Pembahasan Yuk Healing Bersama Budi di FunZone!

Deskripsi Singkat

Sebelum membahas kueri satu per satu, perlu dipikirkan terlebih dahulu struktur data yang optimal untuk menyimpan mesin dan skor. Setelah itu, terdapat 5 kueri yang perlu diselesaikan untuk menjawab permasalahan pada tugas pemrograman ini.

1. **MAIN** → bagaimana cara menambahkan nilai baru dalam mesin permainan?
2. **GERAK** → bagaimana cara bergerak dari satu mesin ke mesin yang di sebelahnya, dengan konsep *wrap around*?
3. **HAPUS** → bagaimana cara menentukan X skor tertinggi dan menggerakkan suatu mesin ke paling belakang?
4. **LIHAT** → bagaimana cara mendapatkan banyaknya nilai pada suatu mesin yang memenuhi syarat *lower bound* dan *upper bound*?
5. **EVALUASI** → bagaimana cara mengurutkan mesin berdasarkan tingkat popularitasnya?

Penyimpanan Mesin

Bagian pertama yang perlu dibuat pada kode TP2 adalah mekanisme penyimpanan mesin pada FunZone. Pada solusi yang ada, terdapat *class* **Funzone** dimana *class* ini akan memiliki atribut **Mesin[]** (Array dari mesin) yang menyimpan seluruh mesin yang ada sekaligus menjadi representasi dari **Doubly Circular Linked List** dengan node-nodenya berupa objek **Mesin.** Berikut merupakan alasan penyimpanan mesin dalam bentuk **Mesin[]** dan **Doubly Circular Linked List:**

- **Mesin[]**
  Untuk kepentingan kueri **EVALUASI,** kode yang dibuat perlu mengimplementasikan algoritma *sorting* pada mesin-mesin yang ada di Funzone. Untuk memperoleh solusi terbaik dari kueri tersebut, diperlukan *sorting* menggunakan algoritma **Merge Sort** (penjelasan alasannya terdapat pada bagian pembahasan kueri EVALUASI). Dengan demikian, diperlukan mekanisme penyimpanan mesin-mesin dalam bentuk **Mesin[]**

- **Circular Doubly Linked List**
  Untuk kepentingan kueri **HAPUS,** kode yang dibuat perlu mengimplementasikan mekanisme perpindahan mesin yang rusak menjadi mesin yang paling kanan. Untuk memperoleh solusi terbaik dari kueri tersebut, diperlukan representasi **Circular Doubly Linked List** dari seluruh mesin pada Funzone (penjelasan alasannya terdapat pada bagian pembahasan kueri HAPUS)**.** Selain itu, penggunaan **Doubly Circular Linked List** juga akan memudahkan implementasi kueri **GERAK** di mesin manapun Budi berada saat ini.

Untuk membuat representasi keduanya sekaligus, berikut merupakan potongan kode yang ditulis:

```
int N = in.nextInt();
Mesin[] listMesin = new Mesin[N + 10];

for (int i = 0; i < N; i++) {
   int id = i + 1;
   int M = in.nextInt();

   ArrayList<Integer> scores = new ArrayList<>();
   for (int j = 0; j < M; j++) {
       int score = in.nextInt();
       scores.add(score);
   }

   Mesin mesin = new Mesin(id, scores);
   listMesin[i] = mesin;
}

FunZone funZone = new FunZone(N, listMesin);
```

Pada potongan kode di atas terlihat bahwa nilai N akan dijadikan argumen dalam inisiasi objek Funzone baru, dengan N adalah jumlah mesin dalam objek funzone. Dalam *looping,* setiap mesin akan dimasukkan ke dalam Array listMesin.

Setelah semua **Mesin** masuk ke dalam Array, maka dibentuklah *object* FunZone. Pada tahap inisiasi, Budi akan ditempatkan pada mesin paling kiri pada daftar mesin (mesin dengan index 0 pada Array mesin) dan method createLinkedList() dari FunZone akan dijalankan. Seluruh mesin pada Array mesin akan dibuatkan representasi **Circular Doubly Linked List-**nya.

```
class FunZone {
   int jumlahMesin;
   Mesin Budi, first, last;
   Mesin[] mesin;

   public FunZone(int jumlahMesin, Mesin[] mesin) {
       this.jumlahMesin = jumlahMesin;
       this.mesin = mesin;
       Budi = mesin[0];
       createLinkedList();
   }

   void createLinkedList() {
       for (int i = 0; i < jumlahMesin; i++) {
           if (i == jumlahMesin - 1) {
```

```
                mesin[i].right = mesin[0];
                last = mesin[i];
            } else {
                mesin[i].right = mesin[i + 1];
            }

            if (i == 0) {
                mesin[i].left = mesin[jumlahMesin - 1];
                first = mesin[i];
            } else {
                mesin[i].left = mesin[i - 1];
            }
        }
    }
}
```

Mekanisme penyimpanan skor dari setiap mesin akan dijelaskan pada bagian selanjutnya.

Penyimpanan Skor

Class Funzone memiliki atribut mesin yang merupakan array berisi objek-objek dari class Mesin. Pada class Mesin itu sendiri terdapat atribut

- bst → **Binary Search Tree** yang menyimpan nilai-nilai skor yang ada pada mesin
- id → nomor identitas mesin
- left → mesin yang posisinya tepat di sebelah kiri
- right → mesin yang posisinya tepat di sebelah kanan

Skor dalam mesin akan di-*insert* ke BST dalam tahap inisiasi.

```
public Mesin(int id, List<Integer> scores) {
    this.id = id;
    bst.root = null;
    for (int x : scores) {
        bst.root = bst.insert(bst.root, x);
    }

    left = null;
    right = null;
}
```

Perlu diperhatikan bahwa Binary Search Tree pada solusi ini terdiri atas *object-object* Node yang nilai skornya **unik**. Setiap skor yang bernilai sama hanya akan membentuk satu buah Node. Pada setiap *object* Node, terdapat atribut count yang menyatakan banyak skor yang memiliki nilai tersebut.

```
class Node {
   Node left, right;
   int score, height;
   int count; // banyaknya score bernilai 'score'
   int nodesCount; // banyaknya node pada subtree
}
```

Kueri MAIN

Inti dari kueri MAIN adalah kita menambahkan nilai baru ke dalam mesin. Jika struktur data penyimpanan mesin dan skor sudah sesuai dengan pembahasan di atas, maka hal yang perlu kita lakukan untuk menyelesaikan kueri ini hanya meng-*insert* nilai baru tersebut ke bst.

Selanjutnya, Kita bisa mendapatkan urutan Y pada barisan skor dengan menghitung

$$banyakNilaiDiMesin \ - \ banyakNilaiDiMesinYangKurangDariSamaDenganY \ + \ 1$$

a. Banyaknya nilai yang ada di mesin dapat dihitung dari nodesCount yang dimiliki root.
b. Banyaknya nilai yang kurang dari sama dengan Y dapat dihitung dengan menggunakan fungsi countLessThan() dengan memasukkan root sebagai parameter dan nilai $Y \ + \ 1$ untuk parameter scoreLimit.
c. Skor Y yang baru saja ditambahkan pada kueri ini akan ikut terhitung pada poin b, sehingga di akhir kita perlu menambahkan 1 ke hasil hitungan.

```
int main(int y) {
   bst.root = bst.insert(bst.root, y);
   return bst.root.nodesCount - bst.countLessThan(bst.root, y + 1) + 1;
}
```

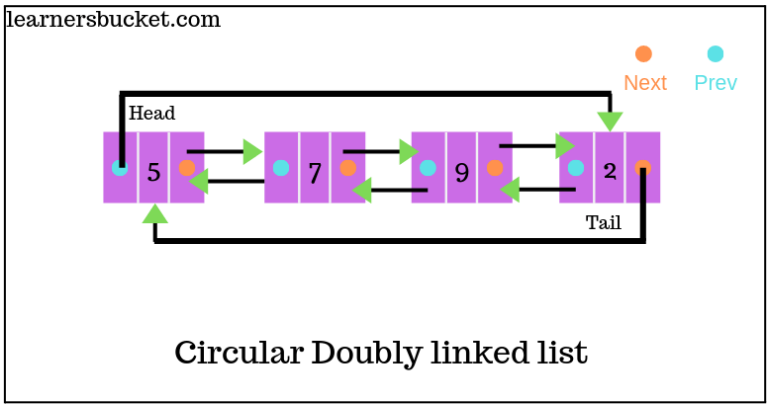
```
int countLessThan(Node root, int scoreLimit) {
   if (root == null)
       return 0;

   int ans = 0;
   if (root.score < scoreLimit) {
       ans += root.count + (root.left != null ? root.left.nodesCount : 0);
       ans += countLessThan(root.right, scoreLimit);
   } else {
       ans = countLessThan(root.left, scoreLimit);
   }
   return ans;
}
```

Kueri GERAK

Pada kueri **GERAK**, Budi akan berpindah ke mesin yang ada di sebelahnya, bisa ke sebelah **kiri** dengan input kueri **GERAK KIRI**, bisa juga ke sebelah **kanan** dengan input kueri **GERAK KANAN**. Untuk mengimplementasikan kueri ini, perlu dibuat pointer **Budi** yang menunjuk ke suatu **Mesin**. Pada awalnya **Budi** akan menunjuk mesin paling kiri, lalu kemudian akan berpindah sesuai dengan kueri yang berjalan. Kueri yang akan mempengaruhi mesin yang ditunjuk Budi adalah kueri **GERAK** dan **HAPUS**. Implementasi dari kueri GERAK akan sangat sederhana jika setiap mesin disimpan dalam representasi **Linked List**. Secara sederhana, implementasi dari kueri GERAK adalah sebagai berikut:

```
int gerak(String arah) {
    if (arah.equals("KIRI")) {
        Budi = Budi.left;
    } else {
        Budi = Budi.right;
    }
    return Budi.id;
}
```



Pada kode solusi, setiap mesin disimpan dalam representasi **Circular Doubly Linked List.** Mesin paling kanan akan langsung terhubung dengan Mesin paling kiri sebagai mesin yang berada di kanannya. Selain itu, Mesin paling kiri juga akan menyimpan Mesin paling kanan sebagai mesin yang berada di kirinya. Dengan demikian, tidak diperlukan lagi mekanisme *handling* untuk *edge case* ketika Budi melakukan GERAK KANAN pada mesin paling kanan maupun GERAK KIRI pada mesin paling kiri.

Kueri HAPUS

Pada kueri HAPUS, Budi ingin menghapus X skor teratas pada mesin yang ada di depannya saat ini. Asumsikan M menyatakan banyaknya skor dari mesin permainan terkait. Jika $M \leq X$, maka secara berurutan yang akan terjadi adalah:

1. Mesin dinyatakan rusak dan seluruh skor tercatat akan dihapus.
2. Budi dan mesin akan berpindah tempat dengan ketentuan sebagai berikut:

- Jika mesin permainan yang rusak merupakan mesin permainan paling kanan, maka mesin tersebut tidak perlu dipindahkan kemana-mana dan Budi akan berpindah ke mesin permainan paling kiri.
- Jika hanya terdapat satu mesin permainan pada FunZone, maka Budi akan tetap berdiri di depan mesin tersebut.
- Selain kedua kasus di atas, maka Budi akan berpindah ke mesin permainan yang berada di sebelah kanannya, lalu mesin akan langsung dipindahkan menjadi mesin permainan paling kanan tanpa menunggu EVALUASI dari pihak FunZone.

3. Mesin yang rusak diperbaiki sehingga dapat digunakan kembali pada aktivitas selanjutnya.

Kita dapat membagi implementasi kueri ini menjadi dua kasus :

- Jika $M \leq X$, maka langkah yang harus dilakukan yaitu :
  1. Pindahkan pointer Budi ke mesin sebelah kanan
  2. Pindahkan mesin yang rusak tadi ke posisi paling kanan linked list
  3. *Reset* BST pada mesin rusak tersebut
- Jika $M > X$, maka langkah yang harus dilakukan yaitu :
  1. Dapatkan skor terbesar pada BST
  2. Hapus skor terbesar tersebut dari BST
  3. Ulangi langkah 1 dan 2 sebanyak X kali

```
class Funzone {
   long hapus(int x) {
       if (Budi.bst.root.nodesCount <= x) {
           Mesin kiri = Budi.left;
           Mesin rusak = Budi;
           Mesin kanan = Budi.right;

           Budi = Budi.right;
           long ans = rusak.hapus(x);
           rusak.bst.root = null;

           if (rusak != last) {
               if (rusak == first) {
                   first = kanan;
               } else {
                   kiri.right = kanan;
                   kanan.left = kiri;

                   last.right = rusak;
                   rusak.left = last;
                   first.left = rusak;
                   rusak.right = first;
               }
               last = rusak;
```

```
            }
            return ans;
        } else {
            return Budi.hapus(x);
        }
    }
}
```

```
class Mesin {
    Long hapus(int x) {
        Long ans = 0;
        for (int i = 0; i < x; i++) {
            if (bst.root == null)
                break;
            Node max = bst.maxValueNode(bst.root);
            bst.root = bst.deleteNode(bst.root, max.key);
            ans += max.key;
        }
        return ans;
    }
}
```

Kueri LIHAT

Pada kueri LIHAT, Budi ingin mengetahui banyaknya skor yang berada dalam rentang L (inklusif) dan H (inklusif). Ide untuk mendapatkan banyaknya skor dalam rentang L dan H adalah dengan mencari banyaknya skor yang kurang dari $H + 1$ dan mencari banyaknya skor yang kurang dari L. Kemudian, jika kita mengurangkan keduanya, maka kita akan medapatkan jumlah skor dalam rentang L dan H

$$ans = countLessThan(H + 1) - countLessThan(L)$$

Jika kita memodelkan skor untuk disimpan dalam suatu BST, maka kompleksitas fungsi *countLessThan()* adalah $O(log\ N)$, dengan N adalah banyaknya skor dalam suatu mesin.

Implementasi fungci countLessThan() sendiri sudah dijelaskan pada kueri MAIN.

Kueri EVALUASI

Pada kueri EVALUASI, pihak FunZone akan mengubah urutan barisan mesin berdasarkan tingkat popularitasnya. Untuk mengimplementasikan kueri ini, perlu disimpan banyaknya skor yang ada pada

suatu mesin untuk mengetahui tingkat popularitas mesin tersebut. Hal ini sudah dijelasikan pada penjelasan sebelumnya.

Implementasi dari kueri EVALUASI dapat menggunakan algoritma *sorting* dengan kompleksitas $O(N\ log\ N)$. Salah satunya yaitu algoritma *merge sort*. Setelah dilakukan *sorting*, linked list perlu dibentuk kembali berdasarkan posisi-posisi mesin yang baru, sehingga createLinkedList() kembali dipanggil.

```
int evaluasi() {
    mergeRec(0, cnt - 1);
    createLinkedList();

    for (int i = 0; i < jumlahMesin; i++) {
        if (Budi.id == mesin[i].id)
            return i + 1;
    }
    return -1;
}
```

Berikut ini kode untuk algoritma *merge sort*.

```
public static void mergeRec(int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeRec(left, mid);
        mergeRec(mid + 1, right);
        merge(left, right);
    }
}
```

Method `merge(left, right)` merupakan method untuk melakukan *sorting* pada mesin-mesin dari index left sampai right dengan menggabungkan mesin-mesin yang sudah terurut dari index left sampai mid dan mid + 1 sampai right.

```
public static void merge(int left, int right) {
    int mid = (left + right) / 2;
    int jumlahMesin = right - left + 1;
    Mesin[] sortedMesin = new Mesin[jumlahMesin];

    int leftPointer = left;
    int rightPointer = mid + 1;
    int sortedPointer = 0;

    while (leftPointer <= mid && rightPointer <= right) {
        if (mesin[leftPointer].compareTo(mesin[rightPointer]) > 0) {
```

```
            sortedMesin[sortedPointer] = mesin[leftPointer];
            leftPointer++;
        } else {
            sortedMesin[sortedPointer] = mesin[rightPointer];
            rightPointer++;
        }
        sortedPointer++;
    }

    while (leftPointer <= mid) {
        sortedMesin[sortedPointer] = mesin[leftPointer];
        leftPointer++;
        sortedPointer++;
    }

    while (rightPointer <= right) {
        sortedMesin[sortedPointer] = mesin[rightPointer];
        rightPointer++;
        sortedPointer++;
    }

    for (int i = 0; i < jumlahMesin; i++) {
        mesin[i + left] = sortedMesin[i];
    }
}
```

Sedangkan kode untuk createLinkedList() sudah dijelaskan pada bagian sebelumnya.

Nilai Bonus

Seperti yang tertera pada soal, nilai bonus dapat diperoleh jika mengimplementasikan AVL Tree. Sehingga, kalian cukup mengubah implementasi Binary Search Tree pada solusi kalian menjadi menggunakan AVL Tree.