Peminimalisasian Total Ongkos pada Aplikasi Transportasi Online Menggunakan Algoritma A*

Erfandi Suryo Putra 13515145
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515145@std.stei.itb.ac.id

***Abstrak*—Pada zaman ini banyak orang yang menggunakan *smartphone*. *Smartphone* memiliki banyak fungsi sehingga bisa digunakan untuk banyak hal sehingga sudah menjadi bagian kehidupan banyak orang. Salah satu fungsi *smartphone* diimplementasikan dengan aplikasi transportasi online. Penentuan biaya dari aplikasi transportasi online adalah dengan mengukur jarak yang akan dilalui. Agar mendapat biaya termurah, jarak yang dilalui harus merupakan jarak terpendek. Hal tersebut dapat diakukan dengan algoritma A*.**

***Kata Kunci*—algoritma, *cost*, graf, jarak, simpul, heuristik.**

I. PENDAHULUAN

Di zaman modern ini kita hampir tidak bisa lepas dari teknologi. Teknologi, terutama teknologi digital, hampir selalu ada dalam segala aspek kehidupan modern manusia. Salah satu contoh teknologi digital yang sudah ada antara lain adalah internet, komputer, smartphone, media social, dll. Kita dapat menggunakan teknologi digital dalam berbagai hal, seperti untuk berkomunikasi, melakukan suatu pekerjaan, mencari informasi atau berita, bahkan sekarang sudah banyak sekolah-sekolah yang menggunakan teknologi digital dalam kegiatan belajar-mengajar. Dalam beberapa hal, orang-orang lebih memilih menggunakan teknologi digital dibanding menggunakan barang atau teknologi yang sebelumnya sudah mereka miliki karena kelebihan-kelebihan yang dimiliki teknologi digital, diantaranya adalah dalam mencari informasi, kita dengan sangat mudah dapat dengan mudah mencari segala informasi dari berbagai sumber melalui internet, selain kemudahan dalam mencari informasi, informasi baru, misalkan berita, juga bisa langsung diakses. Kelebihan lain dari teknologi digital adalah penggunaannya yang jauh lebih mudah, contohnya penggunaan smartphone yang bisa digunakan di mana saja untuk berbagai hal seperti berkomunikasi melalui media sosial atau melalui chatting, untuk menikmati hiburan seperti menonton film, mendengarkan musik, dll., *smartphone* juga bisa digunakan untuk membantu maupun mengerjakan suatu pekerjaan.

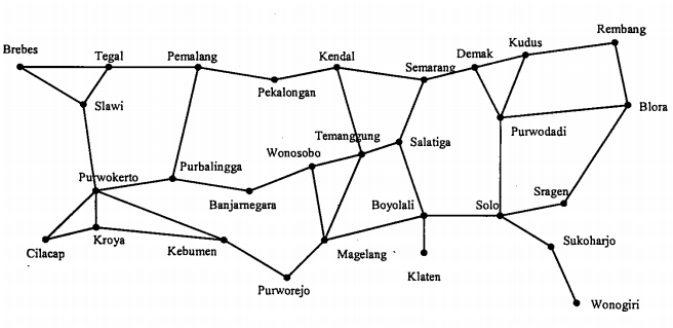
Sebelum adanya *smartphone*, untuk berpergian dari suatu tempat ke tempat lainnya, seseorang harus memiliki kendaraan pribadi agar dapat berpergian dengan mudah. Jika tidak memiliki kendaraan pribadi, orang tersebut harus berpergian melalui transportasi umum, seperti taksi, bis, angkot, bajaj, ojek, dll. Akan tetapi, fasilitas transportasi umum di Indonesia kurang baik sehingga mempersulit kegiatan berpergian seseorang. Kesulitan tersebut dapat disebabkan antara lain seperti sulitnya mendapat transportasi yang tersedia, biaya yang tidak pasti, waktu yang sulit diperkirakan karena kendaraan yang sering berhenti untuk menunggu penumpang, kewajiban untuk berjalan ke tempat transportasi umum tersedia, dll. Dengan adanya *smartphone*, seseorang yang tidak memiliki kendaraan pribadi dapat dengan mudah berpergian dari mana saja ke mana dengan harga yang pasti dan waktu yang sesuai perkiraan. Untuk mendapat harga yang pasti tersebut, aplikasi dapat mengukurnya melalui jarak yang akan dilalui dari tempat asal ke tempat tujuan.

Sebagai pengembang aplikasi, aplikasi yang dibuat harus bermanfaat bagi semua pihak. Aplikasi tidak boleh merugikan pengguna, dan juga tidak merugikan organisasi yang menjalankan bisnis. Oleh karena itu harga yang harus dibayar pengguna disesuaikan dengan penggunaan masing-masing. Selain itu aplikasi tidak boleh dengan sengaja memberikan harga yang mahal di saat terdapat alternative dengan harga lebih murah. Karena harga disesuaikan dengan jarak harus dilalui, aplikasi harus dapat memilih rute perjalanan terdekat dari posisi pengguna ke tempat tujuan.

Untuk dapat memilih rute perjalanan terdekat, aplikasi harus dapat mengidentifikasi tempat-tempat yang ada di wilayah tersebut, aplikasi juga harus mengetahui lintasan-lintasan yang dapat dilalui dan panjang setiap lintasan. Peta juga harus mengetahui data geografis suatu wilayah. Data geografis diperlukan jika lokasi pengguna sedang tidak berada di jalan yang tercatat di peta. Walaupun peta tidak mendeteksi suatu jalan, peta juga bisa mendeteksi jalan yang bisa dilalui pengguna. Hal ini diperlukan untuk menentukan apakah terdapat transportasi yang dapat dengan mudah diakses oleh penumpang yang melakukan pesanan. Hal ini juga diperlukan, terutama di Indonesia, karena Indonesia merupakan negara yang luas dan setiap daerahnya memiliki karateristik geografis yang berbeda-beda sehingga data geografis diperlukan untuk pemilihan retu perjalanan yang efektif dan aman.

II. GRAF

Graf digunakan untuk mempresentasikan suatu objek diskrit dan hubungan antar objek-objek tersebut. Secara visual, suatu objek di graf direpresentasikan dengan sebuah titik, dan suatu hubungan antar objek direpresentasikan dengan sebuah garis yang menghubungkan kedua titik. Graf bisa digunakan untuk mepresentasikan berbagai hal, salah satu kegunaan graf adalah untuk mempresentasikan sebuah peta.



Gambar 1. Graf Peta

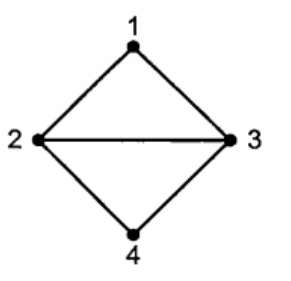
Pada gambar peta diatas, masing-masing kota direpresentasikan dengan sebuah titik dan lintasan-lintasan yang menghubungkan antar kota direpresentasikan dengan sebuah garis. Dengan adanya graf yang mempresentasikan suatu peta seperti graf di atas, kita dapat melihat apakah terdapat lintasan antara dua kota, selain itu kita juga dapat melihat panjang lintasan-lintasan antar kota, dan dengan diketahuinya panjang-panjang lintasan antar kota, kita juga dapat menentukan lintasan terpendek yang dapat dilalui dari satu kota ke kota lainnya.

Secara definisi, graf merupakan pasangan himpunan (V, E) dan dapat dinotasikan dengan G = (V, E) dengan V merupakan himpunan tidak kosong dari simpul-simpul yang mempresentasikan objek-objek, dan E merupakan merupakan himpunan sisi yang mempresentasikan hubungan antar objek atau simpul. Dari definisi tersebut dapat disimpulkan bahwa sebuah graf harus memiliki minimal satu buah objek atau simpul, tetapi boleh sama sekali tidak mempunyai sisi, yang berarti objek-objek yang dipresentasikan dengan simpul-simpul tidak mempunyai hubungan.

Menurut sudut pandang pengelompokannya, graf dapat dikelompokkan menjadi tiga kategori. Pengelompokan graf ini didasarkan pada ada tidaknya sisi ganda atau gelang, berdasarkan jumlah simpul, dan berdasarkan orientasi arah sisi.

Berdasarkan ada tidaknya gelang atau sisi ganda, graf dibagi menjadi dua jenis, yaitu graf sederhana dan graf tidak sederhana.

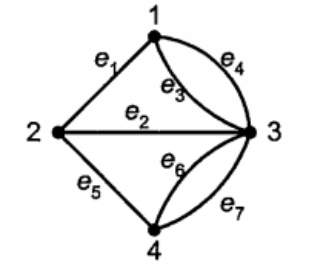
1. Graf Sederhana



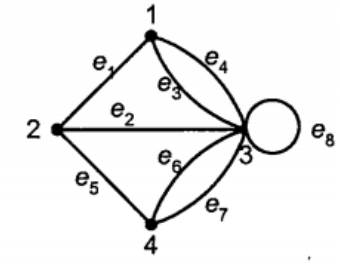
Gambar 2. Graf Sederhana

Graf sederhara merupakan graf yang tidak mempunyai sisi ganda maupun gelang

2. Graf Tidak Sederhana



Gambar 3. Graf Ganda



Gambar 4. Graf Semu

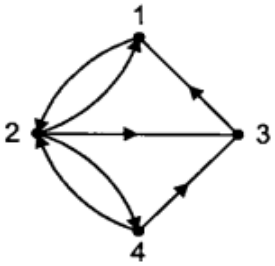
Graf tidak sederhana merupakan graf yang memiliki sisi ganda atau gelang. Graf tidak sederha juga dibagi menjadi dua, yaitu graf ganda dan graf semu. Graf ganda adalah graf yang mempunyai sisi ganda sementara graf semu adalah graf yang mempunyai gelang, walaupun juga mempunyai sisi ganda.

Berdasarkan orientasi sisinya, graf dibagi menjadi dua jenis yaitu tidak berarah dan graf berarah.

1. Graf Tidak Berarah

Graf tidak berarah adalah graf yang sisinya tidak mempunyai orientasi arah. Pada graf berarah, (u, v) dan (v, u) merupakan dua sisi yang sama.

2. Graf Berarah



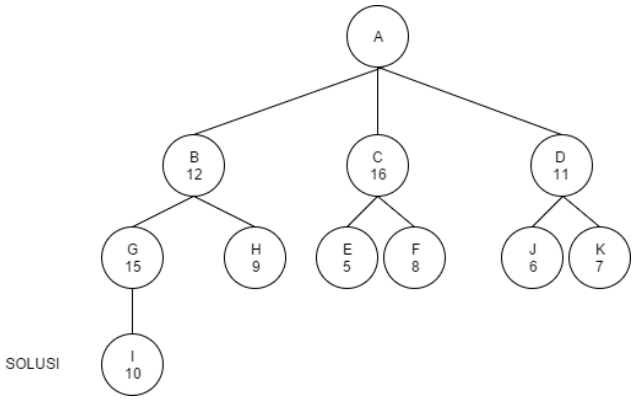
Gambar 5. Graf Berarah

Graf berarah adalah graf yang setiap sisinya mempunyai orientasi arah. Pada graf berarah, (u, v) dan (v, u) merupakan dua sisi yang berbeda).

III. BRANCH AND BOUND

Algoritma Branch and Bound adalah salah satu algoritma yang dapat digunakan untuk menyelesaikan suatu persoalan optimasi dengan meminimalkan atau memaksimalkan suatu fungsi objektif yang tidak melanggar batasan persoalan. Algoritma Branch and Bound mirip dengan algoritma Breadth-first search. Perbedaan algoritma ini dengan algoritma BFS biasa, pada algoritma BFS biasa, pembangkitan simpul-simpul dari simpul berikutnya yang akan diekspansi diurutkan berdasarkan urutan pembangkitannya (FIFO), tetapi pada branch and bound pembangkitan dilakukan berdasarkan nilai *cost*. Pada algoritma Branch and Bound, setiap simpul diberi sebuah nilai cost, yaitu nilai taksiran termurah ke simpul tujuan yang melalui simpul tersebut, dan simpul berikutnya yang diekpansi tidak lagi berdasarkan urutan pembangkitannya, tetapi berdasarkan *cost* yang paling kecil pada kasus minimasi, atau berdasarkan *cost* yang paling besar pada kasus maksimasi.

Contoh penyelesaian masalah dengan algoritma Branch and Bound dengan urutan pembangkitan simpul berdasarkan *cost* terbesar adalah sebagai berikut



Gambar 6. Pohon penyelesaian Branch and Bound

1. Dari simpul awal A, bangkitkan semua simpul anaknya, yaitu B, C, dan D
2. Pilih simpul dengan cost terbesar, yaitu C
3. Bangkitkan semua simpul anak dari simpul C, yaitu E dan F
4. Dari semua simpul-simpul yang sudah dibangkitkan dan belum diekspansi, yaitu B, E, dan F, pilih simpul dengan cost terbesar, yaitu simpul B
5. Lakukan ekspansi simpul-simpul sampai didapatkan simpul dengan cost terbesar yang merupakan simpul solusi

Selain digambar sebagai pohon, persoalan Branch and Bound juga bisa diselesaikan dengan cara direpresentasikan dengan antrian prioritas yang diurutkan mengecil berdasarkan cost-nya

Simpul	Simpul Hidup
A	C, B, D
C	B, D, F, E
B	G, D, H, F, E
G	D, I, H, F, E
D	I, H, F, K, J, E
I	H, F, K, J, E

Tabel 1. Penyelesaian Branch and Bound

I merupakan simpul solusi, jadi solusinya adalah

A -> B -> G -> I

Di bawah ini adalah *pseudocode* untuk algoritma Branch and Bound menggunakan antrian prioritas yang menghasilkan simpul solusi

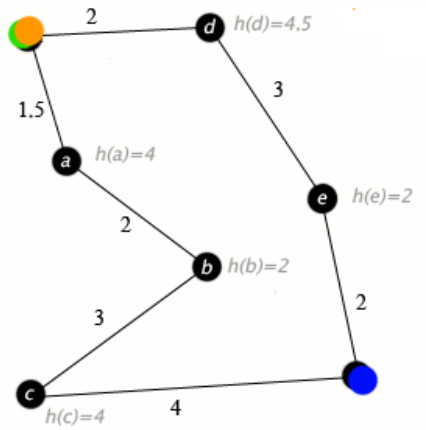
```
PrioQueue Q
Q.enqueue(Awal)
Simpul hasil = NULL
WHILE NOT Q.IsEmpty OR hasil != solusi
    Simpul S = Q.dequeue()
    IF S = solusi THEN
        hasil = S
    ELSE
        FOR semua simpul anak
            Q.enqueue(simpul anak)
RETURN hasil
```

IV. ALGORITMA A*

Algoritma A* merupakan salah satu bentuk algoritma Branch and Bound untuk mencari jalur dengan jarak terpendek dari satu simpul ke simpul lainnya. Algoritma ini pertama kali ditemukan oleh Peter Hart, Nils Nilsson, dan Bertram Raphael pada tahun 1968. Dalam algoritma ini, *cost* setiap simpul merupakan jarak yang telah dilalui ditambah nilai heuristik simpul tersebut. Nilai heuristik digunakan untuk memperkirakan *cost* suatu simpul. Dengan memperhitungkan nilai heuristik suatu simpul, algoritma A* dapat digunakan

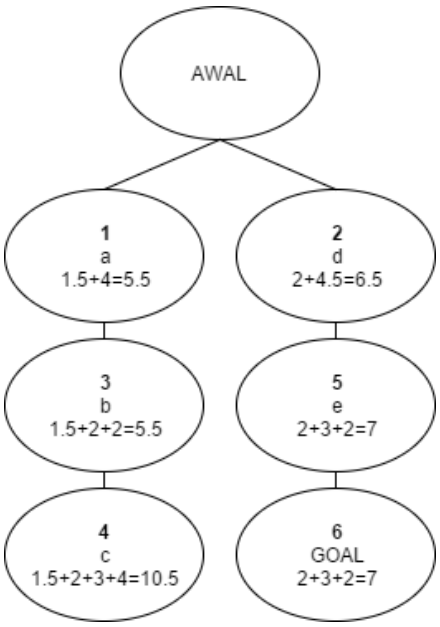
mencari jalur terpendek dengan pencarian yang lebih cepat karena dengan memperhitungkan nilai heuristik, algoritma A* menghindari ekspansi simpul-simpul yang mahal atau simpul-simpul yang tidak diperlukan. Penentuan nilai heuristik tidak dapat ditentukan secara sembarangan, syarat agar suatu nilai heuristik diterima adalah nilai heuristik tidak boleh lebih besar daripada nilai sesungguhnya untuk mencapai simpul tujuan.

Di bawah ini adalah contoh pencarian jalur terpendek dengan menggunakan algoritma A*



Gambar 7. Contoh Persoalan

Misalkan diperlukan jalur terpendek dari titik kuning ke titik ungu, simpul induk adalah saat berada di titik kuning dan simpul tujuan adalah saat berada di simpul ungu. Cara penyelesaiannya sama dengan cara menyelesaikan algoritma Branch and Bound dengan *cost* = c(n) + h(n), c(n) merupakan jarak yang telah dilalui untuk mencapai titik n dan h(n) merupakan nilai heuristik titik n. Karena yang diperlukan adalah jalur terpendek, maka simpul yang diambil adalah simpul dengan *cost* terkecil. Penyelesaian persoalan di atas adalah sebagai berikut



Gambar 8. Pohon Penyelesaian A*

Persoalan ini juga diselesaikan dengan cara direpresentasikan dengan antrian prioritas yang diurutkan mengecil berdasarkan *cost*-nya

Simpul	Simpul Hidup
AWAL	a, d
A	b, d
B	d, c
D	e, c
E	GOAL, c
GOAL	C

Tabel 2. Penyelesaian A*

GOAL merupakan simpul solusi, jadi solusinya adalah

AWAL -> d -> e -> GOAL

V. PEMBAHASAN

Algoritma A* dapat digunakan meminimalisasikan biaya yang harus dibayar pengguna dalam menggunakan transportasi online. Karena harga ditentukan dengan jauhnya perjalanan, maka untuk mendapatkan harga termurah, perjalanan yang dilalui pengguna harus melalui rute perjalanan terpendek. Algoritma A* digunakan dalam mencari jalur terpendek dari lokasi pengguna aplikasi ke lokasi yang dituju. Karena dalam menentukan harga perhitungan hanya dilakukan satu kali, yaitu saat memesan, pencatatan lokasi dan pencarian rute dengan jarak terdekat tidak harus dilakukan setiap saat.

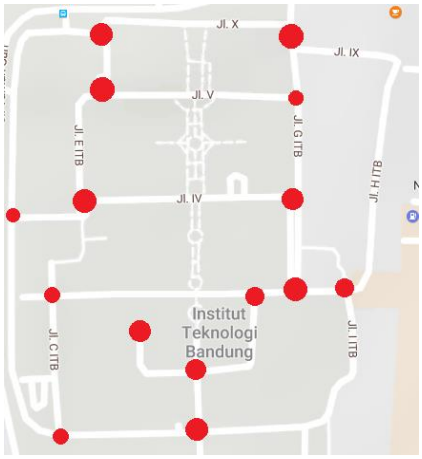
Untuk mencari jarak terdekat pada sebuah peta, peta harus direpresentasikan dengan sebuah graf. Simpul-simpul di graf tersebut merupakan representasi persimpangan-persimpangan di wilayah peta tersebut dan sisinya merupakan representasi jalan yang dapat dilalui. Sisi-sisi graf ini harus merupakan sisi berbobot yang nilainya mempresentasikan panjang lintasan.

Di bawah ini adalah contoh kasus pencarian rute dengan jarak terpendek di kampus ITB.

Gambar 9. Peta ITB

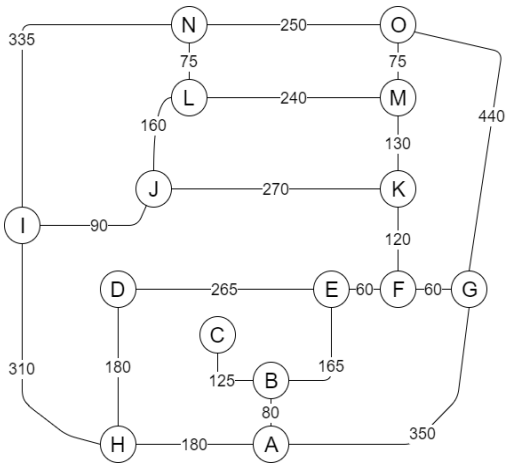
Peta tersebut perlu direpresentasikan dalam bentuk graf dan menyediakan informasi yang cukup untuk mencari rute terpendek, hal tersebut dapat dilakukan dengan langkah-langkah sebagai berikut:

1. Setiap persimpangan direpresentasikan oleh sebuah simpul



Gambar 10. Simpul Peta ITB

2. Sambungkan setiap simpul dengan garis yang memiliki bobot yang mempresentasikan jarak antar persimpangan-persimpangan tersebut. Dalam contoh ini dimisalkan setiap jalur dapat dilalui melalui dua arah sehingga peta direpresentasikan dengan graf tidak berarah



Gambar 11. Graf Peta ITB

Graf perlu direpresentasikan dalam bentuk matriks agar dapat dibaca program

3. *Cost* simpul dalam algoritma A* adalah jarak yang telah dilalui ditambah nilai heuristik, maka nilai heuristik diperlukan, dalam contoh ini nilai heuristiknya adalah jarak garis lurus ke simpul tujuan yang dihitung dari koordinat masing-masing titik ke koordinat titik tujuan, misalnya simpul tujuannya adalah simpul M.

Simpul	Jarak ke M
A	450
B	375
C	360
D	405
E	260
F	250
G	260
H	530
I	390
J	300
K	130
L	240
M	0
N	265
O	75

Tabel 3. Nilai Heuristik

Nilai heuristik tersebut dapat diterima karena jarak garis lurus simpul ke simpul tujuan tidak mungkin lebih besar dari jarak terkecil sebenarnya dari simpul tersebut ke simpul tujuan. contohnya jarak garis lurus N ke M adalah 265, tetapi jarak sebenarnya yang harus melalui simpul L terlebih dahulu adalah 75 + 240 = 310, karena 265 < 310, penentuan nilai heuristik tersebut dapat diterima.

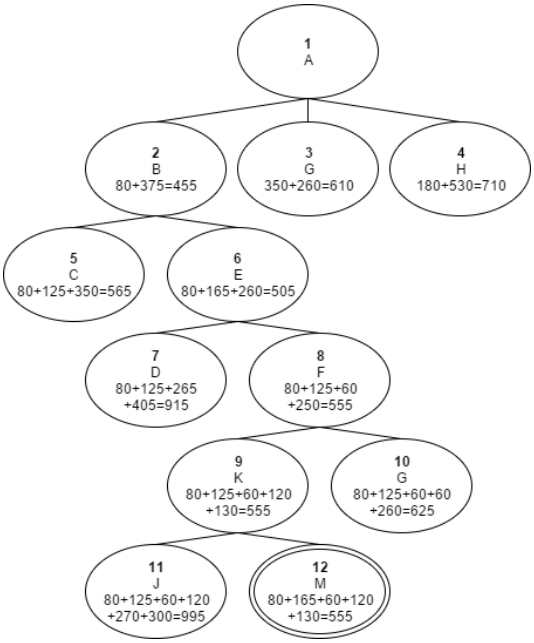
Lokasi keberangkatan atau tujuan mungkin saja tidak berada tepat di persimpangan, sehingga perlu ditambah simpul tambahan yang menandai posisi awal atau tujuan tersebut. Jika simpul tambahan tersebut berada di tengah jalan, maka simpul tersebut akan membagi suatu sisi menjadi dua sisi yang dibagi oleh simpul tambahan tersebut. Jika simpul tambahan tersebut tidak berada di tengah jalan, simpul tersebut perlu dihubungkan dengan jalan tambahan ke jalan terdekat sehingga akan terbentuk simpul baru lagi yang berada di tengah jalan dan sisi tambahan yang menghubungkan simpul awal ke sisi yang dihubungkan.

Di bawah ini adalah contoh penyelesaian rute terdekat yang dimulai dari simpul A ke simpul M

1. Simpul-simpul yang diekspansi dari simpul A adalah simpul B, H, dan G
2. Hitung *cost* masing – masing simpul, yaitu jarak yang telah dilewati ditambah jarak garis lurus dari simpul ke M (dari tabel nilai heuristik)
    a. B: 80 + 375 = 455
    b. H: 180 + 530 = 710
    c. G: 350 + 260 = 610
3. Masukan B, H, dan G ke dalam antrian yang diurutkan membesar berdasarkan *cost*
4. Ambil simpul terdepan dari antrian, yaitu simpul B

5. Ulangi sampai simpul yang terambil adalah simpul tujuan, yaitu simpul M

   Persoalan dapat diselesaikan dengan penyelesaian menggunakan *tree*



Gambar 12. Pohon Penyelesaian

Persoalan juga dapat diselesaikan dengan antrian prioritas

Simpul	Simpul Hidup
A	B_{455} G_{610} H_{710}
B	E_{505} C_{565} G_{610} H_{710}
E	F_{555} C_{565} G_{610} H_{710} D_{915}
F	K_{555} C_{565} G_{610} G_{625} H_{710} D_{915}
K	M_{555} C_{565} G_{610} G_{625} H_{710} D_{915} J_{995}
M	C_{565} G_{610} G_{625} H_{710} D_{915} J_{995}

Tabel 4. Penyelesaian Persoalan

M didapatkan dari antrian sebagai simpul dengan cost terkecil, sehingga solusi sudah ditemukan dan solusi yang didapat adalah

A -> B -> E -> F -> K -> M

Dari algoritma A* tersebut, aplikasi akan menampilkan jalur terpendek dengan hasil seperti di bawah ini

Gambar 13. Solusi Jalur Terpendek

Setelah aplikasi mendapatkan jalur dengan jarak terpendek, aplikasi menghitung harga yang perlu dibayar pengguna. Penghitungan harga dilakukan berdasarkan tarif yang ditentukan. Agar harga yang harus dibayar bulat untuk memudahkan pembayaran dan karena harga yang dibayar pengguna tidak boleh membuat perusahaan rugi, hasil pembagian jarak yang didapat dengan pembagi jarak dibulatkan ke atas terlebih dahulu sebelum dikalikan dengan tarif.

Misalkan tarif pembayaran adalah 2000 setiap 200 meter, maka harga yang harus dibayar adalah 2000 * (555/200). 555/200 menghasilkan bilangan desimal, yaitu 2.775, maka hasil pembagian tersebut dibulatkan menjadi 3, dan harga yang harus dibayar adalah 2000 * 3, yaitu 6000.

VI. IMPLEMENTASI

Program harus mengetahui posisi koordinat masing-masing titik yang akan disimpan sebagai objek Point. Dalam contoh ini posisi masing-masing titik disimpan dalam sebuah file ekternal. File eksternal juga menyimpan jarak masing-masing titik dalam bentuk matriks

- Nilai koordinat masing-masing titik

```
500,700
500,620
435,520
325,520
565,520
625,520
685,520
325,700
225,490
335,470
625,470
375,340
625,340
375,265
625,265
```

- Matriks jarak

```
9999,80,9999,9999,9999,9999,350,180,9999,9999,9999,9999,9999,9999,9999,9999
80,9999,165,9999,165,9999,9999,9999,9999,9999,9999,9999,9999,9999,9999,9999
9999,165,9999,110,150,9999,9999,9999,9999,9999,9999,9999,9999,9999,9999,9999
9999,9999,110,9999,9999,9999,9999,180,9999,9999,9999,9999,9999,9999,9999,9999
9999,165,150,9999,9999,60,9999,9999,9999,9999,9999,9999,9999,9999,9999,9999
9999,9999,9999,9999,60,9999,60,9999,9999,9999,120,9999,9999,9999,9999,9999
350,9999,9999,9999,9999,60,9999,9999,9999,9999,9999,9999,9999,9999,440
180,9999,9999,180,9999,9999,9999,9999,310,9999,9999,9999,9999,9999,9999,9999
9999,9999,9999,9999,9999,9999,9999,9999,310,9999,90,9999,9999,9999,335,9999
9999,9999,9999,9999,9999,9999,9999,9999,9999,90,9999,270,160,9999,9999,9999
9999,9999,9999,9999,9999,9999,120,9999,9999,9999,270,9999,9999,130,9999,9999
9999,9999,9999,9999,9999,9999,9999,9999,9999,9999,160,9999,9999,240,75,9999
9999,9999,9999,9999,9999,9999,9999,9999,9999,9999,9999,130,240,9999,9999,75
9999,9999,9999,9999,9999,9999,9999,9999,9999,335,9999,9999,75,9999,9999,250
9999,9999,9999,9999,9999,9999,9999,440,9999,9999,9999,9999,9999,75,250,9999
```

- Kelas Point

```
public class Point {
    public int x;
    public int y;
    public int id;
    public float cost;

    public Point() {
        x = 0;
        y = 0;
        id = -1;
        cost = 0;
    }

    public Point(int px, int py, int idx) {
        x = px;
        y = py;
        id = idx;
        cost = 0;
    }

    public Point(Point p, float cost) {
        x = p.x;
        y = p.y;
        id = p.id;
        this.cost = cost;
    }
}
```

Setelah nilai-nilai yang dibutuhkan disimpan, pengguna dapat memasukkan tempat berangkat dan tempat tujuan. Dari masukan pengguna tersebut, aplikasi dapat memberikan harga termurah ke pengguna dengan memilih jarak terdekat melalui algoritma A*

```
System.out.println("Tempat berangkat: ");
int start = reader.nextInt();
System.out.println("Tempat tujuan: ");
int goal = reader.nextInt();
reader.close();

// A*
PriorityQueue<Point> pq = new PriorityQueue<>(Nodes.size(), pointComparator);
pq.add(Nodes.get(start));
Map<Integer, Integer> cameFrom = new HashMap<Integer, Integer>();
Map<Integer, Float> costSoFar = new HashMap<Integer, Float>();
cameFrom.put(start, 0);
costSoFar.put(start, 0f);

while (!pq.isEmpty()) {
    Point curPoint = pq.remove();

    if (curPoint.id == Nodes.get(goal).id) {
        break;
    }

    for (int i=0; i<Nodes.size(); i++) {
        if (i==curPoint.id)
        {
            continue;
        }
        Point nextPoint = Nodes.get(i);
        float newCost = costSoFar.get(curPoint.id) + O[curPoint.id][nextPoint.id];
        if (!costSoFar.containsKey(nextPoint.id) || newCost < costSoFar.get(nextPoint.id))
        {
            costSoFar.put(nextPoint.id, newCost);
            // hitung heuristik
            float dist = (float) Math.sqrt((curPoint.x-nextPoint.x) * (curPoint.x-nextPoint.x)
                + (curPoint.y-nextPoint.y) * (curPoint.y-nextPoint.y));
            float f = newCost + dist;
            pq.add(new Point(nextPoint, f));
            cameFrom.put(nextPoint.id, curPoint.id);
        }
    }
}
```

Setelah algoritma mencapai titik tujuan, setiap titik yang merupakan solusi dimasukkan ke dalam list, lalu aplikasi menghitung dan menampilkan harga yang harus dibayar dan menampilkan jalur yang harus dilalui yang terdiri dari titik-titik solusi

- Menghitung harga dan menampilkan jalur dan harga

```
int idx = goal;
Result.add(Nodes.get(idx));
while (idx != start) {
    idx = cameFrom.get(idx);
    Result.add(Nodes.get(idx));
}

harga = 2000 * ((int) Math.ceil(costSoFar.get(goal)/200));
System.out.print("\nTotal harga: ");
System.out.println(harga);

JFrame frame = new JFrame("Jalur");
frame.setSize(1000, 1000);
Main panel = new Main();
frame.add(panel);
frame.setVisible(true);
```

- Prosedur untuk menampilkan jalur

```
public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    int i;
    i = 0;
    while (i < Nodes.size()) {
        g2.setStroke(new BasicStroke(10));
        int x = Nodes.get(i).x;
        int y = Nodes.get(i).y - 100;
        g2.drawLine(x, y, x, y);
        int j = 0;
        while (j < Nodes.size()){
            if (O[i][j] != 9999){
                int x1 = Nodes.get(j).x;
                int y1 = Nodes.get(j).y - 100;
                g2.setStroke(new BasicStroke(1));
                g2.drawLine(x, y, x1, y1);
            }
            j++;
        }
        i++;
    }

    g2.setColor(Color.RED);
    g2.setStroke(new BasicStroke(2));
    i = 1;
    while (i < Result.size()) {
        int x1 = Result.get(i-1).x;
        int y1 = Result.get(i-1).y - 100;
        int x2 = Result.get(i).x;
        int y2 = Result.get(i).y - 100;
        g2.drawLine(x1, y1, x2, y2);
        i++;
    }
}
```

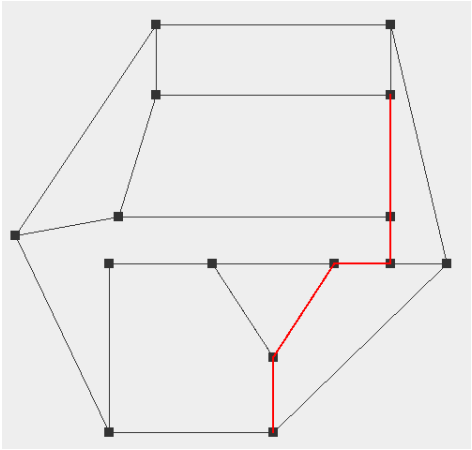
Berikut ini adalah contoh penggunaan dan hasil yang ditampilkan ke pengguna

- Total harga

```
Tempat berangkat:
0
Tempat tujuan:
12

Total harga: 6000
```

- Jalur terpendek



KESIMPULAN

Untuk dapat memberikan harga termurah yang harus dibayar pengguna saat menggunakan transportasi online tanpa merugikan perusahaa, bisa dengan cara mencari jalur dengan jarak terdekat jika harga ditentukan melalui jarak. Penentuan jarak terdekat tersebut dapat menggunakan algoritma A*. Dalam mencari jarak terdekat dengan algoritma A*, peta harus direpresentasikan dengan sebuah graf. Simpul-simpul di graf tersebut merupakan representasi persimpangan-persimpangan wilayah peta tersebut dan sisinya merupakan representasi lintasan antar persimpangan. Sisi-sisi graf ini harus merupakan sisi berbobot yang nilainya bisa mempresentasikan panjang lintasan. Karena algoritma A* memerlukan nilai heuristik, setelah peta sudah direpresentasikan dalam bentuk graf, perlu diketahui informasi tambahan setiap simpul dan informasi tersebut harus dapat diterima agar dapat dijadikan nilai heuristik. Setelah informasi jarak dan nilai heuristik diketahui, pencarian dapat dilakukan menggunakan pohon pencarian dan antrian prioritas. Setelah jarak didapatkan, harga dapat dihitung dan harus dilakukan pembulatan ke atas untuk memudahkan pembayaran dengan tidak merugikan perusahaan.

REFERENSI

[1] Munir, Rinaldi, 2010, Matematika Diskrit edisi III. Bandung: Informatika Bandung.

[2] Munir, Rinaldi, 2004, Diktat Strategi Algoritmik. Bandung: Departemen Teknik Informatika Institut Teknologi Bandung

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 Mei 2018

Erfandi Suryo Putra 13515145