DAA Project - Perencanaan Liburan Optimal

Nama: Victor

NRP: c14230245

Nama: Cornelius Jonathan E

NRP: c14230256

1. Dekomposisi A: Memetakan lokasi destinasi wisata dan atributnya

Algoritma yang digunakan: Brute-Force

Alasan: Pada tahap awal, diperlukan eksplorasi menyeluruh untuk memetakan biaya perjalanan, value pengalaman,

dan durasi waktu dari setiap destinasi. Brute-Force dipilih karena sederhana dan cocok untuk memproses data kecil.

Pseudocode:

```
    destinations = []  # List destinasi
    for i in range(N):
        cost = input("Biaya perjalanan ke destinasi {}: ".format(i))
        value = input("Value pengalaman destinasi {}: ".format(i))
        time = input("Durasi kunjungan ke destinasi {}: ".format(i))
        destinations.append((cost, value, time))
    return destinations
```

Time Complexity: O(N)

2. Dekomposisi B: Memutuskan rute berdasarkan value terbesar dengan biaya minimum

Algoritma yang digunakan: Dynamic Programming

Alasan: Dynamic Programming cocok untuk masalah optimasi seperti Knapsack Problem. Kita memaksimalkan value destinasi dengan batasan biaya dan waktu.

Pseudocode:

```
def maximize_value(destinations, budget, max_time):
    dp = [[0] * (budget + 1) for _ in range(max_time + 1)]
    for cost, value, time in destinations:
        for t in range(max_time, time - 1, -1):
            for b in range(budget, cost - 1, -1):
                dp[t][b] = max(dp[t][b], dp[t - time][b - cost] + value)
    return dp[max_time][budget]
```

Time Complexity: O(N × budget × max_time)

3. Dekomposisi C: Perhitungan seluruh biaya, value, dan waktu secara keseluruhan

Algoritma yang digunakan: Divide and Conquer

Alasan: Divide and Conquer memecah data destinasi ke subset kecil, menghitung total biaya, value, dan waktu

untuk tiap subset, kemudian menggabungkan hasilnya. Hal ini membuat perhitungan lebih efisien.

Pseudocode:

```
def calculate_total(destinations, start, end):
    if start == end:
        return destinations[start]
    mid = (start + end) // 2
    left = calculate_total(destinations, start, mid)
```

```
    right = calculate_total(destinations, mid + 1, end)
    return (left[0] + right[0], left[1] + right[1], left[2] + right[2])  # Total biaya, value, waktu
```

Time Complexity: O(N log N)

4. Dekomposisi D: Menentukan rute perjalanan optimal di setiap langkah

Algoritma yang digunakan: Greedy

Alasan: Greedy memilih destinasi dengan rasio value/cost terbaik pada setiap langkah hingga budget atau waktu habis.

Pendekatan ini cepat dan cocok untuk langkah optimasi lokal.

Pseudocode:

```
def greedy_route(destinations, budget, max_time):
    destinations.sort(key=lambda x: x[1] / x[0], reverse=True)  # Sort by value/cost ratio
    total_value, total_cost, total_time = 0, 0, 0
    for cost, value, time in destinations:
        if total_cost + cost <= budget and total_time + time <= max_time:
            total_value += value
            total_cost += cost
            total_time += time
    return total_value, total_cost, total_time
```

Time Complexity: O(N log N)

5. Dekomposisi E: Efisiensi perhitungan untuk wilayah yang luas

Algoritma yang digunakan: Graph (Dijkstra's Algorithm)

Alasan: Representasi graph membantu menghitung rute terpendek antar destinasi di Jawa Timur

dengan jarak sebagai bobot.

Dijkstra's Algorithm dipilih karena efisien untuk graph dengan bobot non-negatif.

Pseudocode:

```
def dijkstra(graph, start):
    import heapq
    distance = {node: float('inf') for node in graph}
    distance[start] = 0
    pq = [(0, start)]
    while pq:
        curr_dist, curr_node = heapq.heappop(pq)
        if curr_dist > distance[curr_node]:
            continue
        for neighbor, weight in graph[curr_node]:
            distance_neighbor = curr_dist + weight
            if distance_neighbor < distance[neighbor]:
                distance[neighbor] = distance_neighbor
                heapq.heappush(pq, (distance_neighbor, neighbor))
    return distance
```

Time Complexity: O((V + E) log V)

Ringkasan

Setiap dekomposisi diselesaikan dengan algoritma yang sesuai:

1. Brute-Force (O(N)) untuk eksplorasi awal.

2. Dynamic Programming (O(N × budget × max_time)) untuk optimasi value.

3. Divide and Conquer (O(N log N)) untuk efisiensi perhitungan total.

4. Greedy (O(N log N)) untuk keputusan lokal optimal.

5. Graph/Dijkstra (O((V + E) log V)) untuk rute terpendek.

Pendekatan ini mencakup eksplorasi awal hingga optimasi akhir, memastikan efisiensi tanpa kehilangan akurasi.