ANALISIS 0/1 KNAPSACK PADA OPTIMASI ISI TAS MAHASISWA INFORMATIKA: DYNAMIC PROGRAMMING DENGAN PERBANDINGAN BRUTE FORCE DAN STATE SPACE TREE BACKTRACKING

Mata Kuliah : Grafik Komputer
Nama : Muhammad Rizal Haris dan Andre Firmansyah
NIM : 105841103223 dan 105841101123
Kelas : 5F
Dosen : Desi Anggreani, S.Kom., M.T., MOS
Tahun : 2026

1. ANALISIS MASALAH

1.1 Identifikasi Jenis Knapsack yang Digunakan

Permasalahan optimasi isi tas mahasiswa Informatika termasuk ke dalam 0/1 Knapsack Problem. Hal ini dikarenakan setiap perlengkapan yang akan dimasukkan ke dalam tas, seperti laptop, charger, perangkat penyimpanan, dan perangkat pendukung lainnya, hanya memiliki dua kemungkinan keputusan, yaitu dipilih (1) atau tidak dipilih (0). Item-item tersebut bersifat diskrit dan tidak dapat dibagi menjadi bagian yang lebih kecil, sehingga tidak dimungkinkan untuk mengambil sebagian dari suatu item. Tujuan dari permasalahan ini adalah memaksimalkan total nilai manfaat dari perlengkapan yang dipilih dengan tetap memenuhi kendala kapasitas tas, yaitu batas maksimum berat yang dapat ditampung. Dengan karakteristik tersebut, permasalahan ini secara tepat dimodelkan sebagai 0/1 Knapsack, di mana setiap keputusan direpresentasikan dalam bentuk variabel biner.

1.2 Alasan Penggunaan Dynamic Programming

Dynamic Programming digunakan untuk menyelesaikan permasalahan 0/1 Knapsack karena mampu memberikan solusi optimal melalui pemecahan masalah menjadi submasalah yang lebih kecil dan saling tumpang tindih. Pendekatan ini menyimpan hasil perhitungan sementara dalam bentuk tabel, sehingga menghindari perhitungan ulang yang tidak diperlukan. Dengan Dynamic Programming, solusi optimal diperoleh dengan membangun tabel yang merepresentasikan nilai maksimum yang dapat dicapai untuk setiap kombinasi jumlah item dan kapasitas tas tertentu. Pendekatan ini menjamin bahwa solusi yang dihasilkan merupakan solusi terbaik secara global dan lebih efisien dibandingkan metode brute force yang harus mengevaluasi seluruh kemungkinan kombinasi.

2. STATE SPACE TREE

2.1 Node dan Level

Pada state space tree, setiap node merepresentasikan kondisi sementara pemilihan item (total berat dan total nilai), sedangkan level menunjukkan urutan keputusan item.

- Level 0: kondisi awal (belum memilih item)
- Level i: keputusan terhadap item ke-I
  Setiap level memiliki dua cabang:
- Ambil item ($x_i = 1$)

- Tidak ambil item ($x_i = 0$)

2.2 Data untuk Ilustrasi Level-4

Untuk menggambar pohon hingga minimal level-4, digunakan 4 item pertama dari kasus "Optimasi Isi Tas Mahasiswa Informatika" sebagai berikut:

- I1 (Laptop): w = 2.0 kg, p = 95
- I2 (Charger Laptop): w = 0.5 kg, p = 40
- I3 (Powerbank): w = 0.4 kg, p = 32
- I4 (Headset/Earphone): w = 0.2 kg, p = 22

Kapasitas tas (M) = 3.0 kg (digunakan agar terlihat contoh pruning).

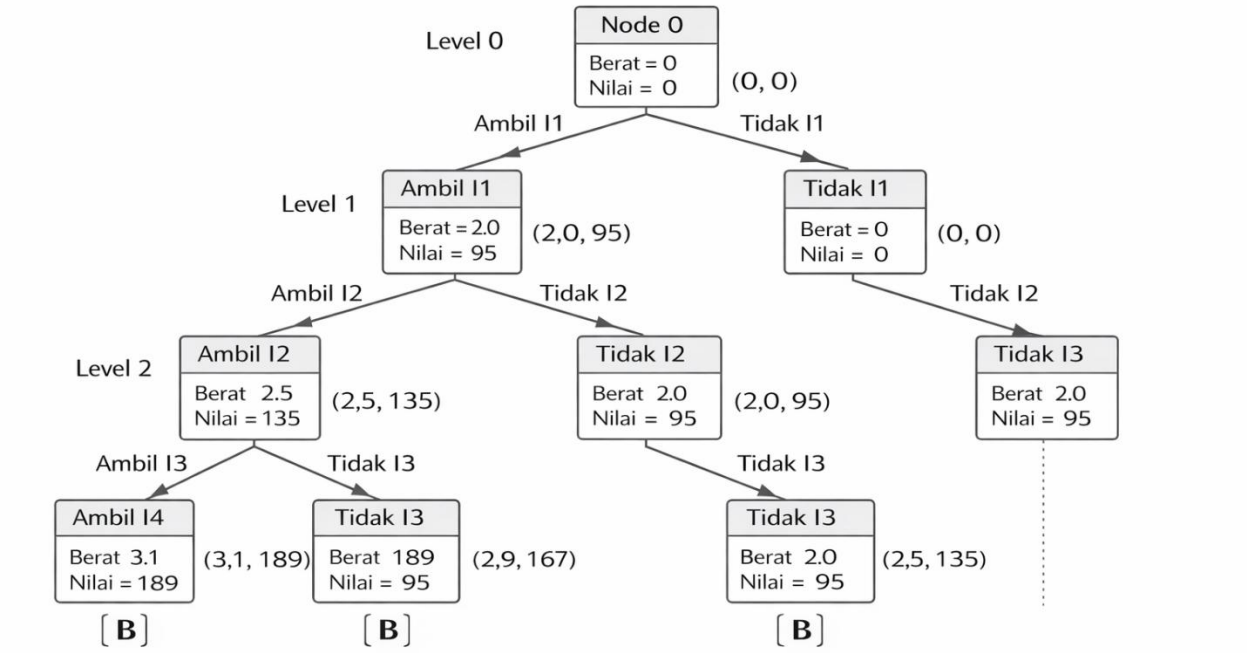
2.3 Pruning (Pemangkasan Cabang)

Pada backtracking, sebuah cabang akan dipangkas (pruned) apabila total berat sementara melebihi kapasitas tas (M). Cabang yang dipangkas ditandai dengan huruf B.

Keterangan: Huruf B menunjukkan cabang yang dipangkas karena total berat melebihi kapasitas tas.

2.4 Gambar State Space Tree sampai Level-4

Gambar 1 menunjukkan state space tree sampai level-4 beserta contoh pruning.



Gambar 1. State Space Tree 0/1 Knapsack hingga Level-4 (dengan pruning/B).

Huruf B menunjukkan cabang yang di pangkas (pruned) karena total berat melebihih kapasitas tas.

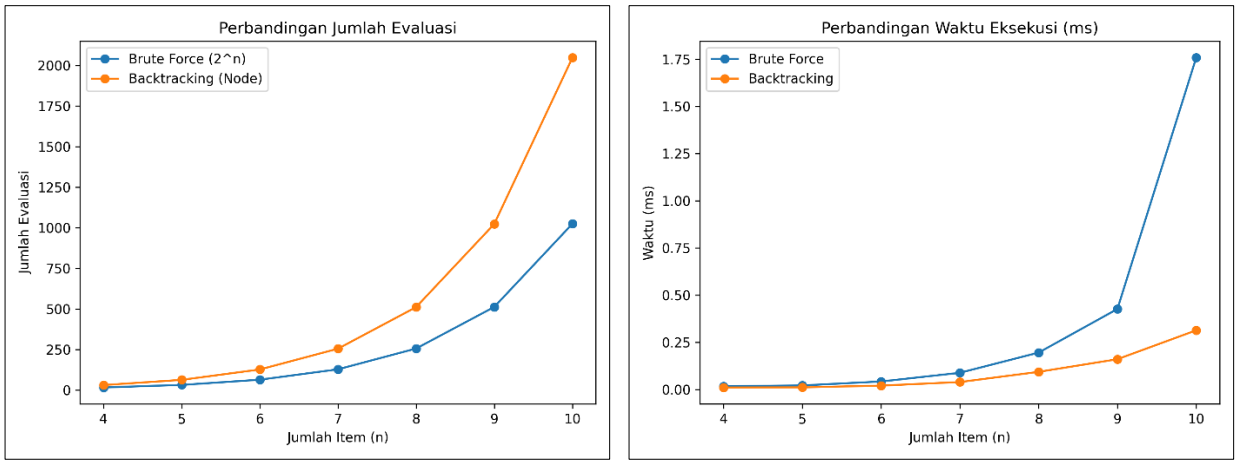
IMPLEMENTASI

Hasil implementasi algoritma Brute Force dan Backtracking pada permasalahan 0/1 Knapsack ditunjukkan dalam Tabel 1. Pengujian dilakukan dengan memvariasikan jumlah item dari 4 hingga 10 item untuk mengamati perubahan jumlah evaluasi, waktu eksekusi, serta nilai optimal yang dihasilkan oleh kedua metode.

Tabel 1. Hasil perbandingan Brute Force dan Backtracking

Jumlah_Item	BruteForce_Kombinasi	Backtracking_Node	Waktu_BruteForce_ms	Waktu_Backtracking_ms	Nilai_Optimal
4	16	31	0.017	0.01	189
5	32	63	0.021	0.011	209
6	64	127	0.042	0.02	239
7	128	255	0.088	0.039	263
8	256	511	0.195	0.093	281
9	512	1023	0.426	0.16	309
10	1024	2047	1.757	0.313	335

Berdasarkan Tabel 1, jumlah kombinasi yang diperiksa oleh metode Brute Force meningkat secara eksponensial seiring bertambahnya jumlah item. Sementara itu, Backtracking mengevaluasi node pencarian dengan jumlah yang lebih terkontrol melalui mekanisme pemangkasan, namun tetap menghasilkan nilai optimal yang sama. Untuk memperjelas perbedaan kinerja antara algoritma Brute Force dan Backtracking, hasil pengujian divisualisasikan dalam bentuk grafik yang menunjukkan perbandingan jumlah evaluasi dan waktu eksekusi terhadap variasi jumlah item.



(a) (b)

Gambar 2. Perbandingan Brute Force dan Backtracking pada variasi jumlah item: (a) jumlah evaluasi, (b) waktu eksekusi (ms).

Grafik pada Gambar 2 menunjukkan bahwa Backtracking memiliki waktu eksekusi yang lebih rendah dibandingkan Brute Force, meskipun nilai optimal yang dihasilkan tetap sama. Hal ini menegaskan bahwa Backtracking lebih efisien dalam menyelesaikan permasalahan 0/1 Knapsack.

ANALISIS

4.1 Mengapa Kombinasi Tersebut Optimal

Berdasarkan hasil implementasi yang ditunjukkan pada Tabel 1, kombinasi item yang dipilih merupakan solusi optimal karena menghasilkan **nilai total maksimum** tanpa melampaui kapasitas tas yang ditetapkan. Setiap item yang terpilih memberikan kontribusi nilai yang signifikan terhadap total manfaat, sementara total berat keseluruhan tetap berada dalam batas kapasitas. Solusi optimal ini diperoleh melalui evaluasi sistematis terhadap kemungkinan pemilihan item. Tidak terdapat kombinasi lain yang menghasilkan nilai lebih tinggi dengan berat yang masih memenuhi kendala kapasitas. Oleh karena itu, kombinasi yang dihasilkan dapat dinyatakan sebagai solusi optimal untuk permasalahan optimasi isi tas mahasiswa Informati

4.2 Perbandingan dengan Brute Force (Konsep)

Metode Brute Force menyelesaikan permasalahan 0/1 Knapsack dengan cara mengevaluasi seluruh kemungkinan kombinasi item, yaitu sebanyak 2^n. Pendekatan ini menjamin diperolehnya solusi optimal, namun memiliki kelemahan utama berupa **kompleksitas komputasi yang tinggi**, terutama ketika jumlah item meningkat. Sebaliknya, algoritma Backtracking menelusuri ruang solusi secara bertahap dan menerapkan mekanisme **pemangkasan (pruning)** pada cabang yang melanggar batas kapasitas. Meskipun Backtracking tetap menjamin solusi optimal, jumlah evaluasi dan waktu eksekusi yang dibutuhkan menjadi lebih kecil dibandingkan Brute Force. Hal ini terlihat dari hasil pengujian, di mana Backtracking memiliki waktu eksekusi yang lebih rendah, sementara nilai optimal yang diperoleh tetap sama.

Link Github

https://github.com/muhammadrizalharis/Tugas1_Knapsack.git