Pembahasan Antrean Unik

Deskripsi Singkat

Terdapat sebuah antrean yang pada awalnya telah terdapat N orang yang terurut dengan definisi terurut adalah orang yang berada di depan memiliki panjang nama yang lebih pendek, atau apabila sama maka yang leksikografis terkecil. Setelah itu diberikan M buah orang tambahan yang ingin mengantre, Anda diminta untuk mengeluarkan orang-orang yang akan dilalui untuk setiap orangnya saat mengurutkan apabila ia merupakan soal tipe 1, dan mengeluarkan hasil akhir Antrean setelah M orang tersebut masuk, apabila ia merupakan soal tipe 2.

Ide

Dapat dilihat dengan jelas bahwa soal ini meminta kita untuk melakukan pengurutan, yang berarti soal ini merupakan soal tentang algoritma sorting.

Kita dapat menyelesaikan soal tipe 2 dengan mudah dengan menggunakan **Collections.sort()** dari library collection. Tapi untuk bisa menggunakan fungsi tersebut, objek yang ingin kita urutkan harus mengimplementasikan Comparable pada class Person. Selain itu kita juga harus meng-*override* fungsi compareTo.

Fungsi compareTo adalah fungsi yang harus diimplementasikan apabila kita meng-*implements* Interface Comparable. Fungsi compareTo menerima parameter berupa Objek yang ingin dibandingkan, dalam kasus ini kita ingin membandingkan Person dengan Person. Fungsi compareTo juga harus mengembalikan nilai bertipe data *int*. Apabila objek A dibandingkan dengan objek B dan objek A lebih kecil dari objek B maka fungsi compareTo harus mengembalikan nilai negatif atau dalam kasus ini -1. Apabila objek A sama dengan objek B maka fungsi compareTo harus mengembalikan nilai 0 dan apabila objek A lebih besar dari objek B maka fungsi compareTo harus mengembalikan nilai positif atau dalam kasus ini 1.

Berikut adalah contoh pengimplementasian dari method compareTo pada class Person.

```
public int compareTo(Person other) implements Comparable<Person> {
    String otherName = other.getName();
    if (name.length() < otherName.length()) return -1;
    else if (name.length() > otherName.length()) return 1;
    else return name.compareTo(otherName);
}
```

Yang dilakukan pada kode diatas adalah, pertama melakukan komparasi panjang nama objek A dengan panjang nama objek B. Apabila panjang nama A lebih kecil dari panjang nama B maka objek A lebih kecil dari objek B, maka fungsi mengembalikan nilai negatif (-1). Apabila objek A memiliki nama yang lebih panjang dari objek B maka objek A lebih besar dari objek B dan fungsi akan mengembalikan nilai positif (+1). Apabila panjang nama pada kedua objek sama, maka dilakukan pengecekan leksikografis dengan memanggil fungsi compareTo yang dimiliki oleh string. Informasi lebih lanjut mengenai Interface Comparable dan fungsi compareTo dapat dibaca di sini.

Contoh:
Apabila dijalankan kode di bawah dengan objek A mempunyai nama “Reynaldo” dan objek B mempunyai nama “Dhipta” maka akan mengeluarkan nilai “1” karena string “Reynaldo” mempunyai panjang 8 dan “Dhipta” mempunyai panjang 6.

```
System.out.println (A.compareTo(B))
```

Setelah itu kalian harus mengimplementasikan fungsi printOutput yang menerima ArrayList Person dan mengeluarkan nama-nama orang sesuai dengan format yang diminta. Hal ini cukup trivial dengan melakukan iterasi pada ArrayList dan mengeluarkan nama dari objeknya. Berikut adalah contoh kode implementasi fungsi printOutput

```
private static void printOutput(ArrayList<Person> people) throws
IOException {
    if (people.size() == 0) {
        out.println("tidak ada");
    }
    else {
        for (int i = 0;  i < people.size(); i++) {
            if (i != 0) out.print(" ");
            out.print(people.get(i));
        }
        out.println();
    }
}
```

Setelah itu kita sekarang dapat menyelesaikan soal tipe 2 dengan mudah menggunakan fungsi sort yang dimiliki Collections.

Berikut adalah contoh implementasi menggunakan Collections.sort()

```
for(int i = 0; i < m; i++) {
    if (t == 2) {
        people.add(new Person(in.next()));
    }
}
if (t == 2) {
    Collections.sort(people);
    printOutput(people);
}
```

Apabila kalian berhasil mengimplementasikan semua ini dengan benar, maka kalian telah berhasil mendapatkan nilai 60 poin. Untuk mendapatkan nilai 40 poin sisanya, kalian harus mengimplementasikan algoritma sorting.

Algoritma yang dapat kalian gunakan untuk menyelesaikan soal tipe 1 adalah dengan menggunakan algoritma **Insertion Sort** atau **Bubble Sort**. Mungkin terdapat algoritma lain yang dapat kalian gunakan selain dua algoritma sorting yang telah ditulis di atas.

Ide utama dari soal tipe 1 ini adalah misalkan kita memiliki sebuah *array* Person berukuran N yang terurut. Dimana Person pada indeks ke-1 merupakan Person terkecil dan Person pada indeks ke N merupakan Person terbesar. Misalkan kita ingin memasukkan objek Person baru sebut saja A. Apabila A dimasukkan ke dalam array maka ukuran array sekarang adalah N + 1. Misalkan setelah kita masukkan A ke dalam array dan kita urutkan posisi A berada di indeks ke-i. Maka dapat kita simpulkan bahwa semua Person dari indeks ke i + 1 sampai N + 1 lebih besar dari Person A. Karena array terurut kita juga bisa tau bahwa Person dengan indeks ke j untuk $i + 1 \leq j \leq N$ pasti lebih keci dari Person dengan indeks ke j + 1. Oleh karena itu kita cukup mengeluarkan semua Person dari indeks N + 1 sampai i + 1. Perhatikan bahwa kita mencetak nama-nama secara terbalik karena kita saat memindahkan sebuah bilangan dari N + 1 ke i, maka kita harus melewati N terlebih dahulu baru N - 1 dan seterusnya.

Untuk mencari indeks i yang merupakan indeks dimana A berada saat *array* terurut dapat menggunakan algoritma **Insertion Sort** atau **Bubble Sort**. Caranya cukup mudah yaitu mulai dari indeks paling belakang kita cek apakah indeks tersebut lebih besar dari Person A atau tidak? Apabila indeks tersebut lebih besar dari Person A maka kita dapat mengurangi indeks (indeks - 1) karena kita tahu bahwa Person A harus berada indeks yang lebih kecil daripada objek tersebut (ingat bahwa indeks lebih kecil menunjukkan bahwa objek tersebut lebih kecil). Apabila ternyata objek pada indeks lebih kecil dibanding objek A maka kita berhenti dan meletakkan objek A tepat di sebelah kanan (indeks + 1) dari objek tersebut.

Berikut adalah contoh potongan kode menggunakan algoritma **Insertion Sort**.

```
private static ArrayList<Person> insertPeople(Person person) {
    // listPerson berisi Person yang lebih besar dari person
    ArrayList<Person> listPerson = new ArrayList<Person>();
    for (int i = people.size() - 1; i >= 0; i--) {
        // apabila person masih lebih kecil dari indeks
        if(person.compareTo(people.get(i)) < 0) {
            listPerson.add(people.get(i));
        }
        else {
            // apabila person lebih besar, tambahkan di kanan
            people.add(i + 1, person);
            break;
        }
        // kasus apabila person sekarang adalah objek paling kecil
        if(i == 0) {
            people.add(0, person);
        }
    }
    return listPerson;
}
```

Berikut adalah contoh potongan kode menggunakan algoritma **Bubble Sort.**

```
private static ArrayList<Person> insertPeople(Person person) {
    // listPerson berisi Person yang lebih besar dari person
    ArrayList<Person> listPerson = new ArrayList<Person>();
    people.add(person);
    for (int i = people.size() - 1; i > 0; i--) {
        // apabila person masih lebih kecil dari indeks
        if(people.get(i).compareTo(people.get(i - 1)) < 0) {
            listPerson.add(people.get(i));
            // pseudocode swap
            swap(people.get(i), people.get(i-1));
        }
        else {
            break;
        }
    }
    return listPerson;
}
```

Maka sekarang fungsi pada readInput() dapat dimodifikasi menjadi potongan kode berikut.

```
for(int i = 0; i < m; i++) {
    if (t == 1) {
        printOutput(insertPeople(new Person(in.next())));
    }
    if (t == 2) {
        people.add(new Person(in.next()));
    }
}

if (t == 2) {
    Collections.sort(people);
    printOutput(people);
}
```

Apabila kalian berhasil mengimplementasikan semua fungsi di atas dengan benar maka kalian akan mendapatkan nilai 100.

Analisis

Untuk soal tipe 2, maka kita melakukan fungsi sort dengan library Collection. Fungsi sort pada library collection berjalan dengan O(N log N) tetapi untuk melakukan komparasi leksikografis membutuhkan iterasi sebanyak |S| atau panjang string. Maka kompleksitas akan menjadi O(|S| * (N + M) log (N + M)).

Untuk soal tipe 1, maka kita melakukan pengurutan dengan algoritma **Bubble Sort** atau **Insertion Sort.** Untuk setiap orang pada M kita perlu melakukan iterasi paling banyak N + M kali sehingga algoritma ini akan berjalan sebanyak O(|S| * (M * (N + M)).