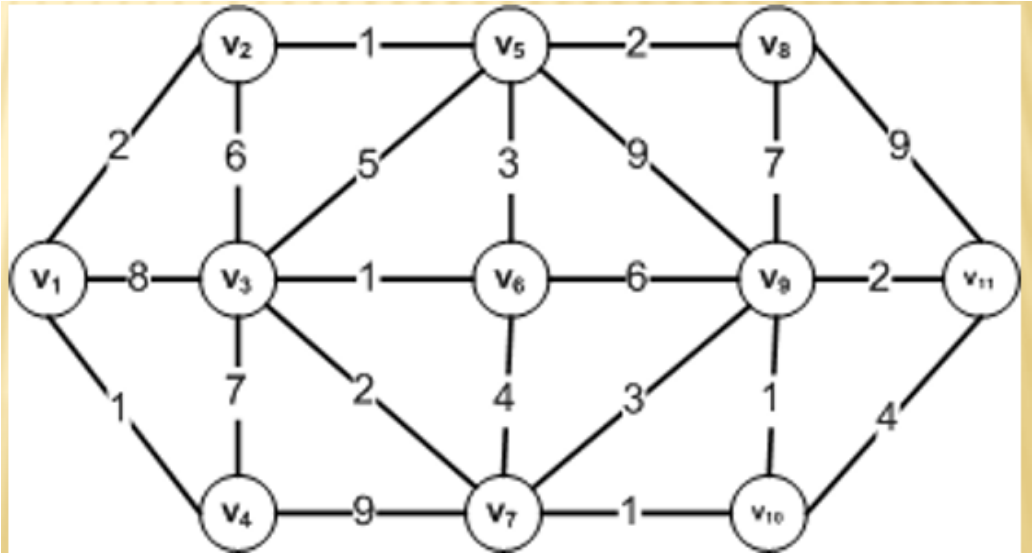
TUGAS MINGGUAN 1

Nama : Muhammad Daffa Rizky Sutrisno

NRP : 5025231207

Graph :



1. Temukan 5 algoritma yang disebutkan pada permasalahan “Shortest Path Problem” dan aplikasikan 5 algoritma tsb utk mencari rute terpendek dari V1 menuju V11 pada weighted graph di atas !

a. Algoritma Dijkstra

Algoritma Dijkstra adalah metode greedy yang bertujuan untuk menemukan jalur terpendek dari satu titik ke semua titik lainnya dalam graf berbobot positif. Proses utamanya melibatkan penilaian jalur-jalur yang ditemukan, memilih yang terpendek, dan melanjutkan ke tetangga simpul.

Tahapan:

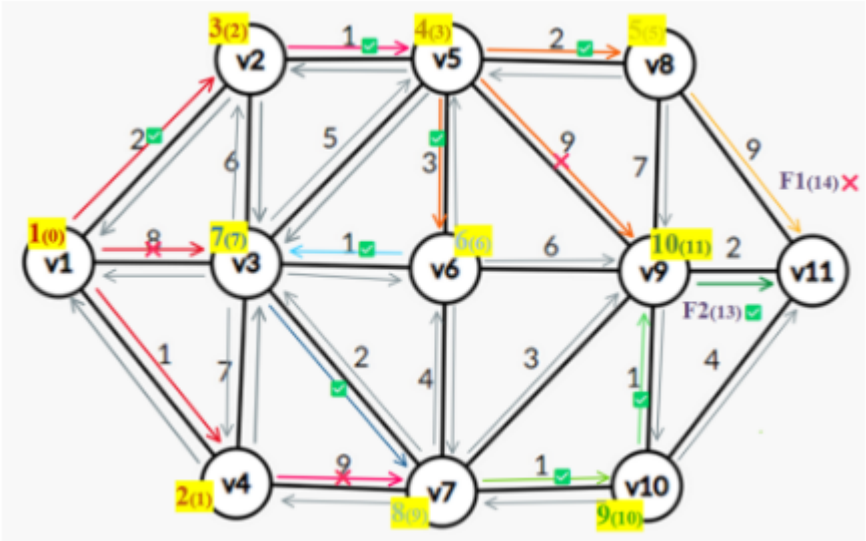
1. Inisialisasi vertex awal dengan jarak 0, dan semua vertex lainnya dengan jarak tak terhingga.
2. Pilih vertex dengan jarak terpendek yang belum dikunjungi.

3. Perbarui jarak semua tetangga vertex tersebut jika melewati vertex ini menghasilkan jalur yang lebih pendek.
4. Tandai vertex yang sudah dikunjungi dan lanjutkan ke vertex terdekat berikutnya.

Potongan program :

```python
def dijkstra(graph, start, goal):
    # Inisialisasi jarak awal untuk setiap simpul
    distances = {node: float('infinity') for node in graph}
    distances[start] = 0
    priority_queue = [(0, start)]
    visited = set()

    # Menyimpan jalur untuk setiap simpul
    path = {start: None}

    step = 1
    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)

        if current_node in visited:
            continue

        visited.add(current_node)

        print(f"Step {step}: Visiting Node: {current_node}, Distance from Start: {current_distance}")
        step += 1

        # Jika mencapai tujuan, keluar dari loop
        if current_node == goal:
            break

        # Proses tetangga-tetangga dari simpul saat ini
        for neighbor, weight in graph[current_node]:
            distance = current_distance + weight

            # Jika ditemukan jalur yang lebih pendek, update jaraknya
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))
                path[neighbor] = current_node
                print(f"   -> Updating path to {neighbor}: New Distance = {distance}")

    # Melacak kembali jalur dari tujuan ke awal
    full_path = []
    current = goal
    while current is not None:
        full_path.insert(0, current)
        current = path[current]

    return distances[goal], full_path
```

Ilustrasi :



Urutan output :

```
Step 1: Visiting Node: V1, Distance from Start: 0
   -> Updating path to V2: New Distance = 2
   -> Updating path to V3: New Distance = 8
   -> Updating path to V4: New Distance = 1
Step 2: Visiting Node: V4, Distance from Start: 1
   -> Updating path to V7: New Distance = 10
Step 3: Visiting Node: V2, Distance from Start: 2
   -> Updating path to V5: New Distance = 3
Step 4: Visiting Node: V5, Distance from Start: 3
   -> Updating path to V6: New Distance = 6
   -> Updating path to V8: New Distance = 5
Step 5: Visiting Node: V8, Distance from Start: 5
   -> Updating path to V9: New Distance = 12
   -> Updating path to V11: New Distance = 14
Step 6: Visiting Node: V6, Distance from Start: 6
   -> Updating path to V3: New Distance = 7
Step 7: Visiting Node: V3, Distance from Start: 7
Step 8: Visiting Node: V7, Distance from Start: 10
   -> Updating path to V10: New Distance = 11
Step 9: Visiting Node: V10, Distance from Start: 11
Step 10: Visiting Node: V9, Distance from Start: 12
Step 11: Visiting Node: V11, Distance from Start: 14

Shortest path from V1 to V11: V1 -> V2 -> V5 -> V8 -> V11
Shortest distance from V1 to V11: 14
```

b. Algoritma Bellman-Ford

Algoritma ini dirancang untuk menemukan jalur terpendek pada graf berbobot, termasuk yang memiliki bobot negatif. Algoritma ini mengupdate jarak dengan melakukan iterasi pada semua edge graf beberapa kali.

Tahapan:

1. Inisialisasi jarak awal dari vertex sumber ke semua vertex lain sebagai tak terhingga, kecuali vertex awal yang memiliki jarak 0.
2. Lakukan iterasi (jumlah vertex - 1) kali, dan periksa apakah ada jalur yang lebih pendek dari vertex saat ini.
3. Setelah iterasi terakhir, lakukan pengecekan untuk mendeteksi siklus negatif.

Potongan program :

```python
    # function to add an edge to graph
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def BellmanFord(self, src):

        dist = [float("Inf")] * self.V
        dist[src] = 0
        print(dist)

        for _ in range(self.V - 1):
            change = False
            for x in range(11):

                for u, v, w in self.graph:
                    if x == u-1 and dist[x] > dist[v-1] + w:
                        dist[x] = dist[v-1] + w
                        change = True
                    if x == v-1 and dist[x] > dist[u-1] + w:
                        dist[x] = dist[u-1] + w
                        change = True
            print(dist)
            if change == False:
                break
```

Ilustrasi :

Iterasi	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	2	8	1	3	6	10	5	12	11	14
3	0	2	8	1	3	6	9	5	12	10	14
4	0	2	7	1	3	6	9	5	11	10	13
5	0	2	7	1	3	6	9	5	11	10	**13**

Output :

```
[0, inf, inf, inf, inf, inf, inf, inf, inf, inf, inf]
[0, 2, 8, 1, 3, 6, 10, 5, 12, 11, 14]
[0, 2, 7, 1, 3, 6, 9, 5, 12, 10, 14]
[0, 2, 7, 1, 3, 6, 9, 5, 11, 10, 13]
[0, 2, 7, 1, 3, 6, 9, 5, 11, 10, 13]
```

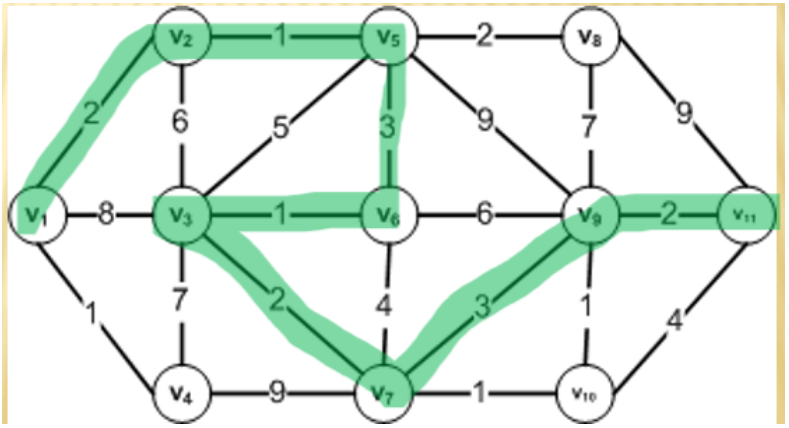
c. Algoritma A*

Algoritma A* menggabungkan prinsip Dijkstra dengan heuristik untuk memperkirakan jarak dari satu simpul ke simpul tujuan. Algoritma ini efektif dalam pencarian jalur pada graf besar.

Tahapan:

1. Hitung fungsi evaluasi f(n) = g(n) + h(n), di mana g(n) adalah cost sampai simpul n, dan h(n) adalah estimasi jarak ke tujuan.
2. Pilih simpul dengan nilai f(n) terkecil dan lanjutkan prosesnya sampai tujuan tercapai.

   Dikarenakan pada soal tidak disediakan informasi mengenai nilai heuristik pada tiap vertexnya, maka dari itu kami mengatur nilai heuristik tiap vertexnya menjadi nol sehingga hasil akhir dari algoritma ini mirip dengan algoritma Dijkstra. Berikut adalah hasil dari implementasi algoritma A* pada soal menggunakan Python:



d. Algoritma Floyd-Warshall

Algoritma ini digunakan untuk menemukan jalur terpendek antara semua pasangan vertex dalam satu eksekusi. Algoritma ini efisien untuk graf lengkap, termasuk graf dengan bobot negatif.

Tahapan:

1. Inisialisasi matriks jarak antara setiap pasangan vertex.
2. Lakukan iterasi untuk setiap vertex, memperbarui matriks jarak jika ditemukan jalur yang lebih pendek.
3. Matriks akhir akan berisi jarak terpendek antara semua vertex.

Potongan program :

```
35      def floydWarshall(self):
36          dist = list(map(lambda i: list(map(lambda j: j, i)), self.graph))
37          self.printSolution(dist)
38          for k in range(self.V):
39              for i in range(self.V):
40                  for j in range(self.V):
41                      dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
42          self.printSolution(dist)
```

Output :

```
Following matrix shows the shortest distances between every pair of vertices
    0       2       7       1       3       6       9       5       11      10      13
    2       0       5       3       1       4       7       3       9       8       11
    7       5       0       7       4       1       2       6       4       3       6
    1       3       7       0       4       7       9       6       11      10      13
    3       1       4       4       0       3       6       2       8       7       10
    6       4       1       7       3       0       3       5       5       4       7
    9       7       2       9       6       3       0       8       2       1       4
    5       3       6       6       2       5       8       0       7       8       9
    11      9       4       11      8       5       2       7       0       1       2
    10      8       3       10      7       4       1       8       1       0       3
    13      11      6       13      10      7       4       9       2       3       0
```

e. Algoritma Johnson

Algoritma Johnson digunakan untuk graf dengan bobot negatif. Langkah utama dalam algoritma ini adalah mengubah bobot negatif menjadi non-negatif sebelum menjalankan Dijkstra pada setiap vertex.

Tahapan:

1. Tambahkan vertex baru ke graf dan hubungkan dengan semua vertex lain dengan bobot 0.
2. Gunakan algoritma Bellman-Ford untuk menghitung jarak dari vertex baru ke setiap vertex.
3. Sesuaikan bobot semua edge agar non-negatif, lalu jalankan algoritma Dijkstra.

Pada kasus graph di soal tidak terdapat weight yang negatif sehingga algoritma Johnson akan menghasilkan graph baru yang sama dengan graph aslinya.

```
Step 1: Augment the graph with a new vertex and zero-weight edges:
v1: {'v2': 2, 'v3': 8, 'v4': 1}
v2: {'v1': 2, 'v3': 6, 'v5': 1}
v3: {'v1': 8, 'v2': 6, 'v4': 7, 'v5': 5, 'v6': 1, 'v7': 2}
v4: {'v1': 1, 'v3': 7, 'v7': 9}
v5: {'v2': 1, 'v3': 5, 'v6': 3, 'v8': 2, 'v9': 9}
v6: {'v3': 1, 'v5': 3, 'v7': 4, 'v9': 6}
v7: {'v3': 2, 'v4': 9, 'v6': 4, 'v9': 3, 'v10': 1}
v8: {'v5': 2, 'v9': 7, 'v11': 9}
v9: {'v5': 9, 'v6': 6, 'v7': 3, 'v8': 7, 'v10': 1, 'v11': 2}
v10: {'v7': 1, 'v9': 1, 'v11': 4}
v11: {'v8': 9, 'v9': 2, 'v10': 4}
new_vertex: {'v1': 0, 'v2': 0, 'v3': 0, 'v4': 0, 'v5': 0, 'v6': 0, 'v7': 0, 'v8': 0, 'v9': 0, 'v10': 0, 'v11': 0}

Step 2: Run Bellman-Ford algorithm to find distances from the new vertex:
{'v1': 0, 'v2': 0, 'v3': 0, 'v4': 0, 'v5': 0, 'v6': 0, 'v7': 0, 'v8': 0, 'v9': 0, 'v10': 0, 'v11': 0, 'new_vertex': 0}

Step 3: Adjust original edge weights to make them non-negative:
v1: {'v2': 2, 'v3': 8, 'v4': 1}
v2: {'v1': 2, 'v3': 6, 'v5': 1}
v3: {'v1': 8, 'v2': 6, 'v4': 7, 'v5': 5, 'v6': 1, 'v7': 2}
v4: {'v1': 1, 'v3': 7, 'v7': 9}
v5: {'v2': 1, 'v3': 5, 'v6': 3, 'v8': 2, 'v9': 9}
v6: {'v3': 1, 'v5': 3, 'v7': 4, 'v9': 6}
v7: {'v3': 2, 'v4': 9, 'v6': 4, 'v9': 3, 'v10': 1}
v8: {'v5': 2, 'v9': 7, 'v11': 9}
v9: {'v5': 9, 'v6': 6, 'v7': 3, 'v8': 7, 'v10': 1, 'v11': 2}
v10: {'v7': 1, 'v9': 1, 'v11': 4}
v11: {'v8': 9, 'v9': 2, 'v10': 4}

Step 4: Run Dijkstra's algorithm to find shortest paths:

Shortest distance from v1 to v11: 13
```

2. Lakukan analisis mengenai kelebihan dan kekurangan masing2 algoritma !

a. Algoritma Dijkstra

Kelebihan:

- Efisien untuk graf dengan bobot tepi non-negatif: Algoritma ini sangat cepat dalam menemukan jalur terpendek pada graf yang tidak memiliki bobot negatif.
- Menggunakan metode greedy: Hanya mengeksplorasi simpul yang dipandang paling menjanjikan, sehingga tidak mengunjungi simpul yang tidak perlu.
- Optimal untuk jalur terpendek tunggal: Sangat cocok untuk menghitung jalur terpendek dari satu simpul sumber ke semua simpul lainnya.

Kekurangan:

- Tidak bisa menangani bobot negatif: Jika graf memiliki bobot negatif, algoritma ini mungkin tidak akan menemukan solusi yang benar.
- Bukan untuk graf yang sangat besar: Meskipun efisien, bisa menjadi lambat pada graf yang sangat besar tanpa menggunakan struktur data yang lebih baik seperti heap Fibonacci.

b. Algoritma Bellman-Ford

Kelebihan:

- Dapat menangani bobot negatif: Algoritma ini mampu menangani graf dengan bobot negatif dan dapat mendeteksi siklus dengan bobot negatif.
- Lebih umum daripada Dijkstra: Tidak ada batasan bahwa bobot harus positif, membuat algoritma ini lebih fleksibel untuk berbagai jenis graf.

Kekurangan:

- Kurang efisien: Bellman-Ford cenderung lebih lambat dibandingkan dengan Dijkstra karena mengunjungi semua simpul dan tepi $V-1V-1V-1$ kali.
- Tidak seoptimal untuk jalur tunggal: Jika hanya butuh jalur terpendek dari satu sumber ke satu tujuan, Bellman-Ford bisa terlalu lambat dibandingkan algoritma lain.

c. Algoritma A*

Kelebihan:

- Menggunakan heuristik untuk mempercepat pencarian: Jika heuristik yang digunakan tepat, A* bisa lebih cepat dari Dijkstra karena bisa melewati simpul yang tidak relevan.
- Optimal dan lengkap: Jika heuristiknya tidak melebih-lebihkan biaya, A* selalu menemukan solusi optimal.

Kekurangan:

- Sangat bergantung pada heuristik: Jika heuristiknya buruk atau tidak akurat, A* bisa bekerja seperti Dijkstra atau bahkan lebih lambat.
- Kompleksitas memori tinggi: Karena perlu menyimpan banyak simpul dalam memori, algoritma ini bisa menjadi tidak efisien dalam hal penggunaan memori.

d. Algoritma Floyd-Warshall

Kelebihan:

- Menghitung jalur terpendek untuk semua pasangan: Sangat efisien untuk graf padat di mana perlu diketahui jalur terpendek untuk setiap pasangan simpul.
- Sederhana dan langsung: Implementasi algoritma ini sangat sederhana dan intuitif untuk dipahami.

Kekurangan:

- Tidak efisien untuk graf besar dan jarang (sparse): Floyd-Warshall memiliki kompleksitas waktu O(V3)O(V^3)O(V3), yang tidak efisien untuk graf dengan jumlah simpul yang besar dan tepi yang sedikit.
- Menggunakan memori yang besar: Harus menyimpan matriks jarak untuk semua pasangan simpul, sehingga memerlukan ruang memori yang signifikan.

e. Algoritma Johnson

Kelebihan:

- Efisien untuk semua-pasangan pada graf besar dan jarang: Johnson menggabungkan kelebihan Bellman-Ford dan Dijkstra, membuatnya sangat efisien untuk graf besar yang jarang.
- Menangani bobot negatif: Dengan terlebih dahulu menggunakan Bellman-Ford untuk menangani bobot negatif, algoritma ini bisa bekerja di graf yang lebih kompleks.

Kekurangan:

- Lebih rumit: Karena menggunakan dua fase (Bellman-Ford dan Dijkstra), implementasi algoritma ini lebih kompleks daripada menggunakan satu algoritma langsung seperti Floyd-Warshall.
- Kurang optimal untuk graf kecil atau padat: Jika graf memiliki banyak simpul dan tepi, Floyd-Warshall bisa lebih cepat dan sederhana.

Kesimpulan

i. Dijkstra cocok untuk graf tanpa bobot negatif dan hanya memerlukan jalur terpendek dari satu sumber ke beberapa tujuan.

ii. Bellman-Ford fleksibel dalam menangani bobot negatif, tetapi lebih lambat dan cocok untuk kasus dengan bobot negatif atau ketika siklus negatif perlu dideteksi.

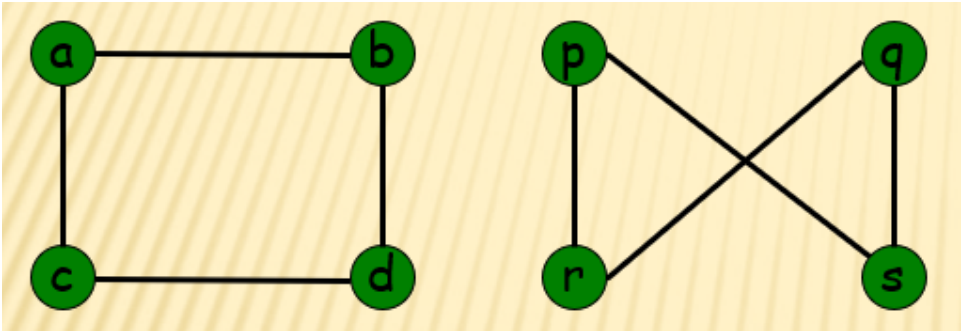
iii. A* berguna dalam pencarian jalur yang membutuhkan kecepatan dan efisiensi, terutama ketika ada heuristik yang baik.

iv. Floyd-Warshall berguna untuk menghitung jalur terpendek antara semua pasangan simpul, tetapi tidak efisien pada graf yang besar dan jarang.

v. Johnson ideal untuk graf besar dan jarang, terutama ketika ada bobot negatif.

3. Lakukan observasi pd pasangan2 graph di bawah utk mengetahui apakah pasangan graph2 tsb identik, isomorphic, atau tidak keduanya !

a. Pasangan Graph 1



Kedua graf pada gambar tersebut tidak isomorfik dan juga tidak identik. Dikarenakan tidak memenuhi kedua sifat tersebut.

1. Isomorfisme: Dua graf dikatakan isomorfik jika terdapat korespondensi satu-ke-satu antara himpunan simpulnya, sehingga hubungan (adjacency) antara simpul-simpulnya terjaga. Pada gambar:
    a. Graf di kiri (dengan simpul a,b,c,d) adalah graf siklik (membentuk loop tertutup seperti persegi).

b. Graf di kanan (dengan simpul p,q,r,s) adalah graf bipartit yang tepinya bersilangan, tetapi tidak membentuk siklus tertutup.

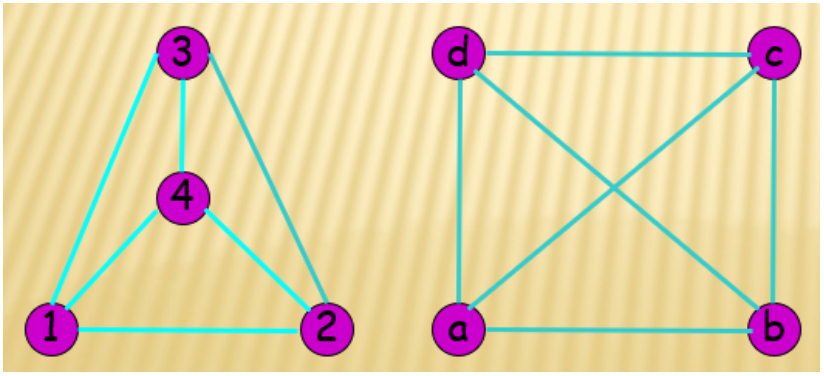
Meskipun kedua graf memiliki jumlah simpul dan sisi yang sama (masing-masing 4 simpul dan 4 sisi), struktur keseluruhan berbeda. Yang satu membentuk siklus tertutup, sementara yang lain membentuk pola "silang" atau "X." Oleh karena itu, kedua graf ini tidak isomorfik.

2. Identitas: Dua graf dikatakan identik jika tidak hanya memiliki jumlah simpul dan sisi yang sama, tetapi juga memiliki susunan simpul dan tepi yang sama. Dalam hal ini:
   a. Graf kiri membentuk siklus, sementara graf kanan memiliki susunan tepi yang bersilangan, yang berbeda dengan siklus di graf kiri.

   Karena susunan tepinya berbeda, kedua graf ini tidak identik.

Jadi, kedua graf tersebut tidak isomorfik dan tidak identik.

b. Pasangan Graph 2



Untuk menentukan apakah dua graf pada gambar tersebut isomorfik atau identik, kita perlu melihat struktur dan koneksi antar simpul pada kedua graf.

1. Graf di sebelah kiri:
   - Simpul-simpul: 1, 2, 3, 4
   - Tepi-tepi:
     - (1, 2), (1, 3), (1, 4)
     - (2, 3), (2, 4)
     - (3, 4)

   Ini adalah graf yang memiliki 4 simpul dan 6 tepi. Ini membentuk sebuah graf lengkap, yang biasa dikenal sebagai K4.

2. Graf di sebelah kanan:
   - Simpul-simpul: a, b, c, d
   - Tepi-tepi:
     - (a, b), (a, c), (a, d)

- (b, c), (b, d)
- (c, d)

Graf ini juga memiliki 4 simpul dan 6 tepi. Ini juga merupakan graf lengkap, K4.

3. Isomorfisme:
    a. Kedua graf memiliki jumlah simpul dan jumlah tepi yang sama.
    b. Koneksi antar simpul di kedua graf juga mirip: setiap simpul dihubungkan dengan simpul lainnya, artinya setiap simpul memiliki derajat (degree) yang sama di kedua graf (masing-masing simpul terhubung ke 3 simpul lainnya).

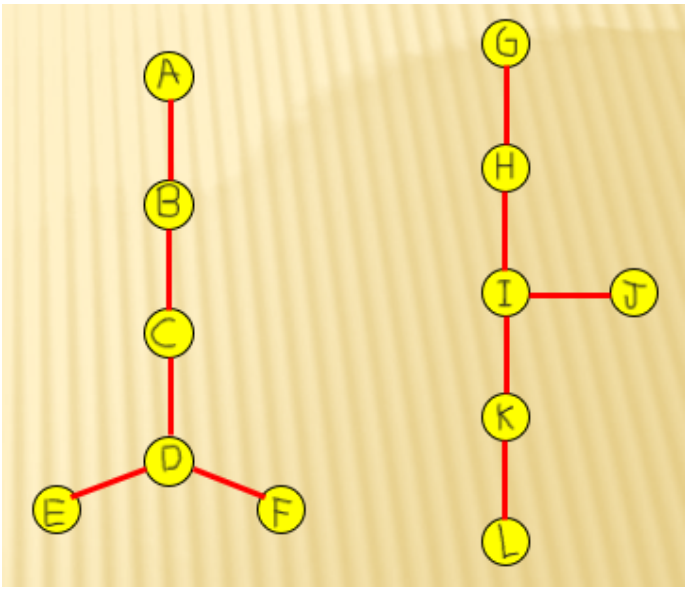
    Karena keduanya memiliki jumlah simpul, jumlah tepi, dan pola koneksi yang sama, kedua graf ini isomorfik. Isomorfik berarti mereka memiliki struktur yang sama, meskipun simpul-simpulnya mungkin diberi label yang berbeda.

4. Identitas:
    a. Kedua graf memiliki simpul dengan label yang berbeda (angka di graf kiri dan huruf di graf kanan).
    b. Bentuk grafnya juga terlihat berbeda (yang satu segitiga, yang lain persegi).

    Karena label simpulnya berbeda dan susunan visualnya berbeda, graf-graf ini tidak identik, namun isomorfik.

Jadi, kedua graf tersebut isomorfik tetapi tidak identik.

c. Pasangan Graph 3



Untuk menganalisis apakah kedua graf ini isomorfik atau identik, bisa dilihat dari masing-masing strukturnya.

1. Graf di sebelah kiri:

- **Simpul-simpul:** A, B, C, D, E, F
- **Struktur:**
  - A → B → C → D
  - D bercabang ke E dan F

Graf ini membentuk **pohon dengan cabang** pada simpul D, di mana simpul D menjadi titik utama yang bercabang.

2. Graf di sebelah kanan:

- **Simpul-simpul:** G, H, I, J, K, L
- **Struktur:**
  - G → H → I
  - I bercabang ke J dan K
  - K → L

Graf ini juga membentuk **pohon dengan cabang**, namun simpul I adalah titik utama yang bercabang, dengan K yang memiliki cabang tambahan ke L.

3. Analisis Isomorfik:

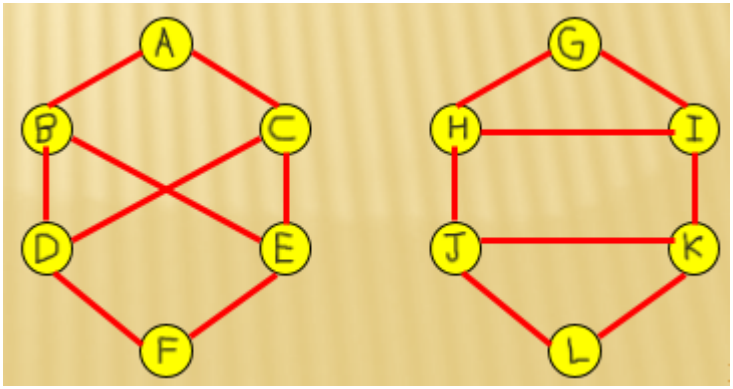
Ada masalah dalam korespondensi antara simpul C dan simpul I. Simpul C memiliki derajat 2, sedangkan simpul I memiliki derajat 3. **Ini menunjukkan bahwa graf-graf ini tidak isomorfik**, karena korespondensi antara derajat simpul tidak sepenuhnya terjaga.

4. Analisis Identik:

Kedua graf ini tidak identik. Meskipun mereka memiliki jumlah simpul dan tepi yang sama, susunan simpul dan tepi berbeda. Bentuk percabangan dari simpul ke simpul juga tidak sama.

Jadi, kedua graf tersebut tidak isomorfik dan tidak identik.

d. Pasangan Graph 4



Untuk menentukan apakah dua graf pada gambar tersebut isomorfik atau identik, perlu membandingkan beberapa hal:

1. Jumlah simpul dan sisi
    - Pada graf pertama (kiri), ada 6 simpul (A, B, C, D, E, F) dan jumlah sisinya adalah 9.
    - Pada graf kedua (kanan), juga ada 6 simpul (G, H, I, J, K, L) dan jumlah sisinya juga 9.
2. Adjacency
    - Pada graf pertama (kiri), hubungan antara simpul-simpul adalah:
        - A terhubung dengan B dan C.
        - B terhubung dengan A, D, dan E.
        - C terhubung dengan A, D, dan E.
        - D terhubung dengan B, C, dan F.
        - E terhubung dengan B, C, dan F.
        - F terhubung dengan D dan E.
    - Pada graf kedua (kanan), hubungan antara simpul-simpul adalah:
        - G terhubung dengan H dan I.
        - H terhubung dengan G, J, dan I.
        - I terhubung dengan G, H, dan K.
        - J terhubung dengan H dan L.
        - K terhubung dengan I dan L.
        - L terhubung dengan J dan K.
3. Isomorfisme

    Graf pertama dan kedua memiliki jumlah simpul dan sisi yang sama, namun ketika melihat secara lebih rinci hubungan antar simpul, pola adjacency tidak persis sama antara kedua graf.

    Berdasarkan teori **isomorfisme**, untuk menjadi isomorfik, harus ada korespondensi satu-ke-satu yang menjaga adjacency antara simpul. Dari analisis di atas, kedua graf tampaknya **tidak isomorfik** karena hubungan antar simpul yang berbeda.

4. Identitas

Berdasarkan teori **identitas**, kedua graf jelas tidak identik karena susunan simpul dan tepi (edge) mereka berbeda.

Jadi, kedua graf tersebut tidak isomorfik dan tidak identik.