Pengenalan Adjacency Matrix, Adjacency List, dan EdgeList

By : Hierony Manurung

Btw, udah ada yang tahu ga ya apa itu **Adjacency Matrix** , **Adjaceny List** , dan **EdgeList**?

Jadi ketiga ini adalah nama tipe data yang digunakan untuk menyimpan informasi dari suatu Graph. Seperti kita tahu Data structure adalah salah satu hal yang penting dalam pertimbangan untuk menyelesaikan soal CP. Lah kalau kita ga tahu cara menyimpan datanya, gimana kita mau mengaplikasikan algoritma seperti **DFS**, **BFS**, **Dijkstra**, **Minimum Spanning Tree**, dan **Max Flow**. Nah, mari kita lupakan sejenak tentang algoritma diatas, dan mempelajari bagaimana cara menyimpan data suatu Graph.

Kita akan menyingkat Adjacency Matrix menjadi **AdjMat**, Adjacency List menjadi **AdjList**, dan untuk EdgeList biar saja tetap menjadi **EdgeList**. Untuk AdjMat dan AdjList aku akan menggunakan contoh **Unweighted Graph** (Tiap Edge memiliki Bobot yang sama) dan setiap Edgenya adalah **one-way**.

Berikut contoh data yang akan digunakan, kita kan menggunakan contoh yang cukup simple :

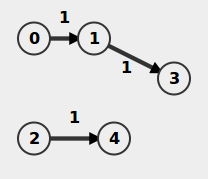
5 3 // 5 adalah jumlah Node (titik) dan 3 adalah jumlah Edge (Garis).

0 1

1 3

2 4

Berikut kira-kira gambar Graph dengan informasi diatas:



Seperti pada gambar, bobot dari tiap edgenya sama, yaitu 1. Mengapa ada nilai 1 sementara di inputan data tidak ada? Ini saya buat agar menunjukkan bahwa **graphnya Unweighted**.

# 1. Adjacency Matrix

Untuk membuat Adjacency Matrix kita membutuhkan **array int 2 dimensi**. Saya pribadi tidak terlalu menyarankan menggunakan AdjMat, karena kompleksitas yang dibutuhkan untuk memproses datanya adalah **O(N^2)**, dimana N adalah jumlah Node dari Graph. Untuk contoh graph diatas, Data yang disimpan akan berbentuk seperti tabel ini:

**int AdjMat[N+5][N+5]**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Node** | **0** | **1** | **2** | **3** | **4** |
| **0** | 0 | 1 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 1 | 0 |
| **2** | 0 | 0 | 0 | 0 | 1 |
| **3** | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 |

Kolom dari tabel ini akan berisi 0 atau 1, **Jika pada kolom (0,1) nilainya bernilai 1 berarti ada edge yang menghubungkan antara node 0 menuju node 1**. Cobalah mencek keatas dan lihat datanya, apakah kamu benar-benar mengerti mengapa isi tabelnya seperti itu?

# 2. Adjacency List

Untuk membuat Adjacency List kita membutuhkan **array of Vector**. Jangan langsung takut dlu ya dengar termnya, hehehhehe. **Jadi Vector itu hampir sama dengan array**, cuman bedanya ukuran dari vector itu dinamis. Hal ini disebabkan, karena vector hanya mengalokasikan memori ketika ada suatu item yang ingin dimasukkan. AdjList ini adalah tipe data favoritku ketika ingin menyimpan informasi graph, karena dia lebih efisien dengan kompleksitas **O(N+M)**, dimana N adalah jumlah Node dan M adalah jumlah edge. Untuk contoh graph diatas, Data yang disimpan dalam bentuk seperti ini.

vector <int> AdjList[N+5]

AdjList[0] : {1} Note : AdjList hanya berfokus untuk menyimpan node

AdjList[1] : {3} Mana yang berhubungan dengan suatu node.

AdjList[2] : {4} Misal untuk **AdjList[0] : {1}**, ini mengartikan

AdjList[3] : {} bahwa ada edge dari node 0 menuju node 1.

AdjList[4] : {}

Untuk **EdgeList**, aku akan menggunakan data yang berbeda. Kenapa? Karena EdgeList menyimpan informasi dari **Weighted Graph** (Tiap edge memiliki bobot yang berbeda). dan setiap Edgenya adalah **two-way**.

Berikut contoh data yang akan digunakan, kita kan menggunakan contoh yang agak lebih ribet :

5 6 // 5 adalah jumlah Node (titik) dan 6 adalah jumlah Edge (Garis).

0 1 3 // Untuk setiap garis terdiri atas **A,B,C**. A adalah Node1, B adalah Node2,

1 2 2 dan C adalah bobot dari Edge yang menghubungkan node1 dan node2.

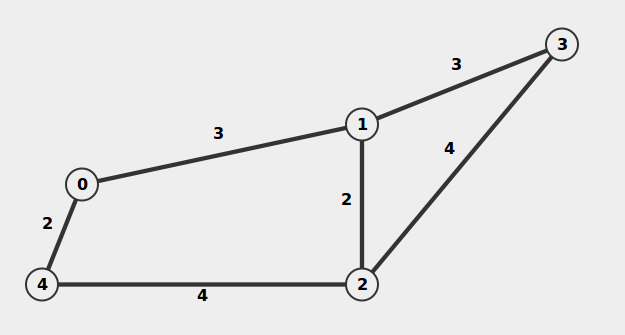
0 4 2

1 3 3

2 3 4

2 4 2

Berikut gambar dari data graph tersebut (Maaf Kurang rapi).



# 3. EdgeList

Untuk membuat EdgeList kita membutuhkan v**ector of Pair< <int, pair<int,int> >**. Memang agak sedikit ribet, aku sarankan membaca buku **Steven Halim Competitive Programming** untuk lebih mengerti lagi. Untuk memproses data EdgeList hanya membutuhkan kompleksitas **O(M)**, dimana M adalah jumlah edgenya. EdgeList dipakai untuk menyelesaikan problem **Minimum Spanning Tree (MST).** Untuk penyimpanan datanya akan seperti ini:

vector < pair <int, pair<int,int> > EdgeList

EdgeList[0] : 3, (0,1) // 3 adalah bobot edgenya, dan (0,1) adalah node yang

EdgeList[1] : 2, (1,2) dihubungkan edge ini.

EdgeList[2] : 2, (0,4)

EdgeList[3] : 3, (1,3)

EdgeList[4] : 4, (2,3)

EdgeList[5] : 2, (2,4)

Nah, sekarang gimana? Mudah-mudahan ada titik teranglah sedikit ya mengenai cara menyimpan data Graph?

Kalau belum jelas, bisa lihat contoh source codenya.