MODUL 9 TUGAS BESAR

Firdaus (13221013)
Reyhan Ghifari Tanjung (13221014)
Galih Siswandi (13221015)
Azita Nadiya Noorazlina (13221017)
Asisten: Muhammad Daffa Rasyid
Kelompok: B1
EL2208-Praktikum Pemecahan Masalah dengan C
Laboratorium Dasar Teknik Elektro - Sekolah Teknik Elektro dan Informatika ITB

Abstrak

Tugas besar Modul 9 ini bertujuan untuk melatih penerapan Bahasa C pada permasalahan kehidupan nyata. Permasalahan yang didapatkan oleh kelompok kami adalah permasalahan yang berkaitan dengan teori Travelling Salesman Problem dimana kelompok dituntut untuk mampu membuat siklik pelayaran dari pelabuhan-pelabuhan yang disediakan di data yang telah ada dengan jarak optimal atau cost seminimal mungkin. Spesifikasi lainnya dari tugas ini adalah kelompok juga harus mampu memberikan data-data Pelabuhan yang tidak dapat dikunjungi dalam siklik yang telah dirancang karena bernilai >2500 km dari seluruh Pelabuhan yang ada. Pemecahan masalah-masalah dari tugas ini adalah dengan pertama mengkonversi lintang-bujur antara dua Pelabuhan ke nilai jarak menggunakan rumus Haversine, kemudian jarak-jarak hasil perhitungan tersebut yang nantinya akan digunakan untuk memecahkan masalah TSP yang ada. Algoritma yang diterapkan untuk menyelesaikan TSP itu sendiri adalah dynamic programming yang menggunakan branch-and-bound method. Hasilnya didapatkan rute pelayaran yang tertera di laporan ini dan data-data Pelabuhan yang tidak dapat dikunjungi dalam siklus pelayaran ini. Dengan demikian, dapat diambil kesimpulan bahwa penggunaan Haversine formula untuk mengkonversi jarak dari dua data lintang-bujur serta menggunakan dynamic programming untuk menyelesaikan TSO sudah efektif dan mampu diterapkan di Bahasa C untuk menyelesaikan permasalahan yang ada di kehidupan nyata.

Kata kunci: Travelling Salesman Problem, Haversine, Pelabuhan, Dynamic Programming, Jarak.

1. PENDAHULUAN

Modul 9 – Tugas Besar adalah modul yang bertujuan untuk menyelesaikan masalah pemrograman secara berkelompok. Tujuan lain dari pengerjaan modul ini adalah untuk mendalami penerapan Bahasa C pada kehidupan sehari-hari serta untuk mendalami materi pemrograman algoritma *Tree and Graph* dari Modul 8. Pada modul ini, terdapat dua soal mengenai *Travelling Salesman Problem* dan *Convex Hull Algorithm*. Kami mendapat bagian untuk mengerjakan soal nomor 1 (*Travelling Salesman Problem*).

Secara singkat, spesifikasi soal nomor 1 adalah membuat program yang dapat mencari rute terpendek yang mengunjungi seluruh pelabuhan lalu kembali lagi ke pelabuhan awal. Selain itu, program juga harus dapat menampilkan total jarak yang ditempuh kapal. Namun, karena jarak tempuh terjauh kapal adalah 2500 km, maka program juga perlu menampilkan pelabuhan yang tidak dapat dikunjungi. Program menerima masukan file "pelabuhan.csv" berupa daftar pelabuhan-pelabuhan yang akan dikunjungi. Hasil yang diharapkan nantinya adalah rute pelayaran paling optimal serta daftar Pelabuhan yang tidak dapat dikunjungi karena berjarak >2500 km dari seluruh Pelabuhan lain.

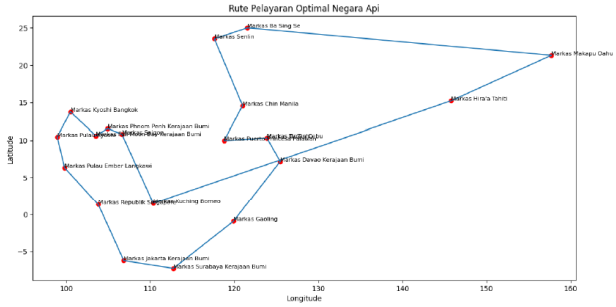
2. STUDI PUSTAKA

Untuk mendukung pengerjaan serta penyelesaian dari permasalahan yang diberikan di modul ini, maka dibutuhkan dasar-dasar teori pendukung serta modul referensi acuan untuk mempermudah pemahaman anggota kelompok sehingga pemecahan masalah dapat dilakukan dengan mudah. Dengan mem-*breakdown* masalah-masalah dari soal yang ada, teori-teori yang diperlukan untuk menyelesaikan soal ini adalah sebagai berikut.

2.1 TRAVELLING SALESMAN PROBLEM

Travelling Salesman Problem (TSP) adalah permasalahan umum dalam optimasi kombinatorial dimana seorang salesman harus mengujungi sejumlah N kota, disyartakan setiap kota hanya dikunjungi sekali. Salesman ini harus memilih rute sehingga jarak total yang dia tempuh minimum (Budi Santosa, 2017). Menurut Smith, dalam jurnal Utomo, dkk (2004) Traveling Salesman Problem (TSP) dapat dengan mudah diubah dalam bentuk network problem dengan formulasi yang serupa dengan model rute terpendek. Konsumen yang dikunjungi diidentifikasikan sebagai simpul-simpul (node)

dari jaringan. Sedangakan menurut Rabi', Persoalan Travelling Salesman (TSP) adalah persoalan optimasi yang dinyatakan sebagai mencari rute perjalanan termurah untuk mengunjungi node (konsumen), dimana setiap konsumen dikunjungi secara pasti satu kali. [1]



Gambar 2. 1. Ilustrasi TSP pelayaran negara api pada soal

Bahwa penentuan rute perjalanan merupakan salah satu permasalahan yang sering dihadapi dalam kehidupan sehari - hari. Salah satu contoh yaitu rute manakah yang memiliki biaya paling murah untuk dilalui seorang salesman ketika harus mengunjungi sejumlah daerah. Tiap daerah tersebut harus dikunjungi tepat satu kali kemudian kembali lagi ke tempat semula. Permasalahan tersebut dikenal sebagai Traveling Salesman Problem (TSP) yaitu mencari rute terpendek dengan syarat kendaraan berawal dan berakhir di depo yang sama dan setiap kota dikunjungi tepat satu kali. [1]

Berikut adalah aturan-aturan yang mengidentifikasikan bahwa permasalahan tersebut adalah TSP: [1]

1. Perjalanan dimulai dan diakhiri di kota yang sama sebagai kota asal sales.
2. Seluruh kota harus dikunjungi tanpa satupun kota yang terlewatkan.
3. Sales tidak boleh kembali ke kota asal sebelum seluruh kota terkunjungi.

2.2 TRAVELLING SALES PROBLEM ALGORITHM IN PROGRAMMING

Dalam menyelesaikan TSP dengan menggunakan bahasa pemrograman C, ada banyak jenis metode yang dapat digunakan untuk menyelesaikannya. Namun, umumnya digunakan dua metode yang paling sering digunakan. Kedua metode tersebut adalah **naïve solution/simple approach** yang menggunakan **brute-force** untuk menyelesaikannya dan **dynamic programming** yang dieksekusi dengan **branch-and-bound method** yang rekursif.

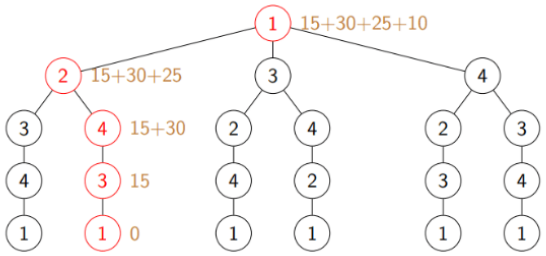
2.2.1 NAÏVE APPROACH

Naïve Approach dilakukan dengan Langkah-langkah sebagai berikut: [4]

1) Pertimbangkan kota 1 sebagai titik awal dan akhir.
2) Hitung (n-1)! permutasi tiap kota.
3) Hitung biaya setiap permutasi dan catat permutasi biaya minimum.
4) Kembalikan permutasi dengan biaya minimum.

Dengan cara ini *time complexity* program adalah **O(n!)** dan *auxiliary space*-nya adalah **O(n).**

2.2.2 DYNAMIC PROGRAMMING



Gambar 2. 2. Ilustrasi Branch-and Bound Method untuk TSP [5]

Cara kerja dari Dynamic Programming adalah berikut ini: [5]

Mari kita asumsikan S adalah himpunan bagian dari kota dan milik {1, 2, 3, …, n} di mana 1, 2, 3…n adalah kota-kota dan i, j adalah dua kota dalam himpunan bagian itu. Sekarang cost(i, S, j) didefinisikan sedemikian rupa sebagai panjang jalur terpendek yang mengunjungi simpul di S, yang tepat sekali dikunjungi dan memiliki titik awal dan akhir masing-masing sebagai i dan j.

Misalnya, biaya (1, {2, 3, 4}, 1) menunjukkan panjang jalur terpendek di mana:

1) Kota awal adalah 1
2) Kota 2, 3, dan 4 dikunjungi hanya sekali
3) Titik akhir adalah 1

Algoritma dynamic programming-nya adalah:

1) Tetapkan biaya(i, , i) = 0, artinya kita mulai dan berakhir di i, dan biayanya adalah 0.
2) Kapan |S| > 1, kita definisikan cost(i, S, 1) = ∝ dimana i !=1 . Karena awalnya kita tidak mengetahui biaya pasti untuk mencapai kota i ke kota 1 melalui kota lainnya.
3) Sekarang, kita harus mulai dari 1 dan menyelesaikan tur. Kita perlu memilih kota berikutnya sedemikian rupa-biaya(i,

S, j)=min biaya (i, S−{i}, j)+dist(i,j) di mana i∈S dan i≠j

Sehingga algoritmanya bekerja dengan cara:

1) Pertimbangkan perjalanan mulai dari kota 1, mengunjungi kota lain satu kali dan kembali ke kota 1.
2) S adalah himpunan bagian dari kota. Menurut algoritme ini, untuk semua |S| > 1, kita akan menetapkan biaya jarak (i, S, 1) = ∝. Di sini cost(i, S, j) berarti kita mulai dari kota i, mengunjungi kota-kota S sekali, dan sekarang kita berada di kota j. Ditetapkan biaya jalur ini sebagai tak terhingga karena belum diketahui jaraknya. Maka nilainya akan menjadi sebagai berikut: Biaya (2, {3, 4}, 1) = ∝ ; notasi menunjukkan kita mulai dari kota 2, melewati kota 3, 4, dan mencapai 1. Dan biaya jalurnya tidak terbatas. Demikian pula-biaya(3, {2, 4}, 1) = ∝, biaya(4, {2, 3}, 1) = ∝
3) Sekarang, untuk semua himpunan bagian dari S, kita perlu mencari yang berikut ini: **biaya(i, S, j)=min biaya (i, S−{i}, j)+dist(i,j), di mana j∈S dan i≠j**. Itu berarti jalur biaya minimum untuk mulai dari i, melewati subhimpunan kota satu kali, dan kembali ke kota j. Mengingat perjalanan dimulai di kota 1, biaya jalur optimal adalah = biaya(1, {kota lain}, 1).

Sekarang S = {1, 2, 3, 4}. Ada empat elemen. Oleh karena itu jumlah himpunan bagian adalah 2^4 atau 16. Subhimpunan tersebut adalah:

1) **|S| = Null:**
   {Φ}
2) **|S| = 1:**
   {{1}, {2}, {3}, {4}}
3) **|S| = 2:**
   {{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}}
4) **|S| = 3:**
   {{1, 2, 3}, {1, 2, 4}, {2, 3, 4}, {1, 3, 4}}
5) **|S| = 4:**
   {{1, 2, 3, 4}}

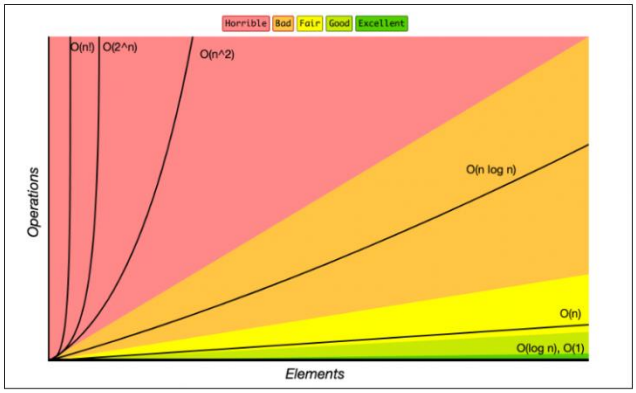
Karena kita mulai dari 1, kita dapat membuang himpunan bagian yang berisi kota 1. Kalkulasi algoritmnya adalah:

1) **|S| = Φ:**
   cost (2, Φ, 1) = dist(2, 1) = 10
   cost (3, Φ, 1) = dist(3, 1) = 15
   cost (4, Φ, 1) = dist(4, 1) = 20
2) **|S| = 1:**
   cost (2, {3}, 1) = dist(2, 3) + cost (3, Φ, 1) = 35+15 = 50
   cost (2, {4}, 1) = dist(2, 4) + cost (4, Φ, 1) = 25+20 = 45
   cost (3, {2}, 1) = dist(3, 2) + cost (2, Φ, 1) = 35+10 = 45
   cost (3, {4}, 1) = dist(3, 4) + cost (4, Φ, 1) = 30+20 = 50
   cost (4, {2}, 1) = dist(4, 2) + cost (2, Φ, 1) = 25+10 = 35
   cost (4, {3}, 1) = dist(4, 3) + cost (3, Φ, 1) = 30+15 = 45
3) **|S| = 2:**
   cost (2, {3, 4}, 1) = min [dist[2,3]+Cost(3,{4},1) = 35+50 = 85, dist[2,4]+Cost(4,{3},1) = 25+45 = 70] = 70
   cost (3, {2, 4}, 1) = min [dist[3,2]+Cost(2,{4},1) = 35+45 = 80, dist[3,4]+Cost(4,{2},1) = 30+35 = 65] = 65
   cost (4, {2, 3}, 1) = min [dist[4,2]+Cost(2,{3},1) = 25+50 = 75 dist[4,3]+Cost(3,{2},1) = 30+45 = 75] = 75
4) **|S| = 3:**
   cost (1, {2, 3, 4}, 1) = min [dist[1,2]+Cost(2,{3,4},1) = 10+70 = 80 dist[1,3]+Cost(3,{2,4},1) = 15+65 = 80 dist[1,4]+Cost(4,{2,3},1) = 20+75 = 95] = 80

Jadi solusi optimalnya adalah **1-2-4-3-1.**

Dengan cara ini *time complexity* program adalah **O(n^2*2^n)** dan *auxiliary space*-nya adalah **O(2^n).** [4]

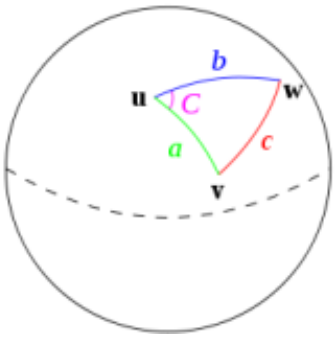
Pada grafik Big O complexity, dapat dilihat bahwa O(n!) lebih buruk implementasinya dari O(n^2) dan O(2^n).



Gambar 2. 3. Big O complexity Graph [6]

2.3 HAVERSINE FORMULA

Teorema Haversine Formula adalah sebuah persamaan yang penting dalam bidang navigasi, untuk mencari jarak busur antara dua titik pada bola dari *longitude* dan *latitude.* Ini merupakan bentuk persamaan khusus dari trigonometri bola, *law of haversines*, mencari hubungan sisi dan sudut pada garis segitiga dalam bidang bola. [2]



Gambar 2. 4. Segitiga bola diselesaikan dengan haversine formula

Hukum Haversine adalah sebuah persamaan yang digunakan berdasarkan bentuk bumi yang bulat (spherical earth) dengan menghilangkan faktor bahwa bumi itu sedikit elips (elipsodial factor). Ini merupakan kasus khusus dari formula umum dalam trigonometri bola, hukum haversine, yang berkaitan dengan sisi dan sudut segitiga bola. Dalam unit bola, sebuah "segitiga" pada permukaan bola didefinisikan sebagai lingkaran-lingkaran besar yang menghubungkan tiga poin u, v, dan w pada bola, Jika panjang dari ketiga sisi, a adalah (dari u ke v), b (dari u ke w), dan c (dari v ke w), dan sudut sudut yang berlawanan c adalah C. maka hukum haversine menjadi: [2]

Haversine (c) = haversine(a-b) + cos(a) cos(b) haversine (C).

Singkatnya, Haversine formula memberikan jarak lingkaran besar antara dua titik pada permukaan bola (bumi) berdasarkan bujur dan lintang dengan mengasumsikan jari-jari R 6.367, 45 km, dan lokasi dari 2 titik di koordinat bola (lintang dan bujur) masing-masing adalah lon1,lat1, dan lon2, lat2. Rumus Haversine dapat ditulis dengan persamaan sebagai berikut: [3]

$$X = (lon2 - lon1) * \cos\left(\frac{lat1 + lat2}{2}\right)$$
$$Y = (lat2 - lat1)$$
$$d = \sqrt{(x^2 + y^2)} * R$$

Dimana:
X = Longitide (Lintang)
Y = Lattitide (Bujur)
D = Jarak
R = Radius Bumi = 6371 km
1 derajat = 0,0174532925

3. METODOLOGI

Setelah memahami dan menafsir soal serta menekuri permasalahan yang diberikan pada soal untuk tugas besar ini, maka diperlukan langkah-langkah besar untuk menyelesaikannya dengan suatu program berbahasa C. Langkah-langkah besar yang perlu diambil untuk menyelesaikan soal ini dan agar pengimplementasiannya sistemati dan akurat adalah sebagai berikut:

Gambar 3. 1. Diagram Alir Rancangan Alur Kerja Penyelesaian Masalah TSP Soal

Dengan kamus dari alur rancangan kerja tersebut adalah berikut ini:

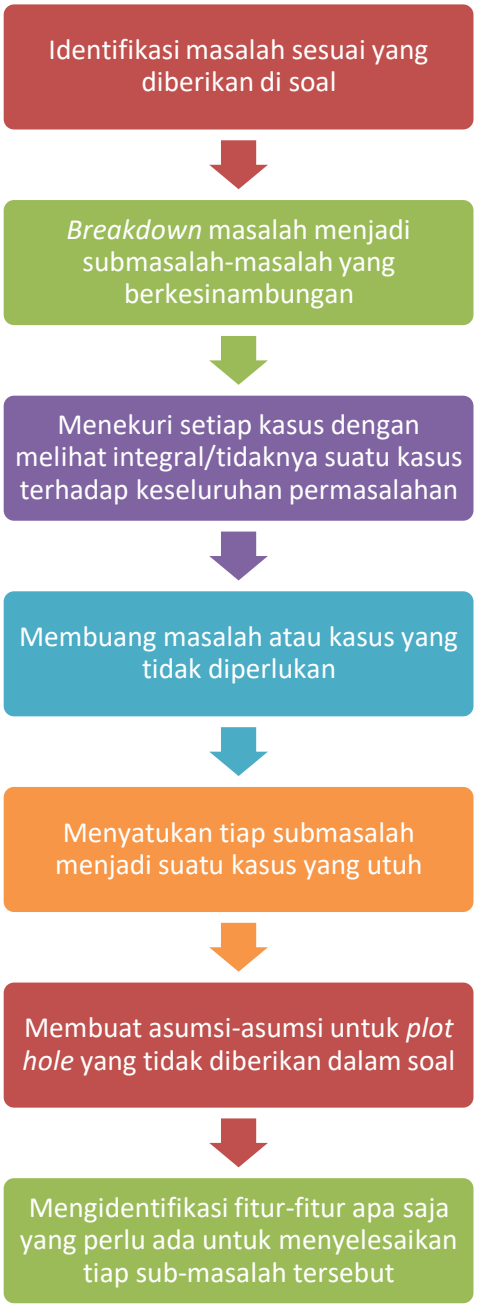
Tabel 3- 1 Deskripsi langkah kerja

Langkah Kerja	Deskripsi
Memahami soal dan masalah dari tugas besar	• Penalaran awal serta pemahaman terhadap soal yang diberikan • Dengan jelas menangkap poin-poin penting permasalahan yang diminta oleh soal • Mengidentifikasi materi yang berkaitan dengan soal
Pendefinisian Ruang Lingkup Masalah	• Mem-*breakdown* tiap masalah menjadi kotak-kotak kasus • Mengaitkan tiap masalah dengan materi yang ada • Mengidentifikasi kasus yang ada dan

	memilah kasus yang penting dan tidak • Membuat asumsi-asumsi
Perancangan *software*	• Mencari contoh implementasi tiap materi yang ditemukan pada Langkah sebelumnya di Bahasa C • Membuat alur berpikir dan DFD bertahap untuk merancang struktur program
Pengimplementasian Rancangan	• Membuat pembagian tugas sesuai dengan *breakdown* yang telah dilakukan. • Mengerjakan implementasi dalam bahasa C sesuai dengan pembagian tugas. • Menyatukan fungsi-fungsi yang telah dibuat masing-masing anggota dalam satu main file.
Pengujian	• Mengecek apakah program yang telah dibuat sudah sesuai dengan asumsi yang telah dibuat dan spesifikasi pada soal. • Memperbaiki kode apabila masih ada kesalahan pada output.

3.1 PENDEFINISIAN RUANG LINGKUP MASALAH

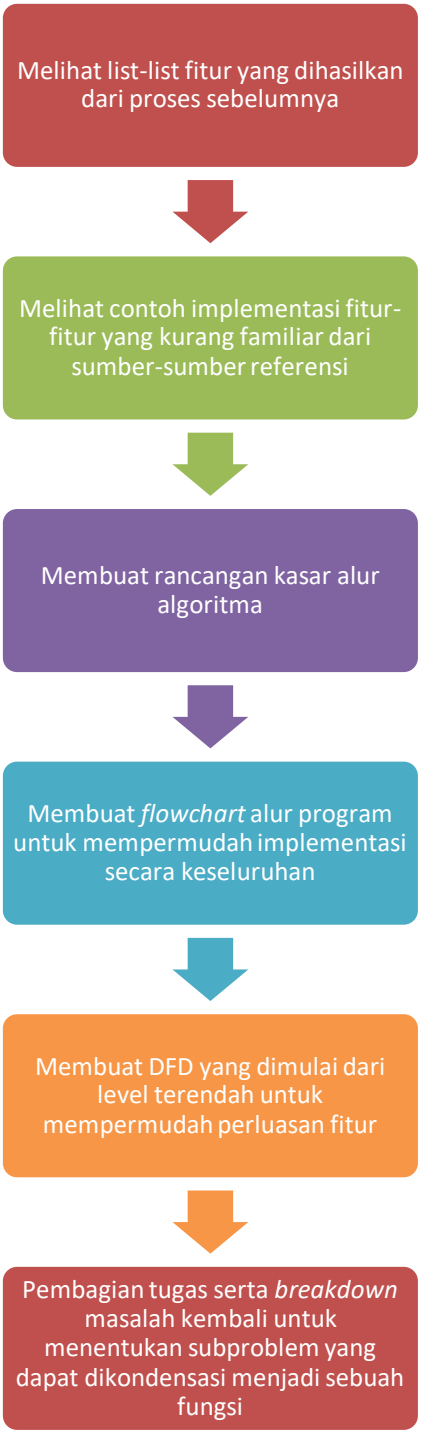
Secara *detail*-nya, Langkah-langkah kecil untuk pendefinisian Ruang Lingkup Masalah adalah:



Gambar 3. 2. Diagram Alir Alur Kerja Pendefinisian Ruang Lingkup Masalah

3.2 PERANCANGAN SOFTWARE

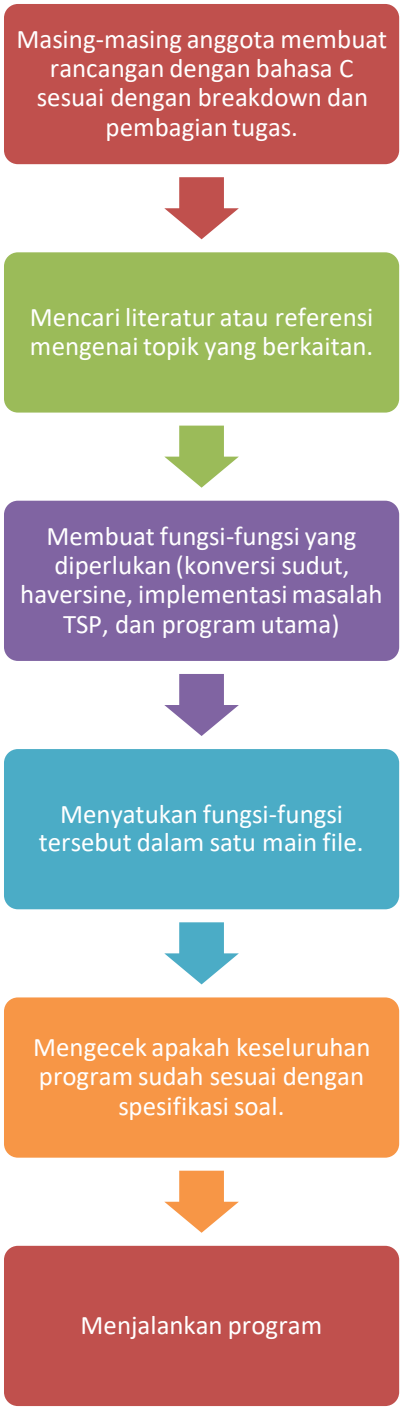
Setelah menentukan ruang lingkup masalah dari soal yang diberikan, maka dapat dilakukan perancangan program serta *software* berdasarkan perancangan kasar di alur berpikir sebelumnya.



Gambar 3. 3. Diagram Alir Alur Kerja Perancangan Software

3.3 PENGIMPLEMENTASIAN RANCANGAN

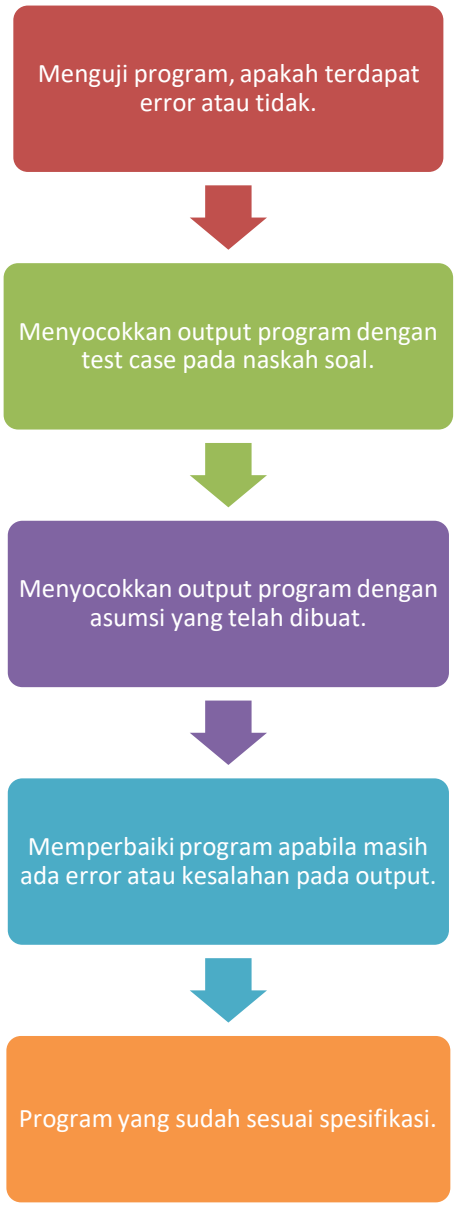
Setelah selesai merancang *software,* langkah selanjutnya adalah mengimplementasikan rancangan tersebut dalam sebuah program berbahasa C. Berikut ini adalah diagram alir dari langkah pengimplementasian rancangan.



Gambar 3. 4. Diagram Alir Pengimplementasian Rancangan

3.4 PENGUJIAN

Setelah rancangan diimplementasikan, langkah selanjutnya adalah pengujian. Tujuan dari pengujian adalah menyesuaikan output program yang telah dibuat apakah sudah sesuai dengan naskah soal dan asumsi atau tidak. Berikut ini adalah langkahnya yang disajikan dalam diagram alir.



Gambar 3. 5. Diagram Alir Pengujian

4. HASIL DAN ANALISIS

Setelah melakukan uji coba serta *problem solving* soal dengan metode-metode yang telah dirancang, dihasilkan hasil yang telah cukup sesuai dengan permintaan soal. Hasil dan pembahasan dari setiap langkah besar akan dijelaskan berikutnya.

4.1 RUANG LINGKUP MASALAH

Setelah melakukan pemahaman terhadap soal yang diberikan serta mengidentifikasi permasalahan yang diberikan di dalamnya, secara umum masalah yang diminta diselesaikan adalah:

1. Membaca *file* sesuai *input* dan mengekstraksi seluruh isinya untuk dapat diolah
2. Menentukan jarak antara dua Pelabuhan dari data lintang dan bujur yang diberikan
3. Mencari rute pelayaran terpendek untuk seluruh Pelabuhan yang dapat dijangkau setidaknya dua Pelabuhan lain dalam jarak <2500 km
4. Memberikan data Pelabuhan yang tidak dapat dijangkau dalam rute pelayaran ini

Dengan menggunakan hasil identifikasi masalah-masalah di atas, maka kelompok kami memberikan asumsi yaitu:

1. *File* csv dari pengguna disimpan di direktori yang sama dengan program
2. Jari-jari bumi 6371 km.

Dan dengan mengacu pada asumsi yang ada maka ruang lingkup masalah dapat dibuat yaitu dengan mem-*breakdown* menjadi kasus-kasus yang perlu diperhatikan yaitu:

1. Rekognisi nama, pembacaan, dan ekstrasi *file*
2. Menghitung jarak dua Pelabuhan dari dua data lintang-bujur
3. Menyeleksi seluruh Pelabuhan yang tidak terjangkau.
4. Membuat rute pelayaran optimal serta menghitung jarak total untuk seluruh Pelabuhan yang terjangkau.

Serta kasus-kasus yang dapat diabaikan yaitu:

1. Mencari *file* hingga ke direktori lainnya sehingga *file* langsung dianggap tidak ada dan program selesai sesuai kasus biasa.

Dengan demikian, program yang akan dibuat dapat dirancang dengan menyertakan fitur-fitur yaitu:

1. Menerima masukan nama *file* dari pengguna
2. Membaca *file*
3. Mengekstrasi isi *file* agar dapat diolah
4. Menghitung jarak antar dua Pelabuhan dari data lintang dan bujur dengan rumus Haversine
5. Mencatat Pelabuhan-pelabuhan yang tidak dapat dijangkau yaitu yang berkondisi tidak terjangkau dua Pelabuhan lain dalam jarak <2500 km dan menyingkirkannya dari titik potensi dalam rute pelayaran.
5. Membuat rute pelayaran terpendek dengan basis Travelling Salesman Problem algorithm dan hitung jarak totalnya.

4.2 RANCANGAN

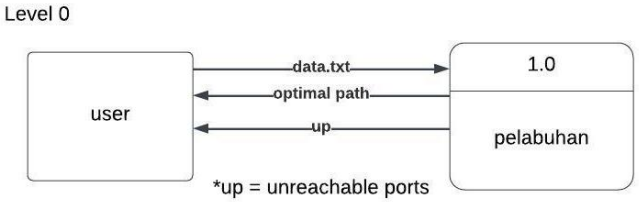
Dalam pembuatan rancangan, sesuai yang diminta pada soal, interaksi pengguna dan program adalah sebagai berikut:

1. Pengguna memberikan *input* nama *file* yang akan dibaca program, dalam tugas ini hanya diperlukan satu *file* yaitu *file* 'pelabuhan.csv'
2. Program membaca *file* dari pengguna yang disimpan di direktori yang sama.
3. Program melakukan seluruh rangkaian kerjanya dan memberikan hasil akhir berupa rute pelayaran akhir dan data Pelabuhan yang tak terjangkau di layer.

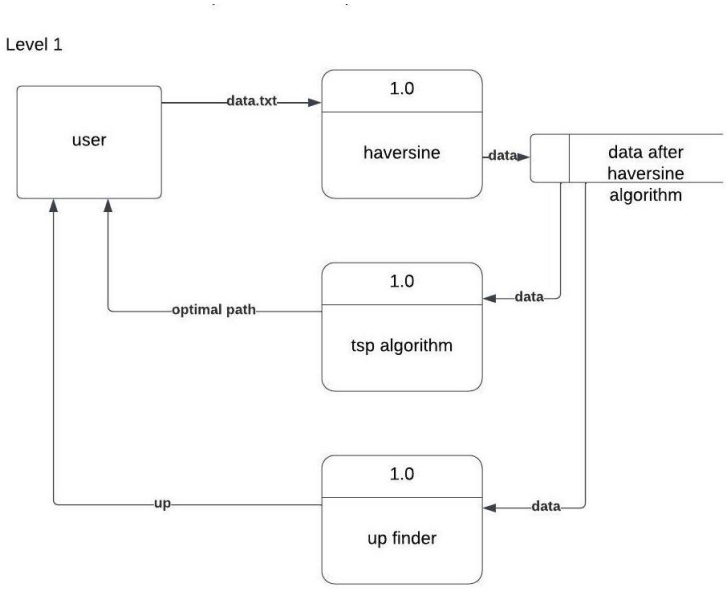
Dengan mengacu pada mekanisme interaksi program dan pengguna serta pendefinisian ruang lingkup masalah di sub-bab sebelumnya, maka akan didapat rancangan algoritma seperti berikut: Terima input nama file dari user dan periksa apakah file tersebut ada, parsing data kedalam Array of Point, yaitu tipedata yang menyimpan alamat, posisi bujur, dan lintang digunakan untuk menyimpan informasi pelabuhan. Setelah itu, panggil fungsi neirest_neighbour_tsp untuk mencari jalan pelabuhan yang dilalui. Fungsi ini menerima Array of Point dan jumlah Point yang bekerja dengan cara membuat 2 Array yang berisikan status dari suatu pelabuhan sudah dikunjungi atau belum dan berisikan jalur yang diambil, lalu dipanggil fungsi define_path yang digunakan untuk mencari jalan paling optimal serta mencari pelabuhan yang tidak dikunjungi. Fungsi define_path bekerja dengan algoritma nearest neighbour atau mencari tetangga terdekat dengan jarak antar pelabuhan dicari dengan fungsi haversine. Setelah fungsi define_path selesai, array of path akan terisi dan dipanggil fungsi print_path yang digunakan untuk memberikan output.

Yang dengan demikian akan menghasilkan rancangan DFD serta Flowchart berikut:

DFD

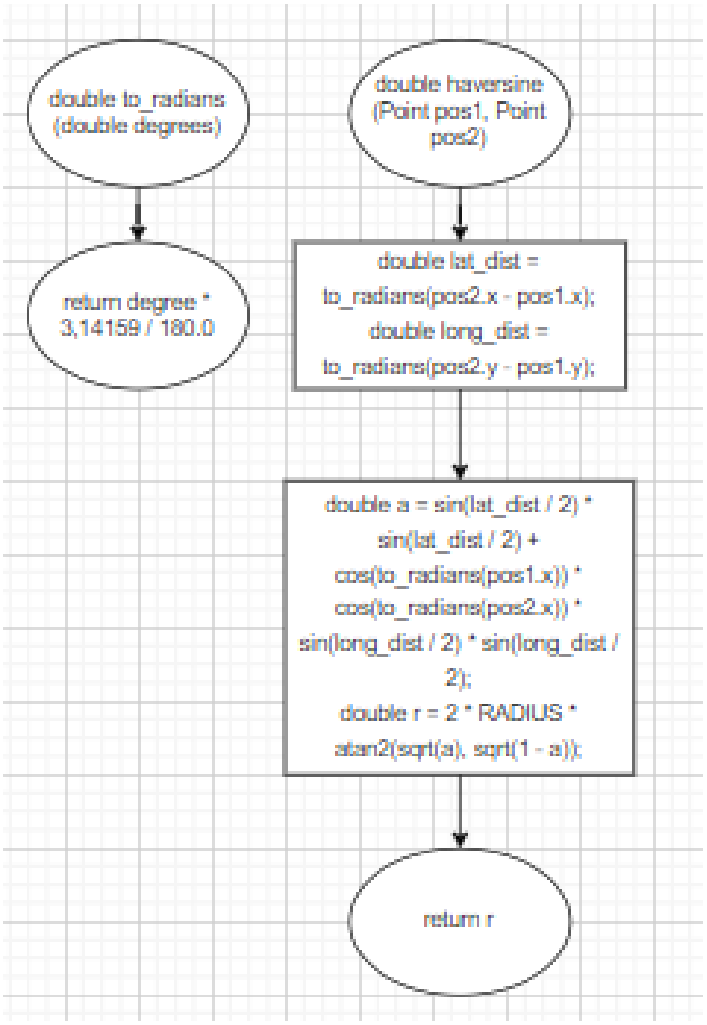


Gambar 4. 1. DFD Level 0

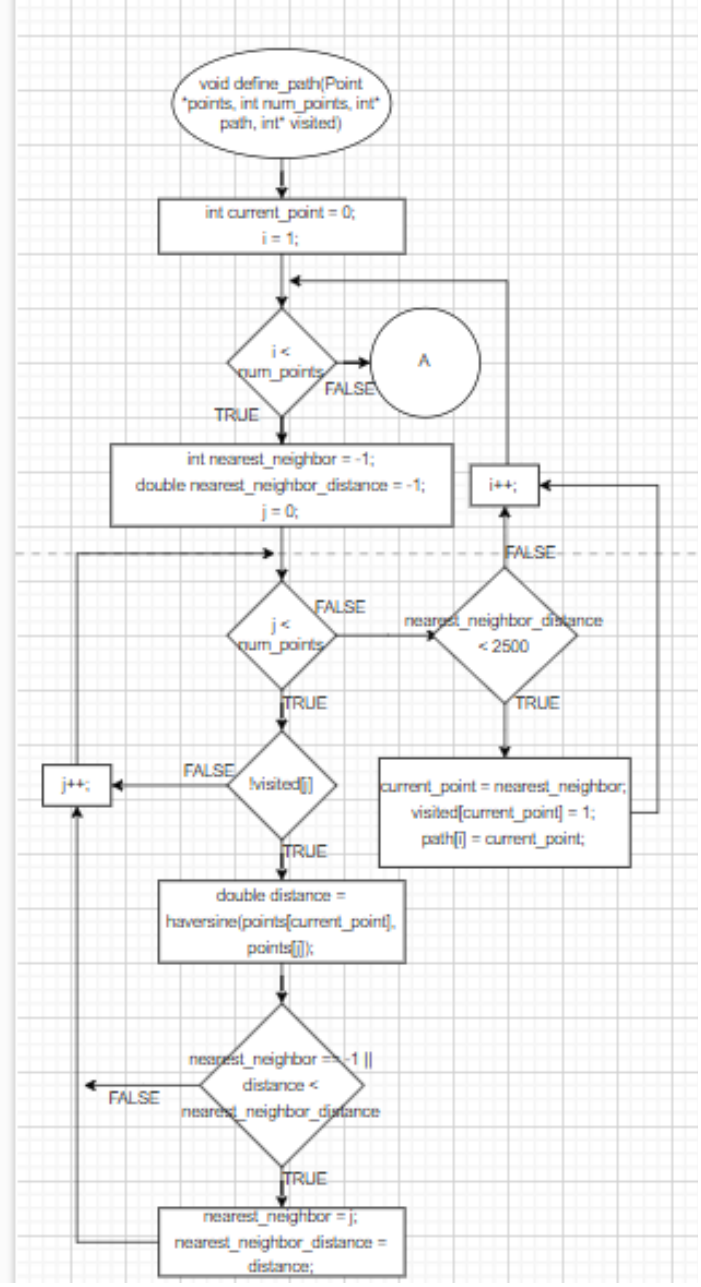


Gambar 4. 2. DFD Level 1

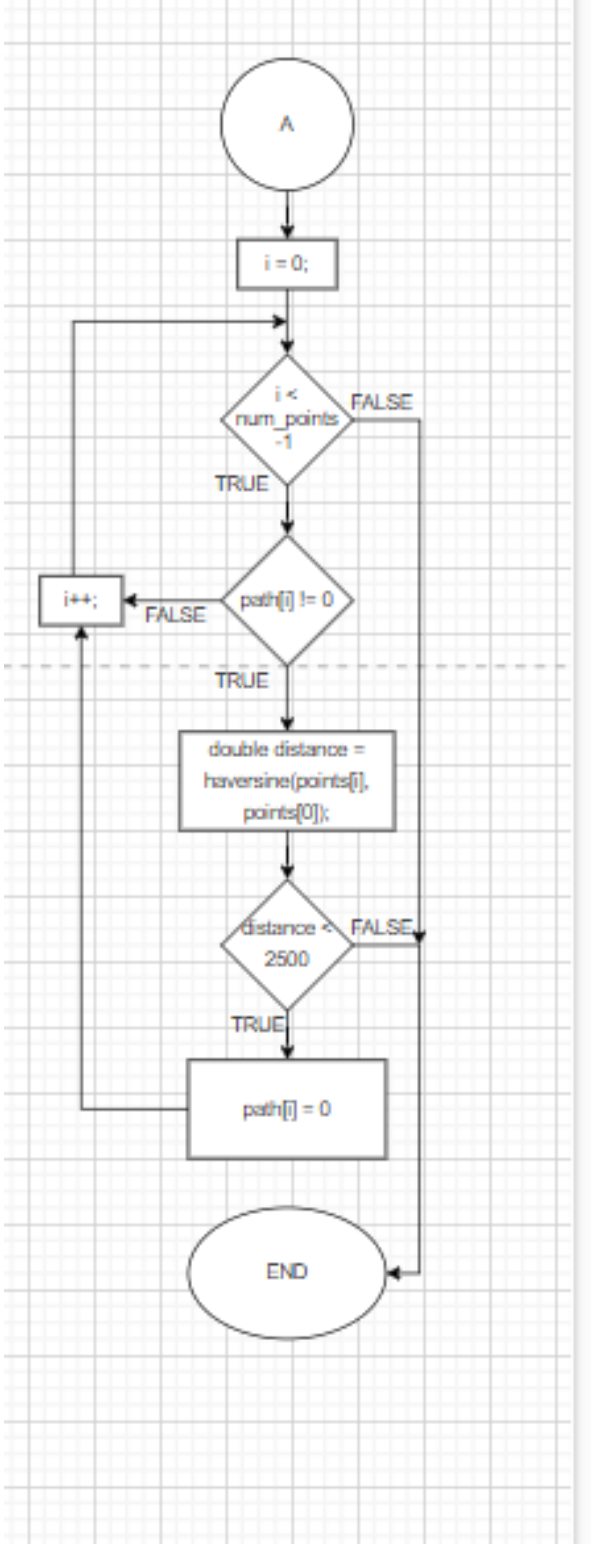
Flowchart



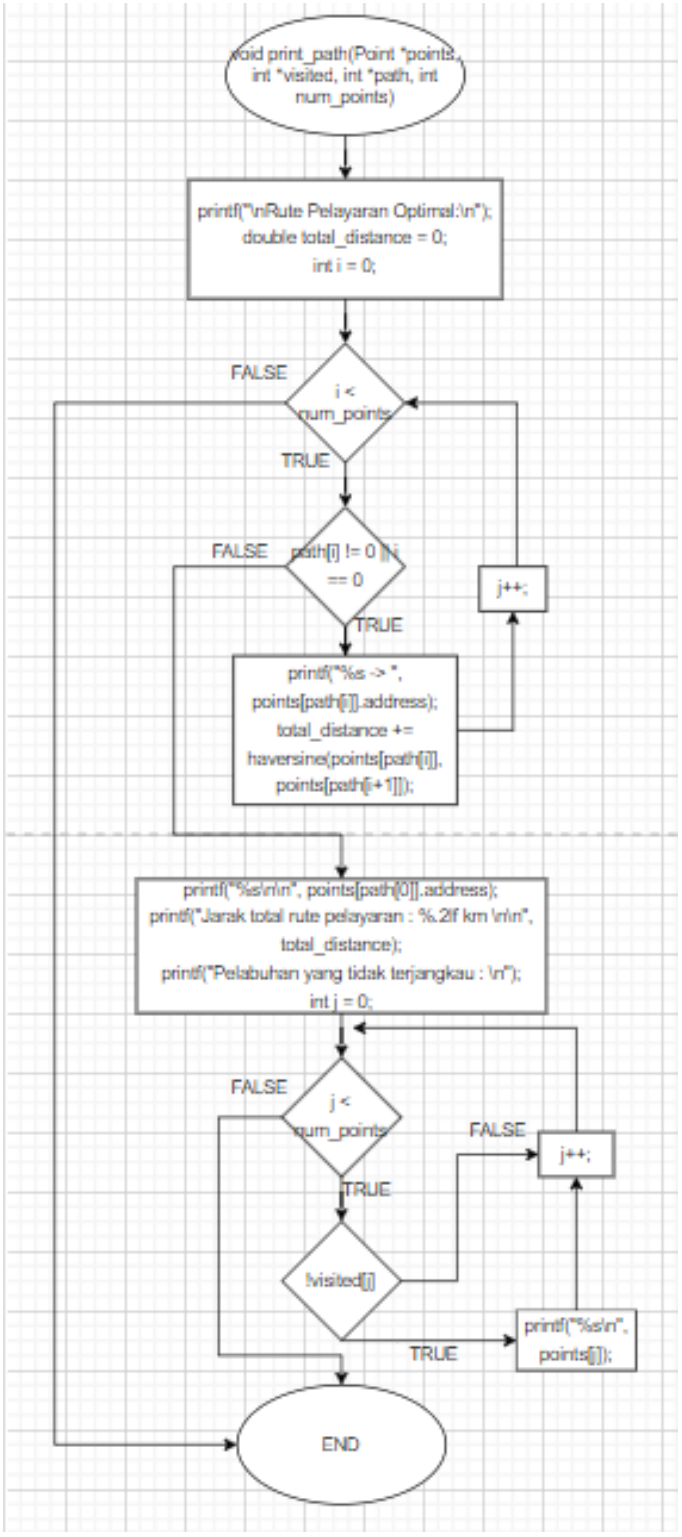
Gambar 4. 3. Flowchart Fungsi to_radian dan haversine



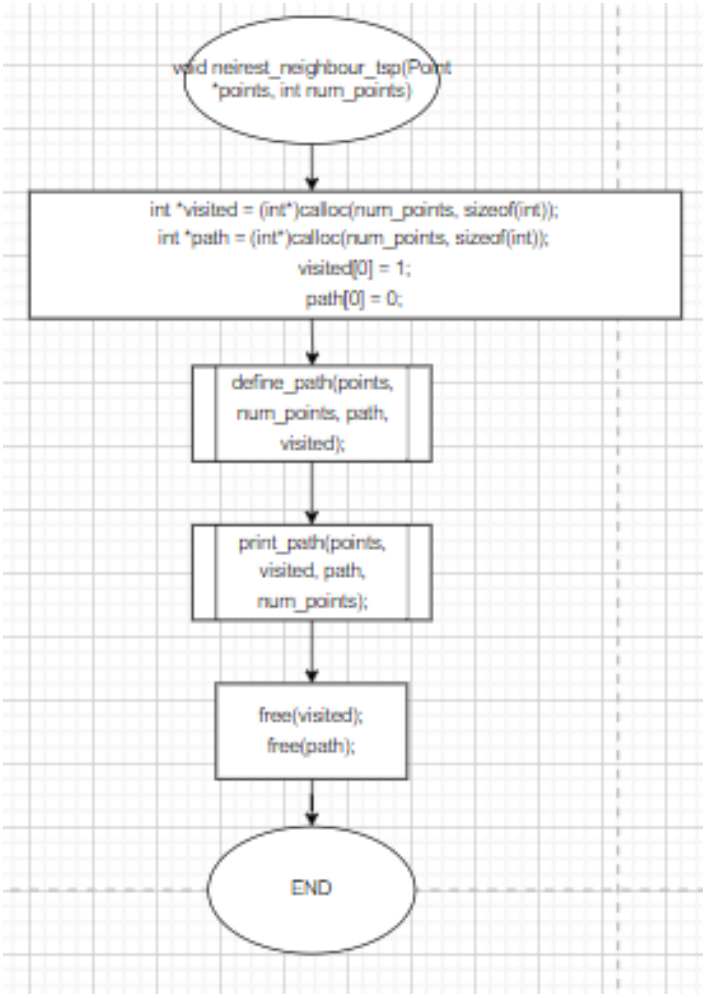
Gambar 4. 4. Flowchart fungsi define_path (i<num_points true)



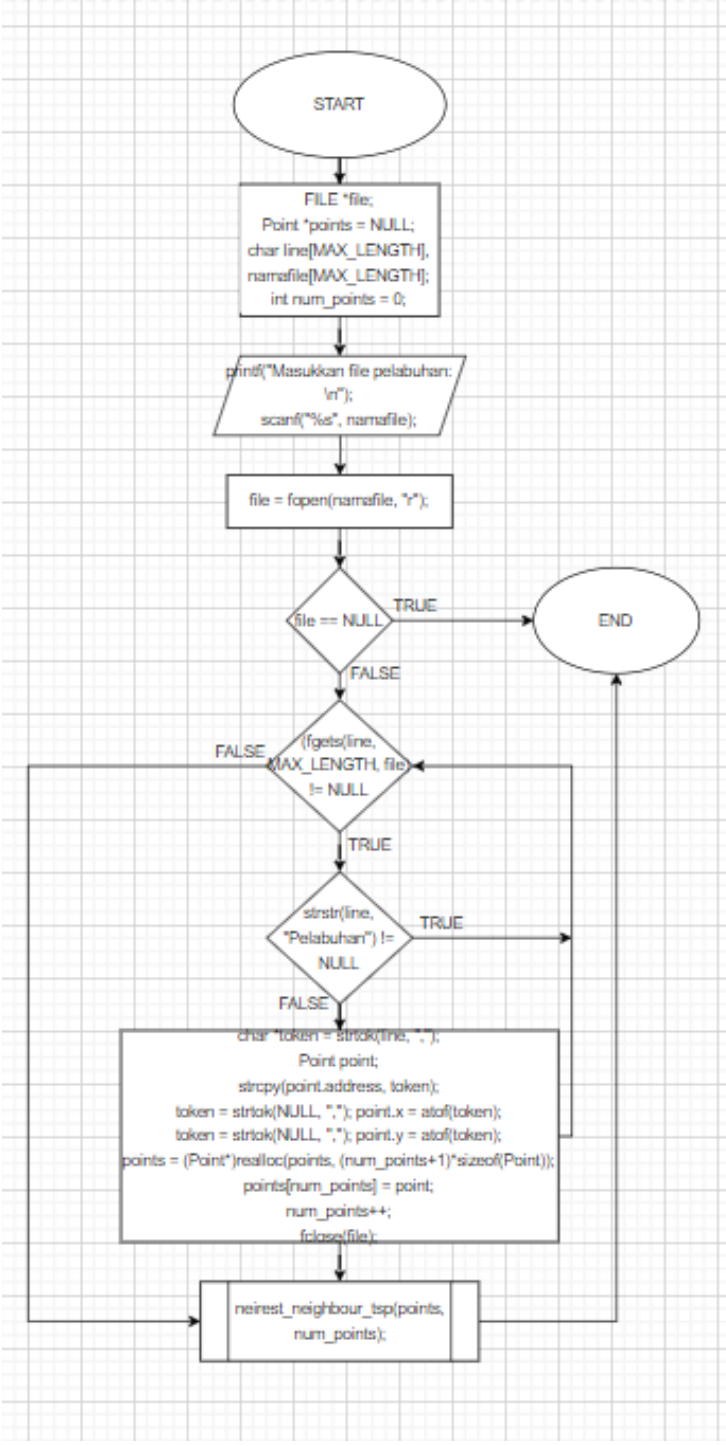
Gambar 4. 5. Flowchart fungsi define_path (i<num_points false)



Gambar 4. 6. Flowchart fungsi find_path



Gambar 4. 7. Flowchart fungsi nearest_neighbor



Gambar 4. 8. Flowchart fungsi main

Untuk menyelesaikan rancangan *software* ini, maka disertakan juga rancangan pembagian tugas awal untuk mempersingkat waktu kerja:

Tabel 4. 1. Rancangan Pembagian Tugas Awal

Task	Pembagian Tugas
Penyelesaian Masalah (main.c)	Desainer: - 13221013 Implementer: - 13221014 Tester: - 13221015 - 13221017

4.3 IMPLEMENTASI

Pembagian tugas masing-masing anggota sudah tertera seperti pada Tabel 4.1. Kami mengimplementasikan kode berikut berdasarkan rancangan yang sudah dibuat sebelumnya. Untuk penjelasan *source code* yang telah dibuat, *source code* terdiri dari beberapa fungsi seperti **to_radians ()**, **haversine(), define_path(), print_path(), neirest_neighbour_tsp(), dan main()**. Fungsi **to_radians()** secara umum berguna untuk mengonversi satuan sudut dari degree ke radians. Fungsi **haversine()** merupakan implementasi dari rumus Haversine yang digunakan untuk menghitung jarak dua titik pada permukaaan Bumi. Fungsi ini menerima masukan berupa koordinat yang merepresentasikan lintang dan bujur dari dua titik yang akan dihitung jaraknya. Variabel a akan menghitung nilai menggunakan rumus Haversine dengan menggunakan fungsi sin, cos, dan atan. Setelah itu, variabel r akan mengonversi hasil variabel a agar fungsi mengembalikan nilai jarak dalam satuan kilometer.

Fungsi **define path()** adalah implementasi algorima Nearest Neighbour yang digunakan untuk mencari lintasan terpendek dalam graf. Parameter yang digunakan pada fungsi ini antara lain array of Point, jumlah titik pada graf, array of int yang merepresentasikan urutan jalur lintasan, serta array of int yang berisi status apakah suatu titik sudah dikunjungi atau belum. *For loop* akan mencari titik terdekat yang belum dikunjungi dengan fungsi haversine() yang sudah dibuat sebelumnya. Jika jarak ke titik terdekat kurang dari 2500km, maka titik tersebut akan dijadikan titik lintasan dan lintasan terpendek akan ditemukan ketika semua titik sudah terlintasi.

Fungsi **print_path()** akan digunakan untuk menampilkan rute pelayaran optimal, total jarak yang ditempuh, dan daftar pelabuhan yang tidak terjangkau dengan format sesuai pada naskah soal. Parameter yang diterima fungsi ini adalah lain array of Point, jumlah titik pada graf, array of int yang merepresentasikan urutan jalur lintasan, serta array of int yang berisi status apakah suatu titik

sudah dikunjungi atau belum. Pada setiap *For loop*, fungsi akan menampilkan alamat titik pada indeks path[i] jika path[i] != 0 atau i = 0. Selanjutnya, total jarak akan ditambahkan dengan jarak antara titik saat ini dan titik berikutnya. Jika path[i] = 0 dan i != 0, maka daftar pelabuhan yang tidak terjangkau akan ditampilkan dengan memerikas status visited[j] terlebih dahulu. Fungsi akan berhenti pada titik terakhir graph dan menampilkan total jarak tempuh serta daftar pelabuhan yang tidak dikunjungi.

Kemudian, Fungsi **neirest_neighbour_tsp()** adalah implementasi dari Travelling Salesman Problem menggunakan pendekatan nearest neighbour. Fungsi akan menerima parameter array of Point dan jumlah titik pada graph. Pertama-tama fungsi akan mengalokasikan memori untuk array visited dan path. Titik pertama diinisialisasi sebagai titik awal dan diberi status telah dikunjungi. Kemudian fungsi ini akan memanggil fungsi define_path() agar dapat mencari rute optimal. Selanjutnya, fungsi ini juga akan memanggil fungsi print_path() agar dapat menampilkan rute pelayaran yang optimal, total jarak tempuh, serta daftar pelabuhan yang tidak terjangkau. Pada bagian akhir dari kode ini, fungsi akan membebaskan memori yang dialokasikan untuk array visited dan path.

Pada program utama **main()**, awalnya program ini akan melakukan inisialisasi variabel dan meminta pengguna memasukkan nama file agar dapat dibuka oleh program. Setelah itu, program akan membaca file sesuai dengan masukan pengguna serta memroses informasi tentang pelabuhan lalu disimpan pada array points dengan menggunakan fungsi strtok() dan atof(). Kemudian, program utama akan memanggil fungsi neirest_neighbour_tsp() yang sudah dibuat sebelumnya untuk menentukan rute pelayaran optilmal dan memanggil fungsi print_path() agar dapat mencetak hasil sesuai dengan spesifikasi. Setelah semua proses tersebut selesai, program akan membebaskan memori yang dialokasikan di awal program dan keseluruhan program akan selesai dengan menampilkan output sesuai spesifikasi.

4.4 PENGUJIAN

Hasil implementasi rancangan di atas kemudian akan diuji kebenarannya serta hasil akhirnya. Hasil eksekusi dari pengujian adalah sebagai berikut:

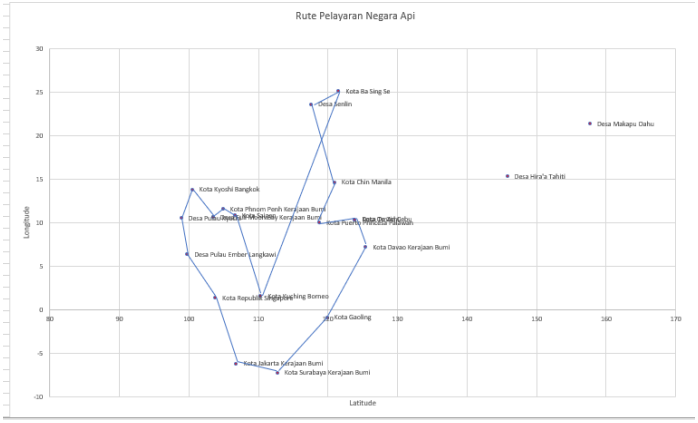
```
Masukkan file pelabuhan: pelabuhan.csv

Rute Pelayaran Optimal:
Kota Ba Sing Se -> Desa Senlin -> Kota Chin Manila -> Kota Puerto Princesa Palawan -> Kota Omashu -> Desa Tu Zin Cebu ->
Kota Davao Kerajaan Bumi -> Kota Gaoling -> Kota Surabaya Kerajaan Bumi -> Kota Jakarta Kerajaan Bumi -> Kota Republik Si
ngapore -> Desa Pulau Ember Langkawi -> Desa Pulau Kyoshi -> Kota Kyoshi Bangkok -> Desa Full Moon Bay Kerajaan Bumi -> K
ota Phnom Penh Kerajaan Bumi -> Kota Saigon -> Kota Kuching Borneo -> Kota Ba Sing Se

Jarak total rute pelayaran : 13161.80 km

Pelabuhan yang tidak terjangkau :
Desa Makapu Oahu
Desa Hira'a Tahiti
```

Gambar 4. 9. Hasil Eksekusi Program

Atau secara jelasnya:

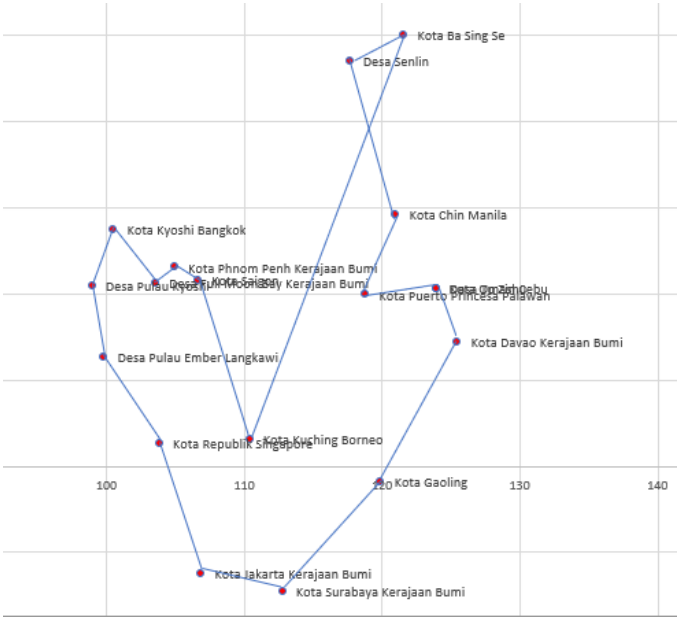
```
Masukkan file pelabuhan: pelabuhan.csv

Rute Pelayaran Optimal:
Kota Ba Sing Se -> Desa Senlin -> Kota Chin
Manila -> Kota Puerto Princesa Palawan -> Kota
Omashu -> Desa Tu Zin Cebu -> Kota Davao Kerajaan
Bumi -> Kota Gaoling -> Kota Surabaya
Kota Davao Kerajaan Bumi -> Kota Gaoling -> Kota
Surabaya Kerajaan Bumi -> Kota Jakarta Kerajaan
Bumi -> Kota Republik Siulau Ember Langkangkok ->
Desa Full Moon Bay Kerajaan Bumingapore -> Desa
Pulau Ember Langkawi -> Desa Pulau Kyoshi -> Kota
Kyoshi Bangkok -> Desa Full Moon Bay Kerajaan
Bumi -> Kta Saigon -> Kotaota Phnom Penh Kerajaan
Bumi -> Kota Saigon -> Kota Kuching Borneo ->
Kota Ba Sing Se

Jarak total rute pelayaran : 13161.80 km

Pelabuhan yang tidak terjangkau :
Desa Makapu Oahu
Desa Hira'a Tahiti
```

Maka akan didapatkan visualisasi data yang berupa:



Gambar 4. 10. Visualisasi Data Hasil Eksekusi Program

Atau secara lebih detail:



Gambar 4. 11. Rute Pelayaran Hasil Akhir Program

Gambar 4. 12. Dua Pelabuhan yang Tidak Terjangkau

Dan secara *detail* hasil akhir pengujian adalah berikut ini:

Tabel 4. 2. Hasil Pengujian Program

No.	Aspek Pengujian	Hasil Pengujian
1.	**Waktu Eksekusi**	**1.035 seconds**
2.	**File .c *space needed***	**5 kb**
3.	**File .exe *space needed***	**48 kb**
2.	**Checklist**	✓ **Menerima *input*** ✓ **Membaca *file*** ✓ **Ekstraksi *file*** ✓ **Menyeleksi kondisi yang telah ditentukan sebelumnya** ✓ **Mengeluarkan *output* yang sesuai dengan permintaan soal**
3.	**Hasil Rute Pelayaran**	**Kota Ba Sing Se -> Desa Senlin -> Kota Chin Manila -> Kota Puerto Princesa Palawan -> Kota Omashu -> Desa Tu Zin Cebu -> Kota Davao Kerajaan Bumi -> Kota Gaoling -> Kota Surabaya Kota Davao Kerajaan Bumi -> Kota Gaoling -> Kota Surabaya Kerajaan Bumi -> Kota Jakarta Kerajaan Bumi -> Kota Republik Siulau Ember Langkangkok -> Desa Full Moon Bay Kerajaan Bumingapore -> Desa Pulau Ember Langkawi -> Desa Pulau Kyoshi -> Kota Kyoshi Bangkok -> Desa Full Moon Bay Kerajaan Bumi -> Kta Saigon -> Kotaota Phnom Penh Kerajaan Bumi -> Kota Saigon -> Kota Kuching Borneo -> Kota Ba Sing Se**
4.	**Total Jarak Akhir**	**13161.80 km**
5.	**Pelabuhan yang Tidak Terjangkau**	**Desa Makapu Oahu** **Desa Hira'a Tahiti**

Dengan mempertimbangkan bahwa:

1. Waktu eksekusi dan *space* yang dibutuhkan cukup
2. Hasil *output* telah keluar sesuai kondisi yang telah di-*breakdown* sebelumnya pada rancangan
3. Program sudah dapat di-*compile* dengan mudah

Maka hasil akhir pengujian sudah dapat membuktikan bahwa implementasi rancangan telah berhasil dan soal tugas besar ini telah terjawab dengan baik.

6. KESIMPULAN

Setelah pemaparan akhir serta menganalisis keseluruhan hasil dari percobaan modul ini, maka didapatkan kesimpulan yaitu:

1. Penggunaan Bahasa C bisa untuk menyelesaikan permasalahan dunia nyata misalnya permasalahan Travelling Salesman Problem ini
2. Dynamic Programming approach yang digunakan untuk menyelesaikan TSP telah sesuai dan optimal
3. Penggunaan Haversine formula untuk menghitung jarak dengan lintang dan bujur telah sesuai.
4. Rancangan sudah dapat dibilang baik dan implementasi berhasil karena sudah menghasilkan hasil uji yang sesuai prekondisi.
5. Rute pelayaran akhir serta data Pelabuhan yang tak terjangkau sudah didapatkan dengan hasil jarak total 13160 km dan dua Pelabuhan yang tak terjangkau adalah Makapu Oahu dan Hira'a Tahiti.

DAFTAR PUSTAKA

[1] Henri Fachrudin, *Optimasi Penentuan Rute Perjalanan Sales pada UD. ASTER*, Universitas Islam Majapahit Mojokerto, Mojokerto, 2019.

[2] Jatra Sulistio, *Implementasi Metode Haversine Formula Dalam Aplikasi Untuk Menentukan*

Lokasi Emergency Service Terdekat Di Daerah Istimewa Yogyakarta, Fakultas Teknologi Informasi dan Elektro Universita Teknologi Yogyakarta, 2019.

[3] Ahmad Fauzi, dkk., *Penerapan Metode Haversine Formula Pada Aplikasi Pencarian Lokasi Tempat Tambal Ban Kendaraan Bermotor Berbasis Mobile Android,* Universitas Bina Sarana Informatika, 2018.

[4] https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/, 26 April 2023, 08.33

[5] https://www.guru99.com/travelling-salesman-problem.html, 26 April 2023, 08.11

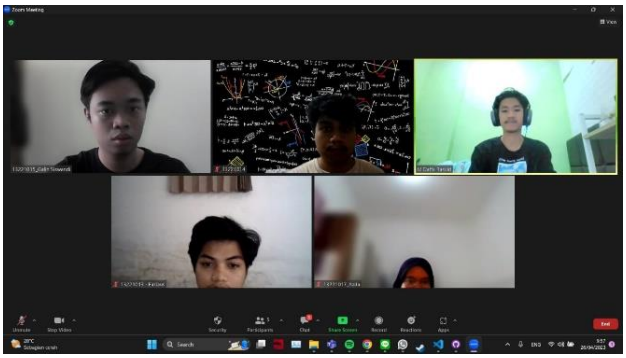
[6] https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/, 26 April 2023, 09.59

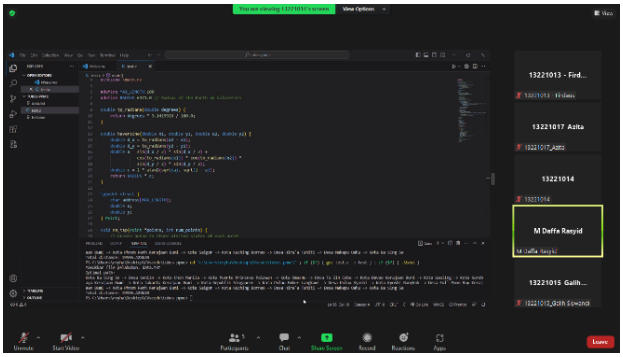
LAMPIRAN

Link Repository

https://github.com/el2208-ppmc-2023/modul-9-tema-b1

Bukti Asistensi





Source Code

```
/*EL2208 Praktikum Pemecahan Masalah dengan C 2022/2023
*Modul               : 9 - Tugas Besar
*Kelompok            : B1
*Hari dan Tanggal    : Rabu, 26/04/2023
*Asisten (NIM)       : Muhammmad Daffa Rasyid (13220059)
*Nama File           : Traveling Salesman Problem
*Deskripsi           : Program menyelesaikan permasalahan Traveling Salesman Problem dengan persyaratan jarak yang ditempuh kurang dari 2500 KM
*/
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX_LENGTH 100 // Maksimal panjang
#define RADIUS 6371.0 // Jari jari bumi

typedef struct {
    char address[MAX_LENGTH];
    double x;
    double y;
} Point;

double to_radians(double degrees) {
    return degrees * 3.1415926 / 180.0;
}

double haversine(Point pos1, Point pos2) {
    double lat_dist = to_radians(pos2.x - pos1.x);
    double long_dist = to_radians(pos2.y - pos1.y);
    double a = sin(lat_dist / 2) * sin(lat_dist / 2) + cos(to_radians(pos1.x)) * cos(to_radians(pos2.x)) * sin(long_dist / 2) * sin(long_dist / 2);
    double r = 2 * RADIUS * atan2(sqrt(a), sqrt(1 - a));
    return r;
}

void define_path(Point *points, int num_points, int* path, int* visited){
    int current_point = 0;

    for (int i = 1; i < num_points; i++) {
        int nearest_neighbor = -1;
        double nearest_neighbor_distance = -1;

        // Menentukan jarak terdekat
        for (int j = 0; j < num_points; j++) {
                if (!visited[j]){
                    double distance = haversine(points[current_point], points[j]);
                    if (nearest_neighbor == -1 || distance < nearest_neighbor_distance){
                        nearest_neighbor = j;
                        nearest_neighbor_distance = distance;
                    }
                }
```

```
            }

        // Menyeleksi jarak agar kurang dari
2500 km
        if(nearest_neighbor_distance < 2500){
            current_point = nearest_neighbor;
            visited[current_point] = 1;
            path[i] = current_point;
        }
    }

    // Menyeleksi jarak agar kembali ke titik
awal kurang dari 2500 km
    for (int i = num_points-1; i >= 0; i--) {
        if (path[i] != 0)
        {
            double distance =
haversine(points[i], points[0]);
            if(distance < 2500)
            {
                break;
            }
            else
            {
                path[i] = 0;
            }
        }
    }
}

void print_path(Point *points, int *visited,
int *path, int num_points){
    printf("\nRute Pelayaran Optimal:\n");
    double total_distance = 0;
    for (int i = 0; i < num_points; i++) {
        if (path[i] != 0 || i == 0)
        {
            printf("%s -> ",
points[path[i]].address);
            total_distance +=
haversine(points[path[i]], points[path[i+1]]);
        }
        else {
            printf("%s\n\n",
points[path[0]].address);
            printf("Jarak total rute
pelayaran : %.2lf km \n\n", total_distance);
            printf("Pelabuhan yang tidak
terjangkau : \n");
            for (int j = 0; j < num_points; j++) {
                if (!visited[j]){
                    printf("%s\n", points[j]);
                }
            }
            break;
        }
    }
}

void neirest_neighbour_tsp(Point *points, int
num_points) {
    // Deklarasi array status tempat dan jalur
    int *visited = (int*)calloc(num_points,
sizeof(int));
    int *path = (int*)calloc(num_points, sizeof(int));

    // Inisialisasi titik sekarang
    visited[0] = 1;
    path[0] = 0;

    // Fungsi pencari jalan
    define_path(points, num_points, path, visited);

    // Fungsi print jalur, total jarak, dan pelabuhan
yang tidak bisa dikunjungo
    print_path(points, visited, path, num_points);

    free(visited);
    free(path);
}

int main() {
    // Inisialisasi
    FILE *file;
    Point *points = NULL;
    char line[MAX_LENGTH], namafile[MAX_LENGTH];
    int num_points = 0;

    // Input FILE
    printf("Masukkan file pelabuhan: \n");
    scanf("%s", namafile);

    // Membuka FILE
    file = fopen(namafile, "r");
    if (file == NULL) {
        printf("Cannot open file.\n");
        return 1;
    }
```

```
      // Membaca FILE
    while (fgets(line, MAX_LENGTH, file) !=
NULL) {
        if (strstr(line, "Pelabuhan") != NULL)
{
            continue;
        }

        char *token = strtok(line, ",");
        Point point;

        strcpy(point.address, token);
        token = strtok(NULL, ","); point.x =
atof(token);
        token = strtok(NULL, ","); point.y =
atof(token);
        points = (Point*)realloc(points,
(num_points+1)*sizeof(Point));
        points[num_points] = point;
        num_points++;
    }
    fclose(file);

    // Algoritma TSP
    neirest_neighbour_tsp(points, num_points);

    free(points);
    return 0;
}
```