LAPORAN DOKUMENTASI TUGAS COIN CHANGE

DESAIN ANALISIS & ALGORITMA

DISUSUN OLEH:

1. Mario Valentino Ardhana (L0123079)
2. Nabil Riano Zaky (L0123103)

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI DAN SAINS DATA

UNIVERSITAS SEBELAS MARET

2024

Pendahuluan

Masalah coin change merupakan permasalahan klasik di bidang algoritma. Tujuan dari masalah ini adalah menentukan berapa jumlah minimum koin yang diperlukan untuk mencapai suatu jumlah target uang (X), dengan asumsi bahwa kita memiliki kumpulan nilai koin tertentu. Pada contoh ini, nilai koin yang digunakan adalah [1, 2, 5, 10].

Metode yang digunakan dalam solusi ini adalah algoritma greedy, di mana kita selalu memilih koin dengan nilai terbesar terlebih dahulu, kemudian terus mengurangi nilai target sampai mencapai 0.

Struktur Kode dan Penjelasan

1. Fungsi Greedy Coin Change

```
def greedy_coin_change(X, arr=[1, 2, 5, 10]):
    arr.sort(reverse=True)  # Mengurutkan array dari besar ke kecil
    coin_count = {}  # Dictionary untuk menyimpan berapa banyak
setiap nilai coin yang digunakan

    for coin in arr:
        if X >= coin:
            count = X // coin  # Banyaknya koin yang digunakan
            X -= coin * count  # Mengurangi nilai X
            coin_count[coin] = count  # Simpan jumlah koin

        # Jika sudah X = 0, keluar dari loop
        if X == 0:
            break

    return coin_count
```

- Fungsi ini menerima parameter X yang merupakan jumlah uang yang ingin dicapai, dan arr sebagai array koin yang akan digunakan.
- Array koin diurutkan secara menurun (arr.sort(reverse=True)) untuk memastikan koin terbesar diproses terlebih dahulu.
- Dalam perulangan for, untuk setiap koin, program memeriksa apakah nilai X lebih besar atau sama dengan nilai koin tersebut. Jika ya, program menghitung berapa banyak koin tersebut yang bisa digunakan (count = X // coin) dan menguranginya dari target X.

- Program menyimpan jumlah koin yang digunakan di dalam dictionary coin_count.

2. Input Dari Pengguna

```
# Meminta input dari user
while True:
    X = int(input("Masukkan jumlah uang (harus lebih dari 0): "))
    if X > 0:
        break
    else:
        print("Error: Jumlah uang harus lebih dari 0. Silakan coba
lagi.")
```

Program meminta pengguna untuk memasukkan nilai X, yaitu jumlah target uang yang ingin dicapai. User harus memasukkan jumlah uang lebih dari nol.

3. Menampilkan Hasil

```
# Memanggil fungsi dan mencetak hasilnya
result = greedy_coin_change(X)

print(f"Hasil untuk jumlah uang {X}:")
for coin, count in result.items():
    print(f"Koin {coin}: {count} kali")
```

Setelah fungsi dieksekusi, program mencetak jumlah koin yang digunakan untuk setiap denominasi.

Analisis Kompleksitas

1. Kompleksitas Waktu
    - Mengurutkan array membutuhkan kompleksitas waktu sebesar O(n log n), di mana n adalah jumlah elemen dalam array (dalam kasus ini, n = 4 karena ada empat nilai koin: [1, 2, 5, 10]).
    - Looping untuk memeriksa setiap koin memiliki kompleksitas O(n). Namun, karena n kecil (hanya 4 elemen), ini tidak menjadi bottleneck.
    - Sehingga total kompleksitas waktu algoritma ini adalah O(n log n + n), yang dalam konteks ini bisa disederhanakan menjadi O(n log n).

2. Kompleksitas Ruang
    - Penggunaan memori tambahan hanya berupa dictionary coin_count untuk menyimpan jumlah koin yang digunakan. Karena kita hanya menyimpan maksimal 4 nilai, kompleksitas ruangnya adalah O(n), di mana n adalah jumlah denominasi koin.

Apakah Selalu Optimal?

1. Optimal untuk kasus tretentu

    Algoritma greedy bekerja optimal jika nilai koin yang digunakan memenuhi properti tertentu. Misalnya, untuk kumpulan koin seperti [1, 2, 5, 10], algoritma greedy akan selalu menghasilkan solusi optimal karena setiap koin dapat direpresentasikan sebagai kombinasi koin yang lebih kecil.

2. Tidak selalu optimal untuk koin arbiter

    Jika kita menggunakan kumpulan koin lain (misalnya [1, 3, 4]), algoritma greedy tidak selalu menghasilkan solusi optimal. Sebagai contoh, untuk mencapai nilai 6 dengan koin [1, 3, 4], algoritma greedy akan memilih satu koin 4 dan dua koin 1 (total 3 koin), padahal solusi optimalnya adalah menggunakan dua koin 3 saja (total 2 koin). Dalam kasus semacam ini, solusi optimal hanya dapat ditemukan dengan algoritma lain seperti dynamic programming.

Kesimpulan

Algoritma greedy yang diterapkan pada masalah coin change merupakan pendekatan yang cepat dan efisien untuk kumpulan koin tertentu yang memenuhi syarat optimal. Dalam implementasi ini, kita dapat melihat bahwa algoritma tersebut mampu menemukan solusi yang optimal jika kumpulan koinnya sesuai (misalnya, [1, 2, 5, 10]). Kompleksitas waktu dari algoritma ini cukup rendah, menjadikannya solusi yang baik untuk masalah ini ketika kumpulan koinnya sederhana dan mengikuti aturan tertentu.

Namun, penting untuk dicatat bahwa metode greedy tidak selalu memberikan solusi optimal untuk semua kumpulan koin. Untuk kumpulan koin yang lebih kompleks atau tidak teratur, pendekatan lain seperti dynamic programming mungkin diperlukan untuk menemukan solusi minimum.

Pembagian Tugas

Mario : Pembuatan Code, Pembuatan Laporan, Pembuatan Video Demo.

Nabil : Pembuatan Code, Pembuatan Laporan, Pembuatan Video Demo.