Nama : Ali Irsyaad Nursyaban
NPM : 1806141132
Kelas : SDA C

PENJELASAN IDE TP-4 SDA

Pada TP4 kali ini struktur data graf yang dipergunakan adalah multigraf dengan weight (jarak dari kota ke kota lain). Setiap vertex (kota) pada graf menyimpan adjList dan edge yang terhubung pada kota tersebut. Saya mengimplementasikannya dengan struktur data HashMap dimana key dari hashmap tersebut adalah adjacentnya (kota tetangga) dan valuenya adalah Object Edge yang saya buat. Class Edge yang saya implementasikan menyimpan int adjacent (kota), int jarak terpendek dan int jumlah jalan yang terhubung.

Pada method addEdge pada graf saya membuat logika, jika belum pernah ada jalan yang terhubung antara kota asal dan kota tujuan maka membuat object edge dengan int adjacent = kota tujuan, int jarakMin = jarak yang diinputkan, dan int jumlahJalan = 1 karena baru ada satu jalan yang terhubung. Dan jika sudah pernah ada jalan yang terhubung maka kita update atribut pada edge tersebut yaitu int jarakMin kita ambil jarak paling minimal antara jarak inputan dan jarakMin sebelumnya, dan kita increase int jumlahJalannya. Alasan saya menyimpan int jarakMin, dan int jumlahJalan saja dikarenakan untuk setiap perintah tidak memerlukan jarak lainnya selain jarakMinimalnya.

PERINTAH

Saya membagi 3 bagian perintah sesuai jenis graf yang menjadi minimal requirement untuk beberapa perintah tersebut.

1. Unweighted Single Graph
   OS dan LAOR hanya memperdulikan ada tidaknya jalan yang terhubung antara satu kota dan kota lain (single edge) dan tidak memperdulikan weight pada setiap edge. Saya mempergunakan algoritma BFS (Breath First Search) dengan struktur data untuk menyimpan seen (visited city) adalah HashSet. Untuk OS BFS dimulai dari berbagai kota sehingga pada kita memasukkan kota-kota tersebut pada queue dan seen, dan hasilnya kita tinggal melihat size dari seen. Untuk LAOR menggunakan BFS biasa tetapi untuk setiap kota kita menyimpan setiap distance (jumlah path) terkecil dari kota start pada sebuah array of int yang indexnya mewakili kota, dan hasilnya kita tinggal return distance dengan index kota tujuan.

2. Weighted Single Graph
   CCWG, SR, dan SM hanya memperdulikan jarak terpendek (weight) yang menghubungkan dari kota ke kota lain dan tidak memperdulikan jarak lain / jumlah jarak. Untuk CCWG menggunakan algoritma BFS tetapi dengan tambahan syarat saat memasukkan pada queue saat looping adjacent yaitu selain belum visited (ada pada HashSet seen) juga memiliki jarak terpendek maksimal sesuai jarak inputan, dan hasilnya sama seperti OS kita tinggal melihat size dari seen. Untuk SR dan SM menggunakan algoritma Dijkstra dengan array of distance yang menyimpan jarak (weight) minimum

dari kota asal. Kita akan mengupdate distance tersebut sampai benar-benar tidak ada distance yang lebih kecil darinya. Awal-awal kita inisiasi distance dengan Integer.MAX_VALUE sehingga dapat terupdate saat awal. Untuk SR hasilnya kita tinggal melihat distance dengan index dari kota tujuan (yang diinginkan). Untuk SM kita tinggal melihat distance-distance terpanjang dari distance-distance kota yang bersesuaian dan mencari distance yang terpendek dari di distance-distance yang terpanjang tersebut. Hal ini dikarenakan pada SM kita harus bertemu di suatu kota bukan di tengah jalan maka kita menggunakan distance terpanjang, dan karena kita mencari jarak yang minimal maka kita mereturn distance yang terpendek. Pada implementasinya SR dan SM menggunakan method yang sama yaitu method dijkstra yang mereturn array of int distance.

3. Unweighted Multi Graph
   LAORC kita mencari banyak kombinasi dari path terpendek yang dihasilkan LAOR. LAOR dan LAORC menggunakan method yang sama yaitu bfs yang mereturn arrayList of array of int distance dan paths. Untuk LAOR hanya menggunakan distance dan untuk LAORC menggunakan paths. Paths adalah banyak kombinasi path terpendek dari setiap kota. Pada saat awal kita menginiasi paths index ke start kota dengan 1 karena hanya ada satu kombinasi dari kota start ke kota start itu sendiri. Dan kita tinggal menambahkan logika pada saat looping adj. Jika distance (path terpendek) dari curr ke adj lebih pendek dari distance adj sebelumnya maka paths diupdate menjadi path curr dikali dengan banyaknya jumlahJalan dari curr ke adj. Dan jika terdapat distance(path terpendek) dari curr ke adj yang sama dengan distance adj sebelumnya maka paths ditambah dengan path curr dikali dengan jumlahJalan dari curr ke adj.

Kompleksitas

Kompleksitas OS, CCWG, LAOR, LAORC adalah kompleksitas dari BFS yaitu :

$O(N + M)$

Kompleksitas SR adalah kompleksitas dari Dijkstra yaitu :

$O((N + M) * \text{Log } N)$

Kompleksitas SM adalah kompleksitas dari Dijkstra dikali dengan banyak kota yaitu :

$O(((N + M) * \text{Log } N) N)$

Keterangan: :
N = jumlah vertex (kota)
M = jumlah edge (ada tidaknya jalan yang menghubungkan kota satu dengan kota lain)