Penggunaan Algoritma *Greedy* dalam Perencanaan Rute Perjalanan Sederhana dengan Beberapa Destinasi

Samuel 13518041
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10, Bandung 40132, Jawa Barat, Indonesia
samnap11@gmail.com 13518041@std.stei.itb.ac.id

***Abstrak*—Makalah ini menjelaskan penggunaan konsep dari algoritma *Greedy* untuk memecahkan masalah terkait perencanaan rute perjalanan sederhana. Algoritma yang digunakan mungkin tidak akurat dengan aplikasi pembuat rute yang sebenarnya, seperti Google Maps atau Waze, tetapi hal ini merupakan dasar dari algoritma-algoritma yang dikembangkan untuk perencanaan rute di aplikasi-aplikasi tersebut.**

Keywords—algoritma Greedy; algoritma Dijkstra; priority queue; graf; Google Maps; perencanaan rute perjalanan

I. PENGENALAN

Di dalam kehidupan sehari-hari, kita sering kali bepergian dan mengunjungi berbagai tempat. Walaupun di masa-masa berat seperti masa karantina ini menghambat kita untuk bepergian, kita tetap bisa berharap dan yakin terhadap secercah titik cerah yang ada di ujung lorong ini. Dengan berdasarkan keyakinan dan harapan tersebut, kita tetap bisa membuat rencana perjalanan yang dapat kita laksanakan setelah masa karantina ini berakhir. Tidak ada yang salah dalam perencanaan dari jauh hari sebelumnya.

Dalam melakukan perjalanan, tentunya kita tidak ingin menghabiskan waktu, uang, dan tenaga kita dengan berlama-lama di perjalanan. Walaupun ada banyak hal yang dapat dinikmati selama perjalanan, seperti pemandangan yang indah, kultur lokal dari tempat yang kita lewati, dan berbagai macam orang yang kita temui dalam perjalanan, fokus kita tetap berada di tujuan perjalanan kita. Oleh karena itu, kita menginginkan rute perjalanan yang efisien dan efektif.

Oleh karena itu, kita harus memilih rute perjalanan yang memiliki *cost* yang paling rendah. Dalam makalah ini, *cost* tersebut hanya sebatas pada jarak tempuh. Namun, seiring dengan perkembangan algoritma dan perkembangan teknologi di abad 21 ini, saat ini sudah banyak aplikasi yang ikut memasukkan faktor-faktor seperti waktu, kondisi jalan, kemacetan, dan kecelakaan sebagai bagian dari perhitungan sehingga meningkatkan efektivitas dan efisiensi dari rute perjalanan yang diambil. Namun, untuk kepentingan makalah ini, penulis hanya akan mengambil *cost* yang berupa jarak tempuh saja.

II. DASAR TEORI

A. *Graf*

Graf merupakan sebuah struktur diskrit yang terdiri dari dua hal, yaitu *vertex* (simpul) dan *edge* (sisi). Gabungan dari kedua hal tersebut menciptakan sebuah struktur yang sangat terkenal dan umum di bidang komputer serta matematika [1].

Berbagai macam persoalan bisa diilustrasikan dan dimodelkan dengan graf. Dengan begitu, kita dapat menyelesaikan permasalahan-permasalahan tersebut dengan menggunakan algoritma-algoritma yang dipakai dalam penyelesaian masalah pada graf.

Ada beberapa contoh masalah yang sering dimodelkan dengan graf, yaitu masalah rute atau *path*, pewarnaan bagian-bagian peta, hubungan komunikasi di dalam sebuah jaringan, atau bahkan dua zat kimia yang memiliki formula kimia yang sama tapi berbeda secara struktural [1]. Fleksibilitas dari graf ini yang membuat graf menjadi salah satu model yang sering dipakai dalam berbagai permasalahan.

Ada berbagai jenis graf. Untuk kepentingan makalah ini, ada beberapa jenis graf yang akan dijelaskan berdasarkan perbedaan yang terletak pada sisi. Sebelum kita masuk ke sana, kita perlu memahami terlebih dahulu apa itu simpul dan sisi.

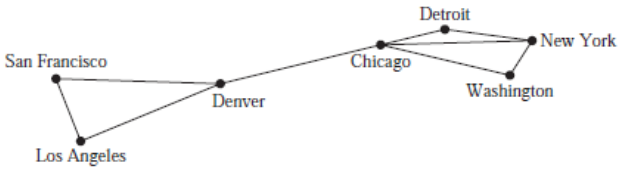


Fig. 1. Graf sederhana

(Sumber: Discrete Mathematics and Its Applications, 7th ed., Kenneth H. Rossen, pp. 642)

Berdasarkan gambar di atas, simpul-simpul dari graf tersebut adalah San Francisco, Los Angeles, Denver, Chicago, Detroit, New York, dan Washington. Sedangkan, sisi adalah

garis-garis yang menghubungkan dua buah simpul secara langsung, seperti sisi yang menghubungkan Los Angeles dengan San Francisco, sisi yang menghubungkan Los Angeles dengan Denver, dan sisi yang menghubungkan Chicago dan Detroit.

Jenis graf yang akan dibahas di sini tidak banyak, hanya beberapa jenis yang memang akan kita pakai saja. Pertama, yaitu graf tidak berarah. Graf tidak berarah (undirected graph) adalah graf yang ada di Fig. 1, yaitu graf yang sisinya tidak memiliki arah. Dengan begitu, kita bisa menelusuri sisi tersebut baik dari kedua arah yang berlawanan. Sisi ini seperti jalan raya pada umumnya yang biasanya dipakai untuk dua arah yang berlawanan—meskipun ada beberapa jalan yang hanya menerima satu arah saja—sehingga jenis graf ini merupakan jenis yang tepat untuk pemodelan masalah kali ini.

Kedua, ada graf yang disebut sebagai graf berbobot (weighted graph). Graf ini disebut berbobot karena sisi-sisi dari graf tersebut memiliki bobot (*cost*) yang harus menjadi faktor yang diperhitungkan saat penyelesaian masalah dari graf tersebut.

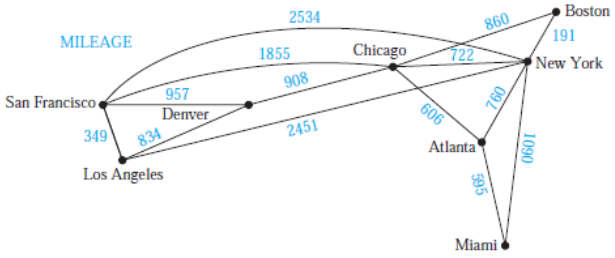


Fig. 2. Graf berbobot

(Sumber: Discrete Mathematics and Its Applications, 7th ed., Kenneth H. Rossen, pp. 708)

Graf jenis ini memang banyak digunakan untuk permasalahan-permasalahan terkait *shortest path* di mana bobot dari sisi tersebut menandakan jarak antarsimpul. Hal ini berlaku pada fig. 2, di mana simpul merepresentasikan kota. Dengan begitu, kita bisa mengetahui jarak dari Los Angeles ke Denver adalah 834 kilometer, San Francisco ke Denver adalah 957 kilometer, dan jarak dari Atlanta ke New York adalah 760 kilometer.

Selain itu, ada juga graf yang disebut sebagai graf berbobot terhubung. Sesuai namanya, graf berbobot terhubung adalah graf yang memiliki bobot pada sisinya sekaligus juga merupakan graf terhubung. Apa itu graf terhubung? Graf terhubung adalah graf yang untuk setiap sepasang simpul, pasti ada jalan (*path*) di antaranya. Dengan begitu, dari sebuah simpul, kita dapat menelusuri dan menemukan semua simpul lainnya.

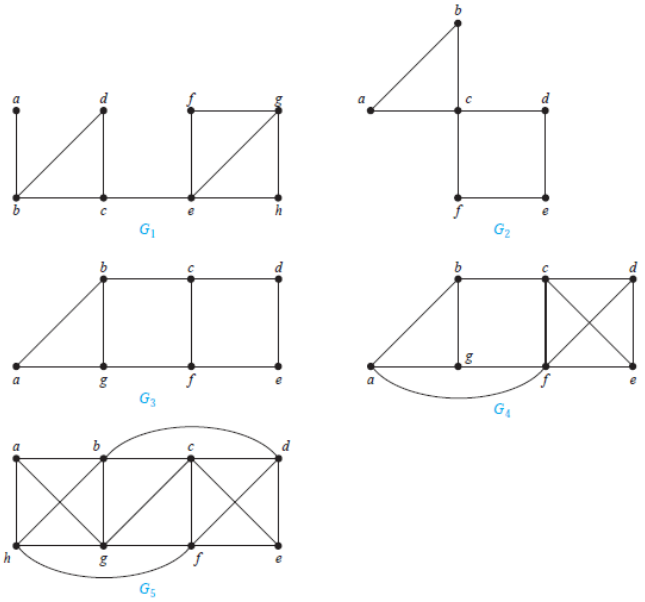


Fig. 3. Graf terhubung

(Sumber: Discrete Mathematics and Its Applications, 7th ed., Kenneth H. Rossen, pp. 683)

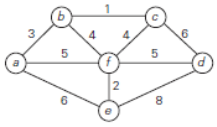
B. *Algoritma Greedy*

Algoritma *greedy* merupakan algoritma yang menyangkut masalah optimasi, yaitu masalah yang ingin memaksimalkan atau meminimalisasikan sebuah fungsi objektif. Algoritma *greedy* memberikan sudut pandang baru untuk pemecahan sebuah masalah dengan cara membuat solusi yang dilakukan secara bertahap. Dalam setiap tahap, solusi tersebut akan diekspansi dari tahap sebelumnya hingga solusi akhir telah ditemukan [2].

Ada tiga hal yang menjadi pertimbangan algoritma *greedy* dalam penemuan solusi di setiap tahapnya. Pertama, *feasible*, di mana pilihan yang diambil harus yang bisa dipakai dan memenuhi batasan (*constraint*) dari permasalahan tersebut. Kedua, *locally optimal*, di mana pilihan yang diambil merupakan pilihan yang optimal di langkah tersebut. Hal ini dilakukan karena algoritma *greedy* berharap tahap-tahap yang optimal lokal dapat mengantar kepada solusi yang optimal secara global. Ketiga, *irrevocable*, di mana pilihan yang sudah diambil tidak dapat dibatalkan dan diubah di suatu waktu di masa depan. Hal ini berarti, saat algoritma *greedy* ternyata tidak mengantarkan kita pada solusi yang optimal secara global, kita tidak dapat mengubah langkah kita di tahap-tahap sebelumnya. Kita hanya bisa menerima bahwa algoritma *greedy* tidak bisa dipakai untuk masalah tersebut.

Inilah mengapa algoritma tersebut dikatakan sebagai *greedy* karena algoritma tersebut berusaha untuk mengambil pilihan yang terbaik di setiap langkahnya. Misalnya, untuk permasalahan *shortest path*, setiap pilihan yang diambil di setiap langkah adalah pilihan yang memiliki *cost* yang paling rendah. Sebagai contoh lain, untuk permasalahan mereduksi sebuah graf menjadi pohon merentang minimum (*minimum spanning tree*) dengan menggunakan algoritma Prism atau

algoritma Kruskal, pada setiap langkah, pilihan yang diambil adalah sisi yang bobotnya paling rendah tetapi tidak menciptakan siklik.



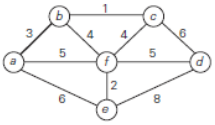
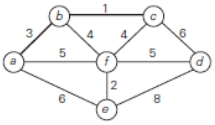
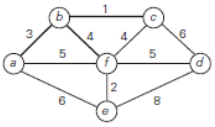
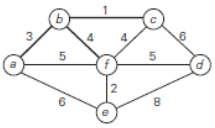
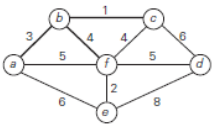
Tree vertices	Remaining vertices	Illustration
a(–, –)	**b(a, 3)** c(–, ∞) d(–, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(–, ∞) e(a, 6) f(b, 4)	
c(b, 1)	d(c, 6) e(a, 6) **f(b, 4)**	
f(b, 4)	d(f, 5) **e(f, 2)**	
e(f, 2)	**d(f, 5)**	
d(f, 5)		

Fig. 4. Salah satu contoh algoritma *greedy*, yaitu algoritma Prim

(Sumber: Introduction to The Design and Analysis of Algorithms, 3rd ed., Anna Levitin, pp. 320)

Berdasarkan fig. 3, kita dapat melihat bagaimana pada setiap langkah, algoritma Prim berusaha untuk menemukan dan memilih sisi yang berbobot paling kecil yang berhubungan langsung dengan simpul sebelumnya, tanpa menciptakan sebuah siklik pun.

C. *Algoritma Dijkstra*

Algoritma ini merupakan salah satu algoritma *greedy* yang dipakai untuk pemecahan masalah mengenai *single source shortest path*. Di mana ada sebuah simpul yang dipilih sebagai sumber tunggal pada sebuah graf berbobot terhubung. Kemudian, dengan menggunakan algoritma Dijkstra, kita bisa mencari *shortest path* dari simpul sumber ke semua simpul lainnya.

Algoritma ini bergerak dari simpul sumber untuk mencari simpul terdekat. Setelah ditemukan, perhitungan akan dimulai dari simpul terdekat kedua tersebut dan mencari simpul terdekat dari simpul terdekat kedua tersebut. Hal ini terus berlanjut hingga semua simpul telah dijelajahi. Dengan begitu, saat solusi sudah ditemukan, kita bisa memastikan bahwa ini merupakan solusi optimal secara global karena algoritma ini memastikan jika kita menemukan *shortest path* hingga simpul ke-i, *path* hingga simpul ke-(i-1) juga merupakan *shortest path* ke simpul ke-(i-1). Dengan begitu, algoritma Dijkstra bisa menemukan semua *shortest path* dari simpul sumber ke semua simpul lainnya.

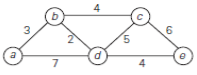
ALGORITHM *Dijkstra(G, s)*

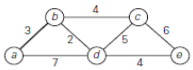
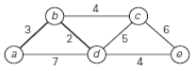
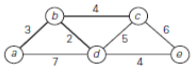
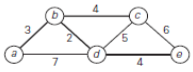
```
//Dijkstra's algorithm for single-source shortest paths
//Input: A weighted connected graph G = ⟨V, E⟩ with nonnegative weights
//       and its vertex s
//Output: The length d_v of a shortest path from s to v
//        and its penultimate vertex p_v for every vertex v in V
Initialize(Q)  //initialize priority queue to empty
for every vertex v in V
    d_v ← ∞;  p_v ← null
    Insert(Q, v, d_v)  //initialize vertex priority in the priority queue
d_s ← 0;  Decrease(Q, s, d_s)  //update priority of s with d_s
V_T ← ∅
for i ← 0 to |V| − 1 do
    u* ← DeleteMin(Q)  //delete the minimum priority element
    V_T ← V_T ∪ {u*}
    for every vertex u in V − V_T that is adjacent to u* do
        if d_u* + w(u*, u) < d_u
            d_u ← d_u* + w(u*, u);  p_u ← u*
            Decrease(Q, u, d_u)
```

Fig. 5. Algoritma Dijkstra

(Sumber: Introduction to The Design and Analysis of Algorithms, 3rd ed., Anna Levitin, pp. 335)

Berikut adalah contoh simulasi dari penggunaan algoritma *greedy* pada sebuah graf berbobot terhubung.



Tree vertices	Remaining vertices	Illustration
a(–, 0)	**b(a, 3)** c(–, ∞) d(a, 7) e(–, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(–, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	
c(b, 7)	**e(d, 9)**	
e(d, 9)		

The shortest paths (identified by following nonnumeric labels backward from a destination vertex in the left column to the source) and their lengths (given by numeric labels of the tree vertices) are as follows:

from *a* to *b*:	*a* – *b*	of length 3
from *a* to *d*:	*a* – *b* – *d*	of length 5
from *a* to *c*:	*a* – *b* – *c*	of length 7
from *a* to *e*:	*a* – *b* – *d* – *e*	of length 9

Fig. 6. Simulasi algoritma Dijkstra

(Sumber: Introduction to The Design and Analysis of Algorithms, 3rd ed., Anna Levitin, pp. 336)

D. Perencanaan Rute (Route Planning)

Perencanaan rute merupakan sebuah langkah yang sangat penting untuk dilakukan sebelum melakukan sebuah perjalanan. Dengan memilih rute perjalanan yang baik, kita bisa menghemat berbagai hal, seperti biaya transportasi, waktu, dan energi, dalam perjalanan kita. Pada akhirnya, rute perjalanan yang baik dapat meningkatkan efisiensi dan efektivitas dari perjalanan kita dengan sangat signifikan sehingga bisa meningkatkan *mood* dan emosi kita juga.

Oleh karena itu, tahap ini merupakan tahap yang sangat krusial. Ada banyak algoritma yang bisa digunakan untuk merencanakan rute. Di makalah ini, perencanaan rute akan melibatkan satu faktor, yaitu jarak sebuah titik dari titik asal. Namun, di zaman modern di mana teknologi sudah sangat berkembang, ada banyak faktor yang bisa ditambahkan untuk kepentingan perencanaan rute, seperti kondisi jalan, frekuensi kendaraan pada suatu jalan, cuaca, dan hal-hal lainnya. Dengan begitu, rute yang dipilih bisa semakin efisien dan efektif dalam memotong waktu perjalanan yang akan ditempuh.

III. IMPLEMENTASI

Untuk membuktikan algoritma Dijkstra dalam perencanaan rute sederhana, penulis membuat implementasinya dalam bahasa pemrograman C++.

Di bawah ini merupakan implementasi dari algoritma Dijkstra yang dituangkan ke dalam fungsi calculateShortestPath.

```
void Graph::calculateShortestPath(int src) {
    // pii merupakan pair int dengan int
    priority_queue<pii, vector<pii>, greater<pii>> pq;

    pq.push(make_pair(0, src));
    dist[src] = 0;

    while(!pq.empty()) {
        int curr = pq.top().second;
        pq.pop();
        for(list<pii>::iterator i = adj[curr].begin(); i != adj[curr].end(); i++) {
            int dest = i->first;
            int weight = i->second;

            if(dist[dest] > dist[curr] + weight) {
                parent[dest] = curr;
                dist[dest] = dist[curr] + weight;
                pq.push(make_pair(dist[dest], dest));
            }
        }
    }
}
```

Dalam implementasi yang dibuat, fungsi tersebut dituangkan ke dalam sebuah *class* Graph yang berguna untuk representasi peta yang berisi kota-kota sebagai simpul dan jarak antarkota sebagai *cost* dari sebuah sisi yang menghubungkan dua buah kota tersebut.

Dalam program tersebut, array dist digunakan sebagai array yang menyimpan data jarak sebuah kota atau simpul dari kota awal atau simpul sumber. Array ini yang akan diperiksa dan diperbarui di setiap langkah algoritma Dijkstra. Kemudian, ada juga array parent yang berguna untuk menyimpan parent dari sebuah simpul atau kota. Parent disini didefinisikan sebagai kota terdekat dengan simpul tersebut yang menjadi kota sebelum kota tersebut di dalam sebuah *path*. Array parent ini digunakan untuk mencetak *path* dari jarak terpendek antara dua buah kota. Hal tersebut dapat dilakukan dengan algoritma backtracking yang dilakukan dengan cara pemanggilan secara rekursif.

```
void Graph::printPath(int dest) {
    if(parent[dest] == -1) return;

    printPath(parent[dest]);
    cout << dest << " ";
}
```

Dengan adanya fungsi ini, *path* menjadi dapat ditunjukkan kepada pengguna.

Selain itu, program ini juga bisa menerima beberapa kota tujuan (simpul *destination*) karena dalam sebuah perjalanan, bisa saja tujuan dari perjalanan tersebut bukan hanya satu saja, tetapi ada beberapa. Kota-kota tujuan tersebut bisa berada di dalam satu *path* yang sama atau dalam *path* yang berbeda-beda. Untuk setiap kota tujuan, akan dibuatkan *shortest path* dari simpul sumber atau kota awal.

IV. HASIL PERCOBAAN

Ada beberapa percobaan yang akan dilakukan untuk melihat bagaimana algoritma ini bekerja dan melihat hasil dari algoritma ini. Untuk setiap percobaan, akan menggunakan graf yang berbeda-beda. Graf ini dibuat dengan bantuan sebuah aplikasi web, yaitu Graph Editor yang terdapat di laman https://csacademy.com/app/graph_editor/. Aplikasi ini mampu membuat graf dengan sangat mudah, efisien, dan interaktif. Selain itu, aplikasi ini juga mampu menggambar graf bagaimanapun bentuknya dan jenisnya. Semua hanya bergantung kepada keinginan pengguna.

Untuk percobaan pertama, graf yang akan digunakan adalah graf sebagai berikut.

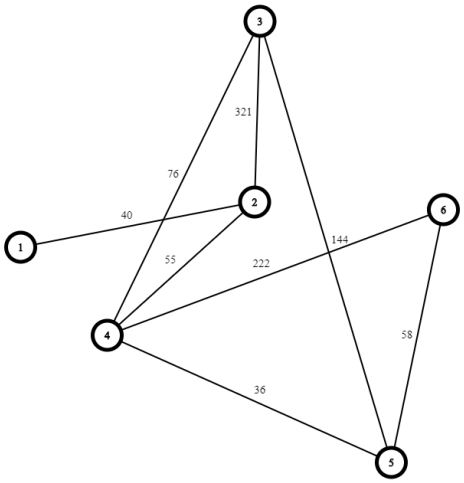


Fig. 7. Graf percobaan pertama

(Sumber: penulis)

Pada graf di atas, simpul-simpul merepresentasikan kota-kota yang memiliki label berupa bilangan bulat dan jaraknya menjadi *cost* dari sisi yang bersesuaian. Kemudian, kita coba menjalankan di program tersebut dengan jumlah tujuan hanya satu kota, yaitu kota dengan label 5. Kota awal adalah kota dengan label 1. Program kemudian menghasilkan luaran seperti di bawah ini.

```
1 --> 5
Path-nya adalah: 1 2 4 5
Cost-nya adalah: 131
```

Jika kita representasikan dalam graf semula, rute tersebut adalah seperti di bawah ini.

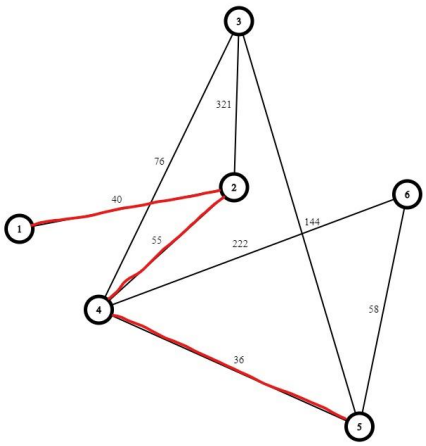


Fig. 8. Hasil percobaan pertama

(Sumber: penulis)

Untuk percobaan kedua, graf yang akan digunakan adalah graf berikut.

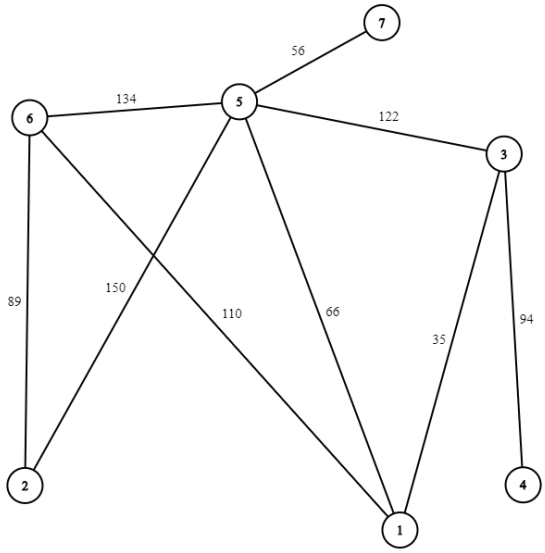


Fig. 9. Graf percobaan kedua

(Sumber: penulis)

Seperti pada percobaan sebelumnya, representasi kota sebagai simpul dan jarak antarkota sebagai *cost* dari sisi masih berlaku. Jika program dijalankan untuk graf di atas dengan kota awal adalah kota 2 dan kota tujuan ada 3 buah, yaitu 5, 7, dan 3, kita akan mendapatkan luaran seperti di bawah ini.

```
2 --> 5
Path-nya adalah: 2 5
Cost-nya adalah: 150

2 --> 7
Path-nya adalah: 2 5 7
Cost-nya adalah: 206

2 --> 3
Path-nya adalah: 2 6 1 3
Cost-nya adalah: 234
```

Jika kita representasikan kembali dalam graf semula, rute tersebut adalah seperti di bawah ini.

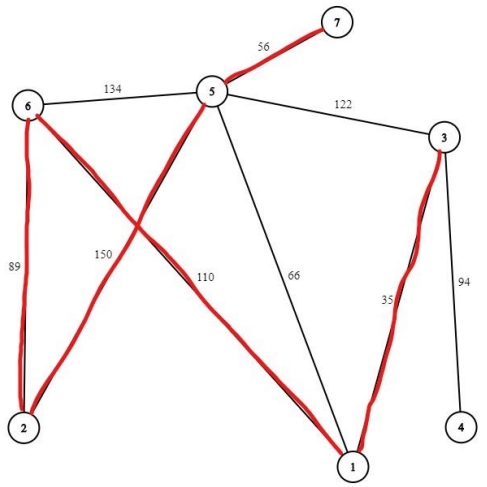


Fig. 10. Hasil percobaan kedua

(Sumber: penulis)

Di dalam dua percobaan tersebut, program berhasil merencanakan rute terpendek untuk setiap pasangan kota awal dan kota tujuan. Walaupun begitu, masih ada banyak perbaikan yang dapat dilakukan terhadap algoritma ini, yaitu bisa menerima bobot negatif, yaitu saat kota yang dikunjungi adalah kota yang sebelumnya sudah kita lewati. Selain itu, program ini juga bisa dilengkapi dengan GUI untuk menghasilkan tampilan rute yang lebih menarik.

VIDEO LINK AT YOUTUBE

Untuk video dari makalah ini, dapat ditemukan di link https://youtu.be/8GakXP0h10M. Video tersebut akan terbuka untuk publik mulai 5 April 2020 pukul 00.00 WIB. Untuk program juga akan tersedia di laman github saya, yaitu https://github.com/samnap11/ setelah makalah ini dipublikasikan.

ACKNOWLEDGMENT

Saya turut berterima kasih kepada Tuhan Yang Maha Esa yang telah menuntun saya dan menyertai saya selama pengerjaan makalah ini berlangsung. Saya juga berterima kasih kepada tim dosen IF2211 Strategi Algoritma, yaitu Dr. Ir. Rinaldi Munir, M.T., Dr. Nur Ulfa Maulidevi, S.T., M.Sc., dan juga Dr. Masayu Leylia Khodra, S.T., M.T. yang telah mengajari kami dengan penuh tanggung jawab dan kasih saying selama satu semester ini. Terakhir, saya juga berterima kasih kepada teman-teman dan keluarga yang selalu hadir bagi saya di setiap saat untuk menemani dan mendukung serta berbagi pikiran.

REFERENCES

[1] Kenneth H. Rossen, Discrete Mathematics and Its Applications, 7th ed., New York: McGraw-Hill, 2012, pp. 641.

[2] Anany Levitin, Introduction to The Design and Analysis of Algorithms, 3rd ed., New Jersey: Pearson Education, Inc., 2012, pp. 315.

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, 3 Mei 2020

13518041 Samuel