TELEKONTRAN: Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan
Vol. 12, No. 1, April 2024
DOI : 10.34010/telekontran.v12i1.12645
p-ISSN : 2303 – 2901; e-ISSN : 2654 – 7384

Pemanfaatan Algoritma A-Star dalam Menentukan Rute Wisata di Kota Palangka Raya

Utilisation of A-Star Algorithm in Determining Tourism Routes in Palangka Raya City

Naufal Ihsan Sriyanto*, Ferdy Afriza Maulana, Rivan Aprilian, Efrans Christian, Viktor Handrianus Pranatawijaya
Program Studi Teknik Informatika, Fakultas Teknik Universitas Palangka Raya, Jl. Yos Sudarso, Palangka Raya
Email* : naufalsriyanto05@gmail.com

Abstrak - Pemanfaatan Algoritma A-Star dalam Menentukan Rute Wisata di Kota Palangka Raya merupakan sebuah studi yang bertujuan untuk memberikan solusi dalam menentukan rute wisata di Kota Palangka Raya. Dengan menggunakan algoritma A-Star, penelitian ini menyediakan alat yang efisien untuk menemukan rute terpendek antara tempat wisata yang dipilih, dengan mempertimbangkan jarak dan biaya perjalanan. Metode ini melibatkan inisialisasi graf yang merepresentasikan hubungan antara tempat-tempat wisata, diikuti dengan implementasi algoritma A-Star untuk pencarian rute terbaik. Hasil penelitian menunjukkan bahwa algoritma ini mampu memberikan rute yang optimal dengan estimasi jarak berdasarkan batasan anggaran yang telah diberikan. Diskusi singkat mengenai hasil penelitian menyoroti fleksibilitas algoritma ini dalam menangani permasalahan rute terpendek dengan pembatasan anggaran dalam konteks penentuan rute wisata. Dalam kesimpulannya, penelitian ini memberikan rekomendasi rute wisata dengan jarak yang optimal serta mengakomodasi batasan anggaran yang ditentukan oleh pengguna. Harapannya implementasi selanjutnya dari penelitian ini dapat menjadi potensi dampak positif pada industri pariwisata dan pengalaman wisatan pengunjung dalam merencanakan perjalanan wisata mereka di Kota Palangka Raya.

Kata kunci : A-Star, rute wisata, implementasi A-Star.

Abstract - *Utilisation of A-Star Algorithm in Determining Tourist Routes in Palangka Raya City is a study that aims to provide solutions in determining tourist routes in Palangka Raya City. By using the A-Star algorithm, this study provides an efficient tool to find the shortest route between selected tourist attractions, considering distance and travelling costs. The method involves initialising a graph that represents the relationship between tourist attractions, followed by the implementation of the A-Star algorithm for the best route search. The results show that the algorithm is able to provide an optimal route with distance estimation based on the given budget constraints. A brief discussion of the results highlights the flexibility of this algorithm in handling shortest route problems with budget constraints in the context of tourism routing. In conclusion, this research provides recommended tourist routes with optimal distance and accommodates the budget constraints set by the user. It is hoped that the further implementation of this research can be a potential positive impact on the tourism industry and visitors' tourist experience in planning their tour in Palangka Raya City.*

Keywords *: A-Star, tourist routes, A-Star implementation.*

I. PENDAHULUAN

Kota Palangka Raya merupakan ibu kota dari Provinsi Kalimantan Tengah. Kendati demikian, sebagian besar wilayah Kota Palangka Raya terdiri atas hutan dan dilewati oleh aliran sungai Kahayan. Profil geografis Kota Palangka Raya melahirkan destinasi wisata alam dan budaya yang beragam. Jumlah destinasi wisata yang ditawarkan pun terbilang banyak [1]. Di samping itu, menurut Dinas Kebudayaan dan Pariwisata Kota Palangka Raya, jumlah wisatawan terus bertambah setiap tahunnya.

Destinasi wisata yang beragam dan jumlah

wisatawan yang terus bertambah menyebabkan kombinasi pemilihan rute yang beragam pula yang justru membingungkan para wisatawan. Tidak jarang calon wisatawan kesulitan mencari informasi terkait rute terdekat menuju destinasi wisata dan memutuskan bertanya kepada warga sekitar. Namun, tindakan ini menjadi tidak efisien secara waktu karena tidak semua warga mampu menjawab pertanyaan yang diajukan [1] [2].

Tabel I. Data dari Dinas Kebudayaan dan Pariwisata Kota Palangka Raya

Tahun	Jumlah Wisatawan
2020	102.521
2021	125.840
2022	248.008
2023	220.314

Dengan berbagai permasalahan yang muncul, diperlukan sebuah metode untuk menentukan rute optimal agar biaya perjalanan dan waktu menjadi efisien. Algoritma A-Star adalah algoritma pencarian yang sangat akurat dalam menemukan rute tercepat karena menghitung simpul-simpul terkecil [2]. Algoritma ini juga mengurangi biaya total dan memberikan solusi terbaik dalam waktu yang optimal dalam kondisi yang tepat. Pencarian jalur terpendek adalah solusi yang tepat untuk menentukan jalur mana yang paling cepat untuk dilalui atau dilintas [3] [4]. Dengan adanya jalur terpendek yang tersedia, perjalanan ke lokasi tempat wisata akan lebih efisien dalam hal waktu yang diperlukan.

Metode A-Star, yang pertama kali dikemukakan oleh Peter Hart, Nils Nilsson, dan Bertram Raphael pada tahun 1968, digunakan untuk menemukan rute terdekat, jarak tempuh terpendek, dan waktu tempuh tercepat. Awalnya disebut sebagai Algoritma A, namun kemudian ditingkatkan dengan heuristik untuk menjadi A-Star (A). Algoritma A-Star adalah salah satu metode pencarian rute yang optimal dan lengkap, artinya dapat menghasilkan rute terbaik dan mencapai tujuan yang diinginkan [6]. Dengan kemampuan ini algoritma A-Star menjadi metode yang tepat untuk menangani permasalahan pencarian rute optimal.

Pada penelitian sebelumnya, Diana dan Herdiana memeriksa algoritma A-Star dan A-Star yang ditimbang dalam berbagai lingkungan pengujian. Mereka menemukan bahwa algoritma A-Star ditimbang lebih baik untuk mencari rute yang ideal daripada algoritma A-Star yang ditimbang; namun, algoritma yang ditimbang menghasilkan waktu pencarian yang lebih lama daripada algoritma A-Star. Analisis perbandingan ini menunjukkan algoritma A-Star unggul dalam menentukan rute optimal. Keunggulan ini membuat algoritma A-Star dapat digunakan dalam berbagai skenario rute wisata [2].

Algoritma A-Star adalah sebuah algoritma yang memperbaiki algoritma BFS (*Breadth-First Search*), dengan memodifikasi fungsi heuristiknya. Dalam algoritma A-Star, pencarian dilakukan dengan memilih simpul baru berdasarkan biaya total yang diperkirakan untuk mencapai tujuan. Mereka mengatakan bahwa algoritma BFS melakukan pencarian menyeluruh dari simpul awal ke semua simpul tetangganya [3].

Algoritma A-Star digunakan untuk menemukan rute terpendek dari dua titik. Purnama et al. (2018) melakukan pemanfaatan algoritma A-Star untuk membantu pengguna dalam menentukan wisata kuliner di Bandarlampung. Penelitian ini menunjukkan algoritma A-Star mampu memberikan rekomendasi destinasi berdasarkan rute dengan jarak terpendek [6].

Mekanisme kerja algoritma A-Star dapat dengan mudah diilustrasikan dengan menggunakan graf. Penggunaan graf dapat mempermudah pemahaman terkait cara kerja algoritma A-Star. Namun, kompleksitas waktu yang dihasilkan algoritma A-Star sangat bergantung pada heuristiknya [10].

Rizky et al. (2020) melakukan penelitian untuk menemukan jalan paling dekat ke tempat makan di Menes Pandeglang, Banten. Penelitian tersebut menghasilkan tiga alternatif rute terdekat dan dari tiga rute tersebut terdapat satu rute dengan jarak tempuh terpendek [15].

Beberapa penelitian terdahulu menunjukkan pemilihan rute menggunakan algoritma A-Star terbukti dapat membantu menentukan rute dengan jarak mininum. Namun, penelitian terdahulu hanya mempertimbangkan jarak sebagai faktor dalam mencari jalur atau rute yang optimal. Dalam pencarian rute wisata, jarak tidak selalu menjadi tolak ukur para wisatawan untuk memilih rute optimal. Biaya juga termasuk faktor primer dalam menentukan jalur atau rute wisata yang optimal. Di sisi lain, tujuan dari penelitian ini adalah menentukan rute terbaik dengan biaya yang sudah ditentukan oleh pengguna. Dengan mempertimbangkan biaya sebagai faktor penentuan rute optimal, wisatawan dapat menentukan rute dengan keseimbangan antara

jarak dan biaya. Hal ini mendorong dilakukannya pengujian algoritma A-Star dalam menentukan rute optimal berdasarkan titik acuan yaitu, jarak dan biaya.

Tujuan penelitian ini adalah memberikan rute yang optimal untuk wisatawan berdasarkan biaya dan jarak. Parameter yang digunakan untuk menguji rute optimal adalah jarak dan biaya. Parameter biaya digunakan untuk membatasi eksplorasi rute sesuai dengan anggaran yang dimiliki pengguna. Sedangkan, parameter jarak digunakan untuk menentukan rute dengan jarak terpendek. Harapannya, keluaran yang dihasilkan tidak hanya rute yang optimal berdasarkan jarak namun juga secara biaya.

Pada bagian selanjutnya akan disusun seperti berikut: Bagian 2 akan membahas persiapan untuk membuat implementasi algoritma. Bagian 3 akan menampilkan hasil dan analisis penelitian, dan bagian 4 akan menyampaikan kesimpulan.

II. METODOLOGI

Pada bab ini akan dibahas persiapan awal serta metode yang diperlukan untuk memanfaatkan dan menyesuaikan algoritma A-Star dalam menentukan rute optimal. Persiapan dimulai dengan menentukan sebuah formula untuk menunjang implementasi algoritma. Kemudian, proses pengumpulan data diambil dari *Google Maps*. Data tersebut berupa informasi terkait jarak dan biaya wisata. Data yang sudah didapat dimanfaatkan untuk merancang sebuah graf yang merepresentasikan keterhungan antar destinasi wisata.

A. Algoritma A-Star

Fungsi *heuristic* algoritma A-Star melibatkan penilaian *heuristic*, yang dapat menilai setiap titik, dan menghasilkan solusi yang diinginkan [5]. Nilai *heuristic* algoritma A-Star mempersempit ruang pencarian dan berfungsi sebagai dasar atau prinsip untuk mencapai tujuan yang efektif [11]. Berikut ini adalah notasi yang digunakan oleh algoritma A-Star:

$$f(n) = g(n) + h(n) \quad \text{.... (1)}$$

Kombinasi nilai $g(n)$, yang merupakan nilai untuk mencapai simpul n dari simpul awal, dan nilai $h(n)$ adalah perkiraan untuk mencapai tujuan simpul n. Hal ini memungkinkan teknik ini untuk menilai setiap titik [13].

Dalam kalkulasi manual dibutuhkan formula untuk membantu memecahkan masalah optimisasi rute. Dengan menggabungkan kedua faktor bobot dengan nilai jarak dan biaya aktual, maka didapatkan formula yang menghasilkan nilai heuristik yang memberikan perkiraan biaya total yang diperlukan untuk mencapai tujuan.

$$h(n) = 0{,}5 * dist \quad \text{.... (2)}$$

Notasi (2) merupakan sebuah formula yang digunakan dalam algoritma A-Star untuk menghitung nilai heuristik dari suatu simpul dalam graf. Nilai $h(n)$ adalah nilai heuristik dari simpul n. Kemudian, nilai $dist$ adalah jarak dari simpul n ke simpul tujuan. Sedangkan, Nilai 0,5 digunakan karena dianggap sebagai nilai yang cukup seimbang dalam menimbang antara biaya sebenarnya yang telah ditempuh dan perkiraan biaya yang tersisa ke tujuan.

$$p(n) = q(n) + r(n) \quad \text{.... (3)}$$

Notasi (3) digunakan untuk mengakumulasi biaya yang telah dikeluarkan selama eksplorasi simpul. Nilai $p(n)$ merupakan total akumulasi dari biaya simpul saat ini dengan biaya ke simpul tetangga. Dengan demikian, nilai $q(n)$ adalah biaya simpul saat ini yang telah terakumulasi dan nilai $r(n)$ adalah biaya simpul saat ini dengan biaya simpul tetangga. Perhitungan ini dapat membantu program dalam menentukan rute optimal dengan pembatasan anggaran.

B. Pengumpulan Data

Pada penelitian ini pengumpulan data diambil dari *Google Maps*. Data yang dikumpulkan adalah data jarak antar destinasi wisata dan biaya wisata. Serangkaian data ini bukanlah data mentah, melainkan data yang sudah diproses untuk dapat ditampilkan pada pengguna. Data yang jarak dan biaya dapat diolah menjadi himpunan simpul dan himpunan *edge*.

Pada penelitian ini terdapat 20 destinasi wisata dengan estimasi biaya yang beragam. Adanya tolak ukur biaya ini dapat menghasilkan solusi rute optimal yang seimbang berdasarkan biaya dan jarak. Adapun daftar destinasi wisata di Kota Palangka Raya pada **Tabel II**.

Tabel II. Daftar destinasi dan harga wisata Kota Palangka Raya

Ikon	Destinasi	Biaya
A	Hotel Aurila	314.500 IDR
B	Saung Telaga Rindu	50.000 IDR
C	RM. Pelangi 3	50.000 IDR
D	Wisata Flamboyan Bawah	-
E	Bundaran Burung	-
F	Dermaga Kereng Bangkirai	5.000 IDR
G	Pesona Alam Lestari	20.000 IDR
H	Bundaran Besar	-
I	Tugu Soekarno	-
J	Jembatan Kahayan	-
K	Kampung Lauk	50.000 IDR
L	Kum Kum & Taman Wisata	70.000 IDR
M	Museum Balanga	2.500 IDR
N	Kalawa Waterpark	35.000 IDR
O	Wisata Agro Larosa	35.000 IDR
P	Sakuyan Side	100.000 IDR
Q	RM. Ayam Penyet	50.000 IDR
R	Dango Playing Ground	50.000 IDR
S	Wisata ATV Tanduhan	60.000 IDR
T	Food Container	120.000 IDR

Daftar destinasi pada **Tabel II** berisi informasi biaya wisata dan ikon atau simbol representasi destinasi tersebut dalam graf. Adapun informasi terkait koneksi antar simpul antar destinasi dapat digambarkan dalam graf pada **Gambar 1**.

Daftar destinasi dan harga tersebut dapat dijadikan informasi *heuristic* yang akan diolah dengan algoritma A-Star. Oleh karena itu, algoritma A-Star melihat baik jarak tempuh maupun biaya destinasi.

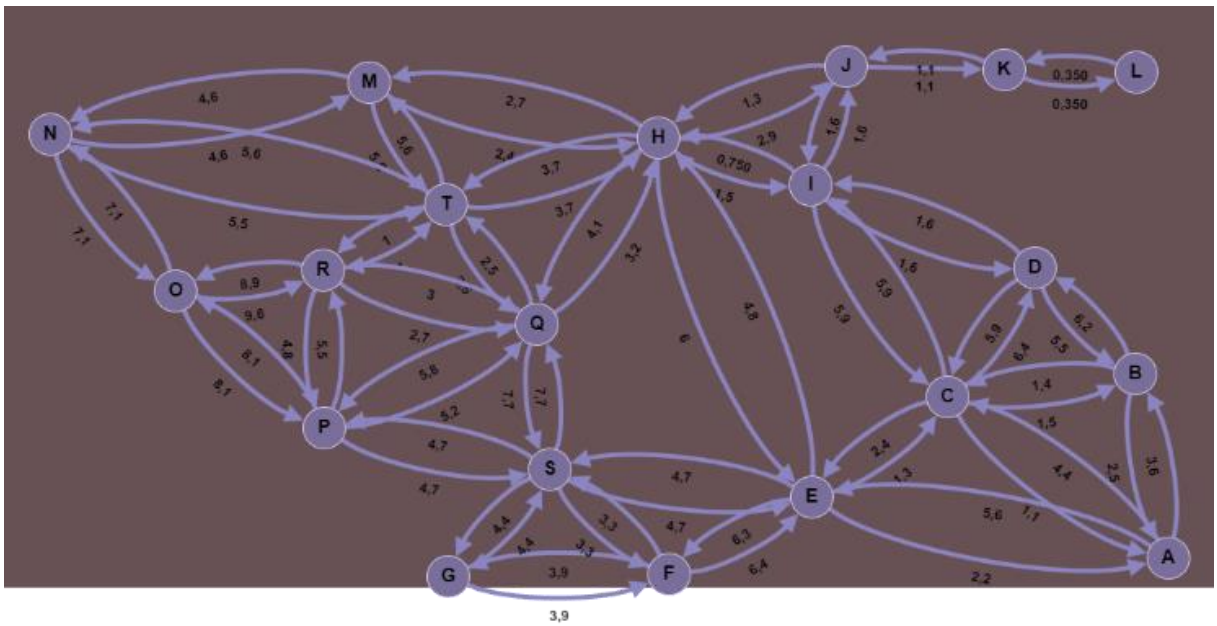
C. Pseudo-code

Pada bagian awal program akan dilakukan inisialisasi struktur data dan variabel yang dibutuhkan seperti *Node* dan *Edge* sebagai struktur data global. Algoritma A-Star dapat dilihat pada **Gambar 1**.

```
function AStar(start, end,
optimalPath, placeNames, cost)
    numDestinations = size of
placeNames

    dist = array of size
numDestinations initialized with
infinity
    cost = array of size
numDestinations initialized with
infinity
    cost[start] = 0
    prev = array of size
numDestinations initialized with -1
    priorityQueue pq
```

Fungsi pada *pseudo-code* di atas diawali dengan inisialisasi variabel dan struktur data. Inisialisasi *array* untuk menyimpan jarak, biaya, dan simpul sebelumnya. Variabel untuk menyimpan simpul-simpul yang akan dieksplorasi ke dalam *priority queue*.



Gambar 1. Graf destinasi wisata

```
pq.push ({calculateHeuristic(start,
end), start})
    dist [start] = 0
    cost [start] = 0
```

Sebelum algoritma dijalankan, titik awal dimasukkan dalam *priority queue* dengan nilai *heuristic* biaya dan jarak yang sesuai. Inisialisasi titik awal pada antrian dilakukan agar algoritma mempunyai titik awal untuk memulai pencarian.

```
while pq is not empty
        current = pq.pop ()

        if current is equal to end
              break
```

Algoritma dijalankan dengan perulangan *while* yang akan terus berjalan selama *priority queue* tidak kosong. Pada setiap iterasi, simpul dengan potensi tertinggi diambil untuk dieksplorasi lebih lanjut.

```
for each neighbor in neighborsof
current
         newDist = dist[current] +
distance to neighbor
         newCost = cost[current] +
cost to neighbor

        if newDist < dist[neighbor]
OR (newDist is equal to
dist[neighbor] AND newCost <
cost[neighbor])
            dist[neighbor] =
newDist
            cost[neighbor] =
newCost
            prev[neighbor] =
current
            pq.push({nenDist +
calculateHeuristic(neighbor, end),
neighbor})
```

Algoritma akan melakukan pencarian melalui simpul tetangga-tetangga dan akan memperbarui jarak dan biaya jika memiliki nilai lebih rendah. Tetangga yang memiliki nilai lebih rendah akan dimasukkan ke dalam *priority queue*.

Sebelum merekonstruksi optimal algoritma akan berhenti jika menemukan rute yang optimal. Namun, jika algoritma berhasil menentukan rute optimal, maka rute tersebut akan direkonstruksi.

```
if dist[end] is infinity OR
cost[end] is infinity
        output "No route found from
start to end."
        return

    optimalPath.clear()
    current = end
    while current is not equal to
start
        add current to optimalPath
        current = prev[current]

    add start to optimalPath
    reverse optimalPath
```

III. HASIL DAN PEMBAHASAN

Pada bab sebelumnya telah dilakukan persiapan awal yang bertujuan untuk merancang kerangka program dan memberikan gambaran implementasi algoritma A-Star. Implementasi algoritma A-Star dimulai dengan inisialisasi pustaka pemprograman. Inisialisasi pustaka bertujuan untuk menyediakan pustaka yang dapat menunjang pembuatan dan implementasi algoritma A-Star dalam program.

Implementasi algoritma A-Star pada bahasa C++ menggunakan beberapa *library* yaitu *iostream*, *vector*, *limits*, *algorithm*, *queue*, *cmath*, dan *ctime*. Pada awal program dilakukan *include library* seperti pada **Gambar 2**.

```
1   #include <iostream>
2   #include <vector>
3   #include <limits>
4   #include <algorithm>
5   #include <queue>
6   #include <cmath>
7   #include <ctime>
```

Gambar 2. Pustaka C++

Pustaka *iostream* merupakan pustaka standar yang menyediakan fungsi-fungsi untuk *input* dan *output*. Pustaka ini biasanya digunakan untuk memanipulasi tata letak *output*. Pustaka *vector* digunakan untuk mengelola struktur data yang dinamis yang dapat bertambah dan berkurang saat program berjalan. *Vector* dapat berguna untuk menyimpan informasi tentang tujuan, jarak dan biaya. Pustaka *limits* digunakan untuk menampilkan batasan nilai maksimum dan minimum untuk tipe data tertentu. Pustaka *algorithm* menyediakan berbagai fungsi algoritma generik untuk bekerja dengan struktur data seperti

vector. Pustaka *queue* menyediakan fungsi untuk bekerja dalam struktur data antrian dengan prinsip *First-in-First-out*. Pustaka *cmath* menawarkan fungsi matematika seperti fungsi *trigonometri*, *logaritma* dan nilai absolut. Pemanfaatan pustaka ini dapat digunakan dalam perhitungan algoritma *A-Star*. Pustaka *ctime* menyediakan fungsi untuk dapat bekerja dengan waktu dan tanggal. Dengan ini, pustaka ini berguna untuk mengukur waktu eksekusi program C++.

```
12  const int MAX_DESTINATIONS = 20;
13
14  struct Edge {
15      int destination;
16      double distance;
17      int cost;
18  };
19
20  struct Node {
21      vector<Edge> neighbors;
22  };
```

Gambar 3. Inisialisasi struktur data

Inisialisasi struktur data dalam kode **Gambar 3** merupakan fondasi yang krusial untuk menggambarkan graf yang akan digunakan dalam program. Pertama-tama, nilai konstan *MAX_DESTINATIONS* ditetapkan sebagai 20, menandakan jumlah maksimum destinasi atau simpul dalam graf yang akan ditangani.

Structure Edge mencerminkan setiap sisi dalam graf. Tiap sisi memiliki atribut *destination*, yang menyimpan nomor simpul tujuan, distance yang menunjukkan jarak fisik antara simpul asal dan tujuan, serta *cost* yang mencatat biaya untuk melewati sisi tersebut. Sementara itu, *structure Node* menggambarkan simpul dalam graf. Setiap simpul memiliki vektor *neighbors* yang memuat daftar sisi yang terhubung dengan simpul tersebut. Hal ini memungkinkan setiap simpul memiliki beberapa tetangga dengan sisi-sisi yang berbeda, menggambarkan hubungan kompleks antara simpul-simpul dalam graf.

```
24  vector<Node> destinations
    (MAX_DESTINATIONS);
25  vector<double> heuristic
    (MAX_DESTINATIONS);
26  vector<double> dist
    (MAX_DESTINATIONS,
    numeric_limits<double>::max());
27  vector<int> cost(MAX_DESTINATIONS,
    numeric_limits<int>::max());
```

Gambar 4. Inisialisasi vektor dan variabel

Bagian pada **Gambar 4** merupakan inisialisasi beberapa vektor dan variabel yang akan digunakan dalam program ini. Vektor *destinations* dimulai dengan kapasitas awal yang ditentukan oleh konstruktor vektor, diatur dengan parameter *MAX_DESTINATIONS*. Hal ini memungkinkan vektor untuk menampung hingga 20 elemen dari tipe data *Node*, mewakili setiap simpul dalam graf yang akan dioperasikan oleh program. Dengan ini, *destinations* berperan sebagai wadah utama untuk menyimpan informasi tentang setiap simpul dan keterhubungannya dengan simpul lain dalam graf.

Variabel *heuristic* juga diinisialisasi sebagai vektor dengan kapasitas *MAX_DESTINATIONS*, menunjukkan bahwa vektor tersebut memuat informasi heuristik untuk setiap simpul dalam graf. Nilai heuristik ini digunakan dalam algoritma *A-Star* untuk mengestimasi jarak tersisa dari simpul saat ini ke tujuan akhir. Dengan memiliki *heuristic*, program dapat menghitung perkiraan heuristik yang diperlukan untuk membimbing pencarian rute yang optimal.

Vektor *dist* dan *cost* diinisialisasi dengan nilai maksimum yang sesuai dengan tipe data yang relevan. Variabel *dist* digunakan untuk menyimpan jarak terpendek yang telah ditemukan dari simpul awal ke setiap simpul lain dalam graf, sementara *cost* menyimpan biaya terendah yang telah dihitung untuk mencapai setiap simpul dari simpul awal. Inisialisasi ini memastikan bahwa nilai awal untuk kedua vektor tersebut sesuai dengan prinsip algoritma A-Star, yang memerlukan inisialisasi nilai jarak dan biaya sebelum memulai pencarian rute.

```
29  // Fungsi heuristik untuk A*
30  double calculateHeuristic(int
    current, int goal) {
31      double distance = heuristic
        [current];
32      double weight_distance = 0.5;
33      double combined_heuristic =
        weight_distance * distance;
34      return combined_heuristic;
35  }
```

Gambar 5. Fungsi *calculateHeuristic*

Fungsi *calculateHeuristic* berfungsi untuk menghitung nilai heuristik dari simpul saat ini (*current*) ke tujuan akhir (*goal*) dalam algoritma A-Star. Di dalam fungsi ini, langkah pertama adalah mengambil nilai heuristik dari simpul saat ini (*current*) dari vektor *heuristic* yang telah diinisialisasi sebelumnya. Nilai heuristik ini digunakan sebagai perkiraan jarak dari simpul saat ini ke tujuan akhir. Selanjutnya, faktor bobot *weight_distance* ditentukan untuk mengatur

seberapa besar pengaruh jarak dalam perhitungan heuristik. Pada implementasi ini, faktor bobot ditetapkan sebagai 0.5, yang mengindikasikan bahwa jarak memiliki pengaruh setengah dari total nilai heuristik.

Dengan mengalikan nilai jarak dengan faktor bobot, fungsi menghasilkan nilai heuristik yang merupakan gabungan dari jarak dan bobotnya. Hasil perhitungan ini kemudian dikembalikan sebagai nilai heuristik yang akan digunakan dalam algoritma A-Star untuk membimbing pencarian rute optimal. Dengan menggunakan nilai heuristik ini, algoritma dapat memperkirakan jarak yang tersisa dari simpul saat ini ke tujuan akhir dan memprioritaskan simpul yang memiliki nilai heuristik yang lebih rendah dalam pencarian rute.

```
39  void AStar(int start, int end,
    vector<int>& optimalPath, const
    vector<string>& placeNames,
    vector<int>& cost, int maxCost) {
40      int numDestinations =
        destinations.size();
41      dist.assign(numDestinations,
        numeric_limits<double>::max());
42      cost.assign(numDestinations,
        numeric_limits<int>::max());
43      cost[start] = 0;
```

Gambar 6. Fungsi *Astar*

Fungsi *AStar* bertugas mencari rute optimal dari simpul awal (*start*) ke simpul tujuan (*end*) dengan menggunakan algoritma A-Star. Jumlah destinasi atau simpul dalam graf ditentukan dengan mengambil panjang vektor *destinations*, disimpan dalam variabel *numDestinations*. Hal ini penting untuk alokasi memori dan penyesuaian ukuran vektor *dist* dan *cost*, yang akan digunakan untuk menyimpan jarak dan biaya terkini dari simpul awal ke setiap simpul lain.

Vektor akan *dist* dan *cost* diisi dengan nilai maksimum yang menandakan nilai tak terbatas, menandakan bahwa pada awalnya jarak dan biaya ke semua simpul lain belum diketahui. Namun, biaya ke simpul awal diatur sebagai 0 karena tidak ada biaya untuk mencapai simpul awal dari dirinya sendiri.

```
45      vector<int> prev(numDestinations, -1);
46      priority_queue<pair<double, int>,
        vector<pair<double, int>>,
        greater<pair<double, int>>> pq;
47
48      pq.push({calculateHeuristic(start, end),
        start});
49      dist[start] = 0;
50      cost[start] = 0;
```

Gambar 7. Vektor *prev* dan *priority queue*

Pada bagian blok funsgi *Astar*, vektor *prev* digunakan untuk melacak jalur yang diambil untuk mencapai setiap simpul selama pencarian rute. Nilai awalnya diinisialisasi sebagai -1, menandakan bahwa pada awalnya belum ada jalur yang diketahui. Selanjutnya, *priority queue* (*pq*) digunakan untuk menyimpan simpul yang akan dieksplorasi berikutnya selama pencarian rute. Setiap elemen dalam *priority queue* adalah pasangan dari nilai heuristik dari simpul saat ini ke tujuan akhir dan indeks simpul itu sendiri. Elemen-elemen dalam *priority queue* diurutkan berdasarkan nilai heuristik, sehingga simpul dengan nilai heuristik yang lebih kecil akan dieksplorasi lebih dulu.

```
52      while (!pq.empty()) {
53          int current = pq.top().second;
54          pq.pop();
55
56          if (current == end) {
57              break;
58          }
```

Gambar 8. Eksplorasi simpul

Bagian ini merupakan bagian dari perulangan utama dalam algoritma A-Star yang bertujuan untuk mencari rute optimal dari simpul awal ke simpul tujuan dalam sebuah graf.

Pada setiap iterasi, program mengambil simpul dengan nilai $f(n)$ terendah dari antrian prioritas *pq*. Nilai $f(n)$ adalah hasil penjumlahan jarak sejauh ini (*dist*) dengan nilai heuristik dari simpul tersebut ke simpul tujuan. Langkah ini dilakukan untuk memilih simpul yang memiliki potensi menjadi bagian dari rute optimal. Setelah simpul terpilih dihapus dari antrian prioritas, program memeriksa apakah simpul tersebut merupakan simpul tujuan. Jika benar, iterasi berhenti dan pencarian rute optimal dihentikan karena simpul tujuan telah tercapai dan rute optimal telah ditemukan. Jika simpul tersebut bukan simpul tujuan, program melanjutkan dengan memeriksa tetangga-tetangga dari simpul tersebut. Untuk setiap tetangga, program memeriksa apakah rute yang melalui simpul tersebut memiliki jarak dan biaya yang lebih rendah dibandingkan rute sebelumnya. Jika benar, nilai jarak dan biaya untuk tetangga tersebut diperbarui, dan simpul tersebut dimasukkan ke dalam antrian prioritas dengan nilai $f(n)$ yang sesuai.

Proses ini terus berlanjut sampai simpul tujuan tercapai atau sampai antrian prioritas kosong, menandakan bahwa tidak ada rute yang memungkinkan dari simpul awal ke simpul tujuan.

```
60          for (const Edge& neighbor :
            destinations[current].neighbors) {
61              if (cost[current] + neighbor.
                cost > maxCost) {
62                  continue;
63              }
```

Gambar 9. Pemeriksaan biaya dengan anggaran

Bagian pada **Gambar 9** merupakan proses iterasi melalui tetangga-tetangga dari simpul saat ini dalam perulangan utama algoritma A-Star. Setiap iterasi bertujuan untuk memeriksa apakah biaya total untuk mencapai tetangga tersebut melebihi batasan maksimum yang telah ditentukan (*maxCost*).

Langkah pertama adalah program mengakses tetangga saat ini dari struktur data array dan menyimpannya dalam variabel *neighbor*. Kemudian, program memeriksa apakah biaya total untuk mencapai tetangga tersebut, yang dihitung dengan menambahkan biaya saat ini (*cost[current]*) dengan biaya dari simpul saat ini ke tetangga tersebut, melebihi batasan maksimum yang telah ditetapkan.

Jika biaya total tersebut melebihi batasan maksimum, iterasi tersebut akan diabaikan dengan menggunakan perintah *continue*. Dengan demikian, program akan melanjutkan ke iterasi selanjutnya untuk mengecek tetangga berikutnya. Hal ini dilakukan untuk memastikan bahwa rute yang dipilih tidak melampaui batasan biaya yang telah ditetapkan.

```
66              double newDist = dist[current] +
                neighbor.distance;
67              int newCost = cost[current] +
                neighbor.cost;
```

Gambar 10. Variabel *newDist* dan *newCost*

Pada **Gambar 10** dilakukan inisialisasi dua variabel baru, yaitu *newDist* dan *newCost*, yang digunakan dalam algoritma A-Star. Variabel *newDist* bertujuan untuk menyimpan jarak baru dari simpul awal ke tetangga yang sedang dipertimbangkan. Jarak baru ini dihitung dengan menambahkan jarak dari simpul awal (*current*) ke tetangga tersebut (*neighbor.distance*) dengan jarak dari simpul awal ke tetangga sebelumnya yang sudah tercatat (*dist[current]*). Sementara itu, variabel *newCost* digunakan untuk menyimpan biaya baru yang diperlukan untuk mencapai tetangga tersebut. Biaya baru ini dihitung dengan menambahkan biaya dari simpul awal (*current*) ke tetangga tersebut (*neighbor.cost*) dengan biaya dari simpul awal ke tetangga sebelumnya yang sudah tercatat (*cost[current]*). Melalui langkah ini, estimasi jarak dan biaya untuk mencapai setiap tetangga diperbarui selama proses pencarian jalur optimal.

```
69              if (newDist < dist[neighbor.destination] ||
                (newDist == dist[neighbor.destination] &&
                newCost < cost[neighbor.destination])) {
70                  dist[neighbor.destination] = newDist;
71                  cost[neighbor.destination] = newCost;
72                  prev[neighbor.destination] = current;
73                  pq.push({newDist + calculateHeuristic
                    (neighbor.destination, end), neighbor.
                    destination});
74              }
```

Gambar 11. Perbarui jarak dan biaya

Bagian pada **Gambar 11** merupakan bagian dari algoritma A-Star yang bertugas untuk memperbarui estimasi jarak dan biaya untuk mencapai suatu tetangga tertentu. Pada awalnya, kondisi *if* melakukan pengecekan apakah jarak baru dari simpul awal ke tetangga saat ini lebih kecil daripada estimasi jarak sebelumnya ke tetangga tersebut. Jika benar, atau jika jarak baru sama dengan estimasi jarak sebelumnya dan biaya baru lebih kecil dari biaya sebelumnya, maka sebuah jalur baru yang lebih baik ditemukan.

Dalam hal ini, estimasi jarak dan biaya untuk mencapai tetangga tersebut diperbarui dengan nilai baru (*newDist* dan *newCost*). Selain itu, simpul sebelumnya yang harus dilalui juga diperbarui menjadi simpul saat ini (*current*). Terakhir, simpul tetangga tersebut dimasukkan ke dalam antrian prioritas (*pq*) dengan estimasi nilai total sebagai kunci. Estimasi ini dihitung dengan menambahkan jarak baru ke tetangga tersebut dengan estimasi heuristik dari tetangga tersebut ke tujuan akhir. Hal ini memastikan bahwa simpul-simpul dengan estimasi jarak terendah akan dieksplorasi terlebih dahulu dalam pencarian jalur optimal.

```
78      if (dist[end] == numeric_limits<double>::max() ||
        cost[end] == numeric_limits<int>::max()) {
79          cout << "Tidak ada rute yang ditemukan dari " <<
            placeNames[start] << " ke " << placeNames[end]
            << ".\n";
80          return;
81      }
```

Gambar 12. Pengecekan rute

Pada **Gambar 12** merupakan bagian yang digunakan untuk mengevaluasi apakah rute dari simpul awal ke simpul akhir dapat ditemukan. Evaluasi ini dilakukan dengan memeriksa apakah estimasi jarak (*dist[end]*) atau biaya (*cost[end]*) untuk mencapai simpul akhir setelah pencarian menggunakan algoritma A-Star masih bernilai maksimum. Jika salah satu dari nilai tersebut masih maksimum, hal ini menandakan bahwa tidak ada

jalur yang tersedia dari simpul awal ke simpul akhir.

Dalam kasus ini, program akan mencetak pesan bahwa rute yang dimaksud tidak ada dan menghentikan eksekusi dengan menggunakan perintah *return*. Langkah ini memastikan bahwa jika tidak ada jalur yang tersedia, program akan memberi tahu pengguna dan berhenti melanjutkan proses lebih lanjut.

```
84      optimalPath.clear();
85      int current = end;
86      while (current != start) {
87          optimalPath.push_back(current);
88          current = prev[current];
89      }
```

Gambar 13. Rekonstruksi rute

Bagian ini berfungsi untuk merekonstruksi rute optimal dari simpul awal ke simpul akhir setelah proses pencarian menggunakan algoritma A-Star. Vektor *optimalPath* dikosongkan untuk membersihkan jalur sebelumnya. Kemudian, nilai *current* diinisialisasi dengan simpul akhir. Dalam iterasi berikutnya, nilai *current* akan diperbarui dengan simpul sebelumnya dalam jalur optimal. Iterasi dilakukan sampai nilai *current* sama dengan simpul awal. Pada setiap iterasi, simpul saat ini ditambahkan ke vektor *optimalPath*. Proses ini memanfaatkan informasi yang tersimpan dalam vektor *prev*, di mana *prev[current]* adalah simpul sebelumnya dalam jalur optimal dari *current*. Dengan demikian, jalur optimal dari simpul awal ke simpul akhir direkonstruksi dan disimpan dalam vektor *optimalPath*.

```
91      optimalPath.push_back(start);
92      reverse(optimalPath.begin(), optimalPath.end());
93      cout << "\nRute optimal dari " << placeNames[start]
        << " ke " << placeNames[end] << ":\n";
94      for (int i = 0; i < optimalPath.size(); ++i) {
95          cout << placeNames[optimalPath[i]];
96          if (i != optimalPath.size() - 1) {
97              cout << " -> ";
98          }
99      }
```

Gambar 14. Cetak rute

Setelah jalur optimal direkonstruksi, simpul awal (*start*) dimasukkan ke dalam *optimalPath*. Selanjutnya, vektor *optimalPath* dibalik menggunakan fungsi *reverse*, sehingga urutan elemen dibalik sehingga elemen terakhir menjadi elemen pertama dan sebaliknya. Tindakan ini diperlukan karena jalur optimal telah direkonstruksi dari simpul akhir ke simpul awal, sedangkan yang diinginkan adalah tampilan jalur dari simpul awal ke simpul akhir.

Setelah pembalikan *optimalPath*, program mencetak pesan yang menggambarkan rute optimal dari tempat awal ke tempat akhir dengan menggunakan nama tempat yang disimpan dalam vektor *placeNames*. Dalam iterasi, setiap elemen *optimalPath* diprint dengan menggunakan indeksnya sebagai referensi untuk mengakses vektor *placeNames*. Setiap nama tempat diprint, dan jika bukan elemen terakhir, tanda panah juga dicetak untuk memisahkan setiap tempat dalam jalur.

```
100     cout <<
        "\n----------------------------------------
        --------------\n";
101     cout << "Estimasi Jarak total: " << dist
        [end] << " Km\n";
102     cout << "Estimasi Biaya total: " << cost
        [end] << " IDR\n";
```

Gambar 15. Estimasi jarak dan biaya

Setelah jalur optimal berhasil direkonstruksi, program kemudian menampilkan estimasi total jarak dari simpul awal ke simpul akhir. Estimasi ini dihitung dari nilai yang tersimpan dalam variabel *dist[end]*, yang merepresentasikan total jarak. Selanjutnya, program mencetak estimasi total biaya dari simpul awal ke simpul akhir. Perhitungan biaya ini diambil dari nilai yang tersimpan dalam variabel *cost[end]*, yang mewakili biaya total. Estimasi biaya ini dinyatakan dalam mata uang Rupiah (IDR).

Pada **Gambar 16** hingga **Gambar 18** adalah konfigurasi awal dari graf yang merepresentasikan relasi antara sejumlah lokasi atau titik di Kota Palangka Raya. Setiap elemen dalam struktur data *destinations* mewakili satu titik atau lokasi di kota tersebut. Tiap titik memiliki tetangga-tetangga yang terhubung secara langsung dengannya. Setiap tetangga direpresentasikan oleh struktur data *Edge*, yang mencakup informasi tentang tujuan tetangga, jarak ke tetangga, dan biaya yang terkait dengan perjalanan ke tetangga tersebut. Contohnya, untuk titik awal A (Hotel Auralia), keterhubungannya dengan tetangga-tetangganya diatur sebagai berikut: Tetangga pertama memiliki nomor indeks 1, jarak sejauh 3.6 kilometer, dengan biaya perjalanan sebesar 50.000 Rupiah. Tetangga kedua memiliki nomor indeks 2, dengan jarak 4.4 kilometer, dan biaya perjalanan yang sama, yakni 50.000 Rupiah. Sementara itu, tetangga ketiga memiliki nomor indeks 4, terletak pada jarak sejauh 5.6 kilometer dari titik awal, dengan biaya perjalanan yang dikecualikan, atau sebesar 0 Rupiah.

Proses ini diulangi untuk setiap titik dalam graf. Setiap elemen dalam struktur data *destinations* diinisialisasi dengan vektor *neighbors*, yang berisi daftar tetangga-tetangga dari titik yang bersangkutan.

Dengan konfigurasi awal ini, graf menyimpan informasi tentang keterhubungan setiap titik dengan tetangga-tetangganya, serta jarak dan biaya yang terkait. Hal ini menjadi dasar untuk melakukan pencarian rute optimal menggunakan algoritma A-Star.

```
115     //HUBUNGAN SIMPUL A (HOTEL) DENGAN SIMPUL SEKITAR
116     destinations[0].neighbors = {{1, 3.6, 50'000},
117                                  {2, 4.4, 50'000},
118                                  {4, 5.6, 0}};
119
120     // B (SAUNG RINDU TELAGA) DENGAN SIMPUL SEKITAR
121     destinations[1].neighbors = {{0, 2.5, 0},
122                                  {2, 1.4, 50'000},
123                                  {3, 6.2, 50'000}};
124
125   // C (RM. PELANGI 3) DENGAN SIMPUL SEKITAR
126     destinations[2].neighbors = {{0, 1.1, 0},
127                                  {1, 1.5, 50'000},
128                                  {3, 6.4, 0},
129                                  {4, 2.4, 0},
130                                  {8, 5.9, 0}};
131
132     // D (WISATA FLAMBOYAN BAWAH) DENGAN SIMPUL SEKITAR
133     destinations[3].neighbors = {{1, 5.5, 50'000},
134                                  {2, 5.9, 50'000},
135                                  {8, 1.6, 0}};
136
137      // E (BUNDARAN BURUNG) DENGAN SIMPUL SEKITAR
138     destinations[4].neighbors = {{0, 2.2, 0},
139                                  {2, 1.3, 50'000},
140                                  {5, 6.3, 5'000},
141                                  {7, 4.8, 0},
142                                  {18, 4.7, 60'000}};
143
144     // F (DERMAGA KERENG BANGKIRAI) DENGAN SIMPUL SEKITAR
145     destinations[5].neighbors = {{4, 6.4, 0},
146                                  {6, 3.9, 20'000},
147                                  {18, 3.3, 60'000}};
```

Gambar 16. Inisialisasi graf

```
149     destinations[6].neighbors = {{5, 3.9, 5'000},      // G (PESONA ALAM LESTARI) DENGAN SIMPUL SEKITAR
150                                  {18, 4.4, 60'000}};
151
152     destinations[7].neighbors = {{4, 4.8, 0},             // H (BUNDARAN BESAR) DENGAN SIMPUL SEKITAR
153                                  {8, 1.5, 0},
154                                  {9, 2.9, 0},
155                                  {12, 2.7, 2'500},
156                                  {16, 4.1, 50'000},
157                                  {19, 3.7, 120'000}};
158
159     destinations[8].neighbors = {{7, 0.750, 0},     // I (TUGU SOEKARNO) DENGAN SIMPUL SEKITAR
160                                  {3, 1.6, 0},
161                                  {2, 5.9, 50'000},
162                                  {9, 1.6, 0}};
163
164     destinations[9].neighbors = {{7, 1.3, 0},       // J (JEMBATAN KAHAYAN) DENGAN SIMPUL SEKITAR
165                                  {8, 1.6, 0},
166                                  {10, 1.1, 50'000}};
167
168     destinations[10].neighbors = {{9, 1.1, 0},
169                                   {11, 1.1, 70'000}};    // K (KAMPUNG LAUK) DENGAN SIMPUL SEKITAR
170
171     destinations[11].neighbors = {{10, 1.1, 50'000}};    // L (RM. KUM KUM & TAMAN WISATA ) DENGAN SIMPUL SEKITAR
172
173     destinations[12].neighbors = {{7, 2.4, 0},
174                                   {13, 4.6, 35'000},
175                                   {19, 5.5, 120'000}};    // M (MUSEUM BALANGA) DENGAN SIMPUL SEKITAR
```

Gambar 17. Inisialisasi graf

```
177     destinations[13].neighbors = {{12, 4.6, 2'500},
178                                   {14, 7.1, 35'000},
179                                   {19, 8.8, 120'000}};    // N (KALAWA WATERPARK) DENGAN SIMPUL SEKITAR
180
181     destinations[14].neighbors = {{13, 7.1, 35'000},
182                                   {15, 8.1, 100'000},
183                                   {17, 9.6, 50'000}};    // O (WISATA AGRO WISATA) DENGAN SIMPUL SEKITAR
184
185     destinations[15].neighbors = {{14, 8.1, 35'000},
186                                   {16, 5.2, 50'000},
187                                   {18, 4.7, 60'000},
188                                   {17, 5.5, 50'000}};    // P (SAKUYAN SIDE) DENGAN SIMPUL SEKITAR
189
190     destinations[16].neighbors = {{15, 5.8, 100'000},
191                                   {17, 2.7, 50'000},
192                                   {19, 2.5, 120'000},
193                                   {18, 7.7, 60'000},
194                                   {7, 3.2, 0}};    // Q (RM. AYAM PENYET) DENGAN SIMPUL SEKITAR
195
196     destinations[17].neighbors = {{19, 1.0, 120'000},
197                                   {16, 3.0, 50'000},
198                                   {15, 4.8, 100'000},
199                                   {14, 8.9, 35'000}};    // R (DANGO PLAYING GROUND) DENGAN SIMPUL SEKITAR
200
201     destinations[18].neighbors = {{4, 4.7, 0},
202                                   {5, 3.3, 5'000},
203                                   {6, 4.4, 20'000},
204                                   {15, 4.7, 100'000},
205                                   {16, 7.7, 50'000}};    // S (WISATA ATV TANDUHAN) DENGAN SIMPUL SEKITAR
206
207     destinations[19].neighbors = {{7, 3.7, 0},
208                                   {12, 5.6, 2'500},
209                                   {13, 8.4, 35'000},
210                                   {16, 2.5, 50'000},
211                                   {17, 1.0, 50'000}};    // T (FOOD CONTAINER) DENGAN SIMPUL SEKITAR
```

Gambar 18. Inisialisasi graf

```
213     vector<string> placeNames = {"Hotel Auralia", "Saung Rindu
214                                  Telaga", "RM Pelangi 3", "Wisata
                                     Flamboyan Bawah", "Bundaran
                                     Burung","Dermaga Kereng
                                     Bangkirai", "Pesona Alam
                                     Lestari", "Bundaran Besar","Tugu
                                     Soekarno", "Jembatan Kahayan",
                                     "Kampung Lauk", "Kum Kum Tempat
                                     Wisata dan Makan","Museum
                                     Balanga", "Kalawa Waterpark",
                                     "Wisata Agro Larosa", "Sakuyan
                                     Side", "RM. Ayam Penyet", "Dango
                                     Playing Park", "Wisata ATV
                                     Tanduhan", "Food Container Area"};
```

Gambar 19. Inisialisasi vektor *placeNames*

Inisialisasi dari vektor *placeNames* merupakan proses pengaturan penyimpanan nama-nama tempat atau lokasi di Kota Palangka Raya. Tiap entitas dalam vektor ini adalah representasi *string* yang memuat nama sebuah tempat. Rentang tempat yang dimasukkan mencakup beragam destinasi wisata, tempat makan, *landmark*, serta objek menarik lainnya di Kota Palangka Raya.

Setiap entitas *string* dalam vektor ini diinisialisasi dengan nama tempat yang secara tepat mencerminkan lokasi fisiknya di Kota Palangka Raya. Nama tempat tersebut mencakup informasi terkait jenis tempat tersebut dan sering kali menyoroti ciri khas atau fitur utamanya, seperti nama hotel, restoran, atau obyek wisata tertentu.

Vektor *placeNames* berfungsi sebagai referensi dalam program untuk memberikan informasi tentang nama tempat yang terasosiasi dengan nomor indeks tempat dalam graf. Dengan penggunaan vektor ini, program dapat dengan akurat menampilkan hasil rute optimal dengan mencantumkan nama tempat yang sesuai,

memudahkan pemahaman pengguna mengenai rute yang dihasilkan.

```
216     cout << "\nDaftar Tempat:\n";
217     for (int i = 0; i < placeNames.size
        (); ++i) {
218         cout << i << ". " << placeNames
            [i] << endl;
219     }
```

Gambar 20. Cetak daftar tempat

Bagian ini memanfaatkan sebuah perulangan *for* guna menampilkan daftar tempat yang tersedia dalam vektor *placeNames*. Iterasi dimulai dari indeks 0 hingga mencapai ujung indeks pada vektor *placeNames*, yang merepresentasikan jumlah elemen yang ada dalam vektor tersebut.

Setiap elemen dalam vektor *placeNames* diakses menggunakan indeksnya masing-masing. Nomor indeks yang bersangkutan, diikuti dengan nama tempat yang sesuai, diprint ke layar menggunakan instruksi *cout*. Penyajian informasi dilakukan dalam format yang terstruktur, di mana nomor indeks dan nama tempat dipisahkan dengan tanda titik dan spasi. Penggunaan penutup baris (*endl*) memastikan bahwa setiap entri baru dalam daftar diprint pada baris yang terpisah, meningkatkan keterbacaan dan pemahaman daftar tempat oleh pengguna.

Bagian pada **Gambat 21** berperan dalam meminta input dari pengguna, yang terdiri dari nomor tempat awal, nomor tempat akhir, dan batasan biaya. Variabel *start*, *end*, dan *maxCost* digunakan untuk menampung input tersebut. Pengguna diminta untuk memasukkan nomor tempat awal dan akhir yang harus berada dalam rentang antara 0 dan ukuran vektor *placeNames* dikurangi 1, yang merepresentasikan jumlah tempat yang tersedia dalam daftar. Selain itu, pengguna diminta untuk memasukkan batasan biaya dalam mata uang Rupiah Indonesia (IDR). Setelah input diberikan oleh pengguna, dilakukan pengecekan untuk memastikan validitas nomor tempat awal dan akhir yang dimasukkan, yaitu harus berada dalam rentang 0 hingga ukuran vektor *placeNames* dikurangi 1. Jika salah satu dari input tersebut berada di luar rentang tersebut, program akan memberikan pesan bahwa nomor tempat tidak valid dan eksekusi program akan dihentikan.

Dalam **Gambar 22**, sebuah variabel *start_time* bertipe *clock_t* digunakan untuk merekam waktu awal eksekusi program dengan memanggil fungsi *clock()*. Selanjutnya, dibuat sebuah vektor *optimalPath* yang akan menyimpan jalur optimal yang akan dihitung oleh algoritma A-Star. Fungsi *AStar()* kemudian dipanggil dengan parameter nomor tempat awal (*start*), nomor tempat akhir (*end*), vektor *optimalPath*, vektor *placeNames* yang berisi nama tempat, vektor *cost* yang menyimpan biaya, dan *maxCost* yang merupakan batasan biaya yang telah dimasukkan oleh pengguna sebelumnya. Setelah eksekusi algoritma selesai, waktu akhir eksekusi direkam dengan memanggil fungsi *clock()* dan disimpan dalam variabel *end_time*. Selanjutnya, waktu eksekusi dihitung dengan mengurangi waktu akhir eksekusi dengan waktu awal eksekusi, kemudian hasilnya dibagi dengan *CLOCKS_PER_SEC* untuk mendapatkan waktu eksekusi dalam satuan detik, yang disimpan dalam variabel *execution_time*.

Pada **Gambar 23** mengimplementasikan sebuah perulangan menggunakan struktur kontrol *do-while*. Perulangan ini dirancang untuk memungkinkan pengguna untuk melakukan pencarian rute lagi atau keluar dari program. Kemudian, perulangan akan berlanjut selama pengguna memilih pilihan lanjut (*y*). Jika pengguna memilih untuk tidak mencari rute lagi (*n*), perulangan akan berakhir dan program akan menampilkan pesan ucapan terima kasih.

```
221     int start, end, maxCost;
222     cout << "\nMasukkan nomor tempat awal (0-" << placeNames.size() - 1 << "): ";
223     cin >> start;
224     cout << "Masukkan nomor tempat akhir (0-" << placeNames.size() - 1 << "): ";
225     cin >> end;
226     cout << "Masukkan batasan biaya (IDR): ";
227     cin >> maxCost;
228
229     if (start < 0 || start >= placeNames.size() || end < 0 || end >= placeNames.
        size()) {
230         cout << "Nomor tempat tidak valid. Program berhenti.\n";
231         return 1;
232     }
```

Gambar 21. Input Pengguna

```
234         clock_t start_time = clock();
235         vector<int> optimalPath;
236         AStar(start, end, optimalPath,
            placeNames, cost, maxCost);
237         clock_t end_time = clock();
238         double execution_time =
            static_cast<double>(end_time -
            start_time) / CLOCKS_PER_SEC;
```

Gambar 22. Total eksekusi program

```
245         cout << "\nIngin mencari rute lagi? (y/n): ";
246         cin >> repeat;
247         cin.ignore(numeric_limits<streamsize>::max(),
            '\n');
248     } while (repeat == 'y' || repeat == 'Y');
249
250     cout << "Terima kasih telah menggunakan program ini.
        Selamat berlibur!\n";
251
252     return 0;
253 }
```

Gambar 23. Akhir program

Perhitungan Jarak dan Biaya dimulai dengan mengambil dua titik sebagai titik awal dan titik akhir. Dua titik ini dihubungkan oleh tiga rute dengan jarak tempuh dan bobot biaya yang berbeda. Contohnya, titik awal diambil dari Bundaran Besar (H) dan titik akhir adalah Sakuyan Side (P). Batasan biaya yang dikeluarkan adalah 150.000 IDR. Informasi terkait keterhubungan simpul-simpul tersebut dapat dilihat pada **Gambar 1** untuk graf dan **Tabel II** untuk daftar biaya.

Tabel III. Kalkulasi Simpul Tetangga H

Simpul	Tetangga	Kalkulasi
H	M	$h(M) = (0{,}5 * 2{,}7) = \mathbf{1,35}$ $f(M) = 2{,}7 + 1{,}35 = \mathbf{4,05}$ $p(M) = 0 + 2500 = \mathbf{2500}$
	Q	$h(Q) = (0{,}5 * 4{,}1) = \mathbf{2,05}$ $f(Q) = 4{,}1 + 2{,}05 = \mathbf{6,15}$ $p(Q) = 0 + 50000 = \mathbf{50000}$
	T	$h(T) = (0{,}5 * 3{,}7) = \mathbf{1,85}$ $f(T) = 3{,}7 + 1{,}85 = \mathbf{5,55}$ $p(T) = 0 + 120000 = \mathbf{120000}$

Pada **Tabel III**, eksplorasi dimulai dengan melakukan perhitungan pada semua simpul bertetanggaan dengan titik awal. Dengan demikian, maka diperoleh nilai heuristik untuk setiap simpul tetangga.

Eksplorasi dilakukan kembali dengan kalkulasi simpul tetangga dari simpul tetangga pada **Tabel III**. Nilai $h(n)$ merupakan perhitungan untuk mendapatkan nilai heuristik dari simpul yang berkaitan. Adapaun $f(n)$ adalah nilai total perkiraan pada lintasan yang telah dilewati dan $p(n)$ adalah total biaya yang telah dikeluarkan. Setelah itu hasil setiap kalkulasi akan dibandingkan dan dipilih dengan bobot jarak terpendek dan biaya tidak melebihi anggaran.

Tabel IV. Kalkulasi Simpul Tetangga M

Simpul	Tetangga	Kalkulasi
M	N	$h(N) = (0{,}5 * 4{,}6) = \mathbf{2,3}$ $f(N) = 4{,}6 + 2{,}3 = \mathbf{6,9}$ $p(N) = 2500 + 35000 = \mathbf{37500}$
	T	$h(T) = (0{,}5 * 5{,}5) = \mathbf{2,75}$ $f(T) = 5{,}5 + 2{,}75 = \mathbf{8,25}$ $p(T) = 2500 + 120000 = \mathbf{122500}$

Pada kenyataannya, eksplorasi dan perhitungan simpul-simpul dalam program dilakukan secara bersamaan dengan menggunakan mesin. Kendati demikian, dalam konteks eksplorasi manual, perhitungan dilakukan secara berurutan.

Eksplorasi pada **Tabel IV** dimulai kembali dari simpul M. Eksplorasi simpul dimulai dengan melakukan perhitungan dengan simpul tetangga. Eksplorasi Proses ini terus diulang selama bobot biaya tidak melebihi anggaran yang sudah ditentukan.

Pada **Tabel V** eksplorasi simpul dimulai dari simpul T. Selama perhitungan, jarak yang didapatkan dievaluasi dengan bobot biaya. Hal ini digunakan agar rute yang dihasilkan memiliki bobot biaya yang sesuai dengan anggaran yang telah ditentukan.

Eksplorasi kembali dilakukan dari simpul Q. Perhitungan kembali dilakukan dengan simpul yang bertetanggaan dengan simpul Q. Dalam perhitungan ini, simpul Q terhubung langsung dengan simpul tujuan, yaitu Sakuyan Side (P). Dengan ini, maka eksplorasi dan perhitungan manual selesai dengan biaya yang kurang dari jumlah anggaran yang telah ditentukan. Rute yang dihasilkan dengan perhitungan manual ini adalah Bundaran Besar (H) melewati RM. Ayam Penyet

(Q) dan sampai pada tujuan akhir yaitu, Sakutan Side (P)

Tabel V. Kalkulasi Simpul Tetangga T

Simpul	Tetangga	Kalkulasi
T	M	$h(M) = (0{,}5 * 5{,}6) = \mathbf{2,8}$ $f(M) = 5{,}6 + 2{,}8 = \mathbf{8,4}$ $p(M)$ $= 120000 + 2500$ $= \mathbf{122500}$
	R	$h(R) = (0{,}5 * 1) = \mathbf{0,5}$ $f(R) = 3 + 0{,}5 = \mathbf{3,5}$ $p(R)$ $= 120000 + 50000$ $= \mathbf{170000}$
	Q	$h(Q) = 0{,}5 * 5{,}8 = \mathbf{2,9}$ $f(Q) = 5{,}8 + 2{,}9 = \mathbf{8,7}$ $p(Q)$ $= 50000 + 100000$ $= \mathbf{150000}$

Tabel VI. Kalkulasi Simpul Tetangga Q

Simpul	Tetangga	Kalkulasi
Q	T	$h(T) = (0{,}5 * 2{,}5) = \mathbf{1,25}$ $f(T) = 2{,}5 + 1{,}25 = \mathbf{3,75}$ $p(T)$ $= 50000 + 120000$ $= \mathbf{170000}$
	R	$h(R) = (0{,}5 * 3) = \mathbf{1,5}$ $f(R) = 3 + 1{,}5 = \mathbf{4,5}$ $p(R) = 50000 + 50000 = \mathbf{100000}$
	P	$h(P) = 0{,}5 * 5{,}8 = \mathbf{2,9}$ $f(P) = 5{,}8 + 2{,}9 = \mathbf{8,7}$ $p(P)$ $= 50000 + 100000$ $= \mathbf{150000}$

Untuk mengetahui kebenaran dari perhitungan manual yang telah dilakukan, maka dilakukan perbandingan dengan program yang sudah dibuat. **Gambar 24** merupakan daftar tempat yang telah diinisialisasikan dalam program. Pada **Gambar 25** titik awal yaitu, Bundaran Besar (H) direpresentasikan dengan nomor 7 dan titik akhir Sakuyan Side (P) direpresentasikan dengan nomor 15.

```
Daftar Tempat:
0. Hotel Auralia
1. Saung Rindu Telaga
2. RM Pelangi 3
3. Wisata Flamboyan Bawah
4. Bundaran Burung
5. Dermaga Kereng Bangkirai
6. Pesona Alam Lestari
7. Bundaran Besar
8. Tugu Soekarno
9. Jembatan Kahayan
10. Kampung Lauk
11. Kum Kum Tempat Wisata dan Makan
12. Museum Balanga
13. Kalawa Waterpark
14. Wisata Agro Larosa
15. Sakuyan Side
16. RM. Ayam Penyet
17. Dango Playing Park
18. Wisata ATV Tanduhan
19. Food Container Area
```

Gambar 24. Daftar tempat

```
Masukkan nomor tempat awal (0-19): 7
Masukkan nomor tempat akhir (0-19): 15
Masukkan batasan biaya (IDR): 150000

Rute optimal dari Bundaran Besar ke Sakuyan Side:
Bundaran Besar -> RM. Ayam Penyet -> Sakuyan Side
--------------------------------------------------
Estimasi Jarak total: 9.9 Km
Estimasi Biaya total: 150000 IDR
Waktu eksekusi: 0.002 detik
--------------------------------------------------
```

Gambar 25. Keluaran Algoritma *A-Star*

Pada **Gambar 25** diambil titik awal adalah Bundaran Besar (H) dan titik tujuan adalah Sakuyan Side (P). Pengambilan titik tersebut disesuaikan dengan pengambilan titik pada perhitungan manual. Selain itu, masukan batasan biaya juga disesuaikan dengan masukan batasan biaya pada perhitungan manual. Hal ini bertujuan agar keluaran rute dari program dapat dibandingkan dengan rute hasil dari perhitungan manual.

Keluaran pada **Gambar 25** menunjukkan rute optimal dari Bundaran Besar (H) menuju Sakuyan Side (P) adalah dengan melewati rute Bundaran Besar, RM. Ayam Penyet dan Sakuyan Side. Selain itu, program juga menampilkan besaran biaya total jika mengunjungi tempat yang dilewati.

IV. KESIMPULAN

Hasil penelitian menunjukan, bahwa A-Star dapat digunakan untuk menemukan rute optimal dari titik awal ke titik akhir dalam graf yang merepresentasikan keterhubungan antara lokasi-

lokasi wisata di Kota Palangka Raya. Pengguna dapat memasukkan lokasi awal dan akhir, serta batasan biaya yang mereka inginkan. Kemudian, A-Star akan mencari jalur terpendek dari titik awal ke titik akhir dengan mempertimbangkan biaya perjalanan. Dari pengujian yang telah dilakukan dengan perhitungan manual, bahwa algoritma A-Star mampu diimplementasikan untuk mencari rute terbaik dengan acuan biaya yang sudah ditentukan. Penerapan algoritma A-Star pada program tersebut memungkinkan pengguna untuk menemukan rute wisata yang optimal dengan mempertimbangkan batasan biaya sebagai parameter untuk eksplorasi simpul.

Saran untuk penelitian selanjutnya untuk dapat mengimplementasikan algoritma dalam sebuah platform website ataupun aplikasi. Selain itu juga, tambahkan rute alternatif dan kondisi khusus sebagai bahan pertimbangan untuk pencarian rute.

DAFTAR PUSTAKA

[1] Galadi, "Studi Kolaboratif Dalam Pembangunan Pariwisata di Kawasan Dermaga Kereng Bangkirai Kota Palangka Raya." Jurnal Ilmu Sosial, Politik, dan Pemerintahan, Universitas Palangka Raya, vol 8, no 1, hlm. 70-92, 2019.

[2] I. Diana dan B. Herdiana, "Analisa Penggunaan Nilai Bobot Heuristik yang Berbeda pada Algoritma Weighted A*," Telekontran: Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan, vol. 9, no. 1, hlm. 82–93, Okt. 2021.

[3] V. Aditiya dan B. Herdiana, "Analisis Perbandingan Algoritma Breadth First Search (BFS) dan Algoritma A*," Telekontran: Jurnal Ilmiah Telekomunikasi, Kendali dan Elektronika Terapan, Universitas Komputer Indonesia, vol. 9, no. 2, hlm.139-153, Okt. 31, 2021.

[4] M. Mayadi dan R. Azhar, "Perbandingan Perhitungan Manual dengan Algoritma A Star dalam Pencarian Jalur Terpendek untuk Pengiriman Pesanan Dodol Khas Lombok," Jurnal Informatika dan Rekayasa Elektronik, vol. 2, no. 2, hlm. 27–34, Des. 2019.

[5] Y. H. Nuryoso, P. Pradjoko, dan L. Lelah, "Implementasi Algoritma A-Star Untuk Mencari Rute Terpendek Angkutan Umum Kota (Studi Kasus Pada Rute Angkutan Umum Kota di Kota Sukabumi)," Jurnal Sarjana Teknik Informatika, vol. 8, no. 1, hlm. 21–35, Feb. 2020.

[6] S. Purnama, D. A. Megawaty, dan Y. Fernando, "Penerapan Algoritma A Star Untuk Penentuan Jarak Terdekat Wisata Kuliner di Kota Bandarlampung," Jurnal Teknoinfo, vol. 12, no. 1, hlm. 28–32, Jan. 2018.

[7] M. Hutabalian, S. Sunanto, dan J. A. Amien, "Sistem Informasi Geografis Pemetaan Tempat Pembuangan Sampah Sementara di Kota Pekanbaru Dengan Mencari Rute Terdekat Menggunakan Algoritma A Star (A*)," Jurnal CoSciTech (Computer Science and Information Technology), vol. 2, no. 2, hlm. 98–107, Des. 2021.

[8] T. Maryana, R. Sudiyarno, dan K. Kusrini, "Comparison Analysis of Best First Search Algorithm with A * (star) in determining the closest route in the district Sleman," CCIT (Creative Communication and Innovative Technology) Journal, vol. 13, no. 1, hlm. 44–56, Feb. 2020.

[9] V. Budiman, H. Agung, dan Y. S. H. Leksmono, "Aplikasi Berbasis Android untuk Mencari Lokasi Puskesmas Terdekat dengan Algoritma A-Star di Provinsi DKI Jakarta," Just IT : Jurnal Sistem Informasi, Teknologi Informasi dan Komputer, vol. 9, no. 1, hlm. 39–48, Okt. 2018

[10] I. B. G. W. A. Dalem, "Penerapan Algoritma A* (Star) Menggunakan Graph untuk Menghitung Jarak Terpendek," Jurnal Resistor, vol. 1, no. 1, hlm. 41–47, Apr. 2018

[11] N. Padila, B. Basri, dan C. R. Sari, "Sistem Informasi Geografis dengan Algoritma A-Star Untuk Menentukan Jalur Terdekat," Journal Peqguruang: Conference Series, vol. 5, no. 1, hlm. 370–375, Jan. 2024.

[12] G. Mutaqin, J. N. Fadilah, dan F. Nugroho, "Implementasi Metode Path Finding dengan Penerapan Algoritma A-Star untuk Mencari Jalur Terpendek pada Game...," UIN Walisongo Semarang, Jun. 15, 2021.

[13] S. Susilawati, "Penerapan Metode A*Star Pada Pencarian Rute Tercepat Menuju Destinasi Wisata Cagar Budaya Menes Pandeglang," Universitas Hamzanwadi, Des. 2020.

[14] S. Andryana, A. Andrianingsih, dan N. Farhan, "Penerapan Algoritma A-star dan Brute Force pada Aplikasi Jakvel (Jakarta Travel) Berbasis Android," STMIK Budi Darma, Jul. 31, 2021.

[15] B. T. D. Irianto, S. Andryana, dan A. Gunaryati, "Penerapan Algoritma A-Star dalam Mencari Jalur Tercepat dan Pergerakan NonPlayer Character Pada Game Petualangan Labirin Tech-Edu," JURNAL MEDIA INFORMATIKA BUDIDARMA, vol. 5, no. 3, hlm. 953–962, Jul. 2021

[16] R. Rizky, T. Hidayat, A. G. Nugroho, dan Z. Hakim "Pencarian Jalur Terdekat dengan Metode A*(Star) Studi Kasus Serang Labuan Provinsi Banten," Prosiding Seminar Nasional Rekayasa Teknologi Informasi | SNARTISI, vol. 1, Nov. 2018.