NAMA : Restu Wibisono
NPM : 2340506061

Algoritma Dijkstra

I. Pengertian

Algoritma Dijkstra adalah metode yang digunakan untuk menemukan lintasan terpendek dari suatu titik tertentu ke setiap titik lain dalam sebuah graf berbobot. Dalam konteks ini, bobot mewakili jarak, biaya, atau nilai yang harus ditempuh untuk berpindah dari satu titik ke titik lainnya. Algoritma ini sangat penting dalam berbagai aplikasi yang melibatkan perencanaan rute atau jaringan.

Algoritma Dijkstra dikembangkan oleh seorang ilmuwan komputer asal Belanda bernama Edger Wybe Dijkstra pada tahun 1959. Beliau menggambarkan algoritma ini dalam sebuah makalah, dan sejak saat itu, algoritma Dijkstra menjadi salah satu algoritma yang paling terkenal dan sering digunakan dalam dunia komputasi.

II. Langkah Langkah

Berikut adalah langkah-langkah Algoritma Dijkstra untuk menemukan jalur terpendek dari satu titik ke titik lain dalam sebuah graf berbobot:

1. Inisialisasi

   Tentukan titik awal (sumber) dan tetapkan jarak awal dari titik awal ke semua titik lain sebagai tak terhingga, kecuali untuk titik awal itu sendiri yang diatur menjadi 0.

2. Tandai Titik Awal

   Tandai titik awal sebagai telah dikunjungi atau “tertentu”.

3. Perbarui Jarak

   Mulai dari titik awal, periksa semua tetangga langsung yang belum ditandai. Hitung jarak dari titik awal ke tetangga-tetangga tersebut melalui jalur yang saat ini telah diketahui. Perbarui jarak terpendek jika jarak baru lebih kecil dari jarak sebelumnya.

4. Pilih Tetangga Terdekat

   Dari tetangga-tetangga yang belum ditandai, pilih titik dengan jarak terpendek sebagai titik berikutnya untuk dikunjungi.

5. Tandai Titik Terpilih

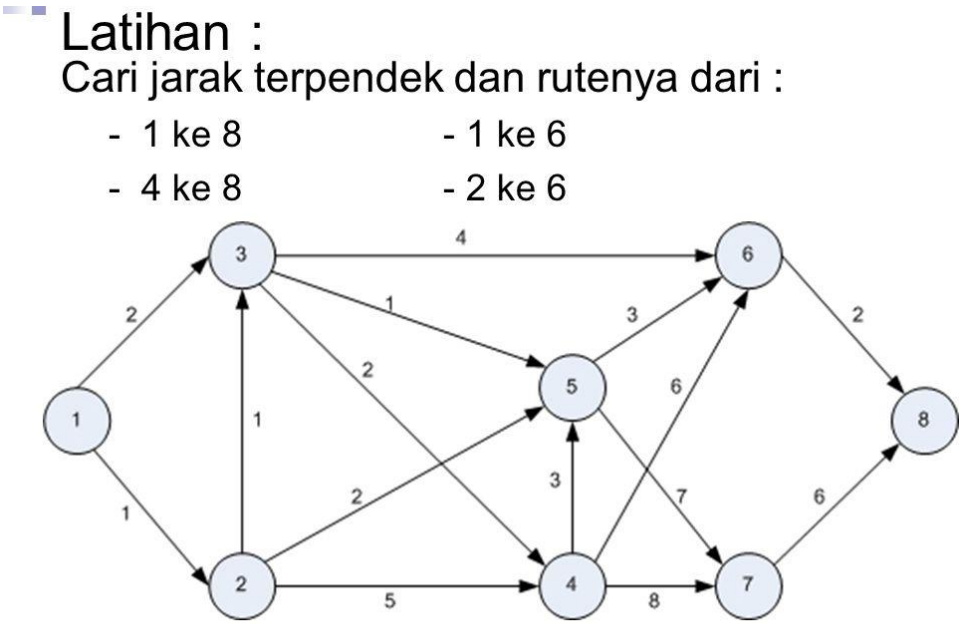
   Tandai titik terpilih sebagai telah dikunjungi atau “tertentu”.

6. Ulangi

   Ulangi langkah 3 hingga langkah 5 sampai semua titik telah dikunjungi atau sampai titik akhir (target) telah ditandai.

7. Hasil

   Setelah semua titik telah dikunjungi, jarak terpendek dari titik awal ke semua titik lain akan terhitung dengan benar. Jalur terpendek dari titik awal ke titik akhir dapat dihitung dengan mengikuti panah dari titik akhir ke titik awal berdasarkan informasi jarak yang telah dihitung.



(https://i1.wp.com/slideplayer.info/slide/3669825/12/images/9/Latihan+:+Cari+jarak+terpendek+dan+rutenya+dari+:+-+1+ke+ke+6.jpg)

III. Istilah-istilah

1. Graf (Graph): Representasi visual atau matematis dari kumpulan simpul (node) yang terhubung oleh sisi (edge). Graf digunakan untuk menggambarkan hubungan antara simpul-simpul dalam konteks algoritma Dijkstra.
2. Simpul (Node): Titik dalam graf yang mewakili entitas atau lokasi tertentu. Setiap simpul memiliki label atau nama yang unik.
3. Sisi (Edge): Koneksi antara dua simpul dalam graf. Setiap sisi dapat memiliki bobot atau jarak yang menggambarkan biaya atau jarak antara dua simpul yang terhubung.

4. Bobot (Weight): Angka yang terkait dengan setiap sisi dalam graf. Bobot menggambarkan biaya atau jarak yang harus ditempuh untuk mencapai simpul tujuan dari simpul awal.
5. Graf Berbobot (Weighted Graph): Graf yang memiliki bobot pada setiap sisi. Dalam algoritma Dijkstra, bobot harus non-negatif.
6. Simpul Awal (Source Node): Simpul awal atau simpul sumber dari mana algoritma Dijkstra dimulai untuk mencari jalur terpendek ke simpul-simpul lain dalam graf.
7. Simpul Tujuan (Destination Node): Simpul yang menjadi tujuan dalam pencarian jalur terpendek dari simpul awal. Algoritma Dijkstra akan mencari jalur terpendek dari simpul awal ke simpul tujuan.
8. Jarak Terpendek (Shortest Distance): Jarak terpendek antara simpul awal dan simpul tujuan dalam graf. Algoritma Dijkstra bertujuan untuk mencari jarak terpendek ini.
9. Tabel Jarak (Distance Table): Tabel yang menyimpan informasi tentang jarak terpendek dari simpul awal ke setiap simpul lain dalam graf. Tabel ini diperbarui secara iteratif selama eksekusi algoritma Dijkstra.
10. Set Simpul Terproses (Processed Node Set): Set simpul yang telah diproses atau dieksplorasi sepenuhnya oleh algoritma Dijkstra. Simpul-simpul dalam set ini telah memiliki jarak terpendek yang final.
11. Set Simpul Belum Diproses (Unprocessed Node Set): Set simpul yang belum diproses atau dieksplorasi sepenuhnya oleh algoritma Dijkstra. Simpul-simpul dalam set ini masih memiliki jarak terpendek yang mungkin berubah selama eksekusi algoritma.

IV. Masalah yang dapat diselesaikan

Algoritma Dijkstra dapat digunakan untuk memecahkan berbagai masalah optimasi dengan mencari jalur terpendek. Beberapa contoh penerapannya meliputi:

1. Sistem Navigasi

   Menemukan rute terpendek antara dua lokasi pada peta.

2. Jaringan Komputer

   Menghitung jalur tercepat dalam suatu jaringan komunikasi.

3. Perencanaan Logistik

   Menentukan jalur terpendek untuk pengiriman barang.

4. Permainan

   Digunakan untuk menemukan jalur terpendek dalam permainan seperti game strategi atau teka-teki.

V. Keuntungan Algoritma Dijkstra

Algoritma Dijkstra menawarkan sejumlah keuntungan yang membuatnya menjadi algoritma yang populer digunakan dalam berbagai bidang. Beberapa di antaranya adalah:

1. Efisiensi

   Algoritma Dijkstra dapat menemukan jalur terpendek dengan efisien, bahkan pada graf yang besar dan kompleks.

2. Keakuratan

   Algoritma ini menghasilkan jalur terpendek dengan bobot yang optimal, sehingga dapat diandalkan dalam banyak kasus.

3. Aplikasi yang Luas

   Dijkstra dapat diterapkan dalam berbagai konteks, seperti rute perjalanan, jaringan komputer, perencanaan logistik, dan banyak lagi.

VI. Kekurangan Algoritma Dijkstra

1. Mengasumsikan Bobot Non-Negatif

   Algoritma Dijkstra tidak dapat bekerja dengan baik jika terdapat bobot negatif pada graf, karena dapat menghasilkan hasil yang tidak tepat.

2. Kompleksitas pada Graf Besar

   Pada graf yang sangat besar, algoritma ini dapat menjadi lambat dan memakan banyak sumber daya.

Directed Acyclic Graph

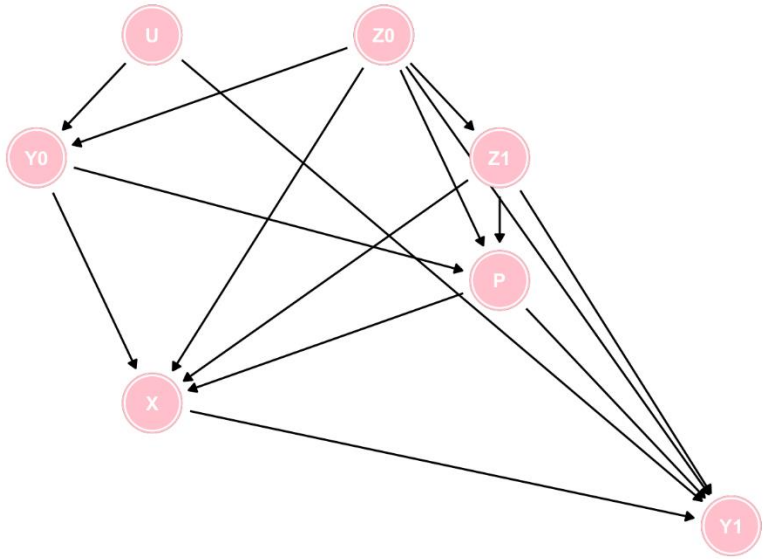
I. Pengertian

Directed Acyclic Graph (DAG) merupakan model struktur data yang digunakan dalam beberapa teknologi blockchain saat ini. DAG memang memiliki beberapa kesamaan dengan blockchain, namun faktanya DAG memiliki sistem yang lebih cepat dan efisien.

DAG hampir serupa dengan blockchain, DAG sama-sama sebagai sebuah database yang memungkinkan para penggunanya untuk melakukan transaksi dan mencatat transaksi.

Blockchain biasanya menggunakan blok untuk mencatat berbagai transaksi di jaringan. Sedangkan Directed Acyclic Graph menggunakan grafik yang diarahkan dan tidak berputar untuk mencatat transaksi. Nah, grafik ini terdiri dari simpul yang terhubung oleh edge, sehingga transaksi akan terjadi diantara simpul-simpul tersebut.

Dalam dunia teknologi blockchain, DAG digunakan sebagai sebuah alternatif untuk memproses transaksi atau blok yang lebih efisien dibandingkan dengan teknologi blockchain yang masih konvensional. Directed Acyclic Graph mampu memproses transaksi secara paralel tanpa harus menunggu blok sebelumnya selesai diproses.



(https://www.brophyj.com/talk/directed-acyclic-graphs-the-view-of-a-clinical-scientist/featured.png)

II. Istilah-istilah

1. Graf (Graph): Representasi visual atau matematis dari kumpulan simpul (node) yang terhubung oleh sisi (edge). Graf ini memiliki arah pada setiap sisi, tetapi tidak ada jalur yang membentuk siklus.
2. Simpul (Node): Titik dalam graf yang mewakili entitas atau lokasi tertentu. Setiap simpul memiliki label atau nama yang unik.
3. Sisi (Edge): Koneksi antara dua simpul dalam graf. Setiap sisi memiliki arah yang menunjukkan aliran dari satu simpul ke simpul lainnya.

4. Graf Berarah (Directed Graph): Graf yang memiliki arah pada setiap sisi. Dalam DAG, arah sisi-sisi tersebut membentuk urutan atau ketergantungan antara simpul-simpul.
5. Siklus (Cycle): Jalur tertutup dalam graf yang membentuk lingkaran. Dalam DAG, tidak ada siklus, artinya tidak ada jalur yang membentuk lingkaran.
6. Simpul Awal (Source Node): Simpul awal atau simpul sumber dari mana algoritma atau proses dimulai dalam DAG. Simpul ini tidak memiliki sisi masuk.
7. Simpul Tujuan (Sink Node): Simpul tujuan atau simpul akhir yang menjadi target atau hasil dari algoritma atau proses dalam DAG. Simpul ini tidak memiliki sisi keluar.
8. Topological Sort: Proses pengurutan simpul-simpul dalam DAG sedemikian rupa sehingga setiap sisi hanya mengarah dari simpul dengan urutan lebih rendah ke simpul dengan urutan lebih tinggi. Topological sort digunakan untuk memastikan bahwa tidak ada ketergantungan siklik dalam DAG.
9. Dependensi (Dependency): Hubungan ketergantungan antara simpul-simpul dalam DAG. Simpul yang tergantung pada simpul lain disebut simpul yang memiliki dependensi.
10. Longest Path: Jalur terpanjang antara dua simpul dalam DAG. Dalam DAG, mencari jalur terpanjang melibatkan mencari jalur dengan jumlah bobot maksimum antara simpul awal dan simpul tujuan.

III. Keunggulan

1. Memiliki skalabilitas yang lebih baik. Faktanya, Directed Acyclic Graph mampu menangani transaksi secara bersamaan tanpa memerlukan blok yang sama seperti pada blockchain. Ini memungkinkan Directed Acyclic Graph untuk mengatasi masalah skalabilitas yang sering terjadi pada blockchain.
2. Biaya Directed Acyclic Graph lebih rendah. Directed Acyclic Graph mengurangi biaya transaksi dengan menghilangkan biaya penambangan. Pengguna hanya membayar biaya jaringan yang lebih rendah, yang memungkinkan pengguna untuk mengirim uang tanpa biaya yang signifikan.
3. Memiliki kecepatan transaksi yang lebih cepat. Directed Acyclic Graph memungkinkan transaksi untuk diproses dengan sangat cepat karena tidak memerlukan waktu untuk menambang blok seperti pada blockchain. Ini memungkinkan transaksi untuk diproses hampir seketika.

4. Desentralisasi yang lebih tinggi. Dalam arsitektur Directed Acyclic Graph, setiap simpul dalam jaringan dapat berfungsi sebagai penyimpanan dan validasi transaksi. Ini memungkinkan jaringan untuk menjadi lebih terdesentralisasi daripada blockchain tradisional, di mana hanya sejumlah kecil penambang yang berfungsi sebagai validator.

IV. **Kelemahan**

1. Permasalahan skalabilitas. Skalabilitas menjadi masalah karena dengan meningkatnya jumlah transaksi yang dibuat di jaringan DAG, semakin sulit bagi node untuk memproses transaksi tersebut. Hal ini karena setiap transaksi harus diverifikasi oleh sejumlah node sebelum dapat dimasukkan ke dalam bukti konsensus.
2. Potensi serangan 51 persen. Walaupun jarang terjadi, ada kemungkinan serangan mayoritas 51% di jaringan DAG, di mana penambang atau kelompok penambang dapat mengambil kendali atas sebagian besar kekuatan pemrosesan jaringan dan mengontrol transaksi di dalamnya.
3. Biaya penyimpanan yang cukup tinggi. Sistem Directed Acyclic Graph memerlukan biaya penyimpanan yang lebih tinggi karena setiap node harus menyimpan salinan transaksi sebelumnya. Hal ini dapat menjadi masalah ketika transaksi dalam jumlah besar dilakukan di jaringan.
4. Ketergantungan pada arsitektur. Sistem Directed Acyclic Graph sangat bergantung pada arsitektur teknis tertentu, seperti cara transaksi disusun dan diproses. Jika ada masalah dalam arsitektur ini, maka jaringan DAG dapat menjadi rentan terhadap serangan.
5. Kurangnya dukungan infrastruktur: Meskipun ada beberapa aset kripto yang menggunakan Directed Acyclic Graph, namun infrastruktur untuk mendukung pengembangan jaringan Directed Acyclic Graph masih kurang. Hal ini dapat menjadi hambatan bagi adopsi lebih luas dari teknologi ini.