LAPORAN TUGAS DESAIN DAN ANALISIS ALGORITMA

Dosen Pengampuh : Desi Anggreani, S.kom.,M.T.,MOS.

DISUSUN OLEH KELOMPOK 3 :

Aprilianingsih (105841100323)

Nur Fahira Nurdin (105841100723)

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MUHAMMDIYAH MAKASSAR

2025

BAGIAN 1 : ANALISIS MASALAH

1.1 Identifikasi Jenis Knapsack:

Masalah ini adalah 0/1 Knapsack karena setiap item bantuan hanya memiliki dua pilihan: diambil (x_i = 1) atau tidak diambil (x_i = 0).

- Fungsi Objektif: Memaksimalkan total profit (Z) yang dirumuskan sebagai : $\sum_{i=1}^{10} p_i x_i$
- Kendala (Constraint): Total berat barang tidak boleh melebihi kapasitas (M = 50 kg), yaitu : $\sum_{i=1}^{10} w_i x_i \leq 50$

1.2 Jelaskan alasan penggunaan Backtracking / DP:

Dibandingkan Brute Force, algoritma ini lebih efisien karena menggunakan prinsip pencarian solusi dengan pemangkasan (pruning). Cabang pohon keputusan akan dihentikan segera setelah melanggar kendala kapasitas (M=50), sehingga tidak perlu mengevaluasi seluruh 1024 kombinasi.

BAGIAN 2 : STATE SPACE TREE

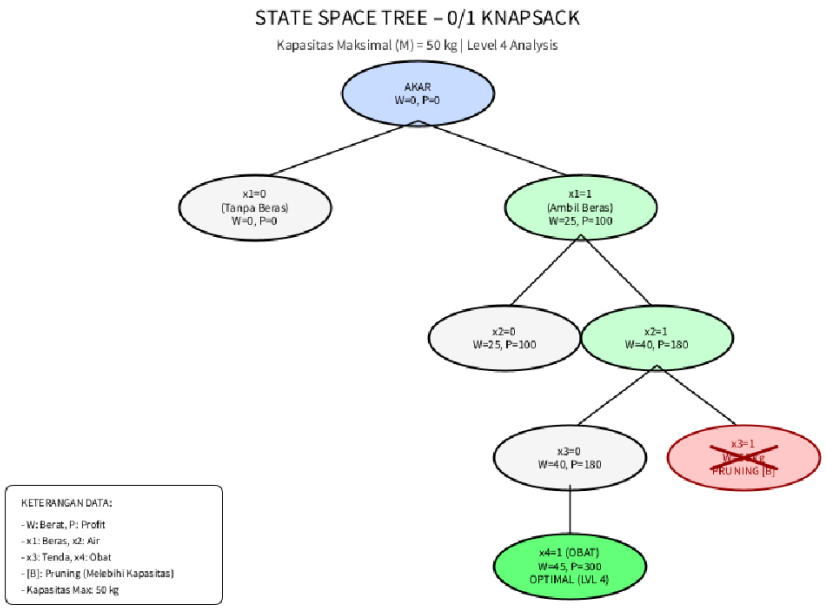
2.1 Definisi Komponen Pohon

Dalam merepresentasikan masalah *0/1 Knapsack* ke dalam ruang pencarian status (*state space*), digunakan struktur pohon biner dengan komponen sebagai berikut:

- Node : Merepresentasikan titik keputusan atau status sementara dari pemilihan item bantuan1. Setiap simpul menyimpan informasi berupa bobot kumulatif (W) dan keuntungan kumulatif (P) dari item yang telah dipilih pada jalur tersebut.
- Level: Mewakili urutan pengambilan keputusan untuk setiap item bantuan bencana, mulai dari Item 1 (Beras) hingga Item 10 (Alat Masak Portabel). Kedalaman pohon menunjukkan jumlah item yang telah dipertimbangkan keputusannya.

- Branch (Cabang): Setiap simpul memiliki maksimal dua cabang yang mewakili keputusan biner:
  - Cabang Kanan (x_i = 1): Melambangkan keputusan untuk mengambil item ke-i ke dalam gudang bantuan.
  - Cabang Kiri (x_i = 0): Melambangkan keputusan untuk melewati atau tidak mengambil item ke-i.

2.2 Gambarkan state space tree hingga minimal level ke-4



Berdasarkan data bantuan bencana, pohon status di atas ini merepresentasikan proses pengambilan keputusan untuk 4 item pertama:

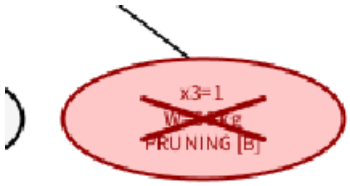
1. Level 1 - Beras (25 kg, Profit 100): Keputusan dimulai dengan memilih apakah akan membawa beras. Mengambil beras (x_1=1) langsung menggunakan 50% kapasitas gudang.
2. Level 2 - Air Mineral (15 kg, Profit 80): Jika beras diambil, total berat menjadi 40 kg. Jalur ini tetap valid karena masih di bawah batas 50 kg.

3. Level 3 - Tenda (20 kg, Profit 150): Jika pada kondisi sebelumnya (Beras + Air = 40 kg) diputuskan untuk mengambil Tenda (x_3=1), maka total berat menjadi 60 kg.
    - Karena 60 > 50, algoritma melakukan **Pruning**. Cabang ini dihentikan karena melanggar kendala kapasitas gudang.
4. Level 4 - Obat-obatan (5 kg, Profit 120): Algoritma mengevaluasi jalur alternatif (Beras + Air + Tanpa Tenda). Dengan menambahkan Obat-obatan (x_4=1), diperoleh total berat 45 kg dengan profit sebesar 300. Ini merupakan solusi optimal sementara pada kedalaman level 4.

2.3 Strategi Pencarian dan Pruning

Algoritma *Backtracking* melakukan penelusuran secara *Depth-First Search* (DFS) dengan menerapkan fungsi pembatas (*bounding function*) untuk efisiensi:

- Kriteria Kelayakan: Sebuah jalur dianggap valid selama total berat kumulatif tidak melebihi kapasitas maksimal gudang yaitu 50 kg.
- Mekanisme Pruning (Pemangkasan) : Jika pada suatu simpul penambahan item berikutnya mengakibatkan total berat > 50 kg, maka simpul tersebut ditandai sebagai simpul mati (ditandai dengan huruf B atau silang). Cabang di bawah simpul tersebut tidak akan dieksplorasi lebih lanjut (pruned), sehingga menghemat waktu komputasi dibandingkan metode Brute Force.
- Contoh Gambar Pruning



BAGIAN 3 : IMPLEMENTASI

3.1 Implementasikan 0/1 Knapsack DP (Java / Python)

- Source Code

```
def knapsack_01(names, weights, values, capacity):
    n = len(names)
    # Inisialisasi tabel DP (n+1 x capacity+1)
    dp = [[0 for _ in range(capacity + 1)] for _ in range(n + 1)]

    # Mengisi tabel DP
    for i in range(1, n + 1):
        for w in range(capacity + 1):
            if weights[i-1] <= w:
                # Memilih antara mengambil barang atau tidak
                dp[i][w] = max(values[i-1] + dp[i-1][w-weights[i-1]], dp[i-1][w])
            else:
                dp[i][w] = dp[i-1][w]

    # Melacak item yang terpilih (Backtracking pada tabel)
    max_value = dp[n][capacity]
    items_terpilih = []
    w_sisa = capacity
    total_berat = 0

    for i in range(n, 0, -1):
        if dp[i][w_sisa] != dp[i-1][w_sisa]:
            items_terpilih.append(names[i-1])
            total_berat += weights[i-1]
            w_sisa -= weights[i-1]

    return max_value, items_terpilih[::-1], total_berat

# --- Data Kasus Bantuan Bencana ---
kapasitas_maks = 50
item_bantuan = [
    ("Beras", 25, 100),
    ("Air Mineral", 15, 80),
    ("Tenda", 20, 150),
```

```
    ("Obat-obatan", 5, 120),
    ("Selimut", 10, 60),
    ("Pakaian", 12, 50),
    ("Makanan Kaleng", 8, 70),
    ("Lampu Senter", 2, 40),
    ("Vitamin", 3, 35),
    ("Alat Masak Portabel", 7, 55)
]

# Memisahkan data ke dalam list masing-masing
nama_item = [x[0] for x in item_bantuan]
berat_item = [x[1] for x in item_bantuan]
nilai_item = [x[2] for x in item_bantuan]

# Menjalankan Fungsi
total_profit, daftar_item, total_w = knapsack_01(nama_item, berat_item, nilai_item,
kapasitas_maks)

# --- Menampilkan Hasil ---
print("=== IMPLEMENTASI 0/1 KNAPSACK: BANTUAN BENCANA ===")
print(f"Kapasitas Maksimal       : {kapasitas_maks} kg")
print("-" * 50)
print(f"Nilai Maksimum (Profit)  : {total_profit}")
print(f"Daftar Item Terpilih     : {', '.join(daftar_item)}")
print(f"Total Berat              : {total_w} kg")
print("-" * 50)
```

- Output

```
=== IMPLEMENTASI 0/1 KNAPSACK: BANTUAN BENCANA ===
Kapasitas Maksimal       : 50 kg
--------------------------------------------------
Nilai Maksimum (Profit)  : 475
Daftar Item Terpilih     : Tenda, Obat-obatan, Selimut, Makanan Kaleng, Lampu Senter, Vitamin
Total Berat              : 48 kg
--------------------------------------------------
```

"Meskipun pada Level 4 profit sementara yang diperoleh adalah 300, proses pencarian algoritma tetap berlanjut hingga Level 10 (Alat Masak Portabel). Melalui pendekatan Dynamic Programming, sistem akhirnya berhasil menemukan kombinasi global yang paling optimal dengan Nilai Maksimum (Profit) sebesar 475 dan Total Berat 48 kg."

3.2 Tampilan Github :

LINK : https://github.com/nurfahira22/Tugas_Klp3_Knapsack

BAGIAN 4 : ANALISIS

4.1 Mengapa kombinasi tersebut optimal?

Kombinasi tersebut dikatakan optimal karena algoritma Dynamic Programming / Backtracking menjamin penemuan solusi global maksimal dengan mengevaluasi jalur yang menghasilkan profit tertinggi tanpa melanggar kendala kapasitas Gudang sebesar 50 kg.

4.2 Perbandingan dengan Brute Force (konsep)

- Brute Force: Mengevaluasi seluruh lintasan dari akar ke daun (1024 kombinasi), baru melakukan cek status valid/tidak di akhir.
- Backtracking: Melakukan cek kendala di tengah proses pembangunan pohon. Jika berat melebihi 50 kg, cabang langsung dibuang (pruning), Seperti terlihat pada *State Space Tree* level 3, ketika berat mencapai 60 kg (Beras + Air + Tenda), cabang tersebut langsung dipotong (**Pruning**), sehingga node yang diperiksa jauh lebih sedikit.

4.3 Kesimpulan

- Optimalitas: Algoritma berhasil menemukan profit maksimal sebesar 475 dengan total berat 48 kg, Hal ini menyisakan hanya 2 kg ruang kosong, yang membuktikan bahwa sumber daya ruang telah digunakan seefektif mungkin karena tidak ada item tersisa di daftar bantuan yang beratnya < 2 kg dengan profit yang signifikan.
- Relevansi: Implementasi ini membuktikan bahwa metode matematis seperti Knapsack Problem sangat aplikatif dalam manajemen logistik bantuan bencana untuk memastikan sumber daya yang terbatas didistribusikan secara maksimal.