**LAPORAN PRAKTIKUM**

**PRAKTIKUM 7 – DYNAMIC PROGRAMMING**

Nama/ NIM : Whisnumurty Galih Ananta/ 1203210002

Mata Kuliah/ Kelas : Perancangan dan Analisis Algoritma/ IF-01-03

Dosen Pengampu : Ibu Vessa Rizky Oktavia, M.Kom.

Source Code : [praktikum-PAA/Praktikum 7 - KNAPSACK/knapsack.cpp at master · whisnumurtyga/praktikum-PAA · GitHub](https://github.com/whisnumurtyga/praktikum-PAA/blob/master/Praktikum%207%20-%20KNAPSACK/knapsack.cpp)

Soal : [SPOJ.com - Problem KNAPSACK](https://www.spoj.com/problems/KNAPSACK/)

**Penjelasan Soal**

**Problem: Masalah Knapsack**

* Anda akan mempacking untuk liburan di tepi pantai dan hanya bisa membawa satu tas dengan kapasitas S.
* Anda memiliki N item yang mungkin ingin Anda bawa, tetapi tidak semua item tersebut dapat masuk ke dalam knapsack.
* Setiap item dideskripsikan oleh ukurannya dan nilainya.
* Tujuan Anda adalah memaksimalkan total nilai dari item yang Anda bawa.

**Input**:

* Pada baris pertama, Anda diberikan dua bilangan bulat, S dan N, yang mewakili kapasitas knapsack dan jumlah item.
* N baris berikutnya masing-masing berisi dua bilangan bulat, yang mewakili ukuran dan nilai masing-masing item.

**Output**:

* Keluarkan satu bilangan bulat yang mewakili nilai total maksimum dari pilihan item terbaik untuk perjalanan Anda.

Contoh:

**Input**:

4 5

1 8

2 4

3 0

2 5

2 3

**Output**:

13

**Penjelasan**:

* Knapsack memiliki kapasitas 4 dan terdapat 5 item untuk dipilih.
* Item-item tersebut dideskripsikan oleh ukurannya dan nilainya.
* Dengan memilih item dengan ukuran 1, 2, dan 2, dan nilai masing-masing 8, 4, dan 5, kita dapat mencapai total nilai sebesar 13.
* Item-item yang dipilih memenuhi kapasitas knapsack dan nilai-nilainya dimaksimalkan.

**Rule Contoh :**

Max Kapasitas : 4

Max Item : 5

|  |  |  |
| --- | --- | --- |
| **Item Ke** | **Kapasitas** | **Nilai** |
| 1 | 1 | 8 |
| 2 | 2 | 4 |
| 3 | 3 | 0 |
| 4 | 2 | 5 |
| 5 | 2 | 3 |

**Table Item**

|  |  |  |  |
| --- | --- | --- | --- |
| **Iterasi** | **Item yang dipertimbangkan** | **Kapasitas Tersisa** | **Total Nilai Maksimum** |
| 1 | 1 | 3 | 8 |
| 2 | 2 | 2 | 8 |
| 3 | 3 | 2 | 8 |
| 4 | 4 | 0 | 13 |
| 5 | 5 | 0 | 13 |

**Table Iterasi**

**Solusi**

* Menyertakan pustaka iostream dan vector yang diperlukan untuk menggunakan input/output dan vektor dalam program.

A screen shot of a computer

Description automatically generated

* Fungsi sederhana yang digunakan untuk mencari nilai maksimum antara dua bilangan.

A computer screen shot of text

Description automatically generated

* Fungsi utama yang memecahkan masalah Knapsack. Ini mengambil kapasitas knapsack (S), vektor ukuran item (sizes), vektor nilai item (values), dan jumlah item (N) sebagai argumen.

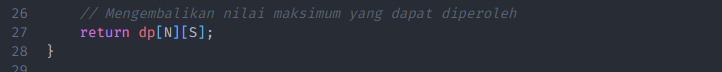


* Dalam fungsi ini, kita membuat sebuah vektor 2D dp dengan ukuran (N+1) x (S+1) dan menginisialisasi semua elemennya dengan 0. Vektor dp ini akan digunakan untuk menyimpan nilai maksimum yang dapat dicapai pada setiap langkah iterasi.
* Ini adalah langkah utama dalam memecahkan masalah Knapsack menggunakan pendekatan dynamic programming.

A screen shot of a computer code

Description automatically generated

* Dalam dua loop for, kita mengiterasi melalui semua kemungkinan ukuran kapasitas knapsack dan jumlah item.
* Pada setiap langkah iterasi, kita mempertimbangkan item saat ini dan memeriksa apakah kita dapat menyertakannya ke dalam knapsack atau tidak.
* Jika ukuran item saat ini lebih kecil atau sama dengan kapasitas yang sedang dipertimbangkan, maka kita membandingkan nilai antara memasukkan item saat ini ke knapsack (values[i - 1] + dp[i - 1][j - sizes[i - 1]]) dengan tidak memasukkan item saat ini (dp[i - 1][j]). Kami mengambil nilai maksimum dari dua opsi ini dan menyimpannya di dp[i][j].
* Jika ukuran item saat ini lebih besar dari kapasitas yang sedang dipertimbangkan, maka kita hanya menyalin nilai dari baris sebelumnya, yaitu dp[i - 1][j].



* Fungsi knapSack mengembalikan nilai maksimum yang dapat diperoleh dari knapsack.
* Dalam fungsi main, kita mengambil input kapasitas knapsack (S) dan jumlah item (N) dari pengguna.
* Kemudian, kita membuat dua vektor, yaitu sizes dan values, untuk menyimpan ukuran dan nilai item-item tersebut.
* Selanjutnya, kita memanggil fungsi knapSack dengan parameter yang sesuai dan menyimpan hasilnya dalam variabel maxTotalValue.

Akhirnya, kita mencetak nilai maksimum total yang dapat diperoleh dari knapsack.

A computer screen shot of code

Description automatically generated

**Algoritma Dynamic Programming**

1. Menerima input kapasitas knapsack (S) dan jumlah item (N) dari pengguna.
2. Membuat dua vektor, yaitu sizes dan values, untuk menyimpan ukuran dan nilai item-item yang akan dipertimbangkan.
3. Membuat sebuah matriks 2D dp dengan ukuran (N+1) x (S+1) dan menginisialisasi semua elemennya dengan 0.
4. Memulai iterasi dari 1 hingga N (item saat ini).
5. Di dalam iterasi item, melakukan iterasi dari 1 hingga S (kapasitas saat ini).
6. Memeriksa apakah ukuran item saat ini lebih kecil atau sama dengan kapasitas yang sedang dipertimbangkan.
   1. Jika ya, maka kita membandingkan nilai antara memasukkan item saat ini ke dalam knapsack (nilai item saat ini + nilai maksimum pada langkah sebelumnya dengan kapasitas yang tersisa) dengan tidak memasukkan item saat ini (nilai maksimum pada langkah sebelumnya dengan kapasitas yang sama).
   2. Mengambil nilai maksimum dari dua opsi tersebut dan menyimpannya di dp[i][j].
7. Jika ukuran item saat ini lebih besar dari kapasitas yang sedang dipertimbangkan, maka hanya menyalin nilai dari baris sebelumnya, yaitu dp[i-1][j].
8. Setelah semua langkah iterasi selesai, nilai maksimum total yang dapat diperoleh dari knapsack terdapat di dp[N][S].
9. Mengembalikan nilai maksimum tersebut.
10. Mencetak nilai maksimum total yang diperoleh sebagai hasil.