Penerapan Algoritma Dijkstra Dalam Penentuan Rute Terpendek Yogyakarta-Bandung

Farhan Raditya Aji - 13522142[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522142@std.stei.itb.ac.id

***Abstrak*—Perkembangan transportasi yang kian pesat, membuat setiap orang dapat melakukan perjalanan yang jauh sekalipun dengan kendaraan masing-masing. Dalam perjalanan menuju tempat yang cukup jauh,beralama-lama berada di dalam kendaraan tidaklah hal yang menyenangkan. Tentunya, setiap orang pasti akan memilih rute tercepat jika akan berkendara ke manapun ia pergi. Menentukan rute tercepat untuk mencapai tujuan dapat dilakukan dengan berbagai cara. Salah satu cara yang dapat digunakan yaitu dengan menggunakan algoritma Dijkstra.**

***Kata Kunci*— Graf, Algoritma Dijkstra, Rute Terpendek.**

I. PENDAHULUAN

Pada era mobilitas pribadi yang semakin mendominasi ini, menentukan keefektifan dan efisiensi sebelum melakukan perjalanan menjadi aspek yang penting untuk dipikirkan. Pasalnya berlama-lama di dalam kendaraan tidaklah suatu hal yang nyaman. Ditambah lagi dengan kondisi jalanan yang belum tentu lancar.

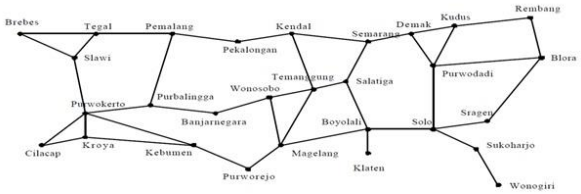
Hal-hal tersebut sangat saya rasakan ketika melakukan perjalanan dari daerah saya Yogyakarta ke Bandung tempat saya berkuliah. Perjalanan dari Yogyakarta ke Bandung bukanlah suatu perjalanan yang singkat. Maka dibutuhkan analisis pada rute perjalanan Yogyakarta-Bandung agar menjadi lebih efektif dan efisien.

Analisis rute Yogyakarta-Bandung untuk menentukan rute yang paling efisien, dilakukan dengan menggunakan teori Graf. Semua rute yang dapat dilewati akan dimodelkan sebagai graf berbobot, dimana setiap daerah yang dilewati sebagai simpulnya dan jarak yang menghubungkan daerah A ke daerah B sebagai bobotnya. Setelah itu proses analisis dilakukan menggunakan Algoritma Dijkstra.

II. LANDASAN TEORI

A. Graf

1) Definisi Graf
Graf adalah struktur data yang merepresentasikan kumpulan objek yang memiliki hubungan dari satu objek dengan objek yang lainnya.



Gambar 1. Contoh Graf [1]

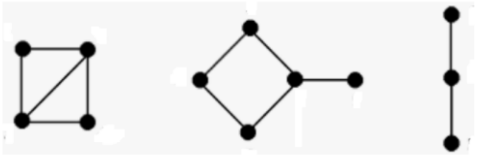
Graf G = (V, E), yang dalam hal ini:

V = himpunan tidak-kosong dari simpul-simpul (vertices) = { v1 , v2 , ... , vn }

E = himpunan sisi (edges) yang menghubungkan sepasang simpul = {e1 , e2 , ... , en }

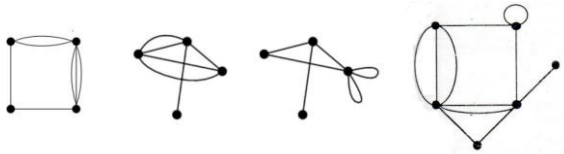
2) Jenis-Jenis Graf
Graf berdasarkan ada atau tidaknya gelang atau sisi ganda pada suatu graf, maka graf digolongkan menjadi dua jenis:

a) Graf Sederhana (*simple graph*)
Graf yang tidak memiliki gelang maupun sisi ganda.



Gambar 2. Graf Sederhana [2]

b) Graf tak-sederhana (*unsimple-graph*)
Graf yang memiliki sisi ganda atau gelang.



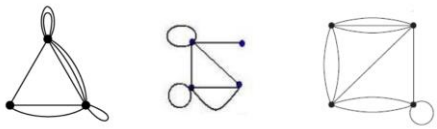
Gambar 3. Graf Tak-Sederhana [2]

Pada graf tak-sederhana ini dibagi menjadi 2 bagian lagi yakni:

- Graf Ganda (*multi-graph*)
  Graf yang memiliki sisi ganda.

Gambar 4. Graf Ganda [2]

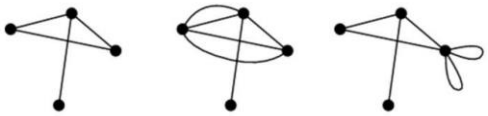
- Graf Semu (*pseudo-graph)*
  Graf yang memiliki sisi gelang.



Gambar 5. Graf Semu [2]

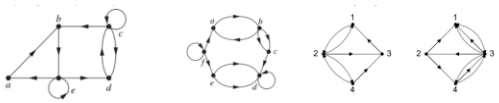
Graf berdasarkan orientasi arah pada sisi, dibedakan atas 2 jenis:

a) Graf Tak-Berarah (*undirected-graph*)
Graf yang sisinya tidak mempunyai orientasi arah.



Gambar 6. Graf Tak-Berarah [2]

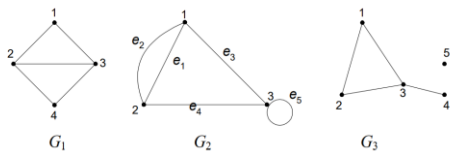
b) Graf Berarah (*directed graph* atau *digraph*)
Graf yang setiap sisinya diberikan orientasi arah.



Gambar 7. Graf Berarah [2]

3) Terminlogi Graf

a) Ketetanggaan (Adjacent)
Dua simpul yang terhubung secara langsung disebut bertetangga. Seperti pada Gambar 8 pada G1 simpul 1 bertetangga dengan simpul 2 dan 3.

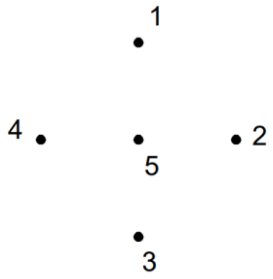


Gambar 8. Ketetanggaan [2]

b) Bersisian (*Incidency*)
Setiap simpul yang terhubung langsung dengan sisi maka disebut bersisian. Seperti pada gambar 8, G1 sisi (2, 3) bersisian dengan simpul 2 dan simpul 3.

c) Simpul Terpencil (*Isolated Vertex*)
Simpul terpencil ialah simpul yang tidak mempunyai sisi yang bersisian dengannya. Seperti pada gambar 8, G3 simpul 5 adalah simpul terpencil.

d) Graf Kosong (*null graph* atau *empty graph*)
Graf yang himpunan sisinya merupakan himpunan kosong (Nn).
Contoh N5:



Gambar 9. Graf Kosong N5 [2]

e) Derajat (*Degree*)
Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Notasi: d(v). Seperti pada gambar 8, G1 d(1) = d(4) = 2 d(2) = d(3) = 3.

f) Lintasan (*Path*)
Lintasan yang panjangnya n dari simpul awal v0 ke simpul tujuan vn di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk

v0, e1, v1, e2, v2, ..., vn –1, en, vn

sedemikian sehingga

e1 = (v0, v1), e2 = (v1, v2), ..., en = (vn-1, vn)

adalah sisi-sisi dari graf G.

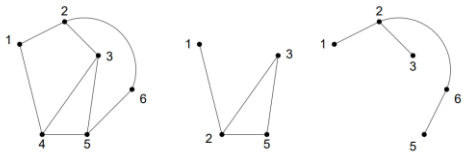
Panjang lintasan adalah jumlah sisi dalam lintasan tersebut. Lintasan 0, 6, 3, 7, 9, 10 pada G memiliki panjang 5.

g) Siklus (*Cycle*) atau Sirkuit (*Circuit*)
Lintasan yang berawal dan berakhir pada simpul yang sama disebut sirkuit atau siklus.

Panjang sirkuit adalah jumlah sisi dalam sirkuit tersebut. Sirkuit 0, 4, 8, 5, 1, 0 pada G memiliki panjang 5.

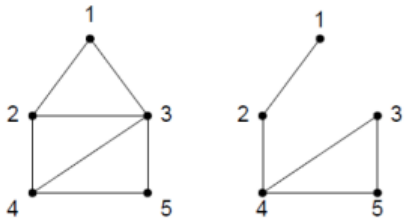
h) Kerterhubungan (*Connected*)
Dua simpul dikatakan terhubung jika ada lintasan dari simpul X ke Y. Jika simpul X dan Y tidak dapat dicapai dari lintasan manapun maka X dan Y disebut dengan graf tak-terhubung.

i) Upagraf (*Subgraph*) dan Komplemen Upagraf
Upagraf adalah kondisi dimana graf G1 = (V1, E1) merupakan himpunan bagian dari G = (V, E). Sedangkan, komplemen dari upagraf G1 digambarkan dengan sebuah graf G2 = (V2, E2) di mana E2 = E – E1.



Gambar 10. Contoh Upagraf [2]

j) Upagraf Merentang (Spanning Subgraph)
Upagraf di mana semua simpul yang terdapat pada graf G dapat ditemukan pada graf satunya.

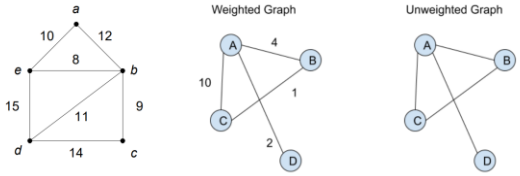


Gambar 11. Contoh Upagraf Merentang [2]

k) *Cut-Set*
Suatu graf G yang jika di *cut-set* akan menjadi dua komponen graf.

Gambar 12. Contoh Cut-Set [2]

l) Graf Berbobot (*Weighted Graph*)
Graf yang setiap sisinya diberi sebuah bobot,seperti jarak,waktu,dan lain-lain.



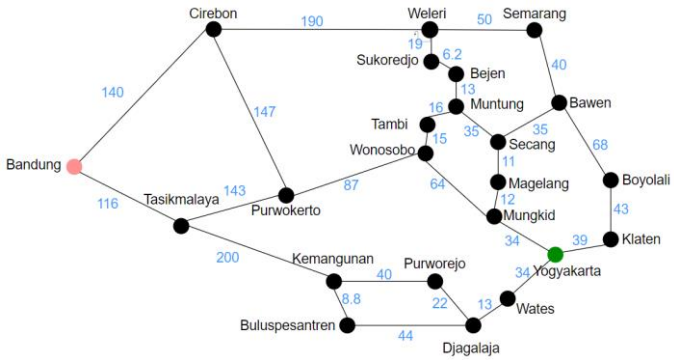
Gambar 12. Contoh Graf Berbobot [2]

B. Algoritma Dijkstra
Dalam melakukan penentuan rute tercepat dari suatu rute yang dimodelkan dalam graf, salah satu algoritma yang dapat digunakan adalah Algoritma Dijkstra. Algortima ini ditemukan oleh seorang ilmuwan komputer asal Belanda yaitu Edsger Wybe Dijkstra. Cara kerja dari algoritma Dijkstra yaitu:

1) Inisialisasi graf berbobot yang memvisualkan rute perjalanannya.
2) Set node awal dengan 0 dan node lain dengan nilai yang sangat besar atau tak-terhingga.
3) Setelah itu cek *node* yang belum terjamah dan hitung jaraknya dari *node* keberanngkatan. Jika jarak yang terhitung lebih kecil dengan jarak yang tersimpan sebelumnya , maka hapus data lama dan simpan dengan data yang baru.
4) Setelah selesai mempertimbangkan setiap node tetangga dari node keberangkatan ,maka tandai node yang telah dijamah menjadi node terjamah. Node yang telah ditandai terjamah maka tidak akan dicek ulang.
5) Lakukan ulang langkah ke-3 dengan set *node* yang belum terjamaah dengan jarak terkecil menjadi titik keberangkatan terlebih dahulu.

III. METODE

A. Insisialisasi Graf Berbobot
Sebelum melakukan analisis dan perhitungan, pertama-tama inisialisasi graf yang memvisualisasikan rute Yogyakarta – Bandung terlebih dahulu. Pada graf yang akan dibuat, tiap *node*-nya akan menggambarkan daerah-daerah yang akan dilewati dan untuk menentukan rute tercepat maka akan digunakan graf berbobot dengan jarak sebagai bobotnya. Untuk mempermudah dalam melihat mana *node* awal dan *node* akhir maka graf yang dibuat untuk *node* awal dan akhir akan dibedakan warna *node*-nya supaya mempermudah dalam menganalisis dan membedakannya. Graf yang digunakan tidak menggunakan graf berarah karena suatu daerah ke daerah lain bisa dilakukan secara bolak-balik sehingga penggambaran graf akan rumit maka dari graf yang digunakan disini adalah graf tak-berarah. Penulisan bobot dalam jarak menggunakan satuan kilometer (Km). Untuk graf lengkap dapat dilihat pada Gambar 13.



Gambar 13. Graf Rute Yogyakarta-Bandung
(sumber: Penulis)

Pada pembuatan graf berbobot ini digunakan referensi menggunakan aplikasi *google maps* untuk menentukan jarak dan arahnya. Rute-rute yang dipilih juga

diasumsikan hanya rute tersebut yang sering dilewati dari Yogyakarta menuju Bandung.

B. Perhitungan

Perhitungan untuk menentukan rute terpendek dari rute Yogyakarta-Bandung akan dilakukan menggunakan algoritma Djikstra. Pengimplementasian algortima Djikstra akan dibuat menjadi sebuah program. Program implementasi akan dibuat menggunakan Bahasa python. Langkah-langkah perhitungan adalah sebagai berikut:

1) *Convert* Graf

Ubah graf dari Gambar 13. kedalam bentuk *code* di pyhton. Respresentasi graf pada pyhton adalah dengan menggunakan *dictionary* dalam *dictionary* atau *nested dictionary*. Penulisan ke *dictionary*-nya dengan menjadikan *node* sebagai *key*-nya dan jarak sebagai *value*-nya. Untuk setiap *node* yang berhubungan akan dituliskan dua kali,seperti:

Graf =
{
‘A’: {‘B’:1, ’C’:2},
‘B’: {‘A’:1,’D’:3}
}

Maka setelah diubah dari Gambar 13. ke python akan menjadi seperti ini:

```
graph = {
    'YOGYAKARTA': {'MUNGKID': 34, 'KLATEN': 39, 'WATES' : 34},
    'MUNGKID': {'YOGYAKARTA':34,'MAGELANG': 12, 'WONOSOBO': 64},
    'MAGELANG': {'SECANG': 11,'MUNGKID': 12},
    'SECANG': {'MAGELANG': 11,'BAWEN': 35, 'MUNTUNG': 35},
    'WONOSOBO': {'MUNGKID': 64,'TAMBI': 15, 'PURWOKERTO': 87},
    'TAMBI': {'WONOSOBO': 15,'MUNTUNG': 16},
    'MUNTUNG': {'TAMBI': 16,'SECANG': 35, 'BEJEN': 13},
    'BEJEN': {'MUNTUNG': 13,'SUKOREDJO': 6.2},
    'SUKOREDJO': {'BEJEN': 6.2,'WELERI': 19},
    'WELERI': {'SUKOREDJO': 19,'SEMARANG': 50,'CIREBON': 190},
    'SEMARANG': {'WELERI': 50,'BAWEN': 40,},
    'BAWEN': {'SEMARANG': 40,'SECANG': 35, 'BOYOLALI': 68},
    'BOYOLALI': {'BAWEN': 68,'KLATEN': 43},
    'KLATEN':{'BOYOLALI': 43,'YOGYAKARTA': 39},
    'CIREBON': {'WELERI': 190,'BANDUNG': 140,'PURWOKERTO': 147},
    'BANDUNG': {'CIREBON': 140,'TASIKMALAYA': 116},
    'TASIKMALAYA': {'BANDUNG': 116,'PURWOKERTO': 143,'KEMANGUNAN': 200},
    'PURWOKERTO': {'TASIKMALAYA': 143,'WONOSOBO': 87,'CIREBON' : 147},
    'KEMANGUNAN': {'TASIKMALAYA': 200,'PURWOREJO': 40,'BULUSPESANTREN': 8
    'PURWOREJO': {'KEMANGUNAN': 40,'DJAGALAJA': 22},
    'DJAGALAJA': {'PURWOREJO': 22,'BULUSPESANTREN': 44,'WATES':13},
    'BULUSPESANTREN': {'DJAGALAJA': 44,'KEMANGUNAN': 8.8},
    'WATES': {'DJAGALAJA': 13,'YOGYAKARTA': 34},
}
```

Gambar 14. Graf Python
(sumber: Penulis)

2) Buat Algoritma Djikstra Ke *Code* Pyhton

a) Pertama, untuk mengimplementasikan algoritma Djisktra ini kita memerlukan sebuah struktur data yang bernama *priority queue*. Maka diperlukan library yang memiliki fungsi-fungsi primitif *priority queue*.

```
import heapq
```

Gambar 15. Import Library heapq
(sumber: Penulis)

b) Kedua, buat sebuah fungsi yang bernama djikstra yang akan menerima 3 parameter yaitu graf, *start*, *end*.

```
def dijkstra(graf, start, end):
```

Gambar 16. Fungsi djikstra
(sumber: Penulis)

c) Ketiga, inisialisasi untuk *node* awal memiliki bobot 0 dan *node* lainnya memiliki bobot *infinity*. Tiap bobot dari tiap *node* akan disimpan dalam dictionary yang menyimpan nilanya float.

```
bobot_node = {}
for node in graf:
    if node == start:
        bobot_node[node] = 0
    else:
        bobot_node[node] = float('infinity')
```

Gambar 17. Inisialisasi Bobot Setiap *Node*
(sumber: Penulis)

d) Keempat, inisialisasi *dictionary* yang menyimpan suatu array. Tiap array dari suatu *node* akan menyimpan rute tercepatnya. Inisialisasi semua dengan array kosong array.

```
rute_tercepat = {}
for node in graf:
    rute_tercepat[node] = []
```

Gambar 18. Inisialisasi Rute Setiap *Node*
(sumber: Penulis)

e) Kelima, inisialisasi *prioty queue* yang menyimpan suatu tuple (int,*node*). Hal tersebut guna untuk menyimpan jarak yang ditempuh dari start menuju *node* yang dimaksud.

```
prioQ = [(0, start)]
```

Gambar 19. Inisialisasi Bobot Setiap *Node*
(sumber: Penulis)

f) Keenam, melakukan proses perhitungan. Proses ini akan dilakukan selama *priority queue* tidak kosong. *Conditional* yang akan menggunakan adalah *while loop*. Selama *looping*, hal yang pertama yang dilakukan adalah mengepop elemen pertama dari *prioQ* dan meyimpannya dalam variabel *current*_jarak dan *current_node*. Sebelum melakukan proses berikutnya cek terlebih dahulu apakah *current*_jarak lebih besar dari bobot_node[*current_node*], jika ya maka skip *loop* sekarang dan lanjut ke *loop* berikutnya. Hal ini karena nilai terkecil yang akan dicari sehingga jika jelas lebih besar maka langsung diskip. Jika tidak masuk pada *conditional*-nya maka lanjut ke proses berikutnya dengan menggunakan *for loop* yang akan memproses

sebanyak tetangga yang dimiliki *current_node*. Lalu setiap loop akan mengambil *node* tetangga dari *current_node* dan jaraknya dari *current_node* ke *node* tetangga. Setelah itu jumlahkan current_jarak dengan jarak dengan tetangga tadi, jika hasilnya lebih kecil dari bobot yang dimiliki tetangga maka *update* bobot nilai tetangga sekarang dengan hasil pertambahannya tadi,lalu *update* rute_tercepat yang dilalui untuk menuju *node* tetangga dengan *current_node* + rute_tercepat[*current_node*]. Lalu push hasil jarak baru tadi dan *node* tetangga kedalam *prioQ*. Setelah itu *for loop* akan berlanjut hingga semua tetangga sudah tercek dan *while loop* belanjut setelah *for loop* selesai.

```
while prioQ:
    current_jarak, current_node = heapq.heappop(prioQ)
    if current_jarak > bobot_node[current_node]:
        continue

    for tetangga, bobot in graf[current_node].items():
        jarak_baru = current_jarak + bobot
        if jarak_baru < bobot_node[tetangga]:
            bobot_node[tetangga] = jarak_baru
            rute_tercepat[tetangga] = rute_tercepat[current_node] + [current_no
            heapq.heappush(prioQ, (jarak_baru, tetangga))
```

Gambar 20. Proses Perhitungan
(sumber: Penulis)

g) Ketujuh, setelah *while loop* berhenti atau *prioQ* sudah kosong maka perhitungan dengan algoritma Dijkstra sudah selesai.Hal terakhir hanya perlu me *return* hasil perhitungannya. Pada fungsi ini ada 2 variabel yang akan di *return* yaitu hasil dari jarak terpendek dari *start node – end node* dan rute mana saja yang ditempuh untuk mencapai *end node.*

```
return bobot_node[end], rute_tercepat[end]
```

Gambar 21. Return Dari Fungsi Dijkstra
(sumber: Penulis)

Berikut adalah implementasi algoritma Dijsktra secara lengkap:

```
def dijkstra(graf, start, end):
    bobot_node = {}
    rute_tercepat = {}
    for node in graf:
        if node == start:
            bobot_node[node] = 0
        else:
            bobot_node[node] = float('infinity')
        rute_tercepat[node] = []

    prioQ = [(0, start)]

    while prioQ:
        current_jarak, current_node = heapq.heappop(prioQ)
        if current_jarak > bobot_node[current_node]:
            continue

        for tetangga, bobot in graf[current_node].items():
            jarak_baru = current_jarak + bobot
            if jarak_baru < bobot_node[tetangga]:
                bobot_node[tetangga] = jarak_baru
                rute_tercepat[tetangga] = rute_tercepat[current_node] + [current_node]
                heapq.heappush(prioQ, (jarak_baru, tetangga))

    return bobot_node[end], rute_tercepat[end]
```

Gambar 22. Program Algoritma Dijkstra
(sumber: Penulis)

3) Buat Main Program
Buat *main* program yang akan menjalankan seluruh algoritma Djikstra dalam mencari jalan tercepat rute Yogyakarta-Bandung.

```
def main():
    graph = {
        'YOGYAKARTA': {'MUNGKID': 34, 'KLATEN': 39, 'WATES' : 34},
        'MUNGKID': {'YOGYAKARTA':34,'MAGELANG': 12, 'WONOSOBO': 64},
        'MAGELANG': {'SECANG': 11,'MUNGKID': 12},
        'SECANG': {'MAGELANG': 11,'BAWEN': 35, 'MUNTUNG': 35},
        'WONOSOBO': {'MUNGKID': 64,'TAMBI': 15, 'PURWOKERTO': 87},
        'TAMBI': {'WONOSOBO': 15,'MUNTUNG': 16},
        'MUNTUNG': {'TAMBI': 16,'SECANG': 35, 'BEJEN': 13},
        'BEJEN': {'MUNTUNG': 13,'SUKOREDJO': 6.2},
        'SUKOREDJO': {'BEJEN': 6.2,'WELERI': 19},
        'WELERI': {'SUKOREDJO': 19,'SEMARANG': 50,'CIREBON': 190},
        'SEMARANG': {'WELERI': 50,'BAWEN': 40,},
        'BAWEN': {'SEMARANG': 40,'SECANG': 35, 'BOYOLALI': 68},
        'BOYOLALI': {'BAWEN': 68,'KLATEN': 43},
        'KLATEN':{'BOYOLALI': 43,'YOGYAKARTA': 39},
        'CIREBON': {'WELERI': 190,'BANDUNG': 140,'PURWOKERTO': 147},
        'BANDUNG': {'CIREBON': 140,'TASIKMALAYA': 116},
        'TASIKMALAYA': {'BANDUNG': 116,'PURWOKERTO': 143,'KEMANGUNAN': 200},
        'PURWOKERTO': {'TASIKMALAYA': 143,'WONOSOBO': 87,'CIREBON' : 147},
        'KEMANGUNAN': {'TASIKMALAYA': 200,'PURWOREJO': 40,'BULUSPESANTREN': 8.8},
        'PURWOREJO': {'KEMANGUNAN': 40,'DJAGALAJA': 22},
        'DJAGALAJA': {'PURWOREJO': 22,'BULUSPESANTREN': 44,'WATES':13},
        'BULUSPESANTREN': {'DJAGALAJA': 44,'KEMANGUNAN': 8.8},
        'WATES': {'DJAGALAJA': 13,'YOGYAKARTA': 34}
        }

    start_node = 'YOGYAKARTA'
    end_node = 'BANDUNG'

    jarak, rute = dijkstra(graph, start_node, end_node)

    print(f"Rute terpendek dari {start_node} menuju {end_node} yaitu :")
    for node in (rute):
        print(f"{node} -> ", end="")
    print(f"{end_node}")
    print(f"Total jarak yang ditempuh: {jarak}")
```

Gambar 23. Main Program
(sumber: Penulis)

IV. HASIL & PEMBAHASAN

A. Hasil
Setelah *main* program dijalankan keluaran yang dihasilkan sebagai berikut:

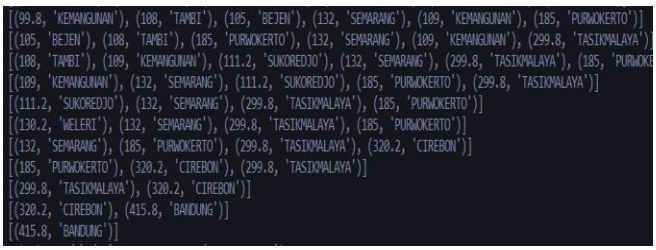
```
Rute terpendek dari YOGYAKARTA menuju BANDUNG yaitu :
YOGYAKARTA -> WATES -> DJAGALAJA -> BULUSPESANTREN -> KEMANGUNAN -> TASIKMALAYA -> BANDUNG
Total jarak yang ditempuh: 415.8
```

Gambar 24. Hasil Perhitungan
(sumber: Penulis)

Untuk detail dari perhitungan fungsi Djikstra,dapat dilihat dari mencetak *prioQ*-nya pada terminal. Maka hasil dari perhitungannya sebagai berikut:

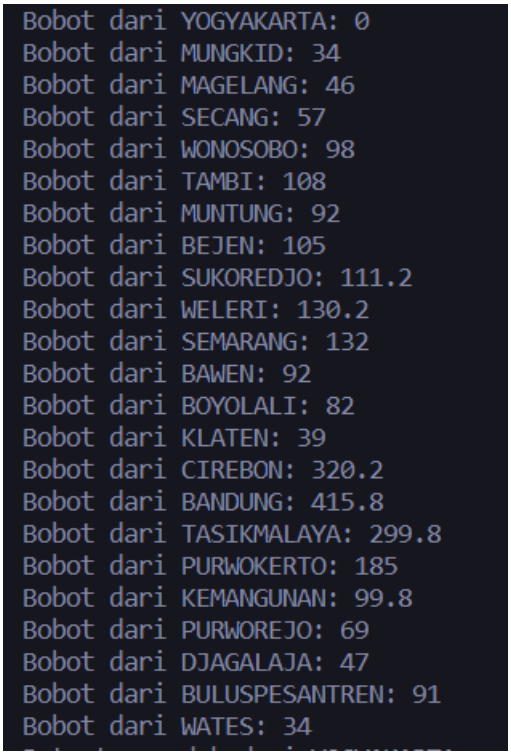
```
[(0, 'YOGYAKARTA')]
[(34, 'MUNGKID'), (39, 'KLATEN'), (34, 'WATES')]
[(34, 'WATES'), (39, 'KLATEN'), (46, 'MAGELANG'), (98, 'WONOSOBO')]
[(39, 'KLATEN'), (47, 'DJAGALAJA'), (46, 'MAGELANG'), (98, 'WONOSOBO')]
[(46, 'MAGELANG'), (47, 'DJAGALAJA'), (98, 'WONOSOBO'), (82, 'BOYOLALI')]
[(47, 'DJAGALAJA'), (57, 'SECANG'), (98, 'WONOSOBO'), (82, 'BOYOLALI')]
[(57, 'SECANG'), (69, 'PURWOREJO'), (98, 'WONOSOBO'), (82, 'BOYOLALI'), (91, 'BULUSPESANTREN')]
[(69, 'PURWOREJO'), (82, 'BOYOLALI'), (92, 'MUNTUNG'), (91, 'BULUSPESANTREN'), (92, 'BAWEN'), (98, 'WONOSOBO')]
[(82, 'BOYOLALI'), (91, 'BULUSPESANTREN'), (92, 'MUNTUNG'), (98, 'WONOSOBO'), (92, 'BAWEN'), (109, 'KEMANGUNAN')]
[(91, 'BULUSPESANTREN'), (92, 'BAWEN'), (92, 'MUNTUNG'), (98, 'WONOSOBO'), (109, 'KEMANGUNAN')]
[(92, 'BAWEN'), (98, 'WONOSOBO'), (92, 'MUNTUNG'), (109, 'KEMANGUNAN'), (99.8, 'KEMANGUNAN')]
[(92, 'MUNTUNG'), (98, 'WONOSOBO'), (99.8, 'KEMANGUNAN'), (109, 'KEMANGUNAN'), (132, 'SEMARANG')]
[(98, 'WONOSOBO'), (108, 'TAMBI'), (99.8, 'KEMANGUNAN'), (132, 'SEMARANG'), (109, 'KEMANGUNAN'), (105, 'BEJEN')]
```

Gambar 25. Step Perhitungan Part 1
(sumber: Penulis)



Gambar 26. Step Perhitungan Part 2
(sumber: Penulis)

Untuk melihat detail-detail bobot dari tiap *node* dapat dilihat dengan mencetak variabel bobot_*node*, berikut adalah hasilnya:



Gambar 27. Bobot Jarak Tiap *Node*
(sumber: Penulis)

Untuk melihat rute-rute tercepat yang dimiliki dari tiap *node* maka kita dapat mencetak variabel rute_tercepat, berikut adalah hasilnya:

Gambar 28. Rute Tercepat Tiap *Node*
(sumber: Penulis)

B. Pembahasan

Berdasarkan dari hasil perhitungan, dapat dilihat rute yang tercepat dari Yogyakarta menuju Bandung yaitu melalui jalur selatan dengan jarak tempuh 415.8 Km. Hal itu berarti menunjukan bahwa dibandingkan dengan jalur utara ataupun dari Klaten terlebih dahulu jarak tempuh yang dihasilkan menjadi lebih jauh lagi. Sebagai contoh pada jika kita akan lewat utara yang mana pasti melewati Cirebon dahulu maka dapat dilihat jarak tempuh terdekat dari Yogyakarta menuju Cirebon adalah 320.2 Km berdasarkan Gambar 27. Maka jika akan menuju Bandung kita perlu menambahkan dengan jarak Cirebon ke Bandung yaitu 140 Km berdasarkan Gambar 13, sehingga total jarak tempuh yang dilalui jika lewat jalur utara adalah 460.2. Hal tersebut memiliki perbedaan jarak yang cukup terasa yaitu sebesar 44.4 Km.

V. KESIMPULAN

Pemodelan dengan menggunakan struktur graf ini dapat menyelesaikan banyak masalah, seperti dalam hal ini pemodelan rute Yogyakarta menuju Bandung. Pemodelan ini juga menggunakan graf berjenis graf berbobot. Pada penentuan untuk mencari rute tercepat juga dapat menggunakan berbagai cara, salah satunya dengan menggunakan algoritma Dijkstra. Seperti pada kasus ini, algoritma Dijkstra dapat menyelesaikan penentuan rute tercepat dari Yogyakarta menuju Bandung yang mana memiliki jarak tempuh total yaitu 415.8 Km.

VI. UCAPAN TERIMAKASIH

Saya panjatkan puji dan syukur kepada Tuhan Yang Maha Esa atas rahmat-Nya, makalah yang berjudul “Penerapan Algoritma Dijkstra Dalam Penentuan Rute Tercepat Yogyakarta-Bandung” dapat terselesaikan tepat waktu dan tidak ada hambatan. Terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. dosen mata kuliah IF2120 Matematika Diskrit yang telah membimbing dan menurunkan ilmu-ilmunya kepada penulis. Penulis juga mengucapkan terima kasih kepada pihak-pihak yang tidak dapat disebut satu-persatu yang telah membantu penulis. Semoga makalah ini dapat bermanfaat bagi banyak orang.

REFERENSI

[1] Alfionita, Shinta. (2015). graph_1. https://shintaalfionita.files.wordpress.com/2015/06/c8927-graph_1.jpg, diakses pada 8 Desember 2023.

[2] Munir, Rinaldi. (2023). Graf (Bagian 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf, diakses pada 8 Desember 2023.

[3] Ardana, Dwi, & Saputra, Ragil. (2023). Penerapan Algoritma Dijkstra pada Aplikasi Pencarian Rute Bus Trans Semarang. https://ilkom.unnes.ac.id/snik/prosiding/2016/45.%20SNIK_334_Algoritma%20Dijkstra.pdf, diakses pada 8 Desember 2023.

[4] Harahap, M. K., Khairina, Nurul. (2017). Pencarian Jalur Terpendek dengan Algoritma Dijkstra. Jurnal & Penelitian Teknik Informatika, 2(2). http://www.jurnal.polgan.ac.id/index.php/sinkron/article/view/61, diakses pada 8 Desember 2023.

[5] Geeksforgeeks. (2023). How to find Shortest Paths from Source to all Vertices using Dijkstra’s Algorithm.

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/, diakses pada 8 Desember 2023.
[6] JSTR Production. (2020). Konsep Lintasan Terpendek Algoritma Dijkstra. [Video]. Youtube. https://www.youtube.com/watch?v=qIzCYrE8570, diakses pada 8 Desember 2023.
[7] Google. (2023). Google Maps. https://www.google.com/maps/@-7.745757,110.3427968,15z?entry=ttu, diakses pada 8 Desember 2023.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2023

Farhan Raditya Aji
13522142