Nama : Ifham Syafwan Fikri

Github : https://github.com/syafwan-ux/PenugasanPraktikumASD

NIM : 24/545184/PA/23161

Dosen Pengampu : I Gede Mujiyatna

PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

DISJOINT SET

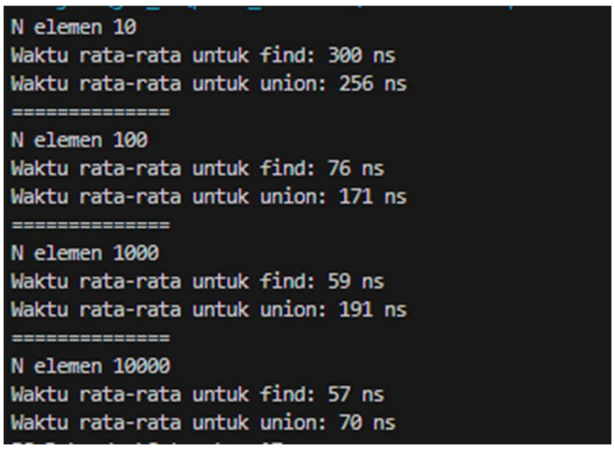
Eksperimen 1: Implementasi Naïve

Dalam permasalahan ini, eksperimen akan dilakukan untuk menguji waktu komputasi yang dijalankan oleh Naïve Disjoint Set dengan variabel yang diuji berupa 10, 100, 1000, dan 10000 elemen. Berikut merupakan langkah-langkah yang dilakukan:

1. Membuat dua file, antara lain benchmark (untuk menguji waktu komputasi) dengan iterasi sebanyak 10 kali dan membuat file main untuk menjalankan program yang telah diberikan
2. Berdasarkan program yang telah diberikan, program akan dipanggil ke dalam benchmark untuk dikalkulasi rata-rata waktu yang dipakai dan akan dipanggil lagi ke dalam file main dengan parameternya sebagai ukuran dari elemen yang ingin diuji
3. Setelah itu, variabel yang dimasukkan ke dalam operasi akan bernilai random dengan rentang ukuran elemen yang telah ditentukan sebelumnya. Operasi find akan mencari nilai random dan operasi union akan membandingkan kedua nilai yang random pula.

```java
import java.util.Random;

public class Benchmark {

    public static void N(int size) {
        Naive input = new Naive(size);
        Random random = new Random();

        long totalFind = 0;
        long totalUnion = 0;

        for (int i = 0; i < 10; i++) {
            long startFind = System.nanoTime();
            input.find(x:1);
            long endFind = System.nanoTime();
            totalFind += (endFind - startFind);

            int a = random.nextInt(size);
            int b = random.nextInt(size);

            long startUnion = System.nanoTime();
            input.union(a, b);
            long endUnion = System.nanoTime();
            totalUnion += (endUnion - startUnion);
        }
        System.out.println("Waktu rata-rata untuk find: " + totalFind / 10 + " ns");
        System.out.println("Waktu rata-rata untuk union: " + totalUnion / 10 + " ns");
    }


}
```

4. Berikut juga merupakan hasil yang telah dijalankan oleh program:



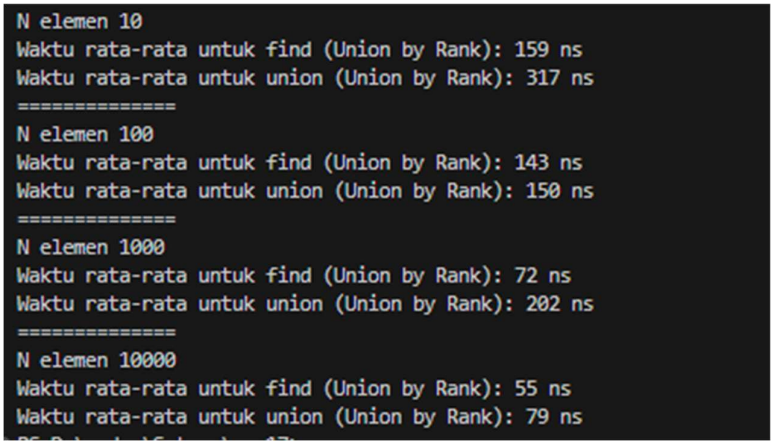
Banyak elemen	Operasi Find	Operasi Union
N(10)	300 ns	256 ns
N(100)	76 ns	171 ns
N(1000)	59 ns	191 ns
N(10000)	57 ns	70 ns

Analisis dan Diskusi:

- **Identifikasi risiko atau masalah potensial dengan implementasi Naïve Disjoint Set**
  Berdasarkan hasil yang diberikan, operasi find dan union pada banyak elemen 10 memiliki waktu komputasi lebih lama daripada yang lain. Hal ini bisa disebabkan Naïve Method itu sendiri dengan operasi find yang dilakukan secara rekursif ke tiap parent tanpa path compression sehingga menyebabkan kompleksitas waktu hingga O(n) pada kasus terburuk dan operasi union yang hanya menghubungkan satu root ke satu root lainnya sehingga pohon menjadi tidak seimbang. Dengan demikian, nilai yang dihasilkan akan dapat jauh berbeda dari yang lain.

Eksperimen 2: Implementasi Union by Rank

Pada permasalahan ini, eksperimen yang dilakukan akan sama seperti eksperimen sebelumnya, tetapi hanya mengganti operasi union dengan Union by Rank. Operasi ini akan menjaga pohon agar pohon yang merepresentasikan himpunan tetap datar (tidak terlalu dalam). Dengan pohon yang lebih datar, operasi find menjadi lebih cepat karena jumlah langkah yang harus dilalui untuk menemukan root lebih sedikit. Berikut merupakan hasil yang telah dijalankan:



Banyak Elemen	Find	Union
N(10)	159 ns	317 ns
N(100)	143 ns	150 ns
N(1000)	72 ns	202 ns
N(10000)	55 ns	79 ns

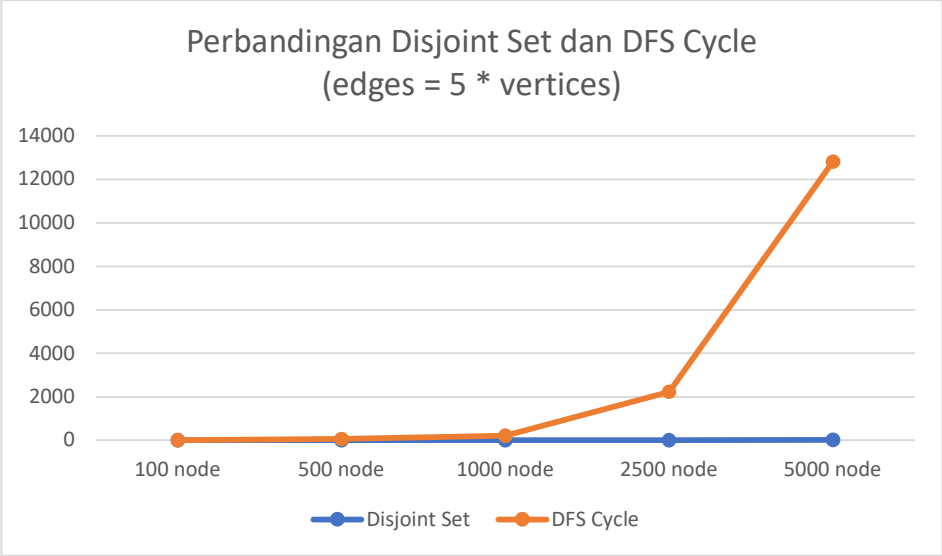
Analisis dan Diskusi:

- **Bagaimana kinerja Union by Rank dengan Naïve Disjoint Set?** Berdasarkan hasil benchmark untuk Union by Rank, banyak elemen 10 tetap lebih lama waktu komputasinya daripada banyak elemen lainnya. Hal ini bisa disebabkan karena operasi find masih menggunakan rekursi yang memiliki overhead. Meskipun demikian, secara umum Union by Rank memiliki performa yang lebih tinggi dalam operasi find daripada Naïve Method karena mengubah pohon menjadi datar agar dapat lebih mudah untuk dicari walaupun operasi union masih tergolong lebih lambat dalam eksperimen ini.

- **Jika kinerjanya berbeda, apa penyebab perbedaan kinerja tersebut?** Seperti yang disimpulkan pada eksperimen sebelumnya, Naïve Method dalam operasi find dilakukan secara rekursif ke tiap parent tanpa path compression dan operasi union yang hanya menghubungkan satu root ke satu root lainnya sehingga pohon menjadi tidak seimbang (struktur pohon dapat lebih panjang antara root awal dan child terakhir), sedangkan Union by Rank menyusun struktur pohon menjadi lebih datar (parent bisa memiliki banyak child) sehingga dapat lebih cepat mengakses elemen untuk operasi find.

Eksperimen 3: Perbandingan MST dan Kruskal

Berdasarkan program yang telah dibuat sebelumnya, yaitu algoritma Kruskal dengan menggunakan DFS Cycle, akan diuji kinerja dengan beberapa variabel seperti banyak node dan edges. Antara lain 100, 500, 1000, 2500, dan 5000 nodes dengan edges lima kali dari jumlah nodes (atau vertices). Berikut merupakan grafik dari hasil program yang telah dijalankan:



Analisis dan Disksui:

- **Apa keuntungan atau peranan struktur data disjoint set dalam Upaya optimasi komputasi?** Berdasarkan grafik dari hasil yang telah diberikan, algoritma Kruskal yang menggunakan Depth First Search (DFS) Cycle waktu komputasinya lebih lama dibandingkan dengan yang menggunakan Disjoint Set sehingga untuk menghadapi masalah-masalah tertentu dengan data yang sangat banyak Disjoint set dapat diimplementasikan untuk menghemat waktu komputasi.

- **Hasil dari program yang telah diuji untuk Disjoint Set (DFS Cycle bisa diabaikan)**