Perbandingan Algoritma BFS, DFS, dan A* dalam Pencarian Jalur dengan Waktu Tercepat Suatu Perjalanan pada Aplikasi Peta Digital

Anissa Putri Dinanti - 13517121
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517121@std.stei.itb.ac.id

***Abstrak*—Seiring dengan berkembangnya teknologi informasi dan informasi maka manusia memiliki kecenderungan untuk melakukan mobilitas yang lebih tinggi. Tidak jarang mobilitas ini memakan waktu yang cukup banyak jika tidak diperhitungkan dengan melewati jalur yang tercepat. Dengan adanya peta digital dan segala fitur-fiturnya yang sangat memudahkan manusia, jalur dengan waktu tercepat dapat dicari untuk efisiensi dan efektifitas waktu yang diperlukan manusia untuk melakukan mobilitas dari satu tempat ke tempat lain. Untuk mencari jalur yang optimal tersebut tentunya diperlukan suatu algoritma yang cocok dan tepat agar jalur yang ditemukan pun optimal dan akurat.**

Kata Kunci—Algoritma A*, Algoritma BFS, Algoritma DFS, waktu tercepat, jalur, peta digital.

I. PENDAHULUAN

Perkembangan teknologi saat ini sudah semakin canggih, dan perkembangan itu akan tetap terus terjadi tanpa pernah berhenti. Terutama perkembangan teknologi informasi seperti internet. Saat ini internet sudah bukan merupakan hal yang asing untuk manusia. Hampir seluruh manusia di dunia sudah merasakan dampak dan manfaat dari internet itu sendiri. Kemudahan akses internet sudah diberikan dengan mudah dimana-mana. Dengan berkembangnya kemudahan akses internet ini juga mempengaruhi kebutuhan-kebutuhan manusia yang semakin mudah terpenuhi.

Manusia selalu menginginkan kemudahan dalam memenuhi kebutuhan-kebutuhannya. Sebisa mungkin kebutuhannya dipenuhi dalam suatu aktivitas yang diharapkan dapat berjalan secara efektif dan efisien. Keefisiensian inilah yang dikejar seluruh manusia agar tidak banyak waktu yang terbuang dengan sia-sia. Dalam melakukan aktivitasnya, manusia membutuhkan mobilisasi dari suatu tempat ke tempat lain. Mobilisasi ini tidak dapat berjalan tanpa adanya transportasi atau pembangunan infrastruktur jalan yang baik. Maka, seiring dengan berkembang teknologi informasi, teknologi dalam bidang lain seperti teknologi transportasi dan infrastruktur juga turut berkembang. Berkembangnya teknologi transportasi dan infrastruktur ini merupakan tuntutan dari manusia yang memiliki kebutuhan untuk berpindah tempat dengan waktu sesedikit mungkin.

Dengan permasalahan kebutuhan ini, dibentuklah sebuah aplikasi peta digital yang dapat memudahkan manusia dalam melakukan mobilisasi dari suatu tempat ke tempat lain. Aplikasi peta digital ini ialah sebuah gambaran yang berisi informasi pada suatu daerah dan merupakan sebuah digitalisasi dari peta fisik yang dahulu sering digunakan. Aplikasi peta digital ini banyak digunakan oleh orang-orang yang berpergian jauh maupun dekat, berjalan kaki atau bertransportasi, dan dari orang dewasa ke orang muda. Manfaat dari aplikasi ini dapat dirasakan oleh segala kalangan dari segala umur. Saat ini banyak perusahaan yang sedang mengembangkan aplikasi peta digital ini karena banyaknya pengguna yang membutuhkannya. Contohnya adalah aplikasi-aplikasi yang sering ditemui sehari-hari seperti, “Google Maps”, “Waze”, dan aplikasi-aplikasi peta digital lainnya.

Gambar 1 Contoh penggunaan peta digital untuk mencari jalur dengan waktu tercepat dalam suatu perjalanan
(sumber: https://www.google.com/maps)

Saat ini karena kemudahan akses internet dan aplikasi-aplikasinya, membuat peta fisik sudah sangat jarang ditemukan dan digunakan lagi. Masyarakat dapat dengan mudah mengakses peta digital pada telepon genggamnya masing-

masing. Fitur-fitur yang dimiliki dalam peta digital juga memudahkan masyarakat untuk mengetahui jarak terdekat untuk mencapai suatu tujuan atau bahkan waktu tercepat tanpa harus mengira-ngira lagi seperti pada peta fisik. Keakuratan prediksi jalan yang ditunjukkan peta digital saat ini sangat memudahkan masyrakat untuk menemukan waktu tempuh tercepat untuk mencapai suatu tujuan.

Sifat manusia yang tidak pernah puas dan selalu ingin memaksimalkan efektifitas dan efisiensi dalam kehidupannya, mendorong para *programmer* untuk membuat algoritma-algoritma yang paling efisien untuk permasalahan-permasalahan yang ditemukan. Contohnya adalah pada kasus ini, penulis mengambil judul "Perbandingan Algoritma DFS, BFS, dan A* dalam Pencarian Jalur dengan Waktu Tercepat Suatu Perjalanan pada Aplikasi Peta Digital", hal ini dilakukan untuk mengetahui algoritma pencarian mana yang dapat menemukan hasil paling optimal dengan waktu yang optimal pula. Dengan dibandingkannya ketiga algoritma ini diharapkan dapat diketahui algoritma yang paling cocok dalam pembuatan aplikasi peta digital agar didapatkan waktu perjalanan yang paling optimal.

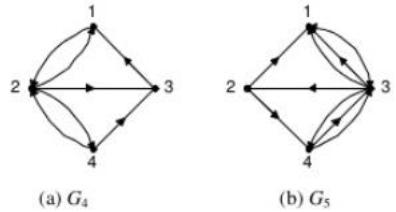
II. DASAR TEORI

A. *Graf*

Graf merupakan sebuah struktur diskrit yang merupakan sebuah representasi dari sebuah objek dan keterhubungan dari objek tersebut. Objek-objek ini dinamakan sebuah simpul, dan simpul-simpul tersebut dihubungkan oleh sebuah garis yang dinamakan sisi. Jika suatu pasangan simpul dihubungkan oleh sebuah sisi maka kedua simpul tersebut disebut sebagai simpul bertetangga.

Secara matematis, graf dapat ditulis menjadi G = (V, E). Dengan G adalah simbol yang merepresentasikan graf, V adalah simbol yang merepresentasikan himpunan simpul-simpul yang tidak kosong, dan E adalah representasi dari himpunan sisi.

Sisi-sisi pada graf dapat memiliki nilai atau bobot ataupun tidak. Nilai tersebut dapat menyatakan jarak, waktu, langkah, dan hal-hal lainnya. Setiap simpul pada graf memiliki derajat yang merupakan representasi dari banyaknya sisi yang dimiliki graf tersebut. Selanjutnya, sisi pada suatu graf dapat memiliki arah ataupun tidak. Jika pada sebuah graf, sisi-sisinya memiliki arah maka graf tersebut dikatakan graf berarah, sedangkan jika suatu graf sisi-sisinya tidak memiliki arah maka graf tersebut dapat dikatakan sebagai graf tidak berarah.



Gambar 2 (a) contoh graf berarah (b) graf ganda berarah

(sumber: Matematika Diskrit oleh Rinaldi Munir, halaman 389)

Dengan adanya arah dan bobot pada suatu graf maka dapat dilakukan algoritma pencarian untuk mencari suatu simpul tujuan pada sebuah graf.

B. *Algoritma Breadth First Search*

Algoritma *Breadth First Search* atau yang lebih sering disingkat menjadi BFS adalah sebuah algoritma pencarian yang dilakukan secara melebar dengan mengunjungi satu persatu simpul yang bertetangga terlebih dahulu oleh simpul tersebut. Selanjutnya akan dicari secara melebar pada graf untuk simpul yang belum dikunjungi sampai seluruh simpul sudah dikunjungi atau menemukan simpul tujuan. Algoritma ini menggunakan *queue* dalam implementasinya.

Dalam algoritma BFS, simpul-simpul anak yang sudah dikunjungi akan disimpan pada suatu *queue*. Selanjutnya simpul tersebut akan diperiksa apakah sudah mencapai simpul tujuan dengan berdasarkan urutan dari *queue*. Sesuai dengan sifat *queue* atau antrian itu sendiri dengan memakai sifat *first in first out*, maka simpul yang disimpan akan disimpan di akhir *queue* sedangkan simpul yang akan diperiksa akan diperiska dari awal *queue*.

C. *Algoritma Depth First Search*

Algoritma *Depth First Search* atau yang lebih sering disingkat menjadi DFS merupakan algoritma lainnya dalam sebuah algoritma pencarian. Algoritma ini melakukan pencarian secara mendalam atau dengan mengunjungi satu persatu simpul anak dari suatu simpul tersebut terlebih dahulu sampai suatu simpul tersebut tidak memiliki anak lagi dan akan dilakukan *backtracking* ke simpul hidup yang masih memiliki anak sampai sebuah solusi ditemukan atau sudah tidak ada simpul hidup lagi untuk diperiksa. Berbeda dengan algoritma BFS yang melakukan pencarian secara ke samping, algoritma DFS melakukan pencarian secara ke bawah. Algoritma ini menggunakan *stack* dalam implementasinya.

Dalam algoritma DFS, simpul-simpul anak yang sudah dikunjungi akan disimpan pada suatu *stack*. Lalu simpul-simpul tersebut akan diperiksa apakah sudah mencapai simpul tujuan dengan berdasarkan urutan dari *stack*. Karena *stack* memiliki sifat *last in first out*, maka simpul yang akan disimpan akan disimpan di awal *stack*. Sedangkan penanganan untuk simpul yang akan diperiksa tetap sama seperti algoritma BFS, yaitu akan diperiksa dari awal *stack*.

Algoritma BFS dan DFS ini merupakan termasuk ke dalam algoritma pencarian dengan menggunakan *Uninformed Cost Search*, sedangkan algoritma selanjutnya yaitu algoritma A* merupakan termasuk algoritma *Informed Cost Search*.

D. *Algoritma A**

Algoritma A* atau yang disebut *A Star* ialah salah satu dari kategori algoritma *Informed Cost Search*. Algoritma A* adalah sebuah algoritma penyempurnaan dari algoritma *Best First Search*. Algoritma ini merupakan algoritma pencarian dengan menggunakan bobot dan fungsi heuristik dari graf untuk mencari solusi yang optimal. Pada algoritma A*, terdapat dua

fungsi yang dibutuhkan untuk mengetahui bobot optimum yang dibutuhkan pada suatu jalur. Fungsi tersebut ialah fungsi g(n) dan h(n). Fungsi g(n) tersebut ialah fungsi yang merupakan penjumlahan bobot-bobot yang didapat dari simpul awal sampai simpul n. Sedangkan, h(n) adalah sebuah fungsi heuristik yang merupakan sebuah bobot perkiraan dari simpul n ke simpul tujuan. Dalam ekskusinya, kedua fungsi tersebut akan digunakan dalam bentuk f(n) = g(n) + h(n). Dengan f(n) tersebut ialah fungsi yang menyatakan nilai bobot apabila memilih jalur dengan simpul n. Penentuan nilai bobot setiap simpul bertujuan agar dapat diketahui nilai optimal bobot suatu simpul untuk nantinya dipilih menjadi simpul selanjutnya. Diharapkan jika dari awal dipilih simpul yang memiliki nilai optimum, maka hasil akhir dari program juga akan menjadi optimum. Pada algoritma A*, implementasi dilakukan dengan memanfaatkan *priority queue*.

Algoritma A* ini akan menyimpan hasil perhitungan bobot setiap kemungkinan simpul dengan fungsi f(n) dan akan disimpan pada *priority queue.* Penyimpanan ini akan dilakukan dengan pengurutan secara otomatis berdasarkan nilai bobotnya. Nilai ini dapat dilakukan dengan terurut membesar atau mengecil sesuai dengan aspek dan optimasi yang diperlukan. Jika ingin mencari nilai bobot terbesar maka dapat dilakukan pengurutan dari yang terbesar ke terkecil, dan jika ternyata yang dicari adalah sebaliknya yaitu nilai bobot terkecil maka pengurutan pun dapat dilakukan dari nilai terkecil ke terbesar.

Pada algoritma A* ini simpul tujuan harus sudah diketahui berada dimana, tidak seperti algoritma BFS dan DFS yang dapat dilakukan pencarian tanpa mengetahui letak dari simpul tujuannya.

E. Peta Digital

Peta digital ialah sebuah representasi dari deskripsi dan informasi suatu daerah dengan sebuah gambar yang dapat diakses secara digital. Saat ini peta digital banyak dikembangkan oleh perusahaan-perusahaan besar. Aplikasi peta digital yang paling terkenal ialah “Google Maps” dan “Waze”.

Pada peta digital ini tidak hanya memuat informasi jalan saja, tetapi fitur-fitur di dalamnya sudah memiliki fitur yang canggih seperti mengetahui apakah jalur tersebut dua arah atau satu arah. Lalu juga terdapat fitur yang dapat mengetahui apakah jalan tersebut kurang nyaman atau rusak, atau sedang ada masalah pada jalan tersebut. Sehingga akan terdapat perhitungan waktu tempuh yang diprediksikan jika memilih jalur tersebut.

Saat ini pilihan pengguna untuk memilih transportasi yang akan digunakan juga disediakan di peta digital, karena waktu tempuh perjalanan setiap transportasi juga akan berbeda antara yang satu dengan yang lainnya. Sehingga fitur ini sangat memudahkan pengguna untuk mengetahui dengan akurat waktu yang akan ditempuh jika menggunakan suatu transportasi.

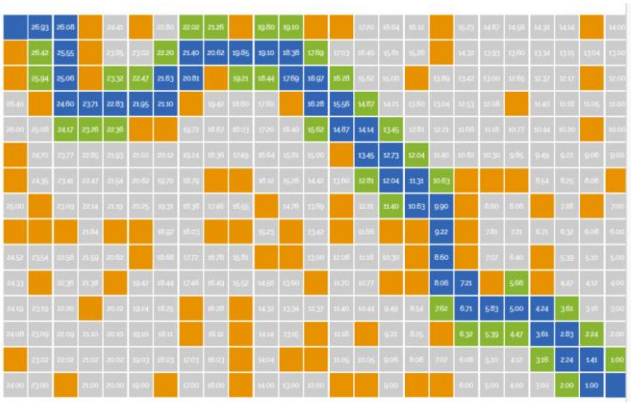
Peta digital pada zaman ini sudah dilengkapi dengan analisis *behavior* dari pengguna yang membuat keakuratan waktu tempuh jalur pun semakin tinggi. Analisis ini mencakup apakah jalur tersebut sedang macet atau tidak yang tentunya akan berpengaruh banyak terhadap waktu tempuh perjalanan pengguna nantinya.

III. IMPLEMENTASI GRAF DAN ALGORITMA PENCARIAN JALUR PADA PENCARIAN WAKTU TEMPUH PERJALANAN TERCAPAT DALAM APLIKASI PETA DIGITAL

A. Implementasi Graf dalam Peta Digital

Pada peta digital, implementasi graf yang dibutuhkan ialah representasi setiap langkah atau persimpangan dengan graf berarah. Agar lebih mudah suatu daerah dapat dibagi menjadi petak-petak persegi yang merepresentasikan langkah-langkah yang harus dilakukan dalam pencarian jalur. Persegi tersebut akan menjadi sebuah simpul dalam graf. Selanjutnya pada setiap persegi tersebut, jika memiliki jalur ke atas, ke bawah, ke kiri, atau ke kananuntuk dilewati maka jalur tersebut akan menjadi sebuah sisi dalam graf yang akan menghubungkan simpul. Jika jalur jalan pada suatu daerah hanya terdapat satu jalur maka pada sisi tersebut akan dimasukkan arah yang sesuai. Jika jalur tersebut dapat dilalui dengan dua arah juga data tersebut dapat dimasukkan ke dalam informasi ketetanggan graf.

Selanjutnya, untuk pencarian bobot yang dimiliki pada setiap sisi pada graf dapat dilakukan dengan mengambil data waktu tempuh normal perjalanan menggunakan suatu jenis transportasi yang ingin dicari. Data ini dapat diambil dengan melihat jarak dari suatu simpul dengan memperhitungkan kecepatan rata-rata dari suatu transportasi. Lalu diambil data seberapa banyak kendaraan yang terdapat pada jalur tersebut sehingga dapat diketahui apakah jalur tersebut macet a1tau tidak dan seberapa cepat dapat melewati jalur tersebut. Perhitungan bobot ini selanjutnya akan dijelaskan dalam pembahasan pada implementasi dengan algoritma A*.



Gambar 3 Contoh pengubahan peta digital ke dalam representasi persegi

(sumber: http://www.algomation.com/)

Representasi graf dalam *source code* yang dibuat akan menggunakan matriks atau *list* sesuai dengan seberapa besar data peta digital yang dibutuhkan. Jika terdapat suatu jalan pada daerah tersebut maka matriks dari koordinat tersebut akan diisi dengan angka 0. Sedangkan, jika tidak terdapat jalan

pada koordinat daerah tersebut maka matriksnya akan diisi dengan angka 1. Selanjutnya simpul awal dan tujuan dapat ditentukan dari input pengguna dan akan direpresentasikan dalam koordinat matriks.

B. *Implementasi Pencarian Jalur dengan Algoritma Pencarian BFS*

Implementasi pencarian jalur dengan algoritma BFS pertama dengan menyimpan simpul awal dan simpul tujuan yang akan digunakan. Selanjutnya, simpul awal dan simpul tujuan itu akan dianalisis menjadi 4 kasus sebagai berikut.

1. Jika simpul tujuan terdapat di sebelah kanan atas dari simpul awal maka pemilihan simpul selanjutnya akan dilakukan dengan mendahulukan memilih jalur ke kanan terlebih dahulu, lalu jalur ke atas, ke bawah dan selanjutnya ke kiri.
2. Jika simpul tujuan terdapat di sebelah kanan bawah dari simpul awal maka pemilihan simpul selanjutnya dilakukan dengan urutan jalur ke kanan, ke bawah,ke atas, dan ke kiri.
3. Jika simpul tujuan berada di sebalah kiri atas dari simpul awal maka pemilihan simpul akan dilakukan dengan prioritas jalur ke kiri, ke atas, ke bawah, ke kanan.
4. Terakhir, jika simpul tujuan berada di sebelah kiri bawah dari simpul awal maka pemilihan simpul akan dilakukan dengan mengurutkan jalur dari ke kiri, ke bawah, ke atas, dan ke kanan.

Pada penulisan makalah ini, penulis akan mengambil asumsi bahwa simpul tujuan akan selalu berada di kanan bawah dari simpul awal. Sehingga algoritma BFS dalam Bahasa Python yang dapat digunakan adalah sebagai berikut.

```python
def BFS(x_awal, y_awal, x_tujuan, y_tujuan):
    global lintasan, peta, queue, crowd
    found = False
    sum_crowd = 0
    x = x_awal
    y = y_awal
    while not found:
        lintasan.append([x,y])
        peta[x][y] = 2 #assign simpul menjadi simpul mati
        sum_crowd += crowd[x][y] #menghitung keramaian jalan
        if x==x_tujuan and y==y_tujuan:
            found = True
        else:
            if (peta[x][y+1] == 0):
                queue.append([x,y+1,deepcopy(lintasan)])
            if (peta[x+1][y] == 0):
                queue.append([x+1,y,deepcopy(lintasan)])
            if (peta[x-1][y] == 0):
                queue.append([x-1,y,deepcopy(lintasan)])
            if (peta[x][y-1] == 0):
                queue.append([x,y-1,deepcopy(lintasan)])
            x = queue[0][0]
            y = queue[0][1]
            lintasan = deepcopy(queue[0][2])
            queue.pop(0)
    for i in range (len(peta)):
        for j in range (len(peta[0])):
            if peta[i][j]==2 and [i,j] not in lintasan:
                peta[i][j] = 3 #membedakan solusi jalur dan bukan
                sum_crowd = sum_crowd - crowd[x][y]
    return len(lintasan)/sum_crowd #rasio
```

Gambar 4 Algoritma BFS

Hasil eksekusi dari algoritma tersebut dengan input koordinat simpul awal adalah (11.4) dan simpul tujuan (27,45) adalah sebagai berikut.

```
Hasil Perhitungan Waktu:
153
```

Gambar 6 Hasil eksekusi dengan algoritma BFS

C. *Implementasi Pencarian Jalur dengan Algoritma Pencarian DFS*

Implementasi pencarian jalur dengan algoritma DFS melakukan langkah awal yang sama dengan pencarian dengan menggunakan algoritma BFS. Pembagian kasus-kasus yang terjadi dan penanganannya juga sama dengan algoritma BFS. Hal yang membedakan ialah dari *source code* yang kan diberikan di bawah ini dengan Bahasa Python dan asumsi penanganan kasus yang terjadi adalah simpul tujuan selalu berada di kanan bawah simpul awal.

```python
def DFS(x_awal,y_awal, x_tujuan, y_tujuan, temp_crowd):
    global lintasan, peta, crowd
    found = False
    x = x_awal
    y = y_awal

    sum_crowd = temp_crowd +  crowd[x][y]
    lintasan.append([x,y])
    peta[x][y] = 2

    if x==x_tujuan and y==y_tujuan:
        return len(lintasan)/sum_crowd
    else:
        if peta[x][y+1] == 0:
            DFS(x,y+1, x_tujuan, y_tujuan, sum_crowd)
            found = True
        elif peta[x+1][y] == 0:
            DFS(x+1,y, x_tujuan, y_tujuan,sum_crowd)
            found = True
        elif peta[x-1][y] == 0:
            DFS(x-1,y, x_tujuan, y_tujuan, sum_crowd)
            found = True
        elif peta[x][y-1] == 0:
            DFS(x,y-1, x_tujuan, y_tujuan, sum_crowd)
            found = True

        if not found:
            lintasan.pop()
            peta[x][y] = 3
            sum_crowd = sum_crowd - crowd[x][y]
            last = lintasan.pop()
            DFS(last[0],last[1], x_tujuan, y_tujuan, sum_crowd)
```

Gambar 7 Algoritma DFS

Hasil eksekusi dari algoritma pencarian DFS dengan input simpul awal dan simpul tujuan sama seperti algoritma BFS, yaitu simpul awal (11,4) dan simpul tujuan (27,45) adalah sebagai berikut.

```
Hasil Perhitungan Waktu:
138
```

Gambar 8 Hasil eksekusi dengan algoritma DFS

*D. Implementasi Pencarian Jalur dengan Algoritma A**

Berbeda dengan dua algoritma sebelumnya, pencarian jalur dengan algoritma A* akan dilakukan analisis fungsi heuristiknya. Pertama tentunya akan disimpan koordinat simpul awal dan simpul tujuan yang diinput dari pengguna. Selanjutnya akan ditentukan fungsi yang akan menentukan nilai bobot setiap simpul. Nilai bobot ini didapatkan dari f(n) + g(n) + h(n).

Pada kasus ini, fungsi g(n) adalah sebagai jumlah langkah atau jarak yang sudah ditempuh dari simpul awal ke simpul n. Dan fungsi heuristik h(n) akan ditentukan dengan perhitungan estimasi jarak dari simpul ke n ke simpul tujuan dengan rumus Jarak Manhattan dibagi dengan estimasi kepadatan jalur sepanjang simpul n ke simpul tujuan.

Rumus Jarak Manhattan yang digunakan ialah rumus perhitungan jarak dengan dua titik dengan rumus sebagai berikut.

$$d_{ij} = |x_i - x_j| + |y_i - y_j|$$

Gambar 9 Rumus Jarak Manhattan

(sumber: http://bisyruljawwad.blogspot.com/2014/04/cara-pengukuran-jarak.html)

Dengan adanya fungsi heuristik dan perhitungan bobot dari simpul awal ke simpul n, diharapkan simpul yang selanjutnya dipilih akan menunjuk ke simpul yang optimal. Jika dari awal langkah sudah memilih simpul yang optimal, diharapkan untuk solusi akhirnya juga akan berakhir dengan optimal.

Pada algoritma A* ini implementasi yang digunakan ialah *priority queue* yang akan selalu mengurutkan bobot yang sudah dihitung. Pengurutan inilah yang menyebabkan hasil dari algoritma A* akan optimal dan akan menemukan tujuan yang optimal pula.

Jika dilihat pada animasi mengenai algoritma A*, ia akan mencari kemungkinan-kemungkinan simpul hidup secara bersamaan, sehingga waktu yang dibutuhkan untuk melakukan pencarian itu pun cenderung lebih sedikit daripada kedua algoritma lainnya. Pencarian ini akan berakhir ketika simpul tujuan sudah berhasil diekspan. Dan lintasan untuk mencapai simpul tujuan itu akan disimpan beserta dengan bobotnnya.

Selanjutnya estimasi kepadatan dapat dihitung dari mengecek kepadatan di simpul n. Sehingga akan didapatkan sebuah *source code* dengan Bahasa Python sebagai berikut.

```python
# Fungsi heuristik
def hn(x_n, y_n, x_tujuan, y_tujuan):
    return (abs(x_n-x_tujuan) + abs(y_n-y_tujuan))/crowd[x][y]

# Mengambil last element list
def takeLast(elem):
    return elem[2]

# Searching dengan A*
def AStar(x_awal, y_awal, x_tujuan, y_tujuan):
    global lintasan, peta,queue, crowd
    found = False
    x = x_awal
    y = y_awal
    sum_crowd = 0
    while not found:
        lintasan.append([x,y])
        peta[x][y] = 2
        sum_crowd += crowd[x][y]
        if x==x_tujuan and y==y_tujuan:
            found = True
        else:
            # fungsi g(n)
            gn = len(lintasan)

            #simpul hidup
            if (peta[x][y+1] == 0):
                queue.append([x,y+1,gn+hn(x,y+1, x_tujuan, y_tujuan),deepcopy(lintasan)])
            if (peta[x+1][y] == 0):
                queue.append([x+1,y,gn+hn(x+1,y, x_tujuan, y_tujuan),deepcopy(lintasan)])
            if (peta[x-1][y] == 0):
                queue.append([x-1,y,gn+hn(x-1,y, x_tujuan, y_tujuan),deepcopy(lintasan)])
            if (peta[x][y-1] == 0):
                queue.append([x,y-1,gn+hn(x,y-1, x_tujuan, y_tujuan),deepcopy(lintasan)])

            #membuat jadi priority queue
            queue.sort(key=takeLast)
            x = queue[0][0]
            y = queue[0][1]
            lintasan = deepcopy(queue[0][3])
            queue.pop(0)
    for i in range (len(peta)):
        for j in range (len(peta[0])):
            if peta[i][j]==2 and [i,j] not in lintasan:
                peta[i][j] = 3
                sum_crowd = sum_crowd - crowd[x][y]
    return len(lintasan)/sum_crowd
```

Gambar 10 Algoritma A*

Berdasarkan algoritma pencarian A* yang sudah penulis buat sedemikian rupa maka hasil eksekusi dengan input simpul awal dan simpul tujuan sama seperti algoritma-algoritma sebelumnya, yaitu simpul awal (11,4) dan simpul tujuan (27,45) adalah sebagai berikut.

```
Hasil Perhitungan Waktu:
183
```

Gambar 11 Hasil eksekusi dengan algoritma A*

IV. PERBANDINGAN HASIL PENCARIAN WAKTU TERCEPAT PADA ALGORITMA DFS, BFS, DAN A*

Berdasarkan hasil yang sudah dilakukan dan di*test* oleh penulis, ketiga algoritma ini memiliki jawaban yang berbeda. Keluaran dari setiap fungsi algoritma ialah perhitungan jarak dibagi dengan jumlah kepadatan kendaraan. Sehingga semakin padat kendaraan akan memakan waktu yang semakin banyak untuk melewati jalur tersebut. Maka hasil yang dibandingkan ialah jika hasil keluaran dari setiap fungsi algoritma adalah besar, maka semakin cepat waktu tempuh yang dilewati dengan menggunakan jalur tersebut. Sedangkan, jika keluaran yang dihasilkan adalah kecil, maka waktu tempuh dengan jalur tersebut termasuk ke dalam jalur dengan waktu tempuh yang lambat. Sehingga hasil eksekusi dari ketiga algoritma tersebut dapat dibandingkan untuk memperoleh hasil.

Pada algoritma BFS, hasil keluaran yang didapat ialah sebanyak 153. Pada algoritma DFS, hasil keluaran yang

diperoleh adalah sebanyak 138 dan pada algoritma A* hasil yang diperoleh ialah mencapai angka 183. Jika ketiga angka tersebut diurutkan dari yang terbesar ke terkecil maka didapatkan hasil urutan algoritma menjadi algoritma A*, algoritma BFS, dan algoritma DFS.

Pada kasus pencarian dengan tujuan untuk mendapatkan nilai optimum, algoritma A* lebih baik untuk digunakan karena algoritma ini termasuk ke dalam algoritma *Informed Cost Search* yang memperhitungkan bobot nilai pada setiap langkah agar hasil yang didapatkan di akhir algoritma juga optimal. Sedangkan pada algoritma pencarian dengan *Uninformed Cost Search*, bobot jarak dan kepadatan tidak dihitung pada saat pemilihan langkah. Pemilihan langkah pada algoritma BFS dan DFS dilakukan secara konsisten tergantung kepada posisi simpul awal dan simpul tujuan yang diberikan dari input pengguna.

Pada kasus algoritma A*, fungsi heuristik haruslah sangat diperhatikan karena jika fungsi heuristik berganti maka hasil pun juga akan berganti. Maka penting sekali untuk memilih fungsi heuristik yang paling tepat untuk digunakan dalam suatu masalah. Dalam hal ini penulis memutuskan fungsi heuristik yang optimal adalah seperti yang sudah dijelaskan pada bab 3.

Algoritma pencarian dengan *Uninformed Cost Search* tidak cocok digunakan pada hal-hal yang menyangkut jarak dan optimasi karena simpul yang dipilih pada setiap langkahnya tidak memperhitungkan perhitungan bobot atau jarak yang sangat krusial untuk penentuan simpul ekspan yang akan dipilih selanjutnya. Algoritma BFS dan DFS lebih cocok untuk pencarian yang tidak memerlukan bobot yang optimal dalam pengerjaannya atau tidak memiliki bobot sama sekali.

Hasil dari perbandingan ketiga algoritma ini ialah algoritma A* merupakan algoritma yang paling efisien dan efektif untuk mencari jalur dengan waktu tempuh tercepat untuk suatu perjalanan dalam peta digital.

V. KESIMPULAN

Algoritma A* merupakan algoritma yang paling optimal dalam pencarian jalur dengan waktu tercepat dalam suatu perjalanan pada aplikasi peta digital. Hal ini dikarenakan algoritma A* memiliki bobot yang dihasilkan dari perhitungan tambah antara fungsi h(n) dan fungsi g(n). Fungsi heuristik yang cocok dalam algoritma ini ialah dengan mencari jarak manhattan dari simpul ke-n sampai simpul tujuan dan membaginya dengan kepadatan kendaraan di simpul ke-n. Sedangkan fungsi g(n) yang digunakan ialah fungsi untuk menghitung jumlah jarak yang sudah ditempuh dari simpul awal ke simpul n.

UCAPAN TERIMA KASIH

Ucapan terima kasih dan syukur penulis sampaikan sebanyak-banyaknya kepada Allah SWT yang sudah memberikan nikmat dan rezeki untuk melancarkan pembuatan makalah ini. Penulis juga mengucapkan terima kasih kepada Bu Ulfa sebagai dosen mata kuliah Strategi Algoritma Kelas 1 yang sudah memberikan ilmu yang sangat bermanfaat untuk penyelesaian makalah ini. Tidak lupa penulis ucapkan terima kasih kedapa kedua orang tua dan teman-teman penulis yang sudah mendukung penulis dalam penyelesaian makalah ini baik dalam bentuk *support* dan doa. Dan terakhir kepada seluruh pihak yang sudah membantu penulis dalam mengerjakan makalah ini, penulis ucapkan terima kasih sebanyak-banyaknya.

REFRENSI

[1] Ledederman, Florian. 2018. Analysing Digital Maps Online: A Reverse Engineering Approach. Vienna: Research Group Cartography.

[2] Levitin, Anany. 2011. Introduction to The Design and Analysis of Algorithms. London: Pearson.

[3] Munir, Rinaldi. 2018. Diktat Kuliah IF2211 Strategi Algortima. Program Studi Teknik Informatika ITB.

[4] Munir, Rinaldi. 2009. Matematika Diskrit, Bandung: Informatika Bandung.

[5] Trudeau, Richard J. (1993). Introduction to Graph Theory (Corrected, enlarged republication. ed.). New York: Dover Pub.

[6] http://www.algomation.com/. Diakses pada 25 April 2019.

[7] http://bisyruljawwad.blogspot.com/2014/04/cara-pengukuran-jarak.html. Diakses pada 25 April 2019.

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, 26 April 2019

Anissa Putri Dinanti
13517121