IF2211 - Strategi Algoritma
Laporan Tugas Kecil 2

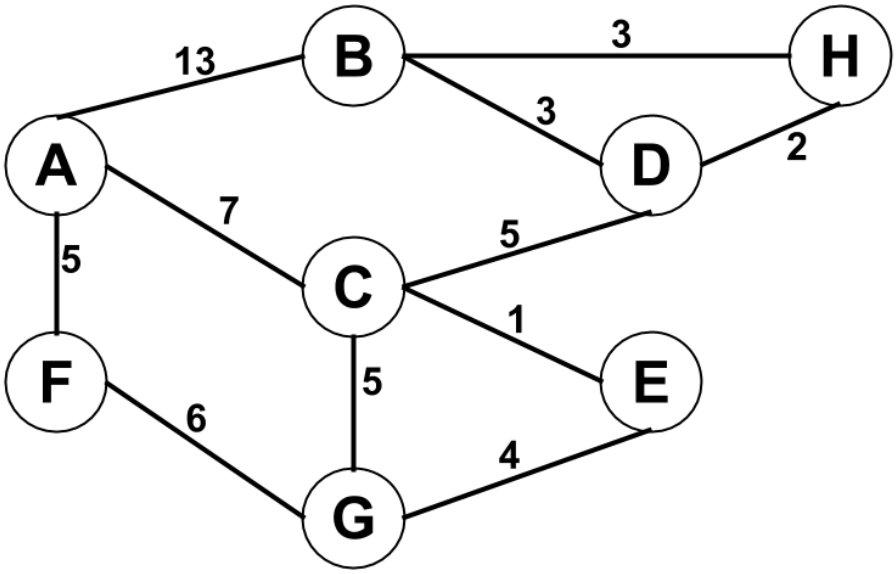
Disusun Oleh:

Henry Anand Septian Radityo	**13521004**
Muhammad Hadar Akita Tresnadi	**13521025**

Institut Teknologi Bandung
Sekolah Teknik Elektro dan Informatika
Tahun Ajaran 2022/2023

Daftar Isi

1. Deskripsi Masalah



Algoritma pencarian rute terpendek merupakan salah satu topik yang sangat penting dalam bidang ilmu komputer dan matematika terapan. Pencarian rute terpendek menjadi dasar dalam banyak aplikasi, seperti perencanaan rute pada navigasi GPS, optimasi pengiriman barang, pengaturan lalu lintas, dan sebagainya. Pada laporan ini, penulis akan membahas mengenai dua algoritma pencarian rute terpendek, yaitu algoritma A* (A-star) dan algoritma Uniform Cost Search (UCS).

Algoritma A* dan UCS digunakan untuk mencari rute terpendek antara dua titik dalam suatu graf atau peta yang direpresentasikan sebagai sebuah graf. Graf tersebut terdiri dari simpul (node) yang mewakili lokasi atau titik dalam peta, dan tepi (edge) yang menghubungkan antara dua simpul yang dapat dilalui. Setiap tepi memiliki bobot atau biaya yang mewakili jarak, waktu, atau biaya yang diperlukan untuk berpindah dari satu simpul ke simpul lainnya. Tujuan dari algoritma pencarian rute terpendek adalah untuk menemukan rute yang memiliki total bobot atau biaya terkecil antara dua simpul yang ditentukan.

2. Metode Penyelesaian

2.1 Algoritma Uniform Cost Search (UCS)

Uniform cost search atau yang biasa disebut pencarian biaya seragam merupakan salah satu cara untuk mencari jarak secara optimal dari suatu node ke node lain yang berada dalam satu graf. Algoritma ini akan mengembalikan biaya dan juga jarak yang ditempuh untuk mencapai node tujuan. UCS juga mempertimbangkan solusi yang memiliki biaya paling rendah sehingga UCS selalu mengembalikan solusi yang paling optimal.

Secara umum, prinsip kerja UCS adalah mencari sedari simpul awal kemudian mengeksplorasi semua simpul yang bertetanggan dengan simpul awal. Simpul yang sudah dicek kemudian akan ditandai bahwa simpul tersebut sudah dikunjungi sehingga tidak dapat dikunjungi kembali. Kemudian algoritma UCS akan mencari rute yang memiliki biaya paling rendah dan mengeksplorasi simpul yang bertetangga dengan simpul tersebut hingga mendapatkan rute menuju simpul tujuan dan tidak ada biaya lain yang lebih kecil.

Uniform Cost Search memiliki keuntungan yaitu selalu mendapatkan sebuah graf dengan biaya paling minimum. Namun di sisi lain, UCS akan mengalami kesulitan apabila graf yang dilakukan memiliki suatu node yang sangat besar karena pencarian akan membutuhkan waktu yang lama dan memori yang tidak sedikit.

2.2 Algoritma A-Star (A*)

A - Star adalah sebuah algoritma yang dilakukan untuk mencari rute dengan biaya terpendek pada suatu graf dengan sebuah bobot pada setiap sisinya. Pencarian ini mirip dengan pencarian Uniform Cost Search namun memperhatikan jarak heuristik antara node yang sedang dilakukan pengecekan dan node tujuan.

Dalam A*, dilakukan pencarian secara sistematis dari simpul awal ke simpul tujuan dengan mengeksplorasi jalur dengan biaya terendah (cost) terlebih dahulu. Biaya ini diperoleh dengan menggabungkan dua nilai yaitu biaya sejauh ini (g), yaitu total biaya untuk mencapai simpul saat ini dan heuristik (h), yaitu estimasi biaya dari simpul saat ini ke simpul tujuan.

A* mengkombinasikan kedua nilai tersebut menggunakan fungsi $f(n) = g(n) + h(n)$, dan memilih simpul dengan nilai f(n) terkecil untuk dieksplorasi selanjutnya. Dengan demikian, A* mampu menemukan jalur terpendek dari simpul awal ke simpul tujuan dengan efisien.

Meskipun bekerja dengan lebih cepat, namun hasil yang didapatkan dari algoritma ini belum tentu hasil yang paling optimal dikarenakan heuristik yang tidak selalu akurat untuk menampilkan jarak dari suatu node ke node tujuan.

3. Source Code Program

3.1 UCS.py

```
import math
import queue

def UCS(start, end, adjacents):
    pque = queue.PriorityQueue ()
    pque.put((0, start))
    node_cost = { }
    node_cost[start] = 0
    path = []
    while not pque.empty():
        current = pque.get()
        if current[1] == end:
            temp = current[1]
            while(temp != start):
                path.append(temp)
                temp = temp.previous
            path.append(start)
            path = path[::-1]
            return path, node_cost[end]

        for i in range(len(adjacents)):
            if adjacents[i].start.name == current[1].name:
                for j in range(len(adjacents[i].adjacent)):
                    next_node = adjacents[i].adjacent[j][0]
                    cost = adjacents[i].adjacent[j][1]
                    new_cost = node_cost[current[1]] + cost
                    if next_node not in node_cost or new_cost < node_cost[next_node]:
                        node_cost[next_node] = new_cost
                        priority = new_cost
                        next_node.setPrev(current[1])
                        pque.put((priority, next_node))

    return None

def UCS_B(start, end, adjacents):
    weight_matrix = []
    pque = queue.PriorityQueue ()
    pque.put((0, start))
    node_cost = {}
    node_cost[start] = 0
    path = []

    for i in range(len(adjacents)):
        weight = []
        for j in range(len(adjacents[i].adjacent)):
```

```
        # weight.append(float((abs(adjacents[i].start.X) -
abs(adjacents[i].adjacent[j][0].X))**2 + (abs(abs(adjacents[i].start.Y) -
abs(adjacents[i].adjacent[j][0].Y))**2) ** 0.5))
        weight.append(distance(adjacents[i].start.X, adjacents[i].start.Y,
adjacents[i].adjacent[j][0].X, adjacents[i].adjacent[j][0].Y))
    weight_matrix.append(weight)

  while not pque.empty():
    current = pque.get()
    if current[1] == end:
      temp = current[1]
      while(temp != start):
        path.append(temp)
        temp = temp.previous
      path.append(start)
      path = path[::-1]
      return path, node_cost[end]

    for i in range(len(adjacents)):
      if adjacents[i].start.name == current[1].name:
        for j in range(len(adjacents[i].adjacent)):
          next_node = adjacents[i].adjacent[j][0]
          cost = weight_matrix[i][j]
          new_cost = node_cost[current[1]] + cost
          if next_node not in node_cost or new_cost < node_cost[next_node]:
            node_cost[next_node] = new_cost
            priority = new_cost
            next_node.setPrev(current[1])
            pque.put((priority, next_node))

  return None

def distance(lat1, lon1, lat2, lon2):
  # Rumus Haversine
  R = 6371
  lat1_rad = math.radians(lat1)
  lon1_rad = math.radians(lon1)
  lat2_rad = math.radians(lat2)
  lon2_rad = math.radians(lon2)
  dlon = lon2_rad - lon1_rad
  dlat = lat2_rad - lat1_rad
  a = math.sin(dlat/2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) *
math.sin(dlon/2)**2
  c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
  distance = R * c
  return distance
```

3.2 Read.py

```
import Node as n
import Graph as g
import Adjacent as a

def read(file):
    # Open the text file
    with open(file, 'r') as file:
    # Read the lines from the file
        lines = file.readlines()

    array_of_nodes = []
    array_of_adjacent = []
    arrOfNode = []
    i = 0
    # buat node
    for j in range (len(lines)):
        num = int(lines[0])
        if j > num :
            break
        if (j != 0):
            arrOfNode.append(n.Node(lines[j]))

    for j in range (num+1,len(lines)):
        adjacent = []
        temp = arrOfNode[i]
        numbers = lines[j].strip().split(' ')
        for node in range(len(numbers)):
            if numbers[node] != '0':
                adjacent.append([arrOfNode[node], int(numbers[node])])
        array_of_nodes.append(temp)
        array_of_adjacent.append(a.Adjacent(temp, adjacent))
        i += 1

    return array_of_adjacent, array_of_nodes

def readWithCoor(file):
    with open(file, 'r') as file:
    # Read the lines from the file
        lines = file.readlines()
    array_of_nodes = []
    array_of_adjacent = []
    i = 0
    prev = 0
    for j in range (len(lines)):
        num = int(lines[0])
        if j > num *2:
            break
```

```
      if (j != 0):
         if (j % 2 == 1):
            array_of_nodes.append(n.NodeB(lines[j]))
         else:
            temporaryCoor = lines[j].strip().split(' ')
            array_of_nodes[prev].setCoor(float(temporaryCoor[0]),
float(temporaryCoor[1]))
            prev +=1
   for j in range(num*2+1,len(lines)):
      adjacent = []
      temp = array_of_nodes[i]
      numbers = lines[j].strip().split(' ')
      for node in range(len(numbers)):
         if numbers[node] != '0':
            adjacent.append([array_of_nodes[node]])
      array_of_adjacent.append(a.AdjacentB(temp, adjacent))
      i += 1
   return array_of_adjacent, array_of_nodes

def getNode(file):
   with open(file, 'r') as f:
      # Read the lines from the file
      lines = f.readlines()
   num = int(lines[0])
   if (len(lines) > num*2+1):
      a, b = readWithCoor(file)
      flag = True
   else:
      a, b = read(file)
      flag = False
   return b, flag
```

3.3 Graph.py

```
import Node as n

class Graph:
   def __init__(self, start, end, nodes):
      self.start = start
      self.end = end
      self.nodes = nodes
      self.cost = 0
      self.state = start
```

```
    def updateNodes(self, nodes):
        self.nodes.append(nodes)
    #graf sebagai array of adjacent
    def __init__(self, adjacent):
        self.list = []
        self.list.append(adjacent)
        self.cost = 0
        self.purecost = 0
        self.state = self.list[len(self.list)-1]
    def setState(self, state):
        self.state = state
    def addAdjNode(self, node):
        self.list.append(node)
        self.state = self.list[len(self.list)-1]
    def getCost(self):
        return self.cost
    def display(self):
        print("List of nodes")
        for i in range (len(self.list)):
            self.list[i].display()
```

3.4 Adjacent.py

```
class AdjacentB:
    def __init__(self, start, adjacent):
        # node
        self.start = start
        # list of nodes
        # [node, node, ...]
        self.adjacent = adjacent

    def display(self):
        print("============================")
        self.start.display()
        print("List adjacent")
        for i in range (len(self.adjacent)):
            self.adjacent[i][0].display()
        print("============================")
    def getName(self):
        return self.start.name

class Adjacent:
    def __init__(self, start, adjacent):
        # node
        self.start = start
```

```
        # list of nodes with weight/distance
        # [[node, distance], [node, distance], ...]
        self.adjacent = adjacent
    def display(self):
        print("==========================")
        self.start.display()
        print("List adjacent")
        for i in range (len(self.adjacent)):
            self.adjacent[i][0].display()
        print("==========================")
```

3.5 Node.py

```
import math

class NodeB:
    def __init__(self,name, previous = None):
        self.name = name
        self.previous = previous

    def setCoor(self,X,Y):
        self.X = X
        self.Y = Y

    def display(self):
        print(self.name)

    def setPrev(self, previous):
        self.previous = previous

    def getDistance(self, node):
        return math.sqrt((self.X - node.X)**2 + (self.Y - node.Y)**2)

class Node:

    def __init__(self, name, previous = None):
        self.name = name
        self.previous = previous

    def display(self):
        print(self.name)

    def setPrev(self, previous):
        self.previous = previous
```

3.6 app.py

```
from flask import Flask, render_template, request, session
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from AStar import AStarBonus, AStar
import reader as read
from UCS import UCS, UCS_B
import folium
import json
# from flask_autoindex import AutoIndex
app = Flask(__name__,template_folder='display')
app.secret_key = 'asterisk'
@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file.filename == '':
            return render_template('home.html')
        else:
            # result = file.read().decode('utf-8')
            result = []

            array_node, flag = read.getNode('./src/test/'+file.filename)
            for i in range (len(array_node)):
                result.append(array_node[i].name)
            # result = result.split('\n')
            session['name'] = file.filename
            return render_template('home.html', result=result, flag = flag)
    else:
        return render_template('home.html')

@app.route('/calc', methods=['POST', 'GET'])
def calc():
    selected_algorithm = request.form.get('options')
    idStart = request.form['Node1']
    idEnd = request.form['Node2']
    coor = False
    if selected_algorithm == 'A*B':
        coor = True
        array_adj, array_node = read.readWithCoor('./src/test/' + session.get('name',
None))
        AstarResult = AStarBonus(array_adj, int(idStart)-1, int(idEnd)-1)
        if (AstarResult):
            result_list = []
            for i in range(len(AstarResult.list)):
                result_list.append([AstarResult.list[i].start.name,
AstarResult.list[i].start.X, AstarResult.list[i].start.Y])
```

```
        cost = AstarResult.purecost
        # make a map folium
        m = folium.Map(location=[result_list[0][1], result_list[0][2]], zoom_start=12)
        # mark all node first
        for i in range(len(array_node)):
            folium.Marker(location=[array_node[i].X, array_node[i].Y],
popup=array_node[i].name).add_to(m)
        # mark all adj
        for el in array_adj:
            for i in range(len(el.adjacent)):
                folium.PolyLine(locations=[[el.start.X, el.start.Y], [el.adjacent[i][0].X,
el.adjacent[i][0].Y]],
                                color='blue', weight=2.5, opacity=1).add_to(m)
        for i in range(len(result_list)):
            folium.Marker(location=[result_list[i][1], result_list[i][2]],
popup=result_list[i][0]).add_to(m)
        # make edge
        for i in range(len(result_list)-1):
            folium.PolyLine(locations=[[result_list[i][1], result_list[i][2]],
[result_list[i+1][1], result_list[i+1][2]]],
                            color="red", weight=2.5, opacity=1).add_to(m)
        m.save('./src/static/result.html')
    else:
        return render_template('home.html', msg="Tidak ada jalur")
  elif selected_algorithm == "A*":
    # clear plot
    plt.clf()
    array_adj, array_node = read.read('./src/test/'+session.get('name', None))
    AstarResult = AStar(array_adj, int(idStart)-1, int(idEnd)-1)
    if (AstarResult):
        result_list = []
        for i in range(len(AstarResult.list)):
            result_list.append(AstarResult.list[i].start.name)
        cost = AstarResult.cost
        # Visualisasi graf
        G = nx.DiGraph()
        for tempAdj in array_adj:
            for i in range(len(tempAdj.adjacent)):
                G.add_edge(tempAdj.start.name, tempAdj.adjacent[i][0].name,
weight=tempAdj.adjacent[i][1])
        # set warna edge
        edge_colors = [G[u][v]['color'] if 'color' in G[u][v] else 'black' for u, v in
G.edges()]
        # set posisi node
        pos = nx.spring_layout(G)
        # gambar graf
        nx.draw_networkx_edges(G, pos, edge_color=edge_colors, width=2,
arrows=False)
        nx.draw_networkx_nodes(G, pos, node_size=2000, node_color='lightblue')
```

```
        nx.draw_networkx_labels(G, pos, font_size=7, font_family='sans-serif')
        # set warna edge hasil
        red_edges = [(AstarResult.list[i].start.name, AstarResult.list[i+1].start.name)
for i in range(len(AstarResult.list)-1)]
        nx.draw_networkx_edges(G, pos, edgelist=red_edges, edge_color='red',
width=2, arrows=False)
        nx.draw_networkx_edge_labels(G, pos,
edge_labels=nx.get_edge_attributes(G, 'weight'))
        #save the graph
        plt.savefig('./src/static/result.png')
      else:
        return render_template('home.html', msg="Tidak ada jalur")

   elif selected_algorithm == "UCS_B":
      coor = True
      array_adj, array_node = read.readWithCoor('./src/test/' + session.get('name',
None))
      try:
        UCS_Result = UCS_B(array_node[int(idStart)-1], array_node[int(idEnd)-1],
array_adj)
      except:
        return render_template('home.html', msg="Terjadi Error")

      if (UCS_Result):
        result_list = []
        for i in range(len(UCS_Result[0])):
          result_list.append([UCS_Result[0][i].name, UCS_Result[0][i].X,
UCS_Result[0][i].Y])
        cost = UCS_Result[1] / 100
        # make a map folium
        m = folium.Map(location=[result_list[0][1], result_list[0][2]], zoom_start=12)
        # mark all node first
        for i in range(len(array_node)):
          folium.Marker(location=[array_node[i].X, array_node[i].Y],
popup=array_node[i].name).add_to(m)
        # mark all adj
        for el in array_adj:
          for i in range(len(el.adjacent)):
            folium.PolyLine(locations=[[el.start.X, el.start.Y], [el.adjacent[i][0].X,
el.adjacent[i][0].Y]],
                    color='blue', weight=2.5, opacity=1).add_to(m)
        for i in range(len(result_list)):
          folium.Marker(location=[result_list[i][1], result_list[i][2]],
popup=result_list[i][0]).add_to(m)
        # make edge
        for i in range(len(result_list)-1):
          folium.PolyLine(locations=[[result_list[i][1], result_list[i][2]],
[result_list[i+1][1], result_list[i+1][2]]],
                  color="red", weight=2.5, opacity=1).add_to(m)
```

```
            m.save('./src/static/result.html')
        else:
            return render_template('home.html', msg="Tidak ada jalur")

    elif selected_algorithm == "UCS":
        # clear plot
        plt.clf()
        array_adj, array_node = read.read('./src/test/'+session.get('name', None))
        try:
            UCS_Result = UCS(array_node[int(idStart)-1], array_node[int(idEnd)-1],
array_adj)
        except:
            return render_template('home.html', msg="Terjadi Error")

        if (UCS_Result):
            result_list = []
            for i in range(len(UCS_Result[0])):
                result_list.append(UCS_Result[0][i].name)
            cost = UCS_Result[1] / 100
            # Visualisasi graf
            G = nx.DiGraph()
            for tempAdj in array_adj:
                for i in range(len(tempAdj.adjacent)):
                    G.add_edge(tempAdj.start.name, tempAdj.adjacent[i][0].name,
weight=tempAdj.adjacent[i][1])
            # set warna edge
            edge_colors = [G[u][v]['color'] if 'color' in G[u][v] else 'black' for u, v in
G.edges()]
            # set posisi node
            pos = nx.spring_layout(G)
            # gambar graf
            nx.draw_networkx_edges(G, pos, edge_color=edge_colors, width=2,
arrows=False)
            nx.draw_networkx_nodes(G, pos, node_size=2000, node_color='lightblue')
            nx.draw_networkx_labels(G, pos, font_size=7, font_family='sans-serif')
            # set warna edge hasil
            red_edges = [(UCS_Result[0][i].name, UCS_Result[0][i+1].name) for i in
range(len(UCS_Result[0])-1)]
            nx.draw_networkx_edges(G, pos, edgelist=red_edges, edge_color='red',
width=2, arrows=False)
            nx.draw_networkx_edge_labels(G, pos,
edge_labels=nx.get_edge_attributes(G, 'weight'))
            #save the graph
            plt.savefig('./src/static/result.png')
        else:
            return render_template('home.html', msg="Tidak ada jalur")

    if (coor):
        cost = cost* 111320
```

```
        return render_template('index2.html', cost = cost, result = result_list)
    else :
        return render_template('index.html', cost = cost, result  = result_list)
if __name__ == "__main__":
    app.run(debug=True)
```

3.7 AStar.py

```
import Node as n
import Graph as g
import Adjacent as a
import reader as read
import copy
import networkx as nx
import matplotlib.pyplot as plt

def inside(arr, el):
    #cek apakah adjacent el sudah ada di dalam arr
    for i in range (len(arr)):
        if (arr[i].start.name == el.name):
            return True
    return False
def isContainsEndNode(arr, el):
    #arr is graph
    for i in range (len(arr.list)):
        if (arr.list[i] == el):
            return True
    return False

def AStar(arrAdj, idxStart, idxEnd):
    found = False
    visited = []
    pqueue = []
    result = []
    temp = arrAdj[idxStart]

    #make a graph with start node
    parent = g.Graph(temp)
    # visited.append(temp)
    pqueue.append(parent)
    #inisiasi endnode
    endNode = arrAdj[idxEnd]
    if (idxEnd==idxStart):
        return parent
    idx = 0
    while not found:
        # print('---------------------------------------')
        #hapus dulu parentnya dari list
```

```
    del pqueue[idx]
    # ambil adjacent dari graf parent sampai dimana kita
    temp = parent.state
    for i in range (len(temp.adjacent)):
        tempParent = copy.deepcopy(parent)
        if not inside(visited, temp.adjacent[i][0]):
            #cari graf yang ingin dimasukkan
            for j in range (len(arrAdj)):
                if arrAdj[j].start.name == temp.adjacent[i][0].name:
                    tempAdj = arrAdj[j]
                    break
            if (not tempAdj == None):
                # print('tempAdj : ' + str(tempAdj.start.name))
                # tempAdj.display()
                tempParent.addAdjNode(tempAdj)
                tempGraph = tempParent
                #update cost
                heuristic = (len(arrAdj)-len(tempGraph.list))*2
                # heuristic = 0
                tempGraph.cost = tempGraph.purecost + temp.adjacent[i][1] + heuristic
                tempGraph.purecost = tempGraph.purecost + temp.adjacent[i][1]
                #masukkan dalam result apabila menemukan hasil
                if (isContainsEndNode(tempGraph, endNode)):
                    result.append(tempGraph)
                #masukkan dalam pqueue dan visited
                pqueue.append(tempGraph)
    # cari node yang belum dikunjungi dengan f(n) terkecil
    if (len(pqueue)==0):
        break
    visited.append(temp)
    min = copy.deepcopy(pqueue[0])
    idx = 0
    for i in range (len(pqueue)):
        if (pqueue[i].cost < min.cost):
            min = pqueue[i]
            idx = i
    parent = min
    # parent.display()
    #cari apakah sudah tidak bisa lagi / pencarian selesai
    if (len(result)>0):
        min = result[0]
        for i in range (len(result)):
            #cari nilai minimal di result
            if (result[i].cost < min.cost):
                min = result[i]
        #cari nilai minimal di pqueue
        min2 = pqueue[0]
        for i in range (len(pqueue)):
            if (pqueue[i].cost < min2.cost):
```

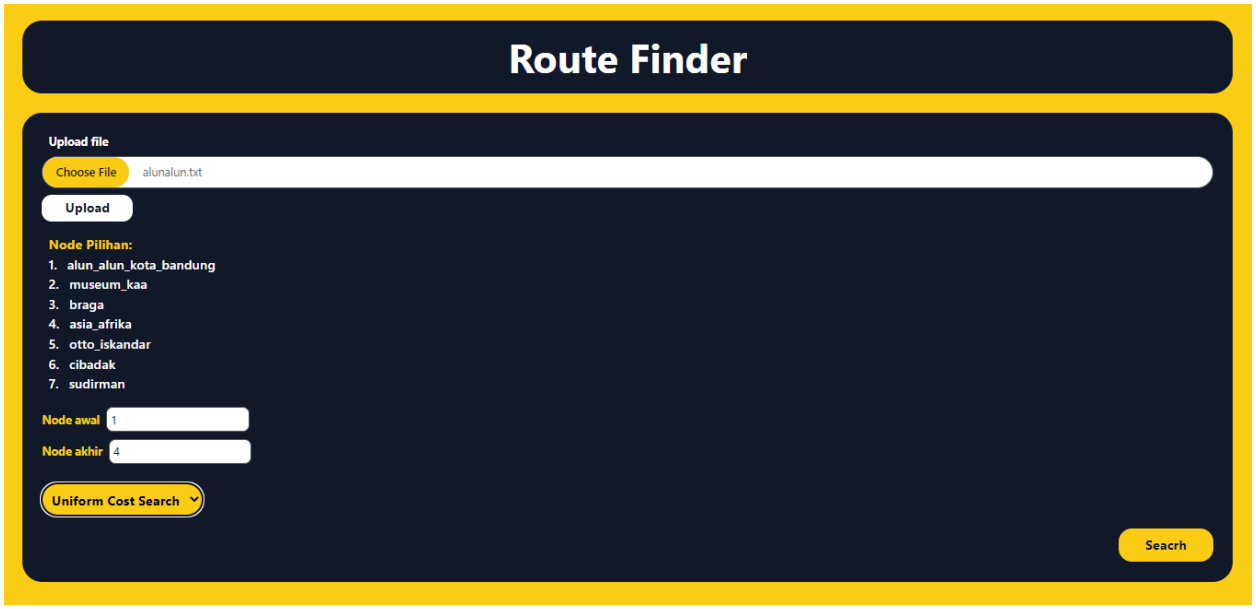
```
                min2 = pqueue[i]
            if (min.cost <= min2.cost):
                found = True
    if found:
        for i in range (len(result)):
            if (result[i].cost == min.cost):
                return result[i]
    else:
        return None
def distance(Adj1, Adj2):
    #menghitung jarak antar node
    return ((Adj1.start.X - Adj2.start.X)**2 + (Adj1.start.Y - Adj2.start.Y)**2)**0.5
def AStarBonus(arrAdj, idxStart, idxEnd):
    found = False
    visited = []
    pqueue = []
    result = []
    # temp akan mengambil adjacent pertama
    temp = arrAdj[idxStart]
    #make a graph with start node
    parent = g.Graph(temp)
    # visited.append(temp)
    pqueue.append(parent)
    #inisiasi endnode
    endNode = arrAdj[idxEnd]
    if (idxEnd==idxStart):
        return parent
    idx = 0
    p = 0
    while not found:
        #hapus dulu parentnya dari list
        del pqueue[idx]
        # ambil adjacent dari graf parent sampai dimana kita
        temp = parent.state
        # cari untuk setiap adjacent dalam temp
        for i in range (len(temp.adjacent)):
            tempParent = copy.deepcopy(parent)
            if not inside(visited, temp.adjacent[i][0]):
                #cari graf yang ingin dimasukkan
                # print('lakukan pengecekan untuk ' + temp.adjacent[i][0].name)
                for j in range (len(arrAdj)):
                    if arrAdj[j].start.name == temp.adjacent[i][0].name:
                        tempAdj = arrAdj[j]
                        break

                previous = copy.deepcopy(tempParent.state)
                tempParent.addAdjNode(tempAdj)
                tempGraph = tempParent
```

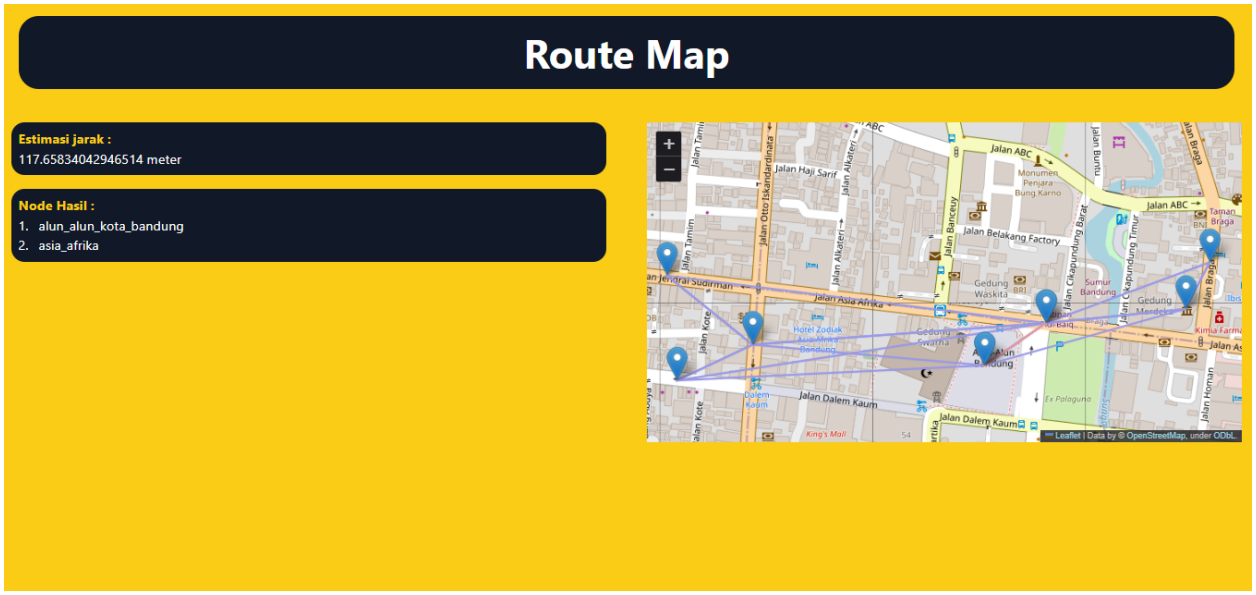
```
            # update cost
            heuristic = distance(tempGraph.state, arrAdj[idxEnd])

            # heuristic = 0
            tempGraph.cost = tempGraph.purecost + distance(previous,
tempGraph.state) + heuristic
            tempGraph.purecost = tempGraph.purecost + distance(previous,
tempGraph.state)

            # masukkan dalam result apabila menemukan hasil
            if (isContainsEndNode(tempGraph, endNode)):
               result.append(tempGraph)
            # masukkan dalam pqueue dan visited
            pqueue.append(tempGraph)
      # cari node yang belum dikunjungi dengan f(n) terkecil
      if (len(pqueue)==0):
         break
      visited.append(temp)
      # print("MASUKIN " + temp.start.name)
      min = copy.deepcopy(pqueue[0])
      idx = 0
      for i in range (len(pqueue)):
         if (pqueue[i].cost < min.cost):
            min = copy.deepcopy(pqueue[i])
            idx = i
      parent = min
      #cari apakah sudah tidak bisa lagi / pencarian selesai
      if (len(result)>0):
         min = result[0]
         for i in range (len(result)):
            #cari nilai minimal di result
            if (result[i].cost < min.cost):
               min = result[i]
         #cari nilai minimal di pqueue
         min2 = pqueue[0]
         for i in range (len(pqueue)):
            if (pqueue[i].cost < min2.cost):
               min2 = pqueue[i]
         if (min.cost <= min2.cost):
            found = True
   for i in range (len(result)):
      if (result[i].cost == min.cost):
         return result[i]
```

4. Eksperimen

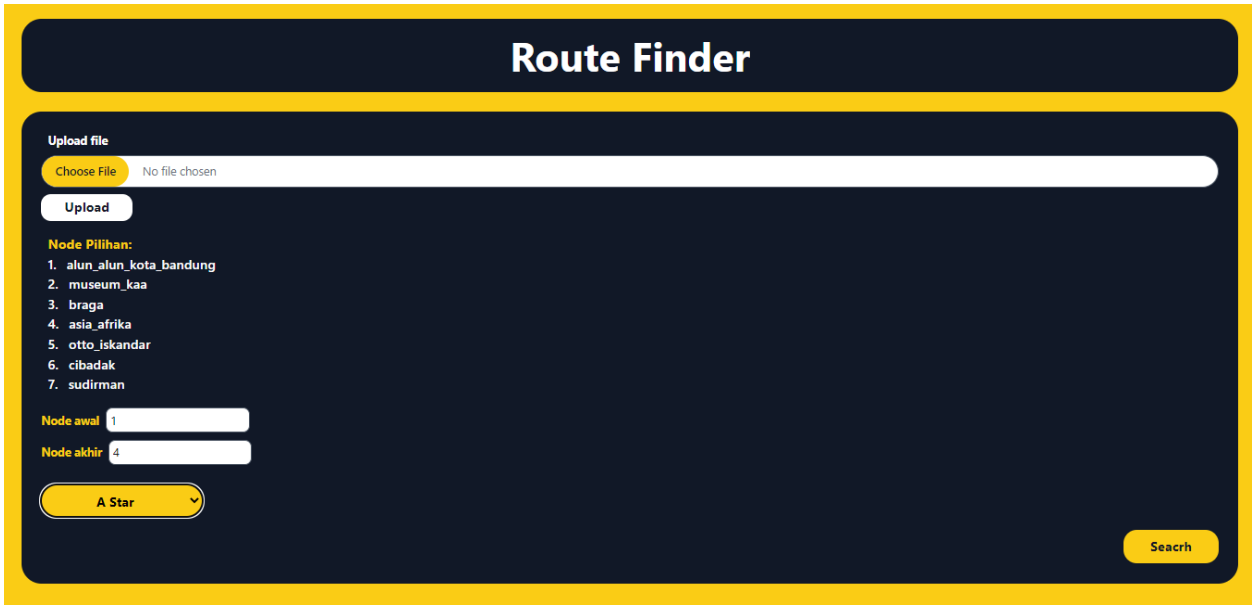
4.1 Alunalun Bandung

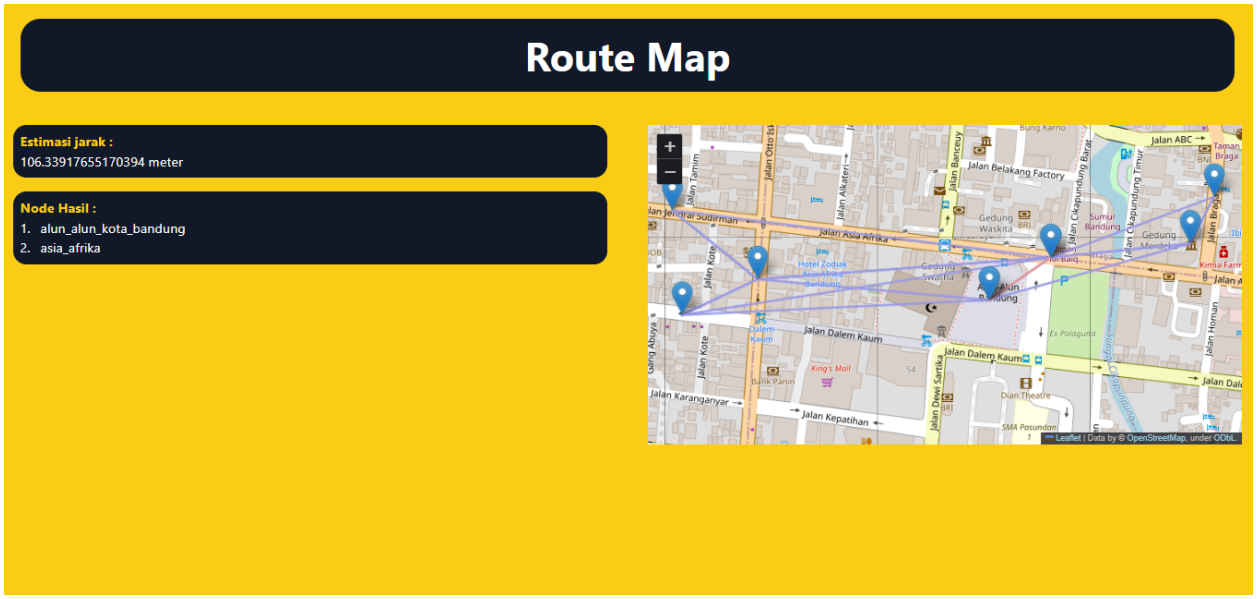
4.1.1 Alunalun.txt dengan UCS Algorithm (1 ke 4)





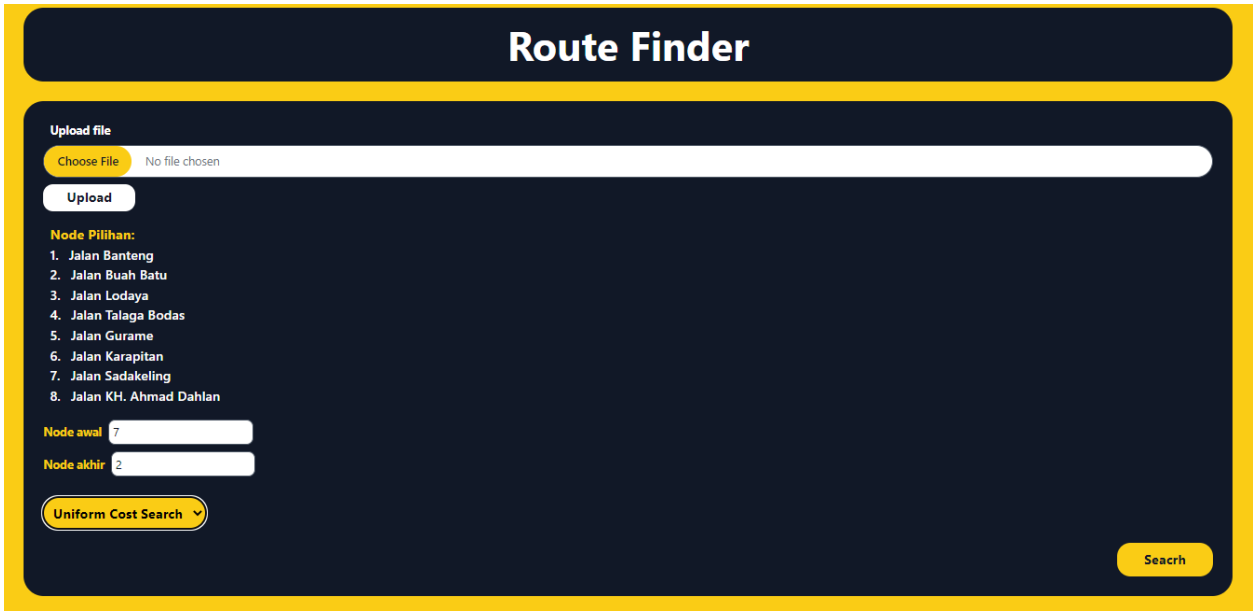
4.1.2 Alunalun.txt dengan A* Algorithm (1 ke 4)

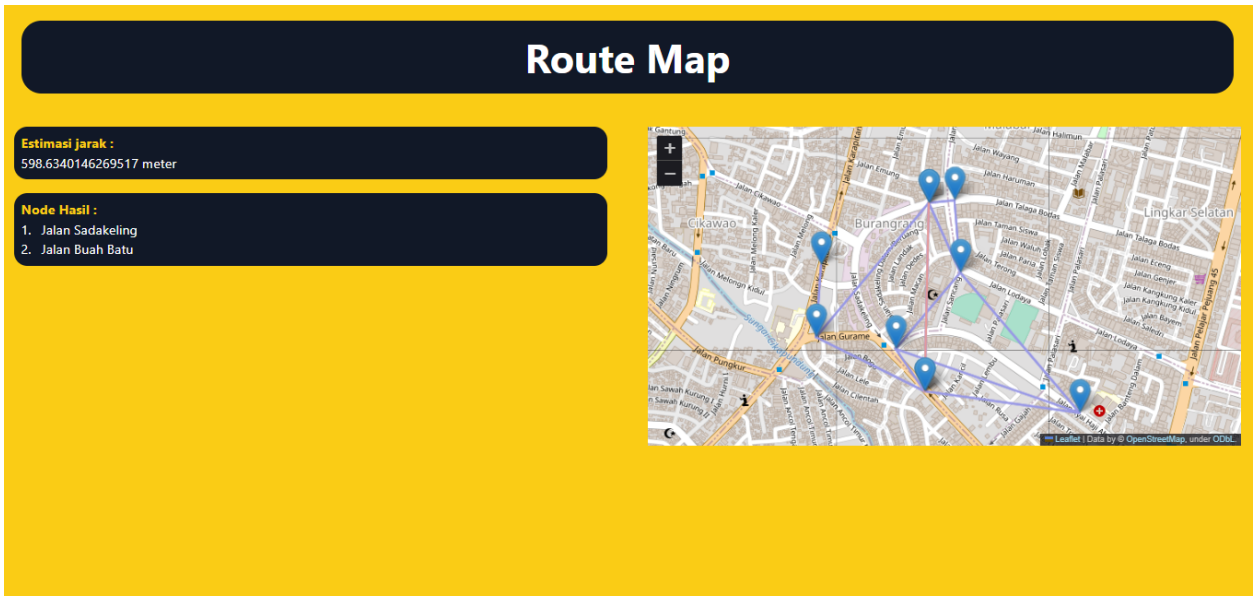




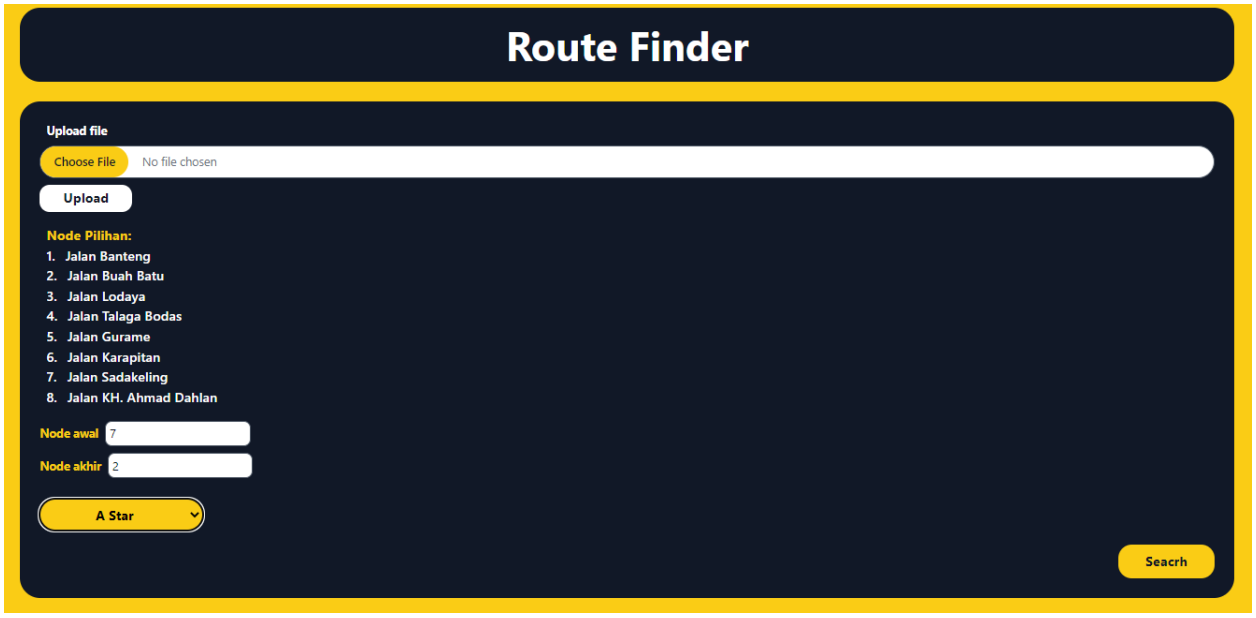
4.2 Buah Batu

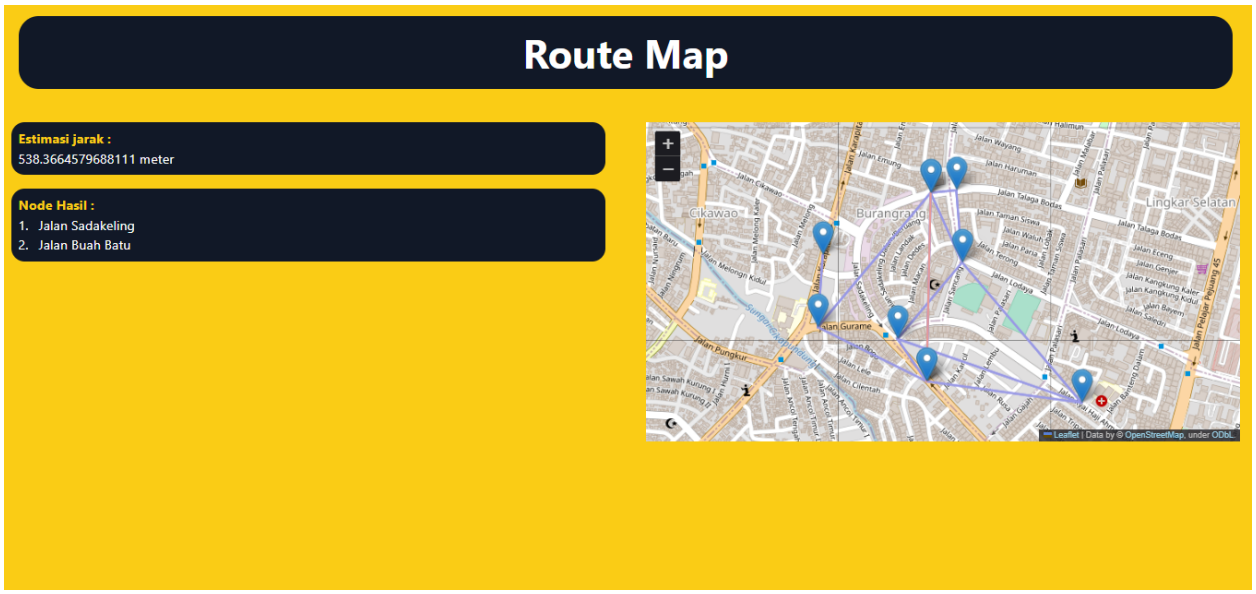
4.2.1 buahbatu.txt dengan UCS Algorithm (7 ke 2)





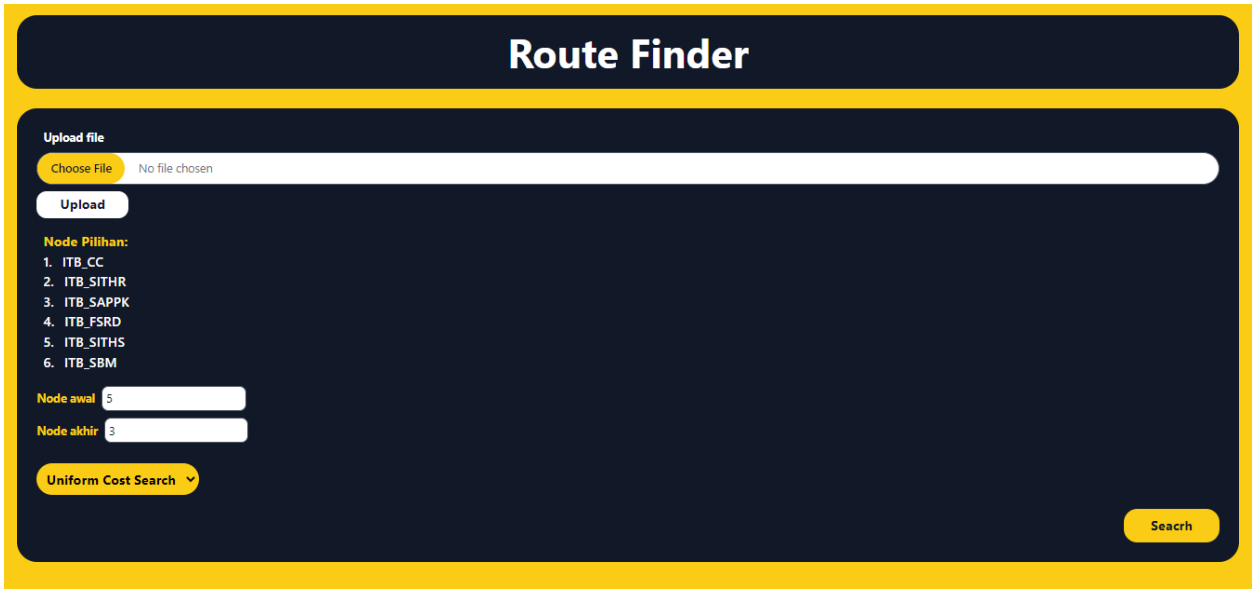
4.2.2 buahbatu.txt dengan A* Algorithm (7 ke 2)

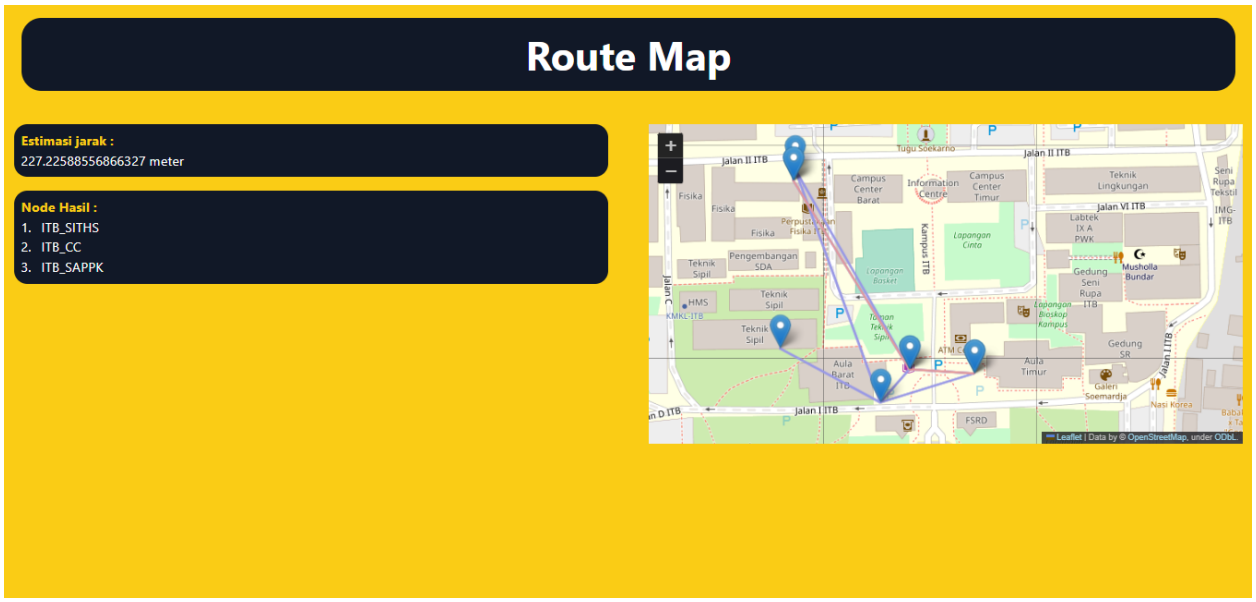




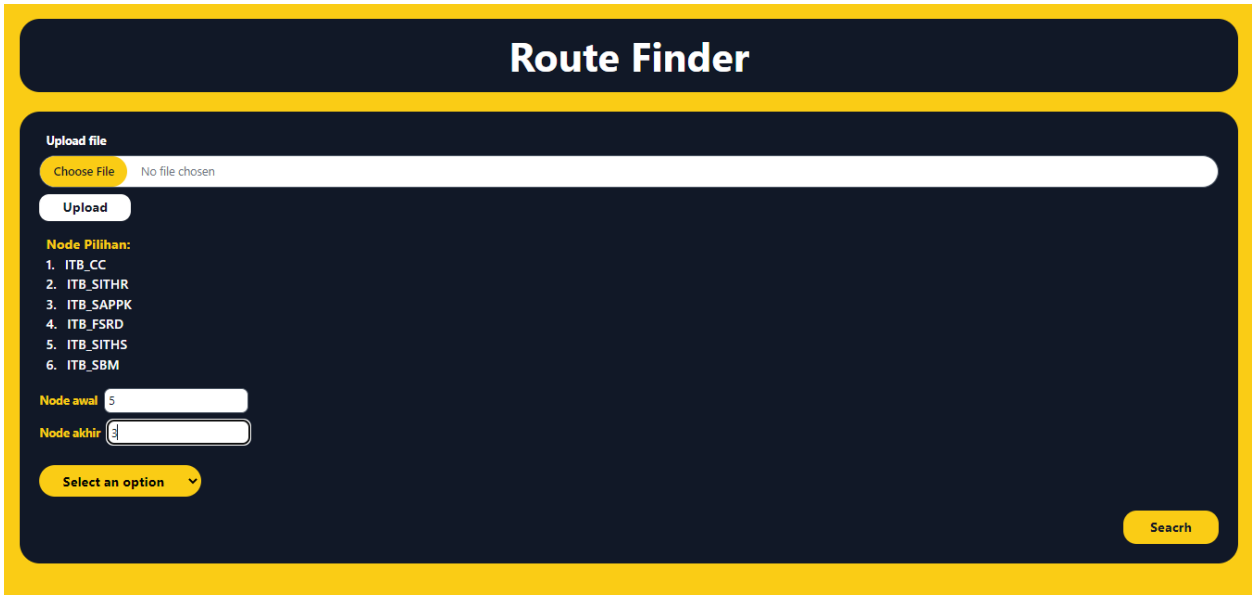
4.3 ITB

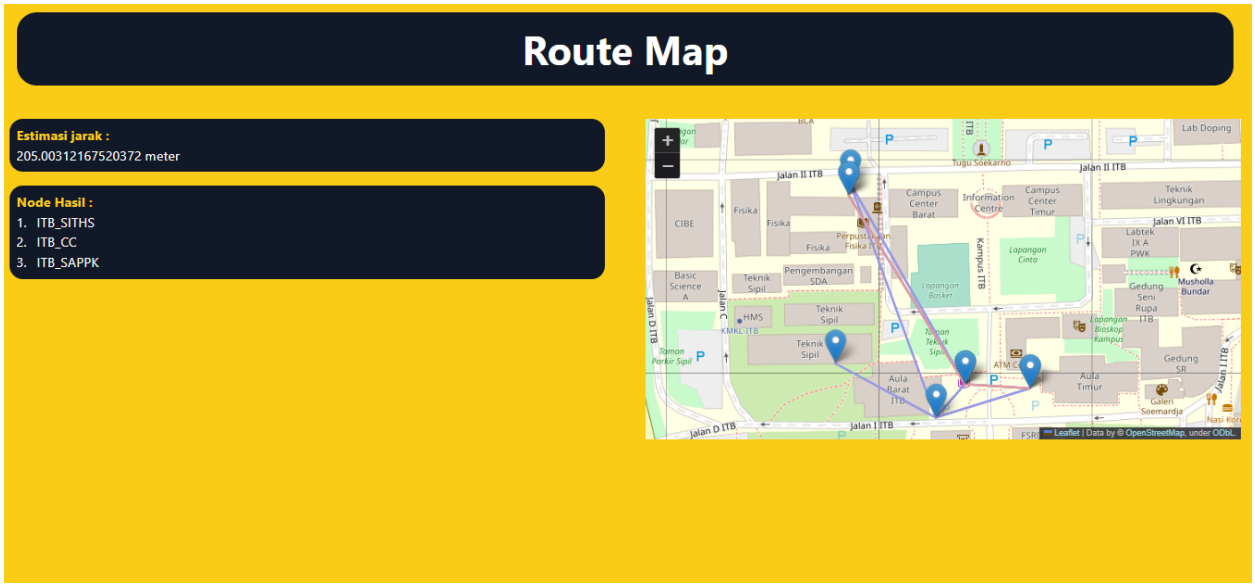
4.3.1 itb.txt dengan UCS Algorithm (5 ke 3)





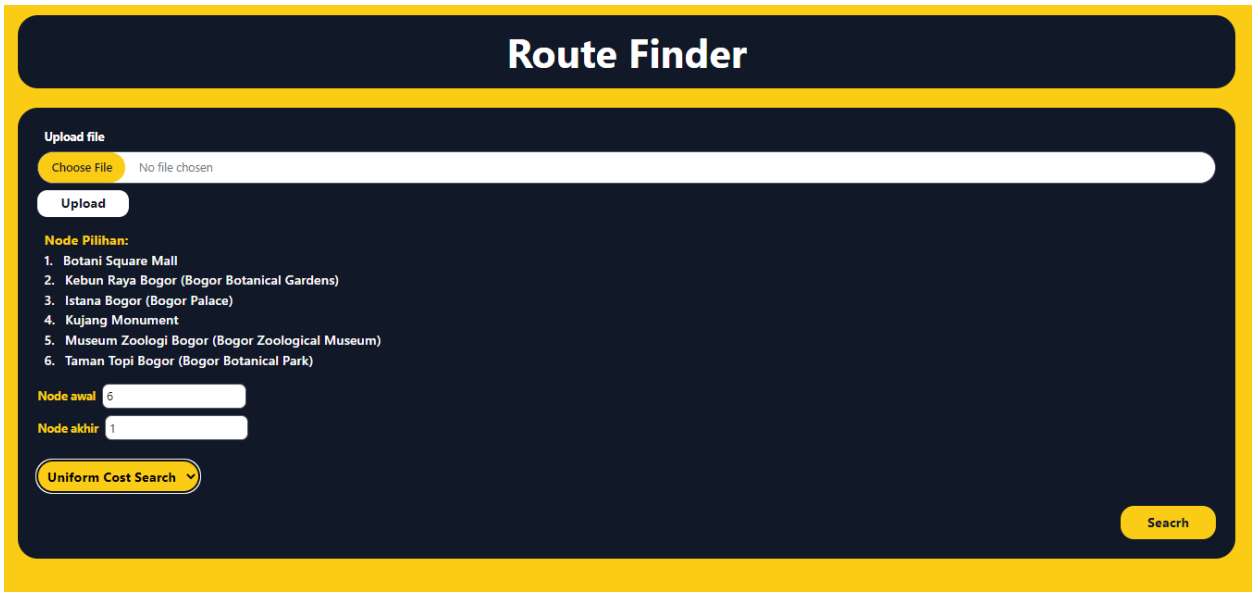
4.3.2 itb.txt dengan A* Algorithm (5 ke 3)

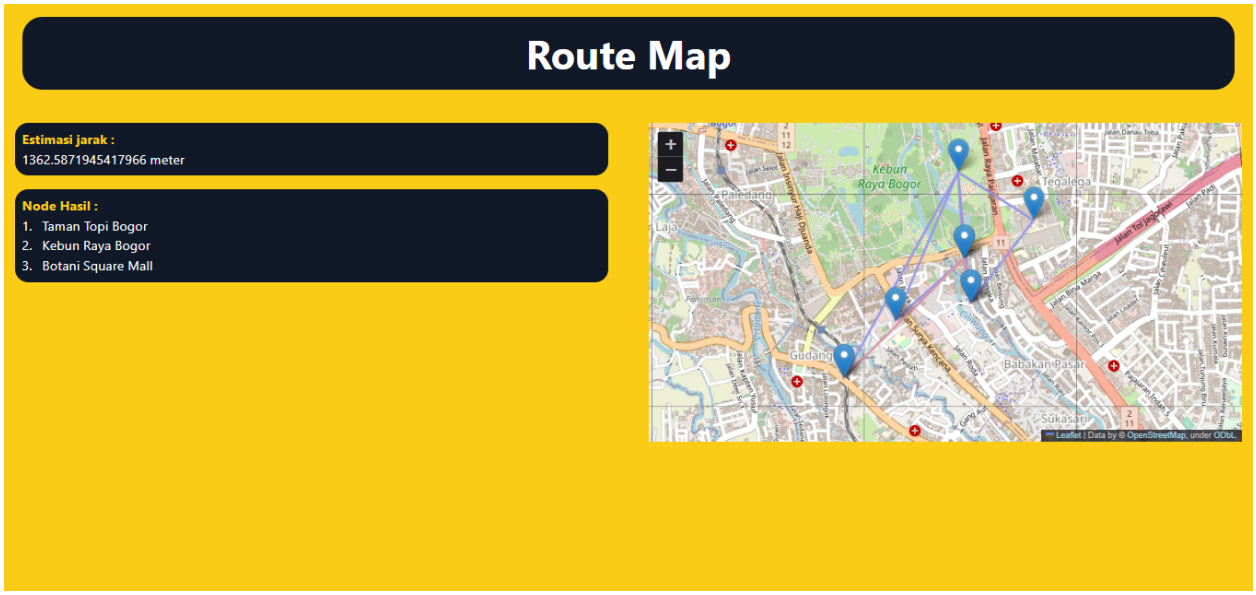




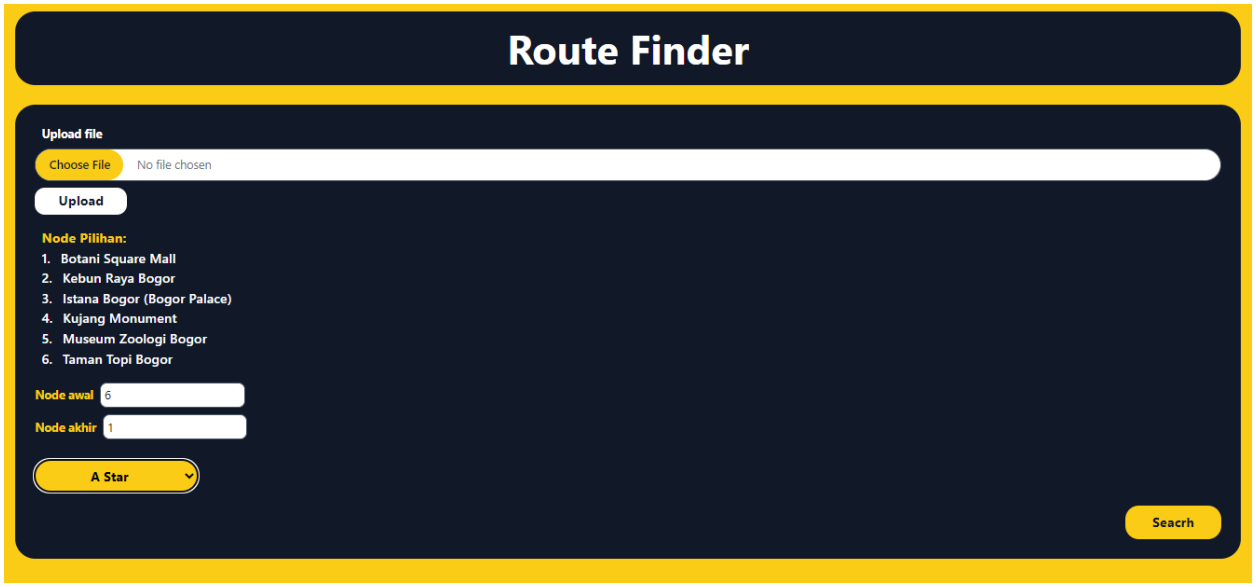
4.4 Bogor Kota

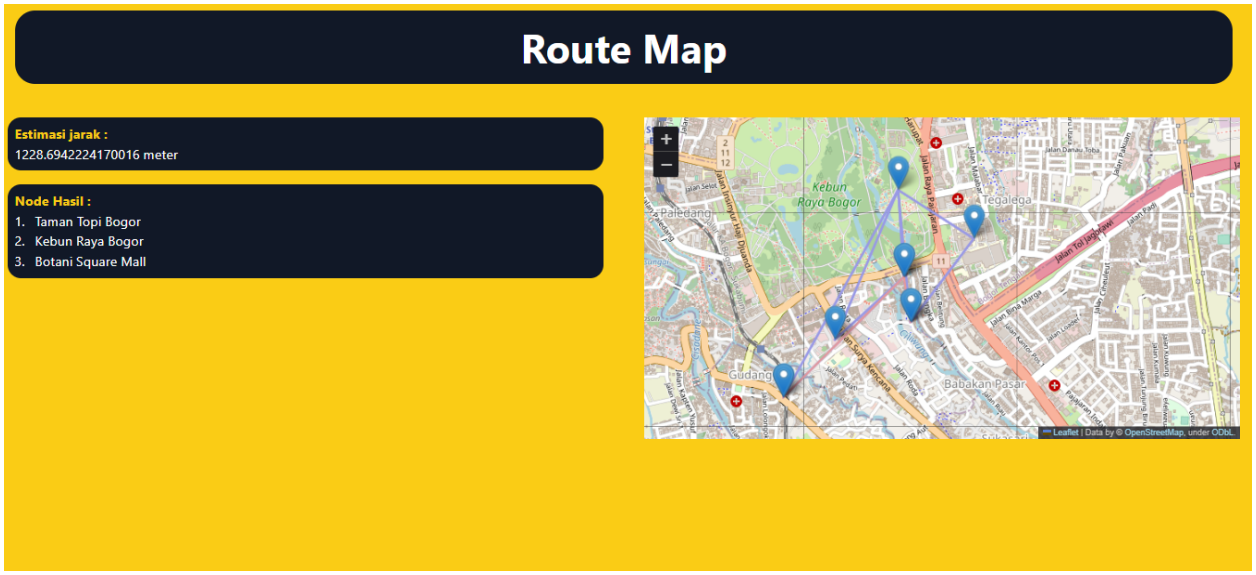
4.4.1 Bogor kota.txt dengan UCS Algorithm (6 ke 1)





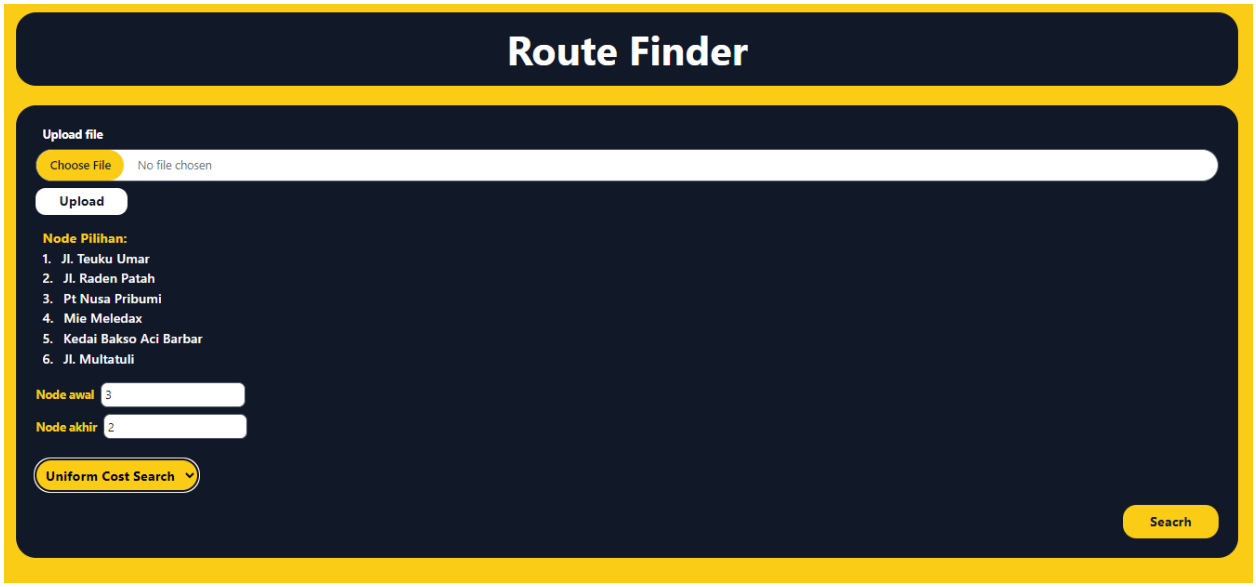
4.4.2 Bogor kota.txt dengan A* Algorithm (6 ke 1)

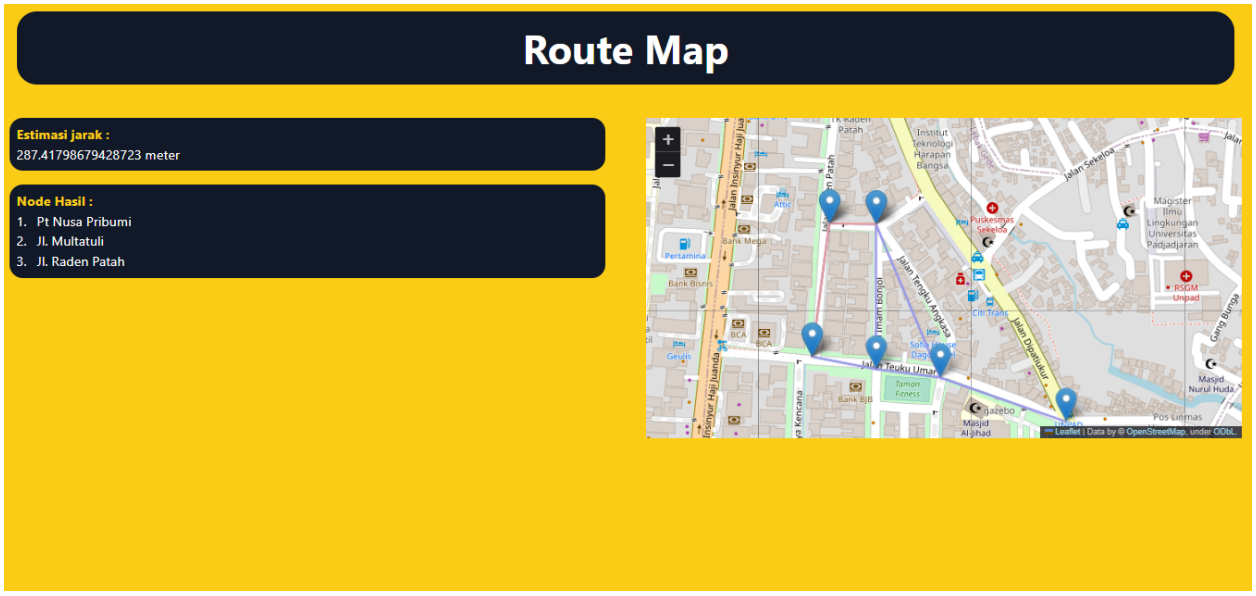




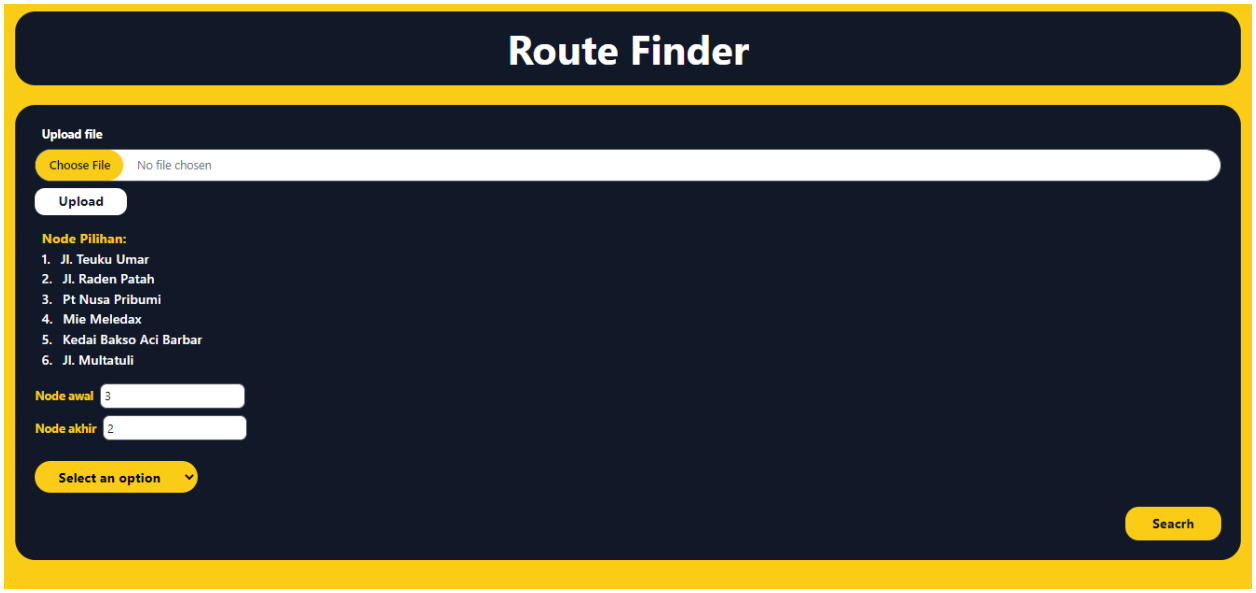
4.5 Dipatiukur

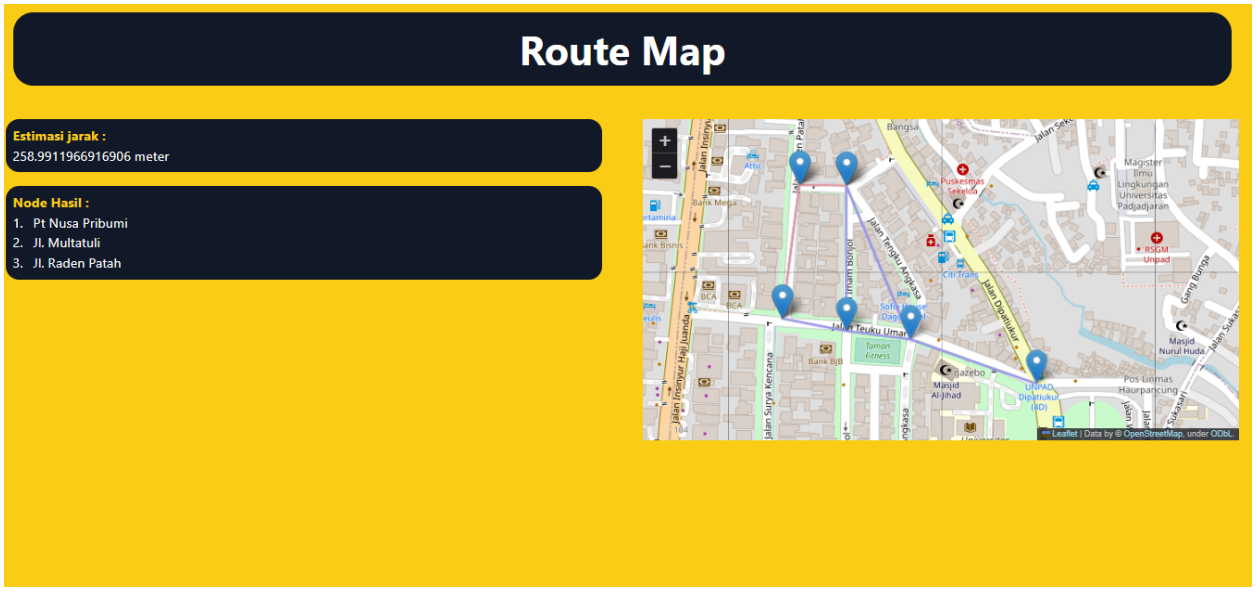
4.5.1 Dipatiukur.txt dengan UCS Algorithm (3 ke 2)





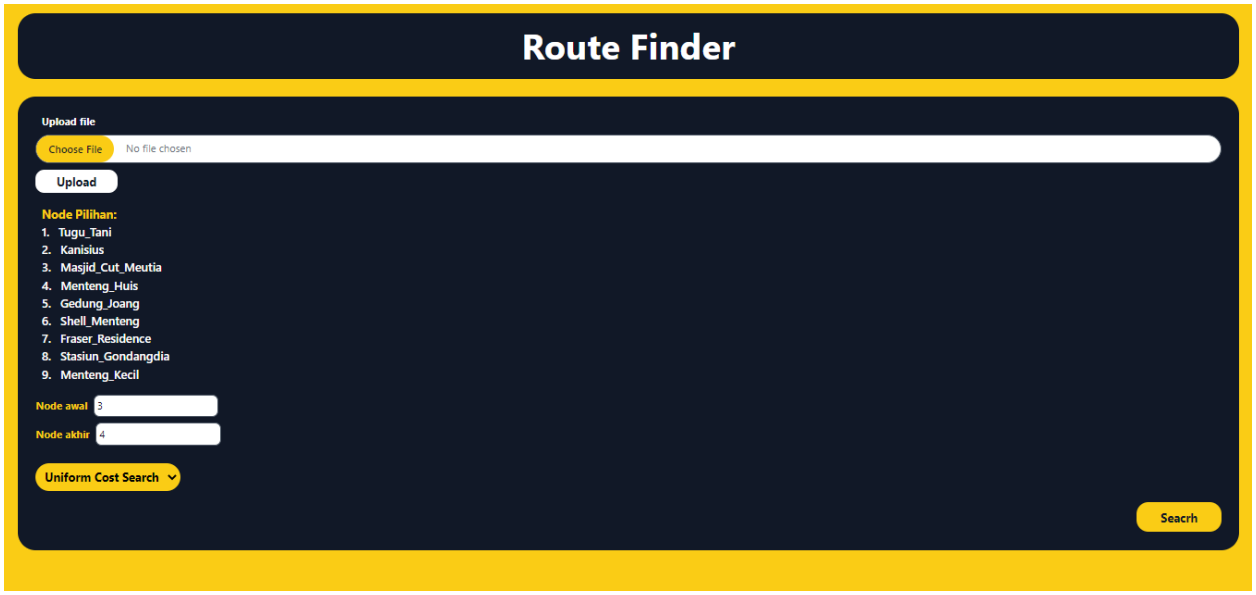
4.5.2 Dipatiukur.txt dengan A* Algorithm (3 ke 2)

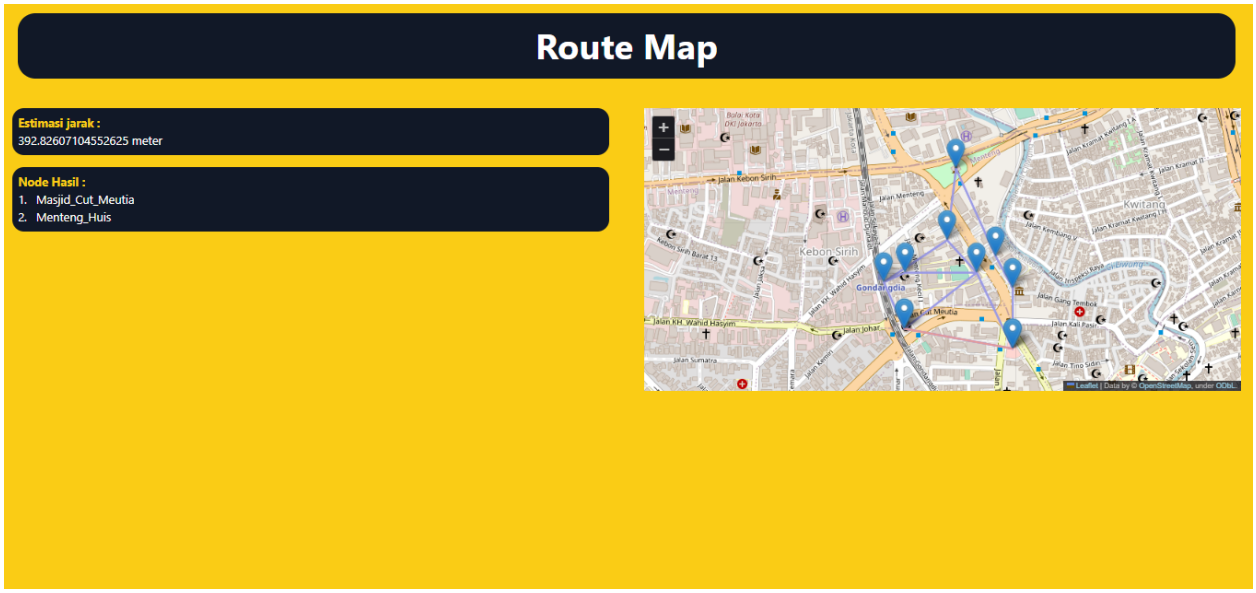




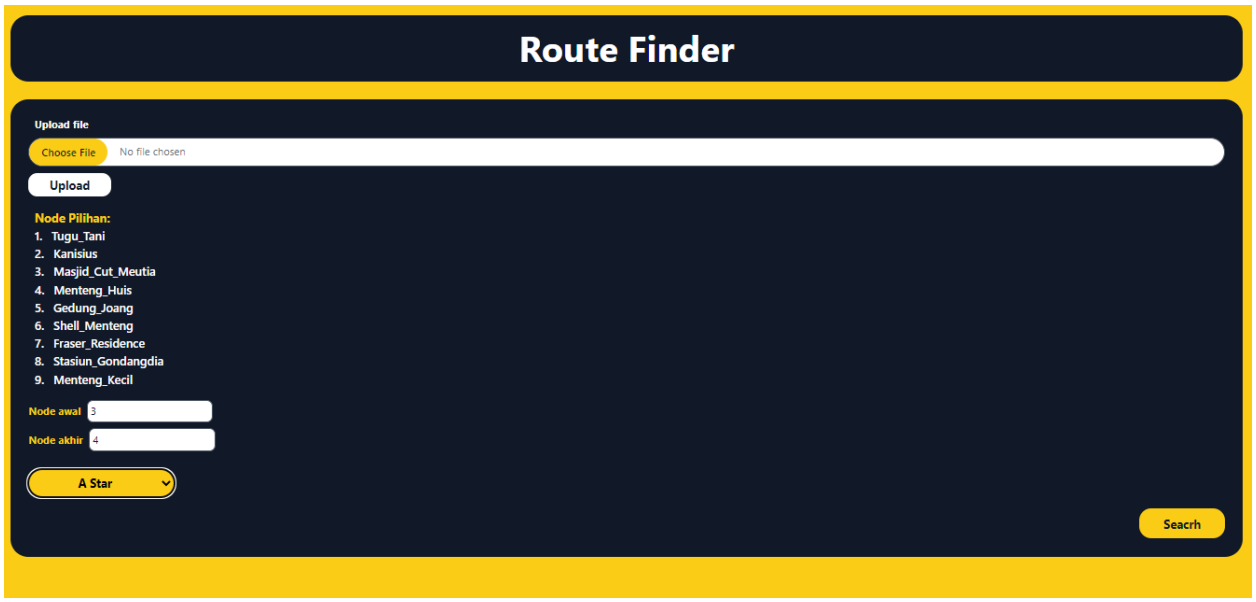
4.7 Jakarta

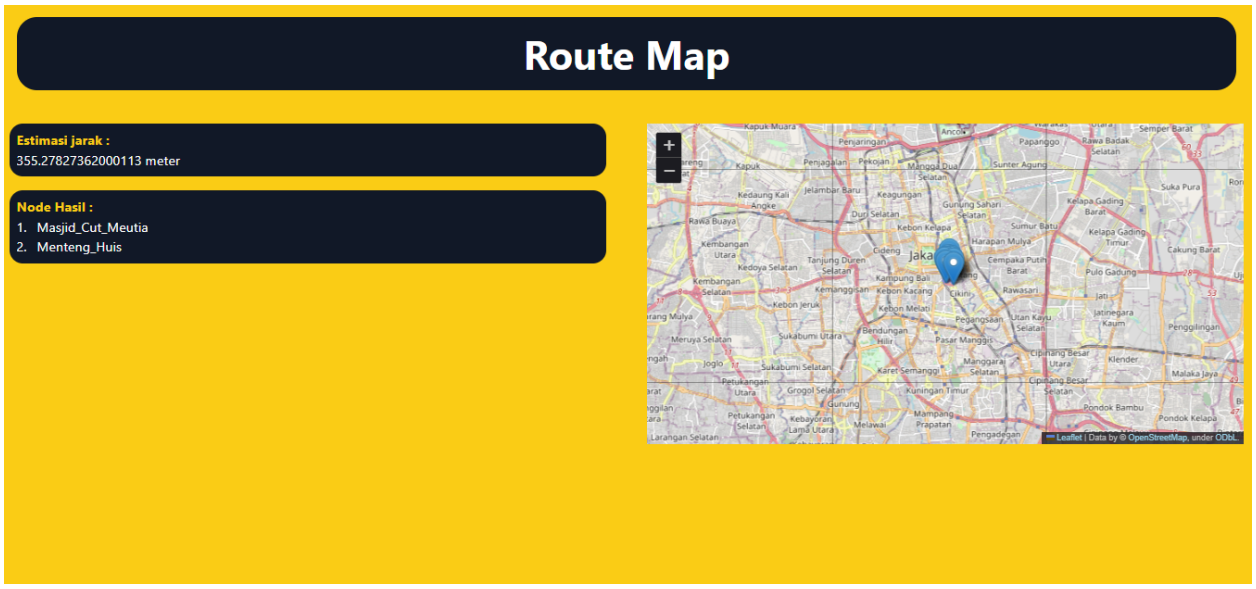
4.6.1 jkt.txt dengan UCS Algorithm (3 ke 4)





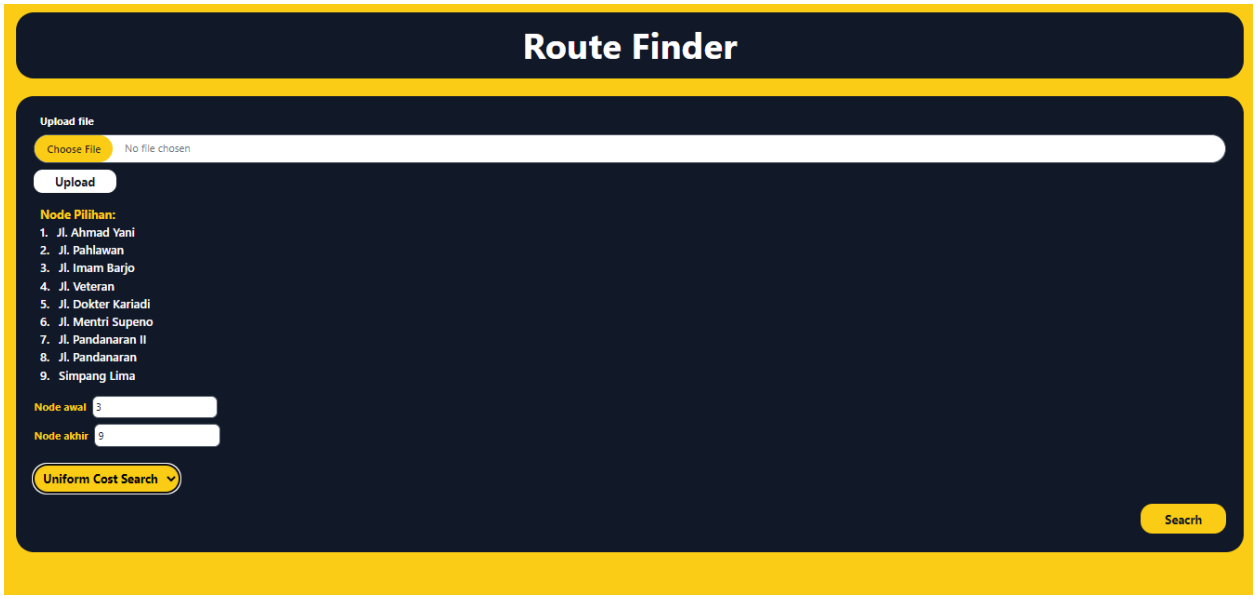
4.6.2 jkt.txt dengan A* Algorithm (3 ke 4)

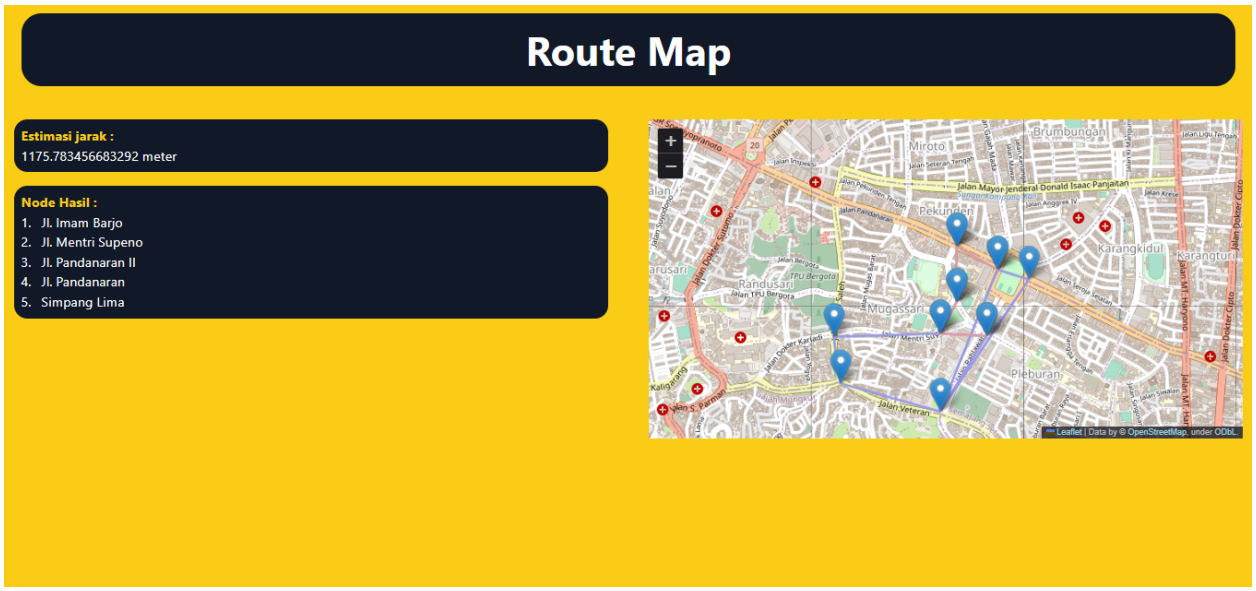




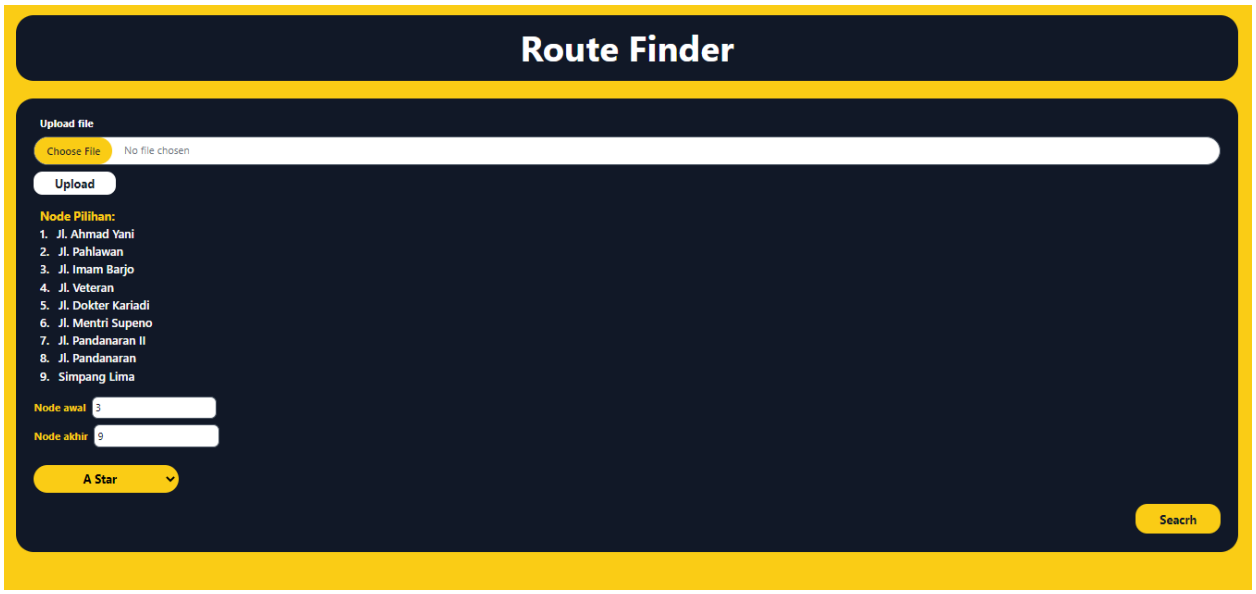
4.7 Mojokerto

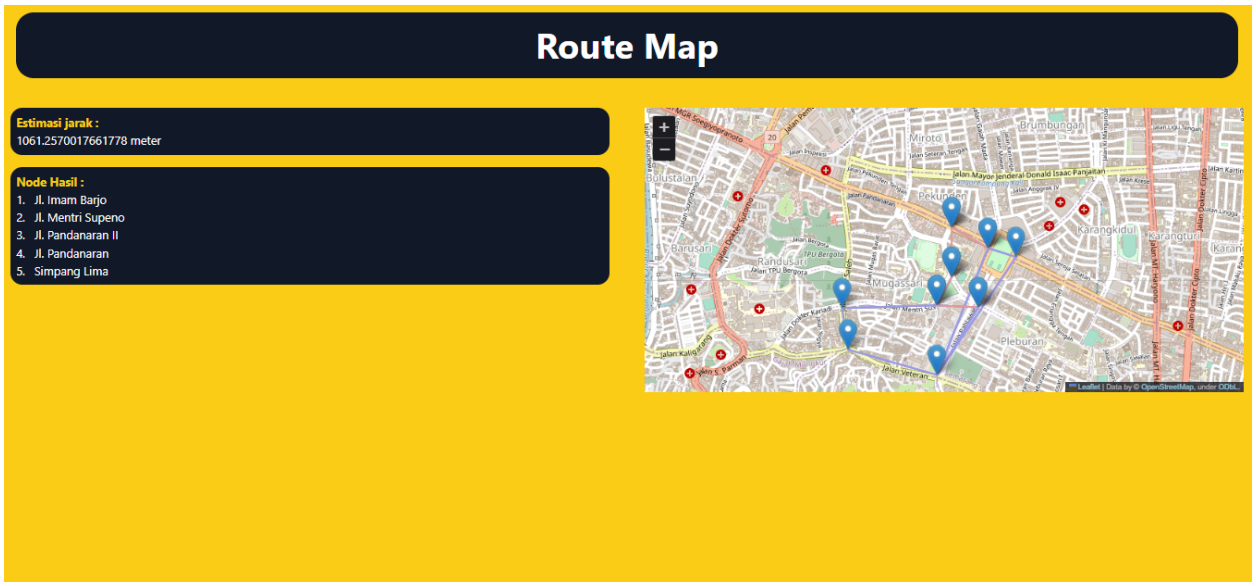
4.7.1 mjk.txt dengan UCS Algorithm (3 ke 9)





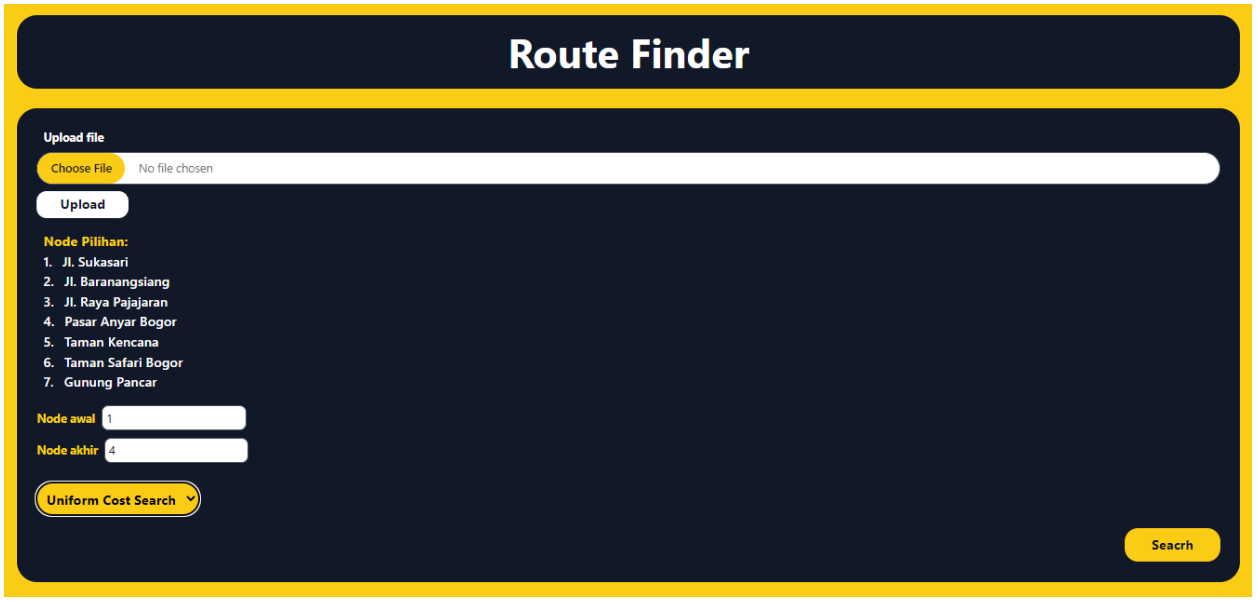
4.7.2 mjk.txt dengan A* Algorithm (3 ke 9)

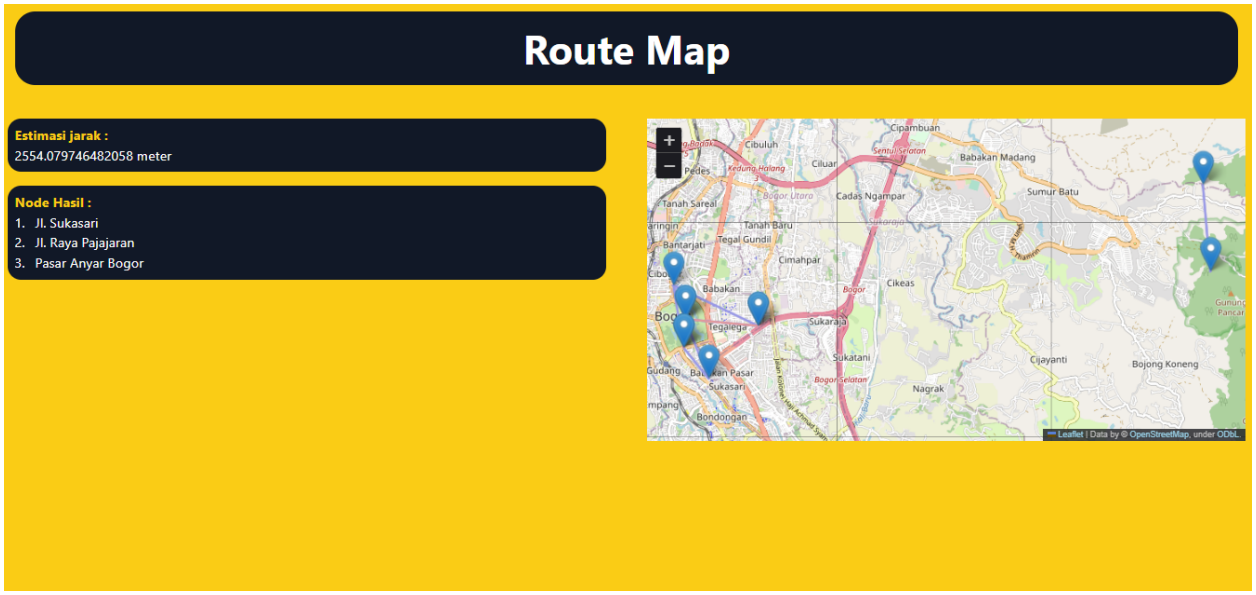




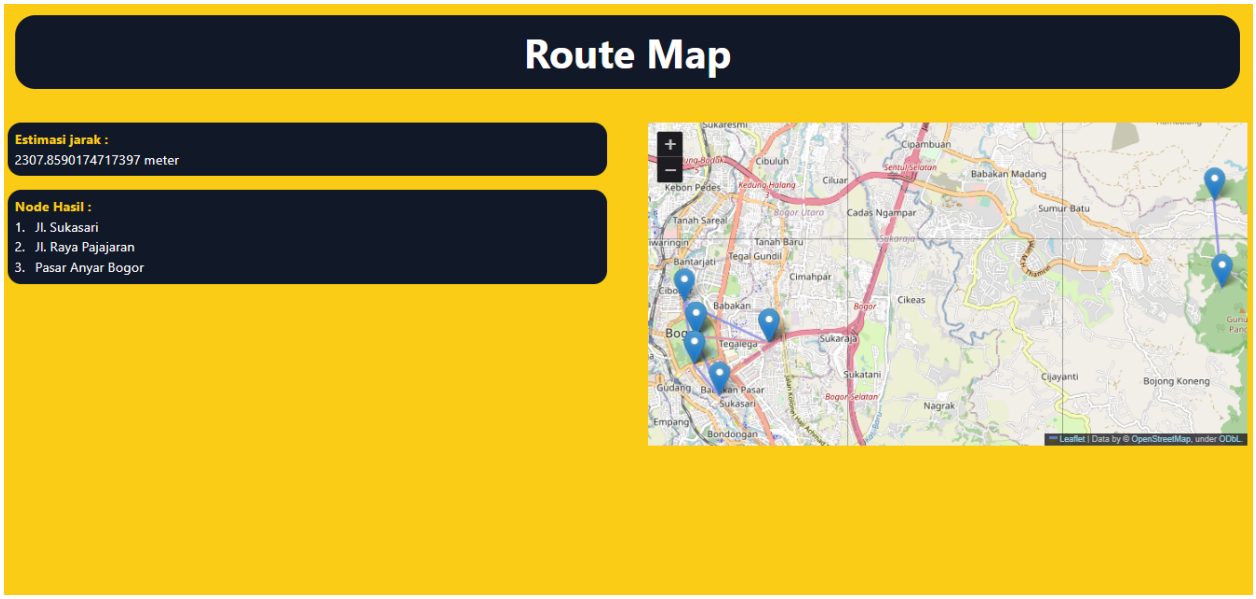
4.8 Bogor Barat

4.8.1 Bogor barat.txt dengan UCS Algorithm (1 ke 4)





4.8.2 Bogor barat.txt dengan A* Algorithm (1 ke 4)



5. Lampiran dan Pranala

Link Github : https://github.com/henryanandsr/Tucil3_13521004_13521025

No	Spesifikasi program	Ya	Tidak
1	Program dapat menerima input graf	✓	
2	Program dapat menghitung lintasan terpendek dengan UCS	✓	
3	Program dapat menghitung lintasan terpendek dengan A*	✓	
4	Program dapat menampilkan lintasan terpendek serta jaraknya	✓	
5	Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	✓	