Pembahasan Donut DUARR

Deksripsi Singkat

Diberikan daftar toko dan tipe-tipe donut yang dijualnya beserta jumlah *chip* setiap tipe donut, cari berapa banyak kombinasi untuk mendapatkan persis T chocolate chip.

Solusi

Pada pembahasan kali ini hanya fokus pada rekursi dan dynamic programming.

Pada dasarnya, kita harus mencoba-coba setiap kombinasi pembelian Borman ★. Yang menjadi pertanyaan adalah apakah strategi yang paling efektif?.

Strategi Rekrusif

Jika kita bayangkan ini terjadi di kehidupan nyata, salah satu strategi Borman ★ yang mengunjungi setiap toko. Lalu untuk setiap toko, periksa setiap donut. Kemudian untuk setiap donut, coba beli setiap jumlah donat yang mungkin. Dalam strategi ini, kapan Borman ★ berhenti?

- Borman ★ berhenti ketika dia sudah sampai toko terakhir, tetapi belum mendapat T chip (gagal).
- Borman ★ berhenti ketika dia sudah sampai toko terakhir, tetapi sudah mendapat T chip (berhasil).
- Borman ★ berhenti ketika sudah mendapat tepat T chip (berhasil).

Kondisi dimana Borman ★ berhenti adalah base case kita. Untuk memudahkan perhitungan, semua donat yang tersedia dalam suatu hari disimpan di satu array dan direpresentasikan sebagai object Donut. Untuk lebih jelasnya dapat dilihat pada pseudocode berikut.

```
class Donut {
    int chip;
    int qty;
    .
    .
    .
}

int f(int jumlahChip, int indexDonat, Donut donut[]){
    if (indexDonat == donut.length) {
        return jumlahChip == T? 1: 0;
    }
    if(jumlahChip == T){
        return 1;
    }

    int answer = 0;
    // i == 0 tidak ambil donat
```

```
    for (int i = 0; i < donut[indexDonat].qty; i++) {
        if (jumlahChip + i*donat[indexDonat].chip <= T) {
            answer += f(jumlahChip + i*donat[indexDonat].chip,
indeksDonat + 1, donut);
        }
    }
    return answer;
}
```

Kompleksitas dari algoritma ini adalah O(J^(N*D)), J adalah jumlah suatu jenis donat, N jumlah toko, dan D jumlah jenis donat toko. Kompleksitas tersebut eksponensial, sehingga sangat tidak mungkin dapat selesai kurang dari tiga detik.

Strategi Memoization

Jika dilihat, dalam proses pemanggilan rekursif di atas, akan ada banyak *state* yang harus dikomputasi ulang. Teknik memoization adalah teknik dimana kita mencatat hasil *state* tersebut sehingga tidak perlu dikomputasi ulang.

```
int memo[][];
.
.
class Donut {
    int chip;
    int qty;
    .
    .
    .
}

int f(int jumlahChip, int indexDonat, Donut donut[]){
    if (indexDonat == donut.length) {
        return jumlahChip == T? 1: 0;
    }
    if(jumlahChip == T){
        return 1;
    }

    if (memo[jumlahChip][indexDonat] != -1) {
        return memo[jumlahChip][indexDonat];
    }

    int answer = 0;
    // i == 0 tidak ambil donat
    for (int i = 0; i < donut[indexDonat].qty; i++) {
        if (jumlahChip + i*donat[indexDonat].chip <= T) {
            answer += f(jumlahChip + i*donat[indexDonat].chip,
indeksDonat + 1, donut);
        }
    }

    memo[jumlahChip][indexDonat] = answer;
    return answer;
}
```

Pada kode di atas, belum dilakukan modulo sehingga jawaban mungkin saja overflow. Lalu bagaimana cara melakukan modulo? Apakah saat jawaban akhir sudah didapat? Tidak. Ingat kembali pelajaran Matematika Diskret 1 tentang modulo, `(a + b) mod c = ((a mod c) + (b mod c) mod c)`. Sehingga kita dapat melakukan modulo sebagai berikut.

```
int memo[][];
int mod = 1000000007;
.
.
.
class Donut {
    int chip;
    int qty;
    .
    .
    .
}

int f(int jumlahChip, int indexDonat, Donut donut[]){
    if (indexDonat == donut.length) {
        return jumlahChip == T? 1: 0;
    }
    if(jumlahChip == T){
        return 1;
    }

    if (memo[jumlahChip][indexDonat] != -1) {
        return memo[jumlahChip][indexDonat];
    }

    int answer = 0;
    // i == 0 tidak ambil donat
    for (int i = 0; i < donut[indexDonat].qty; i++) {
        if (jumlahChip + i*donat[indexDonat].chip <= T) {
            answer = ((f(jumlahChip + i*donat[indexDonat].chip,
indeksDonat + 1, donut) % mod) + answer) % mod;
        }
    }

    memo[jumlahChip][indexDonat] = (answer % mod);
    return answer;
}
```

Kompleksitas setelah dilakukan memoization menjadi O(T*J*N*D), T adalah jumlah chip yang ingin dicari. Kompleksitas tersebut cukup untuk program berjalan kurang dari 3 detik.