Penerapan Algoritma A* dalam Konsep *Matching* pada Aplikasi Kencan

Denise Felicia Tiowanni - 13522013[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522013@std.stei.itb.ac.id

***Abstract*— Aplikasi kencan telah menjadi platform penting dalam menemukan pasangan potensial secara digital. Proses *matching* dalam aplikasi kencan melibatkan analisis berbagai faktor seperti minat, lokasi, dan preferensi pribadi untuk merekomendasikan pasangan yang cocok. Algoritma A* dapat meningkatkan efisiensi dan akurasi proses ini dengan memanfaatkan fungsi biaya dan heuristik untuk mencari pasangan terbaik. Makalah ini membahas penerapan algoritma A* dalam proses *matching*, termasuk implementasi teknis dan keuntungan yang diperoleh, serta studi kasus dalam skenario dunia nyata.**

Keywords—Algoritma, A*, Matching, Aplikasi Kencan, Heuristik, Pencarian Jalur, Optimasi, Minat.

I. Pendahuluan

Aplikasi kencan telah menjadi fenomena global yang memudahkan individu untuk mencari pasangan potensial melalui platform digital. Dalam beberapa tahun terakhir, penggunaan aplikasi kencan meningkat secara signifikan, dengan jutaan pengguna di seluruh dunia yang mencari hubungan romantis ataupun pertemanan baru dalam hidup mereka. Beberapa aplikasi kencan yang populer termasuk Tinder, Bumble, dan banyak lagi. Aplikasi ini menggunakan berbagai algoritma untuk mencocokkan pengguna berdasarkan profil dan preferensi mereka.

Matching adalah proses di mana aplikasi kencan mengidentifikasi dan merekomendasikan pasangan potensial kepada pengguna berdasarkan kriteria tertentu. Proses *matching* biasanya melibatkan analisis berbagai faktor seperti usia, lokasi geografis, minat, nilai, dan preferensi pribadi lainnya. Tujuan utama dari *matching* adalah untuk meningkatkan kemungkinan bahwa dua pengguna akan tertarik satu sama lain dan membentuk hubungan yang bermakna.

Gambar 1. Illustrasi *Match* pada Aplikasi Kencan

Sumber: https://www.facebook.com/javirroyo/photos/a.205929378972/10159332720233973/?type=3

Proses *matching* pada aplikasi kencan umumnya melibatkan langkah-langkah seperti pengumpulan data profil, analisis data, pencarian kandidat, penilaian kecocokan, rekomendasi, serta interaksi pengguna, dimana seseorang dikatakan *match* atau cocok dengan orang lain dalam konteks aplikasi kencan jika mereka memiliki kesamaan atau kecocokan dalam beberapa aspek penting seperti minat dan hobi, nilai dan tujuan hidup, kepribadian, lokasi geografis, ataupun preferensi pribadi lainnya.

Algoritma A* dapat digunakan untuk membantu proses matching dalam aplikasi kencan dengan meningkatkan efisiensi pencarian, optimalitas, serta penggabungan faktor kecocokan dengan menggunakan fungsi biaya dan heuristic sehingga dapat memberikan rekomendasi pasangan yang lebih akurat dan relevan.

Dalam makalah ini, akan dieksplorasi lebih lanjut bagaimana algoritma A* dapat diterapkan dalam proses *matching* pada aplikasi kencan, termasuk implementasi teknis dan keuntungan yang ditawarkannya. Selain itu, juga akan membahas studi kasus penerapan algoritma ini dalam skenario dunia nyata dan hasil yang diperoleh.

II. Dasar Teori

A. Aplikasi Kencan

Aplikasi kencan adalah platform digital yang dirancang untuk membantu individu menemukan dan berinteraksi dengan pasangan potensial. Aplikasi ini biasanya tersedia di perangkat mobile seperti *smartphone* dan *tablet*, serta pada *website*. Dengan memanfaatkan GPS pada perangkat digital, aplikasi ini dapat memfasilitasi pertemuan antara orang-orang yang mencari hubungan romantis, pertemanan, atau bahkan jaringan sosial.

Aplikasi kencan memulai proses dengan meminta pengguna untuk membuat profil yang mencakup informasi pribadi seperti nama, usia, lokasi, foto, minat, dan preferensi pasangan. Algoritma pencocokan aplikasi kemudian menganalisis data ini untuk menemukan pasangan potensial berdasarkan berbagai kriteria seperti lokasi geografis, minat, hobi, dan nilai. Aplikasi kemudian memberikan rekomendasi pasangan yang sesuai. Jika dua pengguna saling menyukai, mereka bisa mulai berinteraksi melalui fitur pesan dalam aplikasi. Setelah interaksi secara *online* atau daring, pengguna dapat memutuskan untuk bertemu

langsung, dengan tujuan membangun hubungan yang lebih dalam.

B. Route Planning

Route Planning adalah proses menentukan jalur terbaik dari satu titik ke titik lain dalam graf berarah berbobot, seperti peta jalan, jaringan komputer, atau permainan. Ini melibatkan pencarian jalur terpendek atau paling efisien berdasarkan kriteria tertentu. Terdapat beberapa jenis algoritma untuk *route planning*, diantaranya:

a. *Breadth-First Search* (BFS)

Breadth-First Search (BFS) adalah algoritma traversal graf yang mengeksplorasi semua simpul dalam graf pada kedalaman saat ini sebelum berpindah ke simpul pada tingkat kedalaman berikutnya. BFS dimulai dari titik tertentu dan mengunjungi semua tetangganya sebelum berpindah ke tetangga pada tingkat selanjutnya. BFS cocok untuk graf tanpa bobot atau semua bobot sama karena ia menjamin menemukan jalur terpendek jika ada.

b. *Depth-First Search* (DFS)

Depth-First Search (DFS) adalah algoritma traversal graf yang mengeksplorasi sebanyak mungkin sepanjang satu cabang sebelum mundur dan mengeksplorasi cabang lain. DFS dimulai dari satu simpul tertentu dan menjelajahi sejauh mungkin di sepanjang cabang sebelum kembali dan mencari jalan lain yang belum dijelajahi. DFS tidak menjamin menemukan jalur terpendek dalam graf tanpa bobot atau graf dengan bobot yang tidak sama, karena DFS tidak memperhitungkan jarak dari titik awal ke simpul lainnya selama penelusuran.

c. *Uniform Cost Search* (UCS)

Uniform Cost Search (UCS) adalah algoritma pencarian yang digunakan untuk menemukan jalur dengan biaya total terendah dalam graf berbobot. UCS mirip dengan *Breadth-First Search* (BFS) namun lebih umum karena mempertimbangkan bobot atau biaya (*cost*) pada setiap sisi graf. UCS mengeksplorasi simpul yang memiliki biaya kumulatif terendah terlebih dahulu, bukan berdasarkan kedalaman simpul dari titik awal.

d. *Greedy Best-First Search* (G-BFS)

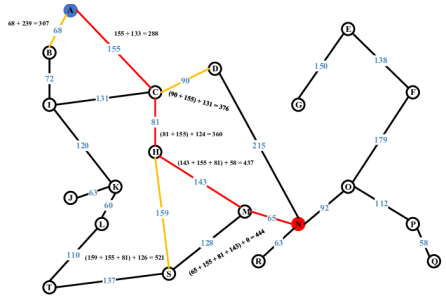
Greedy Best-First Search (G-BFS) adalah algoritma pencarian yang menggunakan heuristik untuk memperkirakan biaya terendah dari simpul saat ini ke tujuan. Algoritma ini berfokus pada perluasan simpul yang tampaknya paling dekat dengan tujuan, berdasarkan fungsi heuristik, dan tidak memperhitungkan biaya jalur yang telah dilalui sejauh ini. Algoritma ini tidak menjamin menemukan jalur terpendek atau paling murah, tetapi sering kali sangat cepat dalam menemukan solusi jika heuristiknya baik.

e. *A-star* (A*)

A-star (A*) adalah algoritma pencarian yang menggabungkan kekuatan *Uniform Cost Search* (UCS) dan *Greedy Best-First Search* (G-BFS). A* mengeksplorasi jalur yang kemungkinan besar memiliki biaya total terendah dari titik awal ke tujuan, menggunakan fungsi heuristik untuk memperkirakan biaya dari simpul saat ini ke tujuan dan biaya sebenarnya yang telah dikeluarkan dari titik awal.

C. Algoritma A*

Algoritma pencarian A* disebut sebagai algoritma pencarian terbaik yang dapat secara efisien menentukan jalur dengan biaya terendah antara dua node baik dalam graf berbobot terarah dengan bobot sisi non-negatif. Algoritma ini merupakan varian dari algoritma Dijkstra.



Gambar 2. Illustrasi Pencarian dengan A*

Sumber: https://www.researchgate.net/publication/348997309/figure/fig5/AS:1143128 1104488654@1670093769058/Testing-A-Star-algorithm-on-graph.png

Fungsi evaluasi, $f(n)$, untuk algoritma pencarian A* adalah sebagai berikut:

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

dimana dalam fungsi tersebut, $g(n)$ merepresentasikan biaya untuk sampai ke node n dan $h(n)$ merepresentasikan estimasi biaya untuk sampai ke node tujuan dari node n. Algoritma ini dapat dikatakan sebagai penggabungan algoritma *Uniform Cost Search* (UCS) dan *Greedy Best-First Search*. UCS hanya mengevaluasi nilai $f(n)$ nya dengan $g(n)$ sedangkan Greedy Best-First Search mengevaluasi nilainya dengan $h(n)$.

Agar heuristik $h(n)$ dapat diandalkan, $h(n)$ harus memenuhi syarat *admissible*. *Admissible* berarti bahwa fungsi heuristik $h(n)$ tidak pernah melebih-lebihkan biaya sebenarnya untuk mencapai tujuan dari node n. Dengan kata lain, nilai $h(n)$ selalu sama dengan atau lebih kecil daripada biaya sebenarnya dari node n ke tujuan.

Heuristik yang *admissible* penting karena menjamin bahwa algoritma A* akan menemukan jalur terpendek. Ini terjadi karena $h(n)$ memberikan perkiraan yang realistis dan tidak terlalu optimistis, sehingga algoritma A* tetap fokus pada jalur yang benar-benar mungkin menjadi jalur terpendek.

Berikut adalah langkah-langkah implementasi algoritma A* secara umum:

1. Siapkan list terbuka yang berisi node awal (nantinya akan berisi node-node yang belum dievaluasi).

2. Siapkan list tertutup yang kosong (nantinya akan berisi node-node yang sudah dievaluasi).
3. Tetapkan nilai biaya awal $g(n)$ dari node awal.
4. Pilih node dengan nilai $f(n) = g(n) + h(n)$ terendah dari list terbuka.
5. Jika node saat ini adalah tujuan, rekonstruksi jalur dari node awal ke node tujuan melalui node induk.
6. Untuk setiap tetangga dari node saat ini, hitung biaya baru $g(n)$ untuk mencapai tetangga. Jika tetangga sudah ada di list tertutup dengan biaya lebih tinggi, lewati. Namun, jika tetangga belum ada di list terbuka atau biaya baru lebih rendah, perbarui tetangga dengan node induk sebagai node saat ini. Tetapkan nilai $g(n)$ dan $f(n)$ yang baru untuk tetangga. Tambahkan tetangga ke list terbuka jika belum ada.
7. Pindahkan node saat ini dari daftar terbuka ke daftar tertutup.
8. Ulangi langkah 4-7 sampai node tujuan ditemukan atau daftar terbuka kosong (yang berarti tidak ada jalur yang ditemukan).

III. Implementasi dan Pengujian

Untuk dapat menggunakan algoritma A* dalam menemukan pasangan terbaik untuk pengguna tertentu dalam suatu aplikasi perjodohan, kita perlu pertama-tama menentukan komponen-komponen utama seperti tetangga, biaya atau *cost*, serta heuristik dari node. Dalam hal ini, node merupakan pengguna yang ingin dicari pasangan terbaik (*match*)-nya. Ketiga komponen utama algoritma A* tersebut dapat direpresentasikan menjadi sebuah kode yang nantinya membantu pengguna mencari pasangan yang paling cocok dengannya.

A. Tetangga

Tetangga dari node merupakan kandidat pasangan potensial untuk pengguna. Dalam kasus pencarian pasangan tercocok pada aplikasi kencan, tetangga dapat didefinisikan sebagai daftar pengguna yang memenuhi kriteria tertentu, seperti tidak sebelumnya ditolak atau diblokir oleh pengguna (*not rejected and not blocked by user*).

```
def get_neighbors(user, users_data):
    """
    mendapatkan kandidat pasangan potensial untuk user tertentu.

    parameters:
    user (dict): profil pengguna saat ini.
    users_data (list): daftar semua profil pengguna dalam aplikasi.

    returns:
    list: daftar kandidat pasangan potensial (profil pengguna lain).
    """
    neighbors = []
    for candidate in users_data:
        if candidate['id'] != user['id']:   # selain diri sendiri
            if not candidate_or_user_has_rejected(user, candidate) and not candidate_or_user_has_blocked(user, candidate):
                neighbors.append(candidate)
    return neighbors


def candidate_or_user_has_rejected(user, candidate):
    # cek apakah pengguna telah "reject" kandidat sebelumnya atau telah di reject oleh kandidat
    return candidate['id'] in user.get('rejected_ids', []) or user['id'] in candidate.get('rejected_ids', [])


def candidate_or_user_has_blocked(user, candidate):
    # cek apakah kandidat telah "block" pengguna sebelumnya dan sebaliknya
    return user['id'] in candidate.get('blocked_ids', []) or candidate['id'] in user.get('blocked_ids', [])
```

Pada potongan kode tersebut,

- Fungsi `get_neighbors` mengambil daftar tetangga (calon pasangan potensial) untuk pengguna tertentu dengan memeriksa apakah pengguna tidak sebelumnya menolak, memblokir, atau diblokir oleh kandidat.
- Fungsi `candidate_or_user_has_rejected` memeriksa apakah pengguna telah sebelumnya menolak kandidat ataupun kandidat telah sebelumnya menolak pengguna.
- Fungsi `candidate_or_user_has_blocked` memeriksa apakah kandidat telah memblokir pengguna atau sebaliknya.

B. Biaya

Biaya mengukur usaha yang diperlukan untuk berinteraksi dengan kandidat pasangan. Dalam konteks ini, biaya dapat berupa kombinasi dari biaya komunikasi dan jarak geografis antara dua pengguna. Biaya komunikasi sendiri didasarkan pada jumlah interaksi yang diinginkan oleh seseorang.

```
def cost(user, candidate):
    """
    menghitung biaya usaha yang diperlukan untuk berinteraksi dan membangun hubungan dengan kandidat.

    parameters:
    user (dict): profil pengguna saat ini.
    candidate (dict): profil kandidat pasangan.

    returns:
    float: biaya usaha.
```

```
    """
    communication_effort = communication_cost(user, candidate)
    distance_effort = geographic_cost(user, candidate)

    return 0.5 * communication_effort + 0.5 * distance_effort

def communication_cost(user, candidate):
    # biaya komunikasi didasarkan pada jumlah interaksi yang diinginkan
    return 1 / (1 + user['desired_interactions'])

def geographic_cost(user, candidate):
    # biaya geografis berdasarkan jarak yang lebih jauh mendapatkan biaya lebih tinggi
    return haversine_distance(user['location'], candidate['location'])

def haversine_distance(loc1, loc2):
    # menghitung jarak Haversine antara dua titik geografis
    import math
    R = 6371  # radius bumi dalam kilometer
    lat1, lon1 = loc1
    lat2, lon2 = loc2
    dlat = math.radians(lat2 - lat1)
    dlon = math.radians(lon2 - lon1)
    a = math.sin(dlat / 2) ** 2 + math.cos(math.radians(lat1)) * math.cos(math.radians(lat2)) * math.sin(dlon / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    return R * c
```

Biaya komunikasi didasarkan pada jumlah interaksi yang diinginkan oleh pengguna. Setiap pengguna dalam aplikasi kencan mungkin memiliki preferensi yang berbeda-beda dalam hal intensitas komunikasi, seperti ‘kencan untuk menikah’, ‘kencan untuk bersenang-senang’, atau opsi-opsi lainnya. Pengguna yang mencari hubungan jangka panjang mungkin menginginkan lebih banyak interaksi dan pengguna yang hanya mencari hubungan kasual mungkin menginginkan lebih sedikit interaksi. Pada potongan kode diatas, fungsi `communication_cost` menghitung biaya komunikasi berdasarkan jumlah interaksi yang diinginkan oleh pengguna.

Jarak geografis antar pengguna juga merupakan faktor penting dalam menentukan biaya. Semakin jauh jarak antara dua pengguna, semakin tinggi biaya usaha yang diperlukan untuk bertemu atau berinteraksi secara langsung. Fungsi `geographic_cost` menghitung biaya geografis berdasarkan jarak Haversine antara lokasi dua pengguna. Fungsi `haversine_distance` menghitung jarak Haversine, yang merupakan metode matematika untuk mengukur jarak antara dua titik di permukaan bumi menggunakan koordinat lintang dan bujur.

Untuk mendapatkan perhitungan akhir biaya, kita menggabungkan setengah dari nilai biaya komunikasi dan setengah lagi dari biaya jarak geografis.

C. Heuristik

Heuristik mengukur kesamaan minat antara pengguna saat ini dan kandidat pasangan. Fungsi heuristik dapat menggunakan berbagai metode untuk mengukur kecocokan, misalnya dengan menghitung jumlah minat yang sama antara dua pengguna.

```
def heuristic(user, candidate):
    """
    menghitung nilai heuristik kecocokan antara pengguna saat ini dengan kandidat.

    parameters:
    user (dict): profil pengguna saat ini.
    candidate (dict): profil kandidat pasangan.

    returns:
    float: nilai heuristik kecocokan.
    """
    interest_score = common_interests(user, candidate)
    return interest_score + 0.3

def common_interests(user, candidate):
    # hitung skor berdasarkan kesamaan minat
    common = set(user['interests']).intersection(set(candidate['interests']))
    return len(common) / len(set(user['interests']).union(set(candidate['interests'])))
```

Pada potongan kode tersebut,

- Fungsi `common_interests` menghitung persentase kesamaan minat antara pengguna dan kandidat pasangan.
- Fungsi `heuristic` menambahkan sedikit konstanta (0.3) untuk memastikan bahwa akan selalu ada sedikit perbedaan dalam nilai heuristik (meskipun minat dan hobi seseorang itu sama, tapi pasti selalu ada yang perbedaan kecil seperti preferensi tertentu atau frekuensi minat yang tidak bisa diukur secara langsung). Konstanta ini membantu algoritma untuk mempertimbangkan variasi kecil tersebut dalam pencarian pasangan terbaik.

D. Algoritma A dalam Mencari Best-Match atau Pasangan Terbaik*

Dengan menggabungkan ketika komponen tersebut, kita dapat membuat sebuah algoritma A* untuk mencari pasangan terbaik seorang pengguna.

```
from queue import PriorityQueue

def astar_search(start_user, goal_test, get_neighbors, heuristic, cost, users_data):
```

```
    open_set = PriorityQueue()
    open_set.put((0, start_user['id']))

    came_from = {}
    cost_so_far = {}
    came_from[start_user['id']] = None
    cost_so_far[start_user['id']] = 0
    best_match = None
    best_score = float('inf')  # buat best_score diawal dengan nilai tak hingga (infinity)

    while not open_set.empty():
        current_priority, current_user_id = open_set.get()
        current_user = next(user for user in users_data if user['id'] == current_user_id)

        if goal_test(current_user):
            return reconstruct_path(came_from, current_user['id'])

        for neighbor in get_neighbors(current_user, users_data):
            new_cost = cost_so_far[current_user['id']] + cost(current_user, neighbor)
            if neighbor['id'] not in cost_so_far or new_cost < cost_so_far[neighbor['id']]:
                cost_so_far[neighbor['id']] = new_cost
                priority = new_cost + heuristic(current_user, neighbor)
                open_set.put((priority, neighbor['id']))
                came_from[neighbor['id']] = current_user['id']

                # UPDATE BEST MATCH
                if priority < best_score:
                    best_score = priority
                    best_match = neighbor

    return reconstruct_path(came_from, best_match['id']) if best_match else None

def reconstruct_path(came_from, current_id):
    path = []
    while current_id is not None:
        path.append(current_id)
        current_id = came_from[current_id]
    path.reverse()
    return path

def goal_test(user):
    return False  # tidak akan pernah mencapai goal, karena kita akan selalu mencari pasangan terbaik
```

Algoritma dimulai dengan pengguna awal dan menambahkan pengguna ini ke dalam antrian prioritas dengan prioritas awal 0. Setiap iterasi, algoritma mengambil pengguna dengan prioritas terendah dari antrian dan memeriksa tetangganya (calon pasangan potensial). Untuk setiap tetangga, algoritma menghitung biaya baru $g(n)$ yang mencakup biaya komunikasi dan jarak geografis. Jika tetangga tersebut belum pernah dikunjungi atau biaya baru lebih rendah daripada biaya yang telah tercatat sebelumnya, algoritma memperbarui biaya dan prioritas tetangga tersebut, serta menyimpannya dalam antrian prioritas. Algoritma juga melacak jalur dari mana setiap pengguna berasal. Proses ini berlanjut hingga antrian kosong atau pasangan terbaik ditemukan. Hasilnya adalah jalur pengguna yang menunjukkan urutan ID pengguna yang mengarah ke pasangan terbaik berdasarkan kombinasi nilai heuristik kesamaan minat dan nilai biaya interaksi.

Baris `best_score = float('inf')` menginisialisasi variabel `best_score` dengan nilai positif tak terhingga untuk melacak skor terbaik (terendah) yang ditemukan sejauh ini selama eksekusi algoritma A*. Dengan menginisialisasi `best_score` ke tak terhingga, kita memastikan bahwa skor valid apa pun yang dihitung selama eksekusi algoritma akan lebih kecil dari nilai awal ini, sehingga saat algoritma mengiterasi melalui node (pengguna) dan menghitung skor mereka masing-masing berdasarkan fungsi biaya dan heuristik, algoritma membandingkan skor-skor ini dengan `best_score` saat ini. Jika skor yang baru dihitung (prioritas) lebih rendah daripada `best_score`, itu berarti kita telah menemukan pasangan yang lebih baik, dan `best_score` diperbarui dengan nilai baru ini.

Fungsi `goal_test` digunakan dalam algoritma A* untuk memeriksa apakah kondisi tujuan telah tercapai. Dalam konteks algoritma A* yang biasa, fungsi ini mengembalikan True jika kita telah mencapai node tujuan dan False jika belum. Namun, dalam konteks pencarian pasangan terbaik dalam aplikasi perjodohan, fungsi `goal_test` diatur untuk selalu mengembalikan False. Dengan selalu mengembalikan False, kita memastikan bahwa algoritma terus mencari pasangan yang lebih baik tanpa menghentikan pencarian pada kondisi tertentu. Ini karena tujuan utama algoritma di sini bukan untuk mencapai suatu titik tujuan tertentu, tetapi untuk mengevaluasi semua kemungkinan pasangan dan menemukan yang terbaik berdasarkan kriteria yang telah ditentukan.

Sebelum dapat menjalankan algoritma A*, kita perlu terlebih dahulu membuat sebuah data pengguna aplikasi kencan seperti contoh berikut.

```
# data pengguna
users_data = [
    {'id': 1, 'interests': ['music', 'movies', 'sports'], 'location': (52.5200, 13.4050), 'rejected_ids': [], 'blocked_ids': [], 'desired_interactions': 10},
    {'id': 2, 'interests': ['music', 'reading', 'travel'], 'location': (48.8566, 2.3522), 'rejected_ids': [1], 'blocked_ids': [], 'desired_interactions': 5},
    {'id': 3, 'interests': ['sports', 'cooking', 'movies'], 'location': (40.7128, -74.0060), 'rejected_ids': [], 'blocked_ids': [], 'desired_interactions': 8},
    {'id': 4, 'interests': ['music', 'movies', 'sports'], 'location': (52.5200, 13.4050), 'rejected_ids': [1], 'blocked_ids': [], 'desired_interactions': 20},
]
```

Setelah membuat data pengguna aplikasi kencan, kita dapat mengetes algoritma A* dengan menjalankan kode dibawah.

```
# fungsi untuk mencari pengguna berdasarkan ID
def find_user_by_id(user_id, users_data):
    for user in users_data:
        if user['id'] == user_id:
            return user
    return None

# input pengguna
try:
    user_id_input = int(input("Masukkan ID user yang ingin dicari pasangan terbaiknya: "))
except ValueError:
    print("ID yang dimasukkan harus berupa angka.")
    exit()

# cari pengguna berdasarkan ID
start_user = find_user_by_id(user_id_input, users_data)

if start_user is None:
    print("ID user tidak ditemukan.")
else:
    path = astar_search(start_user, goal_test, get_neighbors, heuristic, cost, users_data)

    if path:
        # ambil ID pengguna awal dan pasangan terbaiknya
        start_user_id = start_user['id']
        best_match_id = path[-1]
        print(f"Pasangan yang match dengan pengguna dengan ID {start_user_id} adalah: {best_match_id}")
    else:
        print(f"Tidak ditemukan pasangan yang cocok untuk pengguna dengan ID {user_id_input}.")
```

Berikut adalah contoh implementasi program dengan data pengguna aplikasi kencan yang telah dituliskan sebelumnya dan dengan ID pengguna 1.

```
Masukkan ID user yang ingin dicari pasangan terbaiknya: 1
Pasangan yang match dengan pengguna dengan ID 1 adalah: 3
```

Gambar 3. Hasil Pencarian Pasangan Terbaik Pengguna dengan ID 1

Sumber: Dokumen penulis

Dengan data yang digunakan sebelumnya, kita melihat bahwa pengguna dengan ID 1 memiliki kesamaan paling mirip dengan pengguna dengan ID 4, baik secara heuristik (minat dan hobi) maupun secara biaya (geografis). Namun, karena pengguna dengan ID 1 merupakan pengguna yang sebelumnya telah di-*reject* atau ditolak oleh pengguna dengan ID 4, maka pengguna dengan ID 4 tidak masuk kedalam kandidat 'tetangga' pengguna dengan ID 1. Dengan demikian, didapat bahwa pasangan yang paling cocok dengan pengguna dengan ID 1 adalah pengguna dengan ID 3.

Apabila kita merubah data pengguna aplikasi kencan, khususnya pada pengguna dengan ID 4 menjadi tidak menolak pengguna dengan ID 1 (`{'id': 4, 'interests': ['music', 'movies', 'sports'], 'location': (52.5200, 13.4050), 'rejected_ids': [], 'blocked_ids': [], 'desired_interactions': 20}`), maka kita akan mendapatkan keluaran sebagai berikut.

```
Masukkan ID user yang ingin dicari pasangan terbaiknya: 1
Pasangan yang match dengan pengguna dengan ID 1 adalah: 4
```

Gambar 4. Hasil Pencarian Pasangan Terbaik Pengguna dengan ID 1 ketika Pengguna dengan ID 1 Tidak Ditolak oleh Pengguna dengan ID 4

Sumber: Dokumen penulis

Dan apabila kita mengubah data pengguna aplikasi kencan menjadi sebagai berikut,

```
users_data = [
    {'id': 1, 'interests': ['music', 'movies'], 'location': (52.5200, 13.4050), 'rejected_ids': [2, 3, 4], 'blocked_ids': [2, 3, 4], 'desired_interactions': 10},
    {'id': 2, 'interests': ['reading', 'travel'], 'location': (48.8566, 2.3522), 'rejected_ids': [1, 3, 4], 'blocked_ids': [1, 3, 4], 'desired_interactions': 5},
    {'id': 3, 'interests': ['cooking', 'sports'], 'location': (40.7128, -74.0060), 'rejected_ids': [1, 2, 4], 'blocked_ids': [1, 2, 4], 'desired_interactions': 8},
    {'id': 4, 'interests': ['sports', 'movies'], 'location': (34.0522, -118.2437), 'rejected_ids': [1, 2, 3], 'blocked_ids': [1, 2, 3], 'desired_interactions': 20},
]
```

kita tidak akan mendapatkan pasangan yang cocok untuk pengguna dengan ID 1.

```
Masukkan ID user yang ingin dicari pasangan terbaiknya: 1
Tidak ditemukan pasangan yang cocok untuk pengguna dengan ID 1.
```

Gambar 5. Hasil Pencarian Pasangan Terbaik Pengguna dengan ID 1 ketika Pengguna dengan ID 1 Tidak Memiliki Pasangan yang Cocok

Sumber: Dokumen penulis

IV. KESIMPULAN

Aplikasi kencan modern menggunakan algoritma canggih untuk meningkatkan peluang pengguna menemukan pasangan yang sesuai dengan preferensi dan kriteria mereka (*match*). Algoritma A* memungkinkan pencarian pasangan yang efisien dan optimal dengan menggabungkan biaya aktual $g(n)$ dan perkiraan heuristik $h(n)$. untuk menentukan kecocokan terbaik.

Melalui implementasi algoritma A*, kita mampu:

a. Menentukan kandidat pasangan potensial berdasarkan berbagai faktor seperti kesamaan minat dan jarak geografis.

b. Menggunakan fungsi heuristik yang tepat untuk memperkirakan kecocokan antara pengguna dan kandidat.

c. Menghitung biaya usaha yang diperlukan untuk membangun dan mempertahankan hubungan.

Penerapan algoritma A* dalam aplikasi kencan tidak hanya meningkatkan efisiensi pencarian pasangan tetapi juga memberikan hasil yang lebih relevan dan memuaskan bagi pengguna. Dengan menggabungkan berbagai metrik kecocokan, algoritma ini dapat secara efektif menyaring kandidat dan merekomendasikan pasangan yang memiliki peluang lebih tinggi untuk membentuk hubungan yang bermakna.

Implementasi yang tepat dari algoritma ini dapat meningkatkan pengalaman pengguna dalam aplikasi kencan, membuat proses pencarian pasangan menjadi lebih terstruktur dan terinformasi. Di masa depan, pengembangan lebih lanjut pada fungsi heuristik dan biaya dapat semakin menyempurnakan hasil pencocokan dan mendukung keberhasilan aplikasi kencan dalam membantu pengguna menemukan pasangan yang sesuai.

V. Ucapan Terima Kasih

Dengan penuh rasa syukur, penulis ingin menyampaikan ucapan terima kasih kepada Tuhan Yang Maha Esa atas berkat dan izin-Nya, yang telah memungkinkan penulisan makalah berjudul "*Penerapan Algoritma A* dalam Konsep Matching pada Aplikasi Kencan*" dapat diselesaikan dengan baik. Penulis juga ingin menyampaikan rasa terima kasih kepada para dosen pengampu Mata Kuliah Strategi Algoritma IF2211, yaitu Ibu Dr. Nur Ulfa Maulidevi dan Bapak Dr. Rinaldi Munir.

Tak lupa, penulis ingin menyampaikan terima kasih kepada asisten mata kuliah, teman-teman, serta keluarga yang telah memberikan dukungan luar biasa selama proses penulisan makalah ini. Semua bantuan dan motivasi yang diberikan sangat berarti bagi penulis. Penulis juga ingin menyampaikan terima kasih kepada semua sumber referensi yang telah menjadi landasan utama dalam penulisan makalah ini.

Referensi

[1] R. Munir, "Penentuan Rute (Route/Path Planning) (Bag.1)." Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf, pada 1 Juni 2024.

[2] R. Munir, "Penentuan Rute (Route/Path Planning) (Bag.2)." Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf, pada 1 Juni 2024.

[3] Bumble. (n.d.), "How to Use Filters on Bumble." Diakses dari "https://bumble.com/en/the-buzz/how-to-use-filters-on-bumble", pada 1 Juni 2024.

[4] GeeksforGeeks. (n.d.), "Greedy Best-First Search Algorithm." Diakses dari https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/, pada 1 Juni 2024.

[5] GeeksforGeeks. (n.d.), "Breadth-First Search (BFS) for a Graph." Diakses dari https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/, pada 1 Juni 2024.

[6] GeeksforGeeks. (n.d.), "Depth-First Search (DFS) for a Graph." Diakses dari https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/, pada 1 Juni 2024.

[7] GeeksforGeeks. (n.d.), "Uniform Cost Search (Dijkstra for large graphs)." Diakses dari https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/, pada 1 Juni 2024.

[8] GeeksforGeeks. (n.d.), "A* Search Algorithm." Diakses dari https://www.geeksforgeeks.org/a-search-algorithm/, pada 1 Juni 2024.

[9] GetStream.io. (n.d.), "Dating App Algorithms." Diakses dari https://getstream.io/blog/dating-app-algorithms/, pada 1 Juni 2024.

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, 1 Juni 2024

Denise Felicia Tiowanni
13522013