TUGAS

DESAIN ANALISIS DAN ALGORITMA

Dosen pengampuh : DESI ANGGREANI, S.kom.,MT

Disusun oleh :

1. Sarlinda sopalatu (105841101423)

2. Hasriana (105841107623)

PRODI INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MUHAMMADIYAH MAKASSAR

2025

BAGIAN 1

ANALISIS MASALAH

DATA KASUS

LABORATORIUM PENELITIAN

- **Link github : https://github.com/sarlinda-sopalatu/-DESAIN-DAN-ANALISIS-ALGORITMA.git**

No	Item	Volume(cm3)	Nilai (profit)
1	Mikroskop	15	120
2	Tabung reaksi	2	15
3	Centrifuge	8	85
4	Pipet	1	10
5	Spectrophotometer	12	150
6	Hot plate	6	65
7	Gelas ukur	3	25
8	Incubator	20	180
9	pH meter	4	55
10	Autoclave	25	200

Kapasitas rak penyimpanan : 20cm3

Sumber data : inventori laboratorium universitas indonesia 2024

a. Identifikasi jenis knapsack yang digunakan Berdasarkan karakteristik kasus "Laboratorium Penelitian" yang diberikan, jenis knapsack yang digunakan adalah 0/1 Knapsack Problem.

b. Alasan penggunaan dynamic programming

- Optimalitas pada Kapasitas Terbatas: Dengan kapasitas rak yang hanya 50 cm^3, metode manual atau Greedy berisiko melewatkan kombinasi terbaik. DP akan membandingkan apakah lebih untung mengambil satu barang besar seperti Autoclave (Volume 25, Profit 200) atau beberapa barang kecil seperti pH Meter + Hot Plate + Centrifuge yang mungkin memiliki total profit lebih tinggi dengan volume yang lebih efisien.
- Menghindari Perhitungan Berulang (Memoization): Karena ada 10 item, terdapat 2^{10} = 1024 kombinasi kemungkinan. DP memecah masalah ini menjadi sub-masalah kecil (misal: mencari nilai optimal untuk kapasitas 1 cm^3, 2 cm^3, dst.) dan menyimpan hasilnya dalam tabel. Hal ini mencegah komputer menghitung ulang kombinasi yang sama berkali-kali.
- Akurasi Inventori: Dalam konteks laboratorium penelitian, nilai profit (pentingnya alat) harus dimaksimalkan secara presisi. DP memberikan jaminan solusi paling optimal secara matematis dibandingkan metode coba-coba.

BAGIAN 2

STATE SPACE TREE

1. **Diketahui :**
    - Jumlah item n = 10
    - Kapasistas knapsack M = 20 cm3
    - Bobot (weight):

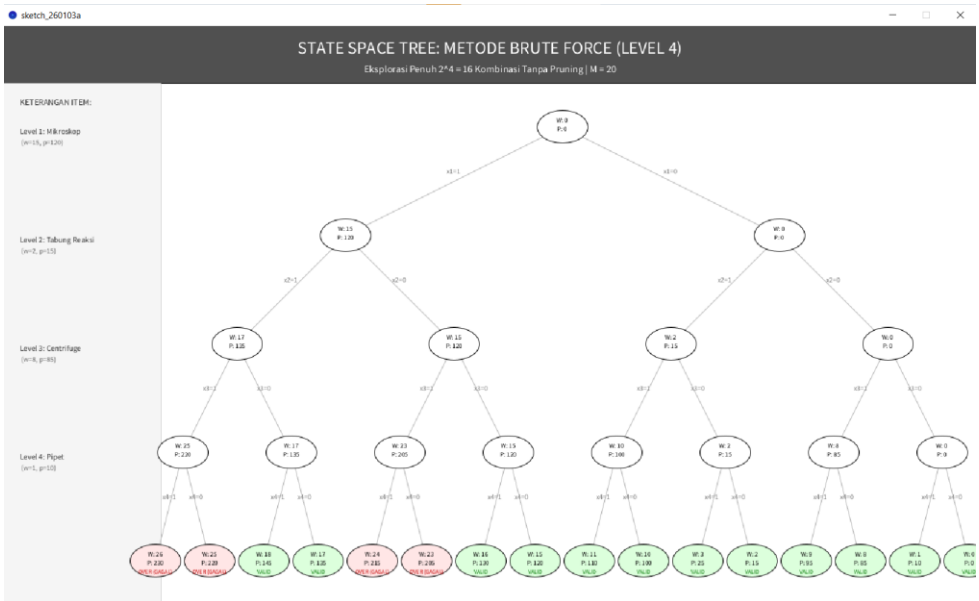
      W = (15, 2, 8, 1, 12, 6, 3, 20, 4, 25)
    - Keuntungan (profit) :

      P = (120, 15, 85, 10, 150, 65, 25 , 18-, 55, 200)
    - Variabel keputusan:

$$x_i = \begin{cases} 1, & \textit{item ke-i diambil} \\ 0, & \textit{item ke-i tidak diambil} \end{cases}$$

2. **Konsep Metode brute force dan knapsack**

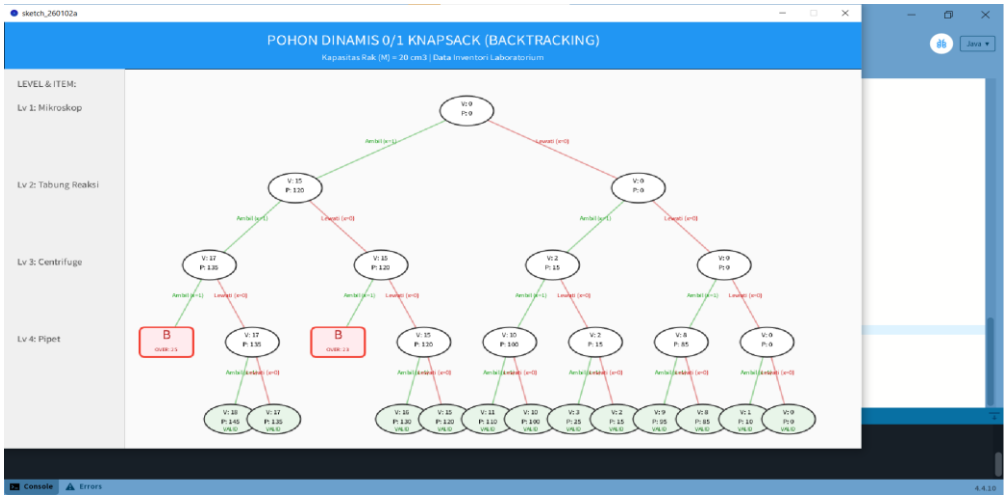
   Jumlah kemungkinan = 2^4 = 16 kombinasi (level 4)
    - Metode brute force



Kombinasi (x1,x2,x3 ,x4)	**Perhitungan Total Berat**	**Total Profit**	**Status (≤20)**
(1, 1, 1, 1)	15+2+8+1 = 26	230	Gagal
(1, 1, 1, 0)	15+2+8 = 25	220	Gagal
(1, 1, 0, 1)	15+2+1 = 18	145	Valid
(1, 1, 0, 0)	15+2 = 17	135	Valid
(1, 0, 1, 1)	15+8+1 = 24	215	Gagal
(1, 0, 1, 0)	15+8 = 23	205	Gagal
(1, 0, 0, 1)	15+1 = 16	130	Valid
(1, 0, 0, 0)	15	120	Valid
(0, 1, 1, 1)	2+8+1 = 11	110	Valid
(0, 1, 1, 0)	2+8 = 10	100	Valid
(0, 1, 0, 1)	2+1 = 3	25	Valid
(0, 1, 0, 0)	2	15	Valid
(0, 0, 1, 1)	8+1 = 9	95	Valid

Kombinasi (x1,x2,x3 ,x4)	Perhitungan Total Berat	Total Profit	Status (≤20)
(0, 0, 1, 0)	8	85	Valid
(0, 0, 0, 1)	1	10	Valid
(0, 0, 0, 0)	0	0	Valid

- **Metode knapshak**



3. Penentuan node dan level

- Node: Setiap titik yang menyimpan informasi Berat (W) dan Profit (P) sementara.
- Level 0: Akar (Root), kondisi awal sebelum memilih item.
- Level 1: Keputusan untuk Item 1 (Mikroskop).
- Level 2: Keputusan untuk Item 2 (Tabung Reaksi).
- Level 3: Keputusan untuk Item 3 (Centrifuge).
- Level 4: Keputusan untuk Item 4 (Pipet).

4. Penjelasan pruning

Mekanisme Pruning pada Kasus (M = 20)

Pada program Processing yang kita buat, pruning ditandai dengan kotak merah dan huruf "B" (Bound/Limit). Perhitungan untuk pruning :

- Item 1 (Mikroskop): Berat = 15 cm^3. (Keputusan $x\_1=1$, status: W=15, Valid karena $15 < 20$).
- Item 2 (Tabung Reaksi): Berat = 2 cm^3. Jika diambil ($x\_2=1$), status: W = 15 + 2 = 17, Valid karena $17 < 20$.
- Item 3 (Centrifuge): Berat = 8 cm^3.
  - Jika kita berada di jalur $x\_1=1$, $x\_2=1$ dan mencoba mengambil Centrifuge ($x\_3=1$), maka:
  - Perhitungan: 17 + 8 = 25.
  - Evaluasi: Karena $25 > 20$, maka terjadi pelanggaran kendala kapasitas.
  - Tindakan (Pruning): Algoritma akan melakukan pemangkasan. Program tidak akan melanjutkan pencarian ke item berikutnya (Pipet, dst.) di bawah cabang ini.

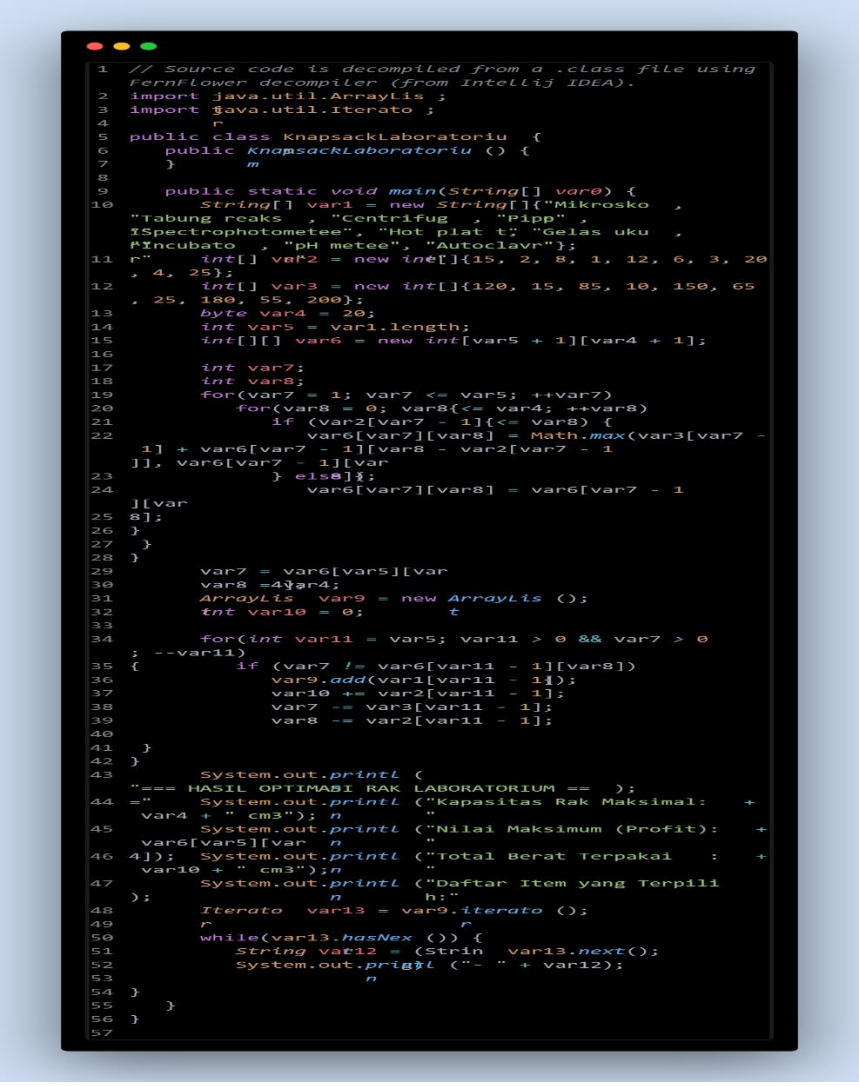
\

BAGIAN 3

IMPLEMENTASI

1. Implementasikan 0/1 kanpsack DP

- **Source Code**



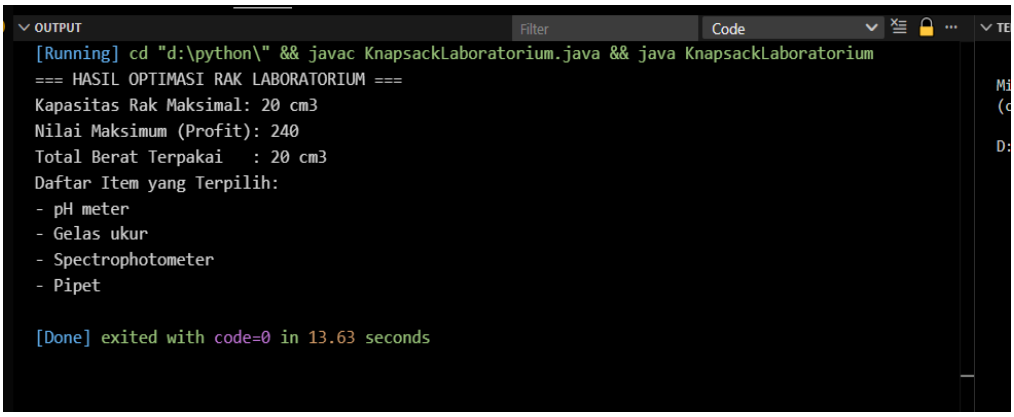
```
// Source code is decompiled from a .class file using FernFlower decompiler (from Intellij IDEA).
import java.util.ArrayLis ;
import java.util.Iterato ;
public class KnapsackLaboratoriu  {
    public KnapsackLaboratoriu () {
    }
    public static void main(String[] var0) {
        String[] var1 = new String[]{"Mikrosko  , "Tabung reaks  , "Centrifug  , "Pipp"  , "Spectrophotometee", "Hot plat t", "Gelas uku  , "Incubato  , "pH metee", "Autoclavr"};
        int[] var2 = new int[]{15, 2, 8, 1, 12, 6, 3, 20, 4, 25};
        int[] var3 = new int[]{120, 15, 85, 10, 150, 65, 25, 180, 55, 200};
        byte var4 = 20;
        int var5 = var1.length;
        int[][] var6 = new int[var5 + 1][var4 + 1];

        int var7;
        int var8;
        for(var7 = 1; var7 <= var5; ++var7)
            for(var8 = 0; var8<= var4; ++var8)
                if (var2[var7 - 1]<= var8) {
                    var6[var7][var8] = Math.max(var3[var7 - 1] + var6[var7 - 1][var8 - var2[var7 - 1]], var6[var7 - 1][var
                } else
                    var6[var7][var8] = var6[var7 - 1][var8];
        }
    }
}
        var7 = var6[var5][var
        var8 =4;
        ArrayLis  var9 = new ArrayLis ();
        int var10 = 0;        t

        for(int var11 = var5; var11 > 0 && var7 > 0; --var11)
        {         if (var7 != var6[var11 - 1][var8])
                var9.add(var1[var11 - 1]);
                var10 += var2[var11 - 1];
                var7 -= var3[var11 - 1];
                var8 -= var2[var11 - 1];

    }
}
        System.out.println (
"=== HASIL OPTIMASI RAK LABORATORIUM ==  );
        System.out.println ("Kapasitas Rak Maksimal:    + var4 + " cm3"); n        "
        System.out.println ("Nilai Maksimum (Profit):    + var6[var5][var  n        "
4]);  System.out.println ("Total Berat Terpakai    :    + var10 + " cm3");n        "
        System.out.println ("Daftar Item yang Terpili
);                  n        h:"
        Iterato  var13 = var9.iterato ();
        r                        r
        while(var13.hasNex  ()) {
            String var12 = (Strin  var13.next();
            System.out.println ("-  " + var12);
                        n
}
    }
}
```

- **Outputnya**


OUTPUT
Filter
Code
[Running] cd "d:\python\" && javac KnapsackLaboratorium.java && java KnapsackLaboratorium
=== HASIL OPTIMASI RAK LABORATORIUM ===
Kapasitas Rak Maksimal: 20 cm3
Nilai Maksimum (Profit): 240
Total Berat Terpakai : 20 cm3
Daftar Item yang Terpilih:
- pH meter
- Gelas ukur
- Spectrophotometer
- Pipet
[Done] exited with code=0 in 13.63 seconds

BAGIAN 4
ANALISIS

1. Mengapa Kombinasi Tersebut Optimal?

   Prinsip Optimalitas: Metode *Dynamic Programming* memecah masalah besar menjadi sub-masalah yang lebih kecil (misalnya, mencari profit maksimal untuk kapasitas rak yang lebih rendah terlebih dahulu). Akurasi Keputusan: Pada setiap langkah, algoritma membandingkan dua opsi secara matematis: mengambil item tersebut (jika kapasitas cukup) atau melewatkannya. Algoritma hanya memilih opsi yang menghasilkan profit tertinggi. Hasil Pasti: Berbeda dengan metode Greedy yang mungkin gagal menemukan solusi terbaik, Dynamic Programming menjamin penemuan solusi global optimal dengan mempertimbangkan semua hubungan antar bobot dan keuntungan secara sistematis.

2. Perbandingan dengan Brute Force (Konsep)

Aspek	Metode Brute Force	Dynamic Programming / Backtracking
Jumlah Evaluasi	Mengevaluasi seluruh 2^n kombinasi (untuk 10 item berarti 1024 kemungkinan).	Mengevaluasi sub-masalah yang relevan atau memangkas cabang yang tidak valid (*pruning*).
Waktu Eksekusi	Sangat lambat karena harus menghitung setiap node hingga level terakhir.	Jauh lebih cepat karena menghindari perhitungan ulang dan menghentikan jalur yang melanggar kendala.

Aspek	Metode Brute Force	Dynamic Programming / Backtracking
Cek Kendala	Pengecekan berat ($\le M$) baru dilakukan di akhir lintasan (level daun).	Pengecekan dilakukan di setiap node (tengah proses). Jika berat melebihi M, langsung dipangkas (*pruned*).
Kompleksitas	Eksponensial (Sangat tidak efisien untuk data besar).	Pseudo-polinomial / Jauh lebih efisien untuk masalah dengan kapasitas terbatas.

Kesimpulanan analisis :

Penggunaan metode Backtracking (dengan pruning) dan Dynamic Programming pada kasus laboratorium ini jauh lebih efektif dibandingkan brute force. Pruning pada pohon dinamis memungkinkan sistem mengabaikan ratusan kombinasi yang tidak valid sejak awal, sehingga mempercepat pencarian profit maksimal di dalam kapasitas rak yang hanya 20 cm^3.