Pembahasan Lab 1

Deksripsi Singkat

Terdapat Nterminal yang ditandai dengan karakter "*" (bintang) dan beberapa toko (minimal 1) di antara setiap perjalanan. Reiva melakukan *one-way trip* sambil berjualan di toko-toko yang ia lewati. X_i adalah pendapatan yang akan diperoleh jika Reiva menjual di toko ke-i dan total pendapatannya adalah penjumlahan pendapatan dari setiap toko yang mana ia menjual barangnya. Berapa total pendapatan tertinggi yang dapat diperoleh jika ia memilih terminal mulai dan terminal selesai secara optimal?

Ide

1. Jumlahkan semua angka di antara setiap dua bintang terdekat dan simpan hasilnya di dalam sebuah List. Dapat dilakukan dengan *looping* biasa.
   a. Misalkan, *sequence*-nya ialah:
      `sequence = * 1 135 -200 * -20 50 * 130 * 3 -3 3 3 -3 -3 -3 *`
   b. Maka List yang dihasilkan:
      `sumSequence = [-64, 30, 130, -3]`
2. Cari nilai maksimal dari List sumSequence untuk menjadi *initial value* dari maxSum pada ***Maximum Contiguous Subsequence Sum***
3. Lalu dapat dilakukan ***Maximum Contiguous Subsequence Sum*** dengan kompleksitas $O(N)$.

Berikut ini adalah potongan program untuk mendapatkan List sumSequence dan nilai maksimumnya dengan kompleksitas $O(M)$.

```
public static int getMaxMoney(int N, int M, List<String> sequence) {
    List<Integer> sumSequence = new ArrayList<>();
    int tempSum = 0;
    int maxSum = Integer.MIN_VALUE;

    for (int i = 1; i < sequence.size(); i++) {
        String temp = sequence.get(i);
        if (temp.equals("*")) {
            sumSequence.add(tempSum);
            maxSum = Math.max(maxSum, tempSum);
            tempSum = 0;
        } else {
            tempSum += Integer.parseInt(temp);
        }
    }
    return maxContiguousSubsequenceSum(maxSum, sumSequence);
}
```

Berikut ini adalah potongan program untuk ***Maximum Contiguous Subsequence Sum*** dengan kompleksitas $O(N)$

```
public static int maxContiguousSubsequenceSum(int maxSum,
List<Integer> sumSequence) {

    int curSum = 0;
    for (int i = 0; i < sumSequence.size(); i++) {
        curSum += sumSequence.get(i);
        if (curSum > maxSum) {
            maxSum = curSum;
            maxSum = curSum;
        } else if (curSum < 0) {
            curSum = 0;
        }
    }
    return maxSum;
}
```

Nilai `maxSum` yang dikembalikan adalah jawaban dari soal ini.

Sehingga kompleksitas program ini adalah $O(M) + O(N) = O(M + N)$