Pembahasan Sofita Juga Butuh *Healing*

Deskripsi Singkat

Terdapat 3 permasalahan yang harus dipecahkan mahasiswa, yakni:

- Bagaimana cara melakukan inisiasi memasukkan kue ke dalam toples dengan kompleksitas $O(1)$ untuk setiap kuenya?
- Bagaimana cara menggeser setiap elemen ke kanan dengan kompleksitas $O(1)$?
- Bagaimana cara menemukan toples (*stack*) yang kue teratasnya (*top of stack*) berisi kue yang diinginkan dengan kompleksitas $O(N)$?

Ide

Ide dasar dari permasalahan ini adalah dengan menggunakan *Deque of Stacks* atau *Queue of Stacks* sebagai *conveyor belt*-nya. Alasan penggunaan *deque* atau *queue* karena kedua ADT tersebut mendukung kompleksitas yang rendah saat terjadi perpindahan elemen dari paling kanan ke paling kiri ketika melakukan *query* GESER_KANAN. Jika menggunakan *array list*, maka elemen-elemen di sebelah kanan elemen pertama perlu digeser untuk mengalokasikan tempat pada elemen pertama. Selain itu, penggunaan kedua ADT tersebut juga dapat mempercepat proses inisiasi elemen yang selalu ditambahkan dari kiri karena elemen-elemen lain pada ADT tersebut tidak perlu digeser seperti pada *array list*.

```
static Deque<Stack<Integer>> conveyorBelt = new ArrayDeque<Stack<Integer>>();
```

Pada tahap inisiasi, mahasiswa diharapkan dapat melakukan proses inisiasi dengan kompleksitas $O(1)$ setiap menambahkan suatu elemen pada posisi terkiri. Tentunya, kompleksitas waktu ini tidak dapat dicapai menggunakan *array list* yang akan memakan waktu $O(N)$ setiap memasukkan 1 elemennya.

```
for (int i = 0; i < N; ++i) {
   Stack<Integer> toplesKeI = new Stack<Integer>();
   conveyorBelt.addFirst(toplesKeI);

   for (int j = 0; j < X; j++) {
       int rasaKeJ = in.nextInt();
       toplesKeI.push(rasaKeJ);
   }
}
```

Solusi untuk *query* BELI_RASA dibentuk berdasarkan *query* GESER_KANAN. *Query* BELI_RASA juga memanfaatkan Iterator untuk melakukan *loop* pada *conveyor belt* dari kiri ke kanan untuk mencari kue dengan rasa yang ingin dibeli. Penggunaan Iterator saat melakukan *search* pada *deque* adalah salah satu cara *loop* yang tidak mengubah urutan toples dalam *conveyor belt*. Setelah menemukan kue dengan rasa yang diinginkan, *conveyor belt* akan bergeser ke kanan menggunakan implementasi yang sama dengan query GESER_KANAN jika toples tidak berada tepat di depan Sofita. Sehingga, *query* BELI_RASA akan memiliki kompleksitas $O(N)$ jika menggunakan struktur data yang tepat.

Berikut ini adalah potongan program untuk *query* GESER_KANAN dengan kompleksitas $O(1)$.

```
static int geserKanan() {

   // Ambil toples terkanan pada Deque
   Stack<Integer> toplesTerkanan = conveyorBelt.pollLast();

   // Letakkan toples terkanan di paling kiri
   conveyorBelt.addFirst(toplesTerkanan);

   // Kembalikan kue paling atas pada toples paling kanan setelah
   // pergeseran selesai, kembalikan -1 jika toples kosong
   return toplesTerkanan.empty() ? -1 : toplesTerkanan.peek();
}
```

Berikut ini adalah potongan program untuk *query* BELI_RASA dengan kompleksitas $O(N)$.

```
static int beliRasa(int rasaDicari) {

    int conveyorBeltSize = conveyorBelt.size();

    // Buat iterator untuk loop conveyor belt, mencari kue dengan
    // rasa yang ingin dibeli
    Iterator<Stack<Integer>> iterator = conveyorBelt.iterator();
    int posisiToples = -1;
    for (int i = 0; i < conveyorBeltSize; i++) {
        Stack<Integer> toples = iterator.next();
        int rasaTeratas = toples.empty() ? -1 : toples.peek();
        if (rasaTeratas == rasaDicari) {
            posisiToples = i;
            break;
        }
    }

    // Geser kanan jika kue ditemukan dan tidak berada di depan Sofita
    if (posisiToples > 0) {
        for (int i = conveyorBeltSize - posisiToples; i > 0; i--) {
            conveyorBelt.addFirst(conveyorBelt.pollLast());
        }
    }

    // Beli kuenya
    if (posisiToples != -1) {
        conveyorBelt.getFirst().pop();
    }

    return posisiToples;
}
```

Karena terdapat C *query*, maka kompleksitas dari solusi ini adalah $O(CN)$