Implementasi Algoritma A* untuk Menemukan Rute Penerbangan Termurah untuk Maskapai AirAsia

Muhammad Farrel Danendra Rachim - 13521048
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13521048@std.stei.itb.ac.id

***Abstract*—AirAsia merupakan salah satu maskapai pesawat yang paling sering digunakan penduduk Indonesia untuk bepergian, khususnya ke luar negeri. Namun, harga tiket penerbangan yang berbeda untuk berbagai macam rute seringkali menjadi bahan pertimbangan. Bahkan, tidak sedikit penumpang yang memilih untuk transit (singgah) di kota lain sebelum mencapai kota tujuan. Mengingat harga tiket pesawat yang tidak murah, calon penumpang cenderung mempersiapkan diri dengan memilih rute penerbangan yang tepat agar biaya yang dikeluarkan minimal. Permasalahan dalam menentukan rute penerbangan yang paling terjangkau dengan maskapai Air Asia dapat diselesaikan dengan konsep graf berbobot berarah dan Algoritma A*.**

***Keywords*—Algoritma A*, AirAsia, Graf Berbobot, Harga**

I. Pendahuluan

Pesawat adalah salah satu sarana transportasi yang paling sering digunakan masyarakat untuk bepergian ke tempat yang jauh, baik dalam negeri maupun luar negeri. Waktu perjalanan yang relatif cepat dan layanan maskapai pesawat yang berkualitas termasuk alasan mengapa orang-orang memilih menggunakan pesawat untuk mencapai kota tujuan mereka.

AirAsia adalah sebuah maskapai penerbangan yang sangat digemari bukan hanya untuk penduduk Indonesia, namun juga penduduk Asia, karena biaya tiketnya yang relatif lebih terjangkau dibandingkan maskapai lain. Maskapai ini didirikan di Kuala Lumpur, Malaysia, pada tahun 1993 dan mulai beroperasi pada tahun 1996. AirAsia termasuk ke dalam lima maskapai terbesar di Asia, dan saat ini terdapat 100 buah pesawat yang beroperasi (edisi Malaysia) dengan lebih dari 160 tujuan dalam 25 negara [1].

Gambar 1. Pesawat AirAsia [2]

Meskipun AirAsia termasuk maskapai penerbangan dengan biaya yang rendah, para calon penumpang tentu lebih memilih rute penerbangan yang lebih murah dari yang lain untuk mencapai kota tujuan mereka. Terdapat banyak pilihan penerbangan yang dapat dipilih untuk mencapai tujuan, namun memilih rute dengan biaya paling terjangkau menjadi tantangan yang sering dihadapi oleh calon penumpang. Bahkan, tidak jarang para penumpang memilih untuk transit di kota lain sebelum mencapai tujuan mereka demi pengeluaran yang minimal.

Perencanaan rute paling murah menjadi hal yang penting untuk dipertimbangkan oleh calon penumpang. Penulis akan memodelkan informasi mengenai penerbangan dengan harganya sebagai sebuah graf berbobot berarah, dengan harga penerbangan antarkota sebagai bobot graf tersebut. Lalu, dari graf tersebut, penulis akan memanfaatkan algoritma A* untuk menentukan rute tersebut.

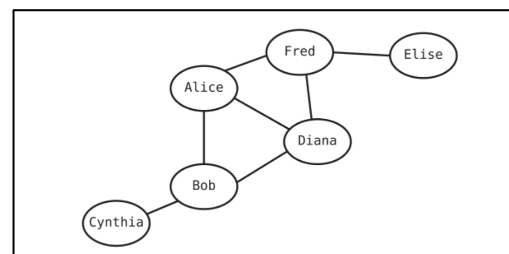
II. Teori Dasar

A. Definisi Graf

Graf (dalam bahasa Inggris: *graph*) dalam dunia matematika adalah sebuah struktur yang merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Objek-objek yang dimaksud digambarkan sebagai sebuah kumpulan simpul (*vertex*), dan hubungan yang dimaksud digambarkan sebagai kumpulan sisi (*edge*). Graf dapat didefinisikan sebagai

$G = (V, E)$

dengan G adalah sebuah graf, V adalah himpunan berisi simpul-simpul yang tidak kosong ($\{v_1, v_2, v_3, \ldots v_n\}$), dan E adalah himpunan berisi sisi-sisi yang menghubungan sepasang simpul ($\{e_1, e_2, \ldots, e_n\}$).

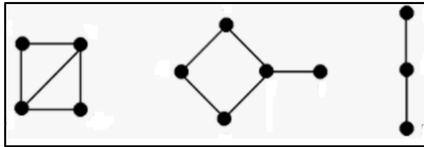


Gambar 2. Sebuah graf yang menggambarkan hubungan pertemanan [3]

B. Jenis Graf

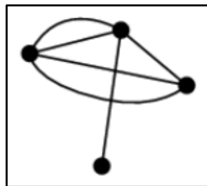
Berdasarkan keberadaan gelang atau sisi ganda pada sebuah graf, graf tergolong menjadi dua macam:

1. Graf sederhana (*simple graph*)
   Graf sederhana adalah graf yang tidak mengandung gelang atau sisi ganda.



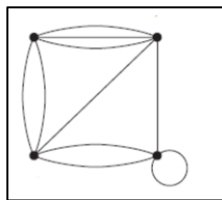
Gambar 3. Graf sederhana [4]

2. Graf tak-sederhana (*unsimple graph*)
   Graf tak sederhana adalah graf yang memiliki sisi ganda atau gelang. Graf tak sederhana terdiri dari:
   a. Graf ganda (*multi-graph*)
      Graf tak sederhana yang mengandung sisi ganda, namun tidak memiliki sisi gelang.



Gambar 4. Graf ganda (*multi-graph*) [5]

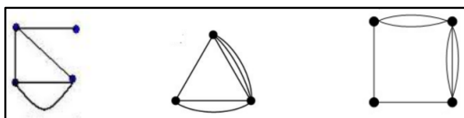
   b. Graf semu (*pseudo-graph*)
      Graf tak sederhana yang mengandung sisi gelang. Graf semu bisa mengandung sisi gelang dan sisi ganda.



Gambar 5. Graf semu (*pseudo-graph*) [4]

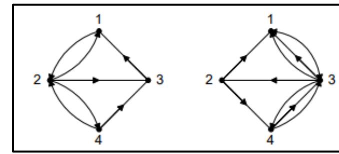
Berdasarkan arah sisi, graf tergolong menjadi dua macam:

1. Graf tak berarah (*undirected graph*)
   Graf tak beararah adalah graf yang tidak memiliki orientasi arah pada masing-masing sisinya.



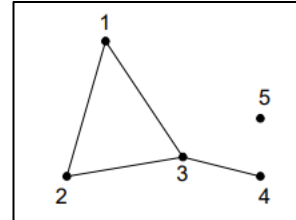
Gambar 6. Graf tak beararah (*undirected graph*) [4]

2. Graf berarah (*directed graph/digraph*)
   Graf berarah adalah graf yang memiliki orientasi arah pada masing-masing sisinya. Graf berarah dapat dibedakan lagi menjadi graf berarah, yang tidak memiliki sisi ganda, dan graf-ganda berarah, yang memiliki sisi ganda.

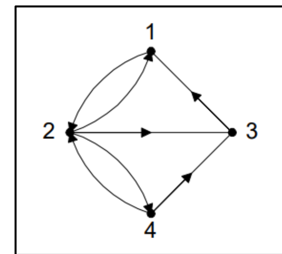


Gambar 7. Graf berarah (*directed graph/digraph*) [4]

C. Terminologi Graf

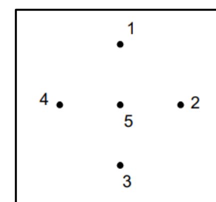


Gambar 8. Sebuah Graf tak berarah bernama G1 [4]



Gambar 9: Sebuah Graf berarah bernama G2 [4]

1. Ketetanggaan (*adjacency*)
   Dua buah simpul disebut bertetangga jika keduanya terhubung melalui sisi secara langsung. Sebagai contoh, pada gambar 8, simpul 2 bertetangga dengan simpul 1 dan 3, namun tidak bertetangga dengan simpul 4.
2. Bersisian (*incidency*)
   Sisi $e = (v_1, v_2)$ yang menghubungkan v_1 dan v_2 bersisian dengan simpul v_1 dan v_2. Pada gambar 8, sisi (2, 3) bersisian dengan simpul 2 dan 3.
3. Simpul terpencil (*isolated vertex*)
   Simpul terpencil adalah simpul yang tidak memiliki sisi yang bersisian dengannya. Pada gambar 8, simpul 5 adalah simpul terpencil.
4. Graf kosong
   Graf kosong adalah graf yang himpunan sisinya merupakan himpunan kosong, dan disimobolkan dengan N_n, dengan n jumlah simpul dalam graf kosong.



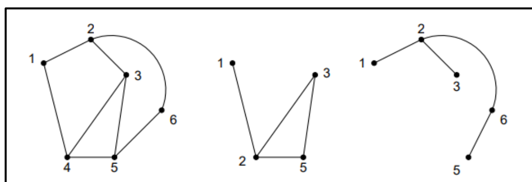
Gambar 10. Graf kosong (N_5) [4]

5. Derajat
   Derajat sebuah simpul mewakili jumlah sisi yang bersisian dengan simpul tersebut, dan dapat disimbolkan dengan notasi d(v). Misal, pada gambar 8, d(1) = d(2) = 2, d(3) = 3, d(4) = 1 (karena derajatnya hanya satu,

simpul 4 juga dapat disebut dengan simpul anting-anting/*pendant vertex*), dan d(5) = 0.

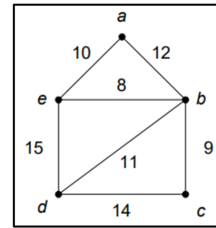
Pada graf berarah, derajat sebuah simpul digolongkan menjadi derajat masuk (*in-degree*, $d_{in}(v)$) dan derajat keluar (*out-degree*, $d_{out}(v)$). Pada gambar 9, $d_{in}(1) = 2$, dan $d_{out}(1) = 1$, karena ada dua sisi yang menuju simpul 1 dan satu sisi yang berasal dari simpul 1.

6. Lintasan (*path*)
   Lintasan dengan panjang n dari simpul awal v_0 sampai simpul tujuan v_n adalah barisan yang berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n$ sedemikian sehingga $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2)$, …, $e_n = (v_{n-1}, v_n)$ adalah sisi-sisi dari graf G. Adapun panjang lintasan adalah jumlah sisi dalam lintasan tersebut. Misal pada gambar 8, lintasan 4, 3, 2, 1 adalah lintasan dengan barisan sisi (4, 3), (3, 2), (2, 1), dan panjang lintasannya 3.
7. Siklus atau Sirkuit
   Sirkuit adalah lintasan yang berawal dan berakhir pada simpul yang sama. Panjang sirkuit adalah jumlah sisi dalam sirkuit tersebut. Pada gambar 8, 1, 2, 3, 1 adalah sebuah sirkuit dengan panjang 3.
8. Keterhubungan
   Simpul v_1 dan v_2 terhubung jika ada lintasan dari v_1 ke v_2, dan graf G disebut graf terhubung jika terdapat lintasan dari v_i ke v_j untuk setiap pasang simpul v_i dan v_j dalam himpunan V.
   Graf berarah G dikatakan terhubung jika graf tidak berarahnya terhubung. Dua buah simpul u dan v terhubung kuat jika ada lintasan berarah dari u ke v dan ada lintasan berarah dari v ke u. Simpul u dan v terhubung lemah jika tidak terhubung kuat namun tetap terhubung pada graf tidak berarahnya. Graf berarah terhubung kuat jika untuk setiap pasang simpul di graf tersebut terhubung kuat. Pada gambar 9, simpul 1 dan 3 terhubung kuat karena terdapat lintasan dari 1 ke 3 (1, 2, 4, 3) dan dari 3 ke 1 (3, 1).
9. Upagraf (*Subgraph*)
   Graf $G_1 = (V_1, E_1)$ adalah upagraf dari $G = (V, E)$ jika V_1 merupakan subset dari V dan E_1 merupakan subset dari E. Graf $G_2 = (V_2, E_2)$ merupakan komplemen upagraf G_1 jika $E_2 = E – E_1$ dan E_2 bersisian dengan tiap simpul V_2.



Gambar 11. Dari kiri ke kanan: Graf G_1, salah satu upagraf G_1, dan komplemen dari upagraf tersebut [4]

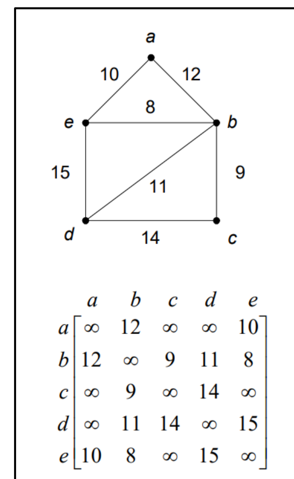
10. Graf berbobot (*weighted graph*)
    Graf berbobot adalah graf yang setiap sisinya diberi sebuah harga/bobot.



Gambar 12. Graf berbobot (*weighted graph*) [4]

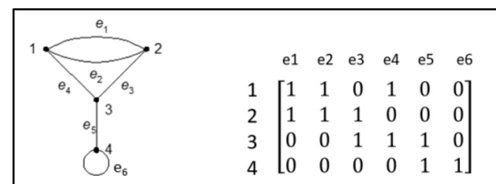
D. Representasi Graf

1. Matriks ketetanggaan (*adjacency matrix*)
   Graf tidak berarah dapat dimodelkan sebagai sebuah matriks A berukuran i x j dengan i = j = jumlah simpul. Jika simpul i dan j bertetangga maka a_{ij} bernilai 1, sedangkan jika tidak bertetangga maka a_{ij} bernilai 0. Khusus untuk graf berbobot, jika i dan j bertetangga, maka nilai a_{ij} sesuai bobot sisi yang menghubungkan kedua simpul tersebut. Untuk graf berarah, nilai a_{ij} adalah 0 jika tidak ada sisi berarah dari simpul j ke i, dan nilai elemen pada diagonal matriks serta a_{ij} dengan simpul i dan j tidak bertetangga adalah ∞.



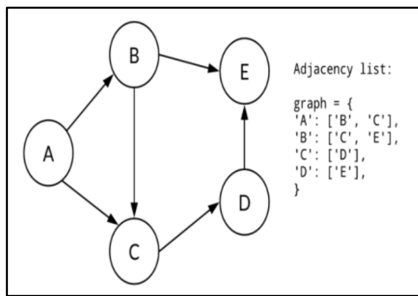
Gambar 13. Matriks ketetanggaan untuk graf berbobot [6]

2. Matriks bersisian (*incidency matrix*)
   Untuk graf tidak berbobot, graf direpresentasikan oleh matriks i x j, dengan i banyak simpul dan j banyak sisi. a_{ij} bernilai 1 jika simpul i bersisian dengan sisi j, dan bernilai 0 jika simpul i tidak bersisian dengan sisi j.



Gambar 14. Matriks bersisian [6]

3. *Adjacency list*
   Untuk graf berarah, setiap simpul didefinisikan simpul tetangganya (yang memiliki sisi berarah dari simpul awal ke simpul tetangga) melalui sebuah tabel.



Gambar 15. *Adjacency list* [7]

E. Shortest Path Problem *dan Algoritma A**

Shortest path problem adalah sebuah permasalahan dalam mencari lintasan terpendek antara sebuah simpul v dan simpul v lain dalam sebuah graf. Ada berbagai algoritma yang dapati digunakan untuk memecahkan masalah ini, namun pada makalah ini penulis akan fokus kepada algoritma A*.

Algoritma A* pada umumnya dipakai untuk mencari lintasan dan melaksanakan proses traversal pada sebuah graf. Algoritma A* mengubah konsep heuristik menjadi algoritma pencarian graf dan merupakan pengembangan dari algoritma Dijkstra dengan beberapa ciri khas dari Breadth-First Search untuk menemukan sebuah lintasan dengan bobot minimum dari simoul awal ke simpul akhir.

Algoritma A* dikenal dengan fungsi estimasi berikut:

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

dimana f(n) adalah total bobot lintasan melalui simpul n, g(n) adalah bobot sejauh ini untuk mencapai simpul n, dan h(n) adalah estimasi bobot dari n ke simpul tujuan yang merupakan bagian heuristik dari persamaan ini. Algoritma ini juga memanfaatkan beberapa list dan *hash table* seperti openlist dan closedlist yang akan dibahas lebih lanjut di bab berikutnya.

Berikut cara kerja algoritma A* secara umum:

1. Tambahkan simpul awal ke openlist
2. Cari simpul dengan bobot paling kecil untuk semua simpul bertetangga
3. Untuk closed list:
    - Iterasi simpul yang bertetangga dengan node terkini
    - Jika simpul tidak dapat diraih, abaikan.
    - *Else*: Jika simpul tidak ada di openlist, pindahkan ke openlist dan hitung fungsi estimasi. Jika simpul ada di openlist, periksa apakah lintasan kurang dari lintasan terkini, dan jika iya, ganti lintasan tersebut menjadi lintasan terkini.
4. Program berhenti jika sudah diraih simpul tujuan atau tidak bisa diraih tujuannya setelah melalui berbagai simpul.

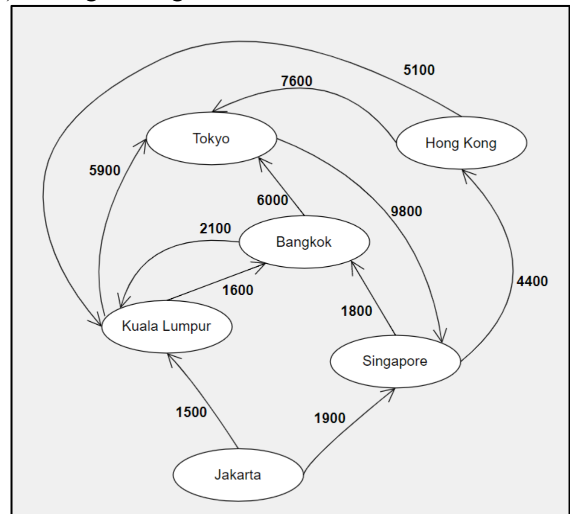
III. APLIKASI ALGORITMA A*

A. Pemodelan Graf Berarah Berbobot

Penulis akan memodelkan sebuah graf berarah berbobot berdasarkan permasalahan rute penerbangan termurah yang sudah dibahas. Simpul-simpul graf merupakan kumpulan kota yang menjadi tempat pemberhentian pesawat. Sisi berarah menggambarkan lintasan penerbangan antara kedua simpul/kota. Bobot merepresentasikan harga tiket tiap penerbangan.

Untuk memudahkan perhitungan, dalam makalah ini, penulis akan membatasi kota-kota yang akan digunakan sebagai simpul graf dan tiap kota tidak terhubung oleh setidaknya sebuah rute penerbangan ke semua kota lain. Tidak ada simpul yang terpencil dan graf tidak kosong. Faktor waktu juga diabaikan dalam persoalan ini. Adapun data harga penerbangan diperoleh dari website AirAsia pada tanggal 9 Desember 2022.

Graf mengandung 6 buah simpul: “Jakarta”, “Kuala Lumpur”, “Singapore”, “Bangkok”, “Hong Kong”, “Tokyo”. Berikut adalah ilustrasi graf beserta bobot harga (dalam ribuan rupiah) masing-masing sisi.



Gambar 16. Model rute penerbangan AirAsia (dokumentasi penulis)

Penulis akan merepresentasikan graf tersebut sebagai *adjacency list* berupa *dictionary/hash table* yang memiliki *key* yang merepresentasikan tiap kota dan *value* berbentuk array yang berisi sebuah pasangan: kota yang bertetangga dan bobot harga. Kota yang bertetangga dalam hal ini artinya terhubung langsung oleh sebuah sisi (sekali penerbangan). Berikut bentuk *adjacency list* berdasarkan model graf di atas.

```
adjacency_list_rute = {
    "Jakarta": [("Kuala Lumpur", 1500), ("Singapore", 1900)],
    "Singapore": [("Bangkok", 1800), ("Hong Kong", 4400)],
    "Kuala Lumpur": [("Bangkok", 1600), ("Tokyo", 5900)],
    "Hong Kong": [("Kuala Lumpur", 5100), ("Tokyo", 7600)],
    "Bangkok": [("Kuala Lumpur", 2100), ("Tokyo", 6000)],
    "Tokyo": [("Singapore", 9800)]
}
```

Gambar 17. *Adjacency list* untuk model graf di atas (dokumentasi penulis)

B. Implementasi Algoritma A dalam Bahasa Python*

Penulis mengimplementasikan tahap algoritma A* untuk graf berbobot berarah yang sudah dibuat di subbab sebelumnya

dengan menggunakan bahasa pemrograman Python. Adapun tahapannya adalah sebagai berikut:

1. Proses inisialisasi:
   a. Buat sebuah class bernama GraphA yang menginisialisasi *adjacent list*.
   b. Definisikan fungsi tetangga yang mengembalikan kota tetangga dari kota asal dengan mengakses *key* yang bersesuaian dengan kota asal pada *adjacent list*.
   c. Asumsikan bahwa tiap simpul memiliki nilai heuristik sama dengan 1 untuk memudahkan perhitungan.

```
class GraphA:
    def __init__(self, adjlist):
        self.adjlist = adjlist
    def tetangga(self, kotaseberang):
        # Daftar kota tetangga/adjacent dari kota asal
        return self.adjlist[kotaseberang]
    def heuristic(self, banyakkota):
        # Inisalisasi h(n)
        H = {"Jakarta": 1, "Singapore": 1, "Kuala Lumpur": 1,
            "Hong Kong": 1, "Bangkok": 1, "Tokyo": 1}
        return H[banyakkota]
```

Gambar 18. Inisialisasi a, b, c (dokumentasi penulis)

   d. Buat fungsi rute_termurah_astar dengan parameter kota asal (asal) dan kota tujuan (tujuan)
   e. Inisialisasi openlist dan closedlist sebagai list. Variabel openlist sudah diinisialisasi dengan asal. Openlist menyimpan kota-kota yang sudah dikunjungi namun tetangga dari kota-kota tersebut belum semuanya diperiksa. Closedlist menyimpan kota-kota yang sudah dikunjungi dan semua tetangganya telah diperiksa.
   f. Buat sebuah *hash table* yang memetakan ketetangaan dari semua node dan inisialisasi nilai pertama dengan variabel asal.
   g. Buat sebuah hash table yang mengandung bobot terkini dari asal ke kota yang lain. Inisialisasi nilai pertama dengan variabel asal yang bernilai 0.

```
def rute_termurah_astar(self, asal, tujuan):
    openlist = [asal]
    closedlist = []
    count = 0
    adjmap = {}
    adjmap[asal] = asal
    harga_terkini = {}
    harga_terkini[asal] = 0
```

Gambar 19. Inisialisasi d, e, f, g (dokumentasi penulis)

2. Masuk ke dalam loop selama openlist tidak kosong. Untuk semua kota di openlist, cari sebuah kota dengan nilai f(n) terkecil dalam fungsi evaluasi. Jika n tidak ditemukan, lintasan tidak ditemukan.

```
while len(openlist) > 0:
    n = None
    for kota in openlist:
        if n == None or harga_terkini[kota] + self.heuristic(kota) < harga_terkini[n] + self.heuristic(n):
            n = kota
    if n == None:
        print("Rute termurah tidak ditemukan.")
        return None
```

Gambar 20. Masuk ke dalam loop while (dokumentasi penulis)

3. Iterasi kota tetangga dari node terkini. Jika kota terkini tidak ada di dalam openlist maupun closedlist, tambahkan kota tersebut ke openlist, dan tandai n sebagai value kota tersebut pada *adjacency map*.
   Jika kota terdapat di openlist atau closedlist, periksa apakah bobot dari a ke n lebih kecil. Jika iya, update adjmap dan harga_terkini. Jika a di dalam closedlist, pindahkan ke openlist.
   Program akhirnya keluar dari for loop. Pindahkan n dari openlist ke closedlist karena semua kota tetangga sudah diperiksa.

```
    for (a, bobot) in self.tetangga(n):
        if a not in openlist and a not in closedlist:
            openlist.append(a)
            adjmap[a] = n
            harga_terkini[a] = harga_terkini[n] + bobot
        else:
            if harga_terkini[a] > harga_terkini[n] + bobot:
                harga_terkini[a] = harga_terkini[n] + bobot
                adjmap[a] = n
                if a in closedlist:
                    closedlist.remove(a)
                    openlist.append(a)
    openlist.remove(n)
    closedlist.append(n)
```

Gambar 21. Program for loop pengecekan tetangga (dokumentasi penulis)

4. Jika node terkini sudah mencapai node tujuan (tujuan), akan direkonstruksi lintasan dari tujuan ke asal, dan akan dikembalikan list yang sudah dibalik. Variabel count menyimpan bobot total harga yang diperoleh dari rute yang dipilih.

```
if n == tujuan:
    count = harga_terkini[n]
    reconstructPath = []
    while adjmap[n] != n:
        reconstructPath.append(n)
        n = adjmap[n]
    reconstructPath.append(asal)
    reconstructPath.reverse()
    print("Biaya penerbangan adalah Rp%d.000" % count)
    print("Rute termurah adalah:", asal, end='')
    for stops in reconstructPath:
        if stops != asal:
            print(" -->", stops, end='')
    return reconstructPath
```

Gambar 22. Program meng-*output* rute dari adjmap dan harga total rute penerbangan dari harga_terkini (dokumentasi penulis)

Pada akhirnya, program akan diuji dengan input dari

pengguna untuk memilih rute penerbangan yang paling murah dari kota asal ke kota tujuan. Dalam contoh di bawah ini, penulis memilih Jakarta sebagai kota asal dan Tokyo sebagai kota tujuan.

```
adjacency_list_rute = {
    "Jakarta": [("Kuala Lumpur", 1500), ("Singapore", 1900)],
    "Singapore": [("Bangkok", 1800), ("Hong Kong", 4400)],
    "Kuala Lumpur": [("Bangkok", 1600), ("Tokyo", 5900)],
    "Hong Kong": [("Kuala Lumpur", 5100), ("Tokyo", 7600)],
    "Bangkok": [("Kuala Lumpur", 2100), ("Tokyo", 6000)],
    "Tokyo": [("Singapore", 9800)]
}

print("Masukkan kota asal:")
asal = input()
print("Masukkan kota tujuan:")
tujuan = input()

graflain = GraphA(adjacency_list_rute)
graflain.rute_termurah_astar(asal, tujuan)
```

Gambar 23. Main program (dokumentasi penulis)

```
Masukkan kota asal:
Jakarta
Masukkan kota tujuan:
Tokyo
Biaya penerbangan adalah Rp7400.000
Rute termurah adalah: Jakarta --> Kuala Lumpur --> Tokyo
PS C:\Users\USER\Desktop\newflight> 
```

Gambar 24. Interface untuk pengguna (dokumentasi penulis)

Jadi, untuk mencapai Tokyo, rute yang paling murah untuk diakses penumpang adalah Jakarta ke Kuala Lumpur ke Tokyo, dengan total biaya Rp7400.000 (bobot pada gambar di bawah dalam ribuan rupiah). Rute yang dimaksud di-*highlight* sebagai berikut.

Gambar 25. Rute penerbangan termurah dari Jakarta ke Tokyo (dokumentasi penulis)

IV. Kesimpulan

Sebagai sebuah struktur, graf merupakan pilihan yang paling cocok untuk merepresentasikan hubungan-hubungan antar objek diskrit, baik berarah maupun tidak berarah. Salah satu kegunaan graf berarah yakni memetakan rute penerbangan antarkota. Besaran tiap hubungan juga dapat direpresentasikan melalui graf berbobot. Algoritma A* sangat bermanfaat untuk memecahkan permasalahan seperti *pathfinding* dalam sebuah graf berbobot berarah. Solusi permasalahan tersebut dapat diterapkan dalam penentuan rute penerbangan dengan biaya termurah, dengan kota-kota pemberhentian pesawat sebagai simpul dan rute penerbangan dengan sisi beserta harga tiap penerbangan sebagai bobot tiap sisi.

V. Ucapan Terima Kasih

Puji syukur penulis panjatkan ke pada Allah SWT, karena berkat rahmat-Nya, penulis dapat menyelesaikan makalah berjudul “Implementasi Algoritma A* untuk Menemukan Rute Penerbangan Termurah untuk Maskapai AirAsia”. Makalah ini dapat diwujudkan berkat para dosen Matematika Diskrit. Oleh karena itu penulis menyampaikan terima kasih kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc. sebagai dosen K01, dan teman-teman yang telah memberikan saran dan dukungan dalam pembuatan makalah ini.

VI. Lampiran

Berikut adalah link repository github implementasi algoritma A*: https://github.com/BreezyDR/Makalah-Matdis-13521048

Daftar Pustaka

[1] https://aviationforaviators.com/2022/02/16/a-brief-history-of-the-airasia-group/, diakses pada 9 Desember 2022, pukul 08.36.
[2] https://www.airasia.com/aa/inflight-comforts/id/id/seats.html, diakses pada 9 Desember 2022, pukul 09.02.
[3] Wengrow, Jay. 2020. *A Common-Sense Guide to Data Structure and Algorithms: Level Up Your Core Programming Skills*. 2nd ed. North Carolina: The Pragmatic Bookshelf.
[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf, diakses pada 9 Desember 2022, pukul 09.46.
[5] https://mathworld.wolfram.com/SimpleGraph.html, diakses pada 9 Desember 2022, pukul 10.35.
[6] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf, diakses pada 10 Desember 2022, pukul 00.02
[7] https://guides.codepath.com/compsci/Graphs, diakses pada 10 Desember 2022, pukul 00.34
[8] https://brilliant.org/wiki/a-star-search/, diakses pada 10 Desember 2022, pukul 09.32.
[9] https://www.airasia.com/en/gb, diakses pada 10 Desember 2022, pukul 10.00.
[10] https://www.edureka.co/blog/a-search-algorithm/, diakses pada 10 Desember 2022, pukul 10.36.

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, 11 Desember 2022

Muhammad Farrel Danendra Rachim
13521048