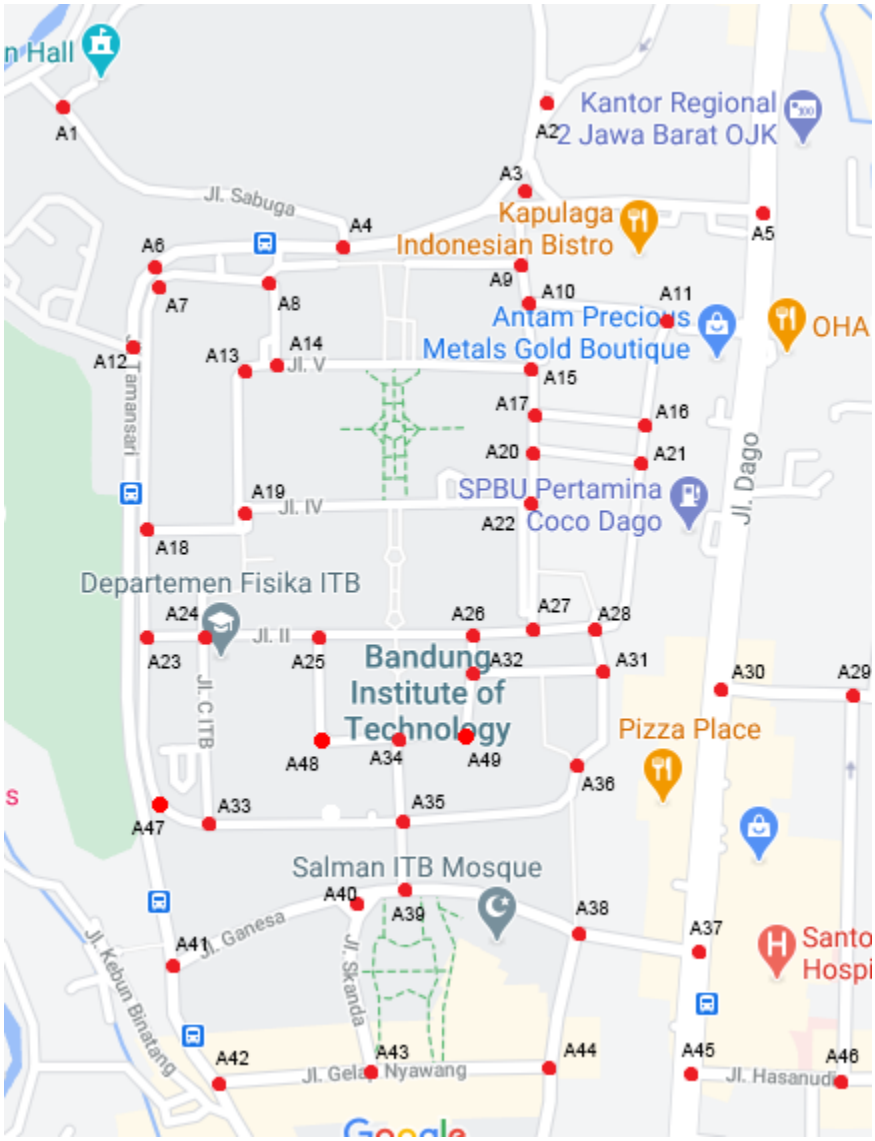
Laporan Tugas Kecil 3 IF2211:
Implementasi Algoritma A* untuk Menentukan Lintasan Terpendek

Oleh:
Muhammad Tito Prakasa - K1 - 13519007

I. Informasi Umum

Algoritma A* merupakan algoritma finding path heuristik yang memperhitungkan jarak dari root ke node saat ini (g(n)) dan jarak estimasi dari node saat ini ke node tujuan (h(n)). Dalam algoritma A*, h(n) atau estimasi jarak harus lebih kecil atau sama dengan jarak sesungguhnya dan dalam implementasi program ini hal tersebut dipenuhi dengan estimasi jarak diambil dari garis lurus antara dua node. Perhitungan jarak menggunakan haversine formula.

Peta yang digunakan:



II. *Source Code*

- Prosedur convertTxtToDict(dict graphDict):

```
def convertTxtToDict(graphDict):
    namaFile = input("Masukkan nama txt file yang ingin diolah (dalam folder test) : ")
    pathFile = "../test/" + namaFile
    objectFile = open(pathFile,'r')
    banyakNode = int(objectFile.readline())
    for i in range (banyakNode):
        tempDict = {}
        line = objectFile.readline()

        #ekstrak nama node
        namaNode = line.split(' ')[0]

        #ekstrak koordinat x dan y dari masing-masing node
        xCoordinate = float(line.split(' ')[1].split(',')[0])
        yCoordinate = float(line.split(' ')[1].split(',')[1])

        #masukkan koordinat
        tempDict = {}
        tempDict['coordinate'] = (xCoordinate,yCoordinate)

        #ekstrak simpul-simpul tetangga dari suatu node
        tempDictForEdge = {}
        listOfEdge = line.split(' ')[2].split(',')
        for edge in listOfEdge:
            edge = edge.strip('\n')
            tempDictForEdge[edge] = float(0)
        tempDict['edge'] = tempDictForEdge

        #masukkan ke dalam graphDict
        graphDict[namaNode] = tempDict
```

Prosedur yang mengisi graphDict (sebelumnya kosong) dengan graph dari sebuah file txt

- Fungsi haversineDistance(double x1,y1,x2,y2):

```
def haversineDistance(lat1,lon1,lat2,lon2):
    rad = math.pi/180
    radius = 6371
    dLat = rad * (lat2-lat1)
    dLon = rad * (lon2-lon1)

    a =  math.sin(dLat/2) * math.sin(dLat/2) + math.cos(rad*lat1) * math.cos(rad*lat2) *  math.sin(dLon/
    2) * math.sin(dLon/2)
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    return radius*c     #dalam kilometer
```

Fungsi yang menghitung jarak antara 2 koordinat menggunakan haversine formula (referensi stack overflow)

- prosedur isiJarakAntarnode(dict graphDict):

```
def isiJarakAntarNode(graphDict):
    for key in graphDict:

        #ambil latitude and longitude dari nodeSrc
        latSrc = graphDict[key]['coordinate'][0]
        lonSrc = graphDict[key]['coordinate'][1]

        for keyEdge in graphDict[key]['edge']:
            latDst = graphDict[keyEdge]['coordinate'][0]
            lonDst = graphDict[keyEdge]['coordinate'][1]

            graphDict[key]['edge'][keyEdge] = haversineDistance(latSrc,lonSrc,latDst,lonDst)
```

Prosedur yang mengisi jarak masing-masing node dengan tetangganya

- Fungsi isNoOtherWay(dict graphDict, dict activeNodes, string[] checkedNodes):

```
#fungsi untuk menentukan apakah sekumpulan activeNodes masih memiliki tetangga yang belum dikunjungi
atau tidak
def isNoOtherWay(graphDict, activeNodes, checkedNodes):
    for key in activeNodes:
        for keyEdge in graphDict[key]['edge']:
            if (keyEdge not in checkedNodes):
                return False
    return True
```

- Fungsi findNearestDistance(dict activeNodes):

```
def findNearestDistance(activeNodes):
    retVal = ""
    min = 100.0
    for key in activeNodes:
        if (activeNodes[key]['f(n)']<min):
            min = activeNodes[key]['f(n)']
            retVal = key
    return retVal
```

Fungsi yang mereturn nama node yang memiliki f(n) terkecil.

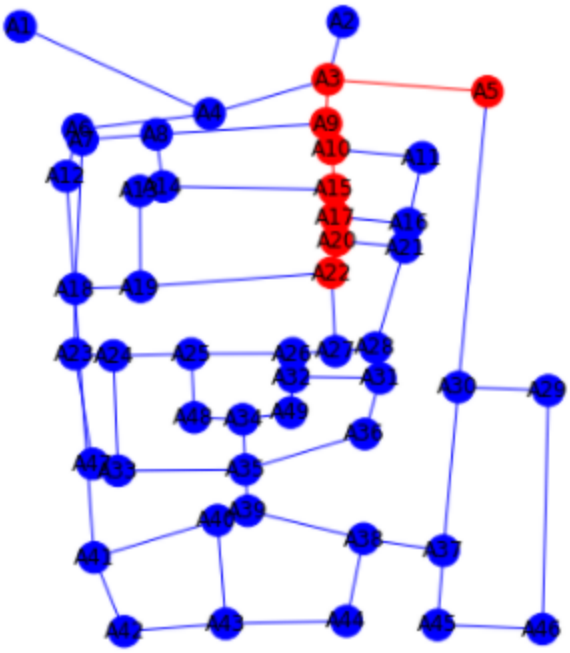
- Fungsi AStarAlgorithm(dict graphDict, string nodeSrc, string nodeDst):

```python
def AStarAlgorithm(graphDict, nodeSrc, nodeDst):
    checkedNodes = []   #list untuk menyimpan node-node yang telah dilalui
    activeNodes = {}    #dict untuk menyimpan node beserta g(n) yang sedang aktif
    foundSolusi = False

    ''' initial state '''
    checkedNodes.append(nodeSrc)
    initialDistance = haversineDistance(graphDict[nodeSrc]['coordinate'][0],graphDict[nodeSrc]['coordinate'][1],graphDict[nodeDst]['coordinate'][0],graphDict[nodeSrc]
    ['coordinate'][1])
    initialDict = {'f(n)' : initialDistance , 'jalur' : []}
    activeNodes[nodeSrc] = initialDict
    nodeNearest = findNearestDistance(activeNodes)

    ''' Looping '''
    while (not isNoOtherWay(graphDict,activeNodes,checkedNodes) and not foundSolusi):
        nodeNearest = findNearestDistance(activeNodes)
        if (nodeNearest==nodeDst):
            activeNodes[nodeNearest]['jalur'] = activeNodes[nodeNearest]['jalur'] + [nodeDst]
            foundSolusi = True
        else:
            for keyEdge in graphDict[nodeNearest]['edge']:
                if (keyEdge not in checkedNodes):
                    #tambahan checkedNodes
                    checkedNodes.append(keyEdge)

                    #hitung g(n)
                    gn = activeNodes[nodeNearest]['f(n)'] - haversineDistance(graphDict[nodeNearest]['coordinate'][0],graphDict[nodeNearest]['coordinate'][1],graphDict
                    [nodeDst]['coordinate'][0],graphDict[nodeDst]['coordinate'][1]) + graphDict[nodeNearest]['edge'][keyEdge]

                    #hitung h(n)
                    hn = haversineDistance(graphDict[keyEdge]['coordinate'][0],graphDict[keyEdge]['coordinate'][1],graphDict[nodeDst]['coordinate'][0],graphDict[nodeDst]
                    ['coordinate'][1])

                    #hitung f(n)
                    fn = gn + hn

                    updateJalur = activeNodes[nodeNearest]['jalur'] + [nodeNearest]
                    activeNodes[keyEdge] = {'f(n)':fn,'jalur':updateJalur}
            del activeNodes[nodeNearest]    #matikan node yang telah membangkitkan tetangganya
    if (foundSolusi):
        return activeNodes[nodeNearest]
    else:
        return {}
```

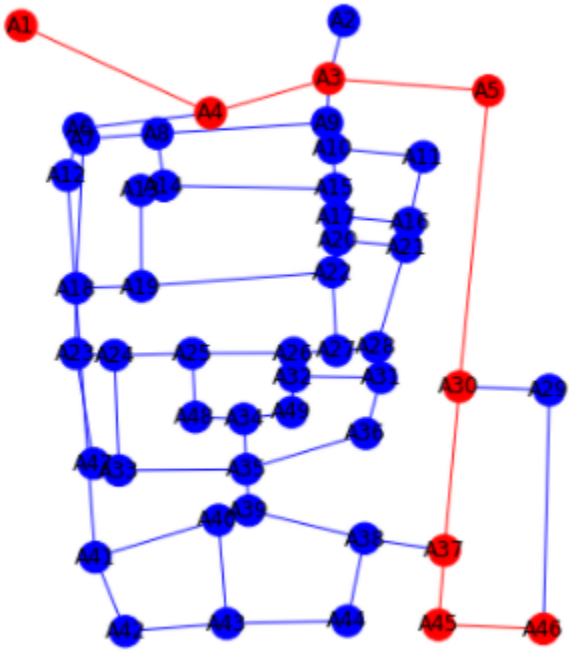
Algoritma yang mengolah graph, node asal, dan node tujuan yang mereturn lintasan terpendek dari node asal ke node tujuan. Jika tidak ditemukan lintasan maka mengembalikan list kosong.

III. *Test Case*

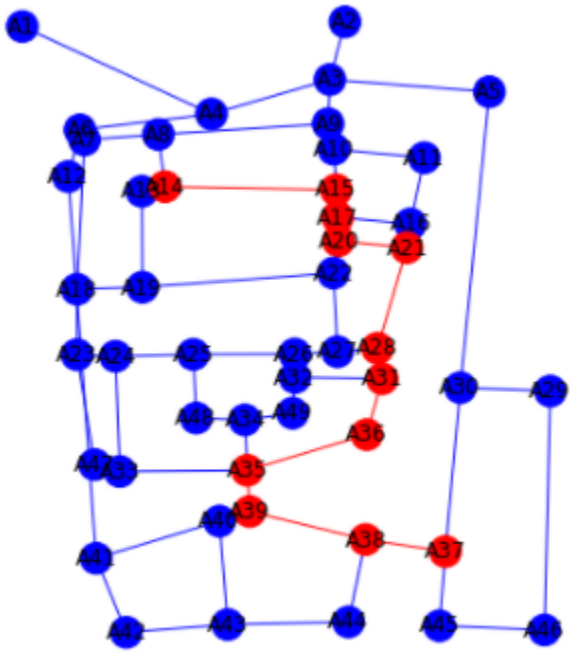
1. A5 ke A22 (petaITBDago.txt):



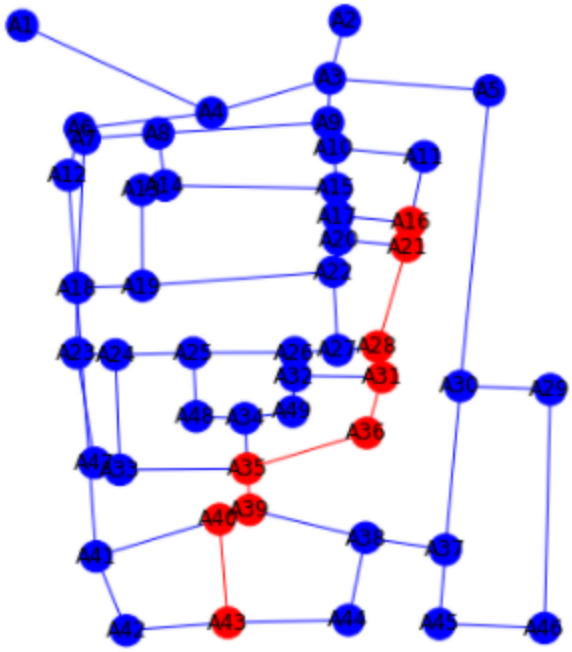
2. A1 ke A46 (petaITBDago.txt):



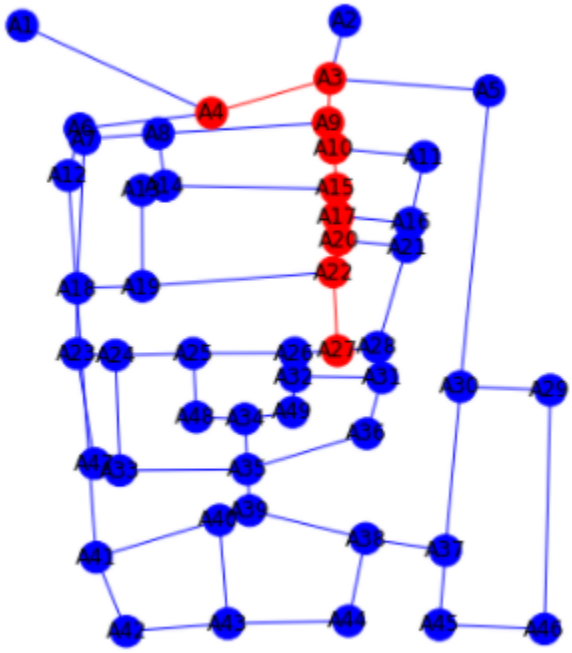
3. A14 ke A37 (petaITBDago.txt) :



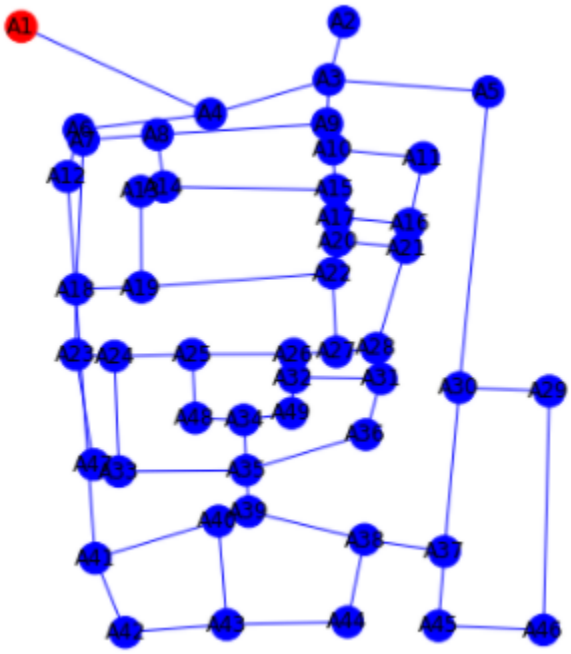
4. A16 ke A43 (petaITBDago.txt) :



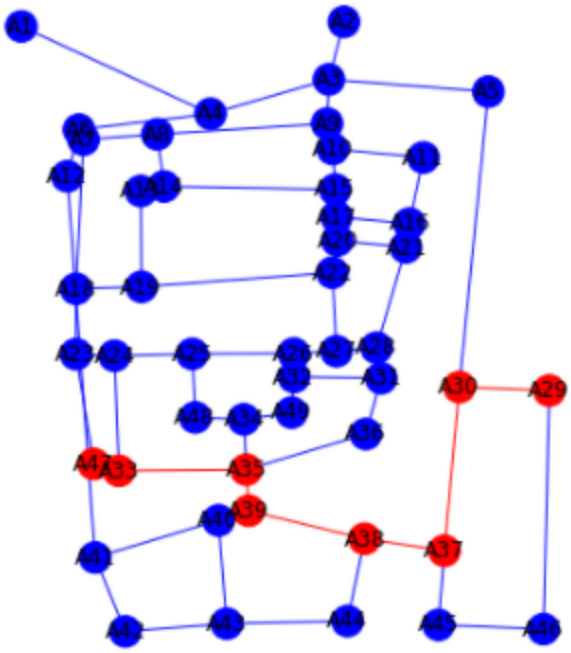
5. A4 ke A27 (petaITBDago.txt) :



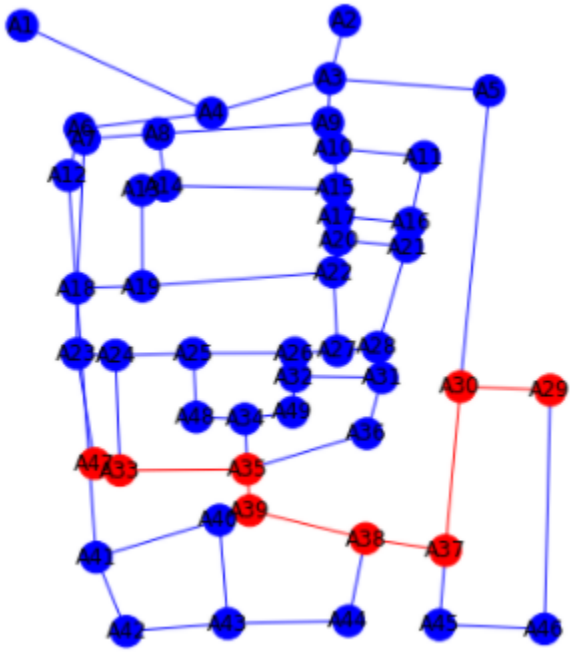
6. A1 ke A1 (petaITBDago.txt) :



7. A29 ke A47 (petaITBDago.txt) :



8. A47 ke A29 (petaITBDago.txt) :



IV. Alamat *Source Code*

grevicoc/Implementation-of-A-star-Algorithm (github.com)

V. Kesimpulan

Poin	Ya	Tidak
1. Program dapat menerima input graf	✔	
2. Program dapat menghitung lintasan terpendek	✔	
3. Program dapat menampilkan lintasan terpendek beserta jaraknya	✔	
4. Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta		✔

VI. Keluh Kesah

Sebelumnya saya minta maaf karena dari 4 test case yang diminta saya hanya bisa menampilkan satu (peta ITB Dago). Hal ini dikarenakan untuk pembuatan testcasenya terlalu memakan waktu (menentukan node dan membuat txt filenya) karena semuanya saya kerjakan sendiri, yep suatu kesalahan memutuskan mengerjakan tucil ini sendiri disamping mencari partner juga sulit dengan diri saya yang kurang pandai bergaul. Tapi dengan berhasilnya program ini, ya walaupun gaada GUI dan ga tersambung API tapi gapapa, aing cukup bangga dengan diri aing hehehehehehe. Bener-bener 4 hari ke belakang keos tapi alhamdulillah bisa terlewati semua :D.