# LAPORAN TUGAS KECIL 3 IF2211 STRATEGI ALGORITMA SEMESTER II TAHUN 2022/2023

# IMPLEMENTASI ALGORITMA UCS dan A* UNTUK MENENTUKAN LINTASAN TERPENDEK


**Disusun oleh :**

**Alex Sander** **13521061**

**Yobel Dean Christopher** **13521067**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG**


**BANDUNG 2022**

# DAFTAR ISI

# BAB I

# DESKRIPSI MASALAH

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil 3 ini, anda diminta menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan (simpang 3, 4 atau 5) atau ujung jalan. Asumsikan jalan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (m atau km) antar simpul. Jarak antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.

Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago). Berdasarkan graf yang dibentuk, lalu program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*. Lintasan terpendek dapat ditampilkan pada peta/graf (misalnya jalan-jalan yang menyatakan lintasan terpendek diberi warna merah). Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan.

Spesifikasi program:

1. Program menerima input file graf (direpresentasikan sebagai matriks ketetanggaan berbobot), jumlah simpul minimal 8 buah.

2. Program dapat menampilkan peta/graf

3. Program menerima input simpul asal dan simpul tujuan.

4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.

5. Antarmuka program bebas, apakah pakai GUI atau command line saja.

# BAB II

# KODE PROGRAM

## A. Uniform-Cost Search (UCS)

```python
def get_neighbors(graph, current,visited):
    neighbors = []

    # Cek tetangga-tetangga di atas, bawah, kiri, dan kanan
    for i in range(len(graph[0])):
        if (i!=current and graph[current][i] != 0 and (i not in visited)):
            neighbors.append(i)

    return neighbors
```

Fungsi get_neighbors akan mengembalikan sebuah array yang berisi nama node/simpul yang belum pernah dikunjungi sebelumnya.

```python
def ucs(dist, graph, start, goal):

    # Inisialisasi queue dan visited
    queue = []
    visited = set()

    # Masukkan titik awal ke queue
    heapq.heappush(queue, (0, start, [start]))

    while queue:

        # Pop titik dengan nilai f(n) terkecil dari queue
        cost, current, path = heapq.heappop(queue)


        # Jika titik saat ini adalah titik tujuan, maka selesai
        if current == goal:
            return cost, path

        # Jika titik saat ini belum pernah dikunjungi, tambahkan ke visited
        if current not in visited:
            visited.add(current)

            # Cari semua tetangga dari titik saat ini
            for neighbor in get_neighbors(graph, current, visited):
                tmppath = path
                # Hitung nilai f(n) dari tetangga tersebut
                neighbor_cost = cost + graph[current][neighbor] * dist[current][neighbor]
                tmppath = tmppath + [neighbor]

                # Masukkan tetangga ke queue
                heapq.heappush(queue, (neighbor_cost, neighbor, tmppath))

    # Jika tidak ditemukan path dari titik awal ke titik tujuan
    return -1,[]
```

Fungsi ucs menginisialisasi sebuah queue yang akan berisi cost total, node yang ditempati sekarang, dan sebuah array yang berisi simpul-simpul yang telah dikunjungi untuk mencapai simpul yang ditempati sekarang, dan sebuah set yang

mengisi simpul-simpul yang sudah pernah dikunjungi. Kemudian akan dienqueue nilai awal, yaitu cost yang bernilai 0, simpul awal, dan array yang berisi simpul awal. Setelah itu, akan dilakukan looping selama queue tidak kosong berisi perintah sebagai berikut.

1. Dequeue/pop anggota dalam queue dengan nilai cost terendah.
2. Jika hasil dequeue/pop merupakan simpul tujuan, maka return cost dan path dan selesai. Jika bukan, lanjut ke langkah 3.
3. Jika hasil dequeue/pop belum pernah dikunjungi sebelumnya, tambahkan ke set visited
4. Mencari semua tetangga yang dimiliki oleh hasil dequeue/pop dan menambahkannya ke queue.

**B. A* Algorithm**

```python
def get_neighbors(graph, current,visited):
    neighbors = []

    # Cek tetangga-tetangga di atas, bawah, kiri, dan kanan
    for i in range(len(graph[0])):
        if (i!=current and graph[current][i] != 0 and (i not in visited)):
            neighbors.append(i)

    return neighbors
```

Fungsi get_neighbors akan mengembalikan sebuah array yang berisi nama node/simpul yang belum pernah dikunjungi sebelumnya.

```python
def astar(strdist, graph, start, goal):

    # Inisialisasi queue dan visited
    queue = []
    visited = set()

    # Masukkan titik awal ke queue
    heapq.heappush(queue, (strdist[goal][start], 0, start, [start]))

    while queue:

        # Pop titik dengan nilai f(n) terkecil dari queue
        totalcost, cost, current, path = heapq.heappop(queue)


        # Jika titik saat ini adalah titik tujuan, maka selesai
        if current == goal:
            return totalcost, cost, path

        # Jika titik saat ini belum pernah dikunjungi, tambahkan ke visited
        if current not in visited:
            visited.add(current)

            # Cari semua tetangga dari titik saat ini
            for neighbor in get_neighbors(graph, current, visited):
                tmppath = path
                # Hitung nilai f(n) dari tetangga tersebut
                neighbor_cost = cost + graph[current][neighbor] * strdist[current][neighbor]
                tmppath = tmppath + [neighbor]
                heuriscost = neighbor_cost + strdist[goal][neighbor]

                # Masukkan tetangga ke queue
                heapq.heappush(queue, (heuriscost, neighbor_cost, neighbor, tmppath))


    # Jika tidak ditemukan path dari titik awal ke titik tujuan
    return -1,[]
```

Fungsi astar menginisialisasi sebuah queue yang akan berisi cost heuristik, cost total, node yang ditempati sekarang, dan sebuah array yang berisi simpul-simpul yang telah dikunjungi untuk mencapai simpul yang ditempati sekarang, dan sebuah set yang mengisi simpul-simpul yang sudah pernah dikunjungi. Kemudian akan dienqueue nilai awal, yaitu cost heuristik, cost awla yang bernilai 0, simpul awal, dan array yang berisi simpul awal. Setelah itu, akan dilakukan looping selama queue tidak kosong berisi perintah sebagai berikut.

5. Dequeue/pop anggota dalam queue dengan nilai cost terendah.
6. Jika hasil dequeue/pop merupakan simpul tujuan, maka return cost dan path dan selesai. Jika bukan, lanjut ke langkah 3.
7. Jika hasil dequeue/pop belum pernah dikunjungi sebelumnya, tambahkan ke set visited
8. Mencari semua tetangga yang dimiliki oleh hasil dequeue/pop dan menambahkannya ke queue

## C. Visualisasi

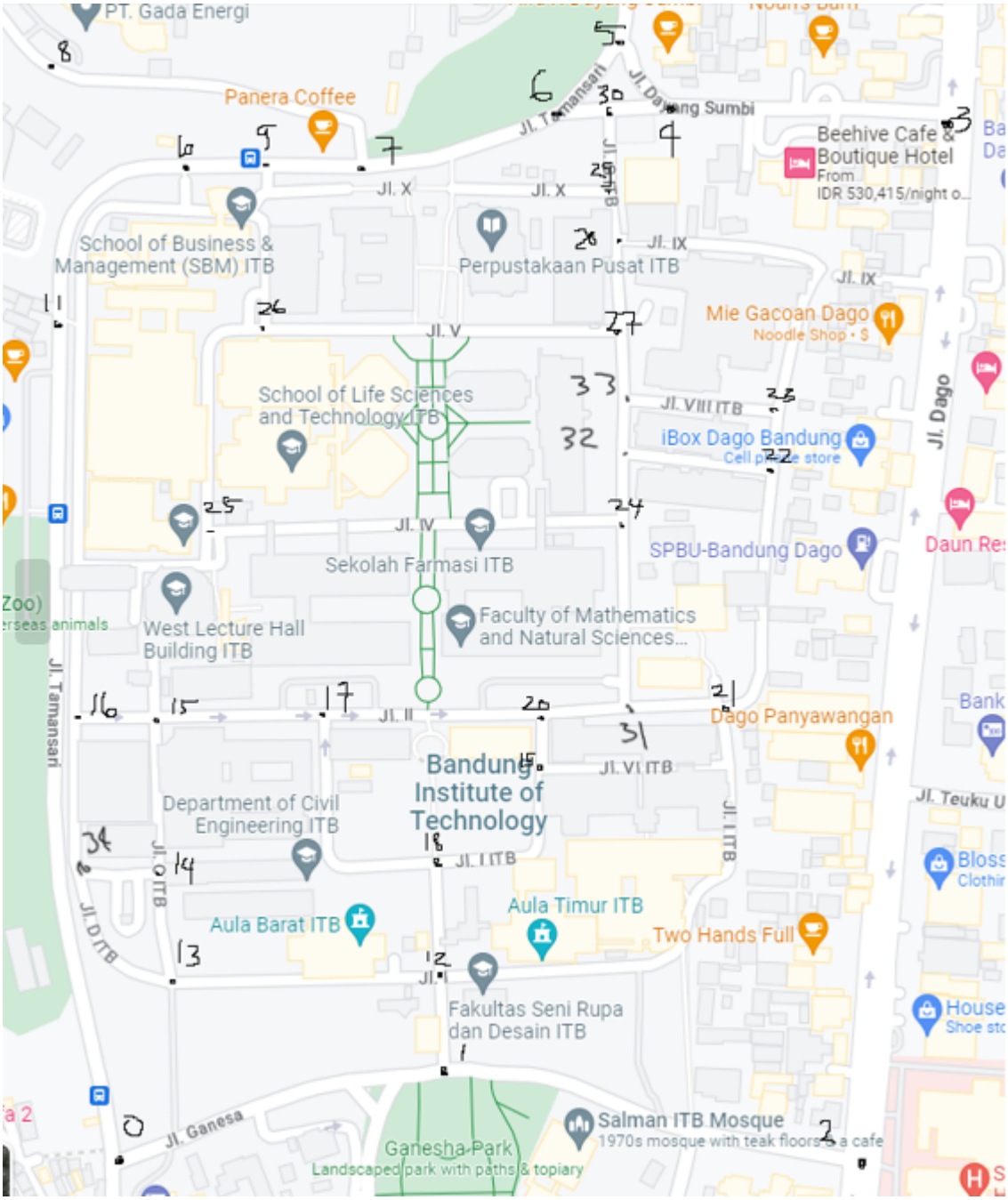
```python
import networkx as nx
import matplotlib.pyplot as plt

def drawGraph(graph, path, adj):
    # Inisialisasi graf
    G = nx.Graph()

    # Tambahkan simpul
    G.add_nodes_from(range(len(graph)))

    # Tambahkan edge
    for i in range(len(graph)):
        for j in range(i+1, len(graph)):
            if adj[i][j] != 0:
                G.add_edge(i, j, weight = graph[i][j])

    # Gambar graf dengan jalur terpendek
    pos = nx.spring_layout(G)
    nx.draw(G, pos, with_labels=True, node_size=250)
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels, font_size=5)
    path_edges = list(zip(path, path[1:]))
    nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='r', width=4)
    plt.show()
```

Visualisasi graf menggunakan matplotlib dan networkx. Hasil visualisasi menghasilkan graf sesuai dengan input, namun secara acak meletakkan simpul-simpulnya.

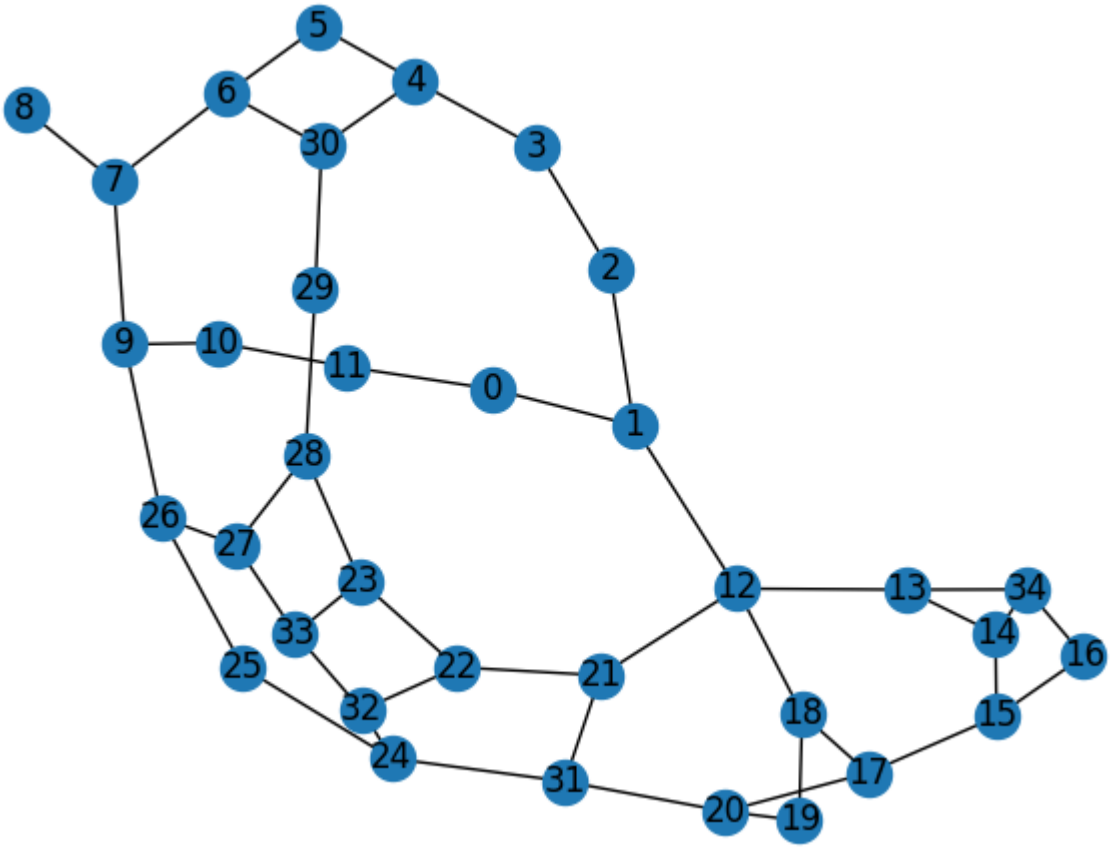
# BAB III

## TEST CASE

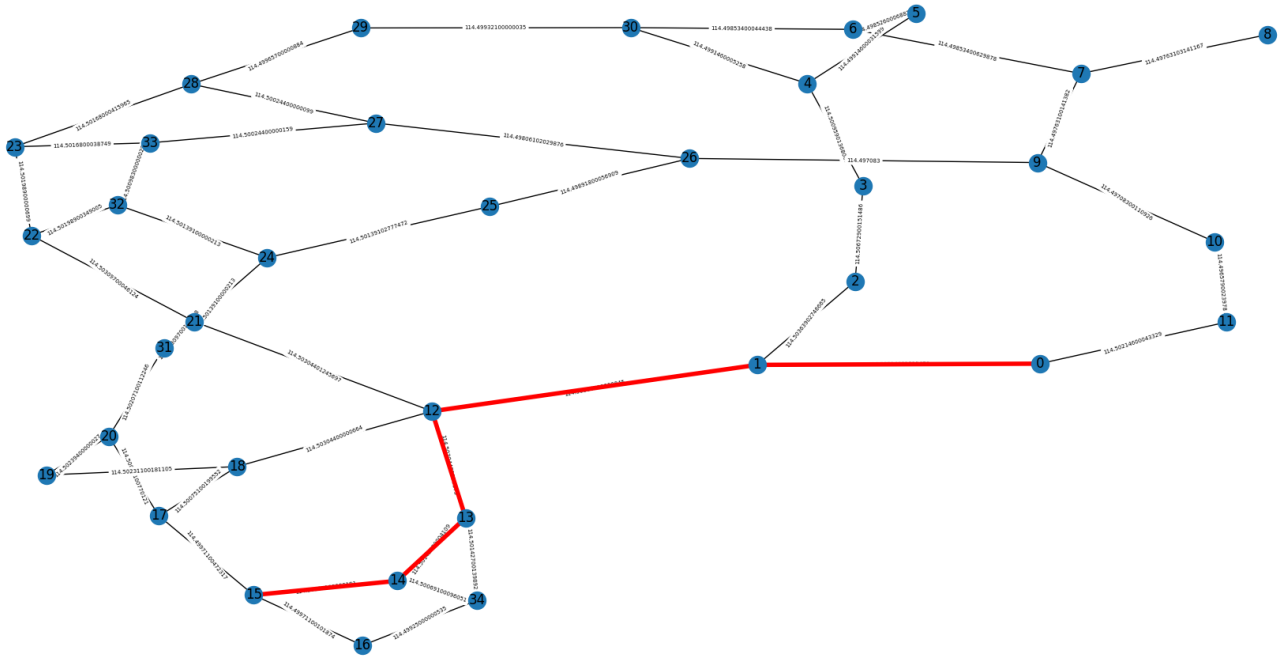
Test case yang diujikan pada program menggunakan peta sekitar ITB dengan simpul-simpul sebagai berikut. File test case simpul disimpan dalam “test/itb.txt” (Untuk menjalankan program pada terminal, cukup input “itb.txt”). Gambar berikut ini tersimpan dalam “test/itbmap.png”.




Berikut adalah graf yang terbentuk tanpa memperdulikan posisi node pada peta dunia nyata. Gambar ini tersimpan dalam “test/itbmap.png”.



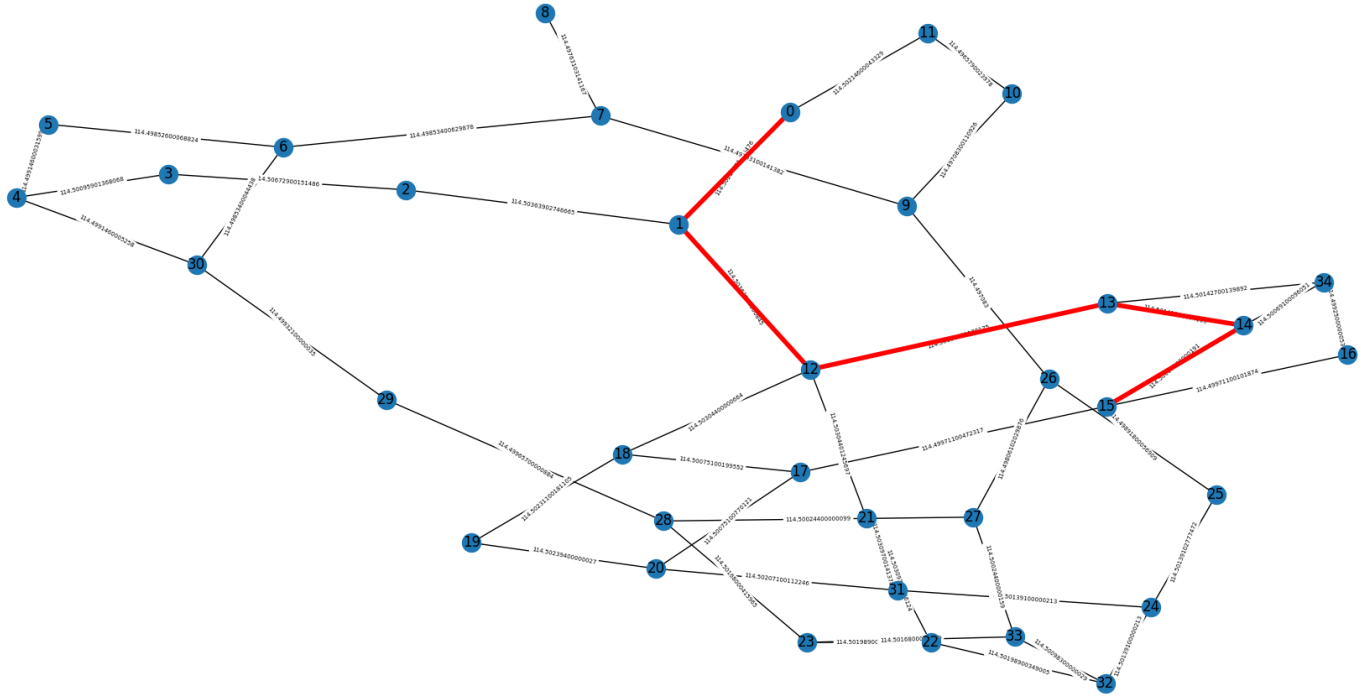

Start : 0
Goal : 15
UCS
Cost : 572.5109470299079
Path : 0 - 1 - 12 - 13 - 14 - 15



A*

Cost : 572.5109470299079

Path : 0 - 1 - 12 - 13 - 14 - 15



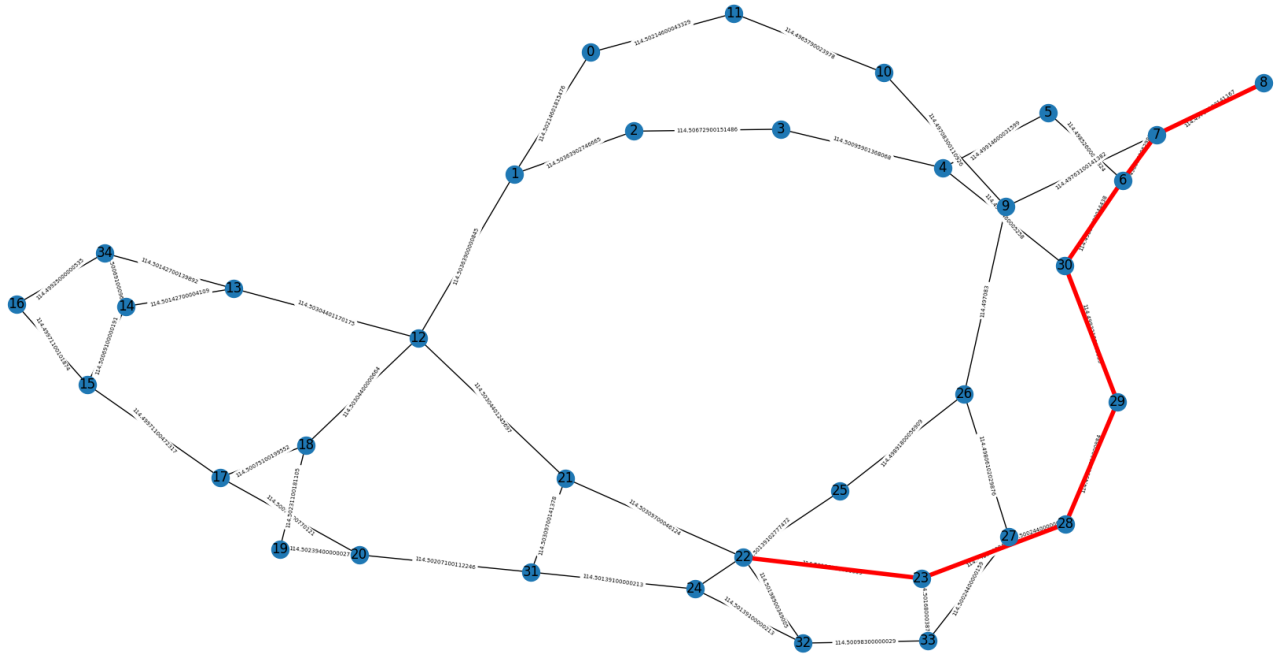
Start : 22

Goal : 8

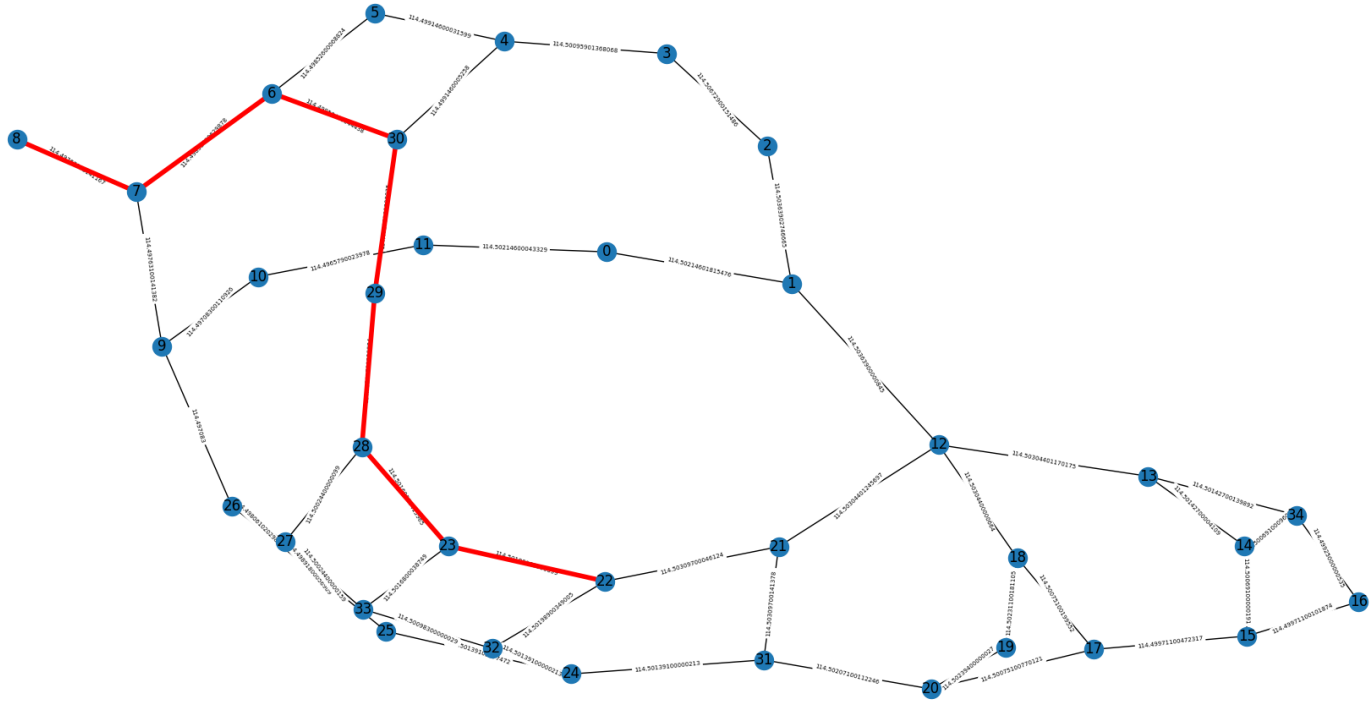
UCS

Cost : 801.4976270423306

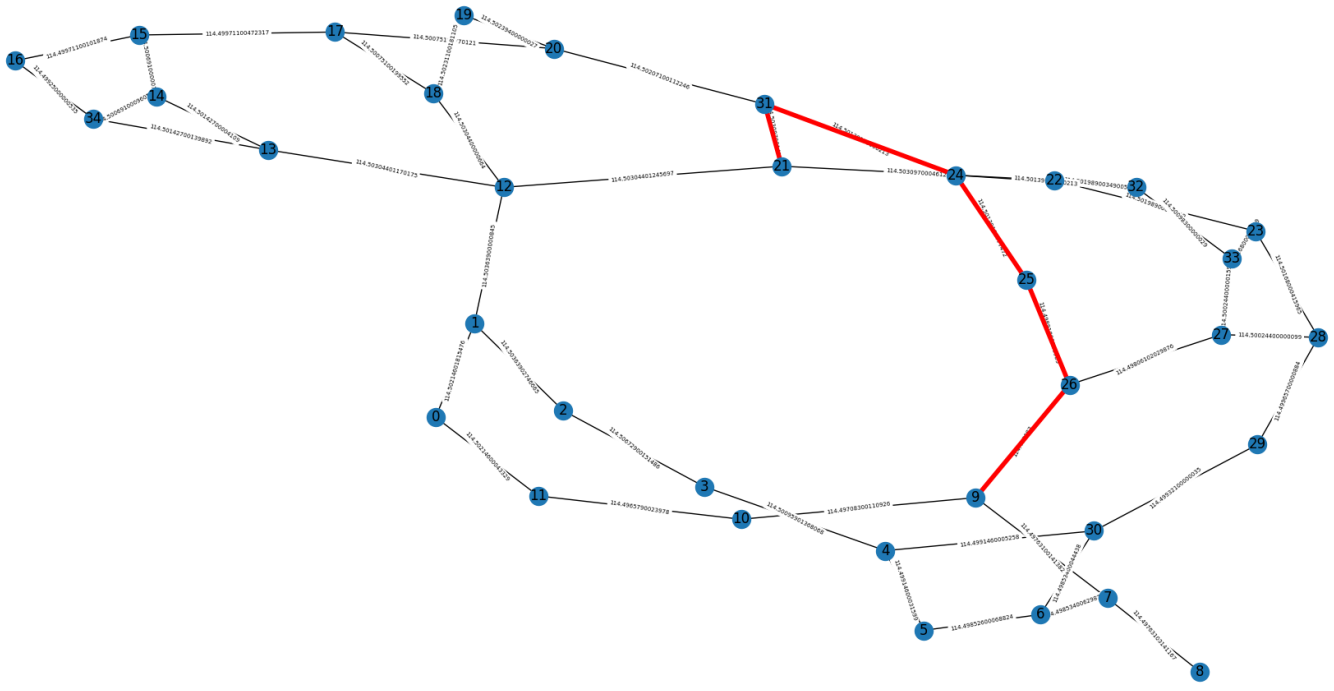
Path : 22 - 23 - 28 - 29 - 30 - 6 - 7 - 8



A*
Cost : 801.4976270423306
Path : 22 - 23 - 28 - 29 - 30 - 6 - 7 - 8



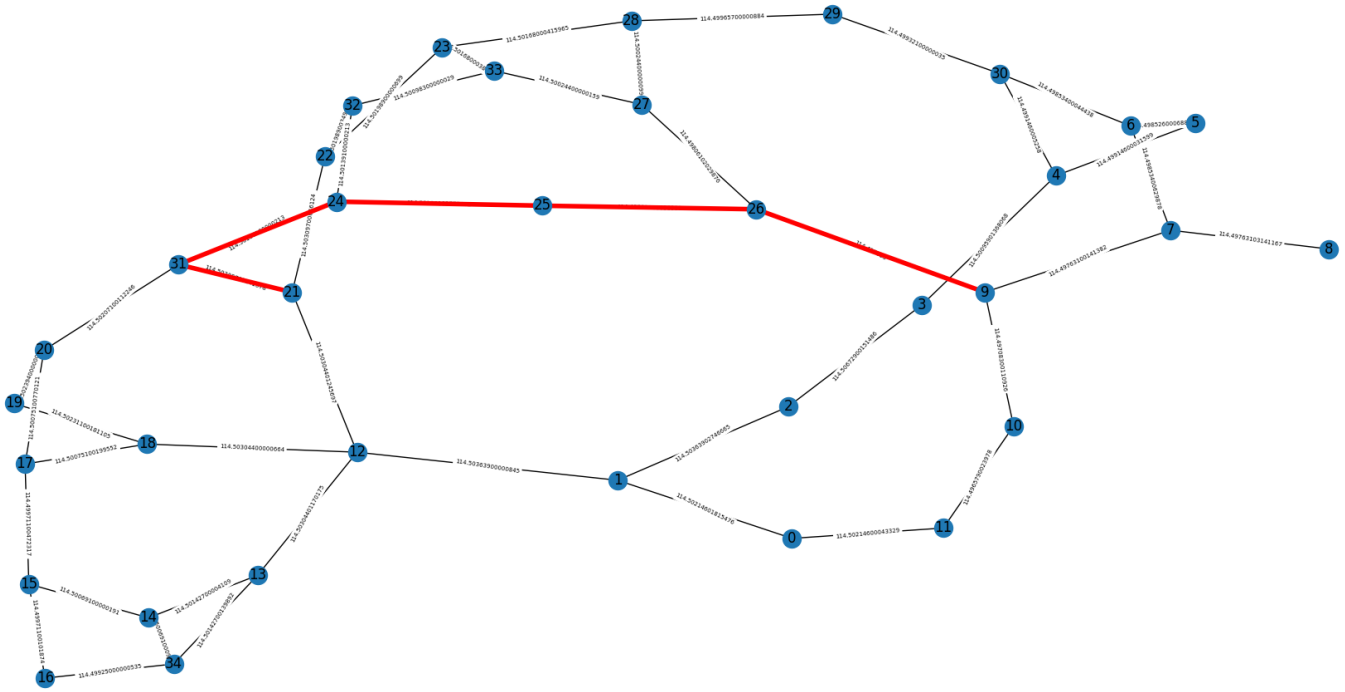
Start : 9
Goal : 21
UCS
Cost : 572.4979880297603
Path : 9 - 26 - 25 - 24 - 31 - 21



A*

Cost : 572.4979880297603

Path : 9 - 26 - 25 - 24 - 31 - 21



# BAB IV

# KESIMPULAN

Program untuk mencari rute terpendek dari suatu tempat ke tempat tujuan dapat dilakukan dengan berbagai macam algoritma. Algoritma yang sangat efisien dalam menyelesaikan persoalan ini adalah algoritma Uniform-cost search (UCS) dan algoritma A* (A star). Namun terdapat kelemahan dalam pengaplikasian algoritma-algoritma tersebut pada program ini, salah satunya adalah penggunaan titik untuk menentukan jarak antara suatu simpul dengan simpul yang lain. Hal ini dikarenakan jarak yang ditentukan adalah jarak garis lurus dari simpul ke simpul, bukan jalan sebenarnya. Jika dikaitkan dengan kasus dunia nyata dimana terdapat jalan-jalan dari suatu tempat ke tempat yang lain yang berupa jalan yang berbelok dan/atau melingkar, estimasi ini dapat mengakibatkan output yang salah jika tidak menempatkan simpul pada belokan jalan.

Pranala Github: https://github.com/maximatey/Tucil3_13521061_13521067

| | | |
|---|---|---|
| 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 | |