangga (adjacency list representation). Metode dan atribut kelas yang dimiliki adalah:
    - __init__(self): Konstruktor untuk inisialisasi graf, di mana self.graph adalah kamus yang akan menyimpan daftar tetangga dari setiap simpul.
    - addEdge(self, u, v): Metode untuk menambahkan tepian (edge) ke graf dari simpul u ke simpul v.
    - DFSUtil(self, v, visited): Metode utilitas rekursif yang digunakan oleh DFS. Ini akan menandai simpul saat ini sebagai telah dikunjungi, mencetaknya, dan kemudian melakukan pemanggilan rekursif untuk setiap tetangga yang belum dikunjungi.
    - DFS(self, v): Metode untuk melakukan penelusuran DFS dari simpul v. Ini membuat sebuah himpunan untuk menyimpan simpul yang telah dikunjungi dan kemudian memanggil metode utilitas DFS untuk memulai penelusuran dari simpul v.
- Di dalam blok kode utama (if __name__ == "__main__":), objek graf g dibuat dari kelas Graph. Kemudian, beberapa tepian ditambahkan ke graf menggunakan metode addEdge.
- Program mencetak pesan "Following is Depth First Traversal (starting from vertex 2)".
- Panggilan fungsi g.DFS(2) dilakukan. Ini akan memulai penelusuran DFS dari simpul 2 dalam graf. Selama penelusuran, setiap simpul yang dikunjungi akan dicetak sesuai dengan urutan DFS.

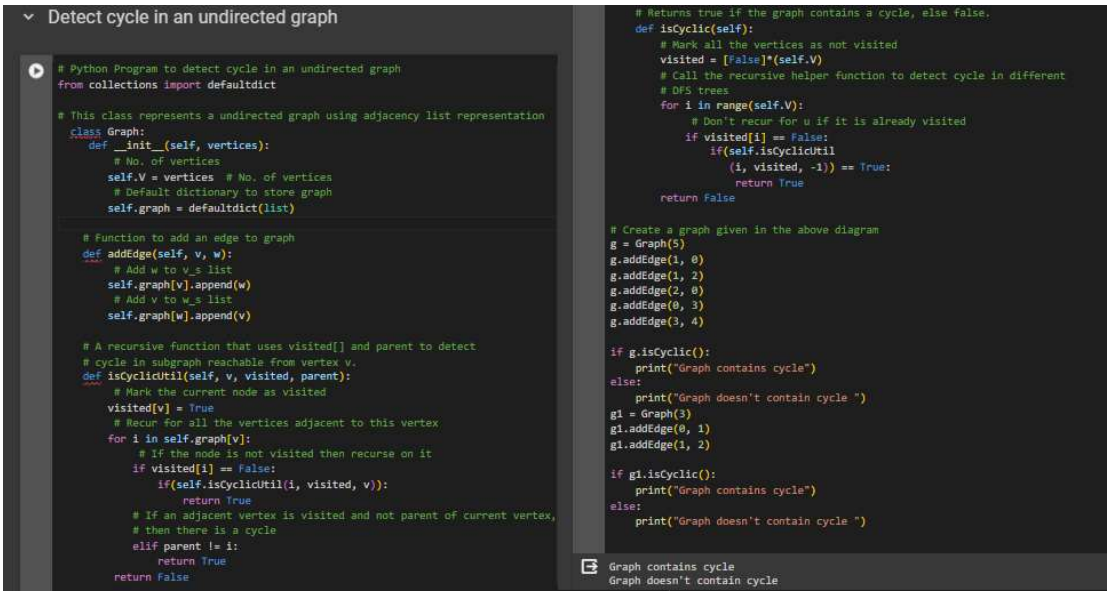
7.4 Siklus

Dalam teori graf, siklus merujuk kepada serangkaian tepian yang membentuk jalur tertutup di dalam graf. Jalur tersebut dimulai dan berakhir di simpul yang sama, dan tidak ada tepian yang dilewati dua kali, kecuali simpul awal dan akhir yang sama.



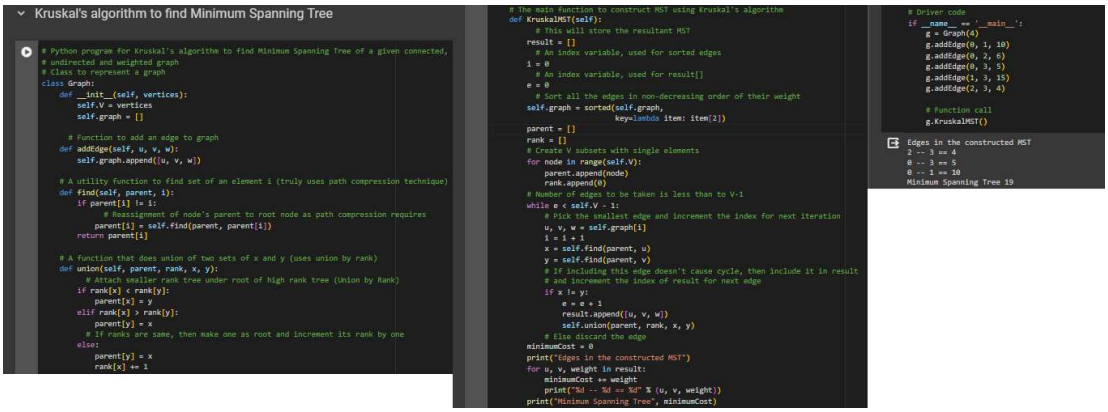
- Kelas Graph digunakan untuk merepresentasikan graf menggunakan struktur data daftar ke tetangga (adjacency list) dan memiliki metode-metode berikut:
    - __init__(self, vertices): Konstruktor kelas untuk menginisialisasi graf dengan jumlah simpul (vertices) yang diberikan. Graf direpresentasikan menggunakan kamus (graph) yang menghubungkan setiap simpul dengan daftar tetangga-tetangganya. Variabel V menyimpan jumlah total simpul dalam graf.
    - addEdge(self, u, v): Metode untuk menambahkan tepian antara dua simpul u dan v ke dalam graf. Ini menambahkan simpul v ke dalam daftar tetangga simpul u.
    - isCyclicUtil(self, v, visited, recStack): Metode rekursif yang digunakan untuk mendeteksi siklus dalam subgraf yang dapat dijangkau dari simpul v. Parameternya adalah simpul saat ini (v), array visited untuk melacak simpul yang telah dikunjungi, dan array recStack untuk melacak simpul yang sedang diproses dalam rekursi saat ini.
- Metode isCyclic digunakan untuk memeriksa keberadaan siklus dalam graf secara keseluruhan. Algoritma yang digunakan adalah DFS yang diimplementasikan dalam metode isCyclicUtil. Setiap simpul di graf diperiksa satu per satu. Jika pada saat DFS menemukan simpul yang sudah pernah dikunjungi dan masih berada dalam tumpukan rekursi (recStack), itu menandakan adanya siklus dalam graf.
- Di dalam blok kode utama, objek graf g dibuat dengan 4 simpul dan beberapa tepian ditambahkan ke graf menggunakan metode addEdge.
- Program memanggil metode isCyclic untuk memeriksa apakah graf mengandung siklus atau tidak. Jika isCyclic mengembalikan True, itu berarti graf mengandung siklus, dan pesan "Graph contains cycle" dicetak. Jika isCyclic mengembalikan

False, itu berarti graf tidak mengandung siklus, dan pesan "Graph doesn't contain cycle" dicetak.



- Dalam program diatas, kelas Graph digunakan untuk merepresentasikan graf tak berarah menggunakan representasi daftar ke tetangga (adjacency list representation).
- Metode __init__ digunakan untuk menginisialisasi graf dengan jumlah simpul (vertices) yang diberikan. Atribut V digunakan untuk menyimpan jumlah simpul, dan graph adalah kamus yang menyimpan daftar tetangga untuk setiap simpul.
- Metode addEdge digunakan untuk menambahkan tepian ke graf. Ini bekerja dengan menambahkan simpul yang saling berhubungan ke daftar tetangga satu sama lain.
- Metode isCyclicUtil adalah fungsi rekursif yang digunakan untuk mendeteksi keberadaan siklus dalam subgraf yang dapat dijangkau dari simpul tertentu (v). Itu menggunakan array visited untuk melacak simpul-simpul yang telah dikunjungi dan parameter parent untuk melacak simpul induk dalam penelusuran saat ini.
- Metode isCyclic digunakan untuk memeriksa keberadaan siklus dalam graf secara keseluruhan. Ini menciptakan array visited untuk menyimpan status kunjungan setiap simpul, dan kemudian memanggil isCyclicUtil untuk setiap simpul yang belum dikunjungi. Jika isCyclicUtil mengembalikan True untuk salah satu simpul, maka graf memiliki siklus dan fungsi mengembalikan True. Jika tidak ada siklus yang ditemukan, fungsi mengembalikan False.
- Program kemudian membuat objek graf g dan menambahkan tepian ke graf sesuai dengan contoh yang diberikan.

- Program kemudian memanggil metode isCyclic untuk memeriksa apakah graf g mengandung siklus atau tidak, dan mencetak pesan sesuai dengan hasilnya.
- Hal yang sama dilakukan untuk objek graf g1, yang memiliki sedikit perbedaan dalam struktur tepian.



- Kelas Graph digunakan untuk merepresentasikan graf. Ini memiliki atribut V untuk menyimpan jumlah simpul dan graph yang merupakan daftar yang akan menyimpan semua tepian bersama dengan bobotnya.
- Metode __init__ digunakan untuk menginisialisasi objek Graf dengan jumlah simpul (vertices) yang diberikan. Graf dinyatakan dalam bentuk daftar kosong.
- Metode addEdge digunakan untuk menambahkan tepian ke graf. Ini menerima tiga parameter: dua simpul yang dihubungkan oleh tepian (u dan v) dan bobotnya (w). Tepian ini kemudian ditambahkan ke daftar graph.
- Metode find digunakan untuk menemukan set dari sebuah elemen i. Ini merupakan implementasi dari teknik path compression dalam struktur data disjoint set. Ini berfungsi untuk menemukan root dari set yang mengandung elemen i.
- Metode union digunakan untuk melakukan penggabungan dua set, yaitu set x dan set y. Ini menggunakan teknik union by rank, di mana yang lebih kecil dari dua set ditambahkan sebagai anak dari yang lebih besar. Jika kedua set memiliki peringkat yang sama, salah satu dipilih sebagai root dan peringkatnya ditingkatkan.
- Metode KruskalMST adalah inti dari algoritma Kruskal. Ini melakukan langkah-langkah berikut:
    - Mengurutkan semua tepian berdasarkan bobotnya dalam urutan non-menaik.
    - Inisialisasi array parent dan rank untuk menyimpan informasi tentang struktur disjoint set.

- Membuat MST secara bertahap dengan memilih tepian terkecil satu per satu dan memasukkannya ke dalam MST jika tidak membentuk siklus dengan tepian yang sudah ada.
- Akhirnya, mencetak semua tepian dalam MST dan bobot total MST.

- Di dalam blok kode utama, objek graf g dibuat dengan 4 simpul dan beberapa tepian ditambahkan ke graf menggunakan metode addEdge.
- Program kemudian memanggil metode KruskalMST untuk membangun Minimum Spanning Tree (MST) menggunakan algoritma Kruskal.

Latihan

Buatlah implementasi dari Topologi Sort dalam bahasa pemrograman python dengan ketentuan sebagai berikut :

Topological Sort adalah metode pengurutan simpul dalam sebuah graf berarah yang memastikan bahwa setiap simpul diposisikan sebelum simpul-simpul yang dituju.

- topologicalSortUtil(v, adj, visited, stack): Ini adalah fungsi utilitas yang melakukan pencarian rekursif untuk mencari urutan topologis dari simpul v dalam graf yang direpresentasikan oleh adj (adjacency list). Fungsi ini menerima argumen v (simpul saat ini), adj (daftar ke tetangga-tetangga setiap simpul), visited (array boolean untuk menandai apakah suatu simpul sudah dikunjungi), dan stack (tumpukan untuk menyimpan urutan topologis).
- Pertama-tama, fungsi menandai simpul saat ini (v) sebagai telah dikunjungi (visited[v] = True).
- Kemudian, fungsi melakukan rekursi untuk semua simpul tetangga dari simpul saat ini (adj[v]). Jika simpul tetangga belum dikunjungi, maka fungsi topologicalSortUtil akan dipanggil lagi untuk simpul tersebut.
- Terakhir, simpul saat ini (v) dimasukkan ke dalam tumpukan stack. Ini karena setelah semua simpul tetangga dari simpul saat ini diproses, simpul saat ini dapat ditambahkan ke urutan topologis.
- topologicalSort(adj, V): Fungsi ini melakukan pemanggilan utama untuk pengurutan topologis. Fungsi ini menerima daftar ke tetangga-tetangga setiap simpul (adj) dan jumlah total simpul (V).
- Inisialisasi tumpukan kosong (stack) dan array visited untuk semua simpul sebagai False.
- Fungsi melakukan panggilan ke topologicalSortUtil untuk setiap simpul yang belum dikunjungi. Ini memastikan bahwa setiap simpul akan diproses.
- Setelah semua simpul diproses, fungsi akan mencetak isi tumpukan secara berurutan, yang merupakan urutan topologis dari graf.
- Di blok if __name__ == "__main__":, program menetapkan jumlah simpul (V) dan daftar edges yang menunjukkan hubungan antara simpul-simpul dalam graf. Kemudian, graf direpresentasikan sebagai daftar ke tetangga-tetangga setiap simpul (adj). Akhirnya, fungsi topologicalSort dipanggil dengan argumen yang sesuai untuk melakukan pengurutan topologis pada graf yang diberikan.

Hasil cetakan adalah urutan topologis dari graf yang diberikan.