MAKALAH STRUKTUR DATA DAN ALGORITMA

“GRAPH (Graph Data Type, Travelsals, Disjoint Sets)”

DISUSUN OLEH KELOMPOK 8:

Nama : 1. Aggra Kurnia Idhan (G1A024003)
2. Ahmad Febri yan (G1A024005)
3. Muhammad Nathan Algibran (G1A024057)
4. Tri Haiji Januarli (G1A024069)
5. Farhan Muhammad Rizki (G1A024093)
6. M. Arif Rahman (G1A024109)

Kelas : A Informatika

DOSEN PENGAMPU:

Arie Vatresia, S.T., M.T.I., Ph.D.

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS BENGKULU

2025

KATA PENGANTAR

Puji dan syukur kami panjatkan kepada Tuhan Yang Maha Esa atas rahmat, berkah, dan hidayah-Nya, sehingga kami dapat menyelesaikan makalah berjudul " GRAPH (Graph Data Type, Traversals, Disjoint Sets)" dengan tepat waktu. Makalah ini disusun sebagai tugas mata kuliah Struktur Data Dan Algoritma, dengan tujuan memperdalam pemahaman tentang teori hingga studi kasus dalam Graph. Kami menyadari bahwa Graph sangat berguna dalam berbagai aplikasi komputasi.

Kami mengucapkan terima kasih kepada Ibu Arie Vatresia, S.T., M.T.I., Ph.D. selaku dosen mata kuliah Struktur Data Dan Algoritma yang telah membimbing dan memberi ilmu yang sangat berarti dalam penyusunan makalah ini. Dan terima kasih juga untuk semua teman dan rekan – rekan yang telah ikut berpartisipasi dalam proses penyusunan makalah. Kami menyadari bahwa makalah ini masih jauh dari kata sempurna, baik dari segi isi maupun penyajiannya, segalah bentuk kritik dan saran yang membangun sangat kami terima demi makalah ini lebih baik lagi dimasa mendatang. Semoga makalah ini dapat bermanfaat untuk kita semua.

Bengkulu, 29 April 2025

Penulis

DAFTAR ISI

KATA PENGANTAR……..……..... ii

DAFTAR ISI……...……...…….……..….... iii

BAB I PENDAHULUAN..…….…….….. 1

1.1 . Latar Belakang..…….. 1

1.2 Rumusan masalah…..…… 1

BAB II PEMBAHASAN.……..2

2.1. Struktur Data Type ……..…….…….....2

2.2 Graph Traversal…..……6

2.3 Shortest Path Tree.…….…..10

2.4 Studi Kasus.…..…….…..13

BAB III PENUTUP.……...….... 21

3.1 .

Kesimpulan…….……..…….... 21

DAFTAR PUSTAKA……. 22

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada era mobilitas pribadi yang semakin tinggi seperti saat ini, menentukan jalur perjalanan yang efektif dan efisien sebelum melakukan perjalanan jauh menjadi aspek penting. Hal ini dikarenakan perjalanan yang terlalu lama dapat menimbulkan rasa tidak nyaman, ditambah lagi kondisi jalan belum tentu selalu lancar. Oleh karena itu, perencanaan rute menjadi sangat penting untuk memastikan perjalanan berlangsung nyaman dan optimal.

Sebagai contoh, ketika ingin melakukan perjalanan dari Bengkulu menuju Banda Aceh, kita perlu memilih rute terpendek dan paling efisien. Untuk menganalisis rute Bengkulu – Banda Aceh, digunakan pendekatan berbasis Graph di mana simpul merepresentasikan kota dan sisi merepresentasikan jalan antar kota. Analisis kemudian dilakukan menggunakan Algoritma Dijkstra untuk menentukan jalur dengan jarak terpendek.

Dalam studi kasus ini, rute yang dapat dipilih adalah melalui jalur darat dengan rute Bengkulu → Lubuklinggau → Palembang → Jambi → Pekanbaru → Duri → Medan → Lhokseumawe → Banda Aceh. Jalur ini lebih efektif dibandingkan dengan alternatif lain yang mungkin lebih panjang atau lebih lambat. Apabila dibandingkan, menggunakan jalur darat ini biasanya lebih efisien daripada jalur laut yang memakan waktu lebih lama. Oleh karena itu, dengan perencanaan rute berbasis Graph dan perhitungan algoritma, kita dapat menemukan solusi perjalanan yang lebih cepat, nyaman, dan efisien.

1.2 Rumusan Masalah

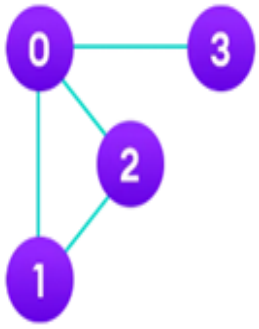
1. Bagaimana karakteristik dasar dari struktur data graph dalam teori graph?
2. Bagaimana cara kerja graph traversal?
3. Bagimana algoritma - algoritma membentuk Shortest Path Tree?
4. Bagaimana bentuk studi kasus dalam graph?

BAB II
PEMBAHASAN

2.1 Struktur Data Type

Graph merupakan jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier), dimana tipe ini mengandung beberapa node atau simpul berhingga untuk menyimpan data dan antara dua simpul terdapat hubungan saling berkaitan. Dalam dunia ilmu computer, graph dianggap sebagai sebuah struktur data atau yang lebih spesifik disebut Abstract Data Type (ADT).

Struktur data terdiri dari sejumlah simpul dan sisi yang menghubungkan diantara simpul – simpul tersebut, dalam graph tidak hanya menyimpan data (simpul) tapi juga menyimpan hubungan eksplisit antar data (tepi/edge). Struktur ini berbentuk jaringan atau network, dimana semuanya terhubung satu sama lain, antar elemennya bersifat one-to-many, many-to-one, atau many-to-many. Simpul pada graph disebut verteks (V), sisi yang menghubungkan antar verteks disebut edge (E), pasangan (x,y) sebagai edge, menyatakan pasangan simpul x terhubung ke simpul y. sebagai contoh :



Gambar 1. Graph

Graph diatas terdiri atas 4 buah verteks dan 4 pasang sisi atau edge. Dengan verteks disimbolkan dengan V, edge dilambangkan E, dan graph dilambangkan G, diilustrasikan dalam notasi:

V = {0, 1, 2, 3}

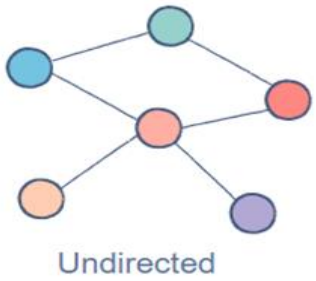
E = {(0,1), (0,2), (0,3), (1,2)}

G = {V,E}

- Jenis-Jenis Graph

Graph dibedakan berdasarkan arah jelajahnya (koneksi):

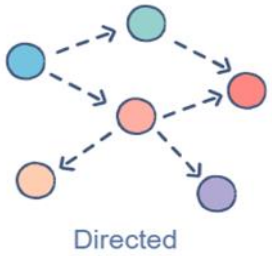
- Undirected graph



Gambar 2. Undirected graph

Simpul simpul saling terhubung dengan edge yang sifatnya dua arah, Ketika simpul 1 dan 2 saling terhubung, kita dapat menjelajahi dari simpul 1 ke simpul 2, begitupun sebaliknya. Contohnya jalan dua arah, aliran data, peta jalan satu arah, system pengiriman barang.

- Directed Graph



Gambar 3. Directed graph

Simpul – simpul terhubung oleh edge yang hanya dapat menjelajahi satu arah pada simpul yang ditunjuk, jika ada simpul A yang terhubung ke simpul B, arah panah menuju simpul B maka hanya dapat menjelajah (traversing) dari simpul A ke simpul B, tidak berlaku sebaliknya. Contohnya jalan satu arah, jaringan komunikasi, jaringan sosial.

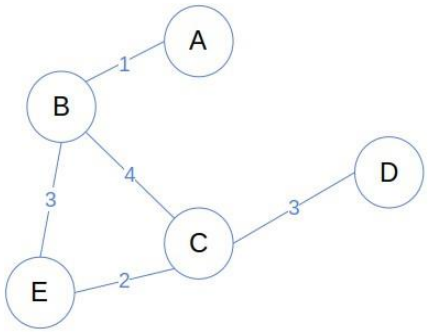
Jika sebuah simpul tidak dapt terhubung dengan simpul lainnya, maka ini disebut dengan isolated verteks.

❖ Graph berdasarkan ada tidaknya label bobot pada nilai koneksinya:

➢ Weighted graph :

Jenis graph yang cabangnya diberi label bobot berupa bilangan numerik, pemberian bobot pada edge biasanya untuk memudahkan algoritma dalam menyelesaikan masalah.

.



Gambar 4. Weighted graph

Pada gambar diatas, implementasinya dimisalkan jika kiata menyelesaikan masalah dalam mencari rute terpendek dari Lokasi A ke Lokasi D, dan dituntut untuk mempertimbangkan kepadatan lalu lintas, Panjang jalan, dll. Untuk menyelesaikan masalah kita dapat mengasosiasikan sebuah edge e dengan bobot w(e) berupa bilangan ril. Nilai bobot ini bisa rlevan untuk masalah yang dihadapi: missal jarak, kepdatan, waktu, biaya, probabilitas, dll.

➢ Unweighted Graph

Jenis graph ini tidak memiliki property bobot pada koneksinya, hanya mempertimbangkan apakah dua node saling terhubung atau tidak.

❖ Ada tiga representasi umum dari stuktur data graph yang dapat diimplementasikan:

1. Adjacency List

Struktur data yang menghubungkan setiap simpul (verteks) dengan daftar simpul yang terhubung (edges). Ini digunakan untuk mengimplementasikan directed graph dan undirected graph.

2. Adjacency Matrix

   Merupakan matrix dimana baris dan kolom mewakili simpul – simpul, dan entri dalam matriks menunjukkan keberadaan sisi antara dua simpul. Waktu akses cepat: O(1), tapi boros memori: O(n2).

3. Edge List

   Merepresentasikan graph dengan hanya mencatat semua tepinya (hubungan antar simpul). Pasangan simpul yang terhubung dapat dengan atau tanpa bobot. Efisien untuk graph yang kecil atau jarang (sparse graph).

❖ Kelebihan sturktur data graph:

- Memudahkan menemukan jalur terpendek dan tetangga dari node.
- Graph digunakan untuk mengimplementasikan algoritma seperti DFS untuk menelusuri kedalam satu arah sejauh mungkin sebelum mundur (eksplorasi) dan BFS untuk menemukan jalur terpendek dalam graph tak berbobot.
- Graph membantu dalam mengatur data.
- Karena strukturnya non-linier, ini membantu dalam memahami maslah yang kompleks dan visualisasinya.

❖ Kekurangan struktur data graph:

- Graph menggunakan banyak pointer yang bisa rumit untuk ditangani.
- Memiliki kompleksitas memori yang besar jika graph direpresentasikan dengan adjacency matrix, maka edge tidak memungkinkan untuk sejajar dan operasi perkalian graph juga sulit untuk dilakukan.

❖ Kegunaan utama dari struktur data graph dalam dunia komputer:

- Merepresentasikan aliran komputasi, dimana graph akan digunakan untuk menggambarkan aliran proses atau komputasi dalam sebuah system.
- Pemodelan grafis, struktur data graph dapat digunakan untuk membuat pemodelan objek dan interaksi diantara mereka.
- Alokasi sumber daya system operasi, graf digunakan untuk alokasi sumber daya seperti memori, prosesor, dan lainnya.
- Rute terpendek pada peta interaktir, menggunakan graf pada Google Maps untuk menemukan rute terpendek antara dua Lokasi.
- Representasikan state-transition dalam system, graf dapat merepresentasikan state dan transisi diantara mereka.
- Pemecahan teka – teki dan permasalahan yang memiliki satu Solusi, seperti pemecahan labirin. Graf digunakan untuk memodelkan permasalahan dengan satu Solusi yang unik.
- Aplikasi Peer to Peer (P2P) dalam jaringan computer, dalam aplikasi P2P graf digunakan untuk mengatur koneksi dan pertukaran data antar node.

Graph digunakan dalam minimum spanning tree(MST) untuk menghubungkan semua titik dengan biaya minimum dalam suatu jaringan, seperti pada perencanaan jaringan Listrik, telekomunikasi, atau distribusi. Algoritma Prim membangun MST dengan memilih tepi terkecil yang menghubungkan dua himpunan simpul terpisah, dengan memperluas MST yang menambahkan simpul menggunakan property pemotongan.

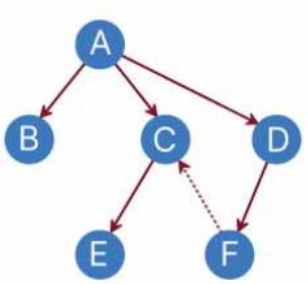
2.2 Graph Traversal

Graph traversal adalah teknik yang digunakan untuk mengunjungi setiap node dalam graf, baik untuk keperluan pencarian data, pemrosesan informasi, maupun analisis struktur jaringan.

❖ Terdapat dua metode Algoritma Traversal Graph yang paling umum digunakan yaitu :

- Breadth-First Search (BFS)

  Pada metode Breadth-First Search, semua node pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi node-node pada level n+1. Pencarian dimulai dari node awal, kemudian berlanjut ke level pertama dari kiri ke kanan, lalu berpindah ke level berikutnya dengan urutan yang sama hingga solusi ditemukan. Proses ini menggunakan struktur data queue untuk menjaga urutan node yang akan dikunjungi berikutnya. Contohnya:



Gambar 5. Breadth-First Search

- Langkah-langkah algoritma traversal Graph menggunakan BFS:
  - Terdapat dua QUEUE yaitu visitQueue untuk menyimpan Node-Node yang sedang dikunjungi dan visitList untuk Node-Node yang sudah dikunjungi.
  - masukan Node Awal yaitu A ke visitQueue.

A							VisitQueue:

							visitList:

- Ambil A dari visitQueue dan masukkan dalam visitList. Cari tetangga dari node A yaitu B, C dan D, masukkan dalam visitQueue.

B	C	D					visitQueue:

visitList

A						

- Ambil B dari visitQueue dan masukkan dalam visitList. Node B tidak memiliki tetangga sehingga tidak ada Node yang dapat dimasukkan dalam visitQueue.

C	D					

visitQueue:

A	B					

visitList:

- Ambil C dari visitQueue masukkan dalam visitList. Cari tetangga dari Node C yaitu E dan F, masukkan dalam visitQueue.

D	E	F				

visitQueue:

A	B	C				

visitList:

- Ambil D dari visitQueue dan masukkan dalam visitList. Cari tetangga dari node D. Tetangga dari node D adalah F tapi sudah masuk visitQueue, sehingga tidak ada node yang dapat dimasukkan dalam visitQueue.

E	F					

visitQueue:

A	B	C	D			

visitList:

- Ambil E dari visitQueue dan masukkan dalam visitList. Node E tidak memiliki tetangga sehingga tidak ada Node yang dapat dimasukkan dalam visitQueue.

F						

visitQueue:

A	B	C	D	E		

visitList:

- Ambil F dari visitQueue dan masukkan dalam visitList. Node F tidak memiliki tetangga sehingga tidak ada Node yang dapat dimasukkan dalam visitQueue.

F						

visitQueue:

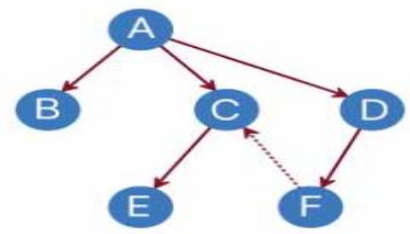
A	B	C	D	E	F	

visitList:

- Karena Queue kosong maka proses traversal selesai.

➢ Depth – First Search (DFS)

Pada Depth First Search, proses pencarian akan dilaksanakan pada semua anaknya sebelum dilakukan pencarian ke node-node yang selevel. Pencarian dimulai dari node akar ke level yang lebih tinggi, Proses ini diulangi terus hingga ditemukaannya solusi. DFS bisa diterapkan menggunakan struktur data stack. Contohnya:



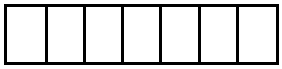
Gambar 6. Depth – First Search [3]

➢ Langkah-langkah algoritma traversal Graph menggunakan DFS:

- Terdapat Stack yaitu visitStack untuk menyimpan Node-Node yang sedang dikunjungi dan Queue untuk visitList untuk Node-Node yang sudah dikunjungi.
- masukan Node Awal yaitu A ke visitStack,

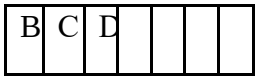
A						

visitQueue:



visitList:

- Ambil A dari visitStack dan masukkan dalam visitList. Cari tetangga dari node A yaitu B, C dan D, masukkan dalam visitStack.

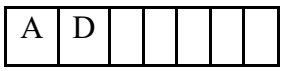


visitQueue:

visitList:

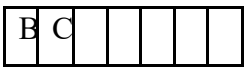
- Ambil D dari visitStack dan masukkan dalam visitList. Cari tetangga dari node D yang belum masuk di visitList maupun visitStack yaitu F, masukkan dalam visitStack

visitQueue:

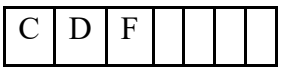


visitList:

- Ambil F dari visitStack dan masukkan dalam visitList. Karena tetangga dari node F yaitu C sudah masuk di visitStack, maka tidak ada yang dimasukkan ke visitStack.

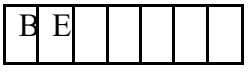


visitQueue:

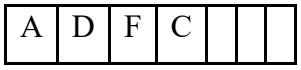


visitList:

- Ambil C dari visitStack dan masukkan dalam visitList. Cari tetangga dari node C yang belum masuk di visitList maupun visitStack yaitu E, masukkan dalam visitStack



visitQueue:



visitList:

- Ambil E dari visitStack dan masukkan dalam visitList. Karena tetangga dari node E yaitu C sudah masuk di visitList, maka tidak ada yang dimasukkan ke visitStack.

B						

visitQueue:

A	D	F	C	E		

VisitList:

- Ambil B dari visitStack dan masukkan dalam visitList. Node B tidak memiliki tetangga sehingga tidak ada Node yang dapat dimasukkan dalam visitStack.

visitQueue:

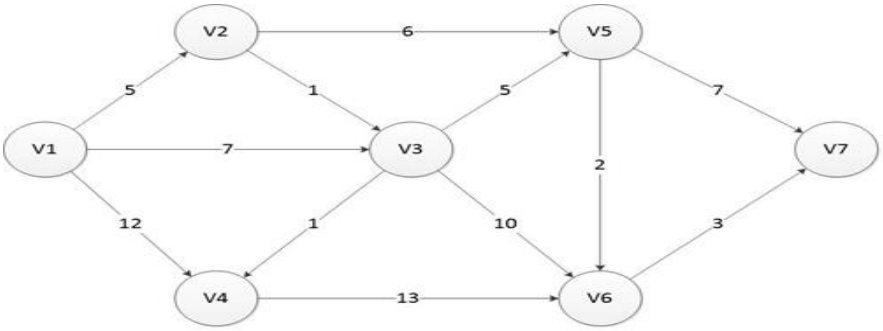
A	D	F	C	E	B	

visitList:

- Karena Stack kosong maka proses traversal selesai.

2.3 Shortest path tree

Shortest path tree adalah pohon jalur pendek yang menggambarkan dari satu titik ke titik lainnya dalam suatu grafik, di shortest path tree ada Breadth-first search (BFS) yang bisa digunakan untuk memecahkan unweighted shortest paths problem.



Gambar 7. Shortest path tree Search

❖ ada 3 variasi Shortest path tree Search yaitu:

- Single pair shortest paths

  Contohnya diberikan titik sumber s dan titik tujuan t, dan mencari berapa lintasan terpendek dari titik s ke titik t.
- Single source shortest paths

  contohnya diberikan titik sumber s, dan mencari apa saja jalur terpendek dari titik s ke semua titik di dalam graph.
- Multiple source shortest paths

  Contohnya diberikan sekumpulan titik sumber s, lalu mencari apa saja lintasan terpendek dari titik manapun di titik s ke semua titik di dalam graph.

❖ Dalam konteks Breadth-first search (BFS) dan unweighted shortest paths, metrik yang digunakan mendefinisikan "shortest" adalah jumlah sisi pada jalur tersebut

- Unweighted shortest paths problem
  - Contoh: Jika diketahui titik sumber s dan titik tujuan t, lintasan dari s ke t manakah yang meminimalkan jumlah sisi, berapa panjang lintasan tersebut, dan sisi apa saja yang menyusunnya?
  - Solusi: base traversal: Breadth-first search (BFS)
  - Modification: menghasilkan pohon jalur terpendek saat bergerak dari sumber titik menuju ke titik tujuan

❖ Dijkstra's algorithm

Dijkstra's algorithm adalah algoritma yang paling terkenal untuk menemukan SPT berbobot dalam suatu graph. Algoritma Dijkstra juga adalah algoritma yang paling terkenal untuk menemukan shortest path tree dalam grafik berbobot Algoritma Dijkstra secara bertahap membangun shortest path tree pada setiap iterasi loop while Algoritma dijkstra's memilih simpul yang belum dikunjungi berikutnya berdasarkan sum total cost of its shortest path**.**

❖ Topological sorting

algorithm digunakan untuk menerapkan algoritma DFS untuk membuat tatanan simpul yang diurutkan secara topologi. Penyortiran topologi

dari directed acyclic graph (DAG) mengurutkan semua simpul dengan cara tertentu untuk memastikan semua prasyarat terpenuhi. Sebuah topological sort berarah g adalah urutan simpul, di mana setiap sisi dalam grafik, asal muncul sebelum tujuan dalam urutan tersebut.

- intuition: a "dependency graph"
  - tepi (u, v) berarti u harus terjadi sebelum v
  - sebuah topological sort ketergantungan memberikan urutan yang menghormati
- Application
  - mengkompilasi beberapa files java
  - alur kerja multi pekerjaan
  - directed acylclic graph
  - grafik berarah tanpa siklus apa pun
  - tepinya mungkin diberi bobot atau mungkin tidak
- Performing topological sort
  - menggunakan BFS, mulai dari titik sudut tanpa sisi masuk
  - menggunakan BFS, dimulai dari semua simpul tanpa tepi yang masuk
- Algoritma yang paling umum untuk menemukan pengurutan topologi
  - DFS

    Depth-first search
  - DFS postorder

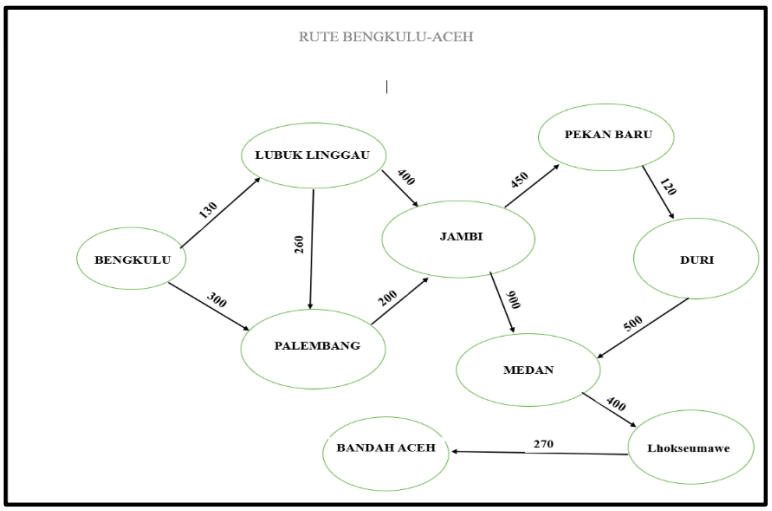
    Cara khusus untuk menjalankan penelusuran mendalam. DFS bersifat rekursif, jadi kita dapat memproses setiap nilai sebelum atau setelah pemanggilan rekursif ke semua tetangga. Preorder memproses nilai sebelum pemanggilan rekursif sementara postorder memproses nilai setelah pemanggilan rekursif.
  - Reverse DFS postorder

    The DFS postorder list, tetapi dalam urutan mundur (terbalik).
- Dalam DFS postorder kita menambahkan simpul-simpul berikut:

- Titik sudut pertama yang ditambahkan ke dalam resultdaftar itu seperti daun di pohon: ia tidak memiliki tetangga yang keluar sehingga ia bukan merupakan “prasyarat” bagi titik sudut lainnya.
- Titik sudut kedua yang ditambahkan pada resultdaftar menunjuk ke titik sudut pertama.
- Titik puncak terakhir yang ditambahkan ke resultdaftar adalah titik puncak awal.

Jadi, urutan pos DFS terbalik memiliki titik puncak pertama ("daun") di akhir daftar, resulttempat titik puncak tersebut berada dalam pengurutan topologi. Di kelas, kita akan mempelajari lebih lanjut tentang signifikansi pendekatan alternatif ini dan apa artinya jika dibandingkan dengan algoritma Dijkstra.

2.4 Studi Kasus



Gambar 8. Graph Rute Bengkulu-Bandah Aceh

1. Deskripsi Kasus

Seorang traveler berencana melakukan perjalanan darat dari Bengkulu menuju Banda Aceh. Karena jarak yang ditempuh cukup jauh dan melibatkan banyak kota besar di Pulau Sumatera, diperlukan strategi perencanaan rute yang tepat untuk menghemat waktu, biaya, serta energi selama perjalanan.

Untuk merancang jalur perjalanan yang paling efisien, digunakan pendekatan graph berbobot, di mana:

1. Simpul (node) merepresentasikan kota-kota yang dilalui,
2. Sisi (edge) mewakili koneksi jalan antar kota,
3. Dan bobot pada setiap sisi menunjukkan jarak tempuh antar kota.

Dalam analisis ini, dua metode algoritmik diterapkan:

1. Traversal Graph (BFS dan DFS) untuk mengeksplorasi semua kemungkinan jalur antar kota.
2. Algoritma Dijkstra untuk menentukan jalur dengan total jarak tempuh terpendek.

Melalui traversal graph, seluruh struktur rute dapat dipetakan. Kemudian, Dijkstra digunakan untuk memilih rute yang paling optimal dari segi jarak.

2. Bobot/Jarak Tempuh Antar Kota

Dalam konteks studi kasus perjalanan dari Bengkulu ke Banda Aceh, bobot pada graph merupakan representasi dari jarak antar kota dalam satuan kilometer (km)

```
{"Bengkulu", {{"Lubuklinggau", 130}, {"Palembang", 300}}},
{"Lubuklinggau", {{"Palembang", 260}, {"Jambi", 400}}},
{"Palembang", {{"Jambi", 200}}},
{"Jambi", {{"Pekanbaru", 450}, {"Medan", 900}}},
{"Pekanbaru", {{"Duri", 120}}},
{"Duri", {{"Medan", 500}}},
{"Medan", {{"Lhokseumawe", 400}}},
{"Lhokseumawe", {{"Banda Aceh", 270}}},
{"Banda Aceh", {}}
```

Gambar 9. Bobot/Jarak tempuh

3. Permasalahan Yang Di Hadapi

a) Terdapat berbagai jalur alternatif dengan bobot jarak yang berbeda.
b) Diperlukan penelusuran graph untuk memahami semua jalur potensial (menggunakan Traversal).

c) Setelah itu, perlu dilakukan seleksi jalur terpendek (menggunakan Dijkstra).

4. Penyelesaian Masalah

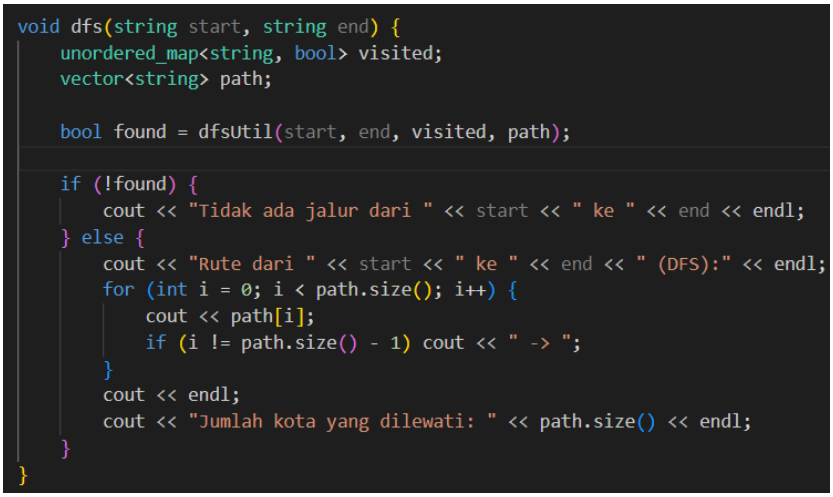
A. Traversal Graph

- Breadth-First Search (BFS) digunakan untuk menelusuri graph berdasarkan tingkat keterhubungan terdekat antar kota.

```
void bfs(string start, string end) {
    unordered_map<string, bool> visited;
    unordered_map<string, string> previous;

    queue<string> q;
    q.push(start);
    visited[start] = true;

    bool found = false;

    while (!q.empty()) {
        string current = q.front();
        q.pop();

        if (current == end) {
            found = true;
            break;
        }

        for (auto neighbor : graph[current]) {
            string nextNode = neighbor.first;
            if (!visited[nextNode]) {
                visited[nextNode] = true;
                previous[nextNode] = current;
                q.push(nextNode);
            }
        }
    }
```

Gambar 10. Breadth-First-Search (BFS)

- Depth-First Search (DFS) digunakan untuk menelusuri jalur hingga sejauh mungkin sebelum kembali ke simpul sebelumnya.



```
void dfs(string start, string end) {
    unordered_map<string, bool> visited;
    vector<string> path;

    bool found = dfsUtil(start, end, visited, path);

    if (!found) {
        cout << "Tidak ada jalur dari " << start << " ke " << end << endl;
    } else {
        cout << "Rute dari " << start << " ke " << end << " (DFS):" << endl;
        for (int i = 0; i < path.size(); i++) {
            cout << path[i];
            if (i != path.size() - 1) cout << " -> ";
        }
        cout << endl;
        cout << "Jumlah kota yang dilewati: " << path.size() << endl;
    }
}
```

Gambar 11. Depth-First Search (DFS)

B. Algoritma Dijkstra

- Setelah seluruh jalur teridentifikasi melalui traversal, Algoritma Dijkstra diterapkan untuk menghitung rute dengan bobot total (jarak) paling kecil dari Bengkulu ke Banda Aceh.

```
void dijkstra(string start, string end) {

    unordered_map<string, int> distances;
    unordered_map<string, string> previous;

    for (const auto& node : graph) {
        distances[node.first] = INT_MAX;
    }
    distances[start] = 0;

    priority_queue<NodeDistance, vector<NodeDistance>, greater<NodeDistance>> pq;
    pq.push({0, start});

    while (!pq.empty()) {
        int currentDistance = pq.top().first;
        string currentNode = pq.top().second;
        pq.pop();

        if (currentNode == end) {
            break;
        }

        for (auto neighbor : graph[currentNode]) {
            string nextNode = neighbor.first;
            int weight = neighbor.second;
            int newDistance = currentDistance + weight;

            if (newDistance < distances[nextNode]) {
                distances[nextNode] = newDistance;
                previous[nextNode] = currentNode;
                pq.push({newDistance, nextNode});
            }
        }
    }
```

Gambar 12. Djikstra Algorithm

5. **Kaitan Traversal dan Dijkstra dalam Studi Kasus**
   a) Traversal (DFS/BFS) berguna untuk menjelajahi semua kota dan memahami struktur graph.
   b) Dijkstra berguna untuk mengoptimalkan jalur berdasarkan bobot jarak.

Pada studi kasus perjalanan Bengkulu - Banda Aceh:

   a) BFS/DFS digunakan untuk mengecek semua kemungkinan jalur.
   b) Dijkstra digunakan untuk mencari jalur dengan jarak total minimum.

6. **Hasil Analisis**
   1. Berdasarkan penerapan algoritma Dijkstra, rute tercepat yang diperoleh adalah:

```
PS D:\Tugas Graph SDA> cd "d:\Tugas Graph SDA\GRAPH\Shortest path tree\" ; if (
Jarak terpendek dari Bengkulu ke Banda Aceh adalah 2070 km.
Rutenya: Bengkulu -> Palembang -> Jambi -> Medan -> Lhokseumawe -> Banda Aceh
PS D:\Tugas Graph SDA\GRAPH\Shortest path tree>
```

Gambar 13. Output Djikstra Algorithm

2. Berdasarkan penerapan algoritma Travelsals (DFS,BFS), rute yang diperoleh adalah:

```
PS D:\Tugas Graph SDA> cd "d:\Tugas Graph SDA\GRAPH\Travelsals\" ; if ($?) { g++ bfs.cpp -o bfs } ; if ($?) { .\bfs
Rute dari Bengkulu ke Banda Aceh (BFS)
Bengkulu -> Lubuklinggau -> Jambi -> Medan -> Lhokseumawe -> Banda Aceh
Jumlah kota yang dilewati: 6
PS D:\Tugas Graph SDA\GRAPH\Travelsals> cd "d:\Tugas Graph SDA\GRAPH\Travelsals\" ; if ($?) { g++ dfs.cpp -o dfs }
Rute dari Bengkulu ke Banda Aceh (DFS):
Bengkulu -> Lubuklinggau -> Palembang -> Jambi -> Pekanbaru -> Duri -> Medan -> Lhokseumawe -> Banda Aceh
Jumlah kota yang dilewati: 9
PS D:\Tugas Graph SDA\GRAPH\Travelsals>
```

Gambar 14. Output Travelsals

7. **Kesimpulan Studi Kasus**

Penerapan traversal graph terbukti efektif dalam memetakan semua jalur yang tersedia, sedangkan algoritma Dijkstra membantu dalam menentukan jalur tercepat. Melalui analisis ini, perencanaan perjalanan dari Bengkulu ke Banda Aceh dapat dilakukan dengan lebih baik, mengoptimalkan waktu tempuh, mengurangi biaya perjalanan, dan meningkatkan kenyamanan.

BAB III
PENUTUP

3.1 Kesimpulan

Graph merupakan struktur data non-linear yang mengandung kumpulan simpul (node) berhingga untuk menyimpan data dan antara dua simpul terdapat hubungan (edge) saling berkaitan, antar elemennya many-to-one, one-to-many, dan many-to-many, dengan melalui undirected, directed, dan weighted, unweighted graf. Digunakannya traversal untuk mengunjungi setiap node dalam graf pada pencarian data, pemrosesan informasi, analisis struktur jaringan dengan Breadth-First Search (BFS) dan Depth-First Search (DFS). Sedangkan disjoint sets mengelolah kumpulan simpul saat memecahkan masalah dalam graf seperti Minimum Spanning Tree (MST). Shortest Path Tree dibangun melalui traversal dimana pohon jalur terpendek menggambarkan dari satu titik ke titik lainnya dalam suatu graf, menggunakan algoritma seperti Breadth-First Search (BFS) untuk memecahkan shortest paths problem, Topological sorting untuk menerapkan algoritma DFS dalam membuat tatanan simpul yang diurutkan secara topological, dan Dijkstra's algorithm untuk menemukan SPT berbobot dalam graph. Graph sangat berguna dalam berbagai aplikasi komputasi.

DAFTAR PUSTAKA

Alfionita,Shinta. (2015). *graph*_1. https://shintaalfionita.files.wordpress.com/2015/06/c8927-graph_1.jpg,

Ardana, Dwi, & Saputra, Ragil. (2023). *Penerapan Algoritma Dijkstra pada Aplikasi Pencarian Rute Bus Trans Semarang.* https://ilkom.unnes.ac.id/snik/prosiding/2016/45.%20SNIK_334_Algoritma%20Dijkstra. Pdf

Geeksforgeeks. (2023). *How to find Shortest Paths from Source to all Vertices using Dijkstra's Algorithm.* https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy algo-7/,

Harahap, M. K., Khairina, Nurul. (2017). Pencarian Jalur Terpendek dengan Algoritma Dijkstra. *Jurnal & Penelitian Teknik Informatika*, 2(2).

JSTR Production. (2020). *Konsep Lintasan Terpendek Algoritma Dijkstra.* [Video]. Youtube. https://www.youtube.com/watch?v=qIzCYrE8570,

Munir, Rinaldi. (2023). Graf (Bagian 1). https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19 Graf-Bagian1-2023.pdf

Robert Sedgewick dan Kevin Wayne. 2020. *Grafik dan Digraf*. https://courses.cs.washington.edu/courses/cse373/23wi/lessons/graphs/