

# PROJECT DESAIN & ANALISIS ALGORITMA

Laporan Problem Greedy

Disusun untuk Memenuhi Tugas Mata Kuliah Desain & Analisis Algoritma

Dosen Pengampu:

**Fajar Muslim, S. T., M. T.,**



Disusun Oleh :

- |                                |            |
|--------------------------------|------------|
| 1. Najwa Adlina Yanida         | (L0123105) |
| 2. Nazila Nur Hanifa Ramadhani | (L0123109) |

**FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA  
UNIVERSITAS SEBELAS MARET  
2024**

## **BAB 1**

### **PENDAHULUAN**

#### **A. LATAR BELAKANG**

Coin Change Problem adalah satu masalah pada algoritma yang sering dihadapi dalam dunia pemrograman dan matematika. Dalam masalah ini, kita diberikan sejumlah koin dengan denominasi (nilai nominal dari setiap koin) tertentu dan tujuan dari ini adalah menentukan cara optimal untuk membuat sejumlah uang dengan menggunakan koin-koin tersebut. Pada dasarnya, masalah ini meminta kita untuk menentukan kombinasi minimum dari sejumlah koin untuk mencapai jumlah uang tertentu yang ditetapkan. Salah satu pendekatan populer untuk menyelesaikan masalah ini adalah dengan menggunakan *algoritma greedy*.

Pada laporan ini, kita akan membahas bagaimana algoritma greedy dapat digunakan untuk menyelesaikan *Coin Change Problem* dengan denominasi koin yaitu 1, 2, 5, dan 10, serta menganalisis mengapa pendekatan ini efektif untuk kasus tertentu. Tantangannya adalah mencari kombinasi koin-koin tersebut untuk menghasilkan nilai tertentu dengan jumlah koin yang seminimal mungkin.

## **BAB 2**

### **TEORI SINGKAT**

#### **A. Definisi Algoritma Greedy**

Algoritma Greedy adalah pendekatan dalam pemecahan masalah optimasi di mana setiap langkah diambil dengan memilih solusi terbaik secara lokal tanpa mempertimbangkan dampak jangka panjang. Algoritma ini bertujuan untuk mencapai solusi yang optimal dengan terus-menerus memilih opsi yang tampak terbaik pada setiap langkah. Contoh klasik penggunaan algoritma greedy meliputi masalah Coin Change, algoritma Dijkstra untuk pencarian jalur terpendek, dan algoritma Kruskal untuk menemukan Minimum Spanning Tree.

Algoritma greedy bekerja dengan prinsip sederhana yaitu memilih solusi yang tampak paling sederhana pada saat itu, kemudian ulangi langkah tersebut hingga masalah selesai. Namun, algoritma ini tidak selalu menjamin solusi optimal untuk semua masalah. Dalam beberapa kasus, solusi optimal global tidak dapat dicapai karena dengan pemilihan lokal tidak selalu akan menghasilkan solusi terbaik secara keseluruhan. Agar algoritma greedy memberikan solusi optimal, masalah harus memenuhi dua syarat utama yaitu greedy choice property (solusi lokal terbaik juga merupakan bagian dari solusi optimal global) dan optimal substructure (masalah dapat dipecah menjadi submasalah yang dapat diselesaikan secara independen). Algoritma ini efisien dan mudah diterapkan, tetapi tidak cocok untuk semua jenis masalah, terutama jika struktur masalahnya kompleks atau memerlukan penarikan keputusan sebelumnya. Algoritma greedy sering memberikan hasil cepat dan baik dalam masalah sederhana, seperti masalah kembalian koin atau pemilihan aktivitas. Namun, algoritma ini dapat gagal pada masalah yang lebih kompleks di mana solusi lokal yang serakah tidak selalu menghasilkan solusi global yang optimal.

#### **B. Penerapan Algoritma Greedy**

Dalam masalah Coin Change, kita diberikan sejumlah koin dengan denominasi tertentu dan jumlah total yang ingin dicapai. Tugas kita adalah menentukan jumlah minimum koin yang diperlukan untuk mencapai total tersebut. Misalkan kita memiliki koin dengan denominasi: 1, 2, 5, dan 10. Kita ingin menghitung kombinasi koin yang diperlukan untuk mencapai total tertentu, misalnya 37. Langkah-langkah penerapan algoritma greedy untuk Coin Change Problem adalah sebagai berikut:

1. **Mengurutkan Koin:** Memastikan koin diurutkan dari yang terbesar ke yang terkecil. Dalam kasus ini, urutannya adalah 10, 5, 2, dan 1.
2. **Menginisialisasi Variabel:** Menyiapkan variabel untuk menyimpan jumlah total yang ingin dicapai dan daftar untuk menyimpan koin yang digunakan.
3. **Mengiterasi Melalui Koin:** Untuk setiap jenis koin, melakukan hal berikut:
  - Menghitung berapa banyak koin tersebut yang dapat digunakan tanpa melebihi total yang diinginkan.

- Mengurangi total yang tersisa dengan jumlah nilai koin yang digunakan.
  - Menyimpan koin yang digunakan dalam daftar.
4. **Hentikan Ketika Total Dicapai:** Proses ini berlanjut sampai total yang diinginkan tercapai.

### C. Contoh Pengimplementasian Algoritma Greedy

Input: 37

- **Langkah 1:** Pilih koin terbesar  $\leq 37$ , yaitu 10.  
Total nilai:  $37 - 10 = 27$ , hasil: [10]
- **Langkah 2:** Pilih koin terbesar lagi, yaitu 10.  
Total nilai:  $27 - 10 = 17$ , hasil: [10, 10]
- **Langkah 3:** Pilih koin terbesar lagi, yaitu 10.  
Total nilai:  $17 - 10 = 7$ , hasil: [10, 10, 10]
- **Langkah 3:** Pilih koin terbesar  $\leq 5$ , yaitu 5.  
Total nilai:  $7 - 5 = 2$ , hasil: [10, 10, 5]
- **Langkah 4:** Pilih koin terbesar  $\leq 2$ , yaitu 2.  
Total nilai:  $2 - 2 = 0$ , hasil: [10, 10, 5, 2]

Proses dan letak *Greedy*:

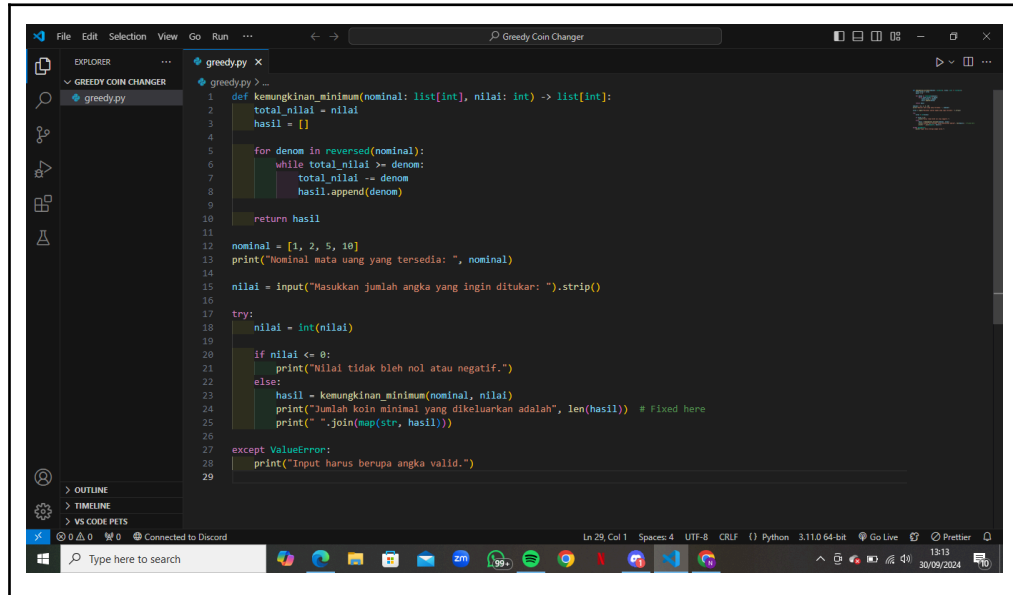
1. **Memilih Denominasi Terbesar:** Algoritma memilih koin terbesar yang tidak melebihi nilai untuk mengurangi total nilai dengan cepat.
2. **Iterasi Hingga Selesai:** Setelah memilih denominasi, algoritma terus mengurangi total nilai hingga tidak bisa lagi dikurangi oleh denominasi tersebut.
3. **Melanjutkan ke Denominasi Berikutnya:** Jika total nilai masih tersisa, algoritma beralih ke denominasi yang lebih kecil dan mengulangi proses.

## BAB 3

### KODE DAN PENGGUNAAN

#### Source code

- greedy.py



```
1 def kemungkinan_minimum(nominal: list[int], nilai: int) -> list[int]:
2     total_nilai = nilai
3     hasil = []
4
5     for denom in reversed(nominal):
6         while total_nilai >= denom:
7             total_nilai -= denom
8             hasil.append(denom)
9
10    return hasil
11
12    nominal = [1, 2, 5, 10]
13    print("Nominal mata uang yang tersedia: ", nominal)
14
15    nilai = input("Masukkan jumlah angka yang ingin ditukar: ").strip()
16
17    try:
18        nilai = int(nilai)
19
20        if nilai <= 0:
21            print("Nilai tidak boleh nol atau negatif.")
22        else:
23            hasil = kemungkinan_minimum(nominal, nilai)
24            print("Jumlah koin minimal yang dikeluarkan adalah", len(hasil)) # Fixed here
25            print(" ".join(map(str, hasil)))
26
27    except ValueError:
28        print("Input harus berupa angka valid.")
29
```

#### Output program

Output	Keterangan
<pre>Nominal mata uang yang tersedia: [1, 2, 5, 10] Masukkan jumlah angka yang ingin ditukar: 37 Jumlah koin minimal yang dikeluarkan adalah 5 10 10 10 5 2 PS C:\Users\SOLUSINDO\OneDrive\Desktop\Kuliah\SEM</pre>	Hasil output dari program menunjukkan nilai minimum dari operasi matematika yang menghasilkan nilai target 37 yaitu (10, 10, 10, 5, 2) sesuai dengan batas angka yang diberikan oleh pengguna dari denominasi (1, 2, 5, dan 10).

Output	Keterangan
<pre>Nominal mata uang yang tersedia: [1, 2, 5, 10] Masukkan jumlah angka yang ingin ditukar: 147 Jumlah koin minimal yang dikeluarkan adalah 16 10 10 10 10 10 10 10 10 10 10 10 10 5 2 PS C:\Users\SOLUSINDO\OneDrive\Desktop\Kuliah\SEM to Discord</pre>	Hasil output dari program menunjukkan nilai minimum dari operasi matematika yang menghasilkan nilai target 147 sesuai dengan batas angka yang diberikan oleh pengguna dari denominasi (1, 2, 5, dan 10).

## **BAB 4**

### **KESIMPULAN**

Change Problem adalah masalah klasik dalam algoritma yang bertujuan menemukan kombinasi koin dengan denominasi tertentu untuk mencapai total uang dengan jumlah koin minimal. Pendekatan algoritma greedy efektif dalam menyelesaikan masalah yang sederhana. Meskipun algoritma greedy memberikan solusi yang cepat dan sederhana, ia tidak selalu menjamin solusi optimal untuk semua masalah, sehingga penting untuk mempertimbangkan strukturnya sebelum penerapan.

Pendekatan algoritma *Greedy* efektif hanya untuk kasus-kasus tertentu. Karena algoritma ini mengandalkan pilihan lokal terbaik tanpa mempertimbangkan konsekuensi jangka panjang. Hal ini menyebabkan hasil yang dihasilkan tidak selalu optimal, terutama pada suatu masalah yang kompleks.