PEMBOBOTAN ULANG PADA GRAF BERBOBOT NEGATIF UNTUK MENERAPKAN ALGORITMA DIJKSTRA DALAM MENENTUKAN LINTASAN TERPENDEK

SKRIPSI

OLEH:
NATASYA THALIA SALSABILLAH
NIM. 210601110080

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2025

PEMBOBOTAN ULANG PADA GRAF BERBOBOT NEGATIF UNTUK MENERAPKAN ALGORITMA DIJKSTRA DALAM MENENTUKAN LINTASAN TERPENDEK

SKRIPSI

Diajukan Kepada
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang
untuk Memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Matematika (S.Mat)

Oleh
Natasya Thalia Salsabillah
NIM. 210601110080

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2025

PEMBOBOTAN ULANG PADA GRAF BERBOBOT NEGATIF UNTUK MENERAPKAN ALGORITMA DIJKSTRA DALAM MENENTUKAN LINTASAN TERPENDEK

SKRIPSI

Oleh
Natasya Thalia Salsabillah
NIM. 210601110080

Telah Disetujui untuk Diuji

Malang, 16 Mei 2025

Dosen Pembimbing I	Dosen Pembimbing II
Mohammad Nafie Jauhari, M.Si	Ach. Nashichuddin, M.A
NIPPPK. 19870218 202321 1 018	NIP. 19730705 200003 1 002

Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc
NIP. 19741129 200012 2 005

PEMBOBOTAN ULANG PADA GRAF BERBOBOT NEGATIF UNTUK MENERAPKAN ALGORITMA DIJKSTRA DALAM MENENTUKAN LINTASAN TERPENDEK

SKRIPSI

Oleh
Natasya Thalia Salsabillah
NIM. 210601110080

Telah Dipertahankan di Depan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
untuk Memperoleh Gelar Sarjana Matematika (S.Mat)
Tanggal 12 Juni 2025

Ketua Penguji	: Prof. Dr. H. Turmudi, M. Si., Ph.D.	.....
Anggota Penguji I	: Hisyam Fahmi, M.Kom	.....
Anggota Penguji II	: Mohammad Nafie Jauhari, M.Si	.....
Anggota Penguji III	: Ach. Nashichuddin, M.A	.....

Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc
NIP. 19741129 200012 2 005

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama	: Natasya Thalia Salsabillah
NIM	: 210601110080
Program Studi	: Matematika
Fakultas	: Sains dan Teknologi
Judul Skripsi	: Pembobotan Ulang pada Graf Berbobot Negatif untuk Menerapkan Algoritma Dijkstra dalam Menentukan Lintasan Terpendek

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini merupakan hasil karya sendiri, bukan pengambilan tulisan atau pemikiran orang lain yang saya akui sebagai pemikiran saya, kecuali dengan mencantumkan sumber cuplikan pada daftar rujukan. Apabila dikemudian hari terbukti skripsi ini adalah hasil jiplakan atau tiruan, maka saya bersedia menerima sanksi yang berlaku atas perbuatan tersebut.

Malang, 12 Juni 2025

Natasya Thalia Salsabillah

NIM. 210601110080

MOTO

“Terima kasih kepada mereka yang pernah meremehkan langkahku. Tanpa kalian, mungkin aku tak akan sekuat ini dalam membuktikan bahwa batas hanya ada ketika kita berhenti mencoba.”

PERSEMBAHAN

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat serta hidayah-Nya sehingga penulis dapat menyelesaikan skripsi ini dengan penuh kerendahan hati dan kesabaran yang luar biasa. Keberhasilan dalam penulisan skripsi ini tentunya tidak terlepas dari berbagai bantuan pihak. Oleh karena itu, penulis menyampaikan terimakasih kepada kedua orang tua saya, Ayah Gatot Subroto dan Mama Entri Rahayu. Gelar sarjana ini saya persembahkan untuk kedua orang tua saya tercinta, yang selalu memberikan dukungan berupa moril maupun materil yang tak terhingga, serta doa yang tidak ada putusnya, sehingga penulis mampu menyelesaikan studi sarjana hingga selesai. Semoga rahmat Allah SWT selalu megiringi kehidupan mereka yang barokah, senantiasa diberi kesehatan dan panjang umur.

KATA PENGANTAR

Assalamualaikum Warahmatullahi Wabarakatuh

Puji syukur kehadirat Allah Swt. atas segala limpahan rahmat dan karunia-Nya kepada penulis agar dapat menyelesaikan penyusunan skripsi yang berjudul "Pembobotan Ulang pada Graf Berbobot Negatif untuk Menerapkan Algoritma Dijkstra dalam Menentukan Lintasan Terpendek". Selawat serta salam semoga selalu tersampaikan kepada manusia paling mulia Nabi Muhammad saw. yang telah membawa kita dari zaman jahiliah menuju jalan yang terang benderang, yakni *ad-dinul* (agama Islam).

Skripsi ini tidak akan terwujud tanpa orang-orang tercinta di sekeliling penulis yang mendukung dan membantu. Terima kasih dan doa terbaik penulis persembahkan kepada:

1. Prof. Dr. H. M. Zainuddin, M.A., selaku rektor Universitas Islam Negeri Maulana Malik Ibrahim.
2. Prof. Dr. Sri Harini, M.Si, selaku dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim.
3. Dr. Elly Susanti, S.Pd., M.Sc, selaku ketua Program Studi Matematika, Universitas Islam Negeri Maulana Malik Ibrahim.
4. Mohammad Nafie Jauhari, M.Si, selaku Dosen Pembimbing I yang senantiasa mengayomi dan membimbing penulis.
5. Ach. Nashichuddin, M.A, selaku Dosen Pembimbing II yang telah memberikan arahan dan ilmunya kepada penulis.
6. Prof. Dr. H. Turmudi, M.Si, Ph.D, selaku Ketua Penguji dalam ujian skripsi yang telah memberikan kritik, saran serta dukungannya kepada penulis.
7. Hisyam Fahmi, M.Kom, selaku Penguji Utama dalam ujian skripsi yang telah memberikan kritik, saran serta dukungannya kepada penulis.
8. Seluruh dosen Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim.
9. Ayahanda tercinta Gatot Subroto, Ibunda tercinta Entri Rahayu, Kakak tercinta Ersya Maulidya Rahayu Subrata, Adik tercinta Julia Agatha Teris Sabrina serta seluruh keluarga tercinta yang selalu memberikan do'a, semangat serta motivasi

kepada penulis.

10. Seluruh mahasiswa program studi matematika angkatan 2021 atas segala kenangan, cerita dan pengalaman berharga selama masa perkuliahan.
11. Seluruh pihak yang tidak dapat penulis sebutkan satu-persatu atas bantuan dalam menyelesaikan skripsi ini.
12. Dan yang terakhir, kepada diri saya sendiri. Terima kasih karena memutuskan tidak menyerah sesulit apapun proses penyusunan skripsi ini dan telah menyelesaikannya sebaik dan semaksimal mungkin, ini merupakan pencapaian yang patut dirayakan untuk diri sendiri.

Semoga Allah Subhanahu wa ta'ala membalas semua kebaikan mereka. Dan semoga skripsi ini dapat bermanfaat khususnya bagi penulis dan pembaca. *Aamiin Wassalamu'alaikum Warahmatullahi Wabarakatuh*

Malang, 12 Juni 2025

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

ABSTRAK

Salsabillah, Natasya Thalia. 2025. **Pembobotan Ulang pada Graf Berbobot Negatif untuk Menerapkan Algoritma Dijkstra dalam Menentukan Lintasam Terpendek.** Skripsi. Program Studi Matematika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Mohammad Nafie Jauhari, M.Si. (II) Ach. Nashichuddin, M.A.

Kata Kunci: Algoritma Dijkstra; Bobot Negatif; Pembobotan Ulang; Algoritma Johnson; Lintasan Terpendek.

Algoritma Dijkstra merupakan algoritma untuk mencari lintasan terpendek yang bekerja secara optimal pada graf berbobot non-negatif. Namun, dalam berbagai permasalahan nyata seperti sistem transportasi dan jaringan keuangan, sering ditemukan sisi dengan bobot negatif yang menyebabkan Algoritma Dijkstra tidak dapat diterapkan secara langsung. Penelitian ini bertujuan untuk mengatasi keterbatasan tersebut dengan menerapkan metode pembobotan ulang menggunakan Algoritma Johnson. Metode ini mengombinasikan Algoritma Bellman-Ford dan Dijkstra untuk mengubah bobot negatif menjadi non-negatif tanpa mengubah struktur solusi optimal. Data yang digunakan berupa dua graf acak berarah yang masing-masing terdiri dari 31 simpul, yang dibuat menggunakan algoritma Erdos-Renyi. Hasil penelitian menunjukkan bahwa pembobotan ulang berhasil membuat bobot graf menjadi non-negatif sehingga Algoritma Dijkstra dapat diterapkan, dan hasil lintasan terpendek yang diperoleh sama dengan hasil dari Algoritma Bellman-Ford. Dengan demikian, metode pembobotan ulang menggunakan Algoritma Johnson terbukti efektif dalam menangani bobot negatif dan tetap menjaga keakuratan hasil pencarian lintasan terpendek menggunakan Algoritma Dijkstra.

ABSTRACT

Salsabillah, Natasya Thalia. 2025. **Reweighting on Negatively Weighted Graphs to Apply Dijkstra's Algorithm in Determining the Shortest Path.** Thesis. Mathematics Study Program, Faculty of Science and Technology, Universitas Islam Negeri Maulana Malik Ibrahim Malang. Advisors: (I) Mohammad Nafie Jauhari, M.Si. (II) Ach. Nashichuddin, M.A.

Keywords: Dijkstra's Algorithm; Negative Weight; Reweighting; Johnson's Algorithm; Shortest Path.

Dijkstra's algorithm is an algorithm for finding the shortest path that works optimally on non-negative weighted graphs. However, in various real problems such as transportation systems and financial networks, edges with negative weights are often found which cause the Dijkstra Algorithm to be unable to be applied directly. This study aims to overcome these limitations by implementing a reweighting method using the Johnson Algorithm. This method combines the Bellman-Ford and Dijkstra Algorithms to change negative weights to non-negative without changing the structure of the optimal solution. The data used are two random directed graphs, each consisting of 31 nodes, which are created using the Erdos-Renyi algorithm. The results of the study show that reweighting successfully makes the graph weights non-negative so that the Dijkstra Algorithm can be applied, and the shortest path results obtained are the same as the results of the Bellman-Ford Algorithm. Thus, the reweighting method using the Johnson Algorithm is proven to be effective in handling negative weights and maintaining the accuracy of the shortest path search results using the Dijkstra Algorithm.
.

مستخلص البحث

سلسبيلا، نتاشا تاليا. ٢٠٢٥. إعادة وزن الرسم البياني ذي الأوزان السالبة لتطبيق خوارزمية ديكسترا في تحديد أقصر مسار. البحث الجامعي، قسم الرياضية، كلية العلوم والتكنولوجيا. جامعة مولانا مالك إبرهيم الإسلامية الحكومية مالانج. المشرف: (١) محمد نافع جوهري، الماجستر في العلوم. (٢) أحمد نصح الدين الماجستيرفي تعليم اللغة العربية.

الكلمات الأساسية: خوارزمية ديكسترا؛ الوزن السالب؛ إعادة الوزن؛ خوارزمية جونسون؛ أقصر مسار.

خوارزمية ديكسترا هي خوارزمية لإيجاد أقصر مسار تعمل على النحو الأمثل على الرسوم البيانية المرجحة غير السالبة. ومع ذلك، في مشاكل العالم الحقيقي مثل أنظمة النقل والشبكات المالية، غالبًا ما يتم العثور على حواف ذات أوزان سالبة مما يجعل خوارزمية ديجكسترا غير قابلة للتطبيق بشكل مباشر. هدف هذا البحث إلى التغلب على هذه القيود من خلال تطبيق طريقة إعادة الترجيح باستخدام خوارزمية جونسون. وتجمع هذه الطريقة بين خوارزمية بليمان فرود وخوارزمية ديجكسترا لتحويل الأوزان السالبة إلى غير سالبة دون تغيير بنية الحل الأمثل. البيانات المستخدمة هي عبارة عن رسمين بيانيين عشوائيين موجهين يتألف كل منهما من ٣١رأسًا، تم إنشاؤهما باستخدام خوارزمية إردوس-ريني. تُظهر النتائج أن إعادة الترجيح تنجح في جعل أوزان الرسم البياني غير سالبة بحيث يمكن تطبيق خوارزمية ديجكسترا، كما أن نتائج أقصر المسارات التي تم الحصول عليها هي نفسها نتائج خوارزمية بيلمان-فورد. وبالتالي، أثبتت طريقة إعادة الترجيح باستخدام خوارزمية جونسون فعاليتها في التعامل مع الأوزان السالبة مع الحفاظ على دقة نتائج إيجاد أقصر مسار باستخدام خوارزمية ديكسترا.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Algoritma Dijkstra merupakan salah satu metode yang digunakan untuk menyelesaikan masalah lintasan terpendek dalam graf berbobot. Namun, algoritma ini hanya dapat digunakan jika semua bobot sisi bernilai tidak negatif (Salaki, 2011). Bobot negatif pada graf mengacu pada nilai negatif yang diberikan sisi, yang dapat diartikan sebagai pengurangan jarak, biaya, atau waktu dalam konteks tertentu. Dalam berbagai aplikasi dunia nyata, bobot negatif sering muncul, misalnya dalam bentuk diskon dalam transaksi ekonomi, keuntungan dalam sistem keuangan, atau pengurangan waktu perjalanan dalam jaringan transportasi akibat faktor tertentu. Namun, jika tidak ditangani dengan benar, bobot negatif dapat menyebabkan ketidakakuratan dalam pencarian lintasan terpendek, terutama jika membentuk siklus negatif, yaitu jalur tertutup dengan total bobot negatif yang membuat algoritma terus berulang tanpa batas (Gozali & Aji, 2017).

Ketidakmampuan Algoritma Dijkstra untuk menangani bobot negatif menjadi kendala dalam berbagai aplikasi. Sebagai contoh, dalam sistem transportasi, biaya perjalanan antar kota mungkin memiliki diskon dinamis yang membuat bobot lintasan negatif dalam kondisi tertentu. Jika siklus negatif terbentuk, Algoritma Dijkstra tidak dapat digunakan karena tidak dapat membedakan apakah jalur tersebut optimal atau tidak valid. Untuk mengatasi keterbatasan ini, metode pembobotan ulang diperlukan agar Algoritma Dijkstra tetap dapat digunakan dalam kondisi graf berbobot negatif (Johnson, 1977).

Salah satu metode yang dapat digunakan untuk pembobotan ulang adalah Algoritma Johnson, yang menggabungkan keunggulan Algoritma Bellman-Ford dan Algoritma Dijkstra untuk mengatasi keterbatasan dalam graf berbobot negatif. Metode ini bekerja dengan mendeteksi siklus negatif dan mentransformasikan graf menjadi non-negatif, lalu menggunakan Algoritma Dijkstra untuk menentukan lintasan terpendek dengan lebih efisien (Ilhamsyah dkk., 2016). Dengan demikian, Algoritma Johnson memungkinkan pencarian lintasan terpendek pada graf berbobot negatif tanpa mengubah struktur graf secara signifikan.

Penelitian ini bertujuan untuk menerapkan metode pembobotan ulang pada graf berbobot negatif agar Algoritma Dijkstra dapat digunakan secara efektif untuk menentukan lintasan terpendek. Dengan memanfaatkan keunggulan Algoritma Johnson, penelitian ini diharapkan dapat memberikan solusi yang efisien dan akurat untuk permasalahan lintasan terpendek pada graf dengan bobot negatif, sekaligus memberikan kontribusi terhadap pengembangan algoritma dalam teori graf dan aplikasinya di dunia nyata. Tantangan dalam penelitian ini, yaitu keterbatasan Algoritma Dijkstra dalam menangani bobot negatif, mengingatkan pada prespektif Islam yang mengajarkan bahwa setiap kesulitan pasti disertai kemudahan, sebagaimana disebutkan dalam firman Allah dalam Surah Al-Insyirah ayat 5-6 yang menegaskan bahwa di balik setiap kesulitan terdapat kemudahan.

Allah berfirman dalam Al-Qur'an surat Al-Insyirah ayat 5-6 yang berbunyi (Kementrian Agama RI, 2017):

فَإِنَّ مَعَ الْعُسْرِ يُسْرًا , إِنَّ مَعَ الْعُسْرِ يُسْرًا

"Maka sesungguhnya bersama kesulitan ada kemudahan, sesungguhnya bersama kesulitan ada kemudahan."

Sebagaimana dijelaskan oleh Quraish Shihab, ayat ini menegaskan bahwa setiap

kesulitan pasti disertai atau diikuti oleh kemudahan. Kesulitan dalam berbagai aspek kehidupan merupakan bagian dari sunnah Allah, dan jika dihadapi dengan tekad serta usaha, akan membawa kemudahan besar di kemudian hari (Shihab, 2002b). Dalam konteks penelitian ini, tantangan utama adalah keterbatasan Algoritma Dijkstra dalam menangani bobot negatif pada graf, yang secara konvensional menghambat penggunaannya pada graf dengan struktur yang kompleks. Namun, seperti yang diajarkan dalam Islam bahwa setiap kesulitan diiringi dengan kemudahan, metode pembobotan ulang memberikan solusi bagi keterbatasan Algoritma Dijkstra, memungkinkan algoritma ini berfungsi secara lebih fleksibel dan akurat dalam graf berbobot negatif. Prinsip ini sejalan dengan pesan ayat tersebut, yang mengingatkan bahwa di setiap kesulitan terdapat solusi yang dapat diupayakan.

Dengan demikian, berdasarkan uraian tersebut, peneliti terinspirasi untuk meneliti pembobotan ulang pada graf berbobot negatif, sehingga Algoritma Dijkstra dapat digunakan untuk menemukan lintasan terpendek dengan hasil yang akurat dan efisien. Penelitian ini diharapkan dapat mengatasi keterbatasan Dijkstra yang tidak mampu menangani bobot negatif secara langsung, serta memberikan kontribusi terhadap pengembangan algoritma pencarian lintasan terpendek yang lebih adaptif dalam berbagai kondisi graf, khususnya dalam konteks aplikasi dunia nyata yang mengandung bobot negatif.

1.2 Rumusan Masalah

Berdasarkan penjelasan pada latar belakang, permasalahan yang akan dibahas dalam penelitian ini adalah bagaimana melakukan pembobotan ulang pada

graf berbobot negatif sehingga Algoritma Dijkstra dapat diterapkan untuk menentukan lintasan terpendek?

1.3 Tujuan Penelitian

Penelitian ini bertujuan untuk menganalisis proses pembobotan ulang pada graf berbobot negatif sehingga Algoritma Dijkstra dapat diterapkan untuk menentukan lintasan terpendek.

1.4 Manfaat Penelitian

Penelitian ini bermanfaat untuk memberikan pemahaman mengenai penerapan pembobotan ulang pada Algoritma Dijkstra, sehingga algoritma tersebut dapat dimanfaatkan secara optimal dalam menentukan lintasan terpendek pada graf dengan bobot negatif.

1.5 Batasan Masalah

Batasan masalah dari penelitian ini adalah:

1. Penggunaan dataset berupa dua graf acak berbobot menggunakan algoritma Erdos-Renyi.
2. Graf yang digunakan terdiri dari dua graf berarah, masing-masing memiliki simpul sebanyak 31.

BAB II

KAJIAN TEORI

2.1 Graf

2.1.1 Definisi dan Struktur Graf

Graf merupakan konsep dalam ilmu pengetahuan yang memiliki berbagai aplikasi praktis. Beragam jenis struktur dapat direpresentasikan dalam bentuk graf, dan banyak permasalahan dapat diselesaikan menggunakan pendekatan ini. Graf adalah salah satu struktur data yang paling umum digunakan.

Secara formal, graf G didefinisikan sebagai himpunan yang terdiri dari dua komponen, yaitu:

$$G = (V, E) \tag{2.1}$$

Dengan:

V: Himpunan yang terdiri dari elemen-elemen yang disebut simpul.

E: Himpunan yang terdiri dari pasangan simpul yang saling terhubung dalam V.

Berdasarkan definisi tersebut, himpunan V harus memiliki setidaknya satu elemen, sedangkan himpunan E dapat kosong. Dengan demikian, graf dapat eksis tanpa memiliki sisi, tetapi harus memiliki minimal satu simpul. Graf yang hanya terdiri atas satu simpul tanpa sisi disebut sebagai graf trivial. Graf dapat dikelompokkan ke dalam berbagai jenis berdasarkan kriteria tertentu, seperti keberadaan sisi ganda, banyaknya simpul, atau orientasi sisinya (Chartrand dkk., 2011).

Graf memiliki struktur yang terdiri dari beberapa elemen utama yang membentuk hubungan antar simpul. Simpul merupakan elemen dasar dalam graf yang merepresentasikan suatu objek dan biasanya dilambangkan sebagai $v_1, v_2, \dots, v_n$. Dalam berbagai aplikasi, simpul dapat merepresentasikan berbagai entitas, seperti individu dalam jaringan sosial atau kota dalam sistem transportasi. Simpul-simpul tersebut dihubungkan oleh sisi yang merepresentasikan hubungan antara dua simpul dalam graf. Sisi ini dilambangkan sebagai $e_1, e_2, \dots, e_m$, dengan $e = (v_i, v_j)$ yang menunjukkan adanya koneksi antara simpul v_i dan v_j. Dalam graf berarah, sisi memiliki arah tertentu yang menunjukkan hubungan dari satu simpul ke simpul lainnya, sedangkan dalam graf tak berarah, sisi tidak memiliki arah sehingga hubungan antar simpul bersifat dua arah (Rahayuningsih, 2018).

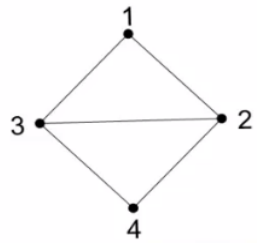
Selain itu, setiap simpul dalam graf memiliki derajat simpul yang menunjukkan banyak sisi yang terhubung ke simpul tersebut. Dalam graf berarah, derajat simpul terbagi menjadi derajat masuk, yaitu banyak sisi yang masuk ke simpul, dan derajat keluar yaitu banyak sisi yang keluar dari simpul tersebut. Sementara itu, dalam graf tak berarah, derajat simpul dihitung berdasarkan banyak total sisi yang terhubung ke simpul tersebut tanpa memperhitungkan arah (Rahayuningsih, 2018). Dengan memahami struktur graf beserta elemen-elemennya, berbagai permasalahan dalam bidang transportasi, komunikasi, dan optimasi jaringan dapat dimodelkan dan diselesaikan secara lebih efektif menggunakan teori graf.

2.1.2 Jenis-Jenis Graf

Graf dapat dibagi menjadi dua kategori berdasarkan keberadaan ada atau tidaknya sisi ganda atau *loop*, yaitu (Chartrand dkk., 2011):

1. Graf Sederhana

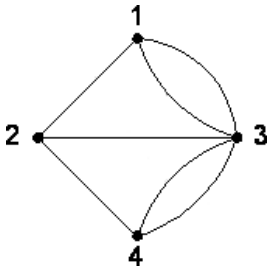
   Graf sederhana adalah sebuah struktur graf yang tidak memiliki koneksi langsung antara simpul dengan dirinya sendiri (*loop*) dan tidak memiliki koneksi ganda antara simpul yang sama (sisi ganda).



Gambar 2.1 Contoh Graf Sederhana

2. Graf Tak Sederhana

   Graf tak sederhana adalah sebuah graf yang memiliki sisi ganda atau *loop*.



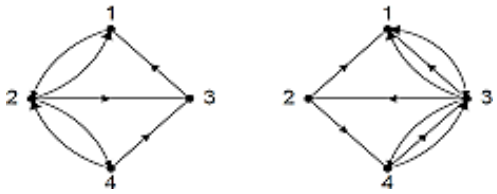
Gambar 2.2 Contoh Graf Tak Sederhana

Orientasi sisi graf mendasari pengelompokan graf menjadi dua kategori utama, yaitu:

1. Graf Berarah

   Graf berarah adalah struktur yang terdiri dari kumpulan simpul dan kumpulan sisi berarah yang menghubungkan simpul-simpul tersebut. Sisi berarah ini sering disebut sebagai busur. Busur dalam graf berarah memiliki arah yang ditandai dengan panah. Ini berarti bahwa pergerakan dari satu

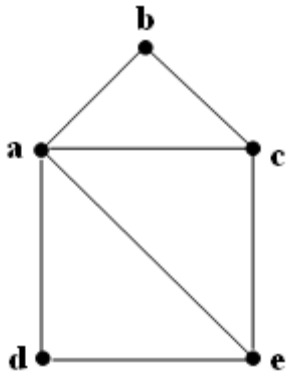
simpul ke simpul lainnya melalui busur hanya dapat dilakukan dalam satu arah saja.



Gambar 2.3 Contoh Graf Berarah

2. Graf Tak Berarah

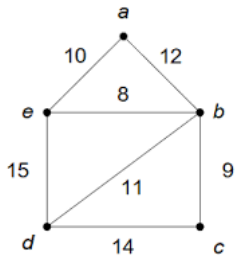
Graf tak berarah adalah sebuah struktur dalam teori graf yang terdiri dari kumpulan simpul dan kumpulan sisi yang menghubungkan pasangan simpul tersebut. Pada graf tak berarah, sisi-sisi ini tidak memiliki arah tertentu, sehingga memungkinkan pergerakan di antara simpul-simpul tersebut dapat dilakukan bolak-balik. Secara formal, graf tak berarah dapat dinyatakan sebagai pasangan (V, E), di mana V adalah kumpulan simpul dan E adalah kumpulan sisi yang terdiri dari pasangan tak berurutan dari simpul-simpul dalam V.



Gambar 2.4 Contoh Graf Tak Berarah

Selain kategori di atas, terdapat jenis graf lainnya yang disebut graf berbobot. Graf berbobot merupakan graf yang memiliki bobot atau nilai tertentu pada setiap sisinya. Nilai bobot ini dapat bervariasi tergantung pada konteks permasalahan yang dimodelkan. Sebagai contoh, bobot dapat menunjukkan jarak

antara dua titik, kapasitas aliran, biaya perjalanan, waktu tempuh dalam jaringan komunikasi, atau ongkos produksi (Nugraha, 2011).



Gambar 2.5 Contoh Graf Berbobot

2.1.3 Representasi Graf dalam Pemrograman

Dalam implementasi graf menggunakan komputer, terdapat beberapa cara representasi yang umum digunakan, yaitu:

1. Matriks Ketetanggaan (*Adjacency Matrix*)

   Suatu graf tidak hanya dapat direpresentasikan dalam bentuk gambar, tetapi juga dalam bentuk matriks yang disebut matriks keterhubungan langsung (*adjacency matrix*). Misalkan diberikan sebuah graf G tanpa *loop* dengan himpunan simpul V dan himpunan sisi E, maka matriks ketetanggaan dari graf G didefinisikan sebagai sebuah matriks A berukuran $n \times n$, di mana n adalah banyak simpul dalam graf. Setiap elemen matriks a_{ij} menunjukkan banyak sisi yang menghubungkan simpul v_i dan v_j. Jika terdapat sisi yang menghubungkan v_i dan v_j, maka nilai a_{ij} diisi dengan banyak sisi tersebut. Jika tidak ada sisi yang menghubungkan kedua simpul tersebut, maka $a_{ij} = 0$. Dalam graf berbobot, elemen matriks tidak hanya bernilai 0 atau 1, tetapi juga dapat berupa bobot dari sisi yang menghubungkan simpul v_i dan v_j. Matriks ketetanggaan ini sangat berguna dalam implementasi algoritma graf, terutama dalam pencarian lintasan terpendek dan perhitungan

keterhubungan antar simpul (Rahayuningsih, 2018).

2. Matriks Keterkaitan (*Incidence Matrix*)

Matriks keterkaitan merupakan salah satu metode untuk merepresentasikan suatu graf dalam bentuk numerik. Representasi ini menggambarkan hubungan antara simpul dan sisi dalam graf tanpa harus menggunakan gambar. Misalkan diberikan suatu graf G tanpa *loop* dengan himpunan simpul $V(G) = \{v_1, v_2, \dots, v_n\}$ dan himpunan sisi $E(G) = \{e_1, e_2, \dots, e_m\}$, maka matriks keterkaitan $I(G)$ dari graf G didefinisikan sebagai sebuah matriks berukuran $n \times m$, di mana n adalah banyak simpul dan m adalah banyak sisi dalam graf. Setiap elemen dalam matriks keterkaitan $I(G) = [a_{ij}]$ memiliki aturan sebagai berikut (Rahayuningsih, 2018).

$$a_{ij} = \begin{cases} 1, & \text{Jika simpul } v_i \text{ terhubung dengan sisi } e_j \\ 0, & \text{Jika simpul } v_i \text{ tidak terhubung dengan sisi } e_j \end{cases} \quad (2.2)$$

Matriks keterkaitan sering digunakan dalam berbagai aplikasi teori graf, seperti analisis struktur jaringan, pemodelan sistem transportasi, serta perancangan sirkuit listrik. Dengan menggunakan representasi ini, keterhubungan antara simpul dan sisi dalam suatu sistem dapat dianalisis dengan lebih mudah.

3. Matriks Derajat (*Degree Matrix*)

Matriks derajat merupakan salah satu cara untuk merepresentasikan graf dalam bentuk matriks. Matriks ini menunjukkan banyak sisi yang terhubung dengan setiap simpul dalam graf. Misalkan diberikan suatu graf G tanpa *loop* dengan himpunan simpul $V(G) = \{v_1, v_2, \dots, v_n\}$ dan himpunan sisi $E(G) = \{e_1, e_2, \dots, e_m\}$, maka matriks dejarat $D(G)$ dari graf G didefinisikan sebagai sebuah matriks berukuran $n \times n$. Setiap elemen dalam

matriks derajat $D(G) = [d_{ij}]$ memiliki aturan sebagai berikut (Rahayuningsih, 2018):

$$d_{ij} = \begin{cases} d(v_i), & \text{Jika } i = j \\ 0, & \text{Jika } i \neq j \end{cases} \quad (2.3)$$

Di mana $d(v_i)$ menyatakan derajat simpul v_i, yaitu banyak sisi yang terhubung dengan simpul tersebut. Matriks derajat digunakan dalam berbagai analisis teori graf, termasuk dalam perhitungan spektrum graf, analisis jaringan, serta permodelan dalam ilmu komputer dan fisika matematika.

2.2 Lintasan Terpendek (*Shortest Path*)

Masalah pencarian lintasan terpendek dalam graf melibatkan identifikasi lintasan antara dua atau lebih simpul dalam graf berbobot. Graf berbobot adalah graf di mana setiap sisinya memiliki nilai atau bobot tertentu, yang biasanya merepresentasikan jarak, misalnya antara kota-kota. Bobot ini umumnya bernilai positif, meskipun dalam beberapa kondisi dapat berupa bilangan negatif. Lintasan terpendek antara simpul awal dan simpul tujuan adalah lintasan dengan total bobot terkecil dan hanya melewati simpul-simpul berbeda. Istilah "terpendek" dalam konteks ini mengacu pada minimalisasi bobot lintasan dalam graf. Beberapa jenis permasalahan lintasan terpendek meliputi *a pair shortest path*, *all pairs shortest path*, *single source shortest path*, dan *intermediate shortest path* (Bunaen dkk., 2022).

2.2.1 *A Pair Shortest Path*

Lintasan terpendek yang menghubungkan dua simpul tertentu dikenal sebagai *a pair shortest path*. Pendekatan yang umum digunakan untuk masalah ini adalah Algoritma A^*. Algoritma A^* memanfaatkan fungsi heuristik untuk memperkirakan jarak dari simpul saat ini ke simpul tujuan, sehingga pencarian lintasan menjadi lebih efisien. Fungsi heuristik ini biasanya dipilih agar tidak melebih-lebihi jarak sebenarnya, sehingga algoritma tetap menghasilkan solusi optimal (Hart et al., 1968).

2.2.2 *All Pairs Shortest Path*

Permasalahan *all pairs shortest path* berfokus pada pencarian lintasan terpendek antara setiap pasangan simpul dalam graf. Algoritma Floyd-Warshall merupakan salah satu metode yang sering digunakan untuk masalah ini. Algoritma ini menggunakan pendekatan dinamik untuk mengupdate jarak antara setiap pasangan simpul dengan kompleksitas waktu sebesar $O(n^3)$, di mana n adalah banyak simpul dalam graf (Junior & Wille, 2018). Meskipun tidak selalu efisien untuk graf dengan banyak simpul yang sangat besar, algoritma ini sangat berguna dalam konteks graf yang padat dan ketika informasi lintasan terpendek antara semua pasangan simpul dibutuhkan, seperti dalam analisis jaringan komunikasi dan transportasi.

2.2.3 *Single Source Shortest Path*

Dalam permasalahan *single source shortest path*, fokus pencarian adalah menemukan lintasan terpendek dari satu simpul ke seluruh simpul lainnya dalam

graf. Algoritma Dijkstra merupakan contoh yang paling dikenal dalam permasalahan ini, terutama pada graf dengan bobot sisi yang non-negatif. Dijkstra bekerja dengan prinsip *greedy* untuk memperbarui jarak terpendek secara iteratif dari simpul sumber ke simpul lainnya. Namun, ketika graf mengandung bobot negatif, Algoritma Bellman-Ford menjadi pilihan yang tepat karena kemampuannya untuk menangani sisi dengan bobot negatif sekaligus mendeteksi siklus negatif. Algoritma Bellman-Ford memiliki kompleksitas $O(n \times m)$, di mana n adalah banyak simpul dan m adalah banyak sisi, sehingga penggunaannya perlu disesuaikan dengan ukuran graf (Cormen dkk., 2009).

2.2.4 *Intermediate Shortest Path*

Intermediate shortest path adalah permasalahan lintasan terpendek yang harus melalui simpul tertentu di antara simpul asal dan simpul tujuan (Bunaen dkk., 2022). Pendekatan untuk menyelesaikan masalah ini yaitu pencarian lintasan yang dilakukan secara terpisah antara simpul perantara ke simpul perantara dan antara simpul perantara ke simpul tujuan. Setelah kedua lintasan tersebut ditemukan, hasilnya digabungkan untuk memperoleh lintasan terpendek yang memenuhi syarat. Pendekatan ini sangat relevan dalam aplikasi praktis, misalnya pada perencanaan rute logistik yang mensyaratkan lintasan optimal melalui titik-titik strategis seperti pusat distribusi.

2.3 Algoritma Penentuan Lintasan Terpendek

Algoritma adalah serangkaian intruksi yang digunakan untuk menyelesaikan suatu permasalahan. Intruksi-intruksi ini diuraikan secara bertahap

dari awal hingga akhir. Masalah yang diatasi dapat beragam, asalkan memenuhi kriteria kondisi awal yang diperlukan sebelum algoritma dijalankan. Algoritma juga melibatkan proses berulang (iterasi) serta pengembalian keputusan hingga permasalahan terselesaikan (Maulana, 2017).

Menurut (Knuth, 1997), bahwa algoritma memiliki lima ciri utama:

1. *Defineteness* (Pasti dan Jelas). Setiap langkah dalam algoritma harus dijelaskan secara tegas tanpa adanya makna yang ambigu.
2. *Finiteness* (Terbatas). Setiap langkah dalam algoritma harus dijelaskan dengan jelas, dan algoritma yang baik harus memiliki titik akhir atau penghentian setelah intruksi selesai. Program yang tidak berhenti menunjukkan bahwa algoritma tersebut tidak valid.
3. *Input* (Masukan). Algoritma yang baik harus dapat memproses data masukan yang benar.
4. *Output* (Keluaran). Algoritma yang baik adalah yang menghasilkan keluaran berdasarkan data masukan yang telah diproses. Algoritma yang efektif adalah algoritma dengan langkah-langkah yang efisien dan dapat dijalankan dalam waktu yang wajar.
5. *Effectiveness* (Efektif). Algoritma yang paling efisien adalah yang memiliki langkah-langkah sederhana dan dapat dijalankan dalam waktu yang wajar.

Meskipun algoritma umumnya dikaitkan dengan bidang ilmu komputer dan matematika, banyak proses dalam kehidupan sehari-hari yang juga dapat dijelaskan menggunakan konsep algoritma. Beberapa algoritma yang umum digunakan dalam pencarian lintasan terpendek adalah Algoritma Dijkstra, Algoritma Bellman-Ford, dan Algoritma Johnson.

2.3.1 Algoritma Dijkstra

Algoritma Dijkstra merupakan salah satu metode yang digunakan untuk menentukan lintasan terpendek dalam graf berbobot, dengan syarat semua bobot sisi bernilai tidak negatif. Algoritma ini termasuk dalam kategori algoritma *greedy*, yang banyak diterapkan dalam penyelesaian masalah optimasi. Dalam konteks graf berbobot, Algoritma Dijkstra berperan untuk menemukan lintasan dengan bobot total terkecil. Hasilnya adalah jarak terpendek antara dua atau lebih simpul dalam graf, dengan nilai total bobot yang seminimal mungkin (Bunaen dkk., 2022). Adapun langkah-langkah yang digunakan dalam penyelesaian masalah lintasan terpendek dengan Algoritma Dijkstra biasanya dijelaskan sebagai berikut (Sunardi dkk., 2019).

1. Menentukan titik awal sebagai simpul pertama, lalu memberikan bobot jarak dari simpul tersebut ke simpul-simpul terdekat secara bertahap. Algoritma Dijkstra kemudian mengembangkan pencarian secara berkelanjutan dari satu simpul ke simpul lainnya hingga seluruh simpul terhubung.
2. Setiap simpul diberi bobot terhadap simpul-simpul lainnya, dengan menetapkan nilai 0 pada simpul awal dan nilai tak hingga pada simpul-simpul lain yang belum ditentukan bobotnya.
3. Semua simpul yang belum dilalui, serta simpul awal ditandai sebagai "simpul keberangkatan".
4. Dari simpul keberangkatan, selanjutnya dihitung jarak ke simpul terhubung yang belum dilalui. Jika jarak yang dihitung lebih pendek daripada jarak yang telah tercatat sebelumnya, maka data lama akan dihapus dan

digantikan dengan jarak yang baru.

5. Setelah mempertimbangkan setiap jarak ke simpul terhubung, simpul yang telah dilalui akan diberi label "simpul dilewati". Simpul yang sudah dilewati tidak akan diperiksa kembali, dan jarak yang tersimpan adalah jarak terakhir yang paling minimal bobotnya.
6. Simpul yang belum dilewati dengan jarak terkecil dari simpul keberangkatan akan ditetapkan sebagai "simpul keberangkatan" berikutnya, dan langkah 4 - 6 akan diulang.

2.3.2 Algoritma Bellman-Ford

Algoritma ini dikembangkan oleh Richard Bellman dan Lester Ford, Jr. Algoritma ini sangat mirip dengan Algoritma Dijkstra namun mampu menangani bobot negatif untuk mencari lintasan terpendek dalam graf berbobot. Algoritma Bellman-Ford merupakan pengembangan dari Algoritma Dijkstra, Algoritma Bellman-Ford akan benar jika dan hanya jika graf tidak memuat siklus negatif yang diperoleh dari sumbernya. Secara umum langkah-langkah algoritmanya adalah sebagai berikut (Bawole & Chernovita, 2019):

1. Menentukan simpul awal serta mencatat seluruh simpul dan sisi dalam graf.
2. Menginisialisasi simpul awal dengan nilai nol, sedangkan simpul lainnya diberi nilai tak hingga.
3. Melakukan iterasi pada seluruh simpul dalam graf, dimulai dari simpul awal hingga semua simpul telah dijelajahi. Perhitungan dilakukan dengan meninjau jarak antara dua simpul, yaitu U sebagai simpul awal dan V sebagai simpul tujuan, dengan UV sebagai sisi penghubung. Jika jarak pada

V lebih besar dibandingkan jumlah jarak U dan bobot UV, maka jarak V diperbarui dengan nilai tersebut hingga seluruh simpul telah dijelajahi.

4. Mengulang proses pada seluruh sisi untuk mendeteksi kemungkinan adanya siklus negatif dalam graf. Selanjutnya, dilakukan pengecekan pada setiap sisi UV. Jika jarak UV lebih besar dibandingkan jumlah jarak U dan bobot UV, maka dapat disimpulkan bahwa graf tersebut mengandung siklus negatif.

2.3.3 Algoritma Johnson

Algoritma Johnson merupakan metode untuk menyelesaikan masalah *shortest path* dengan tujuan menemukan lintasan terpendek dengan bobot total yang seminimal mungkin. Algoritma ini menggabungkan konsep dari Algoritma Bellman-Ford dan Dijkstra. Algoritma ini bekerja dengan langkah-langkah berikut (Ilhamsyah dkk., 2016):

1. Tambahkan simpul tambahan s ke dalam graf dengan sisi berbobot nol, sehingga semua simpul dalam graf terhubung dengan s.
2. Hitung lintasan terpendek dari simpul s ke semua simpul lainnya menggunakan Algoritma Bellman-Ford. Algoritma ini digunakan karena mampu menangani bobot negatif. Namun, proses harus dihentikan jika ditemukan siklus negatif dalam graf.
3. Transformasi bobot dilakukan pada setiap sisi (u, v) di graf menggunakan rumus:

$$\widehat{w}(u,v) = w(u,v) + h(u) - h(v) \qquad (2.4)$$

Dengan:

$\widehat{w}(u,v)$: Bobot baru (setelah pembobotan ulang dari sisi (u,v)

$w(u,v)$: Bobot asli dari sisi (u,v) dalam graf awal.

$h(u)$: Potensial dari simpul u

$h(v)$: Potensial dari simpul v

4. Hapus simpul tambahan s dan semua sisi yang terhubung ke simpul tersebut dari graf. Setelah proses ini, graf kembali ke struktur aslinya, tetapi dengan bobot sisi yang telah diubah menjadi non-negatif.
5. Terapkan Algoritma Dijkstra pada graf yang telah dimodifikasi. Dengan bobot yang telah dimodifikasi menjadi non-negatif, Algoritma Dijkstra dapat digunakan untuk menentukan lintasan terpendek dari simpul sumber ke setiap simpul lainnya secara optimal.

Keunggulan Algoritma Johnson dibandingkan metode lain, seperti menggunakan Bellman-Ford untuk setiap pasangan simpul, terletak pada efisiensinya. Dengan kompleksitas $O(V \times E)$ untuk satu kali Bellman-Ford dan $O(V\ log\ V\ +\ E)$ untuk setiap iterasi Dijkstra, algoritma ini lebih cepat untuk graf besar yang jarang (*sparse*). Selain itu, Johnson juga memungkinkan solusi untuk graf berbobot negatif tanpa mengorbankan keakuratan hasil. Dalam penerapan praktis, Algoritma Johnson sering digunakan pada analisis jaringan, optimasi transportasi, dan perencanaan jalur, terutama jika graf memiliki banyak simpul dan hubungan antar simpul cukup kompleks.

2.4 Teori Graf dalam Pandangan Islam

Islam menekankan pentingnya ilmu pengetahuan sebagai sarana untuk

meningkatkan derajat manusia di sisi Allah SWT, baik di dunia maupun di akhirat. Ilmu memungkinkan manusia memahami dan menyelesaikan berbagai persoalan, termasuk dalam bidang matematika dan informatika. Salah satu permasalahan yang menarik dalam teori graf adalah bagaimana menangani bobot negatif dalam Algoritma Dijkstra. Dalam QS. Al-Mujadilah ayat 11 (Kementrian Agama RI, 2017), Allah SWT berfirman:

يٰٓاَيُّهَا الَّذِيْنَ اٰمَنُوْٓا اِذَا قِيْلَ لَكُمْ تَفَسَّحُوْا فِى الْمَجٰلِسِ فَافْسَحُوْا يَفْسَحِ اللّٰهُ لَكُمْۚ وَاِذَا قِيْلَ انْشُزُوْا فَانْشُزُوْا يَرْفَعِ اللّٰهُ الَّذِيْنَ اٰمَنُوْا مِنْكُمْۙ وَالَّذِيْنَ اُوْتُوا الْعِلْمَ دَرَجٰتٍۗ وَاللّٰهُ بِمَا تَعْمَلُوْنَ خَبِيْرٌ

"Wahai orang-orang yang beriman, apabila dikatakan kepadamu "Berilah kelapangan di dalam majelis-majelis," lapangkanlah, niscaya Allah akan memberi kelapangan untukmu. Apabila dikatakan, "Berdirilah," (kamu) berdirilah. Allah niscaya akan mengangkat orang-orang yang beriman di antaramu dan orang-orang yang diberi ilmu beberapa derajat. Allah Maha Teliti terhadap apa yang kamu kerjakan."

Sebagaimana dijelaskan dalam tafsir Al-Misbah (Shihab, 2002a), ayat ini mengajarkan ilmu harus digunakan untuk memberikan manfaat kepada masyarakat. Dalam konteks teori graf, ilmu memungkinkan manusia menemukan cara untuk menangani bobot negatif dalam pencarian jalur terpendek. Bobot negatif dapat menghambat Algoritma Dijkstra karena dapat menyebabkan solusi yang salah atau perulangan tak hingga. Namun, dengan pendekatan pembobotan ulang, bobot negatif dapat diubah menjadi bobot non-negatif tanpa mengubah solusi optimalnya. Hal ini mencerminkan bagaimana ilmu memberikan solusi terhadap hambatan yang muncul dalam berbagai permasalahan, sebagaimana Islam mengajarkan untuk mencari jalan keluar atas kesulitan yang dihadapi. Dalam tafsir Jalalain (Al-Mahalli & As-Suyuthi, n.d.), perintah untuk melapangkan majelis juga bermakna memberikan ruang kepada orang lain agar dapat memperoleh manfaat dari ilmu. Dalam teori graf, pembobotan ulang dapat diibaratkan sebagai "melapangkan" jalur

dalam suatu graf berbobot negatif, sehingga Algoritma Dijkstra dapat berfungsi dengan benar. Konsep ini mencerminkan bahwa ilmu dapat mengubah kondisi yang tampak sulit menjadi lebih mudah dipecahkan, sebagaimana Islam mengajarkan bahwa ilmu memberikan kelapangan dalam kehidupan. Sedangkan dalam Tafsir Al-Muyassar (Mashudi, 2019), ilmu dipandang sebagai sesuatu yang membedakan individu dalam memahami ajaran Islam secara lebih mendalam dan bermanfaat bagi sesama.

2.5 Kajian Topik Dengan Teori Pendukung

Penelitian ini membahas mengenai penerapan metode pembobotan ulang pada graf berbobot untuk memungkinkan Algoritma Dijkstra menangani bobot negatif. Agar lebih memahami konsep ini, diperlukan penjelasan mengenai teori-teori pendukung yang relevan, yaitu Algoritma Dijkstra, konsep bobot negatif pada graf, dan metode pembobotan ulang dengan Algoritma Johnson sebagai solusi.

Algoritma Dijkstra merupakan metode yang digunakan untuk menyelesaikan permasalahan pencarian lintasan terpendek pada graf dengan bobot sisi yang selalu bernilai tidak negatif. Algoritma ini termasuk dalam kategori *greedy algorithm*, yang sering diterapkan dalam berbagai permasalahan optimasi. Dalam konteks pencarian lintasan terpendek, Algoritma Dijkstra berfungsi untuk menentukan bobot terkecil pada graf berbobot. Hasilnya adalah lintasan terpendek antara dua atau lebih simpul dalam graf, dengan total bobot yang paling minimal (Bunaen dkk., 2022). Meskipun efisien untuk graf berbobot non-negatif, Algoritma Dijkstra memiliki keterbatasan ketika diterapkan pada graf dengan bobot negatif. Hal ini karena algoritma mengasumsikan bahwa setelah jarak terpendek ke suatu

simpul ditemukan, jarak tersebut tidak akan berubah, yang tidak berlaku dalam kasus bobot negatif.

Bobot negatif sering muncul dalam masalah dunia nyata, seperti simulasi biaya atau keuntungan dalam logistik, jaringan transportasi, atau model optimasi kompleks lainnya. Bobot negatif dapat menyebabkan siklus negatif, yang menghasilkan solusi lintasan terpendek yang tidak valid. Oleh karena itu, diperlukan pendekatan untuk menyelesaikan masalah ini tanpa mengubah hasil akhir lintasan terpendek. Salah satu solusi yang efektif adalah dengan menggunakan Algoritma Johnson, yang menggabungkan teknik pembobotan ulang dengan keunggulan Algoritma Dijkstra dan Bellman-Ford. Langkah-langkah Algoritma Johnson untuk pembobotan ulang telah dijelaskan pada subbab 2.3.3, yang dimulai dengan membangun graf baru dan diakhiri dengan penerapan Algoritma Dijkstra pada graf yang telah dimodifikasi untuk menghitung lintasan terpendek. Dengan menggunakan Algoritma Johnson, proses pembobotan ulang dapat mengubah graf berbobot negatif menjadi graf dengan bobot non-negatif tanpa memengaruhi hasil lintasan terpendek. Algoritma ini tidak hanya efisien untuk graf kecil, tetapi juga sangat berguna untuk graf besar dengan banyak pasangan simpul.

Dalam penelitian ini, metode pembobotan ulang dengan Algoritma Johnson diterapkan untuk memanfaatkan kecepatan Algoritma Dijkstra dalam menghitung lintasan terpendek pada graf berbobot negatif. Pendekatan ini memberikan solusi yang efisien dan akurat, terutama dalam konteks aplikasi nyata seperti jaringan transportasi dan optimasi logistik. Dengan memanfaatkan fungsi potensial dari Algoritma Johnson, setiap bobot sisi dalam graf diubah sedemikian rupa sehingga menjadi non-negatif tanpa mengubah hasil lintasan terpendek. Transformasi ini

memungkinkan Algoritma Dijkstra untuk bekerja secara optimal, bahkan pada graf yang awalnya mengandung bobot negatif.

Adapun representasi graf yang digunakan dalam penelitian ini adalah *adjacency matrix* (matriks ketetanggaan karena bentuk tersebut mempermudah proses pengolahan data serta penerapan transformasi bobot pada tahap pembobotan ulang. Matriks ini merepresentasikan hubungan antar simpul dalam bentuk dua dimensi, di mana setiap elemen menunjukkan nilai bobot sisi antara dua simpul. Permasalahan lintasan terpendek yang digunakan merupakan *single source shrotest path*, yaitu pencarian lintasan terpendek dari satu simpul asal menuju seluruh simpul lainnya dalam graf. Permasalahan ini sesuai dengan struktur kerja Algoritma Dijkstra yang dapat diterapkan secara efisien setelah bobot negatif dikonversi menjadi non-negatif melalui penerapan Algoritma Johnson.

Kombinasi teori-teori ini memberikan dasar yang kuat untuk penelitian ini, yang bertujuan untuk mengembangkan solusi optimal dalam menangani graf berbobot negatif dengan memodifikasi Algoritma Dijkstra menggunakan teknik pembobotan ulang pada Algoritma Johnson.

BAB III

METODE PENELITIAN

3.1 Jenis Penelitian

Penelitian ini menggunakan pendekatan kuantitatif dengan melakukan analisis numerik terhadap efektivitas pembobotan ulang dalam graf berbobot negatif. Metode yang diterapkan meliputi simulasi graf berbobot dengan variasi bobot negatif pada graf berjumlah 31 simpul, serta pengukuran kinerja Algoritma Dijkstra sebelum dan sesudah penerapan pembobotan ulang.

Tujuan penelitian ini adalah untuk menganalisis efektivitas pembobotan ulang pada graf berbobot negatif dengan menguji perubahan hasil lintasan terpendek sebelum dan sesudah diterapkannya metode pembobotan ulang, menggunakan simulasi graf acak dengan algoritma Erdos-Renyi. Pendekatan ini memungkinkan analisis kuantitatif mengenai pengaruh pembobotan ulang terhadap kinerja Algoritma Dijkstra pada graf dengan bobot negatif, dengan menggunakan parameter seperti waktu eksekusi, perubahan bobot lintasan, jumlah iterasi, dan akurasi jalur terpendek.

3.2 Data dan Sumber Data

Data dalam penelitian ini diperoleh melalui simulasi graf berbobot yang dibangun menggunakan algoritma Erdos-Renyi. Graf yang digunakan terdiri dari 31 simpul dengan bobot sisi yang dapat bernilai negatif maupun positif. Data ini digunakan untuk menganalisis efektivitas pembobotan ulang dalam penerapan Algoritma Dijkstra untuk menentukan lintasan terpendek.

3.3 Tahapan Penelitian

Penelitian ini dilakukan melalui beberapa tahapan utama yang dirancang untuk menjawab permasalahan mengenai penerapan pembobotan ulang agar algoritma Dijkstra dapat diterapkan pada graf berbobot negatif. Adapun tahapan penelitian yang dilakukan adalah sebagai berikut:

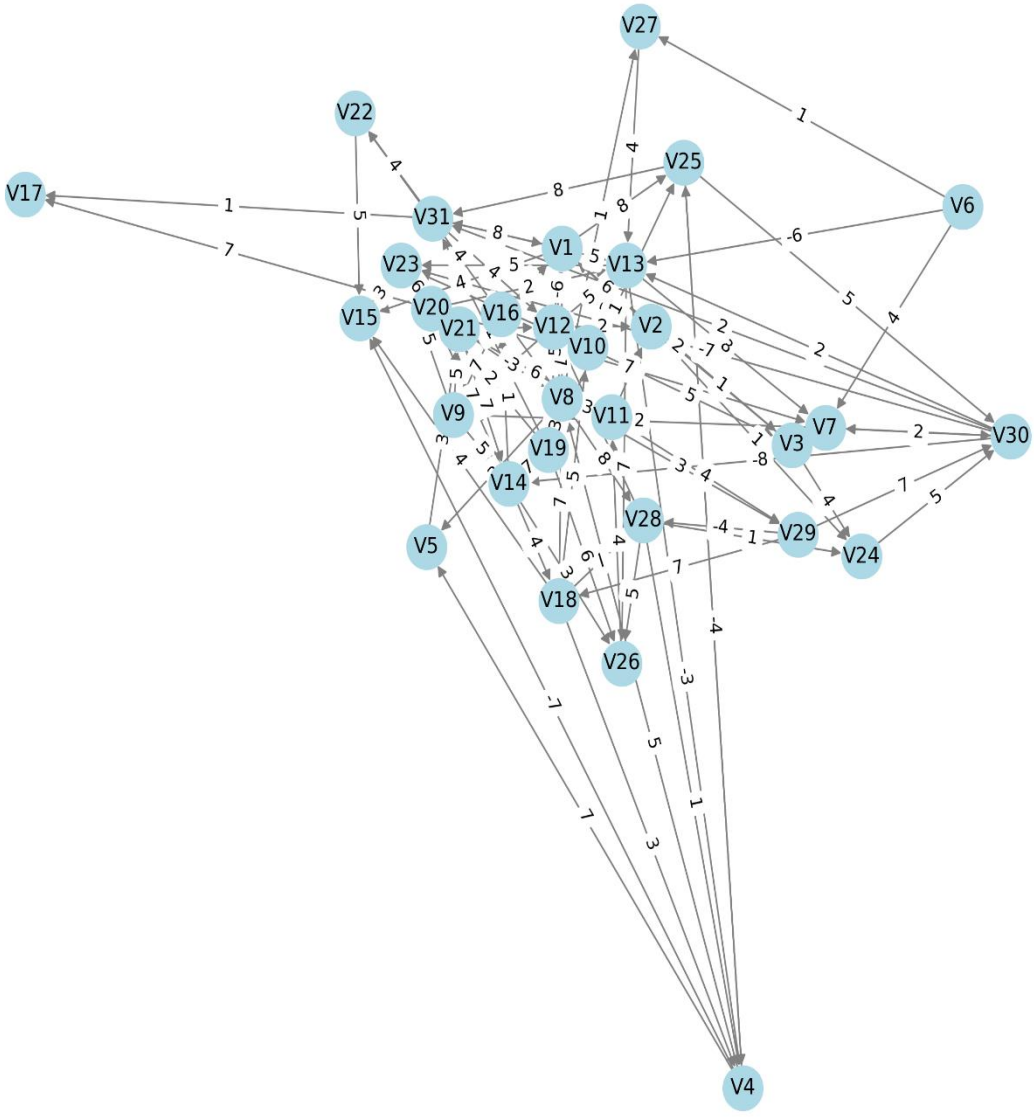
1. Menentukan data graf berbobot menggunakan algoritma Erdos-Renyi.
2. Menambahkan simpul baru (s) yang dihubungkan ke semua simpul dalam graf dengan bobot 0.
3. Menjalankan algoritma Bellman-Ford dari simpul s untuk menghitung nilai potensial $h(v)$ setiap simpul.
4. Melakukan pembobotan ulang pada setiap jalur $w'(u,v) = w(u,v) + h(u) - h(v)$, sehingga semua bobot menjadi non-negatif.
5. Menjalankan algoritma Dijkstra pada graf yang sudah dilakukan pembobotan ulang untuk menentukan lintasan terpendek.
6. Interpretasi dan Analisis Hasil
    a. Hasil dari penerapan algoritma Dijkstra pada graf yang telah dilakukan pembobotan ulang akan dibandingkan dengan hasil lintasan terpendek pada graf asli (sebelum pembobotan ulang).
    b. Analisis ini bertujuan untuk menguji apakah proses pembobotan ulang memungkinkan algoritma Dijkstra menemukan lintasan terpendek dengan benar, meskipun graf yang digunakan sebelumnya memiliki bobot negatif.

BAB IV

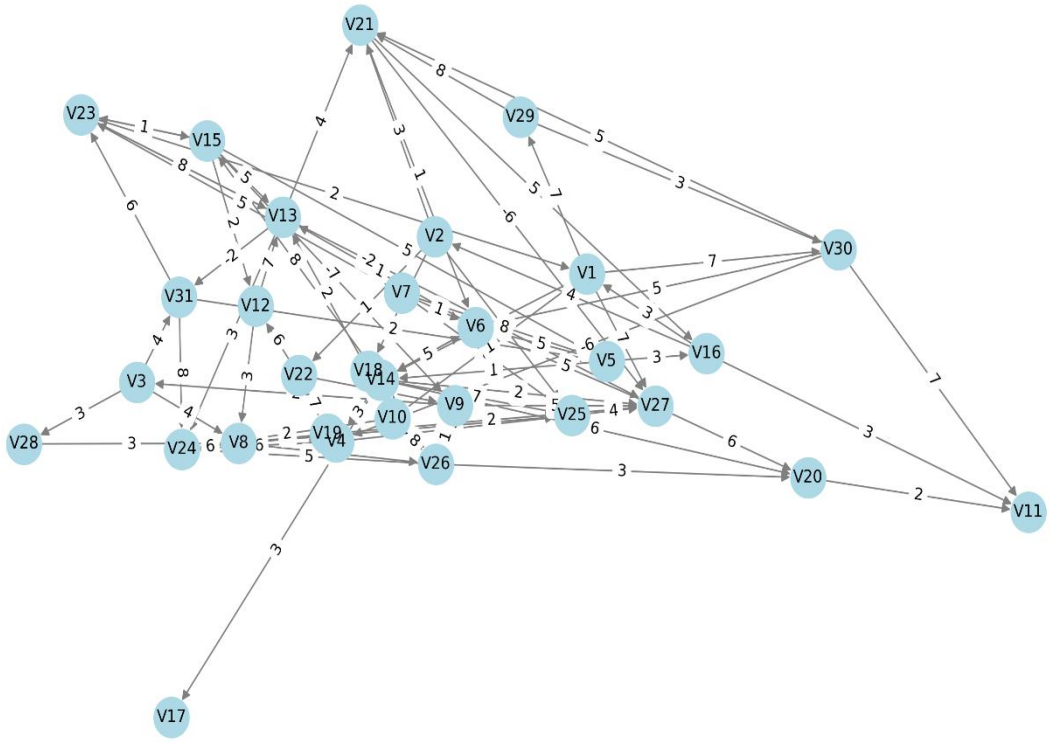
HASIL DAN PEMBAHASAN

4.1 Deskripsi Data

Penelitian ini menggunakan data yang dihasilkan secara acak melalui algoritma Erdos-Renyi, yang direpresentasikan dalam bentuk graf berbobot berarah dengan bobot pada setiap sisi dapat bernilai negatif maupun positif. Simulasi ini dilakukan untuk menguji efektivitas pembobotan ulang dalam memungkinkan Algoritma Dijkstra menangani bobot negatif yang secara teoritis tidak dapat diproses secara langsung oleh algoritma tersebut. Dalam penelitian ini, digunakan dua graf acak berbeda, yaitu graf acak pertama dan graf acak kedua, yang masing-masing memiliki variasi pada nilai bobot, simpul asal, dan simpul tujuan. Graf acak pertama memiliki sebaran bobot negatif dan positif yang berbeda dari graf acak kedua, serta konfigurasi koneksi antar simpul yang tidak sama, sehingga memberikan kondisi yang bervariasi untuk mengamati bagaimana pembobotan ulang memengaruhi hasil pencarian lintasan terpendek. Informasi lengkap mengenai data dari graf acak pertama dapat dilihat pada Lampiran 7, sedangkan data graf acak kedua disajikan dalam Lampiran 15. Representasi graf berbobot berarah yang dibangun menggunakan algoritma Erdos-Renyi ditampilkan pada Gambar 4.1 dan Gambar 4.2.



Gambar 4.1 Model Graf Acak Pertama



Gambar 4.2 Model Graf Acak Kedua

4.2 Penerapan Algoritma Johnson

Algoritma Johnson diterapkan untuk menentukan jarak terpendek antara setiap pasangan simpul dalam graf berbobot, termasuk yang memiliki bobot negatif. Algoritma ini mengombinasikan Algoritma Bellman-Ford dengan Algoritma Dijkstra guna memastikan bahwa bobot negatif dapat ditangani dengan pembobotan ulang sebelum pencarian jalur terpendek dilakukan.

1. Penambahan simpul baru dalam graf

   Graf asli diperluas dengan menambahkan simpul tambahan s, yang terhubung ke seluruh simpul yang ada dengan sisi berbobot nol.

$$s \rightarrow V_1 = 0$$

$$s \rightarrow V_2 = 0$$

$$s \rightarrow V_3 = 0$$

$$\vdots$$

$$s \rightarrow V_{31} = 0$$

2. Menjalankan Algoritma Bellman-Ford

   Algoritma Bellman-Ford dijalankan dari simpul s untuk menghitung nilai potensial $h(v)$ bagi setiap simpul v. Jika ditemukan siklus negatif, maka proses pencarian rute terpendek dihentikan karena graf tidak dapat dihitung dengan metode ini.

   a. Inisialisasi

      Langkah pertama dalam Algoritma Bellman-Ford yaitu melakukan inisialisasi terhadap semua simpul dalam graf. Inisialisasi ini bertujuan untuk menetapkan nilai awal jarak dari simpul sumber ke semua simpul lainnya sebelum dilakukan proses iterasi.

$$dist(s) = 0$$

$$dist(V_1) = \infty$$

$$dist(V_2) = \infty$$

$$\vdots$$

$$dist(V_{31}) = \infty$$

   b. Proses Iterasi

      Setelah proses inisialisasi selesai, dilakukan iterasi sebanyak $|V| - 1$ kali untuk memperbarui nilai jarak pada setiap simpul dalam graf. Pada setiap iterasi, dilakukan relaksasi terhadap semua sisi (u, v, w) dalam graf. Relaksasi bertujuan untuk memastikan bahwa nilai jarak dari simpul awal ke setiap simpul lainnya selalu dalam kondisi optimal. Adapun aturan relaksasi dalam Algoritma Bellman-Ford sebagai berikut.

Jika $dist(u) + w < dist(v)$, maka diperbarui menjadi

$$dist(v) = dist(u) + w$$

Artinya, jika ditemukan jalur baru menuju simpul v yang lebih pendek daripada jalur sebelumnya, maka nilai $dist(v)$ diperbarui.

Iterasi 1

Pada iterasi pertama, algoritma mulai mengevaluasi setiap simpul berdasarkan bobot sisi yang ada. Jika ditemukan jalur dengan bobot lebih kecil daripada nilai jarak sebelumnya, maka nilai $Dist(v)$ diperbarui. Dalam Tabel 4.1, beberapa simpul masih memiliki bobot nol karena belum dilakukan pembaruan nilai jarak yang signifikan. Selain itu, keberadaan bobot negatif menunjukkan perlunya proses pembobotan ulang sebelum Algorima Dijkstra dapat berfungsi dengan benar.

Tabel 4.1 Nilai Iterasi Pertama

Simpul Asal	Simpul Tujuan	Jarak	Bobot
V_1	V_{16}	$Dist(16)$	0
V_1	V_{26}	$Dist(26)$	0
V_2	V_4	$Dist(4)$	0
⋮	⋮	⋮	⋮
V_{29}	V_{19}	$Dist(19)$	-2
V_{29}	V_{25}	$Dist(25)$	-7
V_{31}	V_{18}	$Dist(18)$	0

Nilai iterasi pertama dapat dilihat pada Lampiran 8.

Iterasi 2

Pada proses iterasi kedua, proses relaksasi dalam Algoritma Bellman-Ford kembali dilakukan untuk memperbarui nilai jarak setiap simpul dalam graf berbobot negatif. Pada tahap ini, setiap simpul diperiksa ulang untuk memastikan apakah ada jalur baru yang lebih pendek dibandingkan dengan jarak yang telah tercatat sebelumnya. Jika ditemukan jalur

dengan bobot yang lebih kecil, maka nilai jarak diperbarui agar lebih optimal. Hasil dari iterasi kedua menunjukkan bahwa beberapa simpul mengalami perubahan jarak akibat adanya jalur yang lebih efisien yang ditemukan dalam proses relaksasi. Selain itu, masih terdapat bobot negatif dalam graf, yang menunjukkan bahwa proses pembobotan ulang tetap diperlukan agar Algoritma Dijkstra dapat diterapkan dengan benar. Proses iterasi ini akan terus berlangsung hingga tidak ada lagi perubahan nilai jarak, yang menandakan bahwa jalur terpendek telah ditemukan secara optimal.

Tabel 4.2 Nilai Iterasi Kedua

Simpul Asal	Simpul Tujuan	Jarak	Bobot
V_1	V_{16}	$Dist(16)$	0
V_1	V_{26}	$Dist(26)$	-6
V_2	V_4	$Dist(4)$	0
⋮	⋮	⋮	⋮
V_{29}	V_{19}	$Dist(19)$	-2
V_{29}	V_{25}	$Dist(25)$	-7
V_{31}	V_{18}	$Dist(18)$	0

Nilai iterasi kedua dapat dilihat pada Lampiran 9.

Iterasi 3

Pada iterasi ketiga, proses relaksasi dalam Algoritma Bellman-Ford kembali dilakukan dengan prinsip yang sama seperti iterasi sebelumnya, yaitu memeriksa setiap sisi dalam graf dan memperbarui jarak jika ditemukan jalur yang lebih pendek. Namun, hasil iterasi ketiga menunjukkan bahwa tidak ada perubahan nilai jarak dibandingkan dengan iterasi kedua. Hal ini menunjukkan bahwa proses relaksasi telah mencapai kestabilan, di mana semua jalur terpendek telah ditemukan dan tidak ada lagi pembaruan nilai jarak yang diperlukan. Dengan demikian,

iterasi berikutnya tidak lagi menghasilkan perubahan, sehingga iterasi dapat dihentikan lebih awal karena telah mencapai konvergensi.

Tabel 4.3 Nilai Iterasi Ketiga

Simpul Asal	Simpul Tujuan	Jarak	Bobot
V_1	V_{16}	$Dist(16)$	0
V_1	V_{26}	$Dist(26)$	-6
V_2	V_4	$Dist(4)$	0
⋮	⋮	⋮	⋮
V_{29}	V_{19}	$Dist(19)$	-2
V_{29}	V_{25}	$Dist(25)$	-7
V_{31}	V_{18}	$Dist(18)$	0

Nilai iterasi ketiga dapat dilihat pada Lampiran 10.

Pada tahap ini, nilai potensial $h(v)$ untuk setiap simpul telah diperoleh dari hasil akhir Algoritma Bellman-Ford. Nilai ini mencerminkan jarak minimum dari simpul sumber tambahan ke setiap simpul dalam graf, yang selanjutnya digunakan untuk melakukan pembobotan ulang agar bobot sisi menjadi non-negatif sebelum penerapan Algoritma Dijkstra.

Tabel 4.4 Nilai Potensial $h(v)$

Nilai Potensial $h(v)$	Bobot
$h(16)$	0
$h(26)$	-6
$h(4)$	0
⋮	⋮
$h(19)$	-2
$h(25)$	-7
$h(18)$	0

Nilai potensial $h(v)$ dapat dilihat pada Lampiran 11.

3. Proses Pembobotan Ulang

Setelah menjalankan Algoritma Bellman-Ford dari simpul tambahan s, diperoleh nilai potensial $h(v)$ untuk setiap simpul dalam graf. Nilai ini digunakan untuk melakukan pembobotan ulang pada setiap sisi dalam graf

guna memastikan bahwa semua bobot menjadi non-negatif. Proses pembobotan ulang dilakukan dengan menerapkan transformasi bobot menggunakan rumus:

$$w'(u,v) = w(u,v) + h(u) - h(v)$$

$$w'(V_1, V_{16}) = 2 + 0 - 0$$

$$w'(V_1, V_{16}) = 2$$

Dengan transformasi ini, bobot negatif yang terdapat dalam graf asli diubah menjadi bobot non-negatif tanpa mengubah jalur terpendek yang sebenarnya. Hal ini memungkinkan Algoritma Dijkstra digunakan secara optimal untuk menentukan lintasan terpendek pada graf yang sebelumnya mengandung bobot negatif. Setelah proses pembobotan ulang selesai, langkah selanjutnya adalah memverifikasi apakah semua bobot telah menjadi non-negatif, yang kemudian akan ditampilkan dalam tabel berikut.

Tabel 4.5 Hasil Pembobotan Ulang

Simpul Asal	Simpul Tujuan	Bobot Baru
V_1	V_{16}	2
V_1	V_{26}	11
V_2	V_4	1
⋮	⋮	⋮
V_{29}	V_{19}	6
V_{29}	V_{25}	8
V_{31}	V_{18}	7

Hasil pembobotan ulang dapat dilihat pada Lampiran 12.

4. Menjalankan Algoritma Dijkstra

Setelah seluruh bobot pada graf diubah menjadi non-negatif melalui proses pembobotan ulang, langkah selanjutnya adalah menerapkan Algoritma Dijkstra untuk menentukan lintasan terpendek dari simpul sumber ke seluruh simpul lainnya. Dengan bobot yang telah dimodifikasi, Algoritma Dijkstra

dapat bekerja secara optimal tanpa terpengaruh oleh nilai negatif.

a. Inisialisasi

Langkah pertama dalam penerapan Algoritma Dijkstra adalah melakukan inisialisasi terhadap setiap simpul dalam graf. Inisialisasi dilakukan dengan memberikan nilai jarak awal sebesar 0 pada simpul sumber, sedangkan semua simpul lainnya diberi nilai jarak tak hingga (∞). Hal ini bertujuan untuk memastikan bahwa algoritma dapat membandingkan dan memperbarui nilai jarak yang lebih kecil secara iteratif.

Tabel 4.6 Inisialisasi Algoritma Dijkstra

Simpul	Jarak Awal	Status
V_1	0	Sudah dikunjungi (simpul sumber)
V_2	∞	Belum dikunjungi
V_3	∞	Belum dikunjungi
V_4	∞	Belum dikunjungi
V_5	∞	Belum dikunjungi
⋮	⋮	⋮
V_{27}	∞	Belum dikunjungi
V_{28}	∞	Belum dikunjungi
V_{29}	∞	Belum dikunjungi
V_{30}	∞	Belum dikunjungi
V_{31}	∞	Belum dikunjungi

Inisialisasi Algoritma Dijkstra dapat dilihat pada Lampiran 13. Setelah proses inisialisasi selesai, Algoritma Dijkstra dijalankan untuk menentukan lintasan terpendek dari simpul sumber ke seluruh simpul lainnya. Proses ini dilakukan secara iteratif, di mana pada setiap langkah dipilih simpul dengan jarak terpendek yang belum diproses, kemudian dilakukan pembaruan (relaksasi) terhadap jarak menuju simpul-simpul tetangganya.

b. Proses Relaksasi

Relaksasi dilakukan untuk memperbarui nilai jarak apabila ditemukan lintasan baru yang lebih pendek dari jarak sebelumnya. Jika terdapat lintasan alternatif menuju simpul tertentu dengan total bobot lebih kecil, maka nilai jarak lama digantikan dengan nilai baru. Proses ini diulangi hingga semua simpul telah dikunjungi.

Tabel 4.7 Proses Relaksasi Algoritma Dijkstra

Ite-rasi	Simpul diproses	Simpul diper-barui	Jarak sebelu mnya	Jarak terpe-ndek	Lintasan	Ketera-ngan
1	V_{10}	V_{20}	∞	4	$V_{10} \rightarrow V_{26}$	Diperbarui
1	V_{10}	V_{22}	∞	7	$V_{10} \rightarrow V_{22}$	Diperbarui
1	V_{10}	V_{4}	∞	8	$V_{10} \rightarrow V_{4}$	Diperbarui
1	V_{10}	V_{9}	∞	3	$V_{10} \rightarrow V_{9}$	Diperbarui
2	V_{9}	V_{18}	∞	5	$V_{10} \rightarrow V_{9} \rightarrow V_{18}$	Diperbarui
3	V_{20}	V_{14}	∞	9	$V_{1} \rightarrow V_{16} \rightarrow V_{4} \rightarrow V_{30}$	Diperbarui
3	V_{20}	V_{21}	∞	8	–	Tidak Diperbarui
3	V_{20}	V_{23}	∞	7	–	Tidak Diperbarui
3	V_{20}	V_{25}	∞	17	$V_{1} \rightarrow V_{26} \rightarrow V_{20}$	Diperbarui
⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	V_{27}	V_{29}	∞	27	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_{7} \rightarrow V_{8} \rightarrow V_{27} \rightarrow V_{29}$	Diperbarui
28	V_{29}	V_{19}	13	33	–	Tidak Diperbarui
28	V_{29}	V_{25}	15	35	–	Tidak Diperbarui

Proses relaksai Algoritma Dijkstra dapat dilihat pada Lampiran 14. Berdasarkan Tabel 4.9, proses relaksasi dalam penerapan Algoritma

Dijkstra menunjukkan bahwa jarak dari simpul sumber ke simpul lainnya mengalami pembaruan secara bertahap. Pada setiap iterasi, simpul dengan jarak terkecil yang belum dikunjungi dipilih, lalu dilakukan pembaruan nilai jarak (relaksasi) ke simpul-simpul tetangganya jika ditemukan lintasan yang lebih efisien. Proses ini berlangsung hingga tidak ada lagi nilai jarak yang dapat diperbarui, yang menandakan bahwa algoritma telah mencapai konvergensi dan seluruh lintasan terpendek berhasil ditemukan secara optimal.

Keberhasilan proses relaksasi ini menandakan bahwa pembobotan ulang menggunakan Algoritma Johnson telah berhasil mengubah seluruh bobot sisi negatif menjadi non-negatif tanpa mengubah struktur solusi optimal. Dengan demikian, Algoritma Dijkstra dapat berjalan secara efektif dan menghasilkan lintasan terpendek yang valid pada graf yang sebelumnya mengandung bobot negatif.

Salah satu hasil yang ditinjau secara khusus dalam penelitian ini adalah lintasan terpendek dari simpul V_{10} ke simpul V_{31} diperoleh melalui jalur:

$$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_{31}$$

Lintasan tersebut merupakan jalur dengan bobot total setelah dilakukan pembobotan ulang yang paling minimum di antara seluruh alternatif lintasan yang memungkinkan. Total bobot lintasan ini adalah 16. Hasil ini memperkuat bahwa penerapan pembobotan ulang tidak hanya menyelesaikan permasalahan nilai bobot negatif, tetapi juga tetap menjaga keakuratan hasil perhitungan lintasan terpendek. Dengan demikian, metode ini memungkinkan Algoritma Dijkstra digunakan

secara lebih fleksibel, bahkan pada graf yang kompleks dan mengandung sisi berbobot negatif.

Dengan tahapan yang sama menggunakan graf acak yang memiliki bobot berbeda, pada graf acak kedua proses relaksasi menggunakan Algoritma Bellman-Ford untuk memperoleh nilai potensial $h(v)$ selesai pada iterasi ke-delapan. Setelah nilai potensial diperoleh dan graf dilakukan pembobotan ulang, lintasan terpendek dari simpul V_{15} ke simpul V_{31} kemudian diperoleh menggunakan Algoritma Dijkstra melalui jalur:

$$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31}$$

Lintasan tersebut merupakan jalur dengan bobot total setelah dilakukan pembobotan ulang yang paling minimum di antara seluruh alternatif lintasan yang memungkinkan. Total bobot lintasan ini adalah 1. Adapun tahapan penyelesaiannya dapat dilihat pada Lampiran 15 hingga Lampiran 27.

4.3 Perbandingan Lintasan Terpendek Algoritma Bellman-Ford dengan Algoritma Dijkstra

Keakuratan hasil lintasan terpendek setelah dilakukan pembobotan ulang menggunakan Algoritma Johnson diuji dengan membandingkannya terhadap hasil dari Algoritma Bellman-Ford yang diterapkan secara langsung. Pemilihan Algoritma Bellman-Ford didasarkan pada kemampuannya menangani bobot negatif tanpa proses pembobotan ulang. Setelah dilakukan pembobotan ulang menggunakan Algoritma Johnson dan diterapkan pada Algoritma Dijkstra, lintasan terpendek yang dihasilkan pada kedua graf dibandingkan untuk setiap pasangan simpul.

Perbandingan dilakukan dengan menghitung total panjang lintasan yang dihasilkan oleh kedua algoritma menggunakan bobot pada graf asli sebelum pembobotan ulang. Meskipun lintasan dari Algoritma Dijsktra diperoleh melalui graf yang telah dilakukan pembobotan ulang, penilaian terhadap panjang lintasannya tetap mengacu pada graf asli. Langkah ini bertujuan untuk memastikan bahwa lintasan terpendek yang diperoleh dari Algoritma Dijkstra setelah pembobotan ulang tetap mencerminkan struktur optimal yang sama dengan lintasan hasil Algoritma Bellman-Ford. Apabila total panjang lintasan dari kedua algoritma menunjukkan kesamaan nilai untuk setiap pasangan simpul, maka dapat disimpulkan bahwa pembobotan ulang tidak memengaruhi keakuratan hasil lintasan terpendek.

Tabel 4.8 Perbandingan Lintasan Terpendek Bellman-Ford dan Dijkstra pada Graf Acak Pertama

No.	Simpul Asal → Simpul Tujuan	Algoritma Bellman-Ford	Algoritma Dijkstra	Selisih Panjang Rute
1.	$V_1 \rightarrow V_1$	0	0	0
2.	$V_1 \rightarrow V_{16}$	2	2	0
3.	$V_1 \rightarrow V_{26}$	11	11	0
4.	$V_1 \rightarrow V_2$	20	20	0
5.	$V_1 \rightarrow V_4$	4	4	0
⋮	⋮	⋮	⋮	⋮
957.	$V_{31} \rightarrow V_{14}$	19	19	0
958.	$V_{31} \rightarrow V_{29}$	39	39	0
959.	$V_{31} \rightarrow V_{21}$	18	18	0
960.	$V_{31} \rightarrow V_{28}$	30	30	0
961.	$V_{31} \rightarrow V_{23}$	17	17	0
Total Selisih				0

Tabel 4.9 Perbandingan Lintasan Terpendek Bellman-Ford dan Dijkstra pada Graf Acak Kedua

No.	Simpul Asal → Simpul Tujuan	Algoritma Bellman-Ford	Algoritma Dijkstra	Selisih Panjang Rute
1.	$V_1 \rightarrow V_1$	0	0	0
2.	$V_1 \rightarrow V_{11}$	6	6	0
3.	$V_1 \rightarrow V_{15}$	7	7	0

4.	$V_1 \rightarrow V_{23}$	0	0	0
5.	$V_1 \rightarrow V_{26}$	6	6	0
⋮	⋮	⋮	⋮	⋮
957.	$V_{31} \rightarrow V_{16}$	6	6	0
958.	$V_{31} \rightarrow V_{20}$	6	6	0
959.	$V_{31} \rightarrow V_{19}$	4	4	0
960.	$V_{31} \rightarrow V_{24}$	9	9	0
961.	$V_{31} \rightarrow V_{27}$	12	12	0
Total Selisih				0

Hasil menunjukkan bahwa lintasan terpendek yang diperoleh dari metode pembobotan ulang sama dengan hasil dari Algoritma Bellman-Ford secara langsung. Hal ini membuktikan bahwa transformasi bobot pada Algoritma Johnson tidak mengubah struktur optimal dari lintasan terpendek. Adapun data tabel secara lengkap dapat dilihat pada Lampiran 28 dan Lampiran 29.

4.4 Pembobotan dalam Pandangan Islam

Allah SWT dalam Al-Qur'an telah menjanjikan bahwa setiap permasalahan yang terjadi di muka bumi ini pasti memiliki jalan penyelesaian. Janji ini tidak hanya berlaku dalam ranah kehidupan sosial dan spiritual, tetapi juga mencakup segala bentuk ilmu pengetahuan yang dikaruniakan-Nya kepada manusia, termasuk dalam bidang matematika. Sebagaimana firman Allah dalam Surah Al-Insyirah ayat 5-6, yang mengandung makna mendalam bahwa tidak ada kesulitan yang tidak disertai dengan solusi yang Allah siapkan. Dalam konteks penelitian ini, kesulitan yang dimaksud adalah keterbatasan Algoritma Dijkstra dalam menyelesaikan permasalahan pencarian lintasan terpendek pada graf yang mengandung bobot negatif. Algoritma ini, secara konvensional, hanya dapat berfungsi pada graf dengan bobot sisi yang tidak negatif, sehingga tidak dapat digunakan secara langsung jika terdapat bobot negatif. Permasalahan ini menjadi hambatan serius

karena banyak permasalahan dunia nyata, seperti jaringan transportasi atau sistem keuangan yang mengandung bobot negatif.

Namun, sebagaimana janji Allah, bahwa di balik kesulitan pasti ada kemudahan, solusi terhadap keterbatasan ini dapat ditemukan melalui pendekatan pembobotan ulang. Dengan menerapkan Algoritma Johnson, bobot negatif dalam graf dapat diubah menjadi bobot non-negatif tanpa mengubah struktur solusi optimalnya. Inilah bentuk nyata dari pertolongan dan kemudahan yang Allah janjikan, yakni dengan memberikan manusia akal dan ilmu untuk menemukan penyelesaian atas persoalan yang ada. Oleh karena itu, metode pembobotan ulang ini tidak hanya merupakan solusi teknis dalam bidang matematika, tetapi juga merupakan manifestasi dari janji Allah SWT bahwa setiap kesulitan, pasti ada jalan keluarnya.

Nilai-nilai keislaman dalam penelitian ini juga merujuk pada QS. Al-Mujadilah ayat 11, yang menyatakan bahwa Allah akan mengangkat derajat orang-orang yang beriman dan berilmu. Hal ini menekankan bahwasannya ilmu memiliki peran penting dalam meraih kemuliaan. Oleh karena itu, metode pembobotan ulang dalam penelitian ini tidak hanya berfungsi sebagai solusi matematis, akan tetapi juga mencerminkan semangat Islam dalam menghadapi tantangan melalui ikhtiar, pemanfaatan ilmu pengetahuan, serta keyakinan bahwa usaha yang dilandasi iman akan mendatangkan keberkahan dan hasil yang lebih bermakna.

Dengan demikian, integrasi nilai-nilai Islam dalam penelitian ini bukan sekadar pelengkap, tetapi merupakan landasan filosofis yang memberikan arah dan makna terhadap aktivitas ilmiah. Penelitian ini menjadi contoh bahwa dalam menghadapi keterbatasan teknis maupun metodologis, pendekatan yang dilandasi

oleh nilai spiritual, keilmuan, dan akhlak Islami dapat melahirkan solusi yang efektif, efisien, dan bermanfaat. Oleh karena itu, pembobotan ulang melalui Algoritma Johnson tidak hanya menyelesaikan permasalahan teknis dalam pencarian lintasan terpendek, tetapi juga merepresentasikan nilai-nilai Islam dalam menyelesaikan persoalan dengan hikmah dan ilmu.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil penelitian yang dilakukan, dapat disimpulkan bahwa pembobotan ulang pada graf berbobot negatif dapat dilakukan dengan menggunakan Algoritma Johnson, sehingga Algoritma Dijkstra dapat diterapkan secara efektif untuk menentukan lintasan terpendek. Hasil penerapan Algoritma Dijkstra menunjukkan bahwa lintasan terpendek tetap dapat ditemukan secara akurat, sebagaimana ditunjukkan pada graf pertama dari simpul V_{10} ke V_{31} melalui lintasan $V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_{31}$ dengan total bobot yaitu 16. Sementara itu, pada graf kedua dari simpul V_{15} ke V_{31} melalui lintasan $V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31}$ dengan total bobot yaitu 1. Dengan demikian, pembobotan ulang terbukti efektif dalam mengatasi kendala bobot negatif pada graf tanpa mengubah struktur solusi optimal. Selain itu, hasil lintasan terpendek yang diperoleh setelah dilakukan pembobotan ulang menggunakan Algoritma Johnson memiliki kesesuaian yang sempurna dengan hasil lintasan yang dihitung langsung menggunakan Algoritma Bellman-Ford. Hal ini menunjukkan bahwa metode pembobotan ulang tidak memengaruhi keakuratan hasil perhitungan, namun memungkinkan pemanfaatan Algoritma Dijkstra yang lebih efisien. Dengan demikian, pendekatan ini tidak hanya efektif secara teoritis, tetapi juga terbukti secara praktis dalam menjaga keakuratan dan efisien pencarian lintasan terpendek pada graf berbobot negatif.

5.2 Saran

Berdasarkan hasil penelitian yang diperoleh, peneliti selanjutnya diharapkan dapat menggunakan algoritma lintasan terpendek lainnya untuk membandingkan hasil dan efektivitas dalam menentukan lintasan terpendek pada graf berbobot negatif.

DAFTAR RUJUKAN

Al-Mahalli, & As-Suyuthi. (n.d.). *Tafsir Al-Jalalain*.

Bawole, D. J., & Chernovita, H. P. (2019). Algoritma Bellman-Ford untuk Menentukan Jalur Terpendek dalam Survey Klaim Asuransi (Studi Kasus : PT. Asuransi Sinar Mas, Jakarta). *INOBIS: Jurnal Inovasi Bisnis Dan Manajemen Indonesia*, *3*(1), 41–51.

Bunaen, M. C., Pratiwi, H., & Riti, Y. F. (2022). Penerapan Algoritma Dijkstra Untuk Menentukan RuteTerpendek Dari Pusat Kota Surabaya Ke Tempat Bersejarah. *Jurnal Teknologi Dan Sistem Informasi Bisnis*, *4*(1), 213–223. https://media.neliti.com/media/publications/441390-application-of-the-dijkstra-algorithm-to-f1576853.pdf

Chartrand, G., Lesniak, L., & Zhang, P. (2011). Graphs & Digraphs (5th ed.). *CRC Press Taylor & Francis Group*.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms Third Edition. *Edition, Second*. https://edutechlearners.com/download/Introduction_to_algorithms-3rd Edition.pdf

Gozali, W., & Aji, A. F. (2017). *Pemrograman Kompetitif Dasar: Panduan Memulai OSN Informatika, ACM-ICPC, dan Sederajat*. https://osn.toki.id/data/pemrograman-kompetitif-dasar.pdf

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions of Systems Science and Cybernetics*, *4*(2), 100–107.

Ilhamsyah, M., Mangkudjaja, R., & Hertiana, S. N. (2016). Simulasi dan Uji Kinerja Algoritma Johnson untuk Penentuan Rute Terbaik Pada Jaringan Software Defined Network. *Jurnal Sistem Komputer*, *6*(2), 77–79.

Johnson, D. B. (1977). Efficient Algorithms for Shortest Paths in Sparse Networks. *Journal of the Association for Computing Machinery*, *24*(1), 1–13.

Junior, D. P., & Wille, E. C. G. (2018). FB-APSP: A new efficient algorithm for computing all-pairs shortest-paths. *Journal of Network and Computer Applications*, *121*, 33–43.

Kementrian Agama RI. (2017). *Al-Quran dan Terjemah dilengkapi Panduan waqaf & Ibtida'*. PT. SuaraAgung.

Knuth, D. E. (1997). *The Art of Computer Programming Volume 1: Fundamental Algorithms*. Addison-Wesley L. http://www-cs-faculty.standford.edu/~knuth/tacop.html

Mashudi, K. (2019). Telaah Tafsir Al-Muyassar Jilid VI. *Inteligensia Media*.

Maulana, G. G. (2017). Pembelajaran Dasar Algoritma dan Pemrograman Menggunakan El-Goritma Berbasis Web. *Jurnal Teknik Mesin (JTM)*, *06*(2), 69–73.

Nugraha, D. W. (2011). Aplikasi Algoritma Prim untuk Menentukan Minimum Spanning Tree Suatu Graf Berbobot dengan Menggunakan Pemrograman Berorientasi Objek. *Ilmiah Foristek*, *1*(2), 70–79. deny_wiria_nugraha@yahoo.co.id

Rahayuningsih, S. (2018). Teori Graph dan Penerapannya. *Universitas Wisnuwardhana Press Malang (Unidha Press)*. https://srirahayuningsih82.wordpress.com/wp-content/uploads/2019/02/buku-ajar-teori-graph.pdf

Salaki, D. T. (2011). Penentuan Lintasan Terpendek Dari FMIPA ke Rektorat dan Fakultas Lain di UNSRAT Manado Menggunakan Algoritma Dijkstra. *Jurnal Ilmiah Sains*, *11*(1), 73–76.

Shihab, M. Q. (2002a). *Tafsir Al-Misbah: pesan, kesan dan keserasian Al-Qur'an Volume 14*. Lenter Hati.

Shihab, M. Q. (2002b). *Tafsir Al-Misbah: pesan, kesan dan keserasian Al-Qur'an Volume 15*. Lentera Hati.

Sunardi, Yudhana, A., & Kadim, A. A. (2019). Implementasi Algoritma Dijkstra Untuk Analisis Rute Transportasi Umum Transjogja Berbasis Android. *Jurnal Sistem Informasi Bisnis*, *9*(1), 32–38.

LAMPIRAN

Lampiran 1. Program Membuat Graf Acak Pertama

```
import networkx as nx
import random
import pickle
import matplotlib.pyplot as plt

# Jumlah simpul dan probabilitas sisi
n = 31
p = 0.1

# Gunakan seed untuk konsistensi graf
random.seed(90)
G = nx.erdos_renyi_graph(n, p, directed=True, seed=90)

# Gunakan seed lagi sebelum menambahkan bobot agar bobot tetap sama
random.seed(74)

# Tambahkan bobot acak dengan urutan yang dikunci
for u, v in sorted(G.edges()):  # Mengunci urutan edge agar tetap sama
    weight = random.randint(1, 8)

    # Sekitar 10% sisi mendapatkan bobot negatif
    if random.random() < 0.1:
        weight *= -1

    G[u][v]['weight'] = weight

# Simpan graf menggunakan pickle
with open("graf_G.pkl", "wb") as f:
    pickle.dump(G, f)

# Menampilkan rincian sisi dan bobot
node_labels = {i: f"V{i+1}" for i in range(n)}

print("\nRincian sisi dan bobotnya:")
for u, v, data in G.edges(data=True):
    print(f"Sisi {node_labels[u]} → {node_labels[v]} dengan bobot
{data['weight']}")

# Menampilkan graf
plt.figure(figsize=(26, 25))
pos = nx.spring_layout(G, seed=1500)  # Mengunci layout

nx.draw(G, pos, with_labels=True, labels=node_labels,
node_color='lightblue', node_size=2400, edge_color='gray', arrows=True)

edge_labels = {(u, v): G[u][v]['weight'] for u, v in G.edges()}
```

```
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_size=26)

plt.title("Graf Acak Berbobot (Erdos-Renyi Model)")
plt.show()

print("\nGraf telah disimpan sebagai 'graf_G.pkl'.")
```

Lampiran 2. Program untuk Penerapan Algoritma Johnson Graf Acak Pertama

```
import networkx as nx
import pickle
# Fungsi Bellman-Ford untuk menghitung jarak terpendek dari
sumber
def bellman_ford(g, source):
    # Menghitung jarak terpendek dari sumber ke semua simpul
lainnya
    dist = {node: float('inf') for node in g.nodes}
    dist[source] = 0
    for _ in range(len(g.nodes) - 1):
        for u, v, weight in g.edges(data='weight'):
            if dist[u] + weight < dist[v]:
                dist[v] = dist[u] + weight
    # Deteksi siklus negatif
    for u, v, weight in g.edges(data='weight'):
        if dist[u] + weight < dist[v]:
            raise ValueError("Graf mengandung siklus
negatif")
    return dist
# Fungsi Dijkstra untuk mencari lintasan terpendek
def dijkstra(g, source):
    # Menghitung jarak terpendek dari simpul sumber ke semua
simpul lainnya
    return nx.single_source_dijkstra_path_length(g, source,
weight='weight')
# Fungsi untuk melakukan re-weighting pada graf
def reweight_graph(g, h):
    new_g = nx.DiGraph()
    for u, v, weight in g.edges(data='weight'):
        # Bobot baru dihitung dengan rumus
        new_weight = weight + h[u] - h[v]
        new_g.add_edge(u, v, weight=new_weight)
    return new_g
# Fungsi algoritma Johnson untuk mencari jarak terpendek
def johnson_algorithm(g):
    # Tambahkan node sumber tambahan 'S'
    g.add_node('S')
    for node in g.nodes:
        if node != 'S':
            g.add_edge('S', node, weight=0)
    # Hitung nilai potensi dengan Bellman-Ford
    try:
        h = bellman_ford(g, 'S')
    except ValueError as e:
        print(e)
        return None
```

```
    # Hapus node sumber tambahan 'S'
    g.remove_node('S')
    # Buat graf baru dengan bobot yang telah di-reweight
    reweighted_g = reweight_graph(g, h)
    # Tampilkan hasil re-weighting
    print(f"Nilai potensi (h): {h}")
    print("Nilai re-weighting untuk setiap edge:")
    for u, v, weight in reweighted_g.edges(data='weight'):
        print(f"{u} → {v} = {weight}")
    # Menerapkan Algoritma Dijkstra pada graf yang sudah di-
reweight
    shortest_paths = {}
    for node in g.nodes:
        # Menjalankan Dijkstra pada graf yang telah
dimodifikasi
        dijkstra_result = dijkstra(reweighted_g, node)
        # Menampilkan hasil jarak terpendek setelah Dijkstra
        shortest_paths[node] = dijkstra_result
    return shortest_paths
# Memuat graf yang telah disimpan
with open("graf_G.pkl", "rb") as f:
    G_loaded = pickle.load(f)
# Jalankan algoritma Johnson untuk mendapatkan jarak
terpendek
result = johnson_algorithm(G)
if result:
    print("\nJarak terpendek dari setiap simpul setelah
Dijkstra (setelah re-weighting):")
    for node, paths in result.items():
        print(f"Dari {node}: {paths}")
```

Lampiran 3. Program untuk Menentukan Lintasan Terpendek Algoritma Dijkstra dari V_{10} ke V_{31}Pada Graf Acak Pertama

```
import networkx as nx
import heapq

# Modifikasi fungsi Dijkstra untuk menyertakan lintasan pada
setiap iterasi
def dijkstra_manual_with_paths(g, source):
    dist = {node: float('inf') for node in g.nodes}
    dist[source] = 0
    visited = set()
    pq = [(0, source)]

    prev = {node: None for node in g.nodes}  # untuk
pelacakan lintasan
    logs = []
    logs.append(f">>> Mulai Dijkstra dari {source}")
    iteration = 1

    while pq:
        current_dist, current_node = heapq.heappop(pq)

        if current_node in visited:
            continue
        visited.add(current_node)

        logs.append(f"\n[Iterasi ke-{iteration}] Memproses
node {current_node} (jarak = {current_dist})")
        iteration += 1

        for neighbor in g.neighbors(current_node):
            weight = g[current_node][neighbor]['weight']
            distance = current_dist + weight

            if distance < dist[neighbor]:
                prev[neighbor] = current_node
                dist[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))

                # Rekonstruksi lintasan sementara
                path = []
                temp = neighbor
                while temp is not None:
                    path.insert(0, temp)
                    temp = prev[temp]
                logs.append(f" - Update: {neighbor} ←
{current_node}, jarak baru = {distance}, lintasan = {' →
```

```
'.join(path)}")
            else:
                logs.append(f" - Lewati: {neighbor} dengan
jalur {current_node} → {neighbor} ({distance}) (lebih besar
dari {dist[neighbor]})")

    return dist, prev, logs

# Buat graf
G = nx.DiGraph()
edges = [
    ('V1', 'V16', 2), ('V1', 'V26', 11), ('V2', 'V4', 1),
    ('V2', 'V19', 7),  ('V3', 'V9', 5),  ('V3', 'V24', 6),
    ('V4', 'V30', 11), ('V5', 'V2', 4), ('V5', 'V7', 4),
    ('V6', 'V2', 10), ('V6', 'V17', 5), ('V6', 'V25', 0),
    ('V6', 'V31', 4), ('V7', 'V8', 3), ('V7', 'V18', 4),
    ('V8', 'V26', 0), ('V8', 'V27', 1),('V9', 'V18', 2),
    ('V10', 'V4', 8), ('V10', 'V9', 3), ('V10', 'V20', 4),
    ('V10', 'V22', 7), ('V11', 'V6', 7), ('V11', 'V10', 4),
    ('V11', 'V15', 1), ('V12', 'V14', 11), ('V12', 'V15', 3),
    ('V12', 'V19', 7), ('V13', 'V22', 5), ('V13', 'V26', 11),
    ('V13', 'V29', 5), ('V13', 'V30', 16), ('V14', 'V1', 1),
    ('V15', 'V16', 3), ('V15', 'V21', 6), ('V15', 'V26', 10),
    ('V15', 'V28', 0),  ('V16', 'V4', 2), ('V16', 'V18', 4),
    ('V16', 'V26', 11), ('V17', 'V2', 0), ('V17', 'V21', 8),
    ('V17', 'V23', 2), ('V17', 'V30', 15), ('V18', 'V16', 7),
    ('V18', 'V26', 7), ('V19', 'V6', 2), ('V19', 'V7', 5),
    ('V19', 'V17', 1), ('V19', 'V22', 1), ('V19', 'V31', 3),
    ('V20', 'V3', 6), ('V20', 'V14', 5), ('V20', 'V21', 4),
    ('V20', 'V23', 3), ('V20', 'V25', 13), ('V20', 'V26',
10),
    ('V21', 'V4', 3), ('V21', 'V9', 7), ('V21', 'V23', 2),
    ('V22', 'V9', 1), ('V22', 'V15', 6), ('V23', 'V20', 2),
    ('V23', 'V22', 7), ('V23', 'V24', 2), ('V24', 'V5', 7),
    ('V24', 'V14', 0), ('V24', 'V17', 4), ('V25', 'V9', 0),
    ('V25', 'V21', 0), ('V25', 'V28', 4),  ('V26', 'V9', 0),
    ('V26', 'V20', 0), ('V26', 'V30', 0),  ('V27', 'V29', 5),
    ('V28', 'V2', 5), ('V28', 'V19', 0),  ('V29', 'V19', 6),
    ('V29', 'V25', 8), ('V31', 'V18', 7)

]
for u, v, w in edges:
    G.add_edge(u, v, weight=w)

# Jalankan Dijkstra dari simpul V10
dist_v10, prev_v10, logs_v10 = dijkstra_manual_with_paths(G,
'V10')
```

```
# Mencari lintasan dari V10 ke V31 setelah eksekusi
path_v10_to_v31 = []
temp = 'V31'
while temp is not None:
    path_v10_to_v31.insert(0, temp)
    temp = prev_v10[temp]  # mengakses prev dari hasil
Dijkstra

# Menampilkan log iterasi
print("\n".join(logs_v10))

# Menampilkan lintasan terpendek dari V10 ke V31
print(f"\nLintasan terpendek dari V10 ke V31: {' →
'.join(path_v10_to_v31)}")
def total_rute(R):
  h=0
  for i in range(len(R)-1):
    h+=G[R[i]][R[i+1]]['weight']
  return h
total_rute(path_v10_to_v31)
```

Lampiran 4. Program Membuat Graf Acak Kedua

```
import networkx as nx
import random
import pickle
import matplotlib.pyplot as plt

# Jumlah simpul dan probabilitas sisi
n = 31
p = 0.1

# Gunakan seed agar hasilnya tetap sama setiap kali dijalankan
random.seed(80)
G = nx.erdos_renyi_graph(n, p, directed=True, seed=80)

# Menambahkan bobot acak pada setiap sisi (tetap sama setiap kali
dijalankan)
for u, v in G.edges():
    weight = random.randint(1, 8)  # Bobot antara 1 dan 8

    # Sekitar 10% sisi mendapatkan bobot negatif
    if random.random() < 0.1:
        weight *= -1

    G[u][v]['weight'] = weight

# Simpan graf menggunakan pickle agar tetap konsisten
with open("graf_G3.pkl", "wb") as f:
    pickle.dump(G, f)

# Menampilkan rincian sisi dan bobot di terminal dengan label V1, V2,
...
node_labels = {i: f"V{i+1}" for i in range(n)}

print("\nRincian sisi dan bobotnya:")
for u, v, data in G.edges(data=True):
  print(f"Sisi {node_labels[u]} → {node_labels[v]} dengan bobot
{data['weight']}")

# Menampilkan graf menggunakan Matplotlib
plt.figure(figsize=(30, 18))

# Menentukan posisi node (layout spring) dengan seed agar posisi tetap
sama
pos = nx.spring_layout(G, seed=1100)

# Menggambar simpul (nodes) dan sisi (edges)
nx.draw(G, pos, with_labels=True, labels=node_labels,
node_color='lightblue', node_size=3000, edge_color='gray', arrows=True)

# Menambahkan label bobot pada setiap sisi
edge_labels = {(u, v): G[u][v]['weight'] for u, v in G.edges()}
```

```
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_size=26)

plt.title("Graf Acak Berbobot (Erdos-Renyi Model)")
plt.show()

print("\nGraf telah disimpan sebagai 'graf_G3.pkl'.")
```

Lampiran 5. Program untuk Penerapan Algoritma Johnson Graf Acak Kedua

```
import networkx as nx
import pickle

# Fungsi Bellman-Ford untuk menghitung jarak terpendek dari
sumber
def bellman_ford(g, source):
    # Menghitung jarak terpendek dari sumber ke semua simpul
lainnya
    dist = {node: float('inf') for node in g.nodes}
    dist[source] = 0
    for _ in range(len(g.nodes) - 1):
        for u, v, weight in g.edges(data='weight'):
            if dist[u] + weight < dist[v]:
                dist[v] = dist[u] + weight
    # Deteksi siklus negatif
    for u, v, weight in g.edges(data='weight'):
        if dist[u] + weight < dist[v]:
            raise ValueError("Graf mengandung siklus
negatif")
    return dist

# Fungsi Dijkstra untuk mencari lintasan terpendek
def dijkstra(g, source):
    # Menghitung jarak terpendek dari simpul sumber ke semua
simpul lainnya
    return nx.single_source_dijkstra_path_length(g, source,
weight='weight')

# Fungsi untuk melakukan re-weighting pada graf
def reweight_graph(g, h):
    new_g = nx.DiGraph()
    for u, v, weight in g.edges(data='weight'):
        # Bobot baru dihitung dengan rumus
        new_weight = weight + h[u] - h[v]
        new_g.add_edge(u, v, weight=new_weight)
    return new_g

# Fungsi algoritma Johnson untuk mencari jarak terpendek
def johnson_algorithm(g):
    # Tambahkan node sumber tambahan 'S'
    g.add_node('S')
    for node in g.nodes:
        if node != 'S':
            g.add_edge('S', node, weight=0)

    # Hitung nilai potensi dengan Bellman-Ford
```

```
    try:
        h = bellman_ford(g, 'S')
    except ValueError as e:
        print(e)
        return None

    # Hapus node sumber tambahan 'S'
    g.remove_node('S')

    # Buat graf baru dengan bobot yang telah di-reweight
    reweighted_g = reweight_graph(g, h)

    # Tampilkan hasil re-weighting
    print(f"Nilai potensi (h): {h}")
    print("Nilai re-weighting untuk setiap edge:")
    for u, v, weight in reweighted_g.edges(data='weight'):
        print(f"{u} → {v} = {weight}")

    # Menerapkan Algoritma Dijkstra pada graf yang sudah di-
reweight
    shortest_paths = {}
    for node in g.nodes:
        # Menjalankan Dijkstra pada graf yang telah
dimodifikasi
        dijkstra_result = dijkstra(reweighted_g, node)

        # Menampilkan hasil jarak terpendek setelah Dijkstra
        shortest_paths[node] = dijkstra_result

    return shortest_paths

# Memuat graf yang telah disimpan
with open("graf_G3.pkl", "rb") as f:
    G_loaded = pickle.load(f)


# Jalankan algoritma Johnson untuk mendapatkan jarak
terpendek
result = johnson_algorithm(G_loaded)
if result:
    print("\nJarak terpendek dari setiap simpul setelah
Dijkstra (setelah re-weighting):")
    for node, paths in result.items():
        print(f"Dari {node}: {paths}")
```

Lampiran 6. Program untuk Menentukan Lintasan Terpendek Algoritma Dijkstra dari V_{15} ke V_{31} Pada Graf Acak Kedua

```
import networkx as nx
import heapq

# Modifikasi fungsi Dijkstra untuk menyertakan lintasan pada setiap
iterasi
def dijkstra_manual_with_paths(g, source):
    dist = {node: float('inf') for node in g.nodes}
    dist[source] = 0
    visited = set()
    pq = [(0, source)]

    prev = {node: None for node in g.nodes}  # untuk pelacakan lintasan
    logs = []
    logs.append(f">>> Mulai Dijkstra dari {source}")
    iteration = 1

    while pq:
        current_dist, current_node = heapq.heappop(pq)

        if current_node in visited:
            continue
        visited.add(current_node)

        logs.append(f"\n[Iterasi ke-{iteration}] Memproses node
{current_node} (jarak = {current_dist})")
        iteration += 1

        for neighbor in g.neighbors(current_node):
            weight = g[current_node][neighbor]['weight']
            distance = current_dist + weight

            if distance < dist[neighbor]:
                prev[neighbor] = current_node
                dist[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))

                # Rekonstruksi lintasan sementara
                path = []
                temp = neighbor
                while temp is not None:
                    path.insert(0, temp)
                    temp = prev[temp]
                logs.append(f" - Update: {neighbor} ← {current_node},
jarak baru = {distance}, lintasan = {' → '.join(path)}")
            else:
                logs.append(f" - Lewati: {neighbor} dengan jalur
{current_node} → {neighbor} ({distance}) (lebih besar dari
{dist[neighbor]})")

    return dist, prev, logs
```

```
# Buat graf
G = nx.DiGraph()
edges = [
    ('V1', 'V11', 6), ('V1', 'V15', 7), ('V1', 'V23', 0), ('V1', 'V26',
6), ('V1', 'V29', 8),
    ('V2', 'V5', 0), ('V2', 'V30', 0), ('V3', 'V5', 4), ('V3', 'V7',
7), ('V3', 'V8', 7),
    ('V3', 'V15', 6), ('V3', 'V17', 4), ('V3', 'V18', 8), ('V3', 'V21',
4), ('V3', 'V25', 9),
    ('V4', 'V17', 8), ('V5', 'V13', 0), ('V5', 'V25', 0),  ('V6', 'V4',
2), ('V6', 'V7', 4),
    ('V6', 'V31', 7), ('V7', 'V4', 9), ('V7', 'V26', 3), ('V7', 'V28',
9), ('V7', 'V29', 4),
    ('V7', 'V31', 10), ('V8', 'V23', 8), ('V9', 'V7', 3), ('V10',
'V12', 1), ('V10', 'V18', 3),
    ('V11', 'V13', 4), ('V11', 'V15', 2), ('V11', 'V25', 14),  ('V12',
'V4', 4), ('V12', 'V22', 1),
    ('V13', 'V4', 0),  ('V14', 'V6', 0), ('V14', 'V16', 9), ('V14',
'V17', 7), ('V14', 'V20', 3),
    ('V14', 'V23', 11), ('V15', 'V4', 8), ('V15', 'V6', 11), ('V15',
'V26', 0), ('V15', 'V29', 7),
    ('V16', 'V21', 3), ('V16', 'V29', 2), ('V17', 'V16', 0), ('V17',
'V31', 0), ('V18', 'V4', 3),
    ('V18', 'V7', 7), ('V18', 'V28', 5), ('V19', 'V12', 5), ('V19',
'V24', 5), ('V19', 'V27', 8),
    ('V19', 'V28', 5), ('V20', 'V11', 7), ('V20', 'V12', 7), ('V20',
'V17', 0), ('V20', 'V30', 6),
    ('V21', 'V13', 11),  ('V22', 'V5', 7), ('V22', 'V9', 4), ('V22',
'V16', 12), ('V23', 'V10', 3),
    ('V25', 'V2', 3),  ('V25', 'V8', 1), ('V25', 'V23', 6), ('V26',
'V5', 9), ('V26', 'V21', 2),
    ('V26', 'V28', 0), ('V26', 'V29', 0), ('V26', 'V30', 5), ('V27',
'V2', 0), ('V27', 'V3', 5),
    ('V27', 'V18', 7), ('V27', 'V20', 4), ('V28', 'V9', 0), ('V28',
'V20', 1), ('V28', 'V21', 2),
    ('V29', 'V11', 3), ('V29', 'V18', 7), ('V29', 'V21', 3), ('V29',
'V30', 5), ('V30', 'V6', 6),
    ('V30', 'V17', 8), ('V30', 'V20', 0),  ('V31', 'V18', 0), ('V31',
'V19', 4),
]
for u, v, w in edges:
    G.add_edge(u, v, weight=w)

# Jalankan Dijkstra dari simpul V15
dist_v15, prev_v15, logs_v15 = dijkstra_manual_with_paths(G, 'V15')

# Mencari lintasan dari V15 ke V31 setelah eksekusi
path_v15_to_v31 = []
temp = 'V31'
while temp is not None:
    path_v15_to_v31.insert(0, temp)
```

```
    temp = prev_v15[temp]  # mengakses prev dari hasil Dijkstra

# Menampilkan log iterasi
print("\n".join(logs_v15))

# Menampilkan lintasan terpendek dari V15 ke V31
print(f"\nLintasan terpendek dari V15 ke V31: {' →
'.join(path_v15_to_v31)}")
def total_rute(R):
  h=0
  for i in range(len(R)-1):
    h+=G[R[i]][R[i+1]]['weight']
  return h
total_rute(path_v15_to_v31)
```

Lampiran 7. Data Graf Acak Pertama

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{16}	2	V_{17}	V_2	-3
V_1	V_{26}	5	V_{17}	V_{21}	8
V_2	V_4	4	V_{17}	V_{23}	2
V_2	V_{19}	8	V_{17}	V_{30}	5
V_3	V_9	5	V_{18}	V_{16}	7
V_3	V_{24}	6	V_{18}	V_{26}	1
V_4	V_{30}	1	V_{19}	V_6	4
V_5	V_2	1	V_{19}	V_7	7
V_5	V_7	4	V_{19}	V_{17}	3
V_6	V_2	7	V_{19}	V_{22}	3
V_6	V_{17}	5	V_{19}	V_{31}	5
V_6	V_{25}	-7	V_{20}	V_3	7
V_6	V_{31}	4	V_{20}	V_{14}	3
V_7	V_8	3	V_{20}	V_{21}	5
V_7	V_{18}	4	V_{20}	V_{23}	4
V_8	V_{26}	-6	V_{20}	V_{25}	7
V_8	V_{27}	1	V_{20}	V_{26}	5
V_9	V_{18}	2	V_{21}	V_4	3
V_{10}	V_4	8	V_{21}	V_9	7
V_{10}	V_9	3	V_{21}	V_{23}	2
V_{10}	V_{20}	3	V_{22}	V_9	1
V_{10}	V_{22}	7	V_{22}	V_{15}	6
V_{11}	V_6	7	V_{23}	V_{20}	1
V_{11}	V_{10}	4	V_{23}	V_{22}	7
V_{11}	V_{15}	1	V_{23}	V_{24}	2
V_{12}	V_{14}	8	V_{24}	V_5	7
V_{12}	V_{15}	3	V_{24}	V_{14}	-3
V_{12}	V_{19}	5	V_{24}	V_{17}	4
V_{13}	V_{22}	5	V_{25}	V_9	7
V_{13}	V_{26}	5	V_{25}	V_{21}	7
V_{13}	V_{29}	5	V_{25}	V_{28}	5
V_{13}	V_{30}	6	V_{26}	V_9	6
V_{14}	V_1	4	V_{26}	V_{20}	5
V_{15}	V_{16}	3	V_{26}	V_{30}	-4
V_{15}	V_{21}	6	V_{27}	V_{29}	5
V_{15}	V_{26}	4	V_{28}	V_2	8
V_{15}	V_{28}	-6	V_{28}	V_{19}	4
V_{16}	V_4	2	V_{29}	V_{19}	4
V_{16}	V_{18}	4	V_{29}	V_{25}	1
V_{16}	V_{26}	5	V_{31}	V_{18}	7

Lampiran 8. Nilai Iterasi Pertama Algoritma Bellman-Ford Graf Acak Pertama

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{16}	0	V_{17}	V_2	-3
V_1	V_{26}	0	V_{17}	V_{21}	0
V_2	V_4	0	V_{17}	V_{23}	0
V_2	V_{19}	0	V_{17}	V_{30}	0
V_3	V_9	0	V_{18}	V_{16}	0
V_3	V_{24}	0	V_{18}	V_{26}	-6
V_4	V_{30}	0	V_{19}	V_6	0
V_5	V_2	0	V_{19}	V_7	0
V_5	V_7	0	V_{19}	V_{17}	0
V_6	V_2	0	V_{19}	V_{22}	0
V_6	V_{17}	0	V_{19}	V_{31}	0
V_6	V_{25}	-7	V_{20}	V_3	0
V_6	V_{31}	0	V_{20}	V_{14}	0
V_7	V_8	0	V_{20}	V_{21}	0
V_7	V_{18}	0	V_{20}	V_{23}	0
V_8	V_{26}	-6	V_{20}	V_{25}	-7
V_8	V_{27}	0	V_{20}	V_{26}	-6
V_9	V_{18}	0	V_{21}	V_4	0
V_{10}	V_4	0	V_{21}	V_9	0
V_{10}	V_9	0	V_{21}	V_{23}	0
V_{10}	V_{20}	0	V_{22}	V_9	0
V_{10}	V_{22}	0	V_{22}	V_{15}	0
V_{11}	V_6	0	V_{23}	V_{20}	0
V_{11}	V_{10}	0	V_{23}	V_{22}	0
V_{11}	V_{15}	0	V_{23}	V_{24}	0
V_{12}	V_{14}	0	V_{24}	V_5	0
V_{12}	V_{15}	0	V_{24}	V_{14}	-3
V_{12}	V_{19}	0	V_{24}	V_{17}	0
V_{13}	V_{22}	0	V_{25}	V_9	0
V_{13}	V_{26}	-6	V_{25}	V_{21}	0
V_{13}	V_{29}	0	V_{25}	V_{28}	-6
V_{13}	V_{30}	0	V_{26}	V_9	0
V_{14}	V_1	0	V_{26}	V_{20}	-1
V_{15}	V_{16}	0	V_{26}	V_{30}	-10
V_{15}	V_{21}	0	V_{27}	V_{29}	0
V_{15}	V_{26}	-6	V_{28}	V_2	-3
V_{15}	V_{28}	-6	V_{28}	V_{19}	-2
V_{16}	V_4	0	V_{29}	V_{19}	-2
V_{16}	V_{18}	0	V_{29}	V_{25}	-7
V_{16}	V_{26}	-6	V_{31}	V_{18}	0

Lampiran 9. Nilai Iterasi Kedua Algoritma Bellman-Ford Graf Acak Pertama

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{16}	0	V_{17}	V_2	-3
V_1	V_{26}	-6	V_{17}	V_{21}	0
V_2	V_4	0	V_{17}	V_{23}	0
V_2	V_{19}	-2	V_{17}	V_{30}	-10
V_3	V_9	0	V_{18}	V_{16}	0
V_3	V_{24}	0	V_{18}	V_{26}	-6
V_4	V_{30}	-10	V_{19}	V_6	0
V_5	V_2	-3	V_{19}	V_7	0
V_5	V_7	0	V_{19}	V_{17}	0
V_6	V_2	-3	V_{19}	V_{22}	0
V_6	V_{17}	0	V_{19}	V_{31}	0
V_6	V_{25}	-7	V_{20}	V_3	0
V_6	V_{31}	0	V_{20}	V_{14}	-3
V_7	V_8	0	V_{20}	V_{21}	0
V_7	V_{18}	0	V_{20}	V_{23}	0
V_8	V_{26}	-6	V_{20}	V_{25}	-7
V_8	V_{27}	0	V_{20}	V_{26}	-6
V_9	V_{18}	0	V_{21}	V_4	0
V_{10}	V_4	0	V_{21}	V_9	0
V_{10}	V_9	0	V_{21}	V_{23}	0
V_{10}	V_{20}	-1	V_{22}	V_9	0
V_{10}	V_{22}	0	V_{22}	V_{15}	0
V_{11}	V_6	0	V_{23}	V_{20}	-1
V_{11}	V_{10}	0	V_{23}	V_{22}	0
V_{11}	V_{15}	0	V_{23}	V_{24}	0
V_{12}	V_{14}	-3	V_{24}	V_5	0
V_{12}	V_{15}	0	V_{24}	V_{14}	-3
V_{12}	V_{19}	-2	V_{24}	V_{17}	0
V_{13}	V_{22}	0	V_{25}	V_9	0
V_{13}	V_{26}	-6	V_{25}	V_{21}	0
V_{13}	V_{29}	0	V_{25}	V_{28}	-6
V_{13}	V_{30}	-10	V_{26}	V_9	0
V_{14}	V_1	0	V_{26}	V_{20}	-1
V_{15}	V_{16}	0	V_{26}	V_{30}	-10
V_{15}	V_{21}	0	V_{27}	V_{29}	0
V_{15}	V_{26}	-6	V_{28}	V_2	-3
V_{15}	V_{28}	-6	V_{28}	V_{19}	-2
V_{16}	V_4	0	V_{29}	V_{19}	-2
V_{16}	V_{18}	0	V_{29}	V_{25}	-7
V_{16}	V_{26}	-6	V_{31}	V_{18}	0

Lampiran 10. Nilai Iterasi Ketiga Algoritma Bellman-Ford Graf Acak Pertama

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{16}	0	V_{17}	V_2	-3
V_1	V_{26}	-6	V_{17}	V_{21}	0
V_2	V_4	0	V_{17}	V_{23}	0
V_2	V_{19}	-2	V_{17}	V_{30}	-10
V_3	V_9	0	V_{18}	V_{16}	0
V_3	V_{24}	0	V_{18}	V_{26}	-6
V_4	V_{30}	-10	V_{19}	V_6	0
V_5	V_2	-3	V_{19}	V_7	0
V_5	V_7	0	V_{19}	V_{17}	0
V_6	V_2	-3	V_{19}	V_{22}	0
V_6	V_{17}	0	V_{19}	V_{31}	0
V_6	V_{25}	-7	V_{20}	V_3	0
V_6	V_{31}	0	V_{20}	V_{14}	-3
V_7	V_8	0	V_{20}	V_{21}	0
V_7	V_{18}	0	V_{20}	V_{23}	0
V_8	V_{26}	-6	V_{20}	V_{25}	-7
V_8	V_{27}	0	V_{20}	V_{26}	-6
V_9	V_{18}	0	V_{21}	V_4	0
V_{10}	V_4	0	V_{21}	V_9	0
V_{10}	V_9	0	V_{21}	V_{23}	0
V_{10}	V_{20}	-1	V_{22}	V_9	0
V_{10}	V_{22}	0	V_{22}	V_{15}	0
V_{11}	V_6	0	V_{23}	V_{20}	-1
V_{11}	V_{10}	0	V_{23}	V_{22}	0
V_{11}	V_{15}	0	V_{23}	V_{24}	0
V_{12}	V_{14}	-3	V_{24}	V_5	0
V_{12}	V_{15}	0	V_{24}	V_{14}	-3
V_{12}	V_{19}	-2	V_{24}	V_{17}	0
V_{13}	V_{22}	0	V_{25}	V_9	0
V_{13}	V_{26}	-6	V_{25}	V_{21}	0
V_{13}	V_{29}	0	V_{25}	V_{28}	-6
V_{13}	V_{30}	-10	V_{26}	V_9	0
V_{14}	V_1	0	V_{26}	V_{20}	-1
V_{15}	V_{16}	0	V_{26}	V_{30}	-10
V_{15}	V_{21}	0	V_{27}	V_{29}	0
V_{15}	V_{26}	-6	V_{28}	V_2	-3
V_{15}	V_{28}	-6	V_{28}	V_{19}	-2
V_{16}	V_4	0	V_{29}	V_{19}	-2
V_{16}	V_{18}	0	V_{29}	V_{25}	-7
V_{16}	V_{26}	-6	V_{31}	V_{18}	0

Lampiran 11. Nilai Potensial $h(v)$ Graf Acak Pertama

Nilai Potensial $h(v)$	Bobot	Nilai Potensial $h(v)$	Bobot
$h(1)$	0	$h(17)$	0
$h(2)$	-3	$h(18)$	0
$h(3)$	0	$h(19)$	-2
$h(4)$	0	$h(20)$	-1
$h(5)$	0	$h(21)$	0
$h(6)$	0	$h(22)$	0
$h(7)$	0	$h(23)$	0
$h(8)$	0	$h(24)$	0
$h(9)$	0	$h(25)$	-7
$h(10)$	0	$h(26)$	-6
$h(11)$	0	$h(27)$	0
$h(12)$	0	$h(28)$	-6
$h(13)$	0	$h(29)$	0
$h(14)$	-3	$h(30)$	-10
$h(15)$	0	$h(31)$	0
$h(16)$	0		

Lampiran 12. Hasil Pembobotan Ulang Graf Acak Pertama

Simpul Asal	Simpul Tujuan	Bobot Baru	Simpul Asal	Simpul Tujuan	Bobot Baru
V_1	V_{16}	2	V_{17}	V_2	0
V_1	V_{26}	11	V_{17}	V_{21}	8
V_2	V_4	1	V_{17}	V_{23}	2
V_2	V_{19}	7	V_{17}	V_{30}	15
V_3	V_9	5	V_{18}	V_{16}	7
V_3	V_{24}	6	V_{18}	V_{26}	7
V_4	V_{30}	11	V_{19}	V_6	2
V_5	V_2	4	V_{19}	V_7	5
V_5	V_7	4	V_{19}	V_{17}	1
V_6	V_2	10	V_{19}	V_{22}	1
V_6	V_{17}	5	V_{19}	V_{31}	3
V_6	V_{25}	0	V_{20}	V_3	6
V_6	V_{31}	4	V_{20}	V_{14}	5
V_7	V_8	3	V_{20}	V_{21}	4
V_7	V_{18}	4	V_{20}	V_{23}	3
V_8	V_{26}	0	V_{20}	V_{25}	13
V_8	V_{27}	1	V_{20}	V_{26}	10
V_9	V_{18}	2	V_{21}	V_4	3
V_{10}	V_4	8	V_{21}	V_9	7
V_{10}	V_9	3	V_{21}	V_{23}	2
V_{10}	V_{20}	4	V_{22}	V_9	1
V_{10}	V_{22}	7	V_{22}	V_{15}	6
V_{11}	V_6	7	V_{23}	V_{20}	2
V_{11}	V_{10}	4	V_{23}	V_{22}	7
V_{11}	V_{15}	1	V_{23}	V_{24}	2
V_{12}	V_{14}	11	V_{24}	V_5	7
V_{12}	V_{15}	3	V_{24}	V_{14}	0
V_{12}	V_{19}	7	V_{24}	V_{17}	4
V_{13}	V_{22}	5	V_{25}	V_9	0
V_{13}	V_{26}	11	V_{25}	V_{21}	0
V_{13}	V_{29}	5	V_{25}	V_{28}	4
V_{13}	V_{30}	16	V_{26}	V_9	0
V_{14}	V_1	1	V_{26}	V_{20}	0
V_{15}	V_{16}	3	V_{26}	V_{30}	0
V_{15}	V_{21}	6	V_{27}	V_{29}	5
V_{15}	V_{26}	10	V_{28}	V_2	5
V_{15}	V_{28}	0	V_{28}	V_{19}	0
V_{16}	V_4	2	V_{29}	V_{19}	6
V_{16}	V_{18}	4	V_{29}	V_{25}	8
V_{16}	V_{26}	11	V_{31}	V_{18}	7

Lampiran 13. Inisialisasi Algoritma Dijkstra Graf Acak Pertama

Simpul	Jarak Awal	Status
V_1	0	Sudah dikunjungi (simpul sumber)
V_2	∞	Belum dikunjungi
V_3	∞	Belum dikunjungi
V_4	∞	Belum dikunjungi
V_5	∞	Belum dikunjungi
V_6	∞	Belum dikunjungi
V_7	∞	Belum dikunjungi
V_8	∞	Belum dikunjungi
V_9	∞	Belum dikunjungi
V_{10}	∞	Belum dikunjungi
V_{11}	∞	Belum dikunjungi
V_{12}	∞	Belum dikunjungi
V_{13}	∞	Belum dikunjungi
V_{14}	∞	Belum dikunjungi
V_{15}	∞	Belum dikunjungi
V_{16}	∞	Belum dikunjungi
V_{17}	∞	Belum dikunjungi
V_{18}	∞	Belum dikunjungi
V_{19}	∞	Belum dikunjungi
V_{20}	∞	Belum dikunjungi
V_{21}	∞	Belum dikunjungi
V_{22}	∞	Belum dikunjungi
V_{23}	∞	Belum dikunjungi
V_{24}	∞	Belum dikunjungi
V_{25}	∞	Belum dikunjungi
V_{26}	∞	Belum dikunjungi
V_{27}	∞	Belum dikunjungi
V_{28}	∞	Belum dikunjungi
V_{29}	∞	Belum dikunjungi
V_{30}	∞	Belum dikunjungi
V_{31}	∞	Belum dikunjungi

Lampiran 14. Proses Relaksasi Algoritma Dijkstra Graf Acak Pertama

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
1	V_{10}	V_{20}	∞	4	$V_{10} \rightarrow V_{20}$	Diperbarui
1	V_{10}	V_{22}	∞	7	$V_{10} \rightarrow V_{22}$	Diperbarui
1	V_{10}	V_4	∞	8	$V_{10} \rightarrow V_4$	Diperbarui
1	V_{10}	V_9	∞	3	$V_{10} \rightarrow V_9$	Diperbarui
2	V_9	V_{18}	∞	5	$V_{10} \rightarrow V_9 \rightarrow V_{18}$	Diperbarui
3	V_{20}	V_{14}	∞	9	$V_{10} \rightarrow V_{20} \rightarrow V_{14}$	Diperbarui
3	V_{20}	V_{21}	∞	8	$V_{10} \rightarrow V_{20} \rightarrow V_{21}$	Diperbarui
3	V_{20}	V_{23}	∞	7	$V_{10} \rightarrow V_{20} \rightarrow V_{23}$	Diperbarui
3	V_{20}	V_{25}	∞	17	$V_{10} \rightarrow V_{20} \rightarrow V_{25}$	Diperbarui
3	V_{20}	V_{26}	∞	14	$V_{10} \rightarrow V_{20} \rightarrow V_{26}$	Diperbarui
3	V_{20}	V_3	∞	10	$V_{10} \rightarrow V_{20} \rightarrow V_3$	Diperbarui
4	V_{18}	V_{16}	∞	12	$V_{10} \rightarrow V_9 \rightarrow V_{18} \rightarrow V_{16}$	Diperbarui
4	V_{18}	V_{26}	14	12	$V_{10} \rightarrow V_9 \rightarrow V_{18} \rightarrow V_{26}$	Diperbarui
5	V_{22}	V_{15}	∞	13	$V_{10} \rightarrow V_{22} \rightarrow V_{15}$	Diperbarui
5	V_{22}	V_9	3	8	–	Tidak Diperbarui
6	V_{23}	V_{20}	4	9	–	Tidak Diperbarui
6	V_{23}	V_{22}	7	14	–	Tidak Diperbarui
6	V_{23}	V_{24}	∞	9	$V_{10} \rightarrow V_{20} \rightarrow V_{23} \rightarrow V_{24}$	Diperbarui
7	V_{21}	V_{23}	7	10	–	Tidak Diperbarui
7	V_{21}	V_4	8	11	–	Tidak Diperbarui
7	V_{21}	V_9	3	15	–	Tidak Diperbarui
8	V_4	V_{30}	∞	19	$V_{10} \rightarrow V_4 \rightarrow V_{30}$	Diperbarui

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
9	V_{14}	V_1	∞	10	$V_{10} \rightarrow V_{20} \rightarrow V_{14} \rightarrow V_1$	Diperbarui
10	V_{24}	V_{14}	9	9	–	Tidak Diperbarui
10	V_{24}	V_{17}	∞	13	$V_{10} \rightarrow V_{20} \rightarrow V_{23} \rightarrow V_{24} \rightarrow V_{17}$	Diperbarui
10	V_{24}	V_5	∞	16	$V_{10} \rightarrow V_{20} \rightarrow V_{23} \rightarrow V_{24} \rightarrow V_5$	Diperbarui
11	V_1	V_{16}	12	12	–	Tidak Diperbarui
11	V_1	V_{26}	12	21	–	Tidak Diperbarui
12	V_3	V_{24}	9	16	–	Tidak Diperbarui
12	V_3	V_9	3	15	–	Tidak Diperbarui
13	V_{16}	V_{18}	5	16	–	Tidak Diperbarui
13	V_{16}	V_{26}	12	23	–	Tidak Diperbarui
13	V_{16}	V_4	8	14	–	Tidak Diperbarui
14	V_{26}	V_{20}	4	12	–	Tidak Diperbarui
14	V_{26}	V_{30}	∞	12	$V_{10} \rightarrow V_9 \rightarrow V_{18} \rightarrow V_{26} \rightarrow V_{30}$	Diperbarui
14	V_{26}	V_9	3	12	–	Tidak Diperbarui
16	V_{15}	V_{16}	12	16	–	Tidak Diperbarui
16	V_{15}	V_{21}	8	19	–	Tidak Diperbarui
16	V_{15}	V_{26}	12	23	–	Tidak Diperbarui
16	V_{15}	V_{28}	∞	13	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28}$	Diperbarui

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
17	V_{17}	V_2	∞	13	$V_{10} \rightarrow V_{20} \rightarrow V_{23} \rightarrow V_{24} \rightarrow V_{17} \rightarrow V_2$	Diperbarui
17	V_{17}	V_{21}	8	21	–	Tidak Diperbarui
17	V_{17}	V_{23}	7	15	–	Tidak Diperbarui
17	V_{17}	V_{30}	12	28	–	Tidak Diperbarui
18	V_2	V_{19}	∞	20	$V_{10} \rightarrow V_{20} \rightarrow V_{23} \rightarrow V_{24} \rightarrow V_{17} \rightarrow V_2 \rightarrow V_{19}$	Diperbarui
18	V_2	V_4	8	14	–	Tidak Diperbarui
19	V_{28}	V_{19}	∞	13	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19}$	Diperbarui
19	V_{28}	V_2	13	18	–	Tidak Diperbarui
20	V_{19}	V_{17}	13	14	–	Tidak Diperbarui
20	V_{19}	V_{22}	7	14	–	Tidak Diperbarui
20	V_{19}	V_{31}	∞	16	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_{31}$	Diperbarui
20	V_{19}	V_6	∞	15	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_6$	Diperbarui
20	V_{19}	V_7	∞	18	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_7$	Diperbarui

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
21	V_6	V_{17}	13	20	–	Tidak Diperbarui
21	V_6	V_2	13	25	–	Tidak Diperbarui
21	V_6	V_{25}	∞	15	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_6 \rightarrow V_{25}$	Diperbarui
21	V_6	V_{31}	16	19	–	Tidak Diperbarui
22	V_{25}	V_{21}	8	15	–	Tidak Diperbarui
22	V_{25}	V_{28}	13	19	–	Tidak Diperbarui
22	V_{25}	V_9	3	15	–	Tidak Diperbarui
23	V_{31}	V_{18}	5	23	–	Tidak Diperbarui
24	V_5	V_2	13	20	–	Tidak Diperbarui
24	V_5	V_7	18	20	–	Tidak Diperbarui
25	V_7	V_{18}	5	22	–	Tidak Diperbarui
25	V_7	V_8	∞	21	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_7 \rightarrow V_8$	Diperbarui
26	V_8	V_{26}	12	21	–	Tidak Diperbarui
26	V_8	V_{27}	∞	22	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_7 \rightarrow V_8 \rightarrow V_{27}$	Diperbarui

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
27	V_{27}	V_{29}	∞	27	$V_{10} \rightarrow V_{22} \rightarrow V_{15} \rightarrow V_{28} \rightarrow V_{19} \rightarrow V_{7} \rightarrow V_{8} \rightarrow V_{27} \rightarrow V_{29}$	Diperbarui
28	V_{29}	V_{19}	13	33	–	Tidak Diperbarui
28	V_{29}	V_{25}	15	35	–	Tidak Diperbarui

Lampiran 15. Data Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	6	V_{16}	V_{21}	7
V_1	V_{15}	7	V_{16}	V_{29}	6
V_1	V_{23}	-5	V_{17}	V_{16}	1
V_1	V_{26}	4	V_{17}	V_{31}	1
V_1	V_{29}	8	V_{18}	V_4	2
V_2	V_5	2	V_{18}	V_7	8
V_2	V_{30}	5	V_{18}	V_{28}	5
V_3	V_5	-1	V_{19}	V_{12}	5
V_3	V_7	7	V_{19}	V_{24}	5
V_3	V_8	7	V_{19}	V_{27}	8
V_3	V_{15}	6	V_{19}	V_{28}	4
V_3	V_{17}	-1	V_{20}	V_{11}	8
V_3	V_{18}	7	V_{20}	V_{12}	8
V_3	V_{21}	4	V_{20}	V_{17}	-4
V_3	V_{25}	2	V_{20}	V_{30}	5
V_4	V_{17}	5	V_{21}	V_{13}	8
V_5	V_{13}	2	V_{22}	V_5	2
V_5	V_{25}	-2	V_{22}	V_9	4
V_6	V_4	4	V_{22}	V_{16}	8
V_6	V_7	8	V_{23}	V_{10}	8
V_6	V_{31}	7	V_{25}	V_2	3
V_7	V_4	7	V_{25}	V_8	8
V_7	V_{26}	1	V_{25}	V_{23}	8
V_7	V_{28}	8	V_{26}	V_5	6
V_7	V_{29}	4	V_{26}	V_{21}	4
V_7	V_{31}	6	V_{26}	V_{28}	1
V_8	V_{23}	3	V_{26}	V_{29}	2
V_9	V_7	3	V_{26}	V_{30}	5
V_{10}	V_{12}	1	V_{27}	V_2	-7
V_{10}	V_{18}	2	V_{27}	V_3	5
V_{11}	V_{13}	1	V_{27}	V_{18}	6
V_{11}	V_{15}	2	V_{27}	V_{20}	3
V_{11}	V_{25}	7	V_{28}	V_9	1
V_{12}	V_4	2	V_{28}	V_{20}	1
V_{12}	V_{22}	1	V_{28}	V_{21}	3
V_{13}	V_4	1	V_{29}	V_{11}	3
V_{14}	V_6	-4	V_{29}	V_{18}	6
V_{14}	V_{16}	5	V_{29}	V_{21}	3
V_{14}	V_{17}	2	V_{29}	V_{30}	3
V_{14}	V_{20}	2	V_{30}	V_6	4
V_{14}	V_{23}	6	V_{30}	V_{17}	5
V_{15}	V_4	6	V_{30}	V_{20}	1
V_{15}	V_6	7	V_{31}	V_{18}	3

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	8
V_{15}	V_{29}	7			

Lampiran 16. Nilai Iterasi Pertama Algoritma Bellman-Ford Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	0	V_{16}	V_{21}	0
V_1	V_{15}	0	V_{16}	V_{29}	0
V_1	V_{23}	-5	V_{17}	V_{16}	0
V_1	V_{26}	0	V_{17}	V_{31}	0
V_1	V_{29}	0	V_{18}	V_4	0
V_2	V_5	0	V_{18}	V_7	0
V_2	V_{30}	0	V_{18}	V_{28}	0
V_3	V_5	-1	V_{19}	V_{12}	0
V_3	V_7	0	V_{19}	V_{24}	0
V_3	V_8	0	V_{19}	V_{27}	0
V_3	V_{15}	0	V_{19}	V_{28}	0
V_3	V_{17}	-1	V_{20}	V_{11}	0
V_3	V_{18}	0	V_{20}	V_{12}	0
V_3	V_{21}	0	V_{20}	V_{17}	-4
V_3	V_{25}	0	V_{20}	V_{30}	0
V_4	V_{17}	-1	V_{21}	V_{13}	0
V_5	V_{13}	0	V_{22}	V_5	-1
V_5	V_{25}	-3	V_{22}	V_9	0
V_6	V_4	0	V_{22}	V_{16}	0
V_6	V_7	0	V_{23}	V_{10}	0
V_6	V_{31}	0	V_{25}	V_2	0
V_7	V_4	0	V_{25}	V_8	0
V_7	V_{26}	0	V_{25}	V_{23}	-5
V_7	V_{28}	0	V_{26}	V_5	-1
V_7	V_{29}	0	V_{26}	V_{21}	0
V_7	V_{31}	0	V_{26}	V_{28}	-1
V_8	V_{23}	-5	V_{26}	V_{29}	0
V_9	V_7	0	V_{26}	V_{30}	0
V_{10}	V_{12}	0	V_{27}	V_2	-7
V_{10}	V_{18}	0	V_{27}	V_3	0
V_{11}	V_{13}	0	V_{27}	V_{18}	0
V_{11}	V_{15}	0	V_{27}	V_{20}	0
V_{11}	V_{25}	-3	V_{28}	V_9	0
V_{12}	V_4	0	V_{28}	V_{20}	0
V_{12}	V_{22}	0	V_{28}	V_{21}	0
V_{13}	V_4	0	V_{29}	V_{11}	0
V_{14}	V_6	-4	V_{29}	V_{18}	0
V_{14}	V_{16}	0	V_{29}	V_{21}	0
V_{14}	V_{17}	-1	V_{29}	V_{30}	0
V_{14}	V_{20}	0	V_{30}	V_6	-4
V_{14}	V_{23}	-5	V_{30}	V_{17}	-4
V_{15}	V_4	0	V_{30}	V_{20}	0
V_{15}	V_6	-4	V_{31}	V_{18}	0

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	0
V_{15}	V_{29}	0			

Lampiran 17. Nilai Iterasi Kedua Algoritma Bellman-Ford Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	0	V_{16}	V_{21}	0
V_1	V_{15}	0	V_{16}	V_{29}	0
V_1	V_{23}	-5	V_{17}	V_{16}	-3
V_1	V_{26}	-2	V_{17}	V_{31}	-3
V_1	V_{29}	0	V_{18}	V_4	-2
V_2	V_5	-5	V_{18}	V_7	0
V_2	V_{30}	-2	V_{18}	V_{28}	-1
V_3	V_5	-5	V_{19}	V_{12}	0
V_3	V_7	0	V_{19}	V_{24}	0
V_3	V_8	0	V_{19}	V_{27}	0
V_3	V_{15}	0	V_{19}	V_{28}	-1
V_3	V_{17}	-4	V_{20}	V_{11}	0
V_3	V_{18}	0	V_{20}	V_{12}	0
V_3	V_{21}	0	V_{20}	V_{17}	-4
V_3	V_{25}	-3	V_{20}	V_{30}	-2
V_4	V_{17}	-4	V_{21}	V_{13}	-3
V_5	V_{13}	-3	V_{22}	V_5	-5
V_5	V_{25}	-7	V_{22}	V_9	0
V_6	V_4	0	V_{22}	V_{16}	-3
V_6	V_7	0	V_{23}	V_{10}	0
V_6	V_{31}	0	V_{25}	V_2	-7
V_7	V_4	0	V_{25}	V_8	0
V_7	V_{26}	-2	V_{25}	V_{23}	-5
V_7	V_{28}	-1	V_{26}	V_5	-5
V_7	V_{29}	0	V_{26}	V_{21}	0
V_7	V_{31}	0	V_{26}	V_{28}	-1
V_8	V_{23}	-5	V_{26}	V_{29}	0
V_9	V_7	0	V_{26}	V_{30}	-2
V_{10}	V_{12}	0	V_{27}	V_2	-7
V_{10}	V_{18}	0	V_{27}	V_3	0
V_{11}	V_{13}	-3	V_{27}	V_{18}	0
V_{11}	V_{15}	0	V_{27}	V_{20}	0
V_{11}	V_{25}	-7	V_{28}	V_9	0
V_{12}	V_4	0	V_{28}	V_{20}	0
V_{12}	V_{22}	0	V_{28}	V_{21}	0
V_{13}	V_4	-2	V_{29}	V_{11}	0
V_{14}	V_6	-4	V_{29}	V_{18}	0
V_{14}	V_{16}	0	V_{29}	V_{21}	0
V_{14}	V_{17}	-4	V_{29}	V_{30}	-2
V_{14}	V_{20}	0	V_{30}	V_6	-4
V_{14}	V_{23}	-5	V_{30}	V_{17}	-4
V_{15}	V_4	-2	V_{30}	V_{20}	-1
V_{15}	V_6	-4	V_{31}	V_{18}	0

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	0
V_{15}	V_{29}	0			

Lampiran 18. Nilai Iterasi Ketiga Algoritma Bellman-Ford Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	0	V_{16}	V_{21}	0
V_1	V_{15}	0	V_{16}	V_{29}	0
V_1	V_{23}	-5	V_{17}	V_{16}	-3
V_1	V_{26}	-2	V_{17}	V_{31}	-3
V_1	V_{29}	0	V_{18}	V_4	-2
V_2	V_5	-5	V_{18}	V_7	0
V_2	V_{30}	-2	V_{18}	V_{28}	-1
V_3	V_5	-5	V_{19}	V_{12}	0
V_3	V_7	0	V_{19}	V_{24}	0
V_3	V_8	0	V_{19}	V_{27}	0
V_3	V_{15}	0	V_{19}	V_{28}	-1
V_3	V_{17}	-4	V_{20}	V_{11}	0
V_3	V_{18}	0	V_{20}	V_{12}	0
V_3	V_{21}	0	V_{20}	V_{17}	-5
V_3	V_{25}	-7	V_{20}	V_{30}	-2
V_4	V_{17}	-4	V_{21}	V_{13}	-3
V_5	V_{13}	-3	V_{22}	V_5	-5
V_5	V_{25}	-7	V_{22}	V_9	0
V_6	V_4	-2	V_{22}	V_{16}	-3
V_6	V_7	0	V_{23}	V_{10}	0
V_6	V_{31}	-3	V_{25}	V_2	-7
V_7	V_4	-2	V_{25}	V_8	0
V_7	V_{26}	-2	V_{25}	V_{23}	-5
V_7	V_{28}	-1	V_{26}	V_5	-5
V_7	V_{29}	0	V_{26}	V_{21}	0
V_7	V_{31}	-3	V_{26}	V_{28}	-1
V_8	V_{23}	-5	V_{26}	V_{29}	0
V_9	V_7	0	V_{26}	V_{30}	-2
V_{10}	V_{12}	0	V_{27}	V_2	-7
V_{10}	V_{18}	0	V_{27}	V_3	0
V_{11}	V_{13}	-3	V_{27}	V_{18}	0
V_{11}	V_{15}	0	V_{27}	V_{20}	-1
V_{11}	V_{25}	-7	V_{28}	V_9	0
V_{12}	V_4	-2	V_{28}	V_{20}	-1
V_{12}	V_{22}	0	V_{28}	V_{21}	0
V_{13}	V_4	-2	V_{29}	V_{11}	0
V_{14}	V_6	-4	V_{29}	V_{18}	0
V_{14}	V_{16}	-3	V_{29}	V_{21}	0
V_{14}	V_{17}	-4	V_{29}	V_{30}	-2
V_{14}	V_{20}	-1	V_{30}	V_6	-4
V_{14}	V_{23}	-5	V_{30}	V_{17}	-5
V_{15}	V_4	-2	V_{30}	V_{20}	-1
V_{15}	V_6	-4	V_{31}	V_{18}	0

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	0
V_{15}	V_{29}	0			

Lampiran 19. Nilai Iterasi Keempat Algoritma Bellman-Ford Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	0	V_{16}	V_{21}	0
V_1	V_{15}	0	V_{16}	V_{29}	0
V_1	V_{23}	-5	V_{17}	V_{16}	-4
V_1	V_{26}	-2	V_{17}	V_{31}	-4
V_1	V_{29}	0	V_{18}	V_4	-2
V_2	V_5	-5	V_{18}	V_7	0
V_2	V_{30}	-2	V_{18}	V_{28}	-1
V_3	V_5	-5	V_{19}	V_{12}	0
V_3	V_7	0	V_{19}	V_{24}	0
V_3	V_8	0	V_{19}	V_{27}	0
V_3	V_{15}	0	V_{19}	V_{28}	-1
V_3	V_{17}	-5	V_{20}	V_{11}	0
V_3	V_{18}	0	V_{20}	V_{12}	0
V_3	V_{21}	0	V_{20}	V_{17}	-5
V_3	V_{25}	-7	V_{20}	V_{30}	-2
V_4	V_{17}	-5	V_{21}	V_{13}	-3
V_5	V_{13}	-3	V_{22}	V_5	-5
V_5	V_{25}	-7	V_{22}	V_9	0
V_6	V_4	-2	V_{22}	V_{16}	-4
V_6	V_7	0	V_{23}	V_{10}	0
V_6	V_{31}	-3	V_{25}	V_2	-7
V_7	V_4	-2	V_{25}	V_8	0
V_7	V_{26}	-2	V_{25}	V_{23}	-5
V_7	V_{28}	-1	V_{26}	V_5	-5
V_7	V_{29}	0	V_{26}	V_{21}	0
V_7	V_{31}	-3	V_{26}	V_{28}	-1
V_8	V_{23}	-5	V_{26}	V_{29}	0
V_9	V_7	0	V_{26}	V_{30}	-2
V_{10}	V_{12}	0	V_{27}	V_2	-7
V_{10}	V_{18}	0	V_{27}	V_3	0
V_{11}	V_{13}	-3	V_{27}	V_{18}	0
V_{11}	V_{15}	0	V_{27}	V_{20}	-1
V_{11}	V_{25}	-7	V_{28}	V_9	0
V_{12}	V_4	-2	V_{28}	V_{20}	-1
V_{12}	V_{22}	0	V_{28}	V_{21}	0
V_{13}	V_4	-2	V_{29}	V_{11}	0
V_{14}	V_6	-4	V_{29}	V_{18}	0
V_{14}	V_{16}	-3	V_{29}	V_{21}	0
V_{14}	V_{17}	-5	V_{29}	V_{30}	-2
V_{14}	V_{20}	-1	V_{30}	V_6	-4
V_{14}	V_{23}	-5	V_{30}	V_{17}	-5
V_{15}	V_4	-2	V_{30}	V_{20}	-1
V_{15}	V_6	-4	V_{31}	V_{18}	-1

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	0
V_{15}	V_{29}	0			

Lampiran 20. Nilai Iterasi Kelima Algoritma Bellman-Ford Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	0	V_{16}	V_{21}	0
V_1	V_{15}	0	V_{16}	V_{29}	0
V_1	V_{23}	-5	V_{17}	V_{16}	-4
V_1	V_{26}	-2	V_{17}	V_{31}	-4
V_1	V_{29}	0	V_{18}	V_4	-2
V_2	V_5	-5	V_{18}	V_7	0
V_2	V_{30}	-2	V_{18}	V_{28}	-1
V_3	V_5	-5	V_{19}	V_{12}	0
V_3	V_7	0	V_{19}	V_{24}	0
V_3	V_8	0	V_{19}	V_{27}	0
V_3	V_{15}	0	V_{19}	V_{28}	-1
V_3	V_{17}	-5	V_{20}	V_{11}	0
V_3	V_{18}	-1	V_{20}	V_{12}	0
V_3	V_{21}	0	V_{20}	V_{17}	-5
V_3	V_{25}	-7	V_{20}	V_{30}	-2
V_4	V_{17}	-5	V_{21}	V_{13}	-3
V_5	V_{13}	-3	V_{22}	V_5	-5
V_5	V_{25}	-7	V_{22}	V_9	0
V_6	V_4	-2	V_{22}	V_{16}	-4
V_6	V_7	0	V_{23}	V_{10}	0
V_6	V_{31}	-4	V_{25}	V_2	-7
V_7	V_4	-2	V_{25}	V_8	0
V_7	V_{26}	-2	V_{25}	V_{23}	-5
V_7	V_{28}	-1	V_{26}	V_5	-5
V_7	V_{29}	0	V_{26}	V_{21}	0
V_7	V_{31}	-4	V_{26}	V_{28}	-1
V_8	V_{23}	-5	V_{26}	V_{29}	0
V_9	V_7	0	V_{26}	V_{30}	-2
V_{10}	V_{12}	0	V_{27}	V_2	-7
V_{10}	V_{18}	-1	V_{27}	V_3	0
V_{11}	V_{13}	-3	V_{27}	V_{18}	-1
V_{11}	V_{15}	0	V_{27}	V_{20}	-1
V_{11}	V_{25}	-7	V_{28}	V_9	0
V_{12}	V_4	-2	V_{28}	V_{20}	-1
V_{12}	V_{22}	0	V_{28}	V_{21}	0
V_{13}	V_4	-2	V_{29}	V_{11}	0
V_{14}	V_6	-4	V_{29}	V_{18}	-1
V_{14}	V_{16}	-4	V_{29}	V_{21}	0
V_{14}	V_{17}	-5	V_{29}	V_{30}	-2
V_{14}	V_{20}	-1	V_{30}	V_6	-4
V_{14}	V_{23}	-5	V_{30}	V_{17}	-5
V_{15}	V_4	-2	V_{30}	V_{20}	-1
V_{15}	V_6	-4	V_{31}	V_{18}	-1

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	0
V_{15}	V_{29}	0			

Lampiran 21. Nilai Iterasi Keenam Algoritma Bellman-Ford Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	0	V_{16}	V_{21}	0
V_1	V_{15}	0	V_{16}	V_{29}	0
V_1	V_{23}	-5	V_{17}	V_{16}	-4
V_1	V_{26}	-2	V_{17}	V_{31}	-4
V_1	V_{29}	0	V_{18}	V_4	-2
V_2	V_5	-5	V_{18}	V_7	0
V_2	V_{30}	-2	V_{18}	V_{28}	-1
V_3	V_5	-5	V_{19}	V_{12}	0
V_3	V_7	0	V_{19}	V_{24}	0
V_3	V_8	0	V_{19}	V_{27}	0
V_3	V_{15}	0	V_{19}	V_{28}	-1
V_3	V_{17}	-5	V_{20}	V_{11}	0
V_3	V_{18}	-1	V_{20}	V_{12}	0
V_3	V_{21}	0	V_{20}	V_{17}	-5
V_3	V_{25}	-7	V_{20}	V_{30}	-2
V_4	V_{17}	-5	V_{21}	V_{13}	-3
V_5	V_{13}	-3	V_{22}	V_5	-5
V_5	V_{25}	-7	V_{22}	V_9	0
V_6	V_4	-2	V_{22}	V_{16}	-4
V_6	V_7	0	V_{23}	V_{10}	0
V_6	V_{31}	-4	V_{25}	V_2	-7
V_7	V_4	-2	V_{25}	V_8	0
V_7	V_{26}	-2	V_{25}	V_{23}	-5
V_7	V_{28}	-1	V_{26}	V_5	-5
V_7	V_{29}	0	V_{26}	V_{21}	0
V_7	V_{31}	-4	V_{26}	V_{28}	-1
V_8	V_{23}	-5	V_{26}	V_{29}	0
V_9	V_7	0	V_{26}	V_{30}	-2
V_{10}	V_{12}	0	V_{27}	V_2	-7
V_{10}	V_{18}	-1	V_{27}	V_3	0
V_{11}	V_{13}	-3	V_{27}	V_{18}	-1
V_{11}	V_{15}	0	V_{27}	V_{20}	-1
V_{11}	V_{25}	-7	V_{28}	V_9	0
V_{12}	V_4	-2	V_{28}	V_{20}	-1
V_{12}	V_{22}	0	V_{28}	V_{21}	0
V_{13}	V_4	-2	V_{29}	V_{11}	0
V_{14}	V_6	-4	V_{29}	V_{18}	-1
V_{14}	V_{16}	-4	V_{29}	V_{21}	0
V_{14}	V_{17}	-5	V_{29}	V_{30}	-2
V_{14}	V_{20}	-1	V_{30}	V_6	-4
V_{14}	V_{23}	-5	V_{30}	V_{17}	-5
V_{15}	V_4	-2	V_{30}	V_{20}	-1
V_{15}	V_6	-4	V_{31}	V_{18}	-1

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	0
V_{15}	V_{29}	0			

Lampiran 22. Nilai Iterasi Ketujuh Algoritma Bellman-Ford Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	0	V_{16}	V_{21}	0
V_1	V_{15}	0	V_{16}	V_{29}	0
V_1	V_{23}	-5	V_{17}	V_{16}	-4
V_1	V_{26}	-2	V_{17}	V_{31}	-4
V_1	V_{29}	0	V_{18}	V_4	-2
V_2	V_5	-5	V_{18}	V_7	0
V_2	V_{30}	-2	V_{18}	V_{28}	-1
V_3	V_5	-5	V_{19}	V_{12}	0
V_3	V_7	0	V_{19}	V_{24}	0
V_3	V_8	0	V_{19}	V_{27}	0
V_3	V_{15}	0	V_{19}	V_{28}	-1
V_3	V_{17}	-5	V_{20}	V_{11}	0
V_3	V_{18}	-1	V_{20}	V_{12}	0
V_3	V_{21}	0	V_{20}	V_{17}	-5
V_3	V_{25}	-7	V_{20}	V_{30}	-2
V_4	V_{17}	-5	V_{21}	V_{13}	-3
V_5	V_{13}	-3	V_{22}	V_5	-5
V_5	V_{25}	-7	V_{22}	V_9	0
V_6	V_4	-2	V_{22}	V_{16}	-4
V_6	V_7	0	V_{23}	V_{10}	0
V_6	V_{31}	-4	V_{25}	V_2	-7
V_7	V_4	-2	V_{25}	V_8	0
V_7	V_{26}	-2	V_{25}	V_{23}	-5
V_7	V_{28}	-1	V_{26}	V_5	-5
V_7	V_{29}	0	V_{26}	V_{21}	0
V_7	V_{31}	-4	V_{26}	V_{28}	-1
V_8	V_{23}	-5	V_{26}	V_{29}	0
V_9	V_7	0	V_{26}	V_{30}	-2
V_{10}	V_{12}	0	V_{27}	V_2	-7
V_{10}	V_{18}	-1	V_{27}	V_3	0
V_{11}	V_{13}	-3	V_{27}	V_{18}	-1
V_{11}	V_{15}	0	V_{27}	V_{20}	-1
V_{11}	V_{25}	-7	V_{28}	V_9	0
V_{12}	V_4	-2	V_{28}	V_{20}	-1
V_{12}	V_{22}	0	V_{28}	V_{21}	0
V_{13}	V_4	-2	V_{29}	V_{11}	0
V_{14}	V_6	-4	V_{29}	V_{18}	-1
V_{14}	V_{16}	-4	V_{29}	V_{21}	0
V_{14}	V_{17}	-5	V_{29}	V_{30}	-2
V_{14}	V_{20}	-1	V_{30}	V_6	-4
V_{14}	V_{23}	-5	V_{30}	V_{17}	-5
V_{15}	V_4	-2	V_{30}	V_{20}	-1
V_{15}	V_6	-4	V_{31}	V_{18}	-1

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	0
V_{15}	V_{29}	0			

Lampiran 23. Nilai Iterasi Kedelapan Algoritma Bellman-Ford Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_1	V_{11}	0	V_{16}	V_{21}	0
V_1	V_{15}	0	V_{16}	V_{29}	0
V_1	V_{23}	-5	V_{17}	V_{16}	-4
V_1	V_{26}	-2	V_{17}	V_{31}	-4
V_1	V_{29}	0	V_{18}	V_4	-2
V_2	V_5	-5	V_{18}	V_7	0
V_2	V_{30}	-2	V_{18}	V_{28}	-1
V_3	V_5	-5	V_{19}	V_{12}	0
V_3	V_7	0	V_{19}	V_{24}	0
V_3	V_8	0	V_{19}	V_{27}	0
V_3	V_{15}	0	V_{19}	V_{28}	-1
V_3	V_{17}	-5	V_{20}	V_{11}	0
V_3	V_{18}	-1	V_{20}	V_{12}	0
V_3	V_{21}	0	V_{20}	V_{17}	-5
V_3	V_{25}	-7	V_{20}	V_{30}	-2
V_4	V_{17}	-5	V_{21}	V_{13}	-3
V_5	V_{13}	-3	V_{22}	V_5	-5
V_5	V_{25}	-7	V_{22}	V_9	0
V_6	V_4	-2	V_{22}	V_{16}	-4
V_6	V_7	0	V_{23}	V_{10}	0
V_6	V_{31}	-4	V_{25}	V_2	-7
V_7	V_4	-2	V_{25}	V_8	0
V_7	V_{26}	-2	V_{25}	V_{23}	-5
V_7	V_{28}	-1	V_{26}	V_5	-5
V_7	V_{29}	0	V_{26}	V_{21}	0
V_7	V_{31}	-4	V_{26}	V_{28}	-1
V_8	V_{23}	-5	V_{26}	V_{29}	0
V_9	V_7	0	V_{26}	V_{30}	-2
V_{10}	V_{12}	0	V_{27}	V_2	-7
V_{10}	V_{18}	-1	V_{27}	V_3	0
V_{11}	V_{13}	-3	V_{27}	V_{18}	-1
V_{11}	V_{15}	0	V_{27}	V_{20}	-1
V_{11}	V_{25}	-7	V_{28}	V_9	0
V_{12}	V_4	-2	V_{28}	V_{20}	-1
V_{12}	V_{22}	0	V_{28}	V_{21}	0
V_{13}	V_4	-2	V_{29}	V_{11}	0
V_{14}	V_6	-4	V_{29}	V_{18}	-1
V_{14}	V_{16}	-4	V_{29}	V_{21}	0
V_{14}	V_{17}	-5	V_{29}	V_{30}	-2
V_{14}	V_{20}	-1	V_{30}	V_6	-4
V_{14}	V_{23}	-5	V_{30}	V_{17}	-5
V_{15}	V_4	-2	V_{30}	V_{20}	-1
V_{15}	V_6	-4	V_{31}	V_{18}	-1

Simpul Asal	Simpul Tujuan	Bobot	Simpul Asal	Simpul Tujuan	Bobot
V_{15}	V_{26}	-2	V_{31}	V_{19}	0
V_{15}	V_{29}	0			

Lampiran 24. Nilai Potensial $h(v)$ Graf Acak Kedua

Nilai Potensial $h(v)$	Bobot	Nilai Potensial $h(v)$	Bobot
$h(1)$	0	$h(17)$	-5
$h(2)$	-7	$h(18)$	-1
$h(3)$	0	$h(19)$	0
$h(4)$	-2	$h(20)$	-1
$h(5)$	-5	$h(21)$	0
$h(6)$	-4	$h(22)$	0
$h(7)$	0	$h(23)$	-5
$h(8)$	0	$h(24)$	0
$h(9)$	0	$h(25)$	-7
$h(10)$	0	$h(26)$	-2
$h(11)$	0	$h(27)$	0
$h(12)$	0	$h(28)$	-1
$h(13)$	-3	$h(29)$	0
$h(14)$	0	$h(30)$	-2
$h(15)$	0	$h(31)$	-4
$h(16)$	-4		

Lampiran 25. Hasil Pembobotan Ulang Graf Acak Kedua

Simpul Asal	Simpul Tujuan	Bobot Baru	Simpul Asal	Simpul Tujuan	Bobot Baru
V_1	V_{11}	6	V_{16}	V_{21}	3
V_1	V_{15}	7	V_{16}	V_{29}	2
V_1	V_{23}	0	V_{17}	V_{16}	0
V_1	V_{26}	6	V_{17}	V_{31}	0
V_1	V_{29}	8	V_{18}	V_4	3
V_2	V_5	0	V_{18}	V_7	7
V_2	V_{30}	0	V_{18}	V_{28}	5
V_3	V_5	4	V_{19}	V_{12}	5
V_3	V_7	7	V_{19}	V_{24}	5
V_3	V_8	7	V_{19}	V_{27}	8
V_3	V_{15}	6	V_{19}	V_{28}	5
V_3	V_{17}	4	V_{20}	V_{11}	7
V_3	V_{18}	8	V_{20}	V_{12}	7
V_3	V_{21}	4	V_{20}	V_{17}	0
V_3	V_{25}	9	V_{20}	V_{30}	6
V_4	V_{17}	8	V_{21}	V_{13}	11
V_5	V_{13}	0	V_{22}	V_5	7
V_5	V_{25}	0	V_{22}	V_9	4
V_6	V_4	2	V_{22}	V_{16}	12
V_6	V_7	4	V_{23}	V_{10}	3
V_6	V_{31}	7	V_{25}	V_2	3
V_7	V_4	9	V_{25}	V_8	1
V_7	V_{26}	3	V_{25}	V_{23}	6
V_7	V_{28}	9	V_{26}	V_5	9
V_7	V_{29}	4	V_{26}	V_{21}	2
V_7	V_{31}	10	V_{26}	V_{28}	0
V_8	V_{23}	8	V_{26}	V_{29}	0
V_9	V_7	3	V_{26}	V_{30}	5
V_{10}	V_{12}	1	V_{27}	V_2	0
V_{10}	V_{18}	3	V_{27}	V_3	5
V_{11}	V_{13}	4	V_{27}	V_{18}	7
V_{11}	V_{15}	2	V_{27}	V_{20}	4
V_{11}	V_{25}	14	V_{28}	V_9	0
V_{12}	V_4	4	V_{28}	V_{20}	1
V_{12}	V_{22}	1	V_{28}	V_{21}	2
V_{13}	V_4	0	V_{29}	V_{11}	3
V_{14}	V_6	0	V_{29}	V_{18}	7
V_{14}	V_{16}	9	V_{29}	V_{21}	3
V_{14}	V_{17}	7	V_{29}	V_{30}	5
V_{14}	V_{20}	3	V_{30}	V_6	6
V_{14}	V_{23}	11	V_{30}	V_{17}	8
V_{15}	V_4	8	V_{30}	V_{20}	0
V_{15}	V_6	11	V_{31}	V_{18}	0

Simpul Asal	Simpul Tujuan	Bobot Baru	Simpul Asal	Simpul Tujuan	Bobot Baru
V_{15}	V_{26}	0	V_{31}	V_{19}	4
V_{15}	V_{29}	7			

Lampiran 26. Inisialisasi Algoritma Dijkstra Graf Acak Kedua

Simpul	Jarak Awal	Status
V_1	0	Sudah dikunjungi (simpul sumber)
V_2	∞	Belum dikunjungi
V_3	∞	Belum dikunjungi
V_4	∞	Belum dikunjungi
V_5	∞	Belum dikunjungi
V_6	∞	Belum dikunjungi
V_7	∞	Belum dikunjungi
V_8	∞	Belum dikunjungi
V_9	∞	Belum dikunjungi
V_{10}	∞	Belum dikunjungi
V_{11}	∞	Belum dikunjungi
V_{12}	∞	Belum dikunjungi
V_{13}	∞	Belum dikunjungi
V_{14}	∞	Belum dikunjungi
V_{15}	∞	Belum dikunjungi
V_{16}	∞	Belum dikunjungi
V_{17}	∞	Belum dikunjungi
V_{18}	∞	Belum dikunjungi
V_{19}	∞	Belum dikunjungi
V_{20}	∞	Belum dikunjungi
V_{21}	∞	Belum dikunjungi
V_{22}	∞	Belum dikunjungi
V_{23}	∞	Belum dikunjungi
V_{24}	∞	Belum dikunjungi
V_{25}	∞	Belum dikunjungi
V_{26}	∞	Belum dikunjungi
V_{27}	∞	Belum dikunjungi
V_{28}	∞	Belum dikunjungi
V_{29}	∞	Belum dikunjungi
V_{30}	∞	Belum dikunjungi
V_{31}	∞	Belum dikunjungi

Lampiran 27. Proses Relaksasi Algoritma Dijkstra Graf Acak Kedua

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
1	V_{15}	V_{26}	∞	0	$V_{15} \rightarrow V_{26}$	Diperbarui
1	V_{15}	V_{29}	∞	7	$V_{15} \rightarrow V_{29}$	Diperbarui
1	V_{15}	V_{4}	∞	8	$V_{15} \rightarrow V_{4}$	Diperbarui
1	V_{15}	V_{6}	∞	11	$V_{15} \rightarrow V_{6}$	Diperbarui
2	V_{26}	V_{21}	∞	2	$V_{15} \rightarrow V_{26} \rightarrow V_{21}$	Diperbarui
2	V_{26}	V_{28}	∞	0	$V_{15} \rightarrow V_{26} \rightarrow V_{28}$	Diperbarui
2	V_{26}	V_{29}	∞	0	$V_{15} \rightarrow V_{26} \rightarrow V_{29}$	Diperbarui
2	V_{26}	V_{30}	∞	5	$V_{15} \rightarrow V_{26} \rightarrow V_{30}$	Diperbarui
2	V_{26}	V_{5}	∞	9	$V_{15} \rightarrow V_{26} \rightarrow V_{5}$	Diperbarui
3	V_{28}	V_{20}	∞	1	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20}$	Diperbarui
3	V_{28}	V_{21}	2	2	–	Tidak Diperbarui
3	V_{28}	V_{9}	∞	0	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{9}$	Diperbarui
4	V_{29}	V_{11}	∞	3	$V_{15} \rightarrow V_{26} \rightarrow V_{29} \rightarrow V_{11}$	Diperbarui
4	V_{29}	V_{18}	∞	7	$V_{15} \rightarrow V_{26} \rightarrow V_{29} \rightarrow V_{18}$	Diperbarui
4	V_{29}	V_{21}	2	3	–	Tidak Diperbarui
4	V_{29}	V_{30}	5	5	–	Tidak Diperbarui
5	V_{9}	V_{7}	∞	3	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{9} \rightarrow V_{7}$	Diperbarui
6	V_{20}	V_{11}	3	8	–	Tidak Diperbarui
6	V_{20}	V_{12}	∞	8	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{12}$	Diperbarui

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
6	V_{20}	V_{17}	∞	1	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17}$	Diperbarui
6	V_{20}	V_{30}	5	7	–	Tidak Diperbarui
7	V_{17}	V_{16}	∞	1	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{16}$	Diperbarui
7	V_{17}	V_{31}	∞	1	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31}$	Diperbarui
8	V_{16}	V_{21}	2	4	–	Tidak Diperbarui
8	V_{16}	V_{29}	0	3	–	Tidak Diperbarui
9	V_{31}	V_{18}	∞	1	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31} \rightarrow V_{18}$	Diperbarui
9	V_{31}	V_{19}	∞	5	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31} \rightarrow V_{19}$	Diperbarui
10	V_{18}	V_{28}	0	6	–	Tidak Diperbarui
10	V_{18}	V_4	∞	4	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31} \rightarrow V_{18} \rightarrow V_4$	Diperbarui
10	V_{18}	V_7	3	8	–	Tidak Diperbarui

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
11	V_{21}	V_{13}	∞	13	$V_{15} \rightarrow V_{26} \rightarrow V_{21} \rightarrow V_{13}$	Diperbarui
12	V_{11}	V_{13}	∞	7	$V_{15} \rightarrow V_{26} \rightarrow V_{29} \rightarrow V_{11} \rightarrow V_{13}$	Diperbarui
12	V_{11}	V_{15}	0	5	–	Tidak Diperbarui
12	V_{11}	V_{25}	∞	17	$V_{15} \rightarrow V_{26} \rightarrow V_{29} \rightarrow V_{11} \rightarrow V_{25}$	Diperbarui
13	V_{7}	V_{26}	0	6	–	Tidak Diperbarui
13	V_{7}	V_{28}	0	12	–	Tidak Diperbarui
13	V_{7}	V_{29}	0	7	–	Tidak Diperbarui
13	V_{7}	V_{31}	1	13	–	Tidak Diperbarui
13	V_{7}	V_{4}	4	12	–	Tidak Diperbarui
14	V_{4}	V_{17}	1	12	–	Tidak Diperbarui
15	V_{19}	V_{12}	8	10	–	Tidak Diperbarui
15	V_{19}	V_{24}	∞	10	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31} \rightarrow V_{19} \rightarrow V_{24}$	Diperbarui
15	V_{19}	V_{27}	∞	13	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31} \rightarrow V_{19} \rightarrow V_{27}$	Diperbarui
15	V_{19}	V_{28}	0	10	–	Tidak Diperbarui
16	V_{30}	V_{17}	1	13	–	Tidak

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
						Diperbarui
16	V_{30}	V_{20}	1	5	–	Tidak Diperbarui
16	V_{30}	V_{6}	11	11	–	Tidak Diperbarui
17	V_{13}	V_{4}	4	7	–	Tidak Diperbarui
18	V_{12}	V_{22}	∞	9	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{12} \rightarrow V_{22}$	Diperbarui
18	V_{12}	V_{4}	4	12	–	Tidak Diperbarui
19	V_{22}	V_{16}	1	21	–	Tidak Diperbarui
19	V_{22}	V_{5}	9	16	–	Tidak Diperbarui
19	V_{22}	V_{9}	0	13	–	Tidak Diperbarui
20	V_{5}	V_{13}	7	9	–	Tidak Diperbarui
20	V_{5}	V_{25}	∞	9	$V_{15} \rightarrow V_{26} \rightarrow V_{5} \rightarrow V_{25}$	Diperbarui
21	V_{25}	V_{2}	∞	12	$V_{15} \rightarrow V_{26} \rightarrow V_{5} \rightarrow V_{25} \rightarrow V_{2}$	Diperbarui
21	V_{25}	V_{23}	∞	15	$V_{15} \rightarrow V_{26} \rightarrow V_{5} \rightarrow V_{25} \rightarrow V_{23}$	Diperbarui
21	V_{25}	V_{8}	∞	10	$V_{15} \rightarrow V_{26} \rightarrow V_{5} \rightarrow V_{25} \rightarrow V_{8}$	Diperbarui
23	V_{8}	V_{23}	15	18	–	Tidak Diperbarui
24	V_{6}	V_{31}	1	18	–	Tidak Diperbarui
24	V_{6}	V_{4}	4	13	–	Tidak Diperbarui
24	V_{6}	V_{7}	3	15	–	Tidak

Iterasi	Simpul diproses	Simpul diperbarui	Jarak sebelumnya	Jarak terpendek	Lintasan	Keterangan
						Diperbarui
25	V_2	V_{30}	5	12	–	Tidak Diperbarui
25	V_2	V_5	9	12	–	Tidak Diperbarui
26	V_{27}	V_{18}	1	20	–	Tidak Diperbarui
26	V_{27}	V_2	12	13	–	Tidak Diperbarui
26	V_{27}	V_{20}	1	17	–	Tidak Diperbarui
26	V_{27}	V_3	∞	18	$V_{15} \rightarrow V_{26} \rightarrow V_{28} \rightarrow V_{20} \rightarrow V_{17} \rightarrow V_{31} \rightarrow V_{19} \rightarrow V_{27} \rightarrow V_3$	Diperbarui
27	V_{23}	V_{10}	∞	18	$V_{15} \rightarrow V_{26} \rightarrow V_5 \rightarrow V_{25} \rightarrow V_{23} \rightarrow V_{10}$	Diperbarui
28	V_{10}	V_{12}	8	19	–	Tidak Diperbarui
28	V_{10}	V_{18}	1	21	–	Tidak Diperbarui
29	V_3	V_{15}	0	24	–	Tidak Diperbarui
29	V_3	V_{17}	1	22	–	Tidak Diperbarui
29	V_3	V_{18}	1	26	–	Tidak Diperbarui
29	V_3	V_{21}	2	22	–	Tidak Diperbarui
29	V_3	V_{25}	9	27	–	Tidak Diperbarui
29	V_3	V_5	9	22	–	Tidak Diperbarui
29	V_3	V_7	3	25	–	Tidak Diperbarui
29	V_3	V_8	10	25	–	Tidak Diperbarui

Lampiran 28. Perbandingan Lintasan Terpendek Bellman-Ford dengan Dijkstra pada Graf Acak Pertama

Data lengkap perbandingan Algoritma Bellman-Ford dengan Algoritma Dijkstra pada Graf Acak Pertama dapat diakses melalui tautan berikut: https://bit.ly/44QkseR

No.	Simpul Asal → Simpul Tujuan	Algoritma Bellman-Ford	Algoritma Dijkstra	Selisih Panjang Rute
1.	$V_1 \rightarrow V_1$	0	0	0
2.	$V_1 \rightarrow V_{16}$	2	2	0
3.	$V_1 \rightarrow V_{26}$	11	11	0
4.	$V_1 \rightarrow V_2$	20	20	0
5.	$V_1 \rightarrow V_4$	4	4	0
6.	$V_1 \rightarrow V_{19}$	27	27	0
7.	$V_1 \rightarrow V_3$	17	17	0
8.	$V_1 \rightarrow V_9$	11	11	0
9.	$V_1 \rightarrow V_{24}$	16	16	0
10.	$V_1 \rightarrow V_{30}$	11	11	0
11.	$V_1 \rightarrow V_5$	23	23	0
12.	$V_1 \rightarrow V_7$	27	27	0
13.	$V_1 \rightarrow V_6$	29	29	0
14.	$V_1 \rightarrow V_{17}$	20	20	0
15.	$V_1 \rightarrow V_{25}$	24	24	0
16.	$V_1 \rightarrow V_{31}$	30	30	0
17.	$V_1 \rightarrow V_8$	30	30	0
18.	$V_1 \rightarrow V_{18}$	6	6	0
19.	$V_1 \rightarrow V_{27}$	31	31	0
20.	$V_1 \rightarrow V_{20}$	11	11	0
⋮	⋮	⋮	⋮	⋮
942.	$V_{31} \rightarrow V_9$	14	14	0
943.	$V_{31} \rightarrow V_{24}$	19	19	0
944.	$V_{31} \rightarrow V_{30}$	14	14	0
945.	$V_{31} \rightarrow V_5$	26	26	0
946.	$V_{31} \rightarrow V_7$	30	30	0
947.	$V_{31} \rightarrow V_6$	32	32	0
948.	$V_{31} \rightarrow V_{17}$	23	23	0
949.	$V_{31} \rightarrow V_{25}$	27	27	0
950.	$V_{31} \rightarrow V_{31}$	0	0	0
951.	$V_{31} \rightarrow V_8$	33	33	0
952.	$V_{31} \rightarrow V_{18}$	7	7	0
953.	$V_{31} \rightarrow V_{27}$	34	34	0
954.	$V_{31} \rightarrow V_{20}$	14	14	0
955.	$V_{31} \rightarrow V_{22}$	24	24	0
956.	$V_{31} \rightarrow V_{15}$	30	30	0
957.	$V_{31} \rightarrow V_{14}$	19	19	0

No.	Simpul Asal → Simpul Tujuan	Algoritma Bellman-Ford	Algoritma Dijkstra	Selisih Panjang Rute
958.	$V_{31} \rightarrow V_{29}$	39	39	0
959.	$V_{31} \rightarrow V_{21}$	18	18	0
960.	$V_{31} \rightarrow V_{28}$	30	30	0
961.	$V_{31} \rightarrow V_{23}$	17	17	0
Total Selisih				0

Lampiran 29. Perbandingan Lintasan Terpendek Bellman-Ford dengan Dijkstra pada Graf Acak Kedua

Data lengkap perbandingan lintasan terpendek Algoritma Bellman-Ford dengan Algoritma Dijkstra pada Graf Acak Kedua dapat diakses melalui tautan berikut: https://bit.ly/4kzUrF9

No.	Simpul Asal → Simpul Tujuan	Algoritma Bellman-Ford	Algoritma Dijkstra	Selisih Panjang Rute
1.	$V_1 \rightarrow V_1$	0	0	0
2.	$V_1 \rightarrow V_{11}$	6	6	0
3.	$V_1 \rightarrow V_{15}$	7	7	0
4.	$V_1 \rightarrow V_{23}$	0	0	0
5.	$V_1 \rightarrow V_{26}$	6	6	0
6.	$V_1 \rightarrow V_{29}$	6	6	0
7.	$V_1 \rightarrow V_2$	15	15	0
8.	$V_1 \rightarrow V_5$	12	12	0
9.	$V_1 \rightarrow V_{30}$	11	11	0
10.	$V_1 \rightarrow V_3$	24	24	0
11.	$V_1 \rightarrow V_7$	9	9	0
12.	$V_1 \rightarrow V_8$	13	13	0
13.	$V_1 \rightarrow V_{17}$	7	7	0
14.	$V_1 \rightarrow V_{18}$	6	6	0
15.	$V_1 \rightarrow V_{21}$	8	8	0
16.	$V_1 \rightarrow V_{25}$	12	12	0
17.	$V_1 \rightarrow V_4$	8	8	0
18.	$V_1 \rightarrow V_{13}$	10	10	0
19.	$V_1 \rightarrow V_6$	17	17	0
20.	$V_1 \rightarrow V_{31}$	7	7	0
⋮	⋮	⋮	⋮	⋮
942.	$V_{31} \rightarrow V_7$	7	7	0
943.	$V_{31} \rightarrow V_8$	13	13	0
944.	$V_{31} \rightarrow V_{17}$	6	6	0
945.	$V_{31} \rightarrow V_{18}$	0	0	0
946.	$V_{31} \rightarrow V_{21}$	7	7	0
947.	$V_{31} \rightarrow V_{21}$	12	12	0
948.	$V_{31} \rightarrow V_4$	3	3	0
949.	$V_{31} \rightarrow V_{13}$	12	12	0
950.	$V_{31} \rightarrow V_6$	18	18	0
951.	$V_{31} \rightarrow V_{31}$	0	0	0
952.	$V_{31} \rightarrow V_{28}$	5	5	0
953.	$V_{31} \rightarrow V_9$	5	5	0
954.	$V_{31} \rightarrow V_{10}$	21	21	0
955.	$V_{31} \rightarrow V_{12}$	9	9	0
956.	$V_{31} \rightarrow V_{22}$	10	10	0
957.	$V_{31} \rightarrow V_{16}$	6	6	0

No.	Simpul Asal → Simpul Tujuan	Algoritma Bellman-Ford	Algoritma Dijkstra	Selisih Panjang Rute
958.	$V_{31} \rightarrow V_{20}$	6	6	0
959.	$V_{31} \rightarrow V_{19}$	4	4	0
960.	$V_{31} \rightarrow V_{24}$	9	9	0
961.	$V_{31} \rightarrow V_{27}$	12	12	0
Total Selisih				0

RIWAYAT HIDUP

Natasya Thalia Salsabillah, lahir di Malang pada 08 Juli 2002, biasa dipanggil Natasya atau Bella. Penulis merupakan anak kedua dari Ayah Gatot Subroto dan Mama Entri Rahayu serta adik dari Ersya Maulidyah Rahayu Subrata dan kakak dari Julia Agatha Teris Sabrina. Penulis telah menempuh pendidikan mulai dari TK Kemala Bhayangkari 13 yang lulus pada tahun 2009, dilanjutkan menempuh pendidikan sekolah dasar di SD Sidrap Luar Bontang dan lulus pada tahun 2015. Kemudian penulis melanjutkan pendidikan sekolah menengah pertama di SMPN 4 Bontang dan lulus pada tahun 2018. Setelah itu, melanjutkan pendidikan sekolah menengah atas di SMAN 1 Sumberpucung dan lulus pada tahun 2021. Pada tahun yang sama, penulis melanjutkan pendidikan di Universitas Islam Negeri Maulana Malik Ibrahim Malang pada Program Studi Matematika, Fakultas Sains dan Teknologi. Selama menempuh pendidikan tinggi, penulis juga berperan aktif dalam organisasi HMPS “Integral” Matematika Tahun 2023 sebagai anggota Divisi Kewirausahaan.

KEMENTERIAN AGAMA RI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Gajayana No.50 Dinoyo Malang Telp. / Fax. (0341)558933

BUKTI KONSULTASI SKRIPSI

Nama : Natasya Thalia Salsabillah
NIM : 210601110080
Fakultas / Jurusan : Sains dan Teknologi / Matematika
Judul Skripsi : Pembobotan Ulang pada Graf Berbobot Negatif untuk Menerapkan Algoritma Dijkstra dalam Menentukan Lintasan Terpendek
Pembimbing I : Mohammad Nafie Jauhari, M.Si
Pembimbing II : Ach. Nashichuddin, M.A

No	Tanggal	Hal	Tanda Tangan
1.	17 September 2024	Konsultasi Topik dan Data	1.
2.	31 Oktober 2024	Konsultasi Bab I, II, dan III	2.
3.	7 November 2024	Konsultasi Bab I, II, dan III	3.
4.	19 November 2024	Konsultasi Bab I, II, dan III	4.
5.	21 November 2024	Konsultasi Bab I, II, dan III	5.
6.	05 Desember 2024	ACC Bab I, II, dan III	6.
7.	27 November 2024	Konsultasi Kajian Agama Bab I dan II	7.
8.	2 Desember 2024	ACC Kajian Agama Bab I dan II	8.
9.	15 Desember 2024	ACC Seminar Proposal	9.
10.	10 Februari 2025	Konsultasi Revisi Seminar Proposal	10.
11.	17 Februari 2025	Konsultasi Bab IV dan V	11.
12.	25 Februari 2025	Konsultasi Bab IV dan V	12.
13.	14 Maret 2025	Konsultasi Bab IV dan V	13.
14.	16 April 2025	Konsultasi Bab IV dan V	14.

KEMENTERIAN AGAMA RI
UNIVERSITAS ISLAM NEGERI
MAULANA MALIK IBRAHIM MALANG
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Gajayana No.50 Dinoyo Malang Telp. / Fax. (0341)558933

15.	28 April 2025	ACC Bab IV dan V	15.
16.	22 April 2025	Konsultasi Kajian Agama Bab IV	16.
17.	25 April 2025	ACC Kajian Agama Bab IV	17.
18.	8 Mei 2025	ACC Seminar Hasil	18.
19.	19 Mei 2025	Konsultasi Revisi Seminar Hasil	19.
20.	21 Mei 2025	Konsultasi Revisi Seminar Hasil	20.
21.	23 Mei 2025	Konsultasi Revisi Seminar Hasil	21.
22.	26 Mei 2025	Konsultasi Revisi Seminar Hasil	22.
23.	27 Mei 2025	ACC Sidang Skripsi	23.
24.	12 Juni 2025	ACC Keseluruhan	24.

Malang, 12 Juni 2025
Mengetahui,
Ketua Program Studi Matematika

Dr. Elly Susanti, M.Sc.
NIP. 19741129 200012 2 005