**MAKALAH STRUKTUR DATA DAN ALGORITMA**

“Application (Dynamic Programming, Minimum Spanning Tree, Disjoint Sets) “



**Disusun Oleh :**

Kelompok 6

1. Aulia Afdhal Hafidzah (G1A024029)
2. Angelita Rahmatun Annisa (G1A024067)
3. Yudhistira Rizky Ramadhan (G1A024079)
4. M. Arrfiqi Yoga Dinanti (G1A024085)
5. Alfi Wiradinata (G1A0240101)
6. Khairi Makruf Fathorikan (G1A0240111)

**Dosen Pengampu :**

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

**PROGRAM STUDI INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS BENGKULU**

**2025**

**KATA PENGANTAR**

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, makalah yang berjudul "Application (Dynamic Programming, Minimum Spanning Tree, Disjoint Sets)" ini dapat diselesaikan dengan baik. Makalah ini disusun sebagai salah satu tugas dalam mata kuliah Struktur Data dan Algoritma dan bertujuan untuk memperluas pemahaman mengenai Dynamic Programming, Minimum Spanning Tree, dan Disjoint Sets. Dalam penyusunan makalah ini, penulis berusaha menyajikan pembahasan secara sistematis dan akademis mengenai Dynamic Programming, Minimum Spanning Tree, dan Disjoint Sets.

Penulis menyadari bahwa makalah ini masih jauh dari kata sempurna. Oleh karena itu, segala kritik dan saran yang bersifat membangun sangat penulis harapkan demi perbaikan di masa yang akan datang. Akhir kata, penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan dan bantuan selama proses penyusunan makalah ini. Semoga makalah ini dapat memberikan manfaat dan menambah wawasan bagi pembaca.

Bengkulu, April 2025

Penulis

**DAFTAR ISI**

KATA PENGANTAR ii

DAFTAR ISI iii

BAB I PENDAHULUAN

* 1. Latar Belakang 1
  2. Rumusan Masalah 1

BAB II PEMBAHASAN

2.1 Dynamic Programming 2

2.2 Minimum Spanning Tree (MST) 6

2.3 Disjoint Sets 13

BAB III PENUTUP

3.1 Kesimpulan 20

DAFTAR PUSTAKA

**BAB I**

**PENDAHULUAN**

1. **Latar Belakang**

Dengan teknologi informasi yang berkembang pesat, situasi saat ini adalah kemampuan untuk mengelola dan mengoptimalkan data, kunci untuk pengembangan berbagai aplikasi komputer. Struktur dan algoritma data adalah dasar -dasar penting untuk mendukung efisiensi dan efektivitas dalam menyelesaikan masalah komputer. Dynamic Programming, Minimum Spanning Tree (MST), dan Disjoint Sets, merupakan tiga konsep utama dalam struktur data dan algoritma yang sering dipakai untuk memecah berbagai solusi praktis.

Memecah masalah rumit ke bagian-bagian kecil agar didapat solusi optimal merupakan cara kerja Dynamic Programming. Minimum spanning tree ialah subgraf dengan jumlah bobot sisi paling kecil. Disjoint Set didefinisikan sebagai struktur yang melacak sekumpulan elemen yang dipartisi menjadi sejumlah subset terpisah (tidak tumpang tindih).

1. **Rumusan Masalah**

* Apa itu Dynamic Programming?
* Apa itu Minimum Spanning Tree?
* Apa itu Disjoint Sets?

**BAB II**

**PEMBAHASAN**

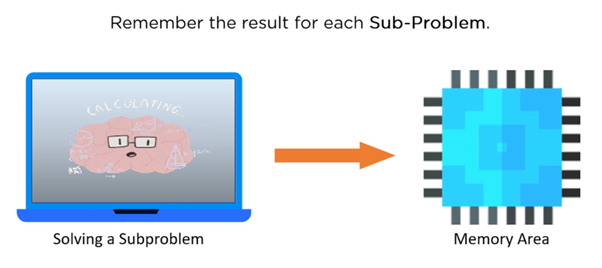
1. **Dynamic Programming**
2. **Pengertian Dynamic Programming**

Memecah masalah rumit ke bagian-bagian kecil agar didapat solusi optimal merupakan cara kerja Dynamic Programming. Hal ini sering kali melibatkan pencarian nilai tertinggi atau terendah untuk kueri tertentu.

Pengoperasian Dynamic Programming sangat mirip dengan rekursi dengan memoisasi solusi antara, yang juga disebut memoisasi. Suatu masalah yang besar di ubah menjadi beberapa bagian tugas yang lebih kecil untuk dikerjakan adalah cara kerja Algoritma rekursif. Algoritme dinamis membagi kesulitan menjadi beberapa bagian dan menghitungnya satu per satu, membangun solusi langkah demi langkah. Oleh karena itu, algoritme dinamis dapat dianggap sebagai rekursi, yang bekerja dari bawah ke atas.

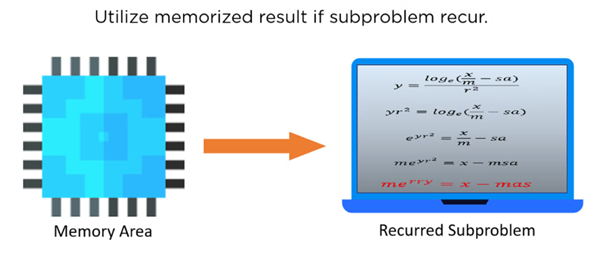
Langkah-langkah dibawah adalah untuk menemukan solusi dari pemograman ini:

* Langkah 1: Memecah masalah yang lebih luas atau kompleks menjadi beberapa submasalah yang lebih kecil.
* Langkah 2: Menghitung solusi dari setaip bagian kecil masalah.
* Langkah 3: Setelah menghitung hasilnya, ia akan mengingat solusi untuk setiap submasalah (Menghafal).



Gambar 1. Langkah 3 Dynamic Programming

* Langkah 4: Apabila bagian masalah yang lebih kecil berulang, gunakan lagi hasil yang tersimpan.



Gambar 2. Langkah 4 Dynamic Programming

* Langkah 5: Integrasikan solusi submasalah untuk merumuskan solusi masalah yang lebih luas.

Pemanfaatan memori untuk menyimpan hasil merupakan langkah tepat dalam pendekatan pembuatan solusi ini. Oleh karena itu, kompleksitas ruang akan meningkat. Namun, karena pemanfaatan ruang yang sama, kompleksitas waktu akan berkurang secara signifikan.

1. **Konsep Dasar Dynamic Programming**

* **Overlapping Subproblems**

Adanya sub masalah yang sama muncul berulang kali ketika dalam proses penyelesaian masalah besar. Dengan Dynamic Programming menyimpan hasil bagian masalah tersebut dan menggunakannya kembali saat dibutuhkan, sehingga mengurangi kompleksitas waktu secara drastis.

* **Optimal Substructure**

Untuk mencari solusi optimal guna memecahkan suatu permasalahan besar dapat diterbentuk dari menggabungkan solusi-solusi optimal penyelesaian sub masalah yang lebih kecil. Jika solusi dari bagian sub masalah diketahui dan optimal, maka solusi akhir yang dibentuk dari kombinasi sub masalah tersebut juga akan optimal.

1. **Teknik Penyelesaian**

* **Top-Down**

Pendekatan top-down mengikuti teknik menghafal. Pendekatan ini terdiri dari dua peristiwa berbeda yaitu, rekursi dan caching. Rekursi ialah proses komputasi dengan memanggil fungsi berulang kali, sedangkan caching yakni penyimpanan hasil antara.

* **Bottom-up**

Pendekatan ini menggunakan teknik tabulasi untuk mengimplementasikan solusi pemrograman dinamis. Jika pada pendekatan sebelumnya digunakan proses rekursi, lain halnya pada pendekatan kali ini, dimana tidak digunakannya rekursi melainkan dengan iterasi sebagai pengganti rekursi. Kesalahan stack overflow atau overhead prosedur rekursif tidak terdapat pada bagian ini.

1. **Pengaplikasian Dynamic Programming**
2. **Fibonacci Sequence**

Dimana masalah utamanya ialah menghitung angka ke-n dalam suatu deret yang dikhawatirkan akan memakan waktu banyak dikarenakan proses hitung berulang. Disinilah Dynamic Programming dibutuhkan, ia memiliki bertugas untuk menyimpan hasil deret sebelum-sebelumnya agar tidak perlu melalui jalan yang sama lagi dan untuk efisiensi waktu juga.

1. **Knapsack Problem (0/1 Knapsack)**

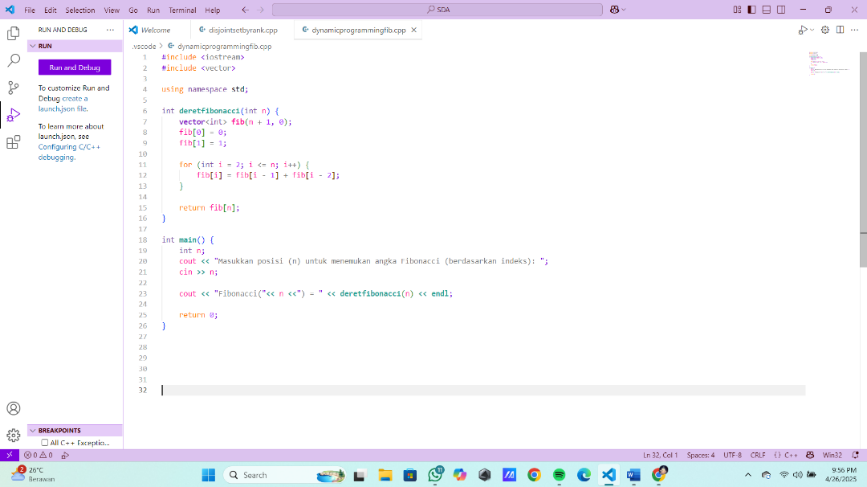
Permasalahan yang dihadapi berupa memilih barang dari barang-barang dengan variasi nilai dan berat untuk mencegah nilai maksimal melebihi kapasitas tas. Dynamic Programming berfungsi untuk menyimpan hasil sub masalah kombinasi barang dan kapasitas yang nantinya akan dicari solusi yang optimal

1. **Shortest Path (contoh: Floyd-Warshall)**

Permasalahnnya adalah dibutuhkan untuk mencari jaln tercepat diantara banyaknya pasangan simpul dalam suatu graf. Dynamic Programming disini dibuthkan untuk menyimpan jarak minimum suatu simpul, lalu update berdasarkan kemungkinan jalur perantara.

1. **Penerapan Kode**

**Printscreen Code:**

****

Gambar 3. Printscreen Code Dynamic Programming

**Source Code:**

#include <iostream>

#include <vector>

using namespace std;

int deretfibonacci(int n) {

vector<int> fib(n + 1, 0);

fib[0] = 0;

fib[1] = 1;

for (int i = 2; i <= n; i++) {

fib[i] = fib[i - 1] + fib[i - 2];

}

return fib[n];

}

int main() {

int n;

cout << "Masukkan posisi (n) untuk menemukan angka Fibonacci (berdasarkan indeks): ";

cin >> n;

cout << "Fibonacci("<< n <<") = " << deretfibonacci(n) << endl;

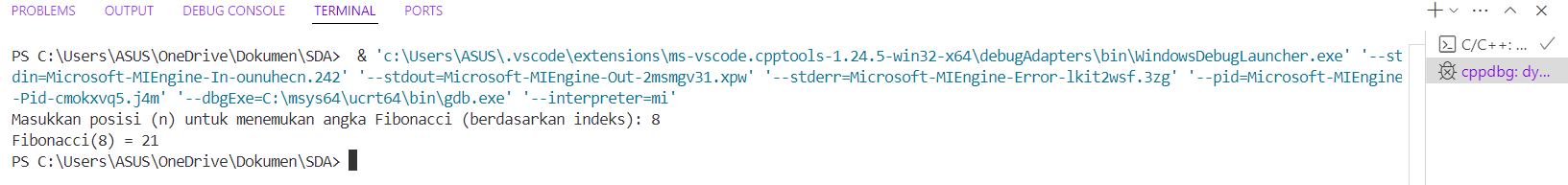
return 0;

}

**Penjelasan Source Code:**

Pada header files terdapat kode **#include <iostream>** dan **#include <vector>** yang masing-masing berguna untuk input dan output serta untuk menggunakan struktur data vector. Fungsi pada **deretfibonacci** ialah untuk menerima satu parameter **n** yang adalah posisi dalam deret tersebut. Membuat vektor **fib** untuk menyimpan nilai fibonacci dari 0 hingga ke-n. Kemudian untuk menginisialisasi dua nilai pertama dari deret fibonacci sebagai 0 dan 1. Loop dibutuhkan dalam hal ini untuk digunakan dalam fungsi menghitung nilai fibonacci selanjutnya dengan menjumlahkan dua nilai sebelumnya dan akhirnya mengembalikan nilai fibonacci pada posisi **n**. Pada fungsi **main** dimintanya input dari user yang berupa nilai indeks dan dari 0 kemudian memanggil fungsi **deretfibonacci** untuk ditampilkan hasil inputan tersebut di bagian output.

**Printscreen Output:**

****

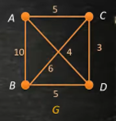
Gambar 4. Printscreen Output Dynamic Programming

**Penjelasan Output:**

Dalam program ini perhitungan dimulai dari angka 0 dan 1, sehingga angka ketiga adalah 1 dan program menjumlahkan seluruh angka sisanya dengan loop sampai posisi 8 itu sendiri, dimana posisi 8 didapat dengan menjumlahkan posisi ke-7 dan ke-6 sehingga didapat 21.

1. **Minimum Spanning Tree (MST)**
2. **Pengertian Minimum Spanning Tree**

Sebuah graf dengan memutuskan beberapa sisi yang membentuk sirkuit tanpa mengurangi simpul atau node ialah pengertian dari spanning tree. Dengan ciri N nodes dan N-1 edges, spanning tree haruslah tersambung satu sama lain tana membentuk sirkuit. Sirkuit dalam konteks graf adalah sebuah lintasan yang dimulai dan berakhir pada node/simpul yang sama. Sehingga, minimum spanning tree ialah subgraf/pohon dengan jumlah bobot sisi (weighted edge) paling kecil.



Gambar 5. Graf 1

Keterangan Gambar:

G = Graf awal dan masih mengandung sirkuit

T = Pohon hasil/subgraf dari G yang menjadi pohon (tidak ada sirkuit)

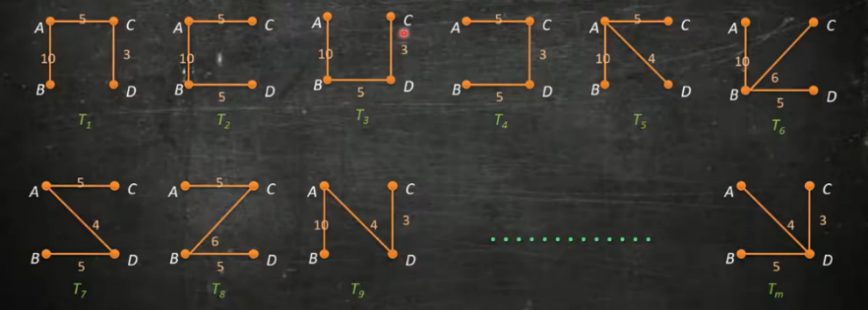
V = Vertex (simpul atau node) yang berupa titik temu dalam graf

E = Edge (sisi) penghubung antar simpul

Weighted Edge = Bobot sisi memiliki nilai atau bobot

Root = Akar atau titik acuan awal

Misalkan G = (V, E) graf tak berarah bukan pohon dan terdapat beberapa sirkuit. G dapat diubah menjadi pohon T = (V, E) dengan memutus sirkuit-sirkuit yang ada.



Gambar 6. Kumpulan Spanning Tree dari Graf 1

Dari kumpulan gambaran diatas merupakan kumpulan spanning tree, dimana terdapat N nodes dan N-1 edges pada masing-masing gambar. Untuk menentukan yang mana merupakan MST nya, diperlukan untuk menjumlahkan hasil tiap spanning tree tersebut untuk mencari jumlah paling kecil.

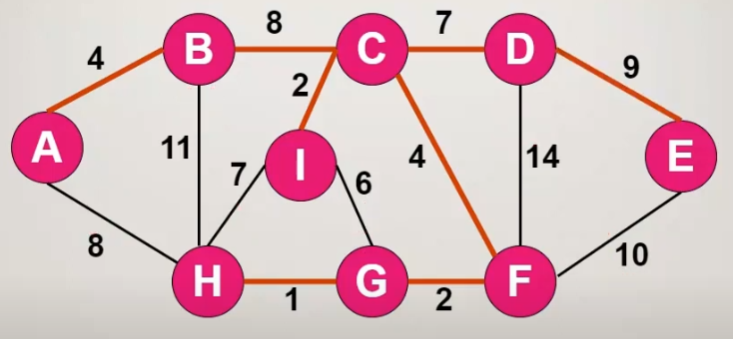
1. **Algoritma Minimum Spanning Tree**

Biasanya dalam menentukan minimum spanning tree ada dua algoritma yang biasa digunakan, yakni Algoritma Prim dan Algoritma Kruskal.

* **Algoritma Prim**

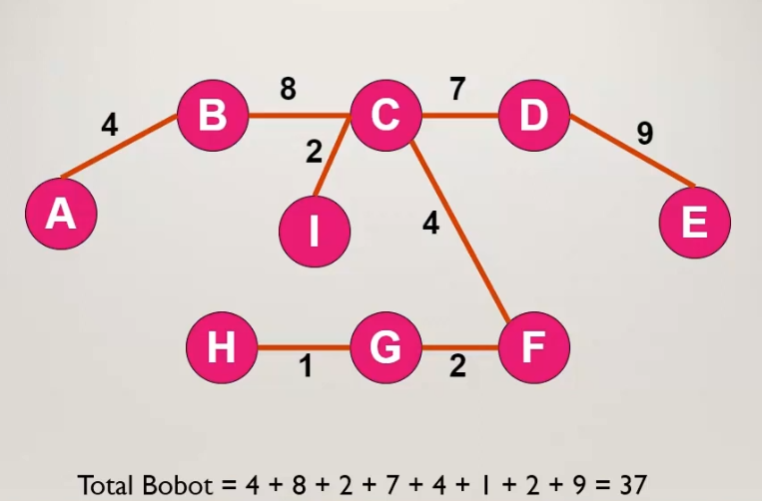
Untuk menyelesaikan masalah MST dapat digunakan Algoritma Prim, algoritma yang paing sederhana. Langkah awal yang dilakukan adalah dengan cara memilih salah satu titik secara random atau secara sembarang untuk dijadikan sebagai root atau akar atau titik acuan awal. Setelah titik root terpilih, maka pilih titik selanjutnya yang bertetangga dengan root tersebut dengan nilai bobot terkecil. Titik bobot terkecil yang bertetangga dapat dipilih asalkan tetap dengan syarat tidak membentuk sirkuit apabila dipilih. Langkah-langkah tersebut terus di lakukan sampai menemukan penyelesaian MST tanpa membentuk sirkuit.

Contoh:



Gambar 7. Penyelesaian Graf 2 dengan Algoritma Prim

Disini dapat dipilih titik sembarang yang pada kasus ini dipihnya titik A, dapat dilihat bobot sisi yang bertetangga dengan A adalah 4 dan 8 dan dilihat dari nilai yang paling kecil, dipilihlah sisi AB dengan bobot 4. Selanjutnya, disini kita dapat melihat jalur yang dapat diambil adalah AH, BC, ataupun BH dengan bobot masing-masing 8, 8, 11. Karena terdapat dua nilai bobot minimum yang sama, kita pilih secara sembarang yakni titik BC. Lalu, sisi Ci dan Cf dapat dipilih dan karena kedua nilai ini lebih minimum dibanding 2 nilai sebelumnya, jelas dua sisinya akan diambil asalah tidak menyebabkan sirkuit. Setelah CI dan CF ditandai, dapat dilihat dari titik F bahwa memungkinkan untuk memilih titik FG yang berbobot 2 dan dilanjut ke GH dengan bobot 1. Setelah GH terpilih, dapat terlihat bahwa jalur AH, BH, dan IG tidak dapat dipilih lagi karena akan membentuk sirkuit. Setelah itu, dilihat dari titiknya, dapat dipilih titik CD, FD, maupun FE dengan masing-masing nilai 7, 14, dan 10, dengan titik CD terpilih karena yang paling minimum diantaranya. Karena akan terbentuknya sirkuit apabila sisi FD dipilih setelah sisi CD terpilih, maka langkah terakhir ialah melihat sisa antara sisi DE dan FE yang dengan mudah diselesaikan, dilihat dari bobot nilai DE yang lebih kecil.



Gambar 8. MST dari Graf 2 dengan Metode Prim

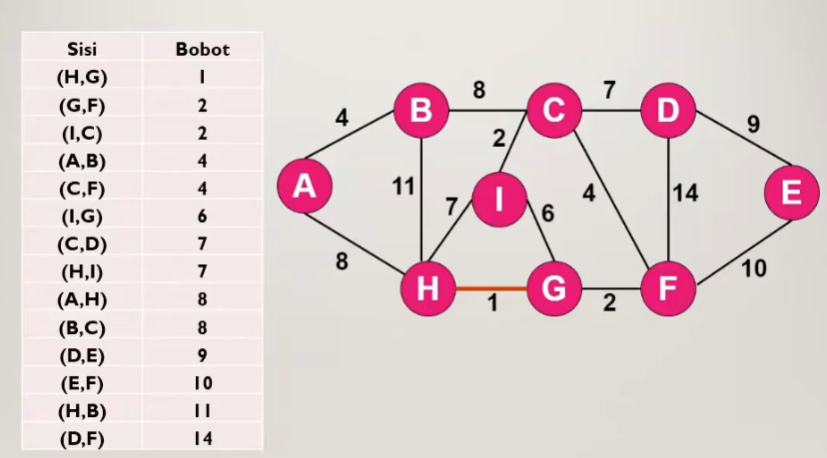
Setelah garis yang tidak digunakan dihapus, dapat kita hitung jumlah MST dari contoh diatas ialah 37.

* **Algoritma Kruskal**

Dengan mengurutkan sisi-sisi yang ada, dari terkecil ke terbesar, merupakan langkah awal algoritma ini. Penentuan dapat dipilih berdasarkan urutan tersebut dengan sisi yang memiliki bobot sama dapat dipilih acak ataupun bersama, selagi tidak membentuk sirkuit apabila dipilih. Langkah-langkah tersebut terus di lakukan sampai menemukan penyelesaian MST tanpa membentuk sirkuit.

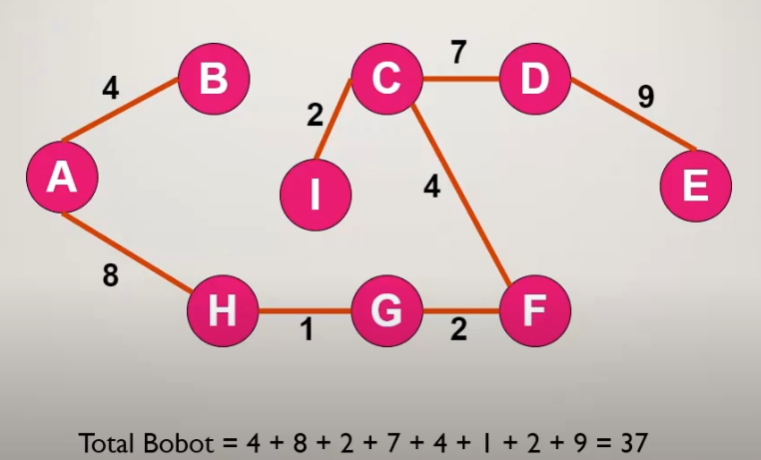
Contoh:

Pada Algoritma Kruskal diharuskan mendata semua nilai bobot pada graf dan selanjutnya diurutkan data tersebut dari bobot nilai paling kecil ke nilai bobot yang paling besar seperti gambar dibawah.



Gambar 9. Penyelesaian Graf 2 dengan Algoritma Kruskal

Ditandai sisi HG karena berupa sisi dengan bobot paling minimum adalah langkah awal. Selanjutnya, sisi IC dan GF dipilih karena tidak akan membentuk sirkuit dilanjutkan sisi AB dan CF yang juga memiliki nilai bobot sama dapat dipilih keduanya karena tidak akan membentuk sirkuit didalamnya. Terbentuknya sirkuit jika IG dipilih, adalah alasan kenapa tidak terpilihnya sisi ini. Selanjutnya tidak dipilihnya sisi HI yang memiliki bobot sama dengan CD karena akan membentuk sirkuit, sehingga cukup sisi CD yang diambilKasus yang sama dengan sisi AH dan BC, dimana AH yang dipiih dan BC tidak. Terakhir, diambilnya sisi DE yang juga mengakhiri operasi ini, karena semua simpul sudah terhubung.



Gambar 10. MST dari Graf 2 dengan Metode Kruskal

Ini adalah hasil dari graf setelah menghapus garis yang tidak perlu dan dengan jumlah 37 sebagai nilai MST setelah melalui sisi bobot nilai paling kecil.

1. **Aplikasi Nyata Minimum Spanning Tree**

* **Desain Jaringan (Network Design)**

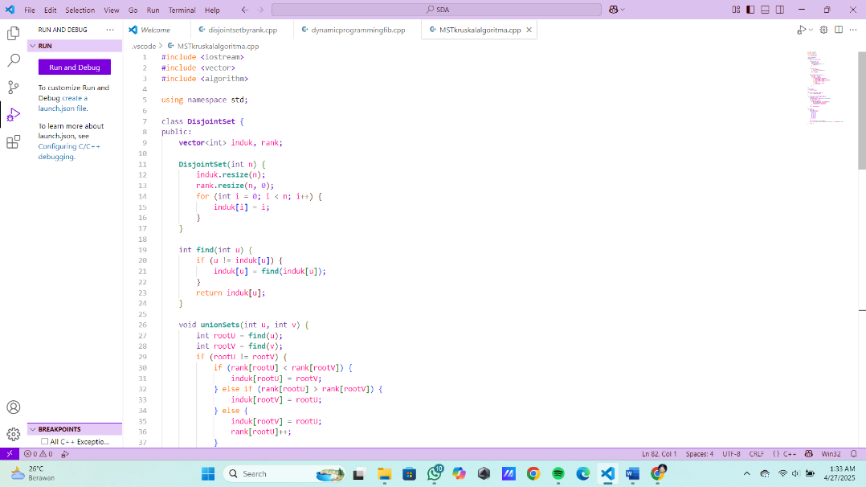
Pada saat ini, sangat jarang kita tidak dikelilingi oleh jaringan yang selalu memerlukan akses. Diantara banyaknya rekomendasi layanan bisnis yang berbasis jaringan seperti telefon, listrik, hidrolik, kabel parabola, komputer, dan jalan raya, kita akan mencari dan memilih tarif yang paling minim. Disinilah dibutuhkannya minimum spanning tree.

* **Algoritma Aproksimasi (Approximation Algorithm)**

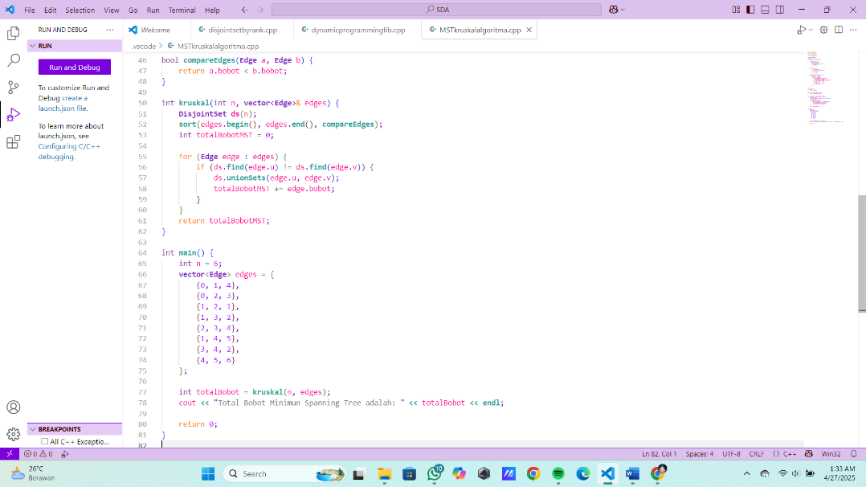
Masalah Travelling Salesman merupakan masalah yang sulit di mana kita diberi daftar kota dan jarak antar masing-masing kota. Diperlukannya untuk menemukan rute terpendek untuk mengunjungi setiap kota tepat satu kembali dan kembali ke kota asal.

1. **Penerapan Kode**

**Printscreen Source Code:**

****

Gambar 11. Printscreen Source Code 1 MST

****

Gambar 12. Printscreen Source Code 2 MST

**Source Code:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

class DisjointSet {

public:

vector<int> induk, rank;

DisjointSet(int n) {

induk.resize(n);

rank.resize(n, 0);

for (int i = 0; i < n; i++) {

induk[i] = i;

}

}

int find(int u) {

if (u != induk[u]) {

induk[u] = find(induk[u]);

}

return induk[u];

}

void unionSets(int u, int v) {

int rootU = find(u);

int rootV = find(v);

if (rootU != rootV) {

if (rank[rootU] < rank[rootV]) {

induk[rootU] = rootV;

} else if (rank[rootU] > rank[rootV]) {

induk[rootV] = rootU;

} else {

induk[rootV] = rootU;

rank[rootU]++;

}

}

}

};

struct Edge {

int u, v, bobot;

};

bool compareEdges(Edge a, Edge b) {

return a.bobot < b.bobot;

}

int kruskal(int n, vector<Edge>& edges) {

DisjointSet ds(n);

sort(edges.begin(), edges.end(), compareEdges);

int totalBobotMST = 0;

for (Edge edge : edges) {

if (ds.find(edge.u) != ds.find(edge.v)) {

ds.unionSets(edge.u, edge.v);

totalBobotMST += edge.bobot;

}

}

return totalBobotMST;

}

int main() {

int n = 6;

vector<Edge> edges = {

{0, 1, 4},

{0, 2, 3},

{1, 2, 1},

{1, 3, 2},

{2, 3, 4},

{1, 4, 5},

{3, 4, 2},

{4, 5, 6}

};

int totalBobot = kruskal(n, edges);

cout << "Total Bobot Minimun Spanning Tree adalah: " << totalBobot << endl;

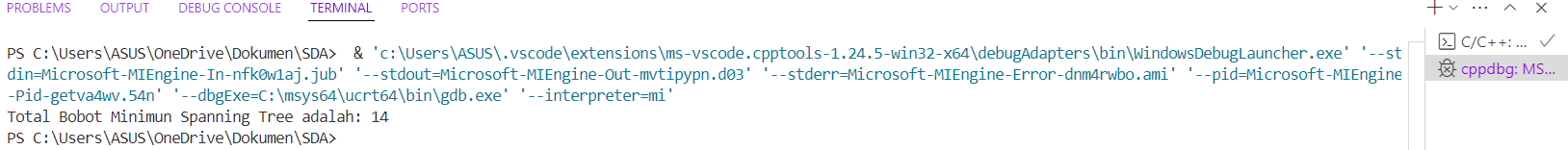
return 0;

}

**Penjelasan Source Code:**

Disjoint Sets digunakan di algoritma ini untuk mengelola dan menggabungkan himpunan yang terisah. Kelas **DisjointSet** yang memiliki dua kelas **induk** dan **rank**, yang berguna untuk menyimpan informasi tentang induk dari setiap simpul dan rank setiap himpunan. Fungsi **find** berguna untuk menemukan akar dari himpunan yang berisi simpul lain, dengan path compression. Fungsi **unionSets** menggabungkan dua himpunan berdasarkan rank, sehingga pohon tetap seimbang. Fungsi **kruskal** dan **find** bekerja sama, dengan **kruskal** mengurutkan bobotnya dan tepi atau edge yang memiliki bobot terkecil akan ditambahkan ke Minimun Spanning Tree, lalu **find** akan memeriksa terbentuknya sirkuit atau tidak. Hasil akhir adalah total bobot MST, seperti contoh yang diberikan, dengan 6 simpul dan beberapa tepi dengan variasi nilai bobot, program menghitung total bobot dari MST yang terbentuk.

**Printscreen Output:**

****

Gambar 13. Printscreen Output MST

**Penjelasan Output:**

Output "Total Bobot Minimun Spanning Tree adalah: 14" menunjukkan bahwa total bobot dari Minimum Spanning Tree (MST) yang dihasilkan dari graf yang diberikan adalah 14.

1. **Disjoint Sets**
2. **Pengertian Disjoint Sets**

Disjoint Set didefinisikan sebagai struktur yang melacak sekumpulan elemen yang dipartisi menjadi sejumlah subset terpisah (tidak tumpang tindih). Disjoint Set merupakan sebuah struktur data yang memiliki 3 buah operasi, antara lain makeSet(x), findSet(x), dan union(x, y).

1. **Operasi makeSet(x)**

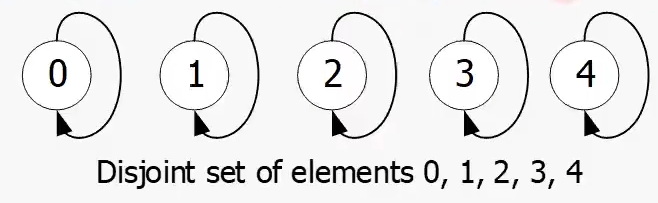
MakeSet(x) adalah sebuah operasi yang membuat sebuah set/himpunan baru yang mengandung sebuah elemen, dimana sebuah set/ himpunan yang memiliki hanya satu elemen disebut sebuah singleton set. Untuk operasi pertama pada disjoint set, makeSet(x) ialah:

1. Asumsikan suatu set/himpunan itu akan dibuat yang berisi sebuah elemen x.
2. Operasi makeSet(x) akan membuat sebuah set baru yang mengandung elemen x.



Gambar 14. Satu Set Berisi Elemen X

1. Induk dari x menunjuk kearah dirinya sendiri, karena itu menggambarkan x adalah root/akar set tersebut.
2. Jika kita membuat 5 elemen yang lain seperti: 0, 1, 2, 3, 4, kita harus memanggil make\_set(0), make\_set(1), make\_set(2), make\_set(3), dan make\_set(4).

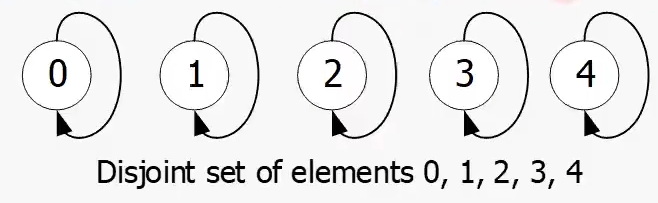


Gambar 15. Kumpulan Set yang Tidak Berkaitan

Dari gambar diatas, disjoint set adalah beberapa himpunan yang tidak memiliki hubungan sama sekali.

1. **Operasi findSet(x)**

Operasi findSet(x) adalah operasi return representasi dari sebuah set/himpunan dimana elemen tersebut berada. Operasi ini berguna untuk mengecek di set/himpunan mana elemen tersebut berada.



Gambar 16. Kumpulan Set yang Tidak Berkaitan

Set diatas merupakan root/akar dari set mereka sendiri dan apabila operasi findSet(x) dipanggil akan mereturn hasilnya sendiri.

findSet(0): 0

findSet(1): 1

findSet(2): 2

findSet(3): 3

findSet(4): 4

Karena dalam hal ini himpunan/set masih berbentuk singleton set.

1. **Operasi Union(x, y)**

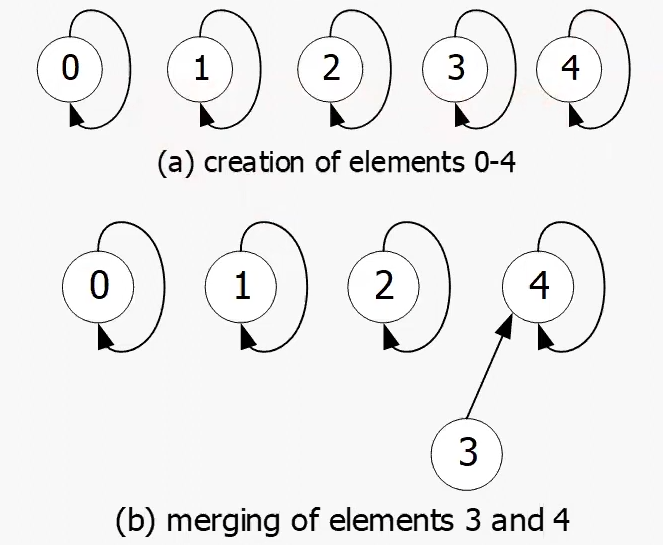
Operasi union(x, y) yang membutuhkan dua elemen yang saat ini dicontohkan x dan y. Operasi ini membuat set/himpunan baru yang mengandung elemen dari set x maupun set y serta menghapus set individual dari x dan y, dengan syarat x dan y berada di set yang berbeda. Selanjutnya ada operasi union, ide utama dari operasi union ini adalah untuk membuat pohon yang lebih kecil menjadi subset dari sebuah pohon yang lebih besar. Jika implemenetasi ini tidak digabungkan dengan optimalisasi, maka operasi union ini akan berjalan dengan lambat.

1. **Optimalisasi Operasi Union(x, y)**

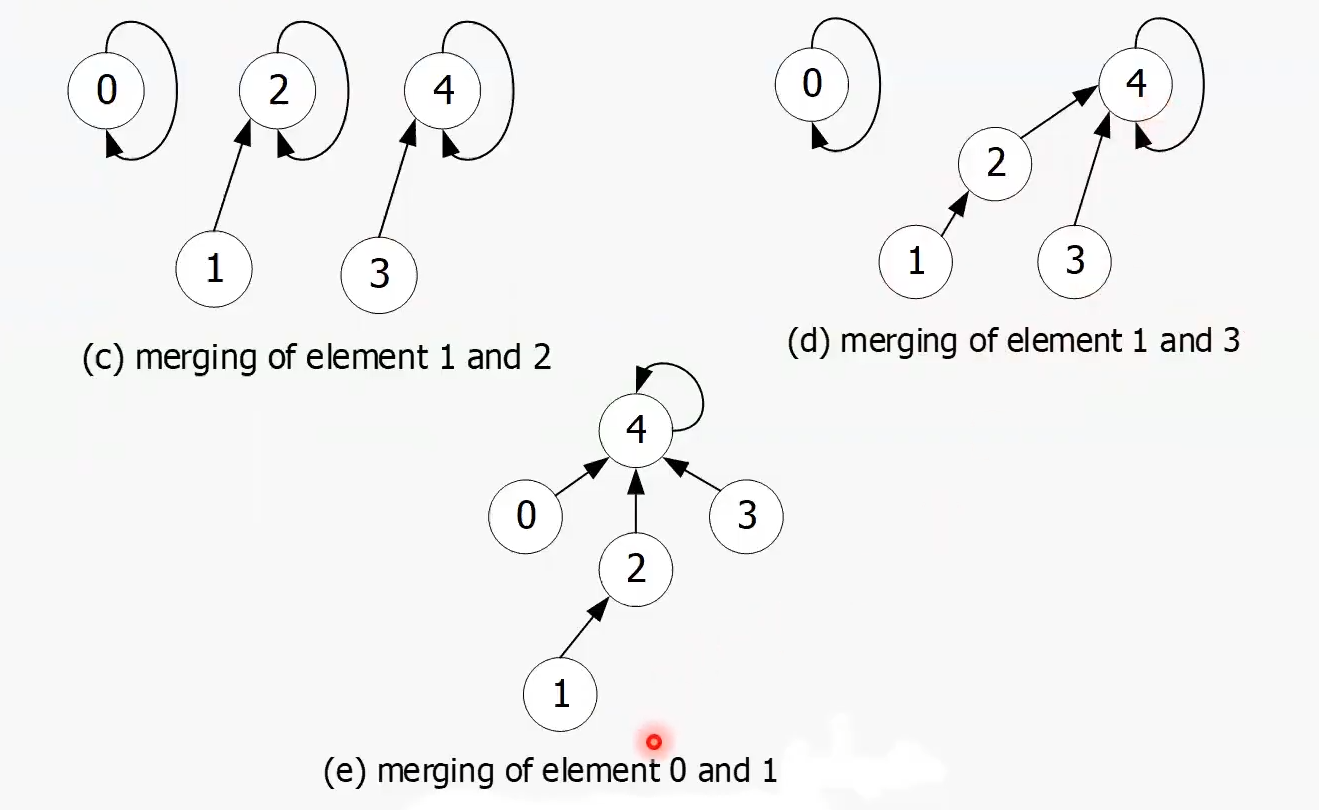
Union by Rank dan Union by Size adalah jawaban untuk optimalisasi dari masalah ini. Dengan dua metode union ini dapat membuat kontrol kedalaman pada pohon, sehingga pohon tetap simpel dan pendek, yang akan membuat operasi findSet dan union berjalan lebih cepat.

Union by rank ini merupakan metode dengan memperhatikan kedalaman pohon. Union ini mempunyai nilai yang mendekati logaritma dari besar sebuah sub-tree dan ini juga menentukan batas atas dari sebuah node/simpul. Union ini intinya menggabungkan pohon dengan kedalaman yang lebih kecil ke kedalaman yang lebih besar. Berikut cara operasi penggambungan/union melalui rank

1. Dua set bisa digabungkan dengan membuat root/akar salah satu set menunjuk ke akar dari set lain.
2. Operasi union rank di ilustrasikan seperti dibawah:



Gambar 17. Tahap Operasi Union A dan B



Gambar 18. Tahap Operasu Union C, D, dan E

1. Dari set berbeda dapat digabungkan dengan optimalisasi rank. Dari set 0 sampai 4 pada gambar A mereka memiliki rank yang sama yaitu 0.
2. Pada gambar b, 3 dan 4 di gabungkan dengan optimalisasi rank dengan menetapkan aturan urutan nilai terbesar menjadi rootnya (tidak ada aturan nilai terbesar selalu menjadi root, kembali ke pilihan pengguna), sehingga 4 terpilih menjadi root dan 3 menunjuk ke root barunya yaitu 4, sehingga sekarang rank dari 4 menjadi 1 sedangkan yang lain tetap 0. Aturan awal untuk memilih nilai mana yang diambil untuk menjadi root tidak berpatokan ke nilai paling besar, tergantung ke pengguna, bisa saja nilai terkecil diambil sebagai root.
3. Kemudian elemen 1 dan 2 disatukan, karena diawal menggunakan aturan nilai yang lebih besar sebagai root, maka set 2 menjadi rootnya ditandai dengan dia menunjuk dirinya sendiri dan 1 menunjuk 2 sebagai rootnya, di posisi ini rank dari 2 menjadi 1 seperti halnya set 4, sedangkan yang lain tetap 0.
4. Kemudian gambar d menunjukkan adanya penggabungan set 2 dan 4 dimana kembali menggunakan aturan yang ditetapkan diawal, dimana nilai yang lebih besar menjadi root dari nilai yang lebih kecil sehingga 2 akan menjadi subset dari 4, tetapi disini 2 akan menunjuk 4 langsung bukan ke set 3 karena awalnya 2 adalah sebuah root dan rank 2 juga lebih tinggi dari 3. Set 4 menjadi induk utama di tahap ini.
5. Dan yang terakhir adalah bergabungnya set 0 ke pohon tersebut, karena 0 adalah sebuah root dari dirinya sendiri, sehingga ia langsung menunjuk ke induk utama.

Union by Size adalah metode dengan memperhatikan banyak jumlah node/simpul dalam sebuah set. Di union ini, pohon dengan node/simpul yang lebih sedikit akan digabungkan ke pohon dengan node yang lebih banyak. Union by Size lumayan mirip dengan langkah union by rank, dengan contoh simple, jika terdapat set A dengan size 3 dan set B dengan size 5, maka A akan digabung ke B.

1. **Path Compression**

Pada operasi findSet(x) terdapat cara optimasi yang biasa dikenal Path Comprssion. Path Compression merupakan metode mempersimple jalur dengan membuat node/simpul dalam jalur langsung menunjuk ke root/induknya, sehingga lebih simple. Contoh: Sebelum menggunakan Path Compression, 0 → 1 → 2 → 3, maka setelah path compression findSet(0), node/simpul akan langsung menunjuk ke 3 (root), yaitu 0 → 3, 1 → 3, 2 → 3, 3 → 3.

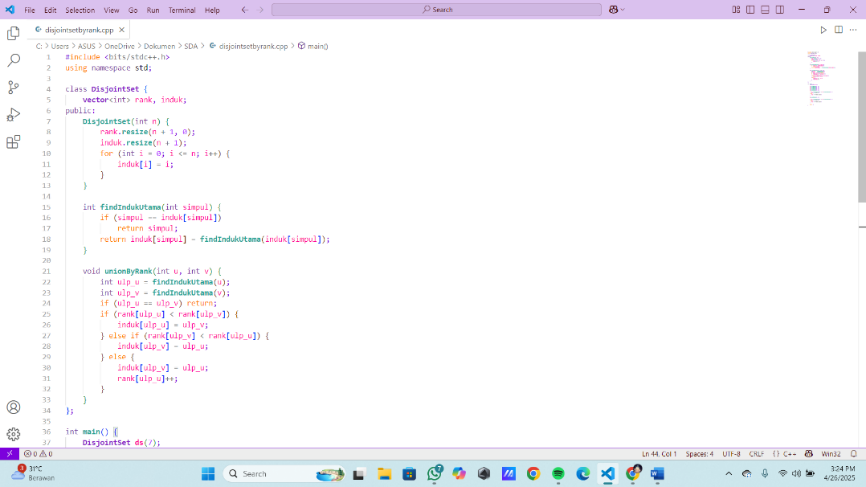
1. **Aplikasi**

Struktur data Disjoint Set banyak digunakan dalam berbagai algoritma dan permasalahan, antara lain:

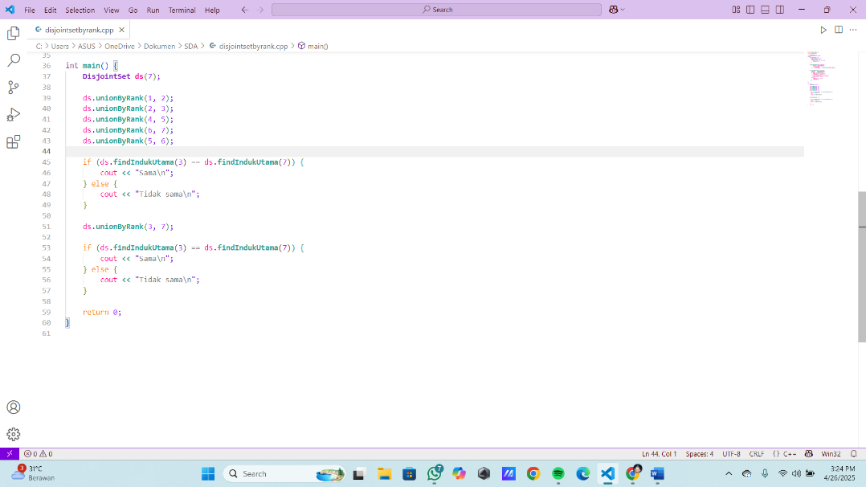
* Untuk mendeteksi siklus dalam graf.
* Dalam algoritma Kruskal untuk mencari Minimum Spanning Tree (MST).
* Untuk menyelesaikan masalah konektivitas dinamis, seperti mengecek apakah dua node berada dalam komponen yang sama

1. **Penerapan Kode**

**Printscreen Code:**

****

Gambar 19. Printscreen Source Code 1 Disjoint Set

****

Gambar 20. Printscreen Source Code 2 Disjoint Set

**Source Code:**

#include <bits/stdc++.h>

using namespace std;

class **DisjointSet** {

**vector**<int> rank, induk;

public:

**DisjointSet**(int n) {

        rank.**resize**(n + 1, 0);

        induk.**resize**(n + 1);

        for (int i = 0; i <= n; i++) {

            induk**[**i**]** = i;

        }

    }

    int **findIndukUtama**(int simpul) {

        if (simpul == induk**[**simpul**]**)

            return simpul;

        return induk**[**simpul**]** = **findIndukUtama**(induk**[**simpul**]**);

    }

    void **unionByRank**(int u, int v) {

        int ulp\_u = **findIndukUtama**(u);

        int ulp\_v = **findIndukUtama**(v);

        if (ulp\_u == ulp\_v) return;

        if (rank**[**ulp\_u**]** < rank**[**ulp\_v**]**) {

            induk**[**ulp\_u**]** = ulp\_v;

        } else if (rank**[**ulp\_v**]** < rank**[**ulp\_u**]**) {

            induk**[**ulp\_v**]** = ulp\_u;

        } else {

            induk**[**ulp\_v**]** = ulp\_u;

            rank**[**ulp\_u**]**++;

        }

    }

};

int **main**() {

**DisjointSet** **ds**(7);

    ds.**unionByRank**(1, 2);

    ds.**unionByRank**(2, 3);

    ds.**unionByRank**(4, 5);

    ds.**unionByRank**(6, 7);

    ds.**unionByRank**(5, 6);

    if (ds.**findIndukUtama**(3) == ds.**findIndukUtama**(7)) {

        cout **<<** "Sama\n";

    } else {

        cout **<<** "Tidak sama\n";

    }

    ds.**unionByRank**(3, 7);

    if (ds.**findIndukUtama**(3) == ds.**findIndukUtama**(7)) {

        cout **<<** "Sama\n";

    } else {

        cout **<<** "Tidak sama\n";

    }

    return 0;

}

**Penjelasan Source Code:**

Digunakannya #include <bits/stdc++.h> dan using namespace std; untuk mengimpor pustaka standar yang umum digunakan di C++. using namespace std; agar tidak perlu menulis std; di setiap fungsi seperti cout, cin, dan lain-lain. class DisjointSet { dan vector<int> rank, induk; adalah untuk mendeklarasikan kelas DisjointSet dengan dua vector yakni rank dan induk. Fungsi findIndukUtama digunakan untuk menemukan induk utama dari suatu simpul/node dengan cara path compression yang tadi dibahas. Fungsi unionByRank adalah dengan menggabungkan dua set berdasarkan induk utama masing-masing simpul/node. Di fungsi main, ada 7 simpul objek ds yang dilakukan beberapa operasi union dan untuk mengecek apakah simpul 3 dan simpul 7 memiliki induk utama yang sama sebelum dan sesudah operasi union.

**Printscreen Output:**

****

Gambar 21. Printscreen Outpun Disjoint Sets

**Penjelasan Output:**

Output yang pertama adalah ”Tidak sama” karena simpul 3 dan 7 belum di dalam set yang sama karena induk utama mereka berbeda dan setelah dilakukan operasi union, simpul 3 dan 7 digabungkan, ketika dicek lagi induk utama mereka sudah sama dan dengan itu output kedua keluar ”Sama”.

**BAB III**

**PENUTUP**

1. **Kesimpulan**

Memecah masalah rumit ke bagian-bagian kecil agar didapat solusi optimal merupakan cara kerja Dynamic Programming. Hal ini sering kali melibatkan pencarian nilai tertinggi atau terendah untuk kueri tertentu. Overlapping Subproblems dan Optimal Substructure merupakan hal dasar yang harus dikuasai pada Dynamic Programming, dengan teknik penyelesaian dengan dua pendekatan, top-down dan bottom-up. Minimum spanning tree ialah subgraf/pohon dengan jumlah bobot sisi (weighted edge) paling kecil. Dengan Algoritma Prim untuk pendekatan penyelesaian masalah yang sederhana atau dengan algoritma kruskal, menggunakan pendekatan disjoint sets dalam pemecahan masalahnya. Disjoint Set didefinisikan sebagai struktur yang melacak sekumpulan elemen yang dipartisi menjadi sejumlah subset terpisah (tidak tumpang tindih). Disjoint Set merupakan sebuah struktur data yang memiliki 3 buah operasi, antara lain makeSet(x), findSet(x), dan union(x, y).

**DAFTAR PUSTAKA**

Anita Sindar, R. M. S. (2019). *Struktur Data Dan Algoritma Dengan C++* (Vol. 1). CV. AA. RIZKY.

Bellman, R. (1966). Dynamic programming. *science*, *153*(3731), 34-37.

Galil, Z., & Italiano, G. F. (1991). Data structures and algorithms for disjoint set union problems. *ACM Computing Surveys (CSUR)*, *23*(3), 319-344.

Patwary, M. M. A., Blair, J., & Manne, F. (2010, May). Experiments on union-find algorithms for the disjoint-set data structure. In International Symposium on Experimental Algorithms (pp. 411-423). Berlin, Heidelberg: Springer Berlin Heidelberg.

Pettie, S., & Ramachandran, V. (2002). An optimal minimum spanning tree algorithm. *Journal of the ACM (JACM)*, *49*(1), 16-34.

Putri, M. P., Barovih, G., Azdy, R. A., Yuniansyah, Y., Saputra, A., Sriyeni, Y., ... & Admojo, F. T. (2022). Algoritma Dan Struktur Data.