Perbandingan Efektivitas Algoritma Djikstra dan Algoritma Pencarian A* untuk digunakan dalam GPS Navigator

Ivana Clairine Irsan 13512041
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
ivanaclairine@s.itb.ac.id

***Abstract*—Makalah ini membahas tentang perbandingan dari dua algoritma yang merupakan pemecahan dari Shortest Path Problem yang ada pada teori graf. Karena setiap algoritma mempunyai kekurangan dan kelebihan masing-masing, akan ditentukan algoritma mana yang paling efisien untuk digunakan pada sistem navigasi GPS guna menunjukkan jalan terpendek menuju tempat tujuan. Algoritma yang akan dibandingkan adalah Algoritma Dijkstra dan Algoritma pencarian A*.**

***Index Terms*— A*, Dijkstra, Shortest Path Problem.**

I. PENDAHULUAN

Seiring dengan perkembangan teknologi dan transportasi, banyak alat telah dikembangkan dalam memudahkan masyarakat, Salah satu alat yang merupakan penemuan besar adalah sebuah peta elektronik yang mudah dibawa ke mana-mana dan memudahkan pengguna untuk mencari jalan yang dituju. Namun, tak cukup sampai di situ saja, sekarang juga banyak perusahaan mengeluarkan sebuah alat yang bernama GPS Navigator yang dapat memberitahu arah ke tempat tujuan sehingga pengemudi tidak perlu melihat peta terus-menerus, melainkan hanya melihat petunjuk dari GPS tersebut.

Kecanggihan alat elektonik tersebut sudah pasti didukung oleh software yang merupakan "otak" dari peranti tersebut. Software inilah yang menentukan bagus tidaknya kinerja suatu peralatan. Oleh karena itu, software ini harus didesain seefisien mungkin agar memberikan performa maksimal sesuai dengan kegunaannya.

Pada GPS Navigator yang memberikan jalan menuju tempat tujuan ini, software yang baik adalah yang dapat memberitahu pengemudi harus mengambil jalan mana agar dapat sampai di tempat tujuan dengan jarak tempuh sependek mungkin tanpa membingungkan pengemudi dengan memberi petunjuk untuk masuk ke jalan yang tidak dapat dilalui karena merupakan jalur satu arah. Untuk merealisasikannya, dapat digunakan beberapa algoritma penyelesaian Shortest Path Problem yang ada dalam teori graf. Algoritma yag sudah ditemukan antara lain adalah Algoritma Dijkstra dan Algoritma pencarian A*. Kedua algoritma ini mempunyai tujuan yang sama, yaitu menentukan jalur terpendek menuju tempat tujuan, namun dengan kompleksitas dan waktu *runtime* yang berbeda. Untuk itulah perlu dibandingkan algoritma mana yang dapat memberikan hasil terbaik menurut teori kompleksitas algoritma.

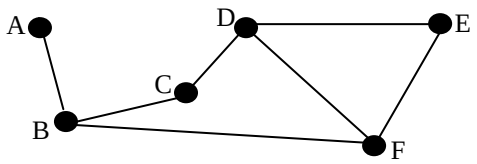
II. DASAR TEORI

A. Teori Graf

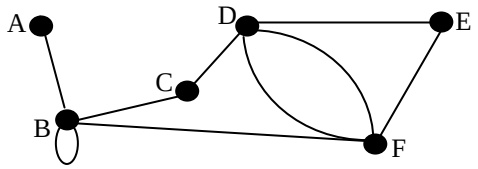
Graf adalah himpunan benda-benda yang dapat merepresentasikan objek-objek dan hubungan dari objek-objek tersebut. Biasanya objek dalam graf digambarkan sebagai simpul *(vertex)* yang keterhubungannya dengan simpul lain disimbolkan oleh garis yang disebut busur *(edge)*.

Secara matematis, graf dapat digambarkan sebagai berikut : $G = (V,E)$, dengan V adalah himpunan tidak kosong dari simpul *(vertices)* = { $V_1, V_2, \ldots, V_n$ } dan E adalah himpunan sisi *(edges)* yang menghubungkan sepasang simpul = { $e_1, e_2, \ldots, e_n$ }.

Graf dibagi menjadi dua jenis berdasarkan ada tidaknya sisi ganda atau sisi gelang, yaitu graf sederhana (tidak mengandung sisi gelang maupun sisi ganda) dan graf tak-sederhana yang mengandung sisi gelang atau sisi ganda. Sisi ganda adalah adanya lebih dari satu sisi yang menghubungkan dua simpul, dan sisi gelang adalah sebuah sisi yang terhubung pada satu simpul (biasanya berbentuk cincin).

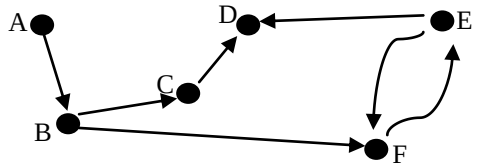


Gambar 1 : Graf G1, contoh graf sederhana dan tak berarah dengan 6 simpul dan 7 busur



Gambar 2 : Graf G2, contoh graf tak-sederhana dengan 6 simpul, 1 sisi gelang, dan sepasang sisi ganda.

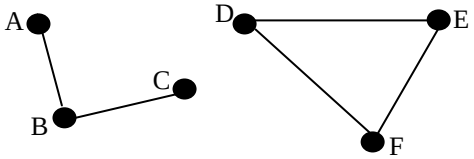
Berdasarkan orientasi arah pada sisi, graf dibagi menjadi dua jenis yaitu graf berarah dan graf tidak berarah. Graf berarah inilah yang dapat digunakan pada sistem navigasi untuk menggambarkan arah arus pada jalan raya.



Gambar 3 : Contoh graf berarah dengan sisi ganda

Beberapa terminology graf antara lain : *(tidak semua akan ditulis di sini, hanya beberapa yang berkaitan dengan kepentingan algoritma)*

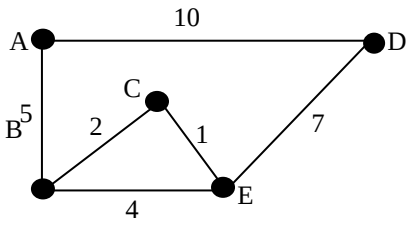
1. Ketetanggaan *(adjacent)*
   Dua buah simpul pada graf dikatakan bertetangga apabila keduanya terhubung langsung oleh sebuah sisi. Pada graf *G1* dalam *Gambar 1*, simpul D bertetangga dengan simpul C, E, dan F, namun tidak bertetangga dengan simpul A dan B.
2. Bersisian *(Incidency)*
   Untuk sembarang sisi $e = (V_j, V_k)$ dikatakan e bersisian dengan simpul V_j, atau e bersisian dengan simpul V_k. Pada graf *G1* dalam *Gambar 1*, sisi (A,B) bersisian dengan simpul A dan simpul B, sisi (B,C) bersisian dengan simpul B dan simpul C, tapi sisi (A,B) tidak berisisan dengan simpul E.
3. Derajat *(Degree)*
   Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Notasi : *d(v).*
   Pada graf *G1* dalam *Gambar 1*, dapat dilihat bahwa
   $d(A) = 1$,
   $d(B) = d(D) = d(F) = 3$,
   $d(C) = d(E) = 2$.
4. Lintasan *(Path)*
   Lintasan (*path)* yang panjangnya *n* dari simpul awal V_0 ke simpul tujuan V_n di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $V_0, e_1, V_1, V_2, \ldots, V_{n-1}, e_n, V_n$ sehingga $e_1 (V_0, V_1)$, $e_n (V_{n-1}, V_n)$ adalah sisi-sisi dari graf G. Panjang lintasan adalah jumlah sisi dalam lintasan tersebut. Lintasan A-B-C-D-E-F pada graf *G1* dalam *Gambar 1* mempunyai panjang lintasan 6.
5. Siklus *(cycle)*
   Siklus *(cycle)* merupakan lintasan yang berawal dan berakhir pada simpul yang sama. Misalnya pada graf *G1* dalam *Gambar1*, B-C-D-F-B adalah sebuah sirkuit. Panjang sirkuit adalah jumlah sisi dalam sirkuit tersebut. Sirkuit B-C-D-F-B pada *G1* memiliki panjang 4.
6. Terhubung *(Connected)*
   Dua buah simpul V_1 dan V_2 disebut terhubung jika terdapat lintasan (sisi) di antara kedua simpul tersebut. Graf G disebut terhubung jika tiap simpul mempunyai pasangan yang terhubung oleh satu atau lebih sisi. Jika ada simpul yang terpisah dan tidak terhubung, maka graf itu disebut graf tak terhubung.



Gambar 4 : Contoh graf tak terhubung

7. Graf Berbobot *(Weighted Graph)*

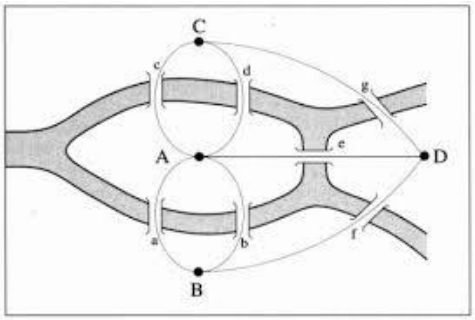
Graf berbobot adalah graf yang sisinya mempunyai nilai, yaitu panjang sisi. Misalnya pada suatu peta, panjang sisi akan melambangkan jarak tempuhnya.



Gambar 5 : Contoh graf berbobot

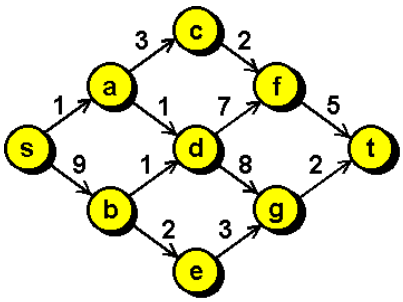
B. Shortest Path Problem

Teori graf untuk pertama kalinya muncul pada tahun 1736 diawali oleh adanya sebuah permasalahan yang disebut masalah jembatan Koningsberg. Masalah jembatan Konigsberg adalah bagaimana melewati tujuh jembatan yang ada tanpa tepat hanya sekali. Untuk membuktikan bahwa tidak ada penyelesaian dari permasalahan ini, seorang matematikawan bernama Leonhard Euler menjawabnya dengan menggunakan graf. Ia melambangkan tiap kota yang dihubungkan oleh jembatan tersebut sebagai sebuah simpul dan jembatan yang menghubungkan setiap kota tersebut adalah sebagai sisi. Persoalan ini tidak memiliki jawaban karena menurut teori graf, sebuah graf dengan tiap simpul berderajat ganjil tidak akan dapat dilalui tepat sekali untuk kembali ke tempat semula (tidak ada lintasan Euler pada graf tersebut).



Gambar 6 : Permasalahan jembatan Koningsberg divisualisasikan dengan graf

Setelah berkembangnya teori graf, ternyata muncul pertanyaan-pertanyaan baru mengenai bidang ini. Salah satunya yang akan dibahas adalah *Shortest Path Problem* yang mencari jarak terpendek dari simpul pertama ke simpul kedua tanpa harus melalui semua simpul.



Gambar 7 : Graf berarah G3

Pada graf *G3* pada gambar di atas, jalan yang dapat ditempuh dari simpul s menuju ke simpul t ada beberapa cara, antara lain :

s → a → c → f → t (11)
s → a → d → f → t (14)
s → a → d → g → t (12)
s → b → e → g → t (16)
s → b → d → g → t (20)
s → b → d → f → t (22)

dapat kita lihat bahwa jalur terpendek yang dapat ditempuh adalah s-a-c-f-t dengan total lintasan 11. Untuk kasus ini, penyelesaiannya masih dapat dihitung secara manual karena jumlah simpul dan bentuk graf belum terlalu kompleks, namun untuk penggunaan dengan skala yang lebih besar, seperti peta, dibutuhkan penyelesaian yang lebih efektif dalam menghitung jarak terpendek yang dapat ditempuh.

Untuk mencari penyelesaian dari masalah ini, beberapa algoritma telah digunakan, di antaranya adalah :

1. *Dijkstra's Algorithm*

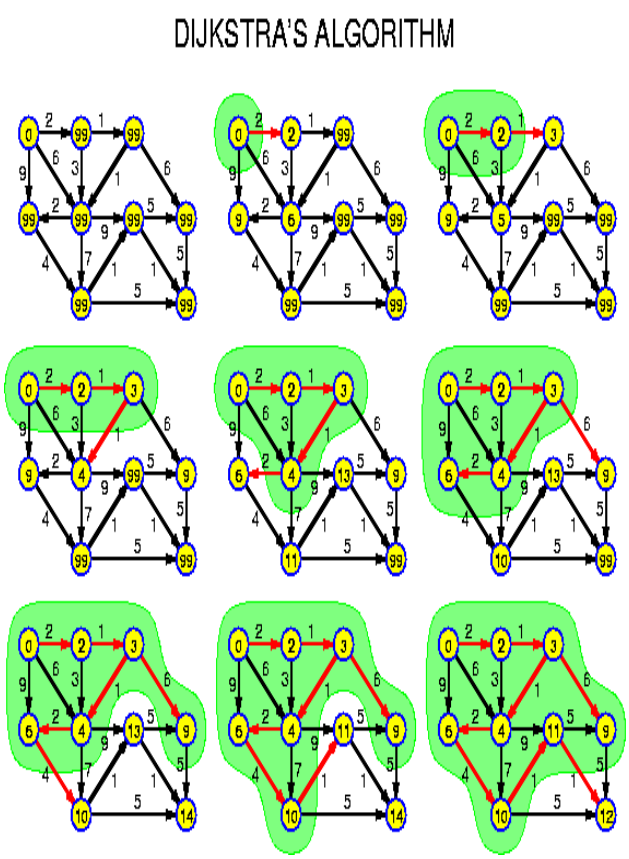
Algoritma ini ditemukan pada tahun 1956 oleh Edsger Dijkstra dah dipublikasikan pada tahun 1959. Algoritma Djikstra ini dapat menyelesaikan permasalahan *Shortest Path Problem* pada graph berarah dengan bobot non-negatif, dan efektif untuk menyelesaikan *single source shortest path problem.* Apabila pada peta setiap kota digambarkan sebagai simpul dan jaraknya digambarkan sebagai bobot graf, maka jarak terpendeknya dapat ditentukan dengan menggunakan algoritma Djikstra ini.

Algoritma asli yang pertama diciptakan oleh Djikstra tidak menggunakan *min-priority queue* dan memiliki *running time* sebesar $O(|V|^2)$ dengan |V| adalah jumlah simpul (*vertex*) yang dimiliki graf. Setelah dijalankan berdasarkan *min-priority queue* dan diimplementasikan dengan *Fibonacci Heap*, *running time* dari algoritma ini meningkat menjadi O(|E| + |V| log |V|) dengan |E| adalah jumlah sisi dari graf.

Kekurangan dari algoritma ini adalah tidak dapat digunakan untuk menghitung jarak terpendek dari graph dengan bobot negatif.

Langkah-langkah dalam proses algoritma ini adalah :
Anggap node di mana kita memulai adalah **node awal**.
Jarak menuju node Y adalah jarak dari **node awal** menuju node Y. Algoritma Djikstra akan menginisialisasi beberapa nilai jarak dan akan memperbaikinya langkah per langkah.

1. Memasukkan sebuah nilai pada tiap simpul. Menginisialisasi angka 0 pada node awal dan nilai tak hingga pada node lainnya
2. Menandai setiap simpul yang belum disinggahi. Mengubah status tiap node yang sedang dikunjungi menjadi ***current node***. Membuat himpunan dari simpul yang belum dikunjungi.
3. Untuk node yang sedang disinggahi, perhitungkan semua tetangga yang belum dikunjungi dan hitung jarak *tentative* terhadap node tetangga tersebut. Misal, jika node A saat ini mempunyai nilai 6, dan sisi yang menghubungkannya dengan node B mempunyai panjang 2, maka jarak ke B menjadi 8.
4. Apabila node tujuan telah ditandai *visited* (node Y sudah tercapai), atau graf yang diberikan merupakan graf tak terhubung dan setiap node yang terhubung telah dikunjungi, maka algoritma telah selesai dan akan berhenti.
5. Pilih node yang belum dikunjungi dengan jarak *tentative* terkecil dan menyetelnya menjadi *current node*, lalu kembali ke step 3.



Gambar 8 : Visualisasi Algoritma Dijkstra (angka 99 dianggap sebagai pengganti simbol tak hingga)

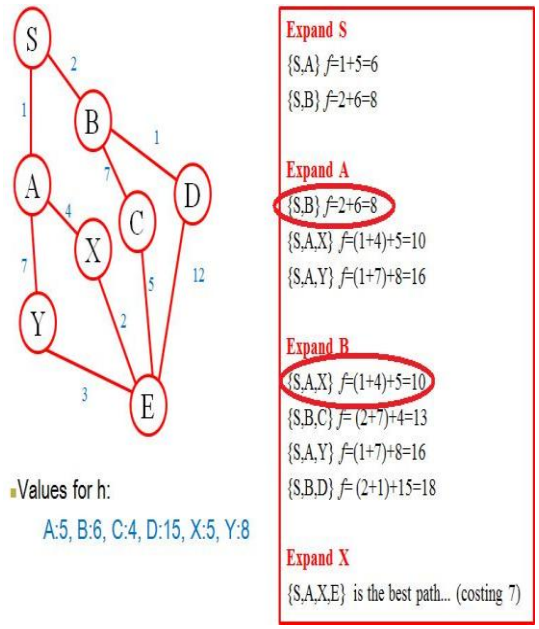
Pseudocode dari Algoritma Dijkstra ini adalah :

```
function Dijkstra(Graph, source):
        for each vertex v in Graph:
            dist[v]      := infinity;
            visited[v]   := false;
            previous[v]  := undefined;
        end for

        dist[source]  := 0;
        insert source into Q;

        while Q is not empty:
            u := vertex in Q with smallest distance
                 in dist[] and has not been visited;
            remove u from Q;
            visited[u] := true

            for each neighbor v of u:
                alt := dist[u] + dist_between(u, v);
                if alt < dist[v]:
                    dist[v]  := alt;
                    previous[v]  := u;
                    if !visited[v]:
                        insert v into Q;
                    end  if
                end if
            end for
        end while
        return dist;
end function
```

2. *Algoritma Pencarian A**

Pada tahun 1968 Nils Nilsson mengusulkan untuk menggunakan pendekatan heuristik pada algoritma Dijkstra yang pada saat itu adalah algoritma path-finding yang umum digunakan. Algoritma pencarian A* akan menghitung semua kemungkinan jalan sampai ujung dan kemudian membandingkannya untuk mengambil jalan yang terpendek.

Kompleksitas waktu dari algoritma A* bergantung pada heuristik. Pada performa terburuknya, *running time* algoritma ini adalah O (log h*(x)).



Gambar 9 : Visualisasi Algoritma Pencarian A*

Pseudocode dari *Algoritma Pencarian A** adalah sebagai berikut:

```
function A*(start,goal)
    closedset := the empty set
    openset := {start}
    came_from := the empty map

    g_score[start] := 0

    f_score[start] := g_score[start] +
          heuristic_cost_estimate(start, goal)

    while openset is not empty
        current := the node in openset having the
                          lowest f_score[] value
        if current = goal
            return reconstruct_path(came_from,
                    goal)

        remove current from openset
        add current to closedset
         for each neighbor in
             neighbor_nodes(current)
            tentative_g_score := g_score[current] +
                 dist_between(current,neighbor)
            tentative_f_score := tentative_g_score +
                 heuristic_cost_estimate(neighbor,
            goal)
            if neighbor in closedset and
            tentative_f_score >= f_score[neighbor]
                  continue

            if neighbor not in openset or
            tentative_f_score < f_score[neighbor]
               came_from[neighbor] := current
               g_score[neighbor] :=
            tentative_g_score
               f_score[neighbor] := tentative_f_score
               if neighbor not in openset
                   add neighbor to openset

    return failure

function reconstruct_path(came_from,
      current_node)
    if current_node in came_from
        p := reconstruct_path(came_from,
                        came_from[current_node])
        return (p + current_node)
    else
       return current_node
```

3. Kompleksitas Algoritma

Selain kebenaran suatu algoritma, yang harus diperhatikan juga ialah efisiensi dari *running time* dan memori yang digunakan.

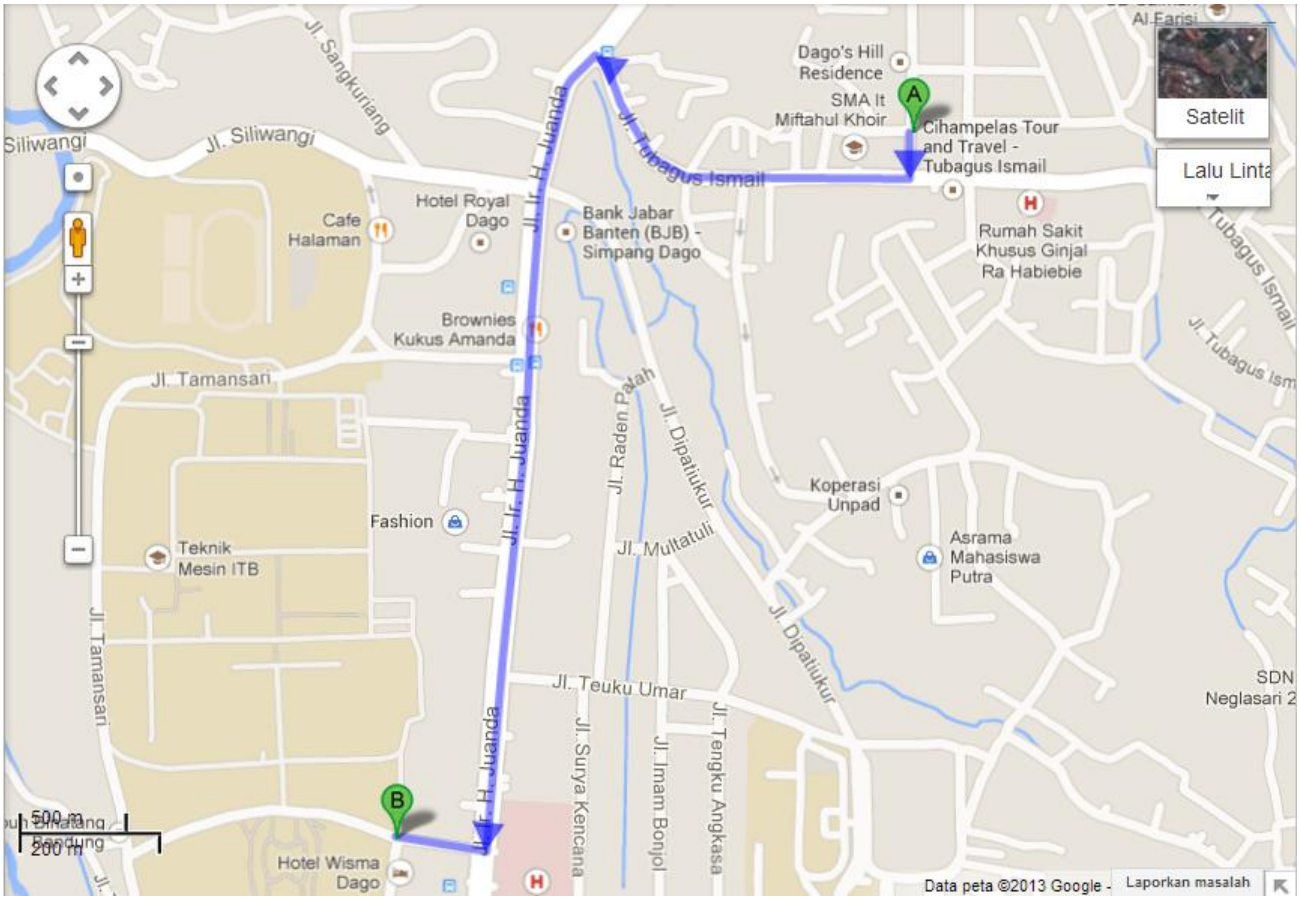
Perhitungan pertumbuhan fungsi sangat krusial dalam menghitung efisiensi sebuah algoritma. Seperti layaknya hal-hal krusial lainnya pada ilmu komputer, tentunya fungsi pertumbuhan ini juga memiliki notasi matematika khusus. Penulisan fungsi pertumbuhan dilakukan dengan menggunakan notasi asimtotik.

Terdapat beberapa jenis notasi asimtotik, tetapi Big-O dipilih karena merupakan notasi yang paling populer dan paling banyak digunakan pada kalangan peneliti ilmu komputer. Notasi Big-O digunakan untuk mengkategorikan algoritma ke dalam fungsi yang menggambarkan batas atas (*upper limit*) dari pertumbuhan sebuah fungsi ketika masukan dari fungsi tersebut bertambah banyak. Singkatnya, perhitungan jumlah langkah dan pertumbuhannya yang kita lakukan pada bagian sebelumnya merupakan langkah-langkah untuk mendapatkan fungsi Big-O dari sebuah algoritma.

III. Perbandingan Efektifitas Algoritma Dijkstra dan Algoritma Pencarian A*

A. Shortest Path Problem pada GPS Navigator

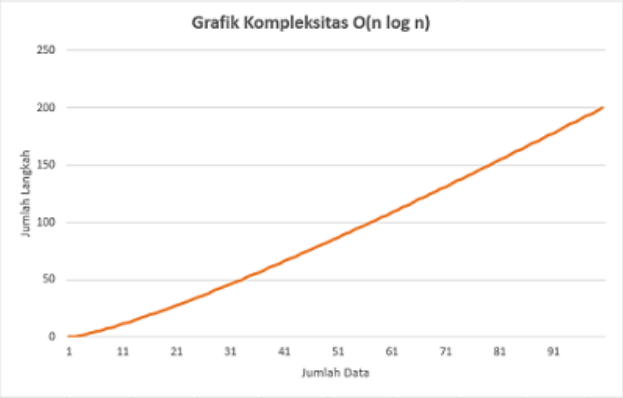
Pada GPS Navigator yang umum digunakan, pengguna ingin mencari jalan dari satu tempat ke tempat lain, dengan kata lain, masalah yang perlu diangkat adalah *single source shortest path problem*. Pada permasalahan yang harus diselesaikan oleh GPS ini, program tidak perlu mencari semua kemungkinan jalan yang ditempuh, cukup menampilkan jalan yang terpendek saja, seperti yang ditampilkan pada gambar di bawah ini :



Gambar 10 : Contoh Peta dari Tubagus Ismail ke ITB, yang ditunjukkan hanya satu jalan dengan jarak terpendek

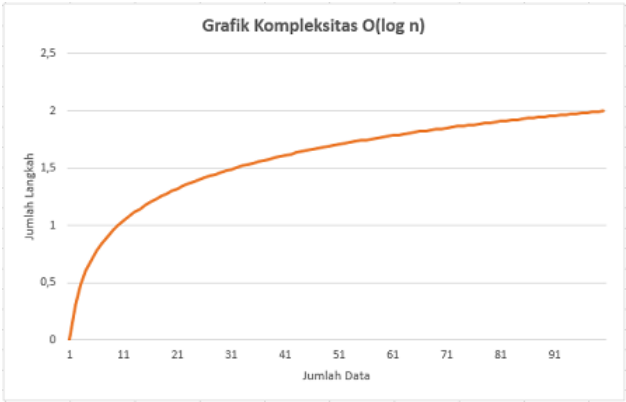
*B. Perbandingan Kompleksitas Algoritma Dijkstra dan Algoritma pencarian A**

Dari data yang ada di atas, diperoleh bahwa *runtime* yang diperlukan oleh algoritma Dijkstra pada performa terburuknya adalah O(|E| + |V| log |V|).



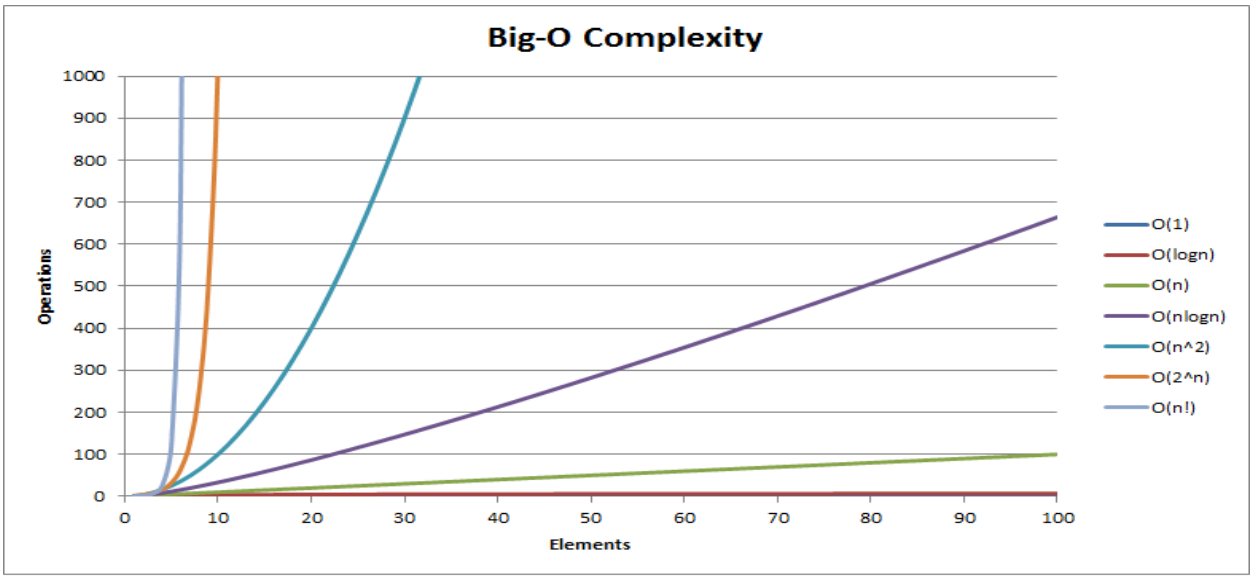
Gambar 10 : Grafik Kompleksitas dari Algoritma Dijkstra

Sementara itu, waktu *runtime* yang diperlukan oleh algoritma pencarian *A pada waktu terburuknya adalah O(log h*(x)), dengan h* adalah heuristik optimal.



Gambar 11 : Grafik Kompleksitas dari Algoritma Pencarian *A

Perbandingan dari kompleksitas-kompleksitas algorirma berdasarkan teori Big-O dapat ditunjukkan dalam gambar berikut :



Gambar 12 : Grafik Kompleksitas Algoritma berdasarkan Big-O Theory

Algoritma pencarian A* biasanya digunakan untuk mencari jalan terpendek dengan adanya halangan seperti *maze*, sedangkan algoritma Dijkstra lebih umum digunakan untuk mencari jalan terpendek dari sebuah peta.

IV. KESIMPULAN

Algoritma Dijkstra dan Algoritma Pencarian A* yang merupakan pemecahan dari *Shortest Path Problem* sama-sama bernilai benar namun memiliki kompleksitas dan keunggulan yang berbeda.

Dari pembahasan dan data yang ada di atas, dapat disimpulkan bahwa Algoritma Dijkstra tidak lebih baik untuk digunakan dalam sistem GPS Navigator karena mempunyai kompleksitas yang kurang efisien jika dibandingkan dengan Algoritma Pencarian A*.

Pada sistem GPS Navigator yang perlu menampilkan semua kemungkinan jalan, algoritma pencarian A* juga dapat dimodifikasi sehingga mampu menampilkan hasil tiap traversalnya yang kemudian akan memunculkan banyak alternatif jalan menuju tempat tujuan.

VII. UCAPAN TERIMA KASIH

Rasa terima kasih saya yang paing besar saya ucapkan kepada Tuhan Yang Maha Esa atas karunia-Nya yang memampukan saya menyelesaikan makalah ini. Rasa terima kasih juga saya ucapkan kepada dosen saya, Dr. Ir. Rinaldi Munir dan Ibu Harlili, M.Sc. atas bimbingan dan pengajarannya selama ini yang memberi saya ilmu yang sangat berguna baik dalam menyelesaikan makalah ini maupun dalam kepeluan kuliah. Tak lupa juga saya ingin menyampaikan terima kasih kepada orang tua dan teman-teman yang telah membantu melalui semangat dan doa untuk saya.

REFERENSI

[1] http://www.cs.uky.edu/~lewis/cs-heuristic/text/dynamic/shortest.html, diakses pada tanggal 14 Desember 2013

[2] Munir, Rinaldi, "Matematika Diskrit", Informatika, Bandung: 2010

[3] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". *Introduction to Algorithms* (Second ed.). MIT Press andMcGraw–Hill. pp. 595–601

[4] Delling, D. and Sanders, P. and Schultes, D. and Wagner, D. (2009). "Engineering route planning algorithms". *Algorithmics of large and complex networks*. Springer. pp. 117–139.

[5] http://bertzzie.com/knowledge/analisis-algoritma/2-KompleksitasAlgoritma.html, diakses pada tanggal 14 Desember 2013

[6] http://www.policyalmanac.org/games/aStarTutorial.htm, diakses pada tanggal 15 Desember 2013

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, 15 Desember 2013

Ivana Clairine Irsan / 13512041