Penerapan Algoritma A* dalam Menentukan Rute Mudik Terpendek Padang-Bukittinggi di Sumatera Barat

Febryola Kurnia Putri - 13520140
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): febryola0402@gmail.com

***Abstrak*—Mudik merupakan suatu kegiatan yang dilakukan oleh orang yang sedang bekerja atau berada di luar kota dan kembali ke kampung halaman. Di Indonesia, mudik merupakan sesuatu yang identik dengan momen lebaran. Saat menjelang lebaran, banyak perantau yang memadati jalan utama dengan tujuan yang sama, yaitu untuk bertemu dengan keluarga tercinta. Rute yang pendek akan memudahkan pemudik untuk melakukan perjalanan dan kembali ke tempat semula. Pada makalah ini akan dibahas bagaimana menentukan rute mudik terpendek antara Kota Padang dan Kota Bukittinggi di Provinsi Sumatera Barat dengan memanfaatkan algoritma A*.**

Keywords—mudik; padang; bukittinggi; A*; simpul; rute

I. PENDAHULUAN

Mudik merupakan kegiatan perantau atau pekerja migran untuk pulang ke kampung halamannya. Mudik di Indonesia identik dengan tradisi tahunan yang terjadi menjelang hari raya besar keagamaan misalnya menjelang Idul Fitri, Natal & Tahun Baru, Idul Adha dan Hari besar Nasional. Pada saat itulah ada kesempatan untuk berkumpul dengan sanak saudara yang tersebar di perantauan, selain tentunya juga berkumpul dengan orang tua. Transportasi yang digunakan antara lain pesawat terbang, kereta api, kapal laut, bus, dan kendaraan pribadi seperti mobil dan sepeda motor, bahkan truk dan bajaj dapat digunakan untuk mudik. Kata mudik berasal dari kata "udik" yang artinya selatan/hulu. Pada zaman dahulu sebelum di Jakarta terjadi urbanisasi besar-besaran, masih banyak wilayah yang bernama akhir udik atau ilir (utara atau hilir) dan kebanyakan akhiran itu diganti dengan kata Melayu selatan atau utara. Contohnya seperti Meruya Udik, Meruya Ilir, Sukabumi Udik, Sukabumi Ilir, dan sebagainya.

Terdapat juga teori yang mengatakan bahwa asal-usul kata "Mudik" berasal dari akronim dua kata dalam bahasa Jawa yaitu "Mulih dhisik" yang bermakna "Pulang dahulu". Walau belum dapat dipastikan kebenarannya, namun teori ini cukup beredar luas, terlebih di kalangan masyarakat pulau Jawa. Di Sumatera Barat sendiri, juga sering terjadi mudik internalnya yaitu mudik antarkotanya. Biasanya, rute awal mudik berasal dari Kota Padang karena sebagai ibu kota, pusat semua transportasi luar provinsi berada di Kota Padang, mulai dari bandara, terminal, stasiun, dan lain sebagainya. Kota Padang memiliki kepadatan 1.101 jiwa per km2. Hal ini akan meningkat seiring dengan bertambahnya pemudik yang datang dari luar provinsi. Kota Bukittinggi dapat dikatakan sebagai ibu kota kedua di Sumatera Barat karena kemajuan dan perkembangan kotanya yang saling beriringan dengan Kota Padang. Selain itu, Kota Bukittinggi juga disebut sebagai kota wisata di Sumatera Barat karena terdapat banyak objek wisata yang dapat ditemui di kota ini. Oleh karena itu, pemudik yang datang dari luar provinsi dan berada di Kota Padang seringkali akan melakukan perjalanan menuju Kota Bukittinggi. Namun, karena kepadatan arus mudik yang terjadi, maka kemacetan seringkali tidak terhindarkan. Oleh karena itu, penting untuk menemukan rute yang tepat dan efisien bagi pemudik agar dapat menempuh perjalanan menuju kota tujuan dan kembali ke kota awal.

Gambar 1. Arus padat mudik lebaran

Makalah ini akan membahas mengenai cara untuk menentukan rute terpendek atau rute dengan bobot terkecil antara Kota Padang dan Kota Bukittinggi dengan menggunakan algoritma A* untuk menentukan rute perjalanan yang optimal. Algoritma A* salah satu algoritma yang digunakan dalam pencarian rute dan termasuk dalam pencarian rute informed search, yaitu pencarian rute yang mengetahui lokasi yang ingin dituju dan dapat digunakan untuk memperhitungkan rute yang ingin dituju dan diharapkan memperoleh hasil yang optimal.

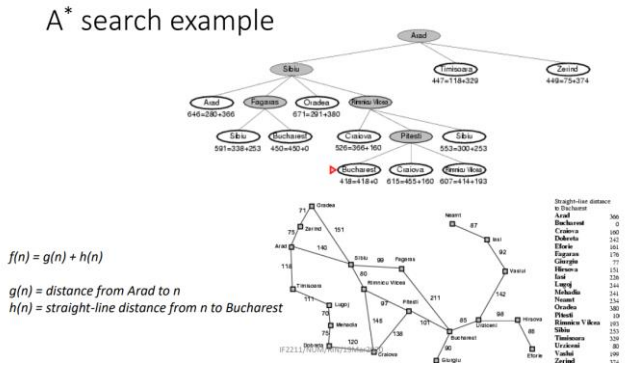
II. LANDASAN TEORI

A. Definisi Algoritma A*

Algoritma A* merupakah salah satu algoritma yang digunakan dalam pencarian rute dan termasuk ke dalam pencarian rute *informed search*, yaitu pencarian rute yang mengetahui lokasi yang ingin dituju dan dapat digunakan untuk memperhitungkan rute yang ingin dituju. Dalam hal ini, lokasi yang ingin dituju sudah teridentifikasi, yaitu Kota Bukittinggi berikut dengan *cost* yang dibutuhkan. Algoritma ini berbeda dengan algoritma lain yang termasuk *informed search*, yaitu *Greedy Best First Search*, dimana algoritma tersebut tidak memperhitungkan cost untuk mencapai tujuan. Oleh karena itu, penulis memilih algoritma A* sebagai algoritma pencarian rute paling mangkus dan optimal karena sesuai dengen tujuan yang ingin dicapai, yaitu rute dengan cost paling sedikit dan risiko yang paling rendah. Algoritma A* memiliki konsep algoritma berupa penghindaran mengekspansi jalur yang mahal. Algoritma ini menggunakan fungsi evaluasi antara lain,

$$f(n) = g(n) + h(n)$$

dimana g(n) adalah *cost* yang dibutuhkan untuk dari tempat asal sampai ke n, h(n) adalah estimasi *cost* yang dibutuhkan untuk sampai dari n ke tujuan, dan f(n) adalah estimasi total *cost* yang dibutuhkan dari tempat asal ke tujuan melalui n. Saat mencari rute dengan menggunakan algoritma A*, algoritma akan selalu mencari nilai dengan besar f(n) yang paling kecil dari yang lain sehingga algoritma A* mampu mendapatkan hasil rute dengan cost yang paling kecil yang mana dibutuhkan dalam penyelesaian permasalahan ini.



Gambar 2. Contoh penerapan algoritma A* pada pencarian rute

B. Penentuan Rute

Algoritma yang digunakan pada penentuan rute merupakan algoritma yang digunakan untuk mencari rute dengan *cost* paling sedikit dan risiko yang paling rendah. Algoritma tersebut dapat dibagi menjadi dua bagian, yaitu *uninformed search* dan *informed search*. *Uninformed search* adalah sebuah metode pencarian rute yang tidak memiliki informasi pendukung mengenai tujuan yang sedang dicari, dimana pencarian dilakukan hanya dengan melihat informasi yang dimiliki sekarang. Informasi tersebut dapat berupa informasi yang beragam, oleh karena itu semua informasi akan diunifikasi menjadi satu besaran yang disebut sebagai *cost.* Algoritma yang termasuk ke dalam *uninformed search* adalah *Breadth First Search, Depth First Search, Uniform Cost Search*. Sedangkan untuk *informed search* merupakan sebuah metode pencarian rute yang memiliki informasi pendukung mengenai tujuan yang ingin dituju. Informasi yang telah diberikan tersebut biasa disebut sebagai nilai heuristik. Pada umumnya, nilai tersebut memberikan keterangan mengenai penilaian terhadap sebuah lokasi, dimana penilaian tersebut harus relevan dengan hal yang ingin dioptimisasikan dalam proses pencarian. Nilai heuristik juga digunakan untuk menentukan titik yang akan dieksplorasi selanjutnya, dimana hasil yang didapatkan akan menjadi lebih baik untuk Sebagian besar permasalahan. Algoritma yang termasuk informed search adalah *Greedy Best First Search* dan A*. Menggunakan persyaratan optimasi dan kendala untuk membatasi sejumlah pilihan yang harus dipertimbangkan pada suatu tahap.

C. Perbedaan dengan Algoritma Greedy Best First Search

Perbedaaan algoritma *greedy* best first search dengan algoritma A* adalah algoritma A* memperhitungkan cost yang dibutuhkan untuk mencapai lokasi tujuan, sedangkan algoritma *greedy best first search* tidak memperhitungkan hal tersebut.

Untuk dapat dikatakan sebagai persoalan program A* pencarian rute, ada beberapa karakteristik yang harus dipenuhi oleh persoalan tersebut. Karakteristik tersebut adalah:

1. Lokasi dari rute yang ingin dituju pada persoalan harus diketahui
2. *Cost* atau harga yang dibutuhkan dari simpul awal menuju simpul tujuan juga harus diketahui

D. Perbedaan dengan Algoritma Branch and Bound

Pada algoritma *branch and bound* dilakukan rute pencarian sebagai berikut:

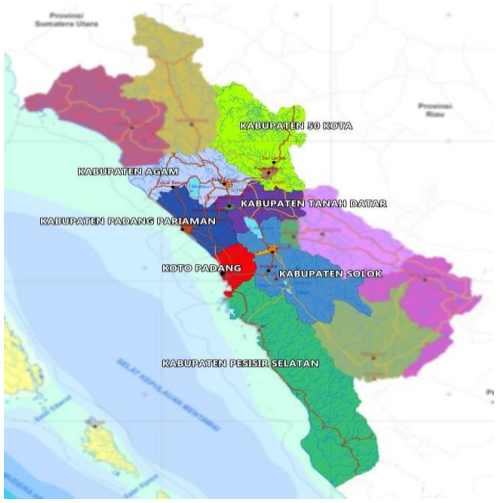
1. Mencari bound yang dijamin lebih rendah daripada *cost* yang sebenarnya
2. Mencari percabangan pohon sesuai dengan cara yang disukai, misalnya dengan menggunakan algoritma *dept first search* atau *best first search*
3. Hentikan pencarian jika *cost* + *bound* > solusi terbaik yang telah ditemukan
4. Jika heuristiknya adalah *cost* + *bound*, maka algoritma pencarian yang dilakukan adalah *best first search.*

III. IMPLEMENTASI DAN PENGUJIAN

Algoritma A* dapat digunakan untuk menentukan rute terpendek antara Kota Padang dan Kota Bukittinggi, berikut mapping persoalan dan kode program pengaplikasian Algoritma A* untuk pencarian rute terpendek:

A. Mapping Persoalan

Hal pertama yang dilakukan adalah melakukan *mapping* peta wilayah Sumatera Barat dan mendapatkan graf untuk menentukan rute yang mungkin ditempuh dari titik awal menuju titik akhir. Berikut adalah peta wilayah Sumatera Barat:



Gambar 3. Peta wilayah Sumatera Barat

Selanjutnya akan dibuat graf representasi peta yang menghubungkan kota/kabupaten dari Kota Padang menuju Kota Bukittinggi. Untuk merepresentasikan peta, akan dibuat graf dual dari peta tersebut. Graf dual dari peta Sumatera Barat dapat dibuat dengan cara merepresentasikan setiap wilayah kota/kabupaten di Sumatera Barat sebagai simpul dan jika dua kecamatan saling bertetangga, maka di graf dualnya, kedua simpul yang bertetangga mempunyai sisi yang menghubungkan kedua simpul tersebut. Dikarenakan pada makalah ini akan dibuat suatu rute, diperlukan pusat dari tiap kota/kabupaten yang bertindak sebagai terminal. Simpul-simpul dari graf dual akan ditempatkan pada pusat dari setiap kota/kabupaten yang direpresentasikan oleh pusat dari setiap kota/kabupaten tersebut, sedangkan sisi dibentuk dari jarak antar kota/kabupaten yang berbatasan langsung dengan kota/kabupaten tersebut. Berikut *mapping* graf terhadap peta wilayah Sumatera Barat:

Gambar 4. Mapping representasi graf peta Sumatera Barat

Untuk memudahkan dalam pengolahan data, nama dari tiap kota diwakilkan oleh simbol simpul.

Tabel 1. Informasi nama kota/kabupaten pada peta beserta simpulnya

No	Kota/Kabupaten	Simpul
1	Kota Padang	v1
2	Kota Pariaman	v2
3	Kabupaten Pesisir Selatan	v3
4	Kota Solok	v4
5	Kabupaten Tanah Datar	v5
6	Kabupaten Agam	v6
7	Kota Bukittinggi	v7

Telah dibentuk graf dual G = (V, E), dimana

V = {v1, v2, v3, v4, v5, v6, v7}

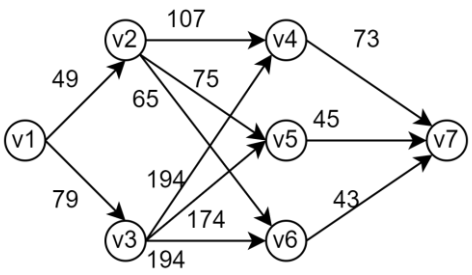
E = {v1v2, v1v3, v2v4, v2v5, v2v6, v3v4, v3v5, v3v6, v4v7, v5v7, v6v7}

Bobot yang menjadi ongkos (*cost*) dalam hal ini adalah jarak tiap kotanya dalam km yang didapatkan dengan menggunakan bantuan *Google Maps*. Berikut cost dari setiap simpul pada peta di atas:

Tabel 2. Informasi cost yang ditempuh tiap kota/kabupaten pada peta beserta simpulnya

No	Sisi (simpul,simpul)	Cost (km)
1	(v1,v2)	49
2	(v1,v3)	79
3	(v2,v4)	107
4	(v2,v5)	75
5	(v2,v6)	65
6	(v3,v4)	194
7	(v3,v5)	174
8	(v3,v6)	194
9	(v4,v7)	73
10	(v5,v7)	45
11	(v6,v7)	43

Mapping persoalan di atas dalam bentuk lintasannya untuk mempermudah pencarian solusi adalah sebagai berikut:



Gambar 5. Mapping graf

B. Implementasi Pada Program

Implementasi program yang dilakukan kali ini adalah menggunakan Bahasa pemograman python. Berikut kode program yang digunakan untuk menyelesaikan permasalahan dan menemukan solusi:

1. Class Graph

```
# Membuat class graph
class Graph:
    # Membuat fungsi __init__
    def __init__(self, adjajency_list):
        self.adjajency_list = adjajency_list

    # Membuat fungsi get_tetangga untuk mendapatkan tetangga simpul yang berdekatan
    def get_tetangga(self, v):
        return self.adjajency_list[v]

    # Membuat fungsi heuristik yang akan membuat semua simpul menjadi equal
    def heuristik(self, n):
        H = {
            'V1': 1,
            'V2': 1,
            'V3': 1,
            'V4': 1,
            'V5': 1,
            'V6': 1,
            'V7': 1
        }
        return H[n]
```

Gambar 6. Potongan kode class graph

Pada potongan kode di atas, dibuat kelas graph sebagai inisialisasi dan dibuat fungsi-fungsi yang dibutuhkan, di antaranya fungsi get_tetangga yang akan mengembalikan list semua tetangga dari simpul. Lalu, terdapat juga fungsi heuristik yang diterapkan di setiap simpul, di mana fungsi ini akan menghasilkan *value* atau nilai 1 untuk setiap simpulnya. Terdapat 7 simpul yang diinisialisasikan pada persoalan ini sesuai dengan mapping persoalan yang telah dilakukan pada bagian sebelumnya.

2. Algoritma A* mapping

Berikut *mapping* persoalan di atas dengan menampilkan simpul ekspan dan simpul hidupnya dengan menggunakan algoritma A*

Tabel 3. *Mapping* persoalan rute terpendek dengan algoritma A*

No	Simpul Ekspan	Simpul Hidup
1	v1	v1v2 = f(v1v2) = 49 v1v3 = f(v1v3) = 79
2	v1v2	v1v2v6== f(v1v2v6) = 49 + 65 = 114 v3 = f(v1v3) = 79
3	v1v3	v1v3v5 = f(v1v3v5) = 79 + 174 = 253 v1v2v6 = f(v1v2v6) = 49 + 65 = 114
4	v1v2v6	v1v2v6v7 = f(v1v2v6v7) = 114 + 43 = 157 v1v3v5 = f(v1v3v5) = 79 + 174 = 253
5	v1v2v6v7	Sudah sampai simpul solusi dengan cost = 157

Pada *mapping* persoalan dengan algoritma A* diperoleh melalui evaluasi antara simpul ekspan dan simpul hidupnya. Pada awal *mapping*, Di mana hanya v1 yang dibangkitkan dan simpul hidup yang ada adalah v1v2 dan v1v3 yang *cost* masing-masingnya jika dibandingkan membuat f(v1v2) < f(v1v3) sehingga simpul selanjutnya yang dibangkitkan adalah simpul v1v2. Selanjutnya dilakukan pencarian kembali di mana pada simpul hidup akan terdapat simpul v1v2v6 dengan f(v1v2v6) lebih besar dari f(v1v3), maka yang akan dibangkitkan selanjutnya adalah simpul v1v3. Selanjutnya, dengan membangkitkan simpul v1v3, maka pada simpul hidup akan terdapat simpul v1v3v5 yang didapatkan dari fungsi evaluasi ke simpul lainnya, simpul yang diekspan berikutnya adalah simpul v1v2v6 karena f(v1v2v6) < f(v1v3v5). Terakhir, dibangkitkan simpul v1v2v6v7 yang pada akhirnya merupakan simpul solusi. Dari pencarian di atas diperoleh *list* dari solusi yang optimal untuk rute terpendek dari simpul awal dan simpul akhir adalah (v1, v2, v6, v7) yang dalam hal ini berarti rute terpendek yang dapat ditempuh dari Kota Padang menuju Kota Bukittinggi adalah dari Kota Padang → Kota Pariaman → Kabupaten Agam → Kota Bukittinggi dengan *cost* atau total jarak yang ditempuh adalah 157 km.

3. Implementasi Algoritma A* pada Program

Berikut merupakan penerapan kode program pencarian rute terpendek Padang-Bukittinggi dengan menerapkan algoritma A*:

```
# Membuat fungsi algoritma A star sebagai algoritma utama
def algoritma_A_Star(self, start, stop):
    # Menginisialisasi semua variabel yang dibutuhkan

    # variabel nodes adalah list yang berisi semua simpul
    nodes = {}
    nodes[start] = start

    # variabel awal adalah simpul awal
    # variabel akhir adalah list simpul yang telah dikunjungi
    awal = set([start])
    akhir = set([])

    # variabel value merepresentasikan cost dari simpul awal ke simpul lain
    value = {}
    value[start] = 0

    # melakukan looping untuk mencari simpul yang telah dikunjungi
    while len(awal) > 0:
        val = None

        # untuk menemukan value terkecil dari fungsi f() -
        for i in awal:
            if val == None or value[i] + self.heuristik(i) < value[val] + self.heuristik(val):
                val = i;
```

Gambar 7. Kode Program Algoritma A*

Pada kode di atas dilakukan inisialisasi setiap variabel yang dibutuhkan, berupa list dari simpul dan nilai atau *cost* yang diperlukan satu simpul untuk menuju simpul yang lain. Berikut adalah kode program secara keseluruhan:

```
# melakukan looping untuk mencari simpul yang telah dikunjungi
while len(awal) > 0:
    val = None

    # untuk menemukan value terkecil dari fungsi f() -
    for i in awal:
        if val == None or value[i] + self.heuristik(i) < value[val] + self.heuristik(val):
            val = i;

    # jika simpul yang dicari tidak ada di list akhir
    if val == None:
        print('Path does not exist!')
        return None

    # jika current simpulnya adalah stop maka ulang kembali dari simpul awal
    if val == stop:
        reconst_path = []


        while nodes[val] != val:
            reconst_path.append(val)
            val = nodes[val]

        reconst_path.append(start)

        reconst_path.reverse()
```

Gambar 8. Kode Program Algoritma A*

Pada bagian ini dilakukan pengecekan, jika current simpul sekarang atau simpul hidupnya 0, maka hanya mengeluarkan *message*. Jika current simpul sama dengan simpul tujuan, maka pencarian akan dilakukan kembali dari awal. Di mana pada bagian ini current simpul akan ditambahkan ke current path yang sudah dilakukan pencarian pada bagian sebelumnya.

```
        print('\033[32mRute yang ditempuh\033[0m: {}'.format(reconst_path))
        for i in reconst_path:
            if i == 'V1':
                i = "\033[34mPadang\033[0m"
            elif i == 'V2':
                i = "\033[33mPariaman\033[0m"
            elif i == 'V3':
                i = "\033[35mPesisir Selatan\033[0m"
            elif i == 'V4':
                i = "\033[32mSolok\033[0m"
            elif i == 'V5':
                i = "\033[36mTanah Datar\033[0m"
            elif i == 'V6':
                i = "\033[31mAgam\033[0m"
            elif i == 'V7':
                i = "\033[34mBukittinggi\033[0m"
            print(i)
        print("\n\033[31mCost yang dibutuhkan\033[0m: {}".format(value[stop]))

        return reconst_path

    # untuk semua current node yang tersisa maka lakukan proses berikut
    for (m, weight) in self.get_tetangga(val):
      # jika simpul yang dicari tidak ada di list akhir maupun list awal
        # tambahkan simpul yang dicari ke list awal
        if m not in awal and m not in akhir:
            awal.add(m)
            nodes[m] = val
```

Gambar 9. Kode Program Algoritma A*

Kode di atas merepresentasikan hasil dari path yang dihasilkan program di mana setiap simpul akan ditranslasikan menjadi nama kota yang bersesuaian dengan yang terdapat pada tabel 1. V1 merepresentasikan Kota Padang, V2 merepresentasikan Kota Pariaman, V3 merepresentasikan Kabupaten Pesisir Selatan, V4 merepresentasikan Kota Solok, V5 merepresentasikan Kabupaten Tanah Datar, V6 merepresentasikan Kabupaten Agam, dan V7 merepresentasikan Kota Bukittinggi.

```
    # untuk semua current node yang tersisa maka lakukan proses berikut
    for (m, weight) in self.get_tetangga(val):
      # jika simpul yang dicari tidak ada di list akhir maupun list awal
        # tambahkan simpul yang dicari ke list awal
        if m not in awal and m not in akhir:
            awal.add(m)
            nodes[m] = val
            value[m] = value[val] + weight

        else:
            if value[m] > value[val] + weight:
                value[m] = value[val] + weight
                nodes[m] = val

                if m in akhir:
                    akhir.remove(m)
                    awal.add(m)

    # hapus val dari list awal dan tambahkan ke list akhir
    # karena semua simpul telah dikunjungi
    awal.remove(val)
    akhir.add(val)

print('Path does not exist!')
return None
```

Gambar 10. Kode Program Algoritma A*

Untuk menemukan simpul berikutnya yang akan dikunjungi, maka dilakukan pengecekan *value* atau *cost* yang dibutuhkan untuk menempuh simpul tersebut di mana f(n) sangat diperlukan dalam hal ini. Jika current simpul sekarang tidak terdapat pada list awal ataupun pada *list* akhir, maka dilakukan penambahan *current* simpul tersebut ke list simpul awal. Akan tetapi, jika *current* simpul tersebut sudah tergolong ke salah satunya, maka akan ditentukan simpul mana yang akan dimasukkan ke *list* akhir sebagai solusi nantinya. Di mana jika, value dari *current* simpul lebih besar dari *value* simpul sebelumnya ditambah dengan cost menuju simpul tersebut maka simpul tersebut akan dihapus dari list simpul akhir atau simpul hidup.

C. Pengujian

Berikut dilakukan pengujian terhadap program untuk membuktikan kebenaran atas mapping persoalan dengan menggunakan simpul ekspan dan simpul hidup yang telah didefinisikan pada bagian B:

```
# Driver program
adjajency_list = {
    'V1': [('V2', 49), ('V3', 73)],
    'V2': [('V4', 107), ('V5', 75), ('V6', 65)],
    'V3': [('V4', 194), ('V5', 174), ('V6', 194)],
    'V4': [('V7', 73)],
    'V5': [('V7', 45)],
    'V6': [('V7', 43)]
}
graph1 = Graph(adjajency_list)
graph1.algoritma_A_Star('V1', 'V7')
```

Gambar 11. Input Program

Inputan program yang dibutuhkan adalah graf yang berisikan daftar list awal atau list dalam graf selain list tujuan yang dalam hal ini merupakan V7. Inputan yang dilakukan

adalah input list awal simpul pada bagian kiri dan setiap simpul akan berisikan list tetangganya yang sesuai dengan gambar 4 (*mapping* graf) pada bagian sebelumnya berikut dengan *cost* yang dibutuhkan menuju simpul tersebut. Dalam hal ini dapat didaftarkan sebagai berikut:

'V1': [('V2', 49), ('V3', 73)],

'V2': [('V4', 107), ('V5', 75), ('V6', 65)],

'V3': [('V4', 194), ('V5', 174), ('V6', 194)],

'V4': [('V7', 73)],

'V5': [('V7', 45)],

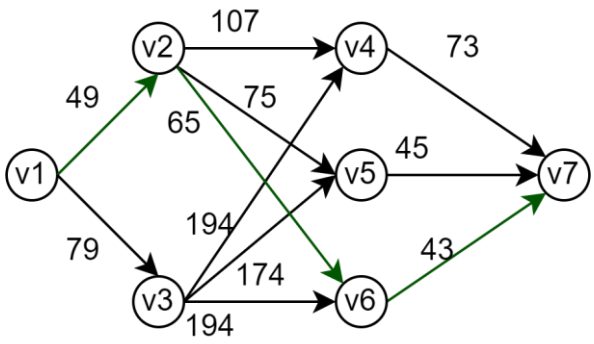
'V6': [('V7', 43)]

```
Rute yang ditempuh: ['V1', 'V2', 'V6', 'V7']
Padang
Pariaman
Agam
Bukittinggi

Cost yang dibutuhkan: 157
```

Gambar 12. Output Program

Dari output program di atas, diperoleh hasil bahwa rute yang harus ditempuh agar perjalanan dari simpul awal ke simpul akhir merupakan rute yang paling optimal dengan *cost* terkecil adalah V1 → V2 → V6 → V7 yang mana merupakan rute Kota Padang → Kota Pariaman → Kabupaten Agam → Kota Bukittinggi dengan *cost* atau jarak yang ditempuh adalah 157 km. Solusi ini merupakan solusi paling optimal dalam pencarian rute terpendek antara Kota Padang dan Kota Bukittinggi. Hasil pengujian program sesuai dengan hasil *mapping* persoalan pada bagian sebelumnya, yang berarti program sudah benar. Berikut diberikan hasil akhir *mapping* persoalan beserta dengan rute solusinya yang ditandai dengan panah berwarna hijau:



Gambar 13. Hasil akhir mapping graf

IV. KESIMPULAN

Dengan mengaplikasikan algoritma A*, dapat dibuat sebuah program yang dapat digunakan untuk menentukan rute terpendek antara titik awal dan titik akhir tujuan dengan optimal. Dengan adanya rute ini, maka pemudik dapat menempuh perjalanan dengan lancar dan efisien dari titik awal keberangkatan sampai pada titik akhir tujuan yang dalam hal ini, titik awalnya adalah Kota Padang dan titik tujuannya adalah Kota Bukittinggi. Algoritma A* yang digunakan adalah algoritma A* pencarian rute yang pangkalnya akan berawal dari Kota Padang dan berakhir di Kota Bukittinggi. Hasil dari penerapan algoritma A* ini didapatkan bahwa rute terbaik yang dapat ditempuh pemudik saat melakukan perjalanan dari Kota Padang menuju Kota Bukittinggi adalah Kota Padang, Kota Pariaman, Kabupaten Agam, dan Kota Bukittinggi di mana pada hasilnya akan menghasilkan *cost* atau jarak terkecil dibandingkan dengan hasil kemungkinan lain.Dengan adanya rancangan rute ini, diharapkan perjalanan mudik yang dilakukan oleh pemudik menjadi lebih baik, di mana terhindar dari kemacetan dan menempuh perjalanan yang cepat dan efisien serta kenyamanan dalam perjalanan.

LINK VIDEO DI YOUTUBE

Berikut link video penulis lampirkan dari tugas makalah ini: https://youtu.be/dQKYfYzUg9w

UCAPAN TERIMA KASIH

Pertama-tama penulis mengucapkan terima kasih dan rasa syukur yang sebesar-besarnya kepada Tuhan Yang Maha Esa karena atas berkat rahmat dan hidayahnya penulis dapat menyelesaikan tugas makalah ini dengan baik. Penulis juga mengucapkan terima kasih kepada Bapak Rinaldi Munir, Ibu Nur Ulfa Maulidevi, dan Ibu Masayu Leylia Khodra selaku dosen pengajar mata kuliah IF2211 Strategi Algoritma yang telah memberikan banyak ilmu yang bermanfaat sebagai bekal dalam penulisan makalah ini. Penulis juga mengucapkan terima kasih sebesar-besarnya kepada keluarga yang telah memberi dukungan penuh dalam perkuliahan dan kehidupan serta teman-teman program studi Teknik Informatika yang bersama-sama menjalani perkuliahan pada Semester II Tahun 2021/2022 di program studi Teknik Informatika ini.

REFERENSI

[1] Munir, Rinaldi. (2009). Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung.

[2] R.Munir, Slide Graf bag.1(2020), Bandung.

[3] R.Munir, Slide Graf bag.2(2020), Bandung

[4] https://news.detik.com/berita/d-6040252/pengertian-mudik-beserta-aturannya-jelang-lebaran-2022 diakses pada 07 Mei 2022

[5] python - How to apply dynamic programming to compute shortest path in graph? - Stack Overflow diakses pada 07 Mei 2022

[6] Shortest Path Python Programming - Floyd Warshall Algorithm Dynamic Programming - Learn in 30 sec from Microsoft awarded MVP (wikitechy.com) diakses pada 07 Mei 2022

[7] Algoritma A* (A-Star) - Pip Tools diakses pada 08 Mei 2022

[8] A* Search Algorithm - GeeksforGeeks diakses pada 08 Mei 2022

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, 08 Mei 2022

Febryola Kurnia Putri - 13520140