JETBUS Journal of Education Transportation and Business
E-ISSN: 3062-8121 P-ISSN: 3062-813X
Vol. 1 No. 2 Desember 2024

Implementasi Algoritma Djikstra Dalam Pencarian Rute Terpendek Antar Fakultas di Universitas Negeri Medan

Albert Ramadhan Manik[1] Jogi Purba[2] Muhammad Budi Akbar[3] Putri Harliana[4]
Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Medan, Kota Medan, Provinsi Sumatera Utara, Indonesia[1,2,3,4]
Email: albertramadan4@gmail.com[1] jogipurba03@gmail.com[2] mbudiakbar12@gmail.com[3] harliana@unimed.ac.id[4]

Abstrak

Seiring dengan perkembangan teknologi dan kebutuhan manusia yang semakin dinamis, penentuan jalur yang efektif dan efisien menjadi salah satu fokus utama dalam berbagai bidang, seperti transportasi, logistik, dan telekomunikasi. Salah satu tantangan utama dalam aktivitas sehari-hari adalah menemukan rute terpendek yang dapat meminimalkan penggunaan sumber daya seperti bahan bakar, waktu, dan tenaga. Dalam konteks ini, algoritma Dijkstra menjadi salah satu solusi yang dapat diterapkan. Algoritma Dijkstra bekerja dengan memanfaatkan struktur graph yang berarah dan berbobot, di mana jarak antar titik diwakili oleh bobot pada setiap sisi. Dengan cara ini, algoritma Dijkstra mampu menentukan jalur dengan biaya atau jarak paling minimum antara dua titik. Selain itu, algoritma ini juga memungkinkan perhitungan total biaya dari jalur terpendek yang telah ditentukan. Penelitian ini bertujuan untuk menerapkan algoritma Dijkstra dalam proses penentuan rute terpendek pada jaringan tertentu, dengan harapan dapat meningkatkan efisiensi operasional. Hasil dari penelitian ini diharapkan dapat diterapkan dalam skenario dunia nyata, seperti penentuan rute transportasi antar kota atau penentuan jalur dalam jaringan komunikasi.

Kata Kunci: Algoritma Dijkstra, Rute Terpendek, Graph Berbobot, Efisiensi Jalur, Penentuan Rute.

PENDAHULUAN

Teori graf mulai dikenal pada saat seorang matematikawan berkebangsaan swiss,bernama Leonhard Euler,berhasil mengungkapkan misteri jembatan konigsberg pada tahun 1736. Dikota konigsberg (sekarang bernama kallingrad,di Uni Sovuet)mengalir sebuah sungai bernama pregel .Ditengah sungai tersebut terdapat dua buah pulau.Dari kedua pulai tersebut terdapat jembatan yang menghubungi ketepian sungau dan diantara kedua pulau. (Ardiansyah Dkk, 2020). Salah satu alasan perkembangan teori graf yang begitu pesat adalah aplikasinya yang sangat luas dalam kehidupan seharihari dalam berbagai bidang ilmu. Keunikan teori graf adalah kesederhanaan pokok bahasan yang dipelajarinya, karena dapat disajikan sebagai titik (vertex) dan sisi (edge). Teori graf dapat menyelesaikan permasalahan yang terdapat dalam dunia Pendidikan, seperti mengatur penjadwalan. Penjadwalan adalah proses yang sangat penting dalam pengambilan sebuah keputusan dalam sekelompok kegiatan atau pekerjaan. Penjadwalan adalah fungsi pengambilan keputusan yang berkaitan dengan mendefinisikan proses yang dijadwalkan (Yakin, 2016 dan Sagala, 2018 dalam Sa'adah dkk, 2023). Seiring dengan pesatnya perkembangan teknologi dan perubahan zaman, aktivitas manusia menjadi semakin dinamis dan kompleks. Kebutuhan akan metode yang efektif dan efisien dalam menjalankan berbagai aktivitas semakin meningkat. Salah satu bidang yang membutuhkan perhatian khusus adalah penentuan jalur optimal untuk mobilitas manusia. Penentuan jalur terpendek menjadi solusi yang signifikan untuk menghemat sumber daya, seperti bahan bakar, waktu, dan tenaga. Metode ini sangat diperlukan, terutama di lingkungan yang memiliki mobilitas tinggi, seperti institusi pendidikan, area perkotaan, maupun sistem

transportasi. Algoritma Dijkstra merupakan salah satu metode yang dapat diandalkan dalam menentukan jalur terpendek. Algoritma ini menggunakan pendekatan graf berarah dan berbobot, di mana jarak antar simpul (node) direpresentasikan oleh bobot pada setiap sisi (edge). Tujuan dari algoritma ini adalah menemukan jalur dengan biaya paling minimum di antara dua titik. Selain itu, algoritma Dijkstra juga mampu menghitung total biaya atau cost dari jalur terpendek yang telah ditemukan. Karakteristik utama dari algoritma Dijkstra adalah pendekatannya yang menggunakan metode greedy, di mana solusi terbaik di setiap langkah diambil untuk menghasilkan jalur terpendek dari satu node ke node lainnya (Arga,dkk,2021).

Algoritma Dijkstra juga memiliki kemampuan untuk menyelesaikan berbagai kasus jalur terpendek, seperti pencarian jalur terpendek antara dua simpul, jalur terpendek antara semua pasangan simpul, jalur terpendek dari satu simpul ke seluruh simpul lain, serta pencarian jalur terpendek yang melalui beberapa simpul tertentu. Oleh karena itu, algoritma ini sangat sesuai untuk diterapkan dalam berbagai skenario dunia nyata, termasuk pada lingkungan kampus untuk mempermudah mobilitas antar bangunan. Pencarian rute terpendek telah diterapkan diberbagai bidang untuk mengoptimasi kinerja suatu sistem baik untuk meminimalkan biaya ataupun mempercepat jalurnya suatu proses. Salah satu aplikasi pencarian rute terpendek yang paling menarik untuk dibahas adalah masalah transportasi. Ada beberapa metode untuk pencarian rute terpendek yaitu algoritma A*, algoritma Djikstra, algoritma Bellman-Ford, algoritma Floyd-Warshall, dan sebagainya. Ada beberapa metode untuk pencarian rute terpendek yaitu algoritma A*, Algoritma Djikstra, algoritma Bellman-Ford, algoritma Floyd-Warshall, dan sebagainya. Sistem routing pertama kali diperkenalkan tahun 2004 oleh tim peneliti yang terdiri dari Ryujiro Fujita, Hiroto Inoue, Naohiko Ichihara, Takehiko Shioda, bertujuan untuk mengatasi kepadatan lalu lintas yang terjadi di beberapa kota besar di Jepang [1]. Sistem yang diperkenalkan oleh para peneliti ini menggunakan algoritma Djikstra pada pencarian rute. Namun seiring dengan berkembangnya ilmu pengetahuan, metode Djikstra mempunyai kekurangan dalam waktu pencarian yang kurang efektif dan dinilai lamban (Luthfita dkk,2022). Penelitian ini akan fokus pada penerapan algoritma Dijkstra dalam menentukan rute terpendek antar fakultas di Universitas Negeri Medan (UNIMED). Dengan lingkungan kampus yang luas dan mobilitas yang tinggi, algoritma ini diharapkan mampu memberikan solusi optimal dalam menentukan rute yang paling efisien bagi mahasiswa, dosen, maupun pengunjung.

Rumusan Masalah

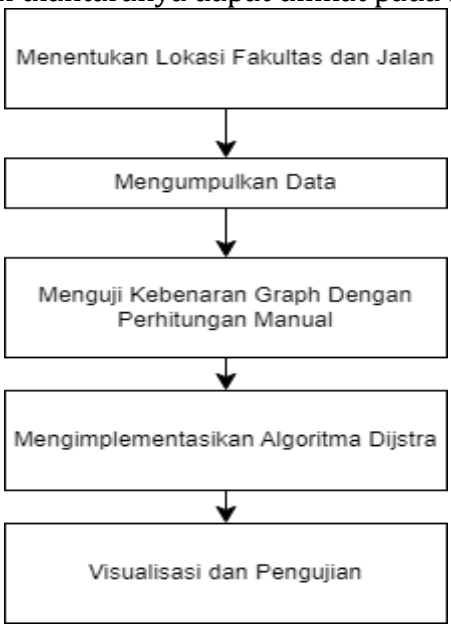
Penelitian ini akan mengkaji beberapa aspek penting terkait penerapan algoritma Dijkstra untuk penentuan rute terpendek di lingkungan kampus Universitas Negeri Medan. Rumusan masalah yang akan dibahas meliputi:

1. Bagaimana cara mengimplementasikan algoritma Dijkstra untuk menentukan rute terpendek antar fakultas di Universitas Negeri Medan?
2. Sejauh mana efektivitas algoritma Dijkstra dalam mencari rute terpendek di lingkungan kampus, khususnya untuk mempermudah mobilitas mahasiswa, dosen, dan pengunjung?
3. Faktor-faktor apa saja yang mempengaruhi hasil pencarian rute terpendek, seperti jarak antar bangunan, aksesibilitas, dan kondisi lingkungan sekitar?
4. Bagaimana penerapan algoritma Dijkstra dapat membantu dalam perencanaan mobilitas yang lebih efisien, dengan studi kasus Fakultas Teknik (FT) sebagai titik awal dan Fakultas Ilmu Pendidikan (FIP) sebagai titik akhir?

Penelitian ini diharapkan mampu memberikan kontribusi signifikan dalam meningkatkan efektivitas mobilitas di kampus, serta menawarkan solusi yang lebih baik dalam pengelolaan jalur pergerakan di lingkungan Universitas Negeri Medan.

METODE PENELITIAN

Metode Penelitian yang digunakan dalam penelitian ini adalah metode kuantitatif, yang berfokus pada pengumpulan dan analisis data numerik untuk menemukan pola atau solusi optimal. Algoritma Dijkstra digunakan untuk menghitung rute terpendek berdasarkan bobot (jarak atau waktu) antara fakultas. Untuk memulai penerapan algoritma Dijkstra dalam penentuan rute terpendek antar fakultas di Universitas Negeri Medan, berikut adalah langkah-langkah awal yang akan dilakukan diantaranya dapat dilihat pada bagan alir berikut ini :



Gambar 1. Flowchart Penelitian

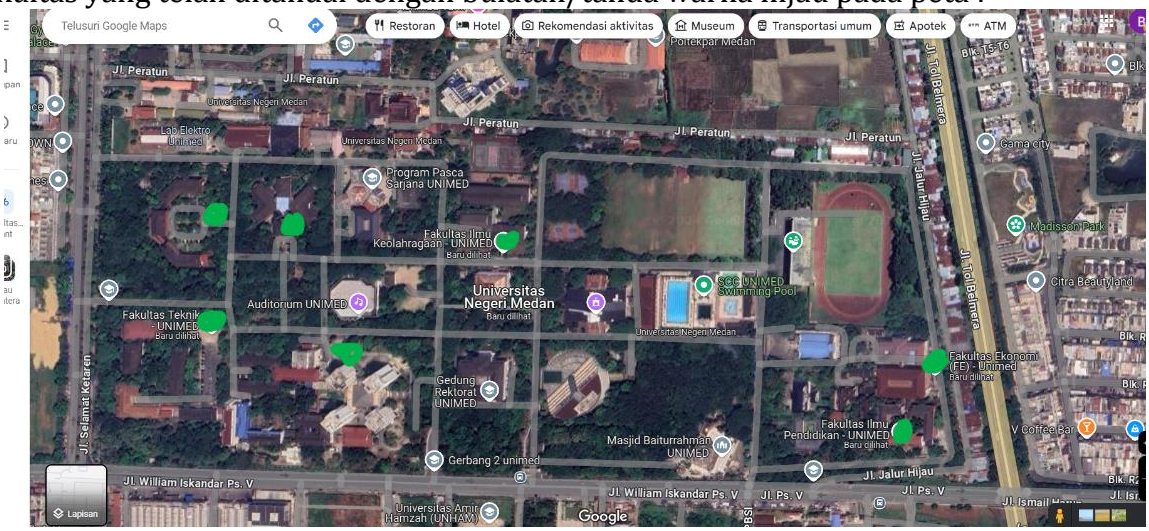
1. Memetakan Lokasi Fakultas dan Jalan. Langkah pertama adalah menentukan lokasi fakultas di Universitas Negeri Medan dan jalan yang menghubungkan setiap fakultas. Dalam hal ini, setiap fakultas akan direpresentasikan sebagai node/vertex, sementara jalan yang menghubungkan fakultas sebagai edge dalam sebuah graph. Penentuan bobot (jarak atau waktu tempuh) pada setiap edge sangat penting untuk memastikan akurasi dalam perhitungan rute terpendek.
2. Mengumpulkan Data. Data yang diperlukan mencakup jarak fisik atau estimasi waktu tempuh antara fakultas. Data ini bisa diperoleh melalui pengukuran langsung, peta digital, atau data dari pihak kampus dalam hal ini kami menggunakan bantuan google maps untuk menghitung jarak antar kampus. Akurasi data ini akan mempengaruhi hasil perhitungan algoritma, sehingga penting untuk melakukan validasi pada data yang terkumpul.
3. Menguji Kebenaran Graph Dengan Perhitungan Manual. Sebelum mengimplementasikan algoritma Dijkstra, lakukan uji manual pada graph yang sudah dibuat. Perhitungan ini untuk memastikan bahwa representasi graph dan bobotnya sudah sesuai. Tes manual ini biasanya melibatkan penghitungan jalur terpendek menggunakan prinsip-prinsip dasar teori graph.
4. Mengimplementasikan Algoritma Dijkstra. Setelah graph dan data selesai diuji, langkah selanjutnya adalah mengimplementasikan algoritma Dijkstra dalam bahasa pemrograman yang dipilih, misalnya Java. Algoritma ini akan mencari rute terpendek dari satu fakultas ke fakultas lainnya, berdasarkan graph yang sudah dibangun. Penting untuk mendefinisikan node awal (starting point) dan node tujuan (ending point) sebagai input dalam proses ini.
5. Visualisasi dan Pengujian. Setelah algoritma berjalan, hasilnya perlu diverifikasi dengan membandingkan rute yang dihasilkan dengan rute aktual di lapangan. Visualisasi graph dan rute terpendek akan membantu memahami hasil perhitungan algoritma. Tahap ini juga

melibatkan pengujian untuk memastikan hasil algoritma konsisten dan akurat untuk berbagai rute antar fakultas.

HASIL PENELITIAN DAN PEMBAHASAN

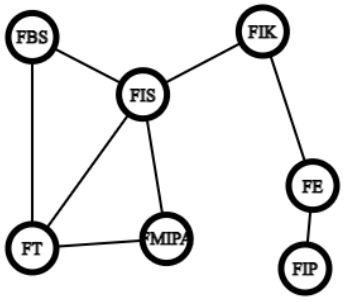
Tahap Analisis

1. Memetakan Lokasi fakultas dan Jalan. Langkah awal dalam penerapan algoritma Dijkstra untuk menentukan rute terpendek antar fakultas di Universitas Negeri Medan dimulai dengan memetakan seluruh lokasi fakultas beserta jalur-jalur yang menghubungkan antar fakultas tersebut. Pada tahap ini, gambar peta Universitas Negeri Medan yang menunjukkan posisi setiap fakultas akan digunakan sebagai acuan untuk menentukan simpul (node) dalam graf. Setiap simpul mewakili fakultas, sedangkan garis yang menghubungkan simpul-simpul tersebut melambangkan jalur yang dapat dilalui, baik jalan utama maupun jalan alternatif di sekitar kampus.Berikut adalah peta Universitas Negeri Medan yang menunjukkan posisi fakultas yang telah ditandai dengan bulatan/tanda warna hijau pada peta :



Gambar 2. Peta Universitas Negeri Medan

Setelah memetakan lokasi pada peta, kita akan mentransformasikan informasi ini ke dalam bentuk graf. Dalam graf ini, setiap fakultas diwakili sebagai simpul, dan setiap jalur penghubung antar fakultas diwakili sebagai sisi (edge). Setiap sisi diberi bobot, yang bisa berupa jarak fisik antar fakultas atau waktu tempuh yang diperlukan. Visualisasi graf ini dapat dilihat pada gambar berikut:



Gambar 3. Implementasi Peta Ke Graph

Graph yang menggambarkan hubungan antar fakultas di Universitas Negeri Medan untuk mencari rute terpendek ini dibuat tidak berarah karena dalam konteks ini, kita biasanya mengasumsikan bahwa jalur antar fakultas dapat ditempuh bolak-balik dengan

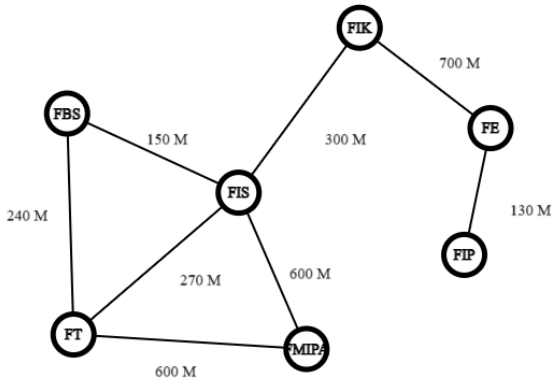
jarak yang sama. Dengan kata lain, mahasiswa atau orang yang bergerak dari satu fakultas ke fakultas lain dapat kembali ke fakultas asal dengan mengikuti jalur yang sama, dan jaraknya akan tetap sama di kedua arah Graf tersebut akan digunakan sebagai input dalam implementasi algoritma Dijkstra untuk menghitung rute terpendek antar fakultas.

2. Mengumpulkan Data. Pada tahap pengumpulan data, kami mengidentifikasi setiap fakultas di Universitas Negeri Medan sebagai simpul dalam graph, dan jalur yang menghubungkan antar fakultas sebagai sisi. Setiap sisi diberikan bobot yang mewakili jarak antar fakultas, yang diukur dalam meter berdasarkan data jarak sebenarnya. Bobot ini akan digunakan oleh algoritma Dijkstra untuk menentukan rute terpendek. Adapun rincian jarak antar fakultas adalah sebagai berikut:

Table 1. Tabel jarak antar fakultas

No	Fakultas	Total Jarak (Meter)
1	FT → FIS	270
2	FT → FMIPA	600
3	FIS → FIK	300
4	FE → FIP	130
5	FIS → FBS	150
6	FT → FBS	240
7	FIS → FMIPA	600
8	FIK FIK → FE	700

Data jarak ini merupakan input penting dalam perhitungan rute terpendek, di mana setiap bobot akan berpengaruh pada hasil akhir yang dihasilkan oleh algoritma Dijkstra.Dengan telah diketahuinya jarak maka dapat dikatakan bahwa bobot dapat mewakili jarak yang mana juga dapat mengubah gambaran dari graph tersebut menjadi seperti berikut:



Gambar 4. Graph Berbobot

3. Menguji Kebenaran Graph Dengan Perhitungan Manual. Langkah-langkah Dijkstra untuk Menemukan Rute Terpendek dalam hal ini Fakultas Teknik(FT) ditetapkan sebagai node awal dan Fakultas Ilmu Pendidikan(FIP) sebagai Node tujuan(akhir):
   a. Inisialisasi: Set jarak dari simpul awal FT ke semua simpul lain sebagai ∞ (tak terhingga), kecuali untuk simpul awal sendiri yang memiliki jarak 0:
      - Jarak FT = 0
      - Jarak FIS = ∞
      - Jarak FMIPA = ∞
      - Jarak FBS = ∞
      - Jarak FIK = ∞

- Jarak FE = ∞
- Jarak FIP = ∞

Set semua node sebagai unvisited.

b. Proses Node Awal (FT): Kita mulai dari simpul FT. Tentukan jarak dari FT ke simpul simpul tetangganya:
- FT → FIS = 270 meter
- FT → FMIPA = 600 meter
- FT → FBS = 240 meter

Update jarak minimum:
- Jarak FIS = 270 meter
- Jarak FMIPA = 600 meter
- Jarak FBS = 240 meter

Simpul FT sekarang dianggap telah dikunjungi.

c. Pilih Simpul dengan Jarak Minimum (FBS): Dari simpul yang belum dikunjungi, kita pilih simpul dengan jarak terpendek dari simpul awal, yaitu FBS (240 meter), Kemudian, hitung jarak dari FBS ke simpul lainnya:
- FBS → FIS = 240 + 150 = 390 meter (lebih besar dari jarak langsung dari FT → FIS, yaitu 270 meter, jadi kita abaikan).

Tidak ada update jarak baru.
Simpul FBS sekarang dianggap telah dikunjungi.

d. Pilih Simpul dengan Jarak Minimum (FIS): Selanjutnya, pilih simpul dengan jarak minimum yang belum dikunjungi, yaitu FIS (270 meter)
Lalu, hitung jarak dari FIS ke simpul tetangganya:
- FIS → FIK = 270 + 300 = 570 meter.
- FIS → FMIPA = 270 + 600 = 870 meter (lebih besar dari jarak langsung FT → FMIPA, yaitu 600 meter, jadi kita abaikan).

Update jarak minimum:
- Jarak FIK = 570 meter

Simpul FIS sekarang dianggap telah dikunjungi.

e. Pilih Simpul dengan Jarak Minimum (FIK): Berikutnya, pilih simpul dengan jarak minimum yang belum dikunjungi, yaitu FIK (570 meter).
Hitung jarak dari FIK ke simpul tetangganya:
- FIK → FE = 570 + 700 = 1270 meter.

Update jarak minimum:
- Jarak FE = 1270 meter

Simpul FIK sekarang dianggap telah dikunjungi.

f. Pilih Simpul dengan Jarak Minimum (FMIPA): Sekarang pilih simpul dengan jarak minimum yang belum dikunjungi, yaitu FMIPA (600 meter). Namun, FMIPA tidak memiliki tetangga baru yang belum diproses, sehingga tidak ada update jarak baru.
Simpul FMIPA sekarang dianggap telah dikunjungi.

g. Pilih Simpul dengan Jarak Minimum (FE): Terakhir, pilih simpul dengan jarak minimum yang belum dikunjungi, yaitu FE (1270 meter)
Hitung jarak dari FE ke simpul tetangganya:
- FE → FIP = 1270 + 130 = 1400 meter.

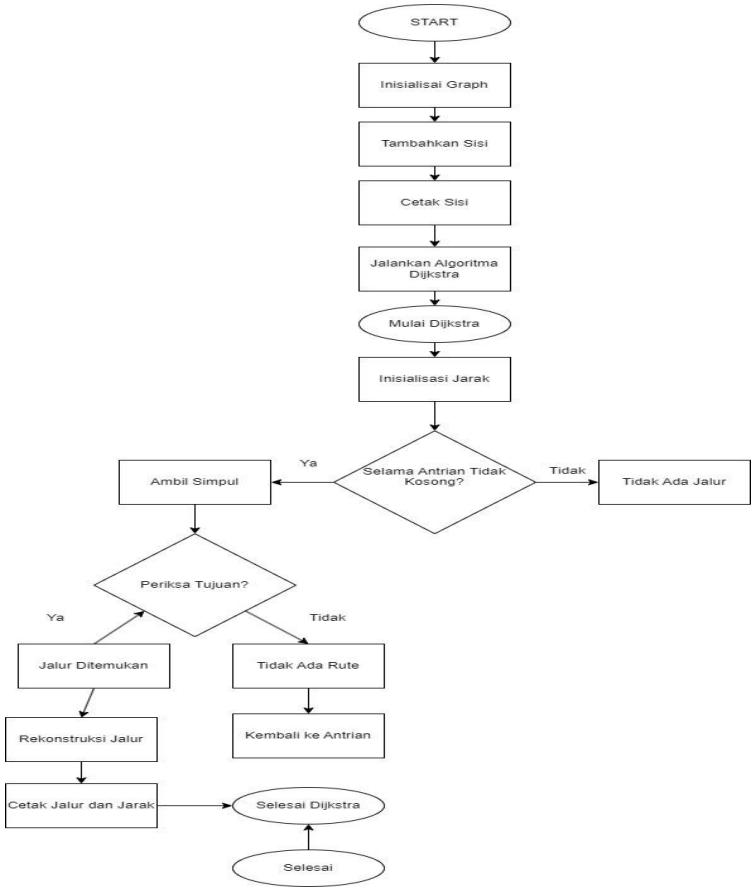
Update jarak minimum:
- Jarak FIP = 1400 meter

Simpul FE sekarang dianggap telah dikunjungi.

h. Simpul Tujuan Tercapai (FIP): Simpul tujuan FIP sekarang sudah diproses dengan jarak terpendek 1400 meter.
Ini adalah jarak minimum dari FT ke FIP melalui jalur:
- FT → FIS → FIK → FE → FIP.
Perhitungan Matematika (Menggunakan Rumus)
1) Inisialisasi Jarak Awal: *d(FT) = 0, d(node lainnya) = ∞*
2) Update Jarak: Rumus update jarak:
d(v) = min(d(v), d(u) + weight(u, v))
Di mana u adalah simpul yang sedang diproses, v adalah simpul tetangga dari u, dan weight(u, v) adalah bobot/jarak dari u ke v.
Pada setiap langkah, jarak diperbarui jika rute melalui simpul baru memberikan jarak yang lebih kecil.
3) Pemilihan Simpul Minimum: Setelah jarak diperbarui, kita memilih simpul dengan jarak terpendek yang belum diproses. Proses ini diulang hingga semua simpul diproses atau simpul tujuan (FIP) ditemukan dengan jarak Dengan demikian, rute ini adalah hasil dari penerapan algoritma Dijkstra secara manual, yang dapat diuji lebih lanjut menggunakan program untuk memastikan keakuratan dan efisiensi perhitungan.

Tahap Implementasi

Pada bagian pengujian ini kmai menggunakan bahasa pemrograman c++ Sebagai bahasa untuk pembuktian perhitungan manual yang telah dilakukan diawal namun pertama sekali kami membuat flowchart dan juga pseudocode sebelum mengaplikasikan nya kedalam bahasa pemrograman yang diinginkan .Flowchart nya adalah sebagai berikut



Gambar 5. Flowchart Program

Flowchart pada gambar tersebut menjelaskan tahapan-tahapan dari algoritma Dijkstra untuk menemukan rute terpendek dalam sebuah graf. Berikut adalah penjelasan dari setiap langkah dalam flowchart tersebut:

1. START: Proses dimulai dari inisiasi algoritma Dijkstra.
2. Inisialisasi Graph: Pada tahap ini, dilakukan inisialisasi graf, yaitu mendefinisikan simpul (nodes) dan sisi (edges) yang membentuk struktur graf. Setiap simpul mewakili titik dan setiap sisi memiliki bobot (jarak).
3. Tambahkan Sisi: Sisi atau edge dengan bobot tertentu ditambahkan ke graf, menghubungkan simpul-simpul.
4. Cetak Sisi: Semua sisi yang telah ditambahkan dicetak, mungkin untuk memastikan bahwa input graf telah benar sebelum menjalankan algoritma.
5. Jalankan Algoritma Dijkstra: Algoritma Dijkstra dijalankan untuk menghitung rute terpendek antara simpul asal dan simpul tujuan.
6. Mulai Dijkstra: Algoritma Dijkstra dimulai dengan inisialisasi nilai awal dari jarak setiap simpul ke simpul awal. Jarak dari simpul asal ke simpul lainnya diatur ke nilai tak terhingga kecuali simpul asal yang diinisialisasi dengan nilai 0.
7. Inisialisasi Jarak: Nilai jarak dari simpul asal diinisialisasi menjadi 0 dan simpul lainnya dengan nilai tak terhingga. Antrian prioritas (priority queue) digunakan untuk menyimpan simpul yang akan diproses.
8. Selama Antrian Tidak Kosong?: Selama ada simpul dalam antrian prioritas, algoritma akan terus berjalan. Jika antrian kosong dan tidak ditemukan rute, algoritma berakhir tanpa menemukan jalur.
   a. Ya: Jika antrian tidak kosong, lanjut ke langkah berikutnya.
   b. Tidak: Jika antrian kosong, berarti tidak ada jalur yang dapat ditemukan, dan proses dihentikan.
9. Ambil Simpul: Ambil simpul dengan nilai jarak terkecil dari antrian prioritas untuk diproses.
10. Periksa Tujuan?: Setelah simpul diambil, diperiksa apakah simpul tersebut merupakan simpul tujuan
    - Ya: Jika simpul tersebut adalah simpul tujuan, jalur telah ditemukan.
    - Tidak: Jika bukan simpul tujuan, maka proses dilanjutkan dengan memeriksa tetangga-tetangga dari simpul tersebut.
11. Jalur Ditemukan/Rekonstruksi Jalur: Jika simpul tujuan ditemukan, algoritma akan merekonstruksi jalur dari simpul asal ke simpul tujuan berdasarkan informasi yang telah dikumpulkan.
12. Cetak Jalur dan Jarak: Setelah jalur direkonstruksi, jalur dan jarak terpendek dicetak sebagai hasil akhir dari algoritma.
13. Selesai Dijkstra: Proses algoritma Dijkstra selesai.
14. Selesai: Proses pencarian rute terpendek selesai.

Alur ini secara keseluruhan menggambarkan bagaimana algoritma Dijkstra mencari rute terpendek dari satu simpul ke simpul lainnya di dalam sebuah graf. Setelah itu dilakukan pembuatan pseudocode nya sebelum diimplementasikan kedalam bahasa pemrogramannya untuk mempermudah penulisan pada saat pembuatan yang mana pseudocodenya yaitu sebagai berikut

PSEUDOCODE:

```
Class Graph:
 Initialize V (number of vertices)
 Initialize adj (adjacency list)
 Function addEdge(u, v, w):
```

```
Add edge from u to v with weight w in adj
Add edge from v to u with weight w in adj (undirected graph)
Function printEdges():
Print distances between faculties
Print visualization of faculties with their corresponding indices
Function dijkstra(src, dest):
Initialize priority queue pq
Initialize dist array with INT_MAX
Initialize prev array with -1
Set dist[src] = 0
Push (0, src) into pq
While pq is not empty:
Set u = pq.top().second
Pop pq
If u == dest:
Break
For each neighbor in adj[u]:
Set v = neighbor.first
Set weight = neighbor.second
If dist[u] + weight < dist[v]:
Update dist[v]
Update prev[v] with u
Push (dist[v], v) into pq
If dist[dest] == INT_MAX:
Print "No route from src to dest"
Else:
Print total distance
Print path from src to dest
Initialize totalDistance = 0
For each i in path:
Print each segment and its distance
Add distance to totalDistance
Print totalDistance
Main:
Create graph g with 7 vertices
Add edges to g
Call g.printEdges()
Call g.dijkstra(0, 5) // From FT to FIP
```

Lalu pseudocode yang telah dirancang sebelumnya kemudian diimplementasikan kedalam kode program .Yang mana disini kami menggunakan bahasa pemrograman c++ sebagai bahasa pengimplementasian nya . Bahasa C++ dipilih karena efisien dalam penggunaan memori dan kecepatan eksekusi, terutama penting untuk algoritma seperti Dijkstra. Dukungan struktur data dari STL (seperti priority queue) memudahkan implementasi, serta C++ menawarkan skalabilitas dan portabilitas tinggi untuk program yang kompleks.

KODE PROGRAM C++

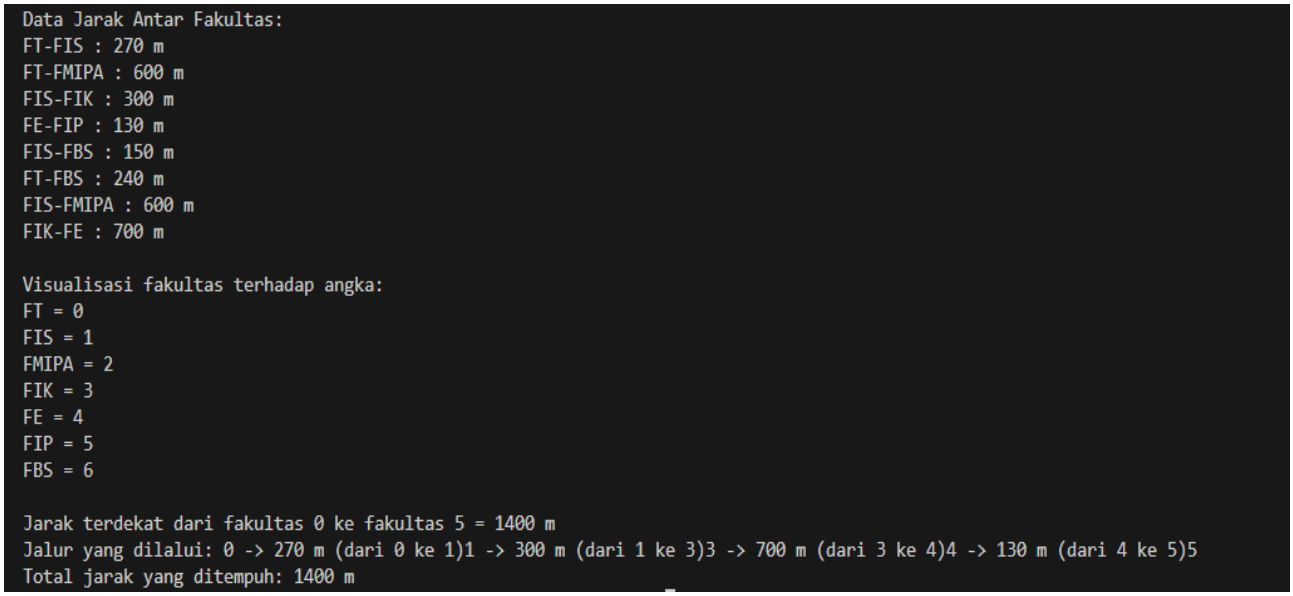
```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
#include <algorithm>
```

```
using namespace std;
class Graph {
public:
 int V;
 vector<pair<int, int>>* adj;
 Graph(int V) {
 this->V = V;
 adj = new vector<pair<int, int>>[V];
 }
 void addEdge(int u, int v, int w) {
 adj[u].push_back(make_pair(v, w));
 adj[v].push_back(make_pair(u, w));
 }
 void printEdges() {
 cout << "Data Jarak Antar Fakultas:\n";
 cout << "FT-FIS : 270 m\n";
 cout << "FT-FMIPA : 600 m\n";
 cout << "FIS-FIK : 300 m\n";
 cout << "FE-FIP : 130 m\n";
 cout << "FIS-FBS : 150 m\n";
 cout << "FT-FBS : 240 m\n";
 cout << "FIS-FMIPA : 600 m\n";
 cout << "FIK-FE : 700 m\n\n";
 cout << "Visualisasi fakultas terhadap angka:\n";
 cout << "FT = 0\n";
 cout << "FIS = 1\n";
 cout << "FMIPA = 2\n";
 cout << "FIK = 3\n";
 cout << "FE = 4\n";
 cout << "FIP = 5\n";
 cout << "FBS = 6\n";
 cout << endl;
 }
 void dijkstra(int src, int dest) {
 priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
 vector<int> dist(V, INT_MAX);
 vector<int> prev(V, -1);
 dist[src] = 0;
 pq.push(make_pair(0, src));
 while (!pq.empty()) {
 int u = pq.top().second;
 pq.pop();
 if (u == dest) {
 break;
 }
 for (auto &neighbor : adj[u]) {
 int v = neighbor.first;
 int weight = neighbor.second;
 if (dist[u] + weight < dist[v]) {
 dist[v] = dist[u] + weight;
 prev[v] = u;
 pq.push(make_pair(dist[v], v));
 }}}
```

```
if (dist[dest] == INT_MAX) {
cout << "Tidak ada rute dari fakultas " << src << " ke fakultas " << dest << ".\n";
} else {
cout << "Jarak terdekat dari fakultas " << src << " ke fakultas " << dest << " = " << dist[dest] << " m\n";
cout << "Jalur yang dilalui: ";
vector<int> path;
for (int at = dest; at != -1; at = prev[at]) {
path.push_back(at);
}
reverse(path.begin(), path.end());
int totalDistance = 0;
for (int i = 0; i < path.size(); i++) {
cout << path[i];
if (i < path.size() - 1) {
cout << " -> ";
for (const auto &neighbor : adj[path[i]]) {
if (neighbor.first == path[i + 1]) {
cout << neighbor.second << " m (dari " << path[i] << " ke " << path[i + 1] << ")";
totalDistance += neighbor.second;
break;
}}}}
cout << endl;
cout << "Total jarak yang ditempuh: " << totalDistance << " m\n";
}}
~Graph() {
delete[] adj;
}};
int main() {
Graph g(7);
g.addEdge(0, 1, 270);
g.addEdge(1, 3, 300);
g.addEdge(3, 4, 700);
g.addEdge(4, 5, 130);
g.addEdge(0, 2, 600);
g.addEdge(1, 6, 150);
g.addEdge(0, 6, 240);
g.addEdge(1, 2, 600);
g.printEdges();
g.dijkstra(0, 5);
return 0;
```

Tahap Hasil

Program tersebut dijalankan dan dapat dilihat hasilnya pada gambar berikut:



Gambar 6. Hasil Program

Dapat dilihat bahwa hasil perhitungan yang dilakukan secara manual sebelumnya sesuai dengan hasil pengujian menggunakan bahasa pemrograman yang kami pilih yaitu C++.Maka dapat dikatakan bahwa percobaan ini telah sesuai dengan apa yg telah diinginkan walaupun masih ada kekurangan lainnya namun hal ini sudah cukup untuk membuktikan kebenaran dari perhitungan secara manual tersebut.

KESIMPULAN

Dalam "Pencarian Rute Terpendek Antar Fakultas di Universitas Negeri Medan Menggunakan Algoritma Dijkstra", algoritma Djikstra digunakan untuk menemukan rute terpendek antara fakultas di Universitas Negeri Medan dapat ditarik kesimpulan sebagai berikut:

1. Penerapan Algoritma Dijkstra: Algoritma ini digunakan untuk menemukan rute terpendek antara fakultas. Ini bertujuan untuk membuat mobilitas di lingkungan kampus lebih efisien untuk semua orang, termasuk dosen dan mahasiswa.
2. Data yang Digunakan: Algoritma ini menggunakan data dari Google Maps untuk menghitung jrrak antar fakultas. Setiap fakultas digambarkan dalam graf sebagai simpul (node) dan jarak antar fakultas digambarkan sebagai sisi (edge).
3. Langkah Penelitian: Penelitian ini melibatkan implementasi algoritma Djikstra melalui pemrograman, pengujian graf secara manual, pengumpulan data jarak antar fakultas, dan pemetaan lokasi fakultas. Selanjutnya, hasil perhitungan manual dibandingkan dengan hasil program untuk memastikan keakuratannya.
4. Hasil: Algoritma Djikstra berhasil menemukan rute terpendek antara Fakultas Teknik (FT) dan Fakultas Ilmu Pendidikan (FIP) dengan jarak total 1400 meter. Rute yang dipilih adalah FT hingga FIS hingga FIK hingga FE hingga FIP.
5. Manfaat: Diharapkan bahwa penggunaan algoritma ini akan mempermudah perencanaan perjalanan antar fakultas di kampus, dengan menghemat waktu dan energi.

Hasil penelitian ini menunjukkan bahwa algoritma Djikstra berhasil menentukan jalur terpendek di lingkungan kampus dan menunjukkan bahwa data jarak yang akurat sangat penting untuk proses perhitungan.

DAFTAR PUSTAKA

Antara Dalem, I. B. G. W. (2018). Penerapan Algoritma A* (Star) Menggunakan Graph untuk Menghitung Jarak Terpendek. *Jurnal Resistor*, *1*(1), April.

Ardiansyah, A., Efendi, F. S., Syaifullah, S., Pinto, M., Pujianto, P., & Tempake, H. S. (2012). Implementasi Algoritma Greedy Untuk Melakukan Graph Coloring: Studi Kasus Peta Propinsi Jawa Timur. *Jurnal Informatika Ahmad Dahlan*, *4*(2), 103610.

Arga, E. S., Firmansyah, G. G., Imam, K., & Fauzi, M. (2021). Penerapan algoritma djikstra pada pencarian jalur terpendek. *Jurnal Bayesian: Jurnal Ilmiah Statistika dan Ekonometrika*, *1*(2), 134-142.

Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs." Numerische Mathematik, 1959.

Luthfita, D., & Aripin, S. (2022). Implementasi Algoritma A* Dalam Menentukan Tarif Minimum Berdasarkan Jarak Terpendek Rute Armada Taksi Bandara. *Journal of Informatics Management and Information Technology*, *2*(1), 43-47.

Mahardika, F. (2019). Penerapan Teori Graf pada Jaringan Komputer dengan Algoritma Kruskal. *Jurnal Informatika: Jurnal Pengembangan IT (JPIT)*, *4*(1).

Sari, P. M., Fauziah, F., & Gunaryati, A. (2021). Implementasi Algoritma Dijkstra pada Aplikasi Go-Tahu dengan Pencarian Rute Terpendek ke Pabrik Tahu. *Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi)*, *5*(2), 103-111.

Sedgewick, Robert. "Algorithms in Java, Parts 1-4." Addison-Wesley Professional, 2002.

Syahputra, T., & Setiawan, D. (2016). Implementasi Algoritma Prim dengan Teori Graph pada WPF Graph. *Jurnal Teknologi dan Sistem Informasi*, *2*(2), 65-71.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. "Introduction to Algorithms." MIT Press, 2009.

Yaqin, A., dkk. (2023). Penerapan Teori Graf pada Pengaturan Lampu Lalu Lintas di Perempatan Alun-Alun Kota Bojonegoro. *Buana Matematika: Jurnal Ilmiah Matematika dan Pendidikan Matematika*, *13*(2).