**JAWABAN UJIAN AKHIR SEMESTER (UAS)**

**ALGORITMA ANALISIS**

Dosen Pengampu :

**Dr. Achmad Hindasyah, M.Si**

****

**OLEH**

**ASEP RIDWAN HIDAYAT**

**231012050036**

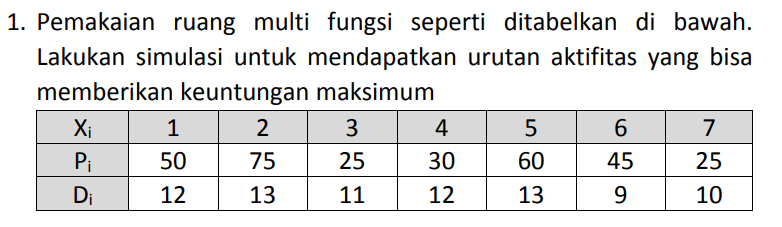
**TI 01MKME001 REGULAR C**

**PROGRAM STUDI MAGISTER TEKNIK INFORMATIKA**

**UNIVERSITAS PAMULANG**

**TANGERANG SELATAN**

**2024**

**SOAL NO. 1**

**Jawaban No 1**

1. **Algoritma Seleksi aktifitas**
2. Langkah pertama urutkan aktivitas berdasarkan Profit terbesar dari waktu finis terkecil (tercepat) hingga terlama
3. Pilih aktifitas pertama dari array yang sudah diurutkan
4. Eksekusi aktifitas berikutnya yang masih ada dalam array tersisa
5. Pilih aktifitas yang mempunyai waktu start lebih besar daripada waktu finish aktifitas sebelumnya
6. kondisi lain ditolak
7. Ulangi langkah 3 hingga semua aktifitas dieksekusi
8. **Implementasi Analitik**

Berikut ini implementasinya, Terdapat 7 job sequence dari table diatas, urutkan aktifitas yang memberikan profit maksimum

Xi = aktifitas ke-i

Pi = Profit ke-i

Di = Durasi dari aktifitas

1. Urutkan berdasarkan profit

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Xi | 2 | 5 | 1 | 6 | 4 | 7 | 3 |
| Pi | 75 | 60 | 50 | 45 | 30 | 25 | 25 |
| Di | 13 | 13 | 12 | 9 | 12 | 10 | 11 |

1. Urutkan sesuai urutan profit dan waktu

|  |  |
| --- | --- |
| Aktifitas (durasi ke-) | Profit |
| X2(13) | P = 75 |
| X2(13) + X5(13) | P = 75 + 60 = 135 |
| X2(13) + X5(13) + X6(9) | P = 75+60+45 = 180 |
|  |  |

Atau bisa juga digambarkan dengan tabel sebagai berikut:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aktivitas | Profit (Pi) | Waktu Penyelesaian (Di) | Waktu Mulai | Waktu Selesai |
| 2 | 75 | 13 | 1 | 13 |
| 5 | 60 | 13 | 14 | 26 |
| 6 | 45 | 9 | 27 | 35 |
| 1 | 50 | 12 | 36 | 47 |
| 4 | 30 | 12 | 48 | 59 |
| 7 | 25 | 10 | 60 | 69 |
| 3 | 25 | 11 | Tidak bisa dijadwalkan | Tidak bisa dijadwalkan |

**Jadi Urutan maksimum didapat [2, 5,6,], Profit maksimum adalah 180**

Note:

Aktifitas 6 dipilih terlebih dahulu dibanding aktifitas 1 karena memiliki profit besar (45) dan membutuhkan waktu penyelesaian yang lebih singkat (9). Aktivitas 1, 4, 7, dan 3 tidak dapat dimulai sebelum aktivitas 6 selesai karena akan bertabrakan dengan waktu penyelesaian aktivitas 6 yang sudah dijadwalkan.

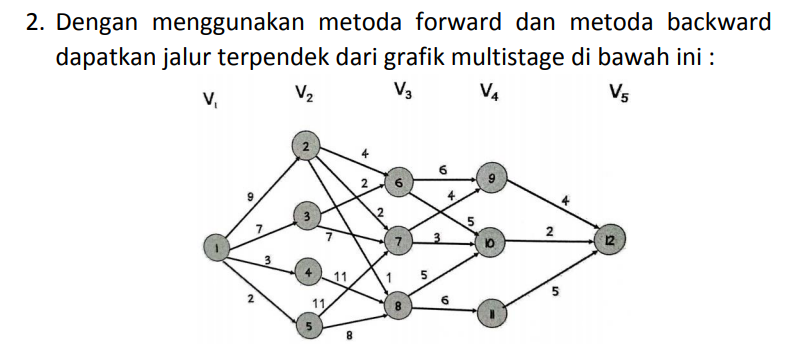
1. **Berikut pemograman menggunakan Bahasa python**

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  def maximize\_profit(n, activities):      Pi = [activities[x][0] for x in range(1, n + 1)]      Di = [activities[x][1] for x in range(1, n + 1)]        activities\_with\_values = list(zip(range(1, n + 1), Pi, Di))      activities\_with\_values.sort(key=lambda x: x[1] / x[2], reverse=True)        max\_profit = 0      current\_time = 0      chosen\_activities = []      cumulative\_profit = []        for activity in activities\_with\_values:          if current\_time + activity[2] <= 50:              chosen\_activities.append(activity[0])              max\_profit += activity[1]              current\_time += activity[2]              cumulative\_profit.append(max\_profit)          else:              break        return max\_profit, chosen\_activities, cumulative\_profit  # Input jumlah jenis aktivitas dan nilai Pi serta Di  def get\_input():      try:          n = int(input("Masukkan jumlah jenis aktivitas: "))          activities = {}            for x in range(1, n + 1):              p = int(input(f"Masukkan nilai Pi untuk jenis aktivitas {x}: "))              d = int(input(f"Masukkan nilai Di untuk jenis aktivitas {x}: "))              activities[x] = (p, d)            return n, activities        except ValueError:          print("Masukkan hanya angka untuk jumlah jenis aktivitas, Pi, dan Di.")          return None, None  # Contoh penggunaan:  n, activities = get\_input()  if n is not None and activities is not None:      max\_profit, chosen\_activities, cumulative\_profit = maximize\_profit(n, activities)      print(f"Urutan aktivitas untuk keuntungan maksimum: {chosen\_activities}")      print(f"Keuntungan maksimum yang dapat diperoleh: {max\_profit}")      # Plotting cumulative profit over time or activities      plt.figure(figsize=(10, 6))      plt.plot(np.arange(len(cumulative\_profit)) + 1, cumulative\_profit, marker='o', linestyle='-', color='b', label='Cumulative Profit')      plt.xlabel('Number of Activities')      plt.ylabel('Cumulative Profit')      plt.title('Cumulative Profit over Selected Activities')      plt.xticks(np.arange(len(cumulative\_profit)) + 1)      plt.grid(True)      plt.legend()      plt.tight\_layout()      plt.show() |

1. **Berikut Output dari ekskusi pemogramannya**

|  |
| --- |
|  |

**SOAL NO. 2**



**Jawaban No 2**

1. **ALGORITMA**
2. **Algoritma metoda forward (menghitung jarak ke depan)**
3. Hitung lintasan path dari suatu simpul ke tujuan

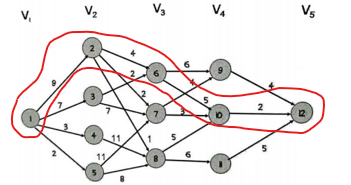
Rumus cost(I,j) = min {c(j,k)+cost(i+1,k)}

1. Perhitungan dimulai dari node k-2

**Perhitungan analitic:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| V4 | | | | |
| Cost (4,9) | = | c (9,12) | = | 4 |
| Cost (4,10) | = | c (10,12) | = | 2 {minimal} |
| Cost (4,11) | = | c (11,12) | = | 5 |
| V3 | | | | |
| Cost (3,6) | =  =  = | Min {c (6,9) +cost (4,9) ; c(6,10) + cost(4,10)}  Min { 6+4;5+2}  Min {10;7} | = | 7 |
| Cost (3,7) | =  =  = | Min {c (7,9) +cost (4,9) ; c(7,10) + cost(4,10)}  Min { 4+4;3+2}  Min {8;7} | = | 7 |
| Cost (3,8) | =  =  = | Min {c (8,10) +cost (4,10) ; c(8,11) + cost(4,11)}  Min { 5+2;6+5}  Min {7;11} | = | 7 |
| V2 |  |  |  |  |
| Cost (2,2) | =  =  = | Min {c(2,6) +cost (3,6) ; c(2,7) + cost(3,7);c(2,8)+cost(3,8) }  Min { 4+7;2+7;1+7}  Min {11;9,8} | = | 8 {min} |
| Cost (2,3) | =  =  = | Min {c (3,6) +cost (3,6) ; c(3,7) + cost(3,7) }  Min { 2+7;7+7}  Min {9;14} | = | 9 |
| Cost (2,4) | =  = | Min {c (4,8) +cost (3,8) }  Min { 11+7}  Min {18} | = | 18 |
| Cost (2,5) | =  = | Min {c (5,7) +cost (3,7) ; c(5,8) + cost(3,8) }  Min { 11+7;8+7}  Min {18;15} | = | 15 |
| V1 |  |  |  |  |
| Cost (1,1) | =  =  = | Min {c (1,2) +cost (2,2); c(1,3) + cost(2,3); c(1,4) + cost(2,4); c(1,5) + cost(2,5)}  Min { 9+8;7+9;3+18;2+15}  Min {17;16;21;17} | = | 16 |

**Jadi jalur terpendek nya yaitu : 1-2-6-10-12 dengan panjang lintasan 33**

****

1. **Algoritma Metoda Backward (Menghitung jarak kebelakang)**
2. Hitung lintasan path dari suatu simpul ke tujuan

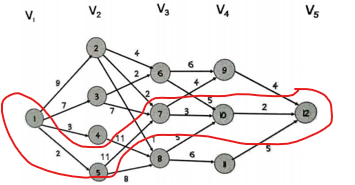
Rumus : bcost(I,j) = min {bc(i-1,1)+cost(1,j)}

1. Perhitungan dimulai dari node-node distage 3

**Perhitungan analitic**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| V2 | | | | |
| bcost b(2,2) | = | C(1,2) | = | 9 |
| bcost (2,3) | = | C(1,3) | = | 7 |
| bcost (2,4) | = | C(1,4) | = | 3 |
| bcost (2,5) | = | C(1,5) | = | 2 {min} |
| V3 |  |  |  |  |
| bCost (3,6) | =  =  = | Min {c (2,6) +bcost (2,2) ; c(2,7) + bcost(2,2);c(2,8)+bcost(2,2)}  Min { 4+9;2+9;1+9}  Min {13;11;20} | = | 11 |
| bCost (3,7) | =  = | Min {c (3,6) +bcost (2,3) ; c(3,7) + bcost(2,3) }  Min { 2+7;7+7}  Min {9;14} | = | 9 {min} |
| bCost (3,8) | =  = | Min {c (3,8) +bcost (2,4) ; c(3,8) + bcost(2,5) }  Min { 11+3;8+2}  Min {14;10} | = | 10 |
| V4 |  |  |  |  |
| bCost (4,9) | =  = | Min {c (9,6) +bcost (3,6) ; c(9,7) + bcost(3,7) }  Min { 6+11;4+9}  Min {17;13} | = | 13 {min} |
| bCost (4,10) | =  =  = | Min {c (10,7) +bcost (3,6) ; c(10,7) + bcost(3,7);c(10,8)+bcos(3,8) }  Min { 5+11;3+9;5+10}  Min {16;14;15} | = | 14 |
| bCost (4,11) | =  =  = | Min {c (11,8) +bcost (3,8) }  Min { 6+10}  Min {16} | = | 16 |
| V1 |  |  |  |  |
| bCost (1,12) | =  =  = | Min {c (12,9) +bcost (4,9); c(12,10) + bcost(4,10); c(12,11) + bcost(4,11) }  Min { 4+13;2+14;5+16}  Min {17;19;21} | = | 17 |

**Jadi jalur terpendek nya yaitu : 12 -10- 7-5-1 dengan panjang lintasan 16**

****

1. **Pemograman dengan Python**

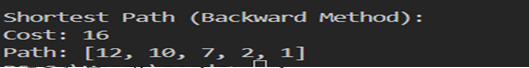
|  |
| --- |
| import networkx as nx  def forward\_multistage\_graph(graph, stages, start):      # Initialize the cost to reach each node and the paths      cost = {node: float('inf') for stage in stages for node in stage}      cost[start] = 0      path = {node: [] for stage in stages for node in stage}      path[start] = [start]        # Iterate over each stage      for stage in stages[:-1]:          for node in stage:              for neighbor, weight in graph[node]:                  if cost[node] + weight < cost[neighbor]:                      cost[neighbor] = cost[node] + weight                      path[neighbor] = path[node] + [neighbor]        # Extract the minimum cost to reach the final stage nodes and their paths      last\_stage = stages[-1]      min\_cost = float('inf')      best\_path = []      for node in last\_stage:          if cost[node] < min\_cost:              min\_cost = cost[node]              best\_path = path[node]        return min\_cost, best\_path  def backward\_multistage\_graph(graph, stages, start, end):      # Initialize the cost to reach each node and the paths      cost = {node: float('inf') for stage in stages for node in stage}      cost[end] = 0      path = {node: [] for stage in stages for node in stage}      path[end] = [end]        # Create a reversed graph with all nodes      reversed\_graph = {node: [] for node in graph}        # Populate the reversed graph with reversed edges      for node in graph:          for neighbor, weight in graph[node]:              reversed\_graph.setdefault(neighbor, []).append((node, weight))        # Reverse the stages      reversed\_stages = stages[::-1]        # Iterate over each stage (in reversed order)      for stage in reversed\_stages[:-1]:          for node in stage:              for neighbor, weight in reversed\_graph.get(node, []):                  if cost[node] + weight < cost[neighbor]:                      cost[neighbor] = cost[node] + weight                      path[neighbor] = path[node] + [neighbor]        # Extract the minimum cost to reach the initial stage nodes and their paths      first\_stage = reversed\_stages[-1]      min\_cost = float('inf')      best\_path = []      for node in first\_stage:          if cost[node] < min\_cost:              min\_cost = cost[node]              best\_path = path[node]        return min\_cost, best\_path  # Define the graph and stages (as per your example)  graph = {      1: [(2, 9), (3, 7), (4, 3), (5, 2)],      2: [(6, 4), (6, 2), (7,2),(7,7),(7,11),(8, 1),(8,11),(8,8)],      3: [(6, 2), (7, 7)],      4: [(8, 11)],      5: [(7, 11), (8, 8)],      6: [(9, 6), (10, 5)],      7: [(9, 4), (10, 3)],      8: [(10, 5), (11, 6)],      9: [(12, 4)],      10: [(12, 2)],      11: [(12, 5)],      12: []  # End node without outgoing edges  }  stages = [      [1],    # v1      [2, 3, 4, 5],  # v2      [6, 7, 8],  # v3      [9, 10, 11],  # v4      [12]  # v5  ]  # Calculate shortest path using forward method  start\_node = 1  min\_cost\_forward, best\_path\_forward = forward\_multistage\_graph(graph, stages, start\_node)  print("Shortest Path (Forward Method):")  print("Cost:", min\_cost\_forward)  print("Path:", best\_path\_forward)  # Calculate shortest path using backward method  end\_node = 12  min\_cost\_backward, best\_path\_backward = backward\_multistage\_graph(graph, stages, start\_node, end\_node)  print("\nShortest Path (Backward Method):")  print("Cost:", min\_cost\_backward)  print("Path:", best\_path\_backward) |

1. **Output Pemograman**

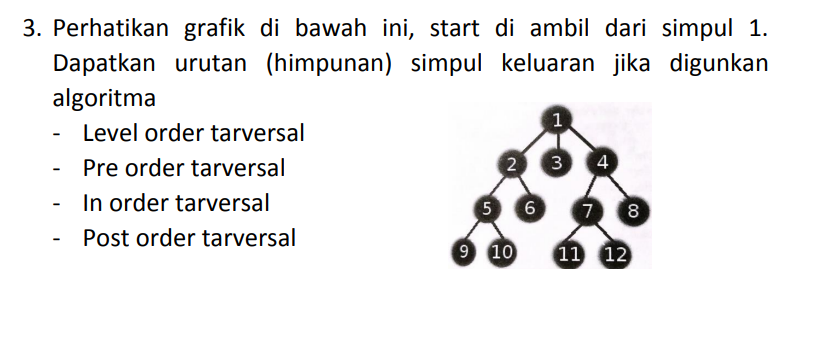
* Forward methode

|  |
| --- |
|  |

* Backward method



**SOAL NO 3**



**Jawaban No 3**

1. **ALGORITMA DAN ANALITIC**
2. **LEVEL ORDER TRAVERSAL**
3. **Algoritma Level Order Traversal**
4. Nyatakan dua daftar kosong : **open list** dan **closed list**
5. Tambahkan simpul awal pada **open list**
6. Selama **open list** tidak kosong maka :
7. Keluarkan simpul pertama dari open list
8. Cek apakah yang dikeluarkan simpul tujuan atau bukan?
9. Jika ya, stop pencarian dan keluar dan tambahkan simpul pada closed list dan kembalikan nilai closed list,
10. jika bukan simpul tujuan kerjakan Langkah c
11. eksplorasi simpul dilevel lebih tinggi dari simpul yang dikeluarkan
12. tambahkan disekeliling simpul akhir dari **open list,** dan tambahkan simpul yang dikeluarkan pada **closed list**
13. **Level Order Traversal sebagai berikut**
14. Langkah 1

* Open list : 1
* Closed list :<empty>

1. Langkah 2

* Open list : 2,3,4
* Closed list : 1

1. Langkah 3

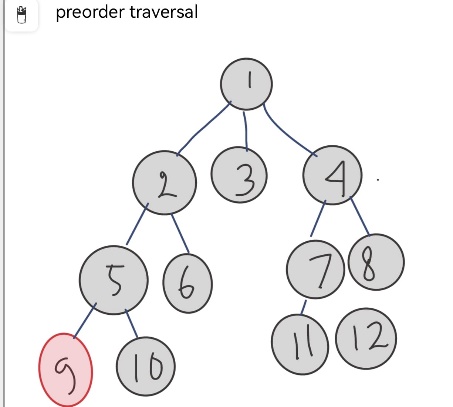
* Open list 5,6,7,8
* Closed list : 1,2,3,4

1. Langkah 4

* Open list 9,10,11,12
* Closed list : 1,2,3,4,5,6,7,8

1. Langkah 5

Simpul 9 dieksplorasi dan ternyata **simpul tujuan**, maka pelacakan berhenti sampai disini dan diperoleh

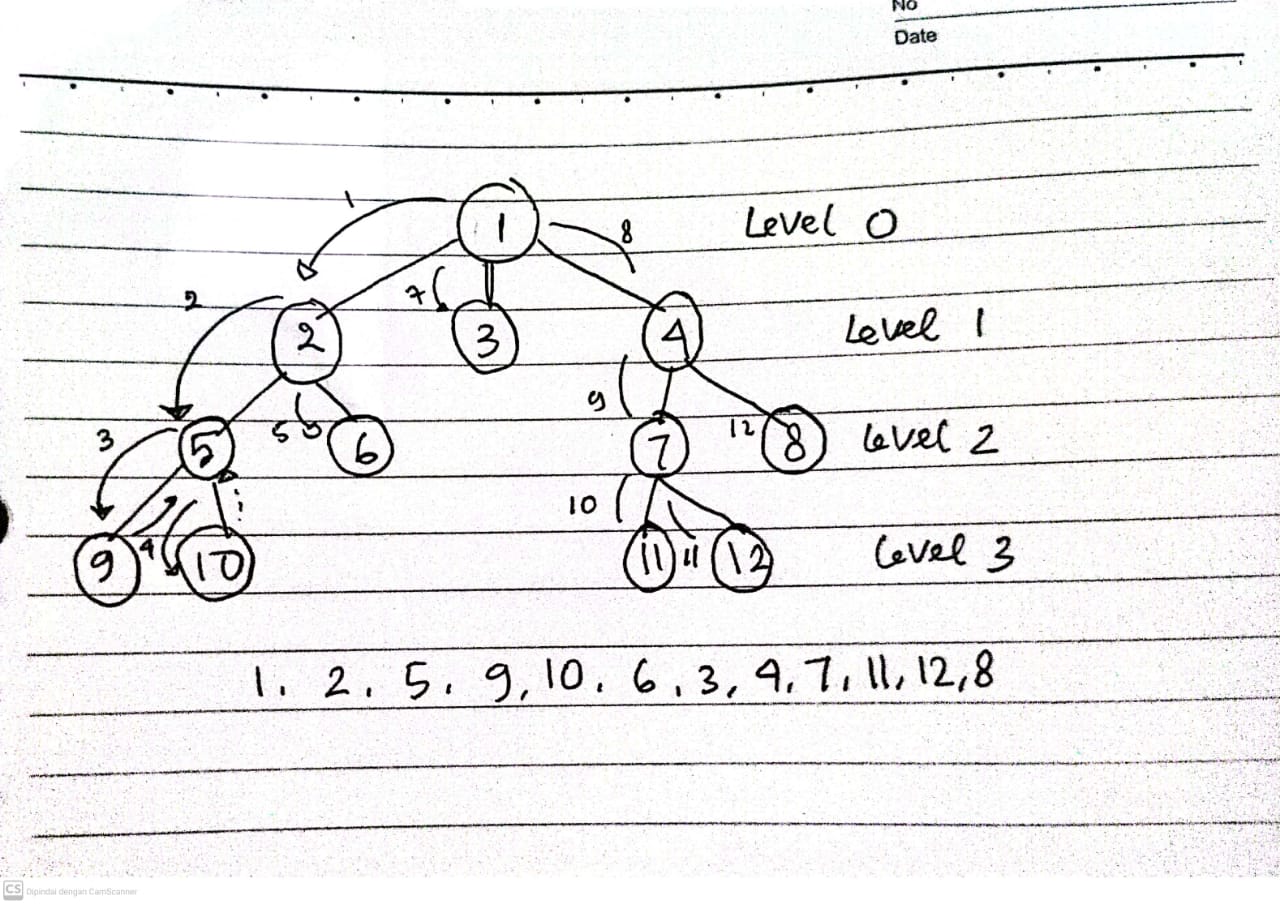


Dengan metode order traversal lintasan pelacakan yang dilalui berada pada daftar close list yaitu : **1,2,3,4,5,6,7,8, 9**

1. **PREORDER TRAVERSAL**
2. **Algoritma Level Order Traversal**
3. Nyatakan dua daftar kosong : **open list** dan **closed list**
4. Tambahkan simpul awal pada **open list**
5. Selama **open list** tidak kosong maka :
6. Keluarkan simpul pertama dari open list
7. Cek apakah yang dikeluarkan simpul tujuan atau bukan?

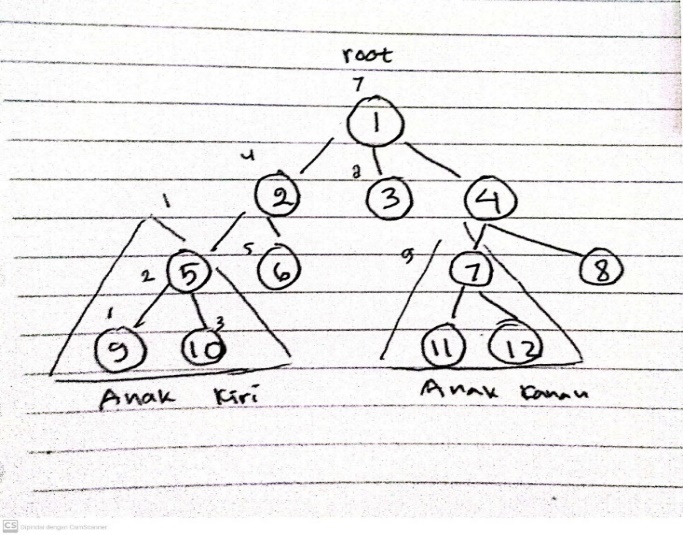
* Jika ya, stop pencarian dan keluar dan tambahkan simpul pada closed list dan kembalikan nilai closed list,
* Jika bukan simpul tujuan kerjakan Langkah c

1. eksplorasi simpul dilevel lebih tinggi dari simpul yang dikeluarkan
2. tambahkan disekeliling simpul akhir dari **open list,** dan tambahkan simpul yang dikeluarkan pada **closed list**
3. **Preorder traversal sebagai berikut**



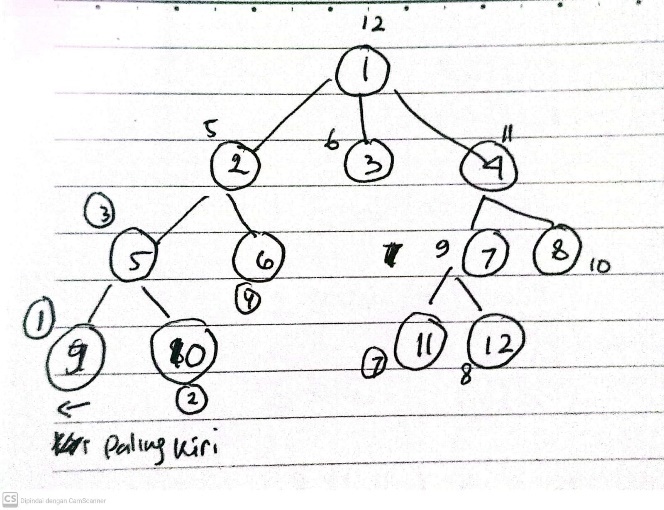
**Jadi : 1, 2, 5, 9, 10, 6, 3, 4, 7, 11, 12, 8**

1. **IN ORDER TRAVERSAL**
2. **Algoritma In Order Traveersa**
3. Eksplorasi simpul anak kiri
4. Eksplorasi simpul induk
5. Eksplorasi simpul kanan
6. **Berikut analitic in order traversaal**



**Jadi : 9, 5, 10, 2, 6, 1, 3**

1. **POST ORDER TRAVERSAL**
2. **Algoritma Post Order Traversal**
3. Lintasi semua eksternal kiti dimulai paling kiri dan ikuti simpul internal diatasnya
4. Lintasi semua simpul kanan mulai dari simpul paling kiri dan ikuti simpul internal diatasnya



**Jadi : [9, 10, 5, 6, 2, 3, 11, 12, 7, 8, 4, 1]**

1. **Pemograman dengan Python**

|  |
| --- |
| import networkx as nx  import matplotlib.pyplot as plt  # Definisi graf menggunakan dictionary  graph = {      1: [2, 3, 4],      2: [5, 6],      5: [9, 10],      4: [7, 8],      7: [11, 12],      # Nodes with no outgoing edges are omitted since they won't affect traversal.  }  # Membuat objek graf  G = nx.DiGraph(graph)  # Mengatur posisi node secara manual untuk visualisasi yang lebih rapi  pos = {      1: (0, 0),      2: (-1, -1),      3: (0, -1),      4: (1, -1),      5: (-2, -2),      6: (-1, -2),      7: (1, -2),      8: (2, -2),      9: (-3, -3),      10: (-2, -3),      11: (1, -3),      12: (2, -3),  }  # Inisialisasi level order traversal menggunakan BFS  def level\_order\_traversal(start):      if start not in graph:          return []        queue = [start]      result = []      visited = set(queue)        while queue:          node = queue.pop(0)          result.append(node)          for neighbor in graph.get(node, []):              if neighbor not in visited:                  queue.append(neighbor)                  visited.add(neighbor)        return result  # Pre order traversal  def pre\_order\_traversal(node):      if node is None:          return []        result = []      stack = [node]        while stack:          current = stack.pop()          result.append(current)          children = graph.get(current, [])          stack.extend(reversed(children))  # Reverse to maintain left-to-right order        return result  # In order traversal  def in\_order\_traversal(node):      if node is None:          return []        result = []      stack = []      current = node        while stack or current:          while current:              stack.append(current)              current = graph.get(current, [])[0] if graph.get(current) else None  # Go leftmost            current = stack.pop()          result.append(current)          current = graph.get(current, [])[1] if graph.get(current) and len(graph.get(current)) > 1 else None  # Go right        return result  # Post order traversal  def post\_order\_traversal(node):      if node is None:          return []        result = []      stack = [node]        while stack:          current = stack.pop()          result.append(current)          children = graph.get(current, [])          stack.extend(children)  # Extend with children to process them next        return result[::-1]  # Reverse the result to get post-order  # Calculate and print traversals  print("Level Order Traversal:", level\_order\_traversal(1))  print("Pre Order Traversal:", pre\_order\_traversal(1))  print("In Order Traversal:", in\_order\_traversal(1))  print("Post Order Traversal:", post\_order\_traversal(1))  # Draw the graph for visualization  plt.figure(figsize=(8, 8))  nx.draw(G, pos, with\_labels=True, node\_size=2000, node\_color='lightblue', font\_size=12, font\_weight='bold', arrows=True)  plt.title('Traversal Graph')  plt.show() |

1. **Output Pemograman Python**

|  |
| --- |
|  |

**Soal No 4**

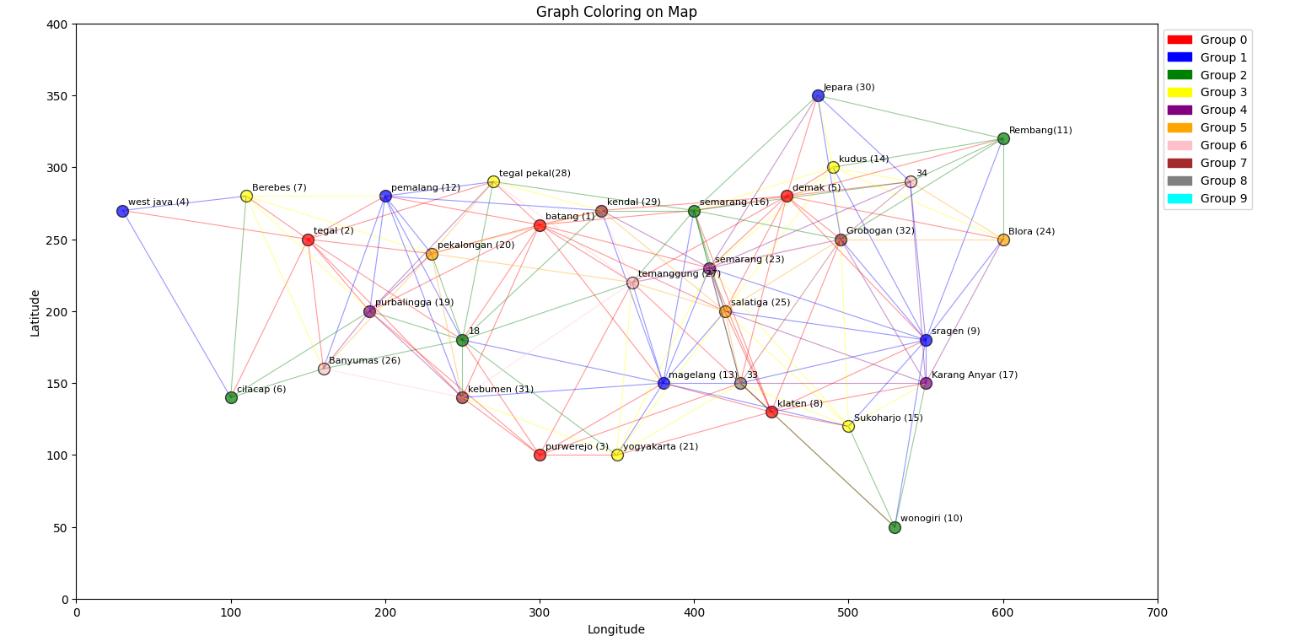


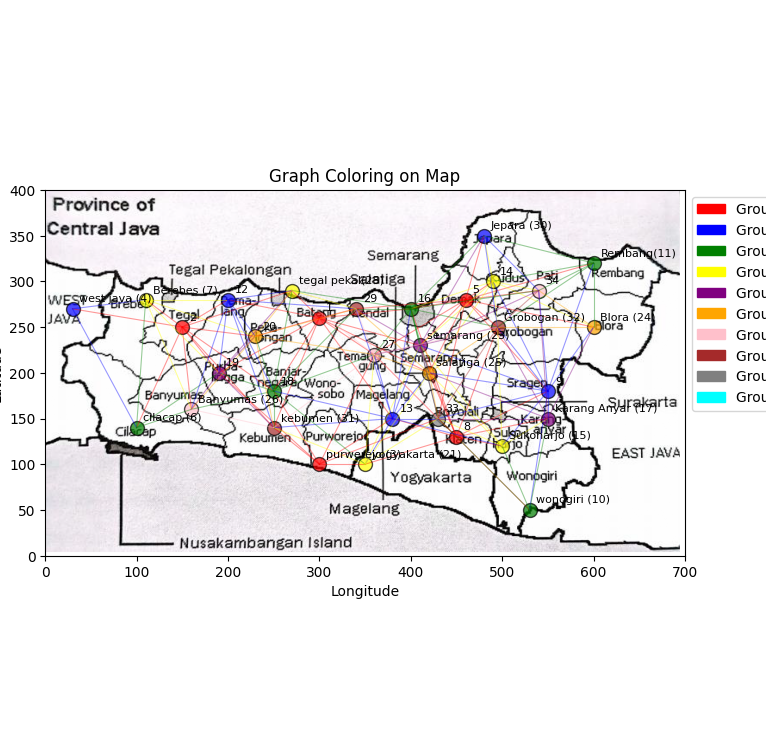
**Jawaban No 4**

1. **Algoritma Graph Coloring**
2. Tentukan jumlah warna
3. Eksplorasi simpul pertama dan beri warna pertama yang ditentukan
4. Eksplorasi simpul kedua yang berdekatan dan beri warna kedua
5. Kerjakan untuk simpul-simpul sisanya, dan beri warna
   * Jika warna sama dengan warna simpul berdekatan ditolak
   * Jika warna berbeda diterima
6. Ulangi Langkah 3 sampai semua simpul diberi warna
7. **Pemograman dengan Python**

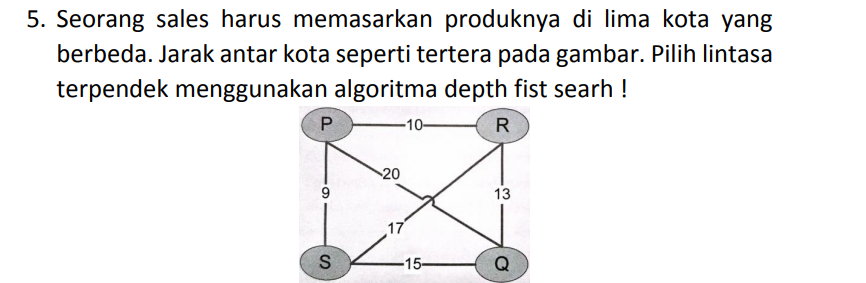
|  |
| --- |
| import networkx as nx  import matplotlib.pyplot as plt  import matplotlib.patches as mpatches  import math  # Definisikan wilayah dan atributnya dengan variabel  wilayah\_1 = 'batang (1)'  wilayah\_2 = 'tegal (2)'  wilayah\_3 = 'purwerejo (3)'  wilayah\_4 = 'west java (4)'  wilayah\_5 = 'demak (5)'  wilayah\_6 = 'cilacap (6)'  wilayah\_7 = 'Berebes (7)'  wilayah\_8 = 'klaten (8)'  wilayah\_9 = 'sragen (9)'  wilayah\_10 = 'wonogiri (10)'  wilayah\_11 = 'Rembang(11)'  wilayah\_12 = 'pemalang (12)'  wilayah\_13 = 'magelang (13)'  wilayah\_14 = 'kudus (14)'  wilayah\_15 = 'Sukoharjo (15)'  wilayah\_16 = 'semarang (16)'  wilayah\_17 = 'Karang Anyar (17)'  wilayah\_18 = '18'  wilayah\_19 = 'purbalingga (19)'  wilayah\_20 = 'pekalongan (20)'  wilayah\_21 = 'yogyakarta (21)'  # wilayah\_22 = '22'  wilayah\_23 = 'semarang (23)'  wilayah\_24 = 'Blora (24)'  wilayah\_25 = 'salatiga (25)'  wilayah\_26 = 'Banyumas (26)'  wilayah\_27 = 'temanggung (27)'  wilayah\_28 = 'tegal pekal(28)'  wilayah\_29 = 'kendal (29)'  wilayah\_30 = 'Jepara (30)'  wilayah\_31 = 'kebumen (31)'  wilayah\_32 = 'Grobogan (32)'  wilayah\_33 = '33'  wilayah\_34 = '34'  # wilayah\_35 = '35'  # Definisikan semua wilayah sebagai daftar  wilayahs = [      wilayah\_1, wilayah\_2, wilayah\_3, wilayah\_4,wilayah\_5, wilayah\_6, wilayah\_7, wilayah\_8, wilayah\_9, wilayah\_10,      wilayah\_11, wilayah\_12, wilayah\_13, wilayah\_14, wilayah\_15, wilayah\_16, wilayah\_17, wilayah\_18, wilayah\_19, wilayah\_20,      wilayah\_21,    wilayah\_23, wilayah\_24, wilayah\_25, wilayah\_26, wilayah\_27, wilayah\_28, wilayah\_29, wilayah\_30,      wilayah\_31, wilayah\_32, wilayah\_33, wilayah\_34 #, wilayah\_35  #   wilayah\_22,  ]  # Bangun grafik menggunakan networkx  G = nx.Graph()  # Tambahkan semua wilayah sebagai node  G.add\_nodes\_from(wilayahs)  # Tentukan posisi node (koordinat untuk plotting)  positions = {      wilayah\_1: (300, 260), wilayah\_2: (150, 250), wilayah\_3: (300, 100),    wilayah\_4: (30, 270),      wilayah\_5: (460, 280), wilayah\_6: (100, 140), wilayah\_7: (110, 280), wilayah\_8: (450, 130),      wilayah\_9: (550, 180), wilayah\_10: (530, 50), wilayah\_11: (600, 320), wilayah\_12: (200, 280),      wilayah\_13: (380, 150), wilayah\_14: (490, 300), wilayah\_15: (500, 120), wilayah\_16: (400, 270),      wilayah\_17: (550, 150), wilayah\_18: (250, 180), wilayah\_19: (190, 200), wilayah\_20: (230, 240),      wilayah\_21: (350, 100),    #  wilayah\_22: (660, 110),      wilayah\_23: (410, 230), wilayah\_24: (600, 250),    wilayah\_25: (420, 200), wilayah\_26: (160, 160), wilayah\_27: (360, 220), wilayah\_28: (270, 290),      wilayah\_29: (340, 270), wilayah\_30: (480, 350), wilayah\_31: (250, 140), wilayah\_32: (495, 250),      wilayah\_33: (430, 150), wilayah\_34: (540, 290)#, wilayah\_35: (140, 130)  }  # Fungsi untuk menghitung jarak Euclidean antara dua titik  def euclidean\_distance(pos1, pos2):      return math.sqrt((pos1[0] - pos2[0])\*\*2 + (pos1[1] - pos2[1])\*\*2)  # Tentukan ambang batas jarak (misalnya 150)  distance\_threshold = 150  # Tambahkan edge antara node yang jaraknya kurang dari ambang batas  for node1 in wilayahs:      for node2 in wilayahs:          if node1 != node2 and euclidean\_distance(positions[node1], positions[node2]) < distance\_threshold:              G.add\_edge(node1, node2)  # Algoritma pewarnaan graf greedy  def greedy\_graph\_coloring(graph):      color\_map = {}      for node in graph.nodes():          available\_colors = set(range(len(graph.nodes())))          for neighbor in graph.neighbors(node):              if neighbor in color\_map:                  available\_colors.discard(color\_map[neighbor])          color\_map[node] = min(available\_colors)      return color\_map  # Panggil fungsi pewarnaan graf  color\_map = greedy\_graph\_coloring(G)  # Tentukan warna untuk setiap kelompok berdasarkan hasil pewarnaan graf  color\_palette = [      'red', 'blue', 'green', 'yellow', 'purple', 'orange', 'pink', 'brown', 'gray', 'cyan'  ]  # Fungsi untuk plot grafik pada peta  def plot\_graph\_on\_map(positions, color\_map):      # Initialize figure and axis      fig, ax = plt.subplots(figsize=(12, 10))      # Tambahkan gambar peta di latar belakang plot      # map\_img = plt.imread("peta.png")      # ax.imshow(map\_img, extent=[0, 700, 0, 400])      # Gambar node (wilayah) dengan warna berdasarkan hasil pewarnaan graf      for node, pos in positions.items():          color = color\_palette[color\_map[node] % len(color\_palette)]          ax.scatter(pos[0], pos[1], s=100, edgecolors='k', facecolors=color, alpha=0.7)          # Tambahkan nama wilayah di posisi node          ax.annotate(node, xy=pos, xytext=(5, 5), textcoords='offset points', fontsize=8, color='black')      # Gambar edge      for edge in G.edges():          color = color\_palette[color\_map[edge[0]] % len(color\_palette)]          ax.plot([positions[edge[0]][0], positions[edge[1]][0]], [positions[edge[0]][1], positions[edge[1]][1]], '-', color=color, alpha=0.4, linewidth=0.8)      # Buat legenda untuk kelompok warna      legend\_patches = []      for idx, color in enumerate(color\_palette):          legend\_patches.append(mpatches.Patch(color=color, label=f'Group {idx}'))      # Tambahkan legenda ke plot      ax.legend(handles=legend\_patches, loc='upper left', bbox\_to\_anchor=(1, 1))      # Atur batas dan label sumbu      ax.set\_xlim(0, 700)      ax.set\_ylim(0, 400)      ax.set\_xlabel('Longitude')      ax.set\_ylabel('Latitude')      ax.set\_title('Graph Coloring on Map')      # Tampilkan grafik      plt.tight\_layout()      plt.show()  # Panggil fungsi untuk menampilkan grafik pada peta  plot\_graph\_on\_map(positions, color\_map) |

1. **Output pemograman Pyhthon**
2. **Graph Coloring Tanpa peta**



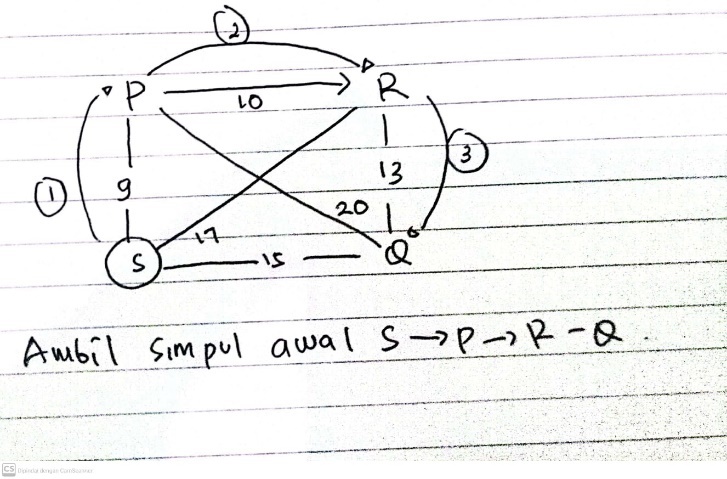
1. **Graph Coloring Dengan peta**

**SOAL NO 5**

****

**Jawaban No. 5**

1. **Algoritma Depth First Search**
2. Mulai dari Node Akar atau Node Awal
3. Tandai node awal sebagai dikunjungi.
4. Jelajahi Tetangga yang Belum Dikunjungi
5. Ulangi langkah 2 dan 3 untuk node tetangga.
6. Jika semua tetangga dari node saat ini telah dikunjungi, kembali ke node sebelumnya (backtrack) dan jelajahi tetangga lainnya yang belum dikunjungi
7. **Analitic nya**



**Jadi lintasan terpendek : S-P-R-Q**

1. **Pemograman Menggunakan Python**

|  |
| --- |
| def dfs(graph, node, visited, path):      # Tandai node sebagai dikunjungi      visited[node] = True      path.append(node)      # Telusuri semua tetangga node yang belum dikunjungi      for next\_node in graph[node]:          if not visited[next\_node]:              dfs(graph, next\_node, visited, path)      return path  # Membangun graf dari gambar  graph = {      "S": ["P", "Q"],      "P": ["R"],      "Q": [],      "R": []  }  # Inisialisasi variabel  start = "S"  visited = {node: False for node in graph}  # Semua node diinisialisasi sebagai belum dikunjungi  path = []  # Menjalankan DFS  path = dfs(graph, start, visited, path)  # Menampilkan hasil  print("Hasil DFS:", path) |

1. **Output pemograman dengan Python**

|  |
| --- |
|  |

Alhamdulilah, Terimakasih