Penerapan Algoritma Greedy pada Dijkstra dalam menentukan rute pesawat terpanjang.

Marcello Faria 13519086
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519086@std.stei.itb.ac.id

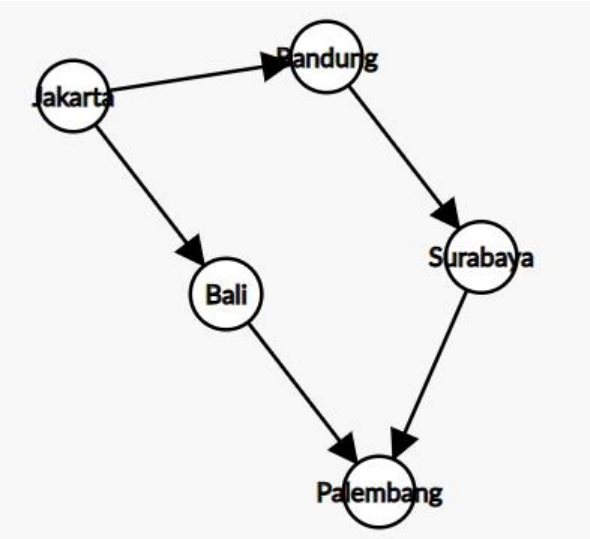
***Abstract*—Dalam sebuah kehidupan, kita akan mendapatkan berbagai macam hadiah yang unik yang didapatkan entah dari undian, doorprize, menang lomba/kontes dan lain-lain. Dalam hadiah sebuah kontes tersebut, umumnya kita bisa mendapatkan hadiah tiket pesawat untuk keliling lokal atau bahkan dunia dengan menggunakan pesawat. Pada makalah kali ini, cakupan bahasan yang akan diangkat adalah penerapan algoritma Dijkstra dalam mencari tur terpanjang yang mengandung kota terbanyak yang dapat dilalui dalam suatu perjalanan sehingga kita mendapat keuntungan yang maksimal. Dalam penyelesaiannya, kita akan membahas penggunaan pendekatan greedy dan diimplementasikan pada Algoritma Dijkstra.**

Keywords—Dijkstra, Greedy, Kontes.

I. PENDAHULUAN

Di berbagai macam lomba, kontes ataupun doorprize, kita akan menemukan hadiah yang memberikan kita kesempatan untuk mengunjungi berbagai kota dalam satu perjalanan. Tentu kita akan memilih rute terpanjang dan memilih kota terbanyak yang dapat kita lalui agar kita tidak rugi. Pada algoritma Dijkstra, umumnya kita mencari shortest path atau lintasan terpendek diantara nodes di dalam sebuah graf, namun pada studi kasus di makalah ini, kita akan membuat sedikit modifikasi pada Dijkstra dengan mencari rute terpanjang dan mengambil nodes terbanyak yang dapat dilalui untuk mendapatkan keuntungan yang maksimal.

Dalam satu tur penerbangan, dimungkinkan kita tidak dapat kembali ke node sebelumnya, karena jadwal yang tidak tersedoa pada hari tersebut, oleh karena itu, kita akan merepresentasikan graf yang dibuat dalam bentuk directed graph, sehingga setiap perjalanan yang dilalui hanya dapat satu arah, dan dipastikan tidak ada cycle. Persoalan ini akan saya generalisasikan dalam bentuk visualisasi gambar agar dapat lebih mudah dipahami dengan setiap node akan melambangkan kota dan edge yang memiliki panah menandakan suatu kota (yang tidak memiliki mata panah) menuju kota lainnya (yang memiliki mata panah).

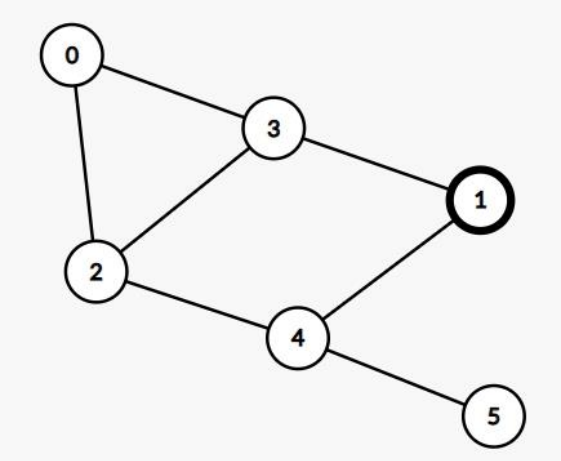


Gambar 1 Contoh Sebuah Directed Graf dengan rute perjalanan
(Sumber: Dokumentasi penulis)

Pada gambar tersebut, dimisalkan kita memiliki 5 node, dengan setiap node melambangkan sebuah kota, yaitu Jakarta, Bandung, Bali, Surabaya, dan Palembang. Pada graf tersebut, edge yang memiliki panah menandakan bahwa tur tersebut hanya dapat dilalui oleh satu arah. Node-node yang terhubung sesuai dengan gambar diatas yaitu Jakarta → Bandung, Jakarta → Bali, Bandung → Surabaya, Bali → Palembang dan Palembang → Surabaya. Dengan ukuran 5 kota, kita dapat memprediksi dengan cara brute force satu per satu lintasan yang dapat dilalui, yaitu dari parent node Jakarta, Bandung, Surabaya, hingga ke Palembang. Namun untuk node yang banyak, dengan upper bound 100 hingga 1000, kompleksitas dari algoritma brute force akan menjadi masalah besar. Oleh sebab itu, kita akan menggunakan pendekatan greedy dan diterapkan pada algoritma Dijkstra untuk menyelesaikan persoalan ini sehingga mendapatkan jalur terpanjang yang dapat dilewati dengan waktu yang polinomial dan cepat.

II. LANDASAN TEORI

A. *Graf*



Gambar 2 Contoh Gambar Graf
(Sumber: Dokumentasi penulis)

Teori graf dalam matematika ataupun dalam ilmu komputer umumnya memiliki sebuah kajian yang mempelajari sifat-sifat "graf" ataupun "grafik" namun memiliki arti yang berbeda dengan "Grafika". Suatu graf pada konteks ilmu computer, terdiri dari sebuah vertices (yang biasa dapat juga dipanggil dengan nodes atau poin) yang terhubung dengan sebuah edge (yang biasa disebut sebagai penghubung atau garis). Perbedaan dari graf dapat dipecah menjadi dua ragam, yang pertama adalah undirected graphs, dimana edge menghubungkan dua nodes secara simetris, yang kedua adalah directed graphs dimana edge menghubung dua nodes secara asimetri.

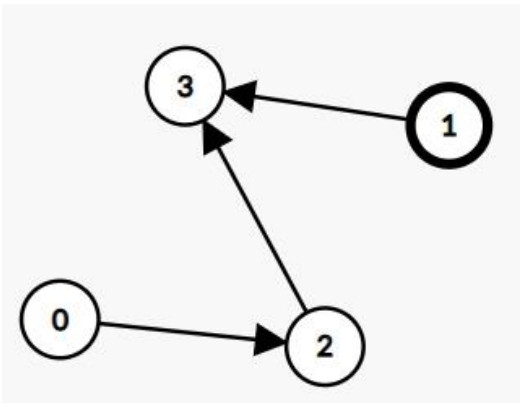
Secara formal, definisi dari graf dapat bermacam-macam disini, saya akan memberikan definisi formal secara ringkas dan dasar. Dalam bentuk yang formal, graf merupakan sebuah himpunan yang terdiri dari pair G = (V, E):

- V, sebuah set/himpunan yang terdiri atas node
- E yang merupakan subset dari {{x,y} | x,y ∈ V dan x tidak sama dengan y}

Sebuah graf dapat digunakan dalam merepresentasikan objek-objek diskrit serta hubungan antara objek-objek tersebut. Sebuah graf pada umumnya terdiri dari beberapa simpul-simpul yang merepresentasikan objek-objek diskrit tersebut, serta sekumpulan sisi yang menghubungkan simpul-simpul tersebut yang merepresentasikan hubungan antara objek-objek diskrit tersebut.

Sebagai contoh, pada graf di gambar di atas dapat dinyatakan dalam graf G = (V, E) di mana V = {0, 1, 2, 3, 4 ,5} dan E = {(0,2), (0,3), (2,3), (2,4), (4,1), (3,1), (4,5)}. Formalisasi ini akan memudahkan saat akan dibahas terminologi lain yang juga berhubungan dengan graf.

B. *Directed graph*



Gambar 3 Contoh Gambar Graf Berarah
(Sumber: Dokumentasi penulis)

Directed graph atau digraph merupakan graph dengan sisi yang memiliki orientasi, sebuah directed graph juga memiliki definisi formal seperti graf pada umumnya yaitu G = (V, E) :

- V, sebuah himpunan dari node
- E, yang merupakan subset dari {{x,y} | (x,y) ∈ V^2 dan x tidak sama dengan y} berupa himpunan sisi yang terdiri dari pasangan node yang tidak teresusun.

Pada sisi (x,y) , node x dapat diartikan menuju node y dim mana node x dan y merupakan endpoint dari sisi, dengan x sebagai kepala dan y sebagai ekor dari sisi. Sebuah sisi dapat dikatakan menggabungkan dx dan y dan insiden pada x dan insiden pada y.

C. *Algoritma Greedy*

Dalam kehidupan kita, terdapat banyak masalah atau persoalan yang menuntut pencarian solusi optimum. Persoalan tersebut dinamakan pesoalan optimasi atau dalam Bahasa inggirs disebut sebagai *optimization problems*. Pada persoalan optimasi, kita diberikan sejumlah kendala (constraint) dan fungsi optimasi. Solusi yang memenuhi semua kendala adalah solusi layak (feasible solution). Solusi layak yang mengoptimumkan fungsi optimasi disebut solusi optimum.

Algoritma greedy pada umumnya merupakan algoritma yang ditujukan untuk memecahkan masalah dalam langkah per langkah. Algoritma greedy ini juga akan mengambil suatu keputusan yang terbaik dalam setiap langkah dan keputusan tersebut tidak dapat diubah lagi pada langkah-langkah setelahnya.

Untuk setiap langkah :

- Ambil sebuah pilihan terbaik, dapat berupa minimum ataupun maksimum yang dapat diperoleh dan juga memperhatikan konsekuensinya.
- Kita harus berharap jika kita memilih optimum lokal untuk setiap langkah, maka nilai optimum tersebut dapat mengarah pada nilai optimum global.

Algoritma greedy akan melakukan pencarian dalam suatu himpunan bagian S, lalu himpunan kandidat C di mana S harus memenuhi kriteria-kriteria yang telah ditentukan yaitu menyatakan hasil suatu solusi dan S dapat dioptimasi dalam sebuah fungsi yaitu fungsi objektif. Secara umum, algoritma greedy ini akan memiliki lima komponen:

1. Himpunan Kandidat (C)

Berisi elemen-elemen pembentuk solusi. Contohnya antara lain dapat berupa himpunan koin, himpunan job yang akan dikerjakan, himpunan simpul di dalah graf, dan lain-lain. Pada setiap langkah, satu buah kandidat diambil dari himpunannya dan dari situ solusi akan dibuat dan dibentuk.

2. Himpunan Solusi (S)

Berisi kandidat-kandidat yang terpilih sebagai solusi persoalan. Himpunan solusi adalah himpunan bagian dari himpunan kandidat.

3) Fungsi Seleksi

Fungsi yang memilih kandidat terbaik yang akan dimasukkan pada solusi. Kandidat yang telah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah berikutnya.

4. Fungsi Kelayakan

Fungsi ini akan melakukan pemeriksaan apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (constrains) yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.

5. Fungsi Objektif

Fungsi ini akan memasukkan sebuah nilai atau hasil pada solusi atau juga dapat berupa solusi yang masih parsial.

D. Algoritma Dijkstra

Algoritma Dijkstra, (dinamai menurut nama penemunya yaitu Edsger Dijkstra), merupakan sebuah algoritma greedy yang dipakai dalam memecahkan persoalan permasalahan terpendek untuk sebuah graf berarah dengan bobot-bobot sisi yang tak bernilai negatif.

Algoritma Dijkstra umumnya digunakan dalam mencari path atau lintasan terpendek pada graf,digraf, maupun graf campuran dengan ketentuan yang memiliki bobot maupun tidak memiliki bobot. Jika diberikan sebuah graf berbobot G = (V, E) dengan sebuah himpunan V = {1,2,3,4 … n} namun dengan syarat bahwa bobot setiap lengan tidak berupa negatif. Setiap Langkah yang dilakukan, dapat didefinisikan terlebih dahulu sebuah PARENT(v) yang menyatakan sebuah vertex yang mendahului atau terlebih dahulu digunakan sebelum vertex v pada lintasan terpendek u0-v yang telah didapat. Variabel tersebut akan terus diupdate (diganti) saat terdapat lintasan u0-v yang lebih kecil atau pendek darinya.

Jika dimisalkan sebuah S berupa suatu himpunan semua vertex dari G, maka berikut langkah-langkah Dalam menemukan rute / path terpendek:

a) input suatu vertex awal u0 dengan jarak d(u0) sebagai 0, dengan (i ←0), S←{u0} dan $\hat{S} \leftarrow V(G) - \{ u0 \}$ dan generate semua jarak lain d(u0) dengan tak hingga atau nol untuk semua $v \in V(G) - \{u0\}$. Jika p bernilai 0 maka langkah yang dilakukan akan dihentikan, selain dari itu, proses tetap dilanjutkan.
b) Untuk setiap $v \in \hat{S}$ sehingga ui $v \in E(G)$, akan kita periksa bila terdapat $l(v) \geq d(ui) + w(ui,v)$, proses akan tetap dilanjutkan; selain dari itu maka l(v) akan diinput dengan d(ui) + w(ui,v) dan PARENT(v) akan diinput ui.
c) Dengan menentukan $m = \min \{d(v)| v \in \hat{S}\}$. Apabila vj merupakan anggota dari $\hat{S}$, dengan d(vj) = m, m akan menjadi jarak antara u0, dengan vj dan ui+1 ← vj.
d) S merupakan irisan dari S dan {ui+1},
e) i ←i+1, namun saat i = p-1, proses tersebut dihentikan dan kembali ke (b).

```
procedure Dijkstra_Insert(x, y : vertex; w_xy : weight);
1.  begin
    Step 1
2.      insert edge (x, y) with weight w_xy in graph G;
    Step 2
3.      if d(x) + w_x,y ≥ d(y) then EXIT;  {no distance from the source has been improved}
    Step 3
4.      Q ⟵ ∅;  {initialization of the global heap}
5.      insert in Q vertex y with priority d(x) + w_x,y and candidate parent x;
6.      for all vertices q ∈ V do mark(q) ⟵ false;
    Step 4
7.      while non-Empty(Q) do
8.          begin
9.              delete from Q vertex q with minimum priority b_ij and candidate parent z;
10.             d(q) ⟵ b_q;
11.             set the new parent of vertex q to be z;
12.             mark(q) ⟵ true;
13.             for all edges (q, r) do
14.                 if d(q) + w_q,r < d(r) and mark(r) = false
15.                     then if r is not in Q
16.                             then insert r in Q with priority d(q) + w_q,r and candidate parent
17.                             else if d(q) + w_q,r ≤ current-priority of r in Q
18.                                     then begin
19.                                             update priority of r in Q to d(q) + w_q,r;
20.                                             set q to be the candidate parent of r;
21.                                         end
22.         end
23. end
```

Gambar 4 Pseudocode algoritma Dijkstra

(Sumber: https://www.researchgate.net/figure/Pseudocode-of-procedure-Dijkstra-Insert_fig1_226102984)

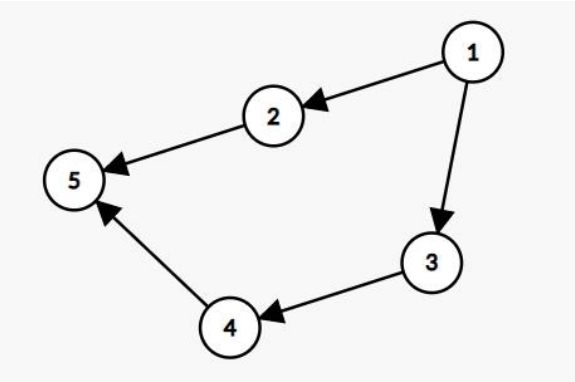
III. IMPLEMENTASI DAN PEMBAHASAN

Contoh penerapan algoritma Dijkstra dengan modifikasi pada penentuan rute terpanjang pesawat dengan jumlah node sebanyak 5.

```
5 5
1 2
2 5
1 3
3 4
4 5
```

Gambar 5 Contoh Gambar Hasil Input Graf Berarah dalam bentuk masukkan pengguna
(Sumber: Dokumentasi penulis)

Berikut merupakan contoh input dari pengguna, dengan baris pertama kolom pertama merupakan banyaknya kota (total node), dan baris pertama kolom kedua merupakan banyaknya jalur penerbangan (total edge). Untuk baris kedua hingga keenam, merupakan pasangan node yang membentuk sebuah hubungan edge dengan arah kolom pertama ke kolom kedua. Dari input tersebut, maka akan dihasilkan graf sebagai berikut:



Gambar 6 Contoh Gambar Hasil Input Graf Berarah dalam bentuk visualisasi
(Sumber: Dokumentasi penulis) Setiap node diberi nomor yang merepresentasikan nama setiap kota agar memudahkan pembacaan dan representasi.

```
cin >> n >> m;

for(int i = 1; i <= n; ++i)
{
    par[i] = -1;
}
int u, v;
for(int i = 0; i < m; ++i)
{

    cin >> u >> v;
    g[u].push_back({v,1});
}
```

Gambar 7 Potongan Kode untuk menerima input pengguna
(Sumber: Dokumentasi penulis)

Berikut merupakan potongan kode untuk menerima input dari user, dengan n sebagai banyak kode (total node), m sebagai total hubungan antara dua kota (total edge), dan array pair of vector g untuk merepresentasikan adjacency list dari sebuah graf. Pada kode tersebut, par merupakan parent dari sebuah node, dan diinisialisasi dengan -1, dan untuk g[u].push_back({v,e}) memiliki maksud u sebagai source node, v sebagai target node dan e sebagai jarak edge, karena jarak setiap edge diasumsikan sama, maka disini kita masukkan 1.

Alasan diasumsikan sama adalah karena pada makalah ini, penulis melakukan asumsi untuk jadwal penerbangan yang telah diatur, sehingga jarak dan waktu dari kota ke kota tidak menjadi permasalahan, namun lebih ditujukan untuk mencari jarak terpanjang atau jumlah kota terbanyak yang dapat dilalui oleh pengguna.

```
void dijkstra()
{
    int u,du;
    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0,1});
    for(int i=1;i<=n;i++)
    {
      dist[i] = 0;
    }
    par[1] = -1;

    while(!pq.empty())
    {

        u = pq.top().second;
        du = pq.top().first;
        pq.pop();

        if(du != dist[u])continue;

        for(auto e: g[u])
        {
            int v = e.first;
            int dv = e.second;
            if(dist[v] < du + dv)
            {
                dist[v] = du + dv;
                pq.push({dist[v],v});
                par[v] = u;
            }
        }
    }
}
```

Gambar 7 Potongan Kode untuk algoritma modifikasi dijkstra
(Sumber: Dokumentasi penulis)

Potongan tersebut merupakan potongan kode fungsi Dijkstra yang telah dimodifikasi. Seusai dengan teori dasar yang telah dijelaskan pada bab sebelumnya.

du: node sumber

u: jarak dari node root ke du

dv: node tetangga

v: jarak dari node root ke dv

vector dist: vector yang menampung jarak dari root node ke suatu node

vector par: vector yang menampung sebuah list yang merupakan leluhurnya/sumbernya.

1. Input suatu vertex awal dengan jarak d(u0) sebagai 0

- Pada potongan kode tersebut, tertulis pq.push({0,1}) yang berarti inisialisasi awal vertex dengan jarak 1, lalu untuk jarak lain kita inisialisasikan dengan jarak nol. Disini penulis menggunakan pq yang memanfaatkan STL priority queue dari c++.

2. Untuk setiap v ∈ Ŝ sehingga ui v ∈ E(G), akan kita periksa bila terdapat l(v) ≥ d(ui) + w(ui,v), proses akan tetap dilanjutkan selain dari itu maka l(v) akan diinput dengan d(ui) + w(ui,v) dan PARENT(v) akan diinput ui.

- Disini penulis melakukan iterasi untuk setiap node yang berhubungan dan berasal dari g[u] (g[u] menandakan node di graf dengan source node u) karena pada kasus ini, kita mencari jarak terpanjang yang dapat ditempuh sehingga pemeriksaan dibalik menjadi l(v) ≤ d(ui) + w(ui,v). Lalu node yang bertanggaan dengan node sumber akan dimasukkan kedalam pq, kemudian dilakukan perulangan while hingga tidak ada lagi elemen pada pq.

```
cout << 1 + (dist[n]) << endl;
vector<int> ans;
int temp = n;
while(temp != -1)
{
    ans.push_back(temp);
    temp = par[temp];
}
reverse(ans.begin(), ans.end());
for(auto u: ans)
{
    cout << u << " ";
}
```

Gambar 8 Potongan kode program untuk menampilkan hasil
(Sumber: Dokumentasi penulis)

Disini penulis menambahkan satu untuk root node, lalu menginisiasi temp dengan n, temp disini merupakan temporary variable untuk parent dari sebuah node. Pada potongan kode tersebut dilakukan pengecekan loop while untuk temp tidak sama dengan -1 yang berarti loop while tetap berjalan hingga menemukan root node (karena root node tidak memiliki parent maka nilainya akan sama dengan inisialisasi parent awal yaitu -1).

Terakhir, dilakukan reverse pada vector ans, (vector ans merupakan vector yang menampung hasil rute yang dikunjungi oleh algoritma dijkstra). Alasan penggunaan reverse adalah penulis ingin menampilkan lintasan terpendek yang diawali dari root node, bukan node terakhir yang dikunjungi.

Berikut merupakan hasil tampilan setelah mengeksekusi seluruh kode program:

```
Total kota yang dapat dikunjungi : 4
Nama kota yang dapat dikunjungi  : 1 3 4 5
```

Gambar 9 Tampilan hasil eksekusi kode program
(Sumber: Dokumentasi penulis)

CONCLUSION

Setelah melakukan berbagai riset dari metode pendekatan algoritma yang ada, terdapat berbagai formulasi untuk menyelesaikan rute penerbangan ini. Salah satunya adalah dengan pendekatan brute-force. Namun pada brute-force, kita akan melakukan permutasi seluruh kemungkinan node yang ada dan akan dilakukan verifikasi apakah jalur tersebut valid atau tidak, kompleksitasnya akan menjadi O(m!) dengan m sebagai jumlah edge pada node. Oleh sebab itu, dilakukan optimisasi dengan metode greedy yang diterapkan pada algoritma Dijkstra, sehingga kompleksitas program menjadi polynomial, Dijkstra pada permasalahan ini, menggunakan modifikasi yang berbeda dari permasalahan Dijkstra pada umumnya, yaitu pada pengambilan rute terpanjang. Secara keseluruhan, kode program yang telah dibahas sebelumnya mampu menangani test case dengan node hingga 10^5 dalam kurun waktu kurang dari 1 detik. Umumnya algoritma Dijkstra akan memiliki kompleksitas O (n^2 + m) dengan n sebagai jumlah node dan m jumlah edge. Tetapi karena pada permasalahan ini bentuk graf merupakan sparse graph, maka penulis melakukan optimasi dengan memanfaatkan library STL priority queue sehingga kompleksitas dapat diturunkan hingga O (n+m log m).

VIDEO LINK AT YOUTUBE

https://youtu.be/o4bS04ouKwE

ACKNOWLEDGMENT

Dalam makalah ini, penulis ingin berterimakasih kepada Tuhan yang Maha Esa karena atas kelimpahan berkat dan anugrah-Nya lah penulis dapat menyelesaikan makalah ini hingga sampai titik ini dengan tepat waktu. Penulis juga ingin berterimakasih terhadap Ibu Nur Ulfa Maulidevi sebagai dosen dari kelas K02 pada mata kuliah Strategi Algoritma ini yang

telah membagikan ilmunya kepada kami, para mahasiswa, selama satu semester ini. Penulis juga ingin berterimakasih pada tim asisten dan dosen dari kelas lain yang telah membuat tugas-tugas dan ujiian sehingga kami dipersiapkan dalam mengerjakan persoalan problem solving dalam berbagai variansi dan kondisi. Penulis juga tidak lupa untuk berterimakasih terhadap semua orang yang telah berkontribusi secara langsung maupun tidak langsung dalam kelancaran penulisan makalah ini. Dan tak lupa, penulis ingin berterimakasih terhadap semua penulis dan pemilik referensi yang tercantum pada makalah ini yang memberi kelancaran dari penulisan ini. Terakhir, penulis ingin mengucapkan permohonan maaf jika ada kesalahan dalam penulisan ini,

REFERENCES

[1] "Graph theory", Encyclopedia of Mathematics, EMS Press, 2001 [1994]

[2] Hoare, C.A.R. (12 October 2010). "The 2010 Edsger W. Dijkstra Memorial Lecture: What Can We Learn from Edsger W. Dijkstra?". Department of Computer Science, The University of Texas at Austin. Retrieved 10 May 2021.

[3] Edsger Dijkstra. A note on two problems in connexion with graphs [1959]

[4] Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein. Introduction to Algorithms [2005]

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, 10 Mei 2021
Marcello Faria - 13519086