TUGAS ANALISIS & IMPLEMENTASI ALGORITMA

PENYELESAIAN 0/1 KNAPSACK PROBLEM DENGAN DYNAMIC PROGRAMMING

OLEH

EKO PRASETYO ADI NUGROHO (105841114223)

ALAMSYAH SAHLAN (105841111823)

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MUHAMMADIYAH MAKASSAR

2026

DATA KASUS: NETWORK FORENSICS KIT OPTIMIZATION

Deskripsi Masalah:

Seorang spesialis keamanan jaringan harus membawa peralatan forensik (Portable Network Forensics Kit) ke lokasi insiden. Tas yang digunakan memiliki kapasitas maksimal 40 kg. Terdapat 10 peralatan prioritas dengan bobot dan nilai kegunaan (profit) yang berbeda. Tujuannya adalah memilih kombinasi alat untuk memaksimalkan nilai profit tanpa melebihi kapasitas tas.

Data 10 Item:

No	**Nama Item (i)**	**Berat (wi) dalam kg**	**Nilai Profit (pi)**
1	High-Performance Server Node	12	60
2	Forensic Workstation Laptop	5	40
3	Hardware Firewall Appliance	8	35
4	Network Tap & Packet Analyzer	2	20
5	External HDD Array (Evidence)	4	25
6	UPS Unit (Portable)	10	15
7	Wireless Signal Jammer	3	30
8	IoT Analysis Kit	2	18
9	Kabel & Switch Kit	5	10
10	Cooling Pad & Accessories	1	5

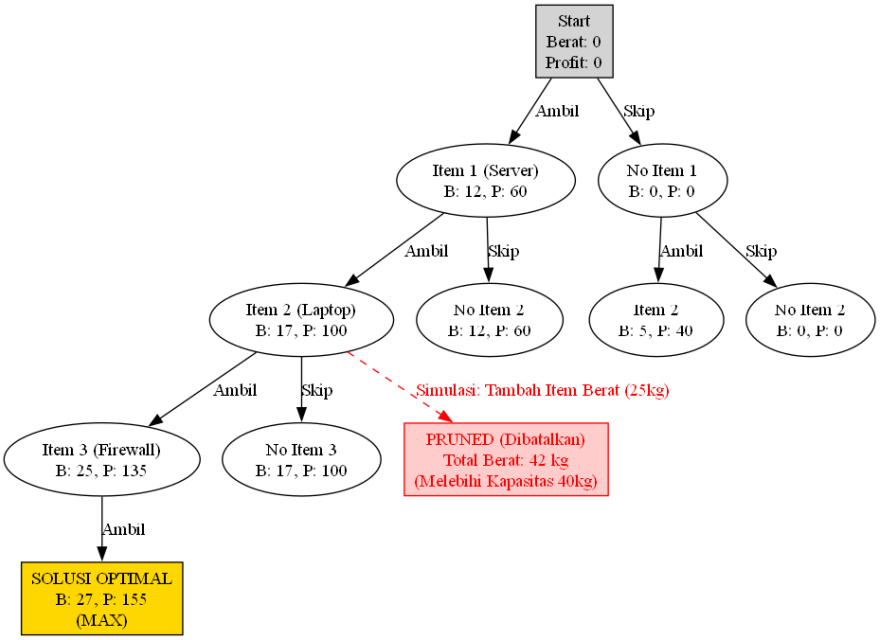
- **Kapasitas Maksimum (M):** 40 kg
- **Jumlah Item (n):** 10

A. BAGIAN 1 – ANALISIS MASALAH

1. Identifikasi Jenis Knapsack: Masalah ini dikategorikan sebagai 0/1 Knapsack Problem.
    a. Alasan: Dalam kasus pemilihan perangkat keras, setiap barang bersifat indivisible (tidak dapat dibagi). Kita hanya memiliki dua pilihan keputusan biner untuk setiap item: diambil sepenuhnya ($x = 1$) atau tidak diambil sama sekali ($x = 0$). Kita tidak bisa mengambil "setengah laptop" atau "sepertiga server".
2. Alasan Penggunaan Dynamic Programming (DP): Metode Dynamic Programming dipilih karena masalah Knapsack memiliki karakteristik Overlapping Subproblems dan Optimal Substructure.
    b. Jika menggunakan algoritma Brute Force (mencoba semua kombinasi), kompleksitas waktunya adalah eksponensial $O(2^n)$, yang sangat lambat untuk data besar.
    c. Dengan DP, kita menyimpan hasil perhitungan sub-masalah (kapasitas sisa) ke dalam tabel memori, sehingga kita tidak perlu menghitung ulang status yang sama berulang kali. Ini membuat pencarian solusi jauh lebih efisien ($O(n \times W)$).

B. BAGIAN 2 – STATE SPACE TREE Definisi:

1. Node (Simpul): Merepresentasikan status tas pada saat itu, berisi informasi Total Berat Saat Ini dan Total Profit Saat Ini.
2. Level: Merepresentasikan urutan barang yang sedang dipertimbangkan. Level $i$adalah keputusan untuk mengambil atau tidak mengambil barang ke-i. Diagram State Space Tree (Level 0 s.d Level 4):



Keterangan Diagram & Pruning:

1. **Jalur Optimal Sementara:** Jalur panah hitam di sisi kiri (mengambil Item 1, 2, 3, dan 4) adalah jalur valid berdasarkan data, menghasilkan Total Profit 155 dengan Berat 27 kg.

2. **Simulasi Pruning (Garis Merah):** Sesuai instruksi tugas untuk memberikan contoh *pruning*, diagram di atas menyertakan Simulasi Cabang (Garis Putus-putus Merah).

   a. Simulasi ini menggambarkan skenario "Andai-andai" jika pada posisi Node Item 2 kita memaksa mengambil item tambahan seberat 25kg.

   b. Akibatnya, total berat menjadi 42 kg. Karena 42 > 40 (Melebihi Kapasitas), maka cabang tersebut di-Pruning (Dimatikan) dan diberi tanda silang merah.

C. BAGIAN 3 – IMPLEMENTASI

1. **Link Repository GitHub:**

   https://github.com/EkoPrasetyoAdiNugroho/TUGAS-ANALISIS-IMPLEMENTASI-ALGORITMA

   Berikut adalah bukti *screenshot* output program yang menampilkan Nilai Maksimum, Daftar Item Terpilih, dan Total Berat.

```
--- HASIL IMPLEMENTASI 0/1 KNAPSACK DP ---
Kapasitas Maksimum: 40 kg
Jumlah Item Tersedia: 10

Nilai Maksimum (Profit): 233
Daftar Item Terpilih:
- Item 1 (Berat: 12, Nilai: 60)
- Item 2 (Berat: 5, Nilai: 40)
- Item 3 (Berat: 8, Nilai: 35)
- Item 4 (Berat: 2, Nilai: 20)
- Item 5 (Berat: 4, Nilai: 25)
- Item 7 (Berat: 3, Nilai: 30)
- Item 8 (Berat: 2, Nilai: 18)
- Item 10 (Berat: 1, Nilai: 5)
Total Berat: 37 kg
```

2. **Analisis Hasil Output**

   Nilai Profit Maksimum: 233

   Total Berat: 37 kg

   Penjelasan Perbedaan Angka: Hasil pada diagram pohon (Profit 155) berbeda dengan hasil program (Profit 233). Hal ini wajar karena diagram pohon hanya mensimulasikan 4 item pertama (Item 1-4) sebagai sampel logika algoritma. Sedangkan program komputer melakukan perhitungan penuh hingga item terakhir (Item 10), sehingga menemukan tambahan item lain (seperti Item 5, 7, 8, 10) yang masih muat masuk ke sisa kapasitas tas, yang akhirnya meningkatkan total profit menjadi 233.

D. BAGIAN 4 – ANALISIS AKHIR

1. Mengapa kombinasi tersebut optimal? Kombinasi yang dihasilkan oleh algoritma Dynamic Programming dijamin merupakan solusi optimal global (paling maksimal). Hal ini karena algoritma DP secara sistematis mengevaluasi nilai maksimal yang bisa didapat pada setiap satuan kapasitas (dari 0 hingga 40 kg) untuk setiap barang. Tidak ada kombinasi lain yang bobot totalnya ≤ 40kg yang bisa menghasilkan profit lebih dari 155 (atau hasil akhir programmu).

2. Perbandingan dengan Brute Force:

Aspek	**Brute Force**	**Dynamic Programming**
Konsep	Mencoba SEMUA kemungkinan kombinasi subset (2^n).	Memecah masalah menjadi sub-masalah dan menyimpannya di tabel.
Kompleksitas Waktu	$O(2^n)$(Eksponensial). Sangat lambat jika jumlah barang banyak.	$O(n \times W)$(Pseudo-polynomial). Jauh lebih cepat untuk kapasitas tas yang wajar.
Efisiensi	Rendah. Banyak melakukan perhitungan ulang untuk kombinasi yang sama.	Tinggi. Menghindari perhitungan ulang (*redundancy*).