Memanfaatkan Modulo (%) operator untuk solve Circular Array Rotation problem di Hackerrank

wuriyan.to

Sebelum kita membahas problem yang akan kita *solve*, kita bahas sedikit apa itu **Modulo operator** dan ketentuannya.

Modulo operator adalah salah satu operator aritmatika yang **menghasilkan hasil sisa bagi X dan Y**.
Berikut beberapa ketentuan dari operasi Modulo.

X%Y

Jika X >= Y = Z => **5 % 4 = 1**, **5 % 5 = 0**, **11 % 3 = 2**

Jika Y > X = X => **4 % 5 = 4**, **3 % 5 = 3, 2 % 5 = 2, 1 % 5 = 1**

Sehingga jika kita memiliki koleksi rangkaian *list/array*, misalnya

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

kemudian kita lakukan operasi Modulo dengan **y = 5**, (**x[n] % y)**, akan menghasilkan

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
y = 5

x[n] % y = z
z = [1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4]

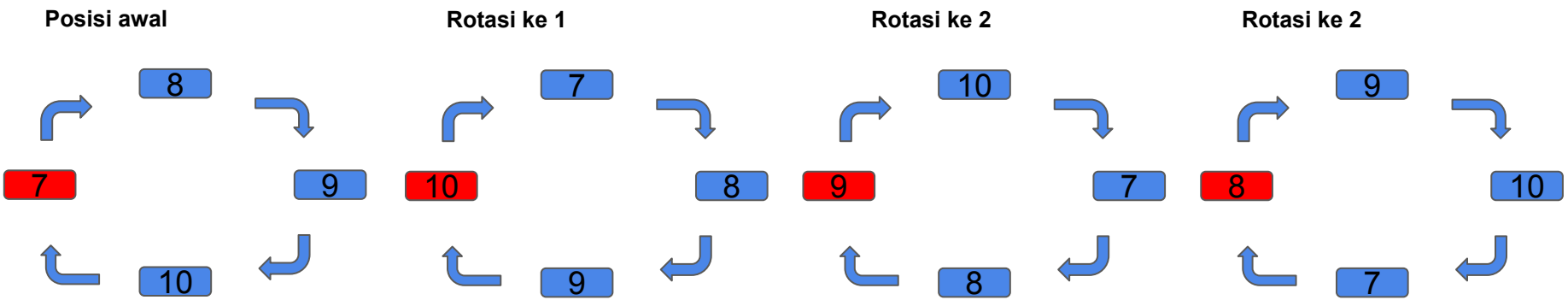
Circular Array Rotation adalah salah satu problem yang ada di **Hackerrank** yang masuk level **Easy** yang bisa kita *solve*, terutama pada sisi *performance* dengan memanfaatkan **Modulo Operator**.

Circular Array Rotation adalah problem dimana kita diminta untuk merotasi posisi array sejumlah **T** kali rotasi.

Misal,

T = 3

dengan array = [7, 8, 9, 10]



hasil akhir dengan 3 kali rotasi = [8, 9, 10, 7]

```
func solve(data []int, t int) []int {
    size := len(data)

    for i := 0; i < t; i++ {
        lastIdxData := data[size-1]
        for j := 0; j < size-1-1; j++ {
            idx := (size - 1 - 1) - j

            currData := data[idx]
            prevData := data[idx-1]

            data[idx+1] = currData
            data[idx] = prevData
        }
        data[0] = lastIdxData

    }

    return data
}
```

Untuk pendekatan awal yang akan kita implementasikan, kita tidak akan begitu mempertimbangkan sisi performa.

- loop pertama akan kita gunakan untuk counting berapa kali rotasi
- tampung value pada index terakhir
- loop kedua (inner loop) untuk iterasi tiap data, ***exclude*** data **index terakhir dan index ke 0**
- menyiapkan variabel index dari **index terakhir - 1**. Karena kita akan iterasi secara *reverse*, misal (7, 8, 9, 10) menjadi (9, 8) dst.
- **currData** kita gunakan sebagai *temporary* value untuk menyimpan data index terakhir - 1, misalnya posisi **index terakhir - 1** saat ini adalah **9**.
- **prevData** kita gunakan sebagai *temporary* value untuk menyimpan data index dari **currData - 1**, misalnya posisi **index currData - 1** saat ini adalah **8**.
- memindahkan temporary **currData** ke posisi selanjutnya. Misalnya value **currData** saat ini adalah **9**, kita pindah dari posisi **index 2 ke 3**.
- memindahkan temporary **prevData** ke posisi selanjutnya. Misalnya value **prevData** saat ini adalah **8**, kita pindah dari posisi **index 1 ke 2**.
- **data[0] = lastIdxData**, kita value dari **lastIdxData** yang saat ini menampung **value 10 ke posisi index 0**.
- Ulangi proses diatas sebanyak **t kali rotasi**.

Complexity dari pendekatan ini adalah **O(n^2)**, karena terdapat nested loop. Pendekatan ini akan sangat berat jika
constraint:
1 <= size <= 10^6 atau **1 <= size <= 10^9** misalnya.

```
func solve(data []int, t int) []int {
    size := len(data)
    res := make([]int, size)

    for i := 0; i < size; i++ {
        idx := i + t
        res[idx%size] = data[i]
    }

    return res
}
```

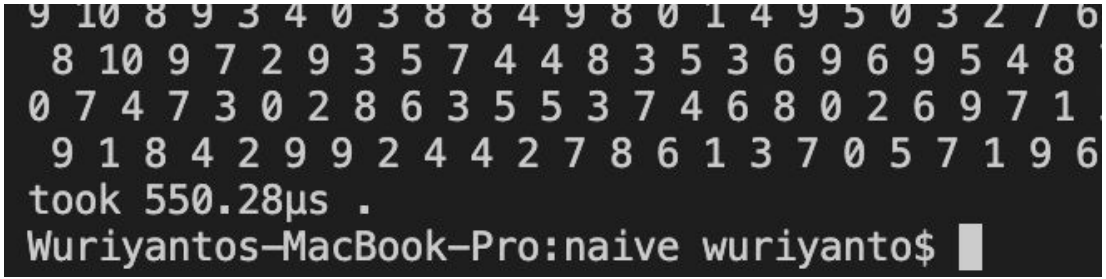
Untuk pendekatan kedua yang akan kita implementasikan, kita akan memanfaatkan Modulo operator untuk menghindari kompleksitas **O(n^2)**.

- siapkan *list/array* baru untuk menyimpan hasil akhir
- loop, untuk iterasi setiap data yang ada pada original *list/array.*
- menyiapkan variabel **idx**, **idx** adalah **hasil penjumlahan** dari **t (jumlah rotasi)** dan **index ke i**.
- Dengan memanfaatkan rule yang kita bahas sebelumnya, jika **X%Y** dan **Y > X** maka akan menghasilkan **X** => (**Y > X = X)**.
- Misalnya dengan data (7, 8, 9, 10), **t = 3**, **size = 4**, dengan **posisi i** saat ini adalah **0**, maka **idx = 0 + 3 = 3**.
- **3%4 = 3**, value data asli index ke **0 = 7** kita pindah ke **posisi 3** di list/array baru.
- pada iterasi selanjutnya **posisi i** saat ini adalah **1**, maka **idx = 1 + 3 = 4**.
- **4%4 = 0**, value data asli index ke **1 = 8** kita pindah ke **posisi 0** di list/array baru.
- Ulangi proses diatas sebanyak jumlah data yang ada.
- sehingga menghasilkan hasil akhir **data = [8, 9, 10, 7]** dengan 3 kali rotasi.

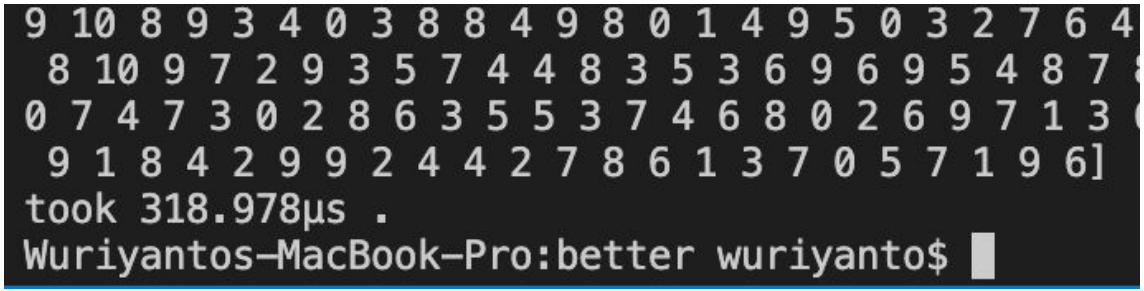
Complexity dari pendekatan ini adalah **O(n)**, karena tidak terdapat nested loop.

Percobaan implementasi pada komputer saya dengan **n = 1000 (jumlah data)**, dan **t = 200 (rotasi)**.

- **Percobaan implementasi pertama (tanpa modulo operator) menghabiskan waktu 550.20 nanoseconds**



- **Percobaan implementasi kedua (dengan modulo operator) menghabiskan waktu 318.97 nanoseconds**



Terdapat perbedaan waktu eksekusi yang cukup signifikan antara dua pendekatan yang coba kita terapkan tadi.