Pembahasan Perang Benteng

Deskripsi Singkat

Kerajaan Fortdom terdiri dari N benteng bernomor 1 hingga N dan E jalan **satu arah** yang menghubungkan antar benteng. Terdapat M benteng yang sedang diserang, dibutuhkan minimal 1 bala bantuan untuk menyelamatkan benteng-benteng tersebut. Pada jalan ke-i yang menghubungkan antara 2 benteng, terdapat W_i musuh yang menjaga serta akan menggugurkan sebanyak W_i bala bantuan yang mencoba melewatinya.

- **[S] [K]**
  Query tersebut akan mengirim sebanyak K bala bantuan dari benteng S, dimana dibutuhkan **minimal 1** bala bantuan untuk menyelamatkan sebuah benteng yang sedang diserang. Raja Fortdom ingin mengetahui jika terdapat rute yang dapat diambil untuk menyelamatkan sebuah benteng yang sedang diserang. (YES jika ada rute yang menyelamatkan sebuah benteng diserang, NO jika tidak ada.)

Query [S] [K]

Untuk setiap *query*, kita perlu mencari *shortest path*. Dalam menyelesaikan permasalahan ini, terdapat setidaknya 2 cara:

1. ***Naive Solution***
   Solusi *naive* dari permasalahan ini adalah kita mencari *shortest path* menggunakan algoritma dijkstra dari node S ke benteng-benteng yang diserang untuk setiap *query*. Namun, apabila menggunakan cara seperti ini, kita akan mengalami TLE karena kita akan mengulangi pemanggilan algoritma dijkstra sebanyak 1 juta kali (Q = 1.000.000). Masing-masing pemanggilan dijkstra memerlukan $O(E\ log\ (N))$, yakni sekitar $250\,000 \cdot log(5000) = 3\,250\,000$ buah iterasi. Dengan kata lain, secara total, cara ini membutuhkan sekitar $1\,000\,000 \cdot 3\,250\,000 = 325 \cdot 10^{10}$ kali iterasi.

2. ***Better Solution***
   Apabila kita memperhatikan *constraint* (batasan) pada jumlah 'benteng-yang-diserang', banyaknya tidak lebih dari 50 buah benteng. Oleh karena itu, kita cukup menerapkan dijkstra **mulai dari setiap benteng yang sedang diserang**. Dengan cara ini, kita hanya perlu melakukan paling banyak 50 kali pemanggilan dijkstra.

   Untuk mengimplementasi cara ini, kita perlu membalik arah dari setiap *edge* agar kita dapat menerapkan algoritma dijkstra **yang berawal dari benteng yang diserang** menuju ke seluruh benteng lainnya. Setelah itu kita perlu melakukan *pre-computation* untuk menghitung *shortest*

path dari masing-masing benteng yang diserang menuju benteng-benteng lainnya. Hasil *pre-computation* tersebut kita simpan untuk dipakai setiap terjadi pemanggilan *query*.

Pada setiap pemanggilan *query*, kita akan melakukan *for-loop* untuk mendapatkan *shortest path* yang disimpan oleh masing-masing 'benteng-yang-diserang'. Apabila terdapat setidaknya satu 'benteng-yang-diserang' yang **mempunyai "shortest" path** (hasil dari *pre-computation*) menuju benteng S kurang dari K (banyaknya bantuan yang dikirim), maka langsung keluarkan "YES" dan *break* iterasi. Jika tidak ada, maka keluarkan "NO".

Solusi ini membutuhkan sekitar $M \cdot E\,log(N) = 50 \cdot 250\,000 \cdot \log(5000) = 162\,500\,000$ kali iterasi untuk melakukan *pre-computation* dan $M \cdot Q = 50 \cdot 1\,000\,000 = 5 * 10^7$ kali iterasi untuk menangani seluruh *query*.

3. ***Additional Optimization***
   Sebagai tambahan, disarankan juga memanfaatkan optimisasi tambahan, yakni memanfaatkan *native java array*. Penggunaan beberapa struktur data java lainnya seperti ArrayList, PriorityQueue, dsb dapat mempengaruhi performa secara signifikan.

Triggering Question

Tentu masih banyak solusi yang jauh lebih efisien dibandingkan solusi yang telah disampaikan pada solusi nomor 2. Misal, apabila banyaknya benteng yang diserang bisa mencapai 5000 benteng, bagaimana caramu menyelesaikannya? 😀

Contoh Kode Solusi

Inisiasi awal

```
int jumlahBenteng = input.nextInt();
int jumlahBentengDiserang = input.nextInt();

ArrayList<ArrayList<Distance>> tempData = new ArrayList<>();
for (int i = 0; i < jumlahBenteng; i++) {
   tempData.add(new ArrayList<>());
}
```

Simpan daftar benteng yang diserang serta daftar jalan

```
int[] daftarBentengDiserang = new int[jumlahBentengDiserang];
for (int i = 0; i < jumlahBentengDiserang; i++) {
   daftarBentengDiserang[i] = input.nextInt() - 1;   // menggunakan 0 indexing
}

long jumlahJalan = input.nextInt();
```

```
for (int i = 0; i < jumlahJalan; i++) {
   // tukar antara destination dan source karena kita akan
   // melakukan Dijkstra dari benteng yang DISERANG
   int destination = input.nextInt() - 1;   // menggunakan 0 indexing
   int source = input.nextInt() - 1;   // menggunakan 0 indexing
   int weight = input.nextInt();
   tempData.get(source).add(new Distance(destination, weight));
}
```

Ubah penggunaan ArrayList menjadi array biasa

```
int[][] neighbours = new int[jumlahBenteng + 2][];
long[][] weights = new long[jumlahBenteng + 2][];

for (int i = 0; i < jumlahBenteng; i++) {
   ArrayList<Distance> currData = tempData.get(i);

   int size = currData.size();
   neighbours[i] = new int[size];
   weights[i] = new long[size];

   for (int j = 0; j < size; j++) {
       Distance distance = currData.get(j);
       neighbours[i][j] = distance.bentengTujuan;
       weights[i][j] = distance.jumlahMusuh;
   }
}
```

Lakukan Dijkstra dari setiap benteng yang diserang

```
ArrayList<ArrayList<Long>> shortestPath = new ArrayList<>();
for (int i = 0; i < jumlahBentengDiserang; i++) {

   ArrayList<Long> currShortestPath = new ArrayList<>();
   for (int j = 0; j < jumlahBenteng; j++) {
       currShortestPath.add((long) 1e18);
   }

   MinHeap pq = new MinHeap();
   pq.add(new Distance(daftarBentengDiserang[i], 0));
   while (pq.size > 0) {
       Distance current = pq.poll();
```

```
        int benteng = current.bentengTujuan;
        long jumlahMusuh = current.jumlahMusuh;

        if (currShortestPath.get(benteng) <= jumlahMusuh)
            continue;

        currShortestPath.set(benteng, jumlahMusuh);

        int[] currNeighbours = neighbours[benteng];
        long[] currWeights = weights[benteng];

        for (int j = 0; j < currWeights.length; j++) {
            if (currShortestPath.get(currNeighbours[j]) > currWeights[j] + jumlahMusuh) {
                pq.add(new Distance(currNeighbours[j], currWeights[j] + jumlahMusuh));
            }
        }
    }

    shortestPath.add(currShortestPath);
}
```

Selanjutnya, lakukan pengecekan seperti berikut untuk setiap *query*

```
int source = input.nextInt() - 1;   // menggunakan 0 indexing
long power = input.nextInt();

for (int i = 0; i < shortestPath.size(); i++) {
    if (shortestPath.get(i).get(source) < power) {
        output.println("YES");
        return;
    }
}
output.println("NO");
```