Menggali Berlian

Deksripsi Singkat

Tiko ingin menggali berlian selama N hari. Ketentuan yang harus diperhatikan berupa Tiko hanya bisa menggali selama satu sesi pada suatu hari, sesi siang atau sesi malam. Selain itu, Tiko dilarang menggali pada sesi yang sama selama dua hari berturut-turut. Perlu diperhatikan bahwa siang-bolos-siang atau malam-bolos-malam diperbolehkan. Tentukan jumlah berlian terbanyak yang bisa Tiko dapatkan dan perlu berapa kali penggalian untuk mendapatkannya (Jika terdapat lebih dari satu cara, keluarkan jumlah penggalian yang paling sedikit)

Ide

Pertama, untuk kasus S_i = 1 dan M_i = 1. Karena hadiah untuk sesi siang ataupun malam sama, maka yang harus kita perhatikan adalah nilai bonusnya, dengan begitu kasus ini dapat diselesaikan dengan hanya mencari nilai Bonus tertinggi pada suatu hari dan ditambah dengan total hari yang digunakan untuk menggali. Salah satu implementasinya sebagai berikut :

```
static private long getAns() {

    int berlian = 0;
    int banyakGalian = 0;

    // Mencoba menggali sebanyak i kali
    for (int i=1;i<=N;i++) {
        int curBerlian = i + bonus.get(i-1);
        if (berlian < curBerlian) {
            berlian = curBerlian;
            banyakGalian = i;
        }
    }

    return new ans(berlian,banyakGalian);
}
```

Namun hal tersebut hanya dapat mengcover 40% dari test-case yang ada. Untuk mendapatkan *full score* dalam permasalahan ini kita dapat menyelesaikannya dengan menggunakan teknik rekursi. Terdapat beragam variasi rekursi yang bisa digunakan untuk menyelesaikan permasalahan ini, pada pembahasan kali ini ada 3 hal yang akan kita perhatikan pada teknik rekursi kita yaitu :

- **Posisi saat ini**
- **Galian yang masih bisa digunakan**
- **Sesi yang diambil sebelumnya**

Idenya adalah berapa maksimal berlian yang bisa didapat apabila kita menggali sebanyak *g*, dengan *g* adalah bilangan bulat yang menjadi nilai awal dari parameter gali. Sehingga base case dari rekursi ini

adalah apabila jumlah gali yang harus digunakan habis sebelum mengecek setiap hari yang ada dan apabila kita telah mengecek setiap hari. Kurang lebih potongan codenya seperti berikut :

```
// lst = 0: bolos, lst = 1: siang, lst = 2: malam
private static long rekurs(int pos, int ambil, int lst) {

    // base case
    if(ambil < 0) {
        return -1000000000000000000L;
    }

    // basecase, harus exactly ngambil sebanyak "ambil"
    if(pos == B.size()) {
        if(ambil == 0) return 0;
        else return -1000000000000000000L;
    }
```

Selanjutnya adalah bagaimana kondisi rekurens untuk pindah ke state lainnya. Terdapat 3 kondisi yang bisa dilakukan untuk pindah ke state selanjutnya yaitu :

- **Case 1** : apabila pada sesi sebelumnya kita tidak mengambil siang, maka akan kita coba untuk menggali pada sesi siang saat ini.
- **Case 2** : apabila pada sesi sebelumnya kita tidak mengambil malam, maka akan kita coba untuk menggali pada sesi malam saat ini.
- **Case 3** : karena tidak ada ketentuan untuk bolos maka kita akan coba untuk bolos saat ini.

```
long ans = Long.-1000000000000000000L;

// case 1: ambil siang
if(lst != 1) {
    ans = Math.max(ans, rekurs(pos+1, ambil-1, 1) + B.get(pos));
}

// case 2: ambil malam
if(lst != 2) {
    ans = Math.max(ans, rekurs(pos+1, ambil-1, 2) + S.get(pos));
}

// case 3: bolos
ans = Math.max(ans, rekurs(pos+1, ambil, 0));

return ans;
```

Dengan demikian apabila kita memanggil fungsi *getMaxDiamond(0,g,0*) kita akan mendapatkan nilai berupa jumlah maksimum berlian yang bisa didapat apabila kita menggali sebanyak g.

Namun rekursi saja tidak cukup karena akan menghabiskan waktu yang sangat banyak dan membuat program kita TLE. *Worst-case* dari pemanggilan seluruh *state* rekursi adalah $O(N^3)$ atau 10^9 operasi hanya untuk satu kali cara dengan banyak penggalian tertentu. Padahal kita perlu mencari tiap kemungkinan banyak menggali sebanyak N kali sehingga total kompleksitasnya adalah $10^9 * 10^3 = 10^{12}$ operasi.

Salah satu cara menanganinya adalah dengan menggunakan konsep Dynamic Programming dimana nilai dari suatu state rekursi disimpan dalam sebuah array. Sehingga kita hanya perlu membuat array tiga dimensi yang berukuran 1000 * 1000 * 3 dan menyimpan nilai suatu state setiap state tersebut diakses. Setelah suatu state memiliki nilai maka kita tidak perlu melakukan rekursi kembali dan hanya perlu memanggil dari isi array dengan state tersebut.

```
if(hascal[pos][ambil][lst]) {
    return dp[pos][ambil][lst];
}
```

Dengan begitu kita dapat melakukan optimisasi fungsi rekursi sebelumnya menjadi suatu fungsi Dynamic Programming untuk mengatasi permasalahan ini, berikut potongan kode fungsi Dynamic

Programming penyelesaian keseluruhan :

```
static long dp[][][];
static boolean hascal[][][];

// lst = 0: bolos, lst = 1: siang, lst = 2: malam
private static long rekurs(int pos, int ambil, int lst) {

    // base case
    if(ambil < 0) {
        return -1000000000000000000L;
    }

    // basecase, harus exactly ngambil sebanyak "ambil"
    if(pos == B.size()) {
        if(ambil == 0) return 0;
        else return -1000000000000000000L;
    }

    if(hascal[pos][ambil][lst]) {
        return dp[pos][ambil][lst];
    }

    long ans = Long.-1000000000000000000L;

    // case 1: ambil siang
    if(lst != 1) {
        ans = Math.max(ans, rekurs(pos+1, ambil-1, 1) + B.get(pos));
    }

    // case 2: ambil malam
    if(lst != 2) {
        ans = Math.max(ans, rekurs(pos+1, ambil-1, 2) + S.get(pos));
    }

    // case 3: bolos
    ans = Math.max(ans, rekurs(pos+1, ambil, 0));

    hascal[pos][ambil][lst] = true;
    dp[pos][ambil][lst] = ans;

    return ans;
```

Setelah memiliki rekursi untuk mencari jumlah maksimal berlian yang bisa didapat Tiko, kita harus mencari berapa jumlah galian paling sedikit untuk bisa mendapatkan jumlah berlian maksimal tersebut. Hal itu bisa diselesaikan dengan membandingkan hasil berlian yang didapat secara *bottom-up* apabila tiko menggali sebanyak 1, 2, 3, … , N.

```java
public static res getMaxDiamond() {

    int N = B.size();

    dp = new long[N+1][N+1][3];
    hascal = new boolean[N+1][N+1][3];

    long mx = 0;
    int num = 0;

    for(int temp=1;temp<=N;temp++) {
        long berlian = rekurs(0, temp, 0) + Bonus.get(temp-1);

        if(berlian > mx) {
            mx = berlian;
            num = temp;
        }
    }



    return new res(mx, num);
}
```

Dengan demikian untuk mendapatkan output akhir, kita dapat memanggil :

```java
res r = getMaxDiamond();
out.println(r.value + " " + r.num);
```

Dengan res adalah sebuah class yang berisi :

```java
static class res {
    long value;
    int num;

    public res(long _value, int _num) {
        value = _value;
        num = _num;
    }
}
```

Untuk kode lengkapnya bisa dilihat di : https://pastebin.com/FQs2rBbZ