Penggunaan Algoritma Dijkstra dalam Penentuan Lintasan Terpendek Graf

Rahadian Dimas Prayudha - 13509009
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509009@std.stei.itb.ac.id

Abstract— **Aplikasi graf dalam kehidupan sehari-hari tidaklah seidkit. Banyak sekali contoh aplikasi dari graf. Salah satunya adalah dalam penyelesaian masalah lintasan terpendek atau *shortest path problem.* Pada umumnya, dalam permasalahan ini, graf dimodelkan sebagai representasi tempat-tempat dan jalan yang menghubungkan tempat-tempat tersebut. Suatu tempat disimbolkan sebagai titik atau *vertex* dan jalan direpresentasikan sebagai garis atau *edge.* Dalam permasalahan pencarian lintasan terpendek, digunakan representasi graf berbobot. Dengan bobot graf sebagai jarak antara dua tempat. Berbagai macam algoritma dapat ditemukan untuk menyelesaikan permasalahan lintasan terpendek suatu graf. Salah satu algoritma yang paling banyak digunakan adalah algoritma Dijkstra. Makalah ini akan membahas mengenai penggunaan algoritma Dijkstra dalam menyelesaikan masalah penentuan lintasan terpendek graf.**

Kata Kunci— **Graf, lintasan terpendek, *shortest path problem,* algoritma Dijkstra.**

I. PENDAHULUAN

Dalam kehidupan sehari-hari, manusia selalu berpindah-pindah dari suatu tempat ke tempat lainnya dengan berbagai alasan. Misalnya, untuk berpindah tempat tinggal, untuk menghadiri kuliah di kampus, atau juga untuk mendistribusikan barang-barang hasil produksi suatu pabrik secara luas ke kawasan atau daerah tertentu. Untuk melakukan perpindahan tempat tersebut, manusia tentu akan memilih jarak terpendek yang akan menghasilkan biaya yang minimal. Untuk jarak yang relatif pendek, mungkin tidak akan menjadi masalah bagi manusia untuk melaluinya. Namun, saat manusia diharuskan untuk menempuh jarak yang cukup jauh, misalnya dari kota yang satu ke kota lainnya dalam suatu negara, maka masalah efisiensi jarak dan pemilihan lintasan teroendek akan menjadi hal yang penting untuk diperhatikan. Apalagi untuk menempuh jarak yang jauh dan dilakukan berulang-ulang seperti yang dilakukan oleh perusahaan jasa transportasi. Lintasan terpendek menjadi hal yang wajib untuk diketahui.

Terdapat banyak sekali cara untuk mencari lintasan terpendek dari suatu tempat ke tempat lainnya. Salah satu metode yang dapat digunakan, yaitu melalui metode graf berbobot. Yaitu, dengan merepresentasikan suatu tempat sebagai titik atau *node,* dan jalan yang mungkin dilalui untuk mencapai tempat tersebut sebagai garis atau *edge.*

Untuk alternatif jalan yang sedikit atau untuk mencapai tempat yang tidak terlalu jauh, model graf berbobot yang didapat mungkin tidak akan terlalu rumit, sehingga kita dapat dengan mudah menemukan jarak terpendeknya. Namun, apabila kita memodelkan suatu negara yang besar dan terdapat banyak sekali jumlah alternatif jalan, tentu akan menjadi sulit dalam mencari lintasan terpendek dari dua tempat karena pembacaan graf tidak akan semudah itu.

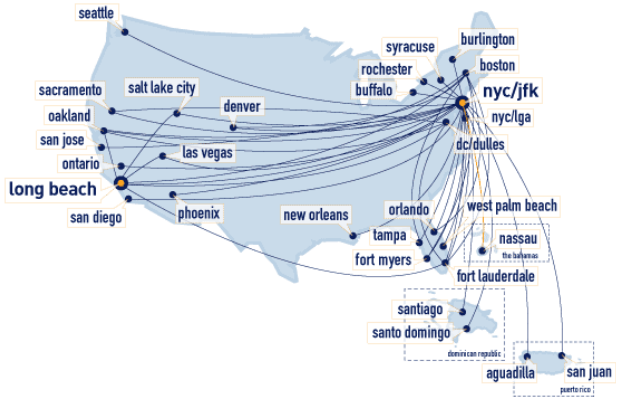
Oleh karena itu, dibutuhkan metode yang dapat memudahkan kita untuk menemukan lintasan terpendek dari graf yang kita miliki tersebut. Salah satu metode yang dapat kita gunakan adalah dengan menggunakan algoritma Dijkstra.

Penggunaan algoritma ini dalam menyelesaikan permasalahan lintasan terpendek membantu dalam meminimalisasi jarak yang perlu ditempuh untuk mencapai suatu tempat yang telah dibuat modelnya dalam bentuk graf berbobot.

Bahkan, algoritma ini terbukti benar dan dapat diterapkan baik untuk graf berarah maupun graf tidak berarah. Hal ini tentu membuat penggunaan algoritma ini menjadi sangat sangkil dalam penemuan lintasan terpendek karena dalam kehidupan nyata, kita cukup banyak menemui jalan-jalan yang hanya dapat dilalui satu arah saja.

II. TEORI DASAR

Teori graf merupakan pokok bahasan yang usianya sudah cukup tua namun memiliki banyak sekali terapannya dalam berbagai hal hingga saat ini. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antar objek-objek tersebut. Representasi visual dari graf adalah dengan menyatakan objek sebagai noktah, bulatan, noktah, titik, atau *node/vertex,* sedangkan hubungan antara objek dinyatakan dengan garis atau *edge*. Dalam kehidupan sehari-hari banyak sekali hal yang sebenarnya dapat direpresentasikan dengan menggunakan graf. Misalnya, untuk menggambarkan peta jalan raya, topologi jaringan, dan untuk menggambarkan kota-kota yang bertetangga dengan warna yang berbeda. Semua hal tersebut pada dasarnya dapat digambarkan representasinya dalam bentuk graf.



Gambar 2.1 Contoh Graf yang Menggambarkan Rute Jet Blue Airlines di Amerika Serikat

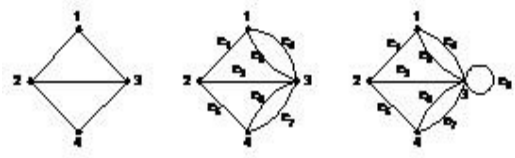
Secara matematis, graf didefinisikan sebagai berikut :

> Graf G didefinisikan sebagai pasangan himpunan (V, E), yang dalam hal ini:
>
> V=himpunan tidak-kosong dari simpul-simpul
> (*vertices* atau *node*) = $\{v_1, v_2, ..., v_n\}$
>
> dan
>
> E = himpunan sisi (*edges* atau *arcs*) yang
> menghubungkan sepasang simpul = $\{e_1, e_2, ..., e_n\}$
>
> atau dapat ditulis singkat notasi $G = (V, E)$.

2.1 Jenis Graf

Berdasarkan keberadaan gelang atau sisi ganda pada suatu graf, maka graf dapat dibedakan menjadi dua jenis :

1. Graf sederhana (*simple graph*), yaitu graf yang tidak mengandung gelang maupun sisi-ganda.
2. Graf tak-sederhana (*unsimple graph*), yaitu graf yang mengandung sisi ganda atau gelang. Ada dua macam graf tak sederhana, yaitu graf ganda (*multigraph*) dan graf semu (*pseudograph*). Graf ganda adalah graf yang mengandung sisi ganda, dan graf semu adalah graf yang mengandung gelang.



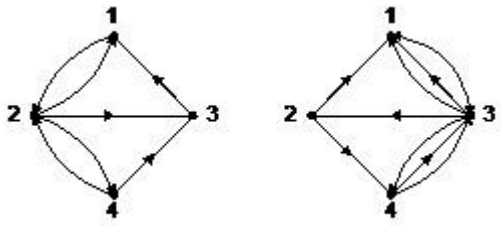
Gambar 2.2 Contoh Graf Sederhana, Graf Ganda, dan Graf Semu

Berdasarkan jumlah simpul dalam suatu graf, maka secara umum graf dapat digolongkan menjadi dua jenis :

1. Graf berhingga (*limited graph*), yaitu graf yang jumlah simpulnya berhingga.
2. Graf tak-berhingga (*unlimited graph*), yaitu graf yang jumlah simpulnya tak berhingga.

Berdasarkan orientasi arah pada sisi, maka secara umum graf dibedakan atas dua jenis :

1. Graf tak-berarah (*undirected graph*), yaitu graf yang sisinya tidak mempunyai orientasi arah. Pada graf tak-berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Jadi, $(v_j, v_k) = (v_k, v_j)$ adalah sisi yang sama.
2. Graf berarah (*directed graph* atau *digraph*), yaitu graf yang setiap sisinya mempunyai orientasi arah. Sisi berarah disebut dengan busur atau *arc*. Pada graf berarah, (v_j, v_k) dan (v_k, v_j) menyatakan dua buah busur yang berbeda. Dengan kata lain, (v_j, v_k) tidak sama dengan (v_k, v_j). Untuk busur (v_j, v_k), simpul v_j dinamakan simpul asal (*initial vertex*) dan simpul v_k dinamakan simpul terminal (*terminal vertex*)

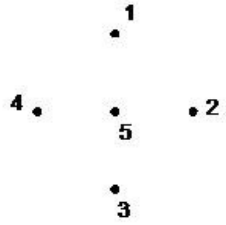


Gambar 2.3 Graf Berarah dan Graf-Ganda Berarah

2.2. **Terminologi Dasar**

Dalam mempelajari graf, dikenal beberapa terminologi dasar yang akan sering digunakan. Berikut akan didefinisikan beberapa terminologi yang akan sering digunakan.

1. Bertetangga (*Adjacent*)
   Dua buah simpul pada graf tak-berarah G dikatakan bertetangga apabila keduanya terhubung langsung dengan sebuah sisi. Dengan kata lain, v_j dikatakan bertetangga dengan v_k jika (v_j, v_k) adalah sebuah sisi pada graf G.
2. Bersisian (*Incident*)
   Untuk sembarang sisi $e = (v_j, v_k)$, sisi e dikatakan bersisian dengan simpul v_j dan simpul v_k.
3. Simpul Terpencil (*Isolated Vertex*)
   Simpul terpencil ialah simpul yang tidak mempunyai sisi yang bersisian dengannya. Atau, dapat juga dinyatakan bahwa simpul terpencil adalah simpul yang tidak satu pun bertetangga dengan simpul lainnya.
4. Graf Kosong (*Null Graph* atau *Empty Graph*)
   Graf yang himpunan sisinya merupakan himpunan kosong disebut graf kosong dan ditulis sebagai N_n , yang dalam hal ini n adalah jumlah simpul.



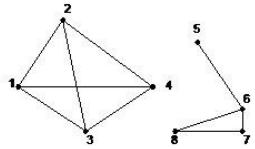
Gambar 2.4 Graf Kosong

5. Derajat (*Degree*)
   Derajat suatu simpul pada graf tak-berarah adalah jumlah sisi yang bersisian dengan simpul tersebut.
   Pada graf berarah, derajat simpul v dinyatakan dengan $d_{in}(v)$ dan $d_{out}(v)$ yang dalam hal ini $d_{in}(v)$ adalah jumlah busur yang masuk ke simpul v, $d_{out}(v)$ adalah jumlah busur yang keluar dari simpul v, dan $d(v) = d_{in}(v) + d_{out}(v)$.
6. Lintasan (*Path*)
   Lintasan yang panjangnya n dari simpul awal v_o ke simpul tujuan v_n di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang terbentuk $v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n$ sedemikian sehingga $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$, ..., $e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf G.

Sebuah lintasan dikatakan lintasan sederhana (*simple path*) jika semua simpulnya berbeda (setiap sisi yang dilalui hanya satu kali).
Lintasan yang berawal dan berakhir pada simpul yang sama disebut lintasan tertutup (*closed path*), sedangkan lintasan yang tidak berawal dan berakhir pada simpul yang sama disebut lintasan terbuka (*open path*).

7. Siklus (*Cycle*) atau Sirkuit (*Circuit*)
   Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus.

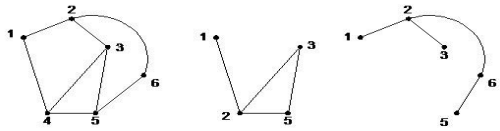
8. Terhubung (*Connected*)
   Dua buah simpul v_j dan simpul v_k dikatakan terhubung jika terdapat lintasan dari v_j ke v_k.
   Graf tak-berarah G disebut graf terhubung (*connected graph*) jika setiap pasang simpul v_j dan v_k di dalam himpunan V terdapat lintasan dari v_j ke v_k (yang juga harus berarti ada lintasan dari v_j ke v_k). Jika tidak, maka graf G disebut graf tak-terhubung (*disconnected graph*).



Gambar 2.5 Graf Tak-Berarah Tidak Terhubung

Graf berarah G dikatakan terhubung jika graf tak-berarahnya terhubung (graf tak-berarah dari G diperoleh dengan menghilangkan arahnya).
Graf berarah G disebut graf terhubung kuat (*strongly connected graph*) apabila untuk setiap pasang simpul sembarang v_j dan v_k di G terhubung kuat. Kalau tidak, G disebut graf terhubung lemah.

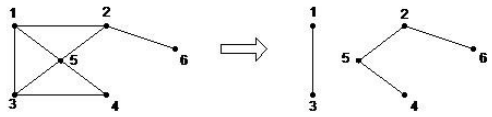
9. Upagraf (*Subgraph*) atau Komplemen Upagraf
   Misalkan $G = (V, E)$ adalah sebuah graf. $G_1 = (V_1, E_1)$ adalah upagraf (*subgraph*) dari G jika V_1 bagian dari V dan E_1 bagian dari E.
   Komplemen dari upagraf G_1 terhadap G adalah graf $G_2 = (V_2, E_2)$ sedemikian sehingga $E_2 = E - E_1$ dan V_2 adalah himpunan simpul yang anggota-anggota E_2 bersisian dengannya.



Gambar 2.6 Graf dan Upagrafnya

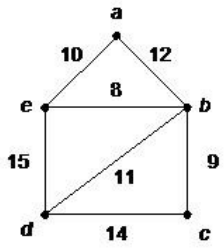
10. Upagraf Merentang (*Spanning Subgraph*)
    Upagraf $G_1 = (V_1, E_1)$ dari $G = (V, E)$ dikatakan upagraf merentang jika $V_1 = V$ (yaitu G_1 mengandung semua simpul dari G).

11. *Cut-Set*
    Cut-Set dari graf terhubung G adalah himpunan sisi yang bila dibuang menyebabkan G tidak terhubung. Jadi *cut-set* selalu menghasilkan dua buah komponen terhubung.



Gambar 2.6 *Cut-Set*

12. Graf Berbobot (*Weighted Graph*)
    Graf berbobot adalah graf yang setiap sisinya diberi sebuah harga (bobot).



Gambar 2.6 Graf Berbobot

III. *THE SHORTEST PATH PROBLEM* (MASALAH LINTASAN TERPENDEK)

Masalah lintasan terpendek atau *shortest path problem* di dalam graf sebenarnya adalah masalah sebuah persoalan optimasi. Dalam penyelesaian masalah lintasan terpendek atau *shortest path problem* digunakan graf berbobot terhubung (*weighted-connected graph*). Bobot dalam graf bisa berarti banyak hal. Jarak, biaya, waktu pengiriman, dan sebagainya. Oleh karena itu, kata "terpendek" tidak sebaiknya diartikan secara harfiah sebagai jarak saja. Namun, inti dari masalah lintasan terpendek atau *shortest path problem* ini adalah menemukan bobot minimal dari lintasan yang dilalui dalam graf.

Hingga saat ini, sudah banyak algoritma untuk mencari lintasan terpendek. Salah satu algoritma yang paling terkenal adalah algoritma Dijkstra.

3.1 Algoritma Dijkstra

Algoritma Dijkstra ditemukan oleh Edsger Wybe Dijkstra. Algoritma ini adalah sebuah algoritma rakus (*greedy algorithm*) yang dipakai dalam memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah graf berarah (*directed graph*) dengan bobot-bobot sisi (*edge weights*) yang bernilai tak-negatif.

Pada awalnya, algoritma ini hanya diperuntukkan untuk graf berarah, namun algoritma ini juga terbukti benar untuk graf tak-berarah.

Properti algoritma Dijkstra:

1. Matriks ketetanggaan M[m_{ij}]

   m_{ij} = bobot sisi (i, j)
   (pada graf tak-berarah $m_{ij} = m_{ji}$)
   $m_{ii} = 0$
   $m_{ij} = \infty$, jika tidak ada sisi dari simpul i ke simpul j

2. Larik S = [s_i] yang dalam hal ini,

   s_i = 1, jika simpul i termasuk ke dalam lintasan terpendek
   s_i = 0, jika simpul i tidak termasuk ke dalam lintasan terpendek

3. Larik/tabel D = [d_i] yang dalam hal ini,

   d_i = panjang lintasan dari simpul awal s ke simpul i

Algoritma Dijkstra dinyatakan dalam notasi *pseudo-code* adalah sebagai berikut :

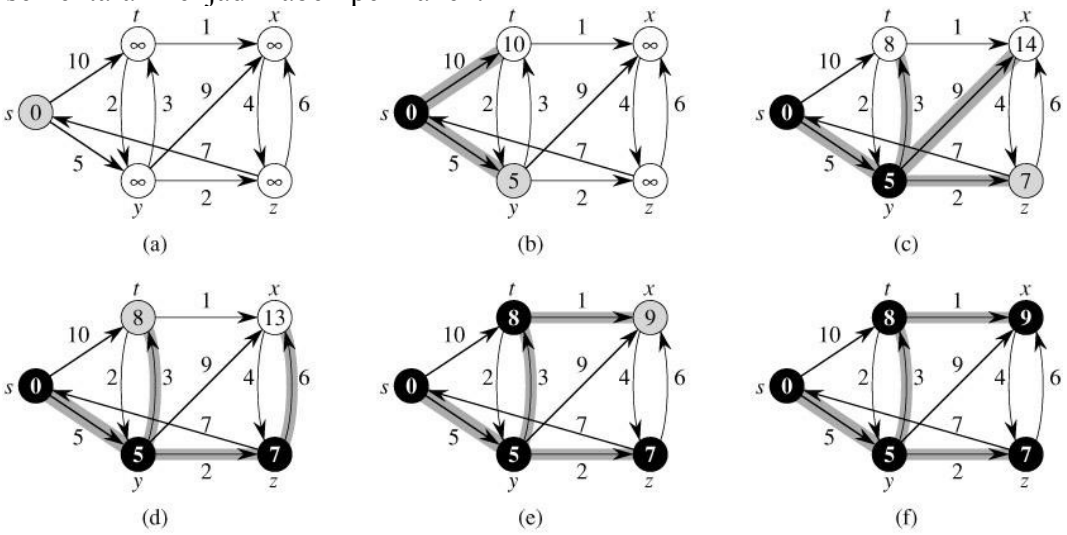
```
Procedure Dijkstra(input m : matriks,
                   a : simpulawal)
{Mencari lintasan terpendek dari simpul awal a
ke semua simpul lainnya
Masukan: matriks ketetanggaan(m) dari graf
berbobot G dan simpul awal a
Keluaran: lintasan terpendek dari a ke simpul
lainnya
}

Deklarasi
   s1, s2, ..., sn : integer {larik integer}
   d1, d2, ..., dn : integer {larik integer}
   i : integer

Algoritma
   { Langkah 0 (inisialisasi) }
   for i←1 to n do
       si ← 0
       di ← mai
   endfor

   { Langkah 1 }
   sa ← 1        {Karena simpul a adalah simpul
                 asal lintasan terpendek, jadi
                 simpul a sudah pasti terpilih
                 dalam lintasan terpendek}
   da ← ∞        {tidak ada lintasan terpendek
                 dari simpul a ke a}

  {Langkah 2, 3, ..., n-1}
  for i←2 to n-1 do
       cari j sedemikian sehingga sj=0 dan
              dj=min(d1, d2, ...dn)
       sj←1 {simpul j sudah terpilih ke dalam
              lintasan terpendek}
       perbarui di, untuk i=1,2,3,...,n dengan:
              di(baru)=min(di(lama), dj+mji)
  endfor
```

IV. ANALISIS

Input dari algoritma Dijkstra berupa sebuah matriks ketetanggan M dari sutu *weighted connected graph* atau graf berbobot terhubung G dan sebuah titik atau *vertex* awal s dalam G. Bobot dari sisi dihitung adalah jarak non-negatif dari suatu *vertex* ke *vertex* lainnya. *Cost* dari sebuah sisi atau *edge* dapat dianggap sebagai jarak antara dua *vertex*, yaitu jumlah jarak semua edge dalam lintasan yang dilalui tersebut. Untuk sembarang pasangan *vertex* s dan t dalam V, algoritma ini menghitung jarak terpendek dari s ke t.

Algoritma dimulai dengan menginisialisasi simpul manapun di dalam graf dengan label permanen bernilai 0 dan simpul-simpul sisanya dengan label sementara bernilai 0. Algoritma ini kemudian memilih nilai sisi terendah yang menghubungkan simpul dengan label permanen ke sebuah simpul lain yang berlabel sementara. Kemudian label simpul sementara di-*update* dari label sementara menjadi label permanen. Nilai simpul sementara yang di-*update* merupakan penjumlahan nilai sisi dan nilai simpul permanen.

Langkah selanjutnya ialah menemukan nilai sisi terendah dari simpul dengan label sementara, kemudian mengubah label simpul sementara terebut menjadi permanen, dan ukur jarak ke simpul permanen awal.



Gambar 4.1 Ilustrasi Pencarian Shortest Path dengan Menggunakan Algoritma Dijkstra

Proses ini diulang terus-menerus sehingga pada akhi
eksekusi algoritma, semua label simpul menjadi permanen.

Lama waktu yang digunakan untuk menjalankan Algoritma Dijkstra's pada suatu graf dengan *E* (himpunan sisi) dan *V* (himpunan simpul) dapat dinyatakan sebagai fungsi E dan V menggunakan notasi Big-O. Waktu yang diperlukan algoritma Dijkstra untuk bekerja ialah sebesar O(V*log V + E). Implementasi paling sederhana dari algoritma Dijkstra ialah penyimpanan simpul dari suatu himpunan ke dalam suatu array atau list berkait.

V. KESIMPULAN

Masalah lintasan terpendek atau *shortest path problem* adalah salah satu permasalahan yang solusinya memanfaatkan aplikasi dari teori graf. Persoalan ini pada dasarnya adalah suatu persoalan optimasi, sehingga semakin mangkus algoritma penyelesaiannya maka akan semakin baik.

Salah satu algoritma yang paling terkenal dalam menyelesaikan masalah lintasan terpendek atau *shortest path problem* dari suatu graf adalah algoritma Dijkstra. Algoritma ini digunakan pada graf berbobot terhubung atau *weighted connected graph* dengan syarat bobot dari masing-masing sisi haruslah bernilai positif.

Waktu yang dibutuhkan algoritma Dijkstra untuk bekerja ialah O(V*log V + E).

DAFTAR REFERENSI

[1]Munir, Rinaldi. 2008. Diktat Kuliah IF2091 Struktur Diskrit. Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[2]Pine Wiki. (2010) http://pine.cs.yale.edu/pinewiki/GraphTheory Tanggal Akses : 14 Desember 2010 pukul 09.40

[3]Wikipedia (2010) http://id.wikipedia.org/wiki/Algoritma_Dijkstra

Tanggal Akses : 14 Desember 2010 pukul 13.23

[4]I.R. (2010) http://rekyanata.blogspot.com/2010_02_01_archive.html Tanggal Akses : 14 Desember 2010 pukul 14.50

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Desember 2010

ttd

Rahadian Dimas Prayudha
13509009