

LAPORAN PROYEK
DESAIN ANALISIS ALGORITMA
“Implementasi Algoritma *BFS* dan *DFS* dalam Penyelesaian
Traveling Salesman Problem (TSP) Kota di Jawa Timur”



Dosen Pengampu:
Dr. Atik Winarti, M.Kom.
Yuni Rosita Dewi, M.Si.

Disusun Oleh:
Kelompok 01 Kelas 2023B

Ivanagita Mayang Palupi	(23031554009)
Sintiya Risla Miftaqul Nikmah	(23031554204)
Theopan Gerard Nainggolan	(23031554227)

PRODI S1 SAINS DATA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI SURABAYA
2024/2025

DAFTAR ISI

BAB I PENDAHULUAN.....	3
1.1 Latar Belakang.....	3
1.2 Rumusan Masalah.....	3
1.4 Manfaat.....	3
BAB II METODE.....	4
2.1 Identifikasi Masalah TSP.....	4
2.1.1 Definisi Masalah.....	4
2.1.2 Batasan Lingkup.....	4
2.2 Pengumpulan dan Pemrosesan Dataset.....	5
2.2.1 Pengumpulan Dataset.....	5
2.2.2 Pre-Processing Dataset.....	6
2.3 Perancangan Algoritma.....	7
2.3.1 Perancangan Algoritma BFS.....	7
2.3.2 Perancangan Algoritma DFS.....	8
2.4 Implementasi Algoritma.....	9
2.4.1 Implementasi Algoritma BFS.....	9
2.4.2 Implementasi Algoritma DFS.....	10
BAB III HASIL DAN ANALISIS.....	11
3.1 Hasil Deployment Proyek.....	11
3.2 Analisis Hasil Algoritma.....	18
3.2.1 Analisis Hasil Algoritma BFS.....	18
3.2.2 Analisis Hasil Algoritma DFS.....	19
3.3 Perbandingan Kedua Algoritma.....	20
BAB IV KESIMPULAN.....	21
DAFTAR PUSTAKA.....	22
KONTRIBUSI ANGGOTA.....	23

BAB I

PENDAHULUAN

1.1 Latar Belakang

Travelling Salesman Problem (TSP) merupakan permasalahan yang kedengarannya sederhana namun telah menjadi permasalahan yang diteliti secara intensif di bidang matematika komputer (Applegate et al., 2011). TSP adalah suatu masalah yang berfokus untuk menemukan rute terpendek dengan mengunjungi setiap node satu kali dan kembali lagi ke node awal. Masalah TSP sering dijumpai dalam berbagai bidang, seperti logistik, transportasi, dan perencanaan rute seperti namanya. Untuk mengoptimalkan biaya, waktu, dan tenaga, maka perlu dicari jalur optimal untuk melakukan perjalanan tersebut (Sutoyo, 2018). Karena itu, kami menggunakan implementasi algoritma pencarian *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS) untuk menyelesaikan TSP dengan batasan maksimal 10 kota di Jawa Timur (sebagai node). Kami memilih kedua algoritma karena sifatnya yang sistematis dan pendekatan yang mereka lakukan juga berbeda dalam mengeksplorasi kemungkinan rute. Secara garis besar, proyek ini juga bertujuan untuk menganalisis kinerja BFS dan DFS, baik dari segi efisiensi waktu, penggunaan memori, maupun kompleksitas algoritmik, guna memahami kelebihan dan kekurangan setiap algoritma dalam menyelesaikan TSP.

1.2 Rumusan Masalah

- Bagaimana algoritma BFS dan DFS diimplementasikan untuk menyelesaikan TSP pada data kota di Jawa Timur?
- Sejauh mana efisiensi masing-masing algoritma dalam menyelesaikan TSP?
- Algoritma mana yang lebih efisien untuk diterapkan pada kasus TSP dengan kompleksitas tertentu?

1.3 Tujuan

- Mengimplementasikan algoritma BFS dan DFS untuk menyelesaikan TSP dengan menggunakan data kota di Jawa Timur.
- Menganalisis efisiensi algoritma BFS dan DFS dalam menemukan solusi TSP.
- Memberikan rekomendasi terkait algoritma yang lebih sesuai untuk digunakan pada masalah TSP berdasarkan hasil analisis.

1.4 Manfaat

Penelitian ini diharapkan dapat memberikan kontribusi dalam memahami penerapan algoritma BFS dan DFS untuk menyelesaikan masalah optimasi rute atau TSP, memberikan referensi bagi pengembangan sistem logistik atau transportasi yang lebih efisien di Jawa Timur, serta menyediakan wawasan bagi mahasiswa dan peneliti mengenai implementasi praktis algoritma BFS dan DFS dalam konteks dunia nyata, yang dapat menjadi dasar pengembangan penelitian lebih lanjut maupun membantu pembuat kebijakan merancang rute perjalanan yang hemat biaya dan waktu.

BAB II

METODE

2.1 Identifikasi Masalah TSP

2.1.1 Definisi Masalah

Travelling Salesman Problem (TSP) adalah masalah optimasi yang bertujuan untuk menemukan rute terpendek yang melewati semua kota dalam suatu graf berbobot tepat satu kali dan kembali ke kota asal. Permasalahan ini dikenal juga sebagai permasalahan optimasi klasik dan bersifat *Non Deterministic Polynomial-time Complete* (NPC), yang artinya harus mencoba semua kemungkinan rute yang ada untuk mendapatkan rute yang paling optimal (Alderio & Sari, 2023). Dalam proyek kami ini, masalah TSP difokuskan pada kota-kota yang ada di Jawa Timur, di mana graf merepresentasikan kota sebagai node dan jarak antar kota sebagai bobot pada edge. Permasalahan ini menuntut eksplorasi rute yang efisien, terutama pada kondisi graf dengan banyak node.

2.1.2 Batasan Lingkup

Untuk memastikan efisiensi implementasi algoritma, kami membatasi masalah pada maksimal 10 kota yang telah kami pilih secara representatif di Jawa Timur. Batasan ini diterapkan agar algoritma *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS) dapat dijalankan secara optimal dengan mempertimbangkan keterbatasan sumber daya komputasi, seperti waktu eksekusi dan memori. Selain itu, batasan ini memungkinkan analisis kinerja kedua algoritma dilakukan dengan lebih terukur.

2.2 Pengumpulan dan Pemrosesan Dataset

2.2.1 Pengumpulan Dataset

Pada analisis ini, kami memanfaatkan dataset yang diambil dari situs resmi [Badan Pusat Statistik \(BPS\) Provinsi Jawa Timur](#). Dataset tersebut berisi informasi mengenai jarak antar kota di Provinsi Jawa Timur, yang sangat relevan untuk mendukung proyek kami.

Kami memilih sumber data ini karena merupakan informasi yang akurat yang disusun oleh BPS untuk memberikan gambaran yang jelas tentang distribusi geografis dan jarak tempuh antar kota di Jawa Timur. Dengan menggunakan data ini, kami dapat mengeksplorasi berbagai solusi dalam pemrograman dan analisis.

Di bawah ini adalah dataset kotor sebelum dilakukan *pre-processing*:

[illegible]

2.2.2 Pre-Processing Dataset

Proses *pre-processing* data yang dilakukan dimulai dengan membaca file CSV yang berisi informasi jarak antar kota dan menetapkan baris kedua sebagai *header* kolom, sambil menghapus dua baris pertama yang tidak diperlukan. Selanjutnya, kolom pertama diubah namanya menjadi "Kota/Kab" untuk meningkatkan konsistensi dan pemahaman. Kolom yang tidak relevan, seperti kolom dengan nilai *Mising Values*, dihapus, dan baris dengan nilai kosong juga dihilangkan. Setelah itu, seluruh dataframe diubah menjadi tipe string untuk menangani nilai yang tidak konsisten, dan karakter (-) yang menandakan nilai yang hilang diganti dengan angka 0. Kolom yang berisi data numerik kemudian dikonversi menjadi tipe integer, dengan nilai yang tidak bisa dikonversi diubah menjadi 0. Akhirnya, hasil pembersihan data ini disimpan dalam file CSV baru untuk analisis lebih lanjut. Proses *pre-processing* ini memastikan data yang bersih, konsisten, dan siap untuk diproses lebih lanjut dalam proyek yang kami lakukan.

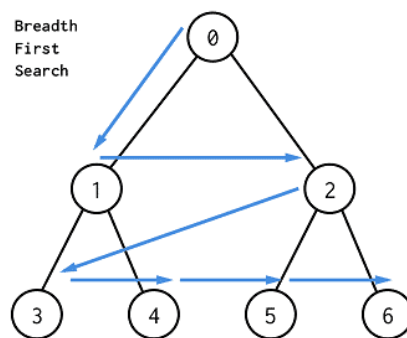
```
,Kota/Kab,Surabaya,Gresik,Sidoarjo,Mojokerto,Jombang,Bojonegoro,Lamongan ,Tuban,Madiun ,Ngawi,Magetan,Ponorogo,Pacitan,Kediri,Nganjuk,
1,Surabaya,0,18,23,49,79,108,45,103,169,181,193,198,276,123,119,154,167,186,89,60,99,145,191,194,197,288,28,90,123,175
2,Gresik,18,0,41,67,97,90,27,85,187,199,211,216,294,141,137,172,185,204,107,78,117,163,209,212,215,306,46,108,141,193
3,Sidoarjo,23,41,0,72,102,131,68,126,192,204,216,221,298,145,142,177,144,208,66,37,76,122,168,171,174,265,51,113,146,198
4,Mojokerto,49,67,72,0,30,115,57,110,120,132,144,149,227,74,70,105,118,137,86,61,100,146,192,195,198,289,77,139,172,224
5,Jombang,79,97,102,30,0,85,80,81,90,102,114,119,197,44,40,75,88,107,119,91,130,176,222,225,228,319,107,169,202,254
6,Bojonegoro,108,90,131,115,85,0,63,65,110,78,112,139,217,129,125,160,173,192,197,168,207,253,298,202,305,396,136,198,231,283
7,Lamongan ,45,27,68,57,80,63,0,58,177,187,201,206,284,131,127,162,175,194,134,105,144,190,236,239,242,333,73,135,168,220
8,Tuban,103,95,126,110,82,65,58,0,172,184,196,201,279,126,122,157,170,189,191,163,202,248,284,297,300,391,131,193,226,278
9,Madiun ,169,187,192,128,90,110,177,182,0,32,24,29,107,78,50,109,122,82,178,181,220,266,312,315,338,407,127,259,292,344
10,Ngawi,181,199,264,132,102,78,189,184,32,0,34,61,139,90,62,121,134,114,190,193,232,278,324,327,330,42,409,271,304,356
11,Magetan,193,211,216,144,114,113,201,190,24,34,0,53,131,102,74,133,146,106,202,205,244,290,336,339,342,433,221,383,316,368
12,Ponorogo,198,216,221,148,119,139,206,201,29,61,53,0,78,115,79,84,117,52,195,210,249,211,341,344,347,438,226,283,321,373
13,Pacitan,276,294,298,227,197,217,284,179,107,139,131,78,0,180,157,149,182,117,270,213,352,276,390,421,358,462,364,366,399,451
14,Kediri,123,141,146,47,44,129,131,126,78,90,102,115,180,0,28,31,44,63,100,156,194,217,386,289,299,383,151,213,246,296
15,Nganjuk,119,137,142,70,40,125,127,122,50,62,74,79,157,28,0,59,72,96,128,131,170,216,282,285,268,359,147,209,242,294
16,Tulungagung,154,172,177,105,75,160,162,157,109,121,132,84,149,31,59,0,33,32,111,66,205,205,297,300,303,394,182,244,277,329
17,Blitir,167,185,144,118,86,173,175,170,122,134,146,117,182,44,72,33,0,64,78,133,172,172,264,267,270,361,195,257,290,342
18,Trenggalek ,196,204,206,137,107,192,194,189,82,144,106,52,117,63,90,32,64,0,142,197,236,236,328,331,334,425,214,276,359,361
19,Malang,89,107,66,89,119,197,134,192,178,190,202,195,290,100,26,111,78,142,0,55,94,117,194,189,192,259,117,179,212,264
20,Pasuruan ,60,78,37,61,91,168,105,163,181,193,205,210,313,155,131,166,133,197,55,0,39,85,131,134,137,228,86,150,183,235
21,Probolinggo,90,117,76,100,130,207,144,202,220,132,244,240,352,194,170,205,172,236,94,39,0,46,92,95,96,189,127,189,222,274
22,Lumajang,145,163,122,149,176,253,190,248,266,276,290,221,276,217,216,205,172,236,117,85,46,0,105,140,172,177,173,235,268,320
23,Bondowoso,191,209,168,192,222,299,236,294,312,324,336,341,390,386,362,297,264,328,186,131,92,105,0,35,32,126,219,281,314,396
24,Situbondo,194,212,171,195,225,302,239,297,345,327,339,344,421,289,208,300,267,331,167,134,95,140,35,0,67,94,222,284,317,389
25,Jember,197,215,174,196,228,305,242,300,318,330,342,347,358,292,268,303,270,334,192,137,96,72,32,97,0,105,225,287,320,372
26,Banyuwangi,288,306,265,289,319,396,333,391,407,421,433,436,462,383,389,394,361,425,239,228,189,177,136,94,105,0,316,378,411,463
27,Bangkalan,28,46,51,77,107,136,73,131,197,209,221,226,304,151,147,182,195,214,117,86,127,173,219,222,225,316,0,62,96,147
28,Sampang,90,108,113,139,169,196,135,193,259,271,283,288,366,213,209,144,257,276,179,150,189,235,281,294,287,375,62,0,33,85
29,Pamekasan,123,141,145,172,202,231,168,226,292,304,316,321,399,246,242,277,290,309,212,183,222,268,314,317,320,411,95,33,0,52
30,Sumenep,175,193,196,224,254,283,220,278,344,356,368,373,451,286,294,329,342,361,264,235,274,320,396,369,372,463,147,85,52,0
```

2.3 Perancangan Algoritma

2.3.1 Perancangan Algoritma BFS

Breadth First Search (BFS) adalah algoritma penelusuran *graph* yang arah penelusurannya mendahulukan ke arah lebar graf tersebut (Liman, 2018). BFS menggunakan *queue* untuk menyimpan dan mendapatkan vertex selanjutnya untuk kemudian dilakukan pencarian. *Queue* menggunakan prinsip FIFO (*First In, First Out*), yang berarti elemen yang pertama kali dimasukkan ke dalam *queue* akan diproses terlebih dahulu dibandingkan node lainnya, sehingga node yang pertama masuk tersebut menjadi yang pertama kali keluar.

Kompleksitas dari BFS sendiri adalah $O(V+E)$ dimana V merupakan jumlah *vertex* dan E merupakan jumlah *edge*, sehingga kompleksitas kemungkinan terburuk dari algoritma BFS adalah $O(n^2)$ dimana setiap *vertex* terhubung dengan semua *vertex* yang ada (Liman, 2018). Hal ini terjadi karena BFS harus mengeksplorasi setiap simpul dan sisi yang ada. Dalam aplikasi praktis, BFS lebih efisien untuk graf yang tidak terlalu padat dan untuk pencarian jalur terpendek dalam graf tanpa bobot, namun kurang cocok untuk masalah optimasi seperti Traveling Salesman Problem (TSP), yang membutuhkan algoritma lebih efisien dalam mencari solusi optimal.



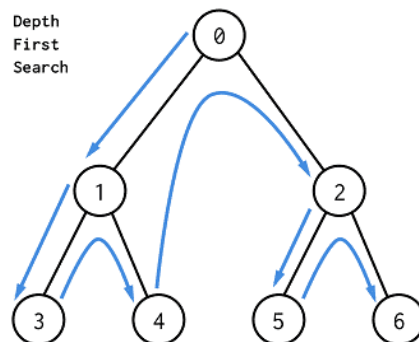
Sumber: <https://www.freelancinggig.com/blog/wp-content/uploads/2019/02/BFS-and-DFS-Algorithms.png>

Dalam Gambar diatas node yang pertama kali ditemukan akan diproses lebih dulu, kemudian mengunjungi setiap node di level tersebut sebelum bergerak ke level berikutnya. Berikut adalah alur pencarian BFS diatas:

1. Mulai dari node 0.
2. Mengunjungi semua tetangga langsung node 0: yaitu 1 dan 2.
3. Dari node 1, mengunjungi semua tetangganya: yaitu 3 dan 4.
4. Dari node 2, mengunjungi semua tetangganya: yaitu 5 dan 6.
5. Urutan traversal BFS: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$.

2.3.2 Perancangan Algoritma DFS

Depth First Search (DFS) merupakan sebuah pencarian *uninformed* yang cara kerjanya dengan memperluas simpul pada anak pertama dari pohon pencarian yang muncul, dengan demikian akan lebih dalam dan lebih sama node tujuan yang ditemukan atau hingga ditemukannya node yang tidak beranak, lalu dilanjutkan pencarian backtracks, dan kembali menuju node terbaru yang belum selesai dijelajah (Alderio & Sari, 2023). Pada implementasinya, DFS menggunakan struktur data tumpukan (*stack*) untuk menyimpan node yang sedang diproses, sehingga memungkinkan algoritma untuk melanjutkan pencarian ke dalam cabang yang lebih dalam terlebih dahulu. *Stack* menggunakan prinsip FILO (*First In, Last Out*), yang berarti elemen yang pertama kali dimasukkan adalah yang terakhir keluar. Dalam konteks DFS, node yang pertama kali dimasukkan ke dalam *stack* akan menjadi yang terakhir diproses ketika DFS kembali melakukan eksplorasi melalui pencarian *backtrack*. Pendekatan ini dapat mengakibatkan pemrosesan cabang-cabang yang lebih dalam sebelum akhirnya mengeksplorasi cabang yang lebih dangkal.



Sumber: <https://www.freelancinggig.com/blog/wp-content/uploads/2019/02/BFS-and-DFS-Algorithms.png>

Algoritma akan menjelajah hingga ke kedalaman maksimum terlebih dahulu sebelum kembali ke node sebelumnya untuk mengeksplorasi cabang lainnya.

1. Mulai dari node 0.
2. Kunjungi tetangga pertama node 0 : yaitu 1.
3. Dari node 1, kunjungi tetangga pertama yang belum dikunjungi: yaitu 3.
4. Setelah mencapai node 3, kembali ke node 1, lalu ke node 4.
5. Kembali ke node 0, lalu kunjungi tetangga lainnya: yaitu 2.
6. Dari node 2, kunjungi tetangga pertama yang belum dikunjungi: yaitu 5.
7. Dari node 5, kunjungi tetangga berikutnya: yaitu 6.
8. Urutan traversal DFS: $0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 6$.

2.4 Implementasi Algoritma

2.4.1 Implementasi Algoritma BFS

Algoritma *Breadth-First Search* (BFS) dirancang untuk menjelajahi graf dengan pendekatan level per level, mulai dari simpul awal dan mengunjungi semua simpul tetangga sebelum melanjutkan ke simpul yang lebih jauh. Dalam konteks Travelling Salesman Problem (TSP), BFS digunakan untuk mengeksplorasi rute yang menghubungkan semua kota, memastikan bahwa setiap kota dikunjungi tepat satu kali sebelum kembali ke kota asal. Berikut adalah implementasi *Python* untuk menyelesaikan masalah TSP dengan menggunakan algoritma BFS:

```
def bfs(adj, start_city):
    cities = list(adj.keys())
    s = cities.index(start_city)
    visited = [False] * len(adj)
    path = []
    min_path = []
    min_cost = [float('inf')]
    q = deque([s])
    visited[s] = True

    while q:
        curr = q.popleft()
        current_city = cities[curr]
        path.append(current_city)

        if len(path) == len(adj):
            path_cost = path_cost_function(path, adj, start_city)
            if path_cost < min_cost[0]:
                min_cost[0] = path_cost
                min_path[:] = path[:]

        for neighbor in adj[current_city]:
            neighbor_index = cities.index(neighbor)
            if not visited[neighbor_index]:
                visited[neighbor_index] = True
                q.append(neighbor_index)

    # Menghitung penggunaan memori
    memory_usage = sys.getsizeof(visited) + sys.getsizeof(path) + sys.getsizeof(q)
    return min_path, min_cost[0], memory_usage
```

2.4.2 Implementasi Algoritma DFS

Algoritma *Depth-First Search* (DFS), berbeda dengan BFS, mengeksplorasi secara mendalam dengan memilih simpul yang belum dikunjungi dan menelusuri cabang yang lebih jauh sampai tidak ada pilihan, sebelum kembali dan mencoba jalur alternatif. Dalam perancangan TSP, DFS digunakan untuk mencari rute dengan mengeksplorasi setiap kemungkinan jalur dari satu kota ke kota lainnya secara berurutan. Berikut adalah implementasi *Python* untuk menyelesaikan masalah TSP dengan menggunakan algoritma DFS:

```
def dfs_rec(adj, visited, current_city, path, min_path, min_cost, start_city):
    visited[current_city] = True
    path.append(current_city)

    if len(path) == len(adj):
        path_cost = path_cost_function(path, adj, start_city)
        if path_cost < min_cost[0]:
            min_cost[0] = path_cost
            min_path[:] = path[:]

    for neighbor in adj[current_city]:
        if not visited[neighbor]:
            dfs_rec(adj, visited, neighbor, path, min_path, min_cost, start_city)

    visited[current_city] = False
    path.pop()

def dfs(adj, start_city):
    visited = {city: False for city in adj}
    path = []
    min_path = []
    min_cost = [float('inf')]
    dfs_rec(adj, visited, start_city, path, min_path, min_cost, start_city)

    # Menghitung penggunaan memori
    memory_usage = sys.getsizeof(visited) + sys.getsizeof(path)
    return min_path, min_cost[0], memory_usage
```

BAB III

HASIL DAN ANALISIS

3.1 Hasil Deployment Proyek

Kami mendemonstrasikan hasil proyek menggunakan local web streamlit agar lebih mudah dipahami. Berikut adalah langkah-langkah yang harus dilakukan user untuk menjalankan program:

- Menentukan minimal 10 kota yang akan digunakan dalam perhitungan untuk mencari solusi dari masalah Traveling Salesman Problem (TSP). Pemilihan kota-kota ini penting untuk memastikan keberagaman dalam rute yang akan dieksplorasi.
- Menetapkan satu kota sebagai kota asal atau titik awal. Kota ini akan menjadi titik acuan untuk memulai perjalanan, dan semua perhitungan rute akan berpusat dari kota asal ini.
- Memilih algoritma yang sesuai untuk menyelesaikan masalah TSP. Algoritma yang tersedia adalah *Breadth-First Search* (BFS) dan *Depth-First Search* (DFS).

Berikut ini adalah beberapa contoh hasil deployment dan implementasi algoritma BFS dan DFS pada TSP:

1. User harus memilih 10 kota secara bebas, contohnya **Lamongan, Tuban, Pacitan, Tulungagung, Lumajang, Bangkalan, Ngawi, Blitar, Madiun, dan Surabaya**. Kemudian, menentukan kota asal, pada contoh ini menggunakan Kota **Lamongan**.

Optimal Path Finder

Pilih kota (minimal 10 kota):

Surabaya ×

Lamongan ×

Tuban ×

Blitar ×

Ngawi ×



Lumajang ×

Tulungagung ×

Pacitan ×


Madiun ×

Bangkalan ×

Kota yang dipilih: ['Surabaya', 'Lamongan', 'Tuban', 'Blitar', 'Ngawi', 'Lumajang', 'Tulungagung', 'Pacitan', 'Madiun', 'Bangkalan']

Masukkan kota awal:

Lamongan 

Kemudian user harus menentukan algoritma yang akan dipilih, kemudian output akan muncul. Berikut adalah output untuk input user diatas:

BFS

BFS

Jalur optimal:

Lamongan → Surabaya → Tuban → Madiun → Ngawi → Pacitan → Tulungagung → Blitar → Lumajang → Bangkalan → Lamongan

Total jarak perjalanan: 1091 Km

Waktu eksekusi: 0.0000 detik

Penggunaan memori (kompleksitas ruang): 944 bytes

Operasi yang dilakukan: 19

Jumah Node yang dikunjungi: 10

Jumlah edge yang di kunjungi: 10

DFS

DFS

Jalur optimal:

Lamongan → Surabaya → Bangkalan → Lumajang → Blitar → Tulungagung → Ngawi → Madiun → Pacitan → Tuban → Lamongan

Total jarak perjalanan: 948 Km

Waktu eksekusi: 2.1035 detik

Penggunaan memori (kompleksitas ruang): 416 bytes

Operasi yang dilakukan: 9240570

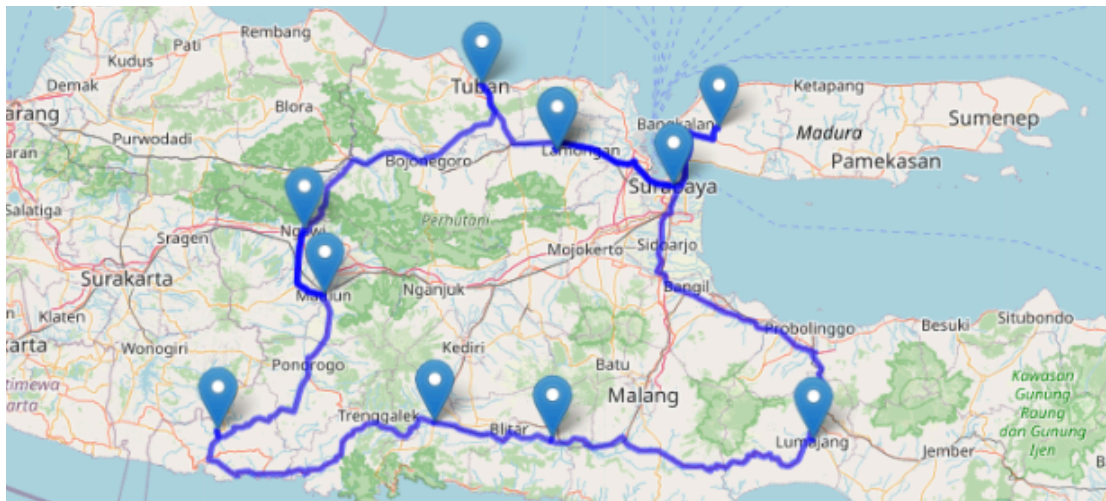
Jumah Node yang dikunjungi: 10850519

Jumlah edge yang di kunjungi: 986410

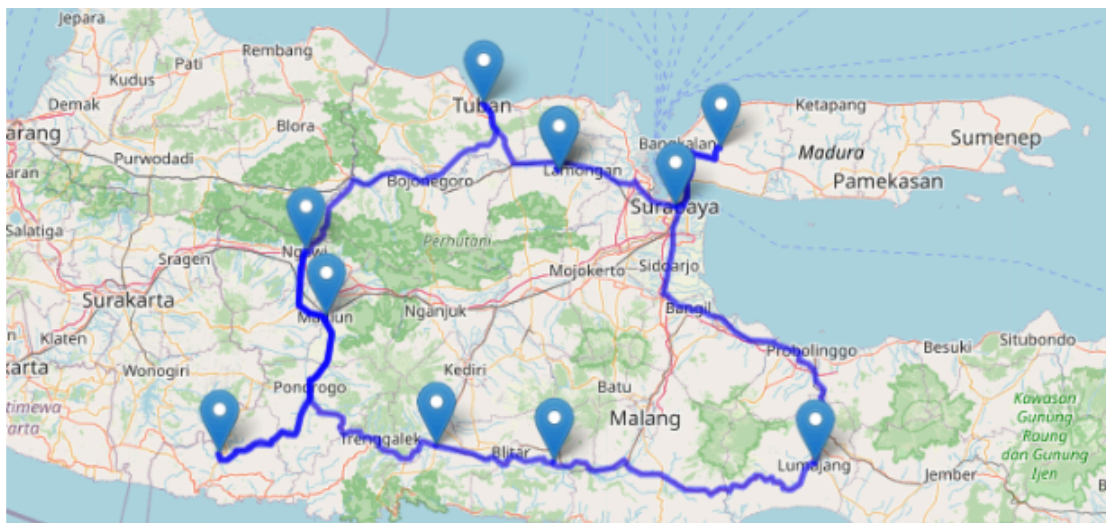
Dari hasil diatas dapat disimpulkan bahwa BFS menemukan solusi dalam waktu yang sangat cepat dengan penggunaan memori yang lebih besar dibanding DFS. Namun, jarak perjalanan **1091 km** menunjukkan bahwa BFS tidak menghasilkan solusi yang optimal dibandingkan dengan solusi dari DFS dengan jarak total **948 km**. Namun, membutuhkan waktu eksekusi yang jauh lebih lama (**2.0157 detik**) dan operasi yang sangat banyak untuk mencapai solusi ini.

Hasil pencarian rute terbaik kemudian ditampilkan menggunakan Map agar lebih mudah dipahami oleh user.

Peta rute Algoritma BFS



Peta rute Algoritma DFS



2. Untuk mencari tahu seberapa berpengaruh kota asal terhadap keseluruhan hasil, 10 kota sebelumnya, yakni **Lamongan, Tuban, Pacitan, Tulungagung, Lumajang, Bangkalan, Ngawi, Blitar, Madiun, dan Surabaya** tetap gunakan, namun pada contoh ini menggunakan kota asal **Lumajang**. Berikut hasilnya:

BFS

Pilih algoritma:

BFS

Jalur optimal:

Lumajang → Surabaya → Lamongan → Tuban → Madiun → Ngawi → Pacitan → Tulungagung → Blitar → Bangkalan → Lumajang

Total jarak perjalanan: 1141 Km

Waktu eksekusi: 0.0000 detik

Penggunaan memori (kompleksitas ruang): 944 bytes

Operasi yang dilakukan: 19

Jumah Node yang dikunjungi: 10

Jumlah edge yang di kunjungi: 10

DFS

Pilih algoritma:

DFS

Jalur optimal:

Lumajang → Blitar → Tulungagung → Ngawi → Madiun → Pacitan → Tuban → Lamongan → Surabaya → Bangkalan → Lumajang

Total jarak perjalanan: 948 Km

Waktu eksekusi: 2.0539 detik

Penggunaan memori (kompleksitas ruang): 416 bytes

Operasi yang dilakukan: 9240570

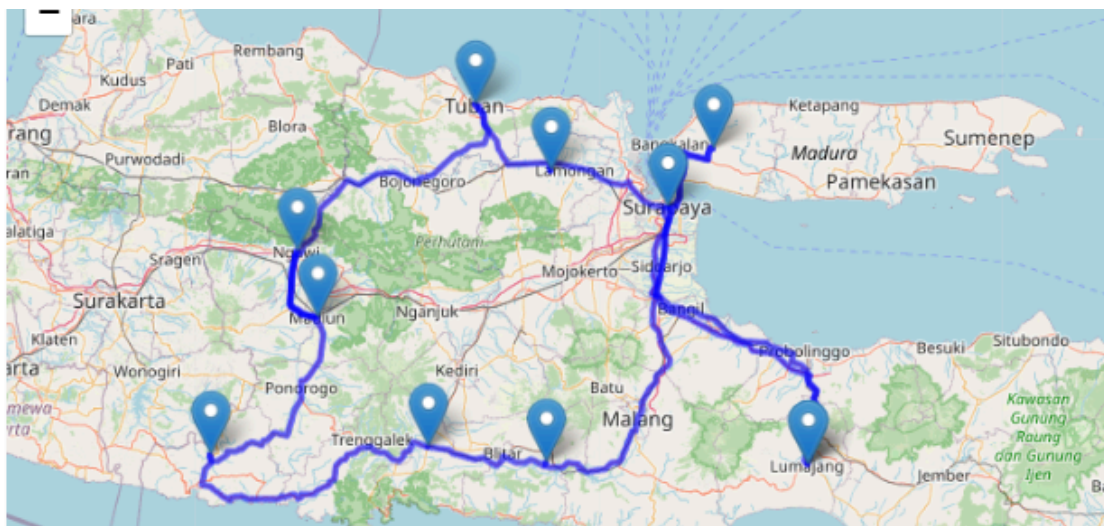
Jumah Node yang dikunjungi: 10850519

Jumlah edge yang di kunjungi: 986410

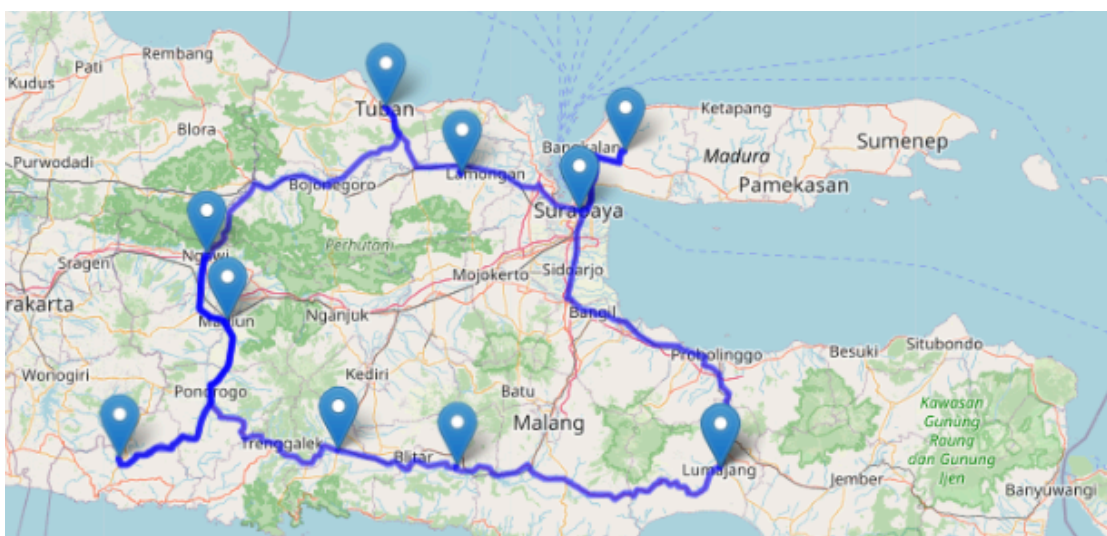
Dengan mengubah titik awal menjadi kota Lumajang output yang dihasilkan oleh program berbeda, yang berarti **Titik awal memengaruhi jalur optimal dan total jarak perjalanan**, tetapi tidak memengaruhi jumlah operasi, waktu eksekusi, atau kompleksitas ruang dan memori. Performa algoritma tidak berubah, DFS tetap menghasilkan jalur yang lebih pendek dibandingkan BFS, tetapi dengan waktu dan jumlah operasinya jauh lebih besar daripada BFS.

Hasil peta rute yang dihasilkan oleh algoritma BFS berbeda dengan sebelumnya, tetapi hasil peta rute algoritma DFS sama ketika menggunakan 10 kota yang sama dengan titik awal yang berbeda.

Peta Rute BFS



Peta Rute DFS



3. Untuk mendukung analisis perbandingan, dilakukan pemilihan 10 kota lagi secara bebas, contohnya **Lamongan, Tuban, Pacitan, Tulungagung, Lumajang, Bangkalan, Ngawi, Blitar, Madiun, dan Surabaya**, dengan kota asal **Blitar**.

Optimal Path Finder

Pilih kota (minimal 10 kota):

Blitar ×

Ngawi ×

Pacitan ×

Madiun ×

Ponorogo ×

Malang ×

×

▼

Surabaya ×

Bojonegoro ×

Lamongan ×

Tulungagung ×

Kota yang dipilih: ['Blitar', 'Ngawi', 'Pacitan', 'Madiun', 'Ponorogo', 'Malang', 'Surabaya', 'Bojonegoro', 'Lamongan', 'Tulungagung']

Masukkan kota awal:

Blitar

▼

Hasilnya:

BFS

Pilih algoritma:

BFS

▼

Jalur optimal:

Blitar → Surabaya → Bojonegoro → Lamongan → Madiun → Ngawi → Ponorogo → Pacitan → Tulungagung → Malang → Blitar

Total jarak perjalanan: 1024 Km

Waktu eksekusi: 0.0000 detik

Penggunaan memori (kompleksitas ruang): 944 bytes

Operasi yang dilakukan: 19

Jumah Node yang dikunjungi: 10

Jumlah edge yang di kunjungi: 10

Peta Rute:

DFS

Pilih algoritma:

DFS

Jalur optimal:

Blitar → Tulungagung → Pacitan → Ponorogo → Madiun → Ngawi → Bojonegoro → Lamongan → Surabaya → Malang → Blitar

Total jarak perjalanan: 674 Km

Waktu eksekusi: 2.0663 detik

Penggunaan memori (kompleksitas ruang): 416 bytes

Operasi yang dilakukan: 9240570

Jumlah Node yang dikunjungi: 10850520

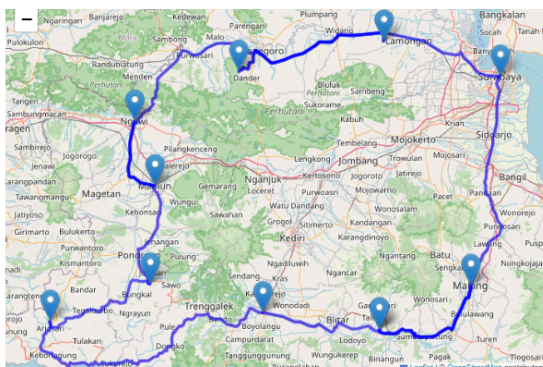
Jumlah edge yang di kunjungi: 986410

Peta Rute:

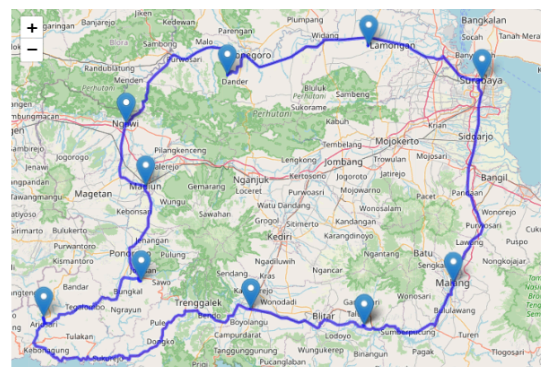
Dari hasil diatas dapat dibuktikan bahwa DFS lebih baik dalam menemukan solusi dengan jarak perjalanan yang lebih pendek, tetapi memakan waktu dan sumber daya komputasi lebih besar dibanding BFS.

Peta rute yang dihasilkan oleh kedua algoritma juga berbeda pada contoh kali ini, berikut hasilnya.

Peta Rute BFS



Peta Rute DFS



3.2 Analisis Hasil Algoritma

3.2.1 Analisis Hasil Algoritma BFS

Berikut adalah analisis hasil dari ketiga contoh implementasi algoritma BFS yang diberikan:

- **Jarak:** Algoritma BFS cenderung menghasilkan jarak yang lebih besar dibandingkan dengan DFS. Hal ini disebabkan oleh cara BFS yang menjelajahi graf secara level-order, yaitu mengunjungi semua simpul pada satu level sebelum melanjutkan ke level berikutnya. Pendekatan ini dapat mengarah pada jalur yang tidak optimal, terutama dalam graf yang memiliki banyak cabang atau simpul yang saling terhubung. Sebaliknya, Algoritma DFS, yang menjelajahi satu cabang hingga kedalaman maksimum sebelum kembali, sering kali menemukan jalur yang lebih pendek dalam situasi tertentu, terutama ketika solusi terletak dekat dengan titik awal.
- **Waktu Eksekusi:** Waktu eksekusi BFS umumnya lebih cepat dibandingkan dengan DFS. Pada proyek kami ini, algoritma BFS menunjukkan waktu eksekusi yang sangat efisien, yakni 0 detik yang bisa dilihat dari ketiga contoh implementasi di atas. Ini terjadi karena BFS melakukan pencarian secara luas (*breadth-first*), sehingga dapat menemukan solusi lebih cepat tanpa perlu menggali terlalu dalam seperti pada DFS. Algoritma DFS mungkin lebih cepat dalam situasi di mana solusi terletak dekat dengan simpul awal, tetapi dalam kasus umum, BFS lebih unggul dalam hal kecepatan eksplorasi.
- **Penggunaan Memori:** Penggunaan memori adalah salah satu kelemahan utama dari algoritma BFS. Algoritma ini membutuhkan lebih banyak ruang untuk menyimpan informasi tentang semua simpul yang telah dieksplorasi dan antrian dari simpul-simpul yang akan dieksplorasi. Dalam implementasinya, BFS menyimpan seluruh level cabang yang telah diperiksa dalam antrian, sehingga dapat mengakibatkan penggunaan memori yang signifikan, terutama pada graf besar atau kompleks.

3.2.2 Analisis Hasil Algoritma DFS

Berikut adalah analisis hasil dari ketiga contoh implementasi algoritma DFS yang diberikan:

- **Jarak:** Algoritma DFS cenderung lebih kecil daripada algoritma BFS sebelumnya. Dalam hal jarak perjalanan, algoritma DFS sering kali dapat menemukan jalur yang lebih pendek dibandingkan dengan BFS. Hal ini disebabkan oleh cara kerja DFS yang lebih terfokus pada eksplorasi mendalam. Algoritma ini menjelajahi satu cabang hingga mencapai simpul terdalam sebelum kembali dan mencoba cabang lain. Pendekatan ini memungkinkan DFS untuk menemukan solusi yang lebih dekat dengan titik awal, terutama ketika solusi terletak di cabang yang dalam atau di sisi tertentu dari graf. Namun, perlu dicatat bahwa meskipun DFS bisa menemukan jalur yang lebih pendek dalam beberapa kasus, tidak ada jaminan bahwa jalur tersebut selalu optimal.
- **Waktu Eksekusi:** Meskipun DFS dapat menemukan jarak yang lebih pendek, waktu eksekusinya sering kali lebih lama dibandingkan dengan BFS. Proses pencarian DFS dilakukan dengan menjelajahi jalur satu per satu secara mendalam, yang dapat menyebabkan algoritma terjebak dalam cabang yang panjang tanpa menemukan solusi. Dalam graf yang memiliki banyak cabang atau simpul, ini dapat mengakibatkan waktu pencarian yang signifikan, terutama jika jalur optimal terletak jauh di dalam cabang-cabang tersebut. Oleh karena itu, meskipun DFS efisien dalam menemukan jalur yang lebih pendek, waktu eksekusinya dapat meningkat secara drastis dalam situasi tertentu.
- **Penggunaan Memori:** Salah satu keunggulan utama dari algoritma DFS adalah penggunaan memori yang relatif rendah dibandingkan dengan BFS. DFS tidak menyimpan semua simpul pada level tertentu seperti BFS dan sebaliknya, ia menggunakan struktur data *stack* untuk menyimpan informasi tentang jalur saat ini. Ini berarti bahwa hanya simpul-simpul di sepanjang jalur pencarian yang perlu disimpan dalam memori pada satu waktu. Akibatnya, penggunaan memori DFS biasanya lebih efisien dan cocok untuk graf besar atau ketika memori terbatas. Namun, perlu dicatat bahwa dalam kasus graf yang sangat dalam atau kompleks, penggunaan *stack* dapat menyebabkan masalah seperti *stack overflow* jika kedalaman pencarian terlalu besar.

3.3 Perbandingan Kedua Algoritma

Aspek	Breadth-First Search (BFS)	Depth-First Search (DFS)
Metode Eksplorasi	Menjelajahi graf secara <i>level-order</i> (luas).	Menjelajahi graf secara mendalam (kedalaman).
Jarak	Cenderung menghasilkan jarak yang lebih besar; tidak selalu menemukan jalur terpendek.	Sering kali menemukan jalur yang lebih pendek, terutama jika solusi dekat dengan titik awal.
Waktu Eksekusi	Umumnya lebih cepat dalam menemukan solusi pada graf sederhana; efisien dalam eksplorasi luas.	Waktu eksekusi bisa lebih lama, terutama jika solusi terletak jauh di cabang yang dalam.
Penggunaan Memori	Memerlukan lebih banyak memori karena menyimpan semua simpul pada level yang sama dalam antrian.	Memerlukan memori yang relatif rendah; hanya menyimpan jalur saat ini dalam stack.
Struktur Data	Menggunakan <i>queue</i> untuk menyimpan simpul yang akan dieksplorasi.	Menggunakan <i>stack</i> untuk menyimpan jalur pencarian saat ini.
Kelebihan	<ul style="list-style-type: none">• Menemukan jalur terpendek dalam graf tidak berbobot.• Lebih baik untuk graf yang lebar dan dangkal.	<ul style="list-style-type: none">• Penggunaan memori yang efisien.• Lebih baik untuk graf yang dalam dan memiliki solusi di node yang dalam.
Kekurangan	<ul style="list-style-type: none">• Penggunaan memori tinggi dapat menjadi masalah pada graf besar.• Tidak efisien untuk graf dengan kedalaman besar.	<ul style="list-style-type: none">• Waktu eksekusi bisa sangat lama jika terjebak di cabang panjang.• Tidak menjamin menemukan jalur terpendek.

BAB IV

KESIMPULAN

Dalam proyek ini, kami menganalisis dan membandingkan algoritma *Breadth-First Search (BFS)* dan *Depth-First Search (DFS)* dalam konteks penyelesaian *Traveling Salesman Problem (TSP)*, yang merupakan masalah optimasi klasik yang bertujuan untuk menemukan jalur terpendek yang mengunjungi setiap simpul dalam graf tepat satu kali dan kembali ke simpul awal. Hasil dari proyek kami menunjukkan bahwa DFS cenderung menghasilkan jarak yang lebih pendek karena fokusnya pada eksplorasi mendalam yang memungkinkan algoritma ini untuk menemukan solusi yang lebih dekat dengan titik awal. Namun, waktu eksekusi dari algoritma DFS sering kali lebih lama dibandingkan dengan BFS, terutama ketika solusi terletak jauh di cabang yang dalam, karena DFS menjelajahi jalur satu per satu secara mendalam. Di sisi lain, BFS lebih efisien dalam eksplorasi luas dan dapat menemukan solusi lebih cepat pada graf sederhana, meskipun ia tidak selalu menghasilkan jalur terpendek. Dari segi penggunaan memori, DFS menunjukkan efisiensi yang lebih baik dengan penggunaan memori yang relatif rendah karena hanya menyimpan jalur saat ini dalam *stack*, sementara BFS memerlukan lebih banyak memori untuk menyimpan semua simpul pada level yang sama dalam *queue*. Oleh karena itu, untuk graf kecil atau sedang di mana solusi terletak dekat dengan titik awal, DFS dapat menjadi pilihan yang baik karena kemampuannya menemukan jalur yang lebih pendek. Sebaliknya, BFS lebih sesuai untuk graf besar atau kompleks di mana efisiensi waktu menjadi prioritas utama. Dengan demikian, pemilihan algoritma pencarian yang tepat harus didasarkan pada karakteristik spesifik dari masalah TSP yang dihadapi serta kebutuhan akan efisiensi waktu dan penggunaan memori, memberikan informasi berharga untuk pengembangan metode penyelesaian TSP di masa depan.

DAFTAR PUSTAKA

- Alderio, R., & Sari, R. P. (2023). Perbandingan pendekatan Breadth First Search, Depth First Search dan Best First Search pada pencarian rute menggunakan metode branch and bound. *Jurnal Serambi Engineering*, 8(3). <https://doi.org/10.32672/JSE.V8I3.6525>
- Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2011). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.
- Badan Pusat Statistik Provinsi Jawa Timur. (n.d.). Jarak antar kota di Jawa Timur - tabel statistik. Retrieved December 13, 2024, from <https://jatim.bps.go.id/id/statistics-table/1/MTMjMQ==/jarak-antar-kota-di-jawa-timur.html>
- Liman, C. (2018). Desain dan analisis algoritma Breadth First Search sebagai metode penyelesaian permasalahan pencarian cycle graf: Studi kasus SPOJ 28409 - Ada and Cycle.
- Sutoyo, I. (2018). Penerapan algoritma nearest neighbour untuk menyelesaikan travelling salesman problem. *Jurnal Khatulistiwa Informatika*, 20(1), 101–106. <https://doi.org/10.31294/P.V20I1.3155>
- GeeksforGeeks. (2025, January 2). Breadth First Search or BFS for a graph. Retrieved from <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph>
- GeeksforGeeks. (2025, January 4). Depth First Search or DFS for a graph. Retrieved from <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph>

KONTRIBUSI ANGGOTA

Nama	Kontribusi
Ivanagita Mayang Palupi 23031554009	Perancangan Algoritma BFS, Menyusun Laporan, PPT.
Sintiya Risa Miftaql Nikmah 23031554204	Dataset, Perancangan Algoritma BFS, Menyusun Laporan, PPT.
Theopan Gerard Nainggolan 23031554227	Perancangan Algoritma DFS, Deployment Proyek menggunakan Streamlit, Menyusun Laporan.