PROJECT UAS ALGORITMA PEMROGRAMAN

(Tugas ini bertujuan untuk melengkapi project UAS mata kuliah Algoritma Pemrograman yang dibimbing oleh Dosen Christian Sri Kusuma Aditya, S.Kom., M.Kom.)

Kelompok 2:

1. Adinda Okta Rubiyanti (2023-104)
2. Muhammad Fadhil Yusran Zuhair (2023-293)
3. Adriansyah Pratama (2023-481)
4. Muhammad Wahyudiono Putra (2023-408)

JURUSAN INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MUHAMMADIYAH MALANG

2024/2025

Transport Management Application

Dalam proyek ini, kami menggunakan bahasa pemrograman Java untuk mengimplementasikan beberapa konsep algoritma yang telah dipelajari selama perkuliahan. Implementasi ini mencakup tiga topik utama, yaitu:

1. **Array - Interpolation Search**
    - Algoritma ini digunakan untuk mencari rute transportasi berdasarkan stasiun asal atau tujuan dalam daftar rute yang telah diurutkan. **Interpolation Search** dipilih karena memberikan kinerja yang lebih efisien dibandingkan pencarian linier atau binary search dalam kondisi data yang seragam dan terurut, seperti halnya daftar stasiun yang terorganisir dengan baik.
2. **Tree - InOrder Traversal**
    - Traversal ini digunakan untuk mengorganisir dan menyusun stasiun atau rute transportasi berdasarkan kategori tertentu, misalnya berdasarkan abjad atau jarak. Dengan menggunakan metode **InOrder Traversal**, sistem dapat mengunjungi setiap node dalam urutan tertentu, yang sangat berguna untuk menampilkan data rute atau stasiun secara terstruktur dan terorganisir.
3. **Graph - Algoritma Dijkstra**
    - Algoritma ini diterapkan untuk menemukan jalur terpendek menuju tujuan transportasi yang dipilih, dari stasiun asal. **Dijkstra's Algorithm** sangat relevan untuk menyelesaikan masalah pencarian rute optimal dalam jaringan transportasi yang memiliki banyak stasiun dan jalur transportasi.

Melalui implementasi algoritma-algoritma ini, kami berharap dapat memperdalam pemahaman mengenai cara kerja dan aplikasi nyata dari algoritma tersebut, khususnya dalam konteks manajemen rute transportasi yang kompleks. Selain itu, proyek ini menjadi kesempatan untuk menggabungkan teori dan praktik dalam menyelesaikan masalah berbasis algoritma, sekaligus mengasah keterampilan pemrograman berbasis Java.

README
Transport Management Application

Aplikasi Transport Management ini bertujuan untuk membantu pengguna mencari rute transportasi dari satu stasiun ke stasiun lain dengan berbagai fitur seperti pencarian jadwal berdasarkan tanggal, pencarian rute, traversal pohon, dan perhitungan rute terpendek menggunakan algoritma Dijkstra.

Fitur

1. Pencarian Jadwal Berdasarkan Tanggal
    - Pengguna dapat memasukkan tanggal (DD), bulan (MM), dan tahun (YYYY) untuk mencari jadwal transportasi pada tanggal tertentu.
    - Format input harus valid, yaitu DD-MM-YYYY.
2. Pencarian Rute Transportasi

- Pengguna dapat memilih stasiun asal dan tujuan untuk mencari rute yang tersedia.
- Aplikasi mendukung pencarian semua jalur yang menghubungkan asal dan tujuan (bukan hanya jalur terpendek).

3. Traversal Pohon (InOrder)
   - Menampilkan traversal InOrder untuk stasiun yang tersedia (fungsi simulasi).
4. Pencarian Rute Terpendek dengan Dijkstra
   - Menggunakan algoritma Dijkstra untuk menemukan rute terpendek dari stasiun asal ke tujuan yang dipilih.
   - Menampilkan total jarak dalam kilometer untuk rute terpendek.

Penjelasan Bagian Kode

1. Array - Interpolation Search

- Pencarian menggunakan Interpolation Search bertujuan untuk menemukan elemen dalam array berdasarkan pendekatan yang lebih efisien dibandingkan dengan pencarian biner. Meskipun tidak langsung diimplementasikan dalam kode ini, penggunaan algoritma pencarian lain dapat diterapkan menggunakan konsep dasar Interpolation Search pada array stasiun atau rute yang tersimpan.
- Interpolasi Pencarian umumnya lebih cepat untuk data yang terurut, dengan memanfaatkan informasi nilai untuk memprediksi posisi elemen yang dicari, daripada membagi ruang pencarian secara seragam seperti pada Binary Search.

Contoh penggunaan Interpolation Search (pada array terurut):

```
public int interpolationSearch(int[] array, int value) {
   int low = 0;
   int high = array.length - 1;
   while (low <= high && value >= array[low] && value <= array[high]) {
      int mid = low + ((value - array[low]) * (high - low)) / (array[high] - array[low]);
      if (array[mid] == value) {
         return mid; // Element found
      }
      if (array[mid] < value) {
         low = mid + 1;
      } else {
         high = mid - 1;
      }
   }
```

```
    return -1; // Element not found
}
```

2. Tree - Traverse Binary Tree (InOrder)

- Traversal Pohon (InOrder) digunakan untuk mengakses elemen-elemen dalam urutan tertentu dari pohon biner (binary tree). Dalam konteks aplikasi ini, pohon bisa digunakan untuk mengatur stasiun atau rute transportasi dalam urutan tertentu. Traversal InOrder mengunjungi node kiri, kemudian root, dan terakhir node kanan.
- Traversal InOrder akan mengunjungi setiap node dalam pohon secara berurutan, yang sering digunakan untuk mendapatkan hasil yang terurut dari struktur pohon biner.

Contoh penggunaan Traversal InOrder (untuk Binary Tree):

```
public void inorderTraversal(Node root) {
    if (root == null) {
        return;
    }
    inorderTraversal(root.left);  // Kunjungi subpohon kiri
    System.out.print(root.value + " "); // Akses node
    inorderTraversal(root.right);  // Kunjungi subpohon kanan
}
```

Penerapan dalam aplikasi: Fungsi traversal ini menampilkan stasiun dalam urutan tertentu berdasarkan struktur data yang ada.

3. Graph - Algoritma Dijkstra

- Algoritma Dijkstra adalah algoritma graf yang digunakan untuk menemukan rute terpendek antara dua node dalam sebuah graf berbobot. Pada aplikasi ini, graf representasi digunakan untuk menggambarkan stasiun dan jarak antar stasiun.
- Dijkstra bekerja dengan memulai dari node asal dan kemudian mencari jalur terpendek ke setiap node tetangga, mengupdate jarak yang lebih kecil jika ditemukan jalur yang lebih efisien. Proses ini terus dilakukan sampai seluruh graf dieksplorasi.

Implementasi Dijkstra dalam aplikasi:

```
private List<String> findShortestPath(Map<String, Map<String, Integer>> graph, String
start, String end) {
    Map<String, Integer> distances = new HashMap<>();
    Map<String, String> previous = new HashMap<>();
    PriorityQueue<String> nodes = new
PriorityQueue<>(Comparator.comparingInt(distances::get));


    // Inisialisasi jarak dan node
```

```
    for (String node : graph.keySet()) {
        distances.put(node, node.equals(start) ? 0 : Integer.MAX_VALUE);
        nodes.add(node);
    }

    // Mencari rute terpendek
    while (!nodes.isEmpty()) {
        String closest = nodes.poll();
        if (closest.equals(end)) {
            List<String> path = new ArrayList<>();
            while (previous.containsKey(closest)) {
                path.add(0, closest);
                closest = previous.get(closest);
            }
            path.add(0, start);
            return path;
        }
        // Update jarak untuk tetangga
        for (Map.Entry<String, Integer> neighbor : graph.get(closest).entrySet()) {
            int alt = distances.get(closest) + neighbor.getValue();
            if (alt < distances.get(neighbor.getKey())) {
                distances.put(neighbor.getKey(), alt);
                previous.put(neighbor.getKey(), closest);
                nodes.add(neighbor.getKey());
            }
        }
    }
    return null; // Tidak ditemukan rute
}
```

- Fungsi ini menggunakan Priority Queue untuk memilih node dengan jarak terpendek yang belum dieksplorasi. Hasil akhirnya adalah jalur terpendek dari stasiun asal ke tujuan yang diinginkan.

Cara Menggunakan

1. Jalankan Aplikasi
   Aplikasi ini menggunakan Java Swing untuk GUI. Pastikan Anda telah menginstal Java Development Kit (JDK).
    - Untuk menjalankan aplikasi, cukup kompilasi dan jalankan kelas TransportManagementApp:
2. javac TransportManagementApp.java
3. java TransportManagementApp
4. Pencarian Jadwal
    - Masukkan tanggal, bulan, dan tahun yang diinginkan di kolom yang tersedia, kemudian klik tombol Cari Jadwal. Aplikasi akan menampilkan hasil jadwal pada tanggal tersebut.
5. Pencarian Rute
    - Pilih stasiun asal dan tujuan dari dropdown dan klik tombol Temukan Rute untuk mencari semua rute yang menghubungkan kedua stasiun tersebut.
6. Traversal InOrder
    - Klik tombol Traversal InOrder untuk melihat traversal urutan pohon dari stasiun yang tersedia.
7. Pencarian Rute Terpendek
    - Pilih stasiun asal dan tujuan, kemudian klik tombol Temukan Rute Terpendek. Aplikasi akan menampilkan rute terpendek dan total jarak antara kedua stasiun.

Struktur Kode

1. TransportManagementApp
    - Kelas utama yang mengatur antarmuka pengguna dan logika untuk masing-masing fitur (pencarian jadwal, rute, dan Dijkstra).
    - Menggunakan JFrame, JTextArea, JTextField, dan JComboBox untuk interaksi pengguna.
2. Graph
    - Menggunakan struktur data Map<String, Map<String, Integer>> untuk mewakili graf yang menggambarkan stasiun dan jarak antar stasiun.
3. Algoritma Dijkstra
    - Digunakan untuk mencari rute terpendek antara dua stasiun berdasarkan jarak.

Dependensi

- Java 8 atau versi yang lebih baru.
- Tidak ada dependensi eksternal, hanya menggunakan pustaka standar Java.

Cara Menambah Stasiun dan Rute Baru

Untuk menambah stasiun atau rute baru, Anda dapat memodifikasi metode setupGraph() pada kelas TransportManagementApp. Berikut adalah contoh cara menambahkan stasiun baru:

```
graph.put("Stasiun E", Map.of("Stasiun A", 15, "Stasiun B", 10));
```

Ini akan menambahkan Stasiun E dengan rute yang menghubungkannya ke Stasiun A dan Stasiun B.
