Lab Week 4 Vispro

Mata Kuliah: Visual Programming

Materi Praktikum ke 4

Nama: Aryo Karel Merentek

NIM: 0806022310017

IMT-AI

Tanggal Praktikum: Oktober 18, 2024

BAB I

PENDAHULUAN

1.1. Latar Belakang

"Mengingat daftar kota dan jarak antara masing-masing pasangan kota, rute terpendek manakah yang dapat mengunjungi setiap kota tepat satu kali dan kembali ke kota asal?" adalah pertanyaan yang diajukan oleh travelling salesman problem (di singkat TSP) dalam teori kompleksitas komputasi. Dalam optimalisasi kombinatorial, ini adalah masalah NP-hard yang penting bagi teori ilmu komputer dan riset operasi.

1.2. Tujuan

Tujuan dari proyek ini adalah untuk memberikan pemahaman lebih mendalam tentang Travelling Salesman Problem (TSP) dan teknik algoritmik yang dapat diterapkan untuk menyelesaikannya.

1.3. Tinjauan Pustaka

Traveling Salesman Problem (TSP) adalah tantangan optimasi kombinatorial NP-hard yang terkenal dan signifikan di berbagai bidang. Penyelesaian TSP muncul dari beragam metodologi, termasuk komputasi kuantum, algoritma heuristik, dan teknik pembelajaran mesin. Metode heuristik, seperti hibridisasi simulasi anil dan algoritma tetangga terdekat, telah menunjukkan hasil yang menjanjikan dalam hal kecepatan dan kualitas solusi (Polupanova & Rybalko, 2023). Selain itu, integrasi pembelajaran penguatan mendalam dengan heuristik konvensional telah memberikan hasil yang mengesankan dalam penyelesaian kasus TSP (Tian et al., 2024). Secara keseluruhan, penelitian TSP masih aktif, dengan kemajuan yang dicapai baik dalam penerapan teoretis maupun praktis (Balinskyy & Kitun, 2023).

BAB II

ALAT DAN BAHAN

2.1 Alat

a. Hardware

- Laptop
  - Lenovo Legion Slim 5 - Intel i7 13th Gen, RTX 4060 Mobile

b. Software

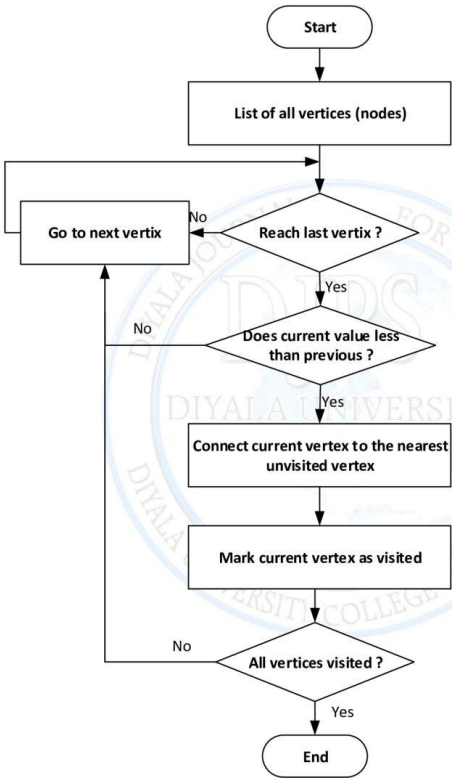
- Integrated Development Environment (IDE)
  - Visual Studio Code
- Windows Powershell Terminal
- Windows Powershell
- ChatGPT

2.2 Bahan

a. Dart Programming Language

- Dart SDK version: 3.5.3 (stable)

b. Flowchart Referensi



BAB III

PROSEDUR KERJA

Pendekatan yang akan saya pakai untuk menyelesaikan “Travelling Salesman Problem” atau di singkat TSP adalah dengan menggunakan Algoritma Greedy. Algoritma Greedy adalah pendekatan yang mengambil keputusan terbaik dalam jangka pendek di setiap langkah dengan harapan mendapatkan solusi keseluruhan yang baik. Dalam TSP, masalah yang di angkat adalah mencari rute terpendek untuk seorang penjual yang harus mengunjungi sejumlah kota, di mana setiap kota harus di kunjungi sekali dengan rute yang paling optimal.

Cara Kerja Algoritma Greedy untuk TSP:

1. Tentukan Titik Awal: Mulai dari suatu kota awal yang dipilih (biasanya dipilih secara acak atau sebagai kota pertama).
2. Pilih Kota Terdekat Berikutnya: Dari kota saat ini, pilih kota terdekat yang belum dikunjungi. Ini dilakukan dengan memilih jalur yang memiliki bobot atau jarak terkecil.
3. Kembali ke Langkah 2: Lanjutkan dengan kota terdekat berikutnya yang belum dikunjungi, dan tambahkan ke rute yang telah terbentuk.
4. Kembali ke Kota Awal: Setelah semua kota telah dikunjungi, kembali ke kota asal untuk menutup siklus.

Kelebihan dari pendekatan ini adalah caranya yang sederhana dan mudah di implementasikan serta waktu eksekusinya yang cepat. Tetapi kekurangan dari pendekatan ini adalah algoritma ini tidak menjamin solusi optimal secara global karena memilih langkah terbaik secara local tanpa mempertimbangkan konsekuensi jangka panjang. Oleh sebab itu pendekatan ini tidak cocok untuk masalah yang “memiliki banyak kota”. Pendekatan Algoritma Greedy dapat memberikan solusi cepat tetapi tidak selalu merupakan rute terpendek secara keseluruhan. Algoritma ini cocok di pakai jika pencarian Solusi lebih cepat lebih di utamakan dari pada optimalitasi rute.

BAB IV

HASIL & PEMBAHASAN

1. **Kelas Vertex**

```
1   class Vertex {
2     int id;
3     bool visited = false;
4     Vertex(this.id);
5   }
```

Kelas Vertex mewakili satu node dalam grafik. Ini memiliki:

- Id untuk mengidentifikasi vertex secara unik.
- Flag visited yang dikunjungi (awalnya di set ke false) untuk melacak apakah vertex sudah diproses.
- Konstruktor Vertex(this.id) menginisialisasi vertex dengan yang diberikan ID

2. **Kelas Graph**

```
7   class Graph {
8     List<Vertex> vertices;
9     List<List<int>> adjacencyMatrix;
10
11    Graph(this.vertices, this.adjacencyMatrix);
```

Kelas Graph memodelkan grafik menggunakan:

- vertices: Daftar objek Vertex, yang mewakili node dalam grafik.
- adjacencyMatrix: Daftar 2D (matriks) bilangan bulat yang mewakili koneksi dan bobot antar vertices.
    - Setiap elemen adjacencyMatrix[i][j] mewakili bobot atau jarak dari vertex i ke vertex j.
    - Nilai 0 menunjukkan bahwa tidak ada hubungan langsung antara vertices tersebut.

3. Mencari Vertex Terdekat yang Belum Dikunjungi

```
13    // cari titik terdekat yang belum dikunjungi dari indeks saat ini yang diberikan
14    int findNearestUnvisitedVertex(int currentIndex) {
15      int nearestIndex = -1;
16      int minValue = double.maxFinite.toInt();
17
18      for (int i = 0; i < adjacencyMatrix[currentIndex].length; i++) {
19        if (!vertices[i].visited &&
20            adjacencyMatrix[currentIndex][i] < minValue &&
21            adjacencyMatrix[currentIndex][i] != 0) {
22          minValue = adjacencyMatrix[currentIndex][i];
23          nearestIndex = i;
24        }
25      }
26
27      return nearestIndex;
28    }
```

- Metode ini menemukan vertex terdekat yang belum dikunjungi dari vertex tertentu (currentIndex).
    - nearestIndex diinisialisasi ke -1, menunjukkan bahwa belum ada vertex terdekat yang ditemukan.
    - minValue diatur ke nilai integer maksimum yang mungkin (double.maxFinite.toInt()), mewakili bobot terkecil yang ditemukan.
    - For loop mengulangi semua vertices:
        - jika sebuah vertex belum dikunjungi (!vertices[i].visited), dan bobotnya (adjacencyMatrix[currentIndex][i]) lebih kecil dari minValue saat ini, dan bukan 0, maka:
            - perbarui minValue ke bobot saat ini.
            - Tetapkan nearestIndex ke indeks vertex ini (i).
    - Metode ini mengembalikan Indeks terdekat, yang merupakan indeks dari vertex terdekat yang belum dikunjungi, atau -1 jika tidak ada vertex tersebut.

4. Melintasi Grafik (graph)

```
30    // melakukan algoritma sesuai flowchart
31    traverse() {
32      for (int i = 0; i < vertices.length; i++) {
33        if (vertices[i].visited) continue;
34
35        int currentIndex = i;
36        while (!vertices[currentIndex].visited) {
37          vertices[currentIndex].visited = true;
38          int nearestIndex = findNearestUnvisitedVertex(currentIndex);
39
40          if (nearestIndex != -1) {
41            print('Connecting vertex ${vertices[currentIndex].id} to vertex ${vertices[nearestIndex].id}');
42            currentIndex = nearestIndex;
43          } else {
44            break;
45          }
46        }
47      }
48    }
49  }
```

- Metode ini melakukan traversal dan koneksi vertices berdasarkan algoritma:
    - Pengulangan for loop bagian luar melakukan iterasi pada semua vertices.
        - Jika vertex tersebut sudah dikunjungi, ia akan melompat ke vertex berikutnya.
    - Perulangan while loop akan terus berlanjut selama vertex saat ini tidak dikunjungi.
        - Menandai vcrtex saat ini sebagai telah dikunjungi (vertices[currentIndex].visited = true).
        - Menemukan vertex terdekat yang belum dikunjungi menggunakan findNearestUnvisitedVertex(currentIndex).
        - Jika vertex terdekat yang belum dikunjungi ditemukan (Indeks terdekat != -1), maka:
            - Mencetak pesan yang menunjukkan hubungan antara vertex saat ini dan vertex terdekat.
            - Memperbarui currentIndex ke nearestIndex, sehingga algoritma berlanjut dari vertex baru ini.
- Jika tidak ditemukan vertex terdekat yang belum dikunjungi, perulangan terputus, berpindah ke vertex berikutnya di perulangan for loop terluar.

5. Fungsi Utama (main function)

```
Run | Debug
main() {
  // contoh graph dengan 5 vertices
  var vertices = List.generate(5, (index) => Vertex(index));

  // matriks ketetanggaan mewakili koneksi grafik dan bobot (0 menunjukkan tidak ada koneksi)
  var adjacencyMatrix = [
    [0, 2, 0, 6, 0],
    [2, 0, 3, 8, 5],
    [0, 3, 0, 0, 7],
    [6, 8, 0, 0, 9],
    [0, 5, 7, 9, 0]
  ];

  var graph = Graph(vertices, adjacencyMatrix);
  graph.traverse();
}
```

- Fungsi utama mendemonstrasikan penggunaan kelas Graph:
  - Ini membuat contoh grafik dengan 5 vertices (0 hingga 4).
  - AdjacencyMatrix mewakili jarak antar vertices. Misalnya:
    - adjacencyMatrix[0][1] adalah 2, menunjukkan hubungan antara vertex 0 dan vertex 1 dengan bobot 2.
    - adjacencyMatrix[0][2] adalah 0, menunjukkan tidak ada koneksi langsung.
  - Sebuah instance dari Graph dibuat dengan daftar vertices dan adjacency matrix.
  - Metode traverse() dipanggil untuk memulai algoritma.

6. Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\USER\Desktop\dartCircleofHell> dart main.dart
Connecting vertex 0 to vertex 1
Connecting vertex 1 to vertex 2
Connecting vertex 2 to vertex 4
Connecting vertex 4 to vertex 3
PS C:\Users\USER\Desktop\dartCircleofHell>
```

BAB V
KESIMPULAN

Terdapat berbagai pendekatan yang dapat digunakan untuk menyelesaikan masalah “Travelling Salesman Problem” atau TSP. Pendekatan yang di pakai umumnya dikategorikan menjadi dua jenis: Metode Eksak dan Metode Heuristic/Approximate dan Metaheuristic. Beberapa contoh pendekatan melalui metode Eksak antara lain: Brute Force, Dynamic Programming (Algoritma Held-Karp), Branch and Bound, dan Integer Linear Programming (ILP). Sedangkan beberapa contoh pendekatan melalui Metode Heuristic/Approximate dan Metaheuristic termasuk: Algoritma Greedy, Algoritma Nearest Neighbor, Algoritma Simulated Annealing, Algoritma Genetic (GA), Ant Colony Optimization (ACO), Tabu Search, Particle Swarm Optimization (PSO), Algoritma Koloni Lebah Buatan, dsb.

Pendekatan yang saya pakai untuk menyelesaikan masalah TSP dalam bentuk program dart adalah dengan menggunakan “Algoritma Greedy” dari Metode Heuristic/Metaheuristic. Mirip dengan algoritma Prim atau Djikstra, algoritma ini membuat keputusan terbaik secara local pada setiap Langkah dengan memilih kota terdekat berikutnya. Walaupun hasilnya biasa tidak optimal, tetapi kelebihan dari Algoritma Greedy ini adalah hasilnya yang cepat dengan tidak mempertimbangkan keputusan jangka panjang. Saya memilih menggukan algoritma ini karena proses komputasinya yang sederhana dan kompleksitas waktu yang lebih rendah. Lain dari itu, menurut saya melalui pendekatan ini, saya bisa lebih mudah mengubah instruksi dari flowchart menjadi sebuah program dart tanpa menggunakan library.

Pada akhirnya, pendekatan yang di pakai untuk menyelesaikan “Travelling Salesman problem”, kapan di pakai dan mengapa bergantung kepada banyak variabel. Karena setiap metode memiliki kelebihan dan kekurangannya masing-masing, seperti ukuran sebuah “Kota” dan kebutuhan solusi (optimalisasi vs waktu komputasi). Pendekatan serta metode yang di pilih sebaiknya sesuai dengan kebutuhan use-case atau objektif yang ingin di capai karena TSP memiliki banyak aplikasi bahkan dalam formulasinya yang paling murni, seperti perencanaan, logistik, dan pembuatan microchip.

DAFTAR PUSAKA

Wikipedia contributors. (2024, October 14). *Travelling salesman problem*. Wikipedia. https://en.wikipedia.org/wiki/Travelling_salesman_problem

Polupanova, E., & Rybalko, A. A. (2023). SEQUENTIAL HYBRIDIZATION ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM SOLVING. *Izvestiâ ÛFU. Tehničeskie Nauki*, *3*, 108–118. https://doi.org/10.18522/2311-3103-2023-3-108-118

Travelling Salesman Problem (Greedy approach). (n.d.). https://www.tutorialspoint.com/data_structures_algorithms/travelling_salesman_problem.htm

Heuristic Methods for Solving the Traveling Salesman Problem (TSP): A Comparative Study. (2023, September 5). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/10293957

Al-Neama, M. W., Ahmed, I. A., & Ali, S. M. (2023). A parallel algorithm to find the exact solution of the travelling salesman problem. *Indonesian Journal of Electrical Engineering and Computer Science*, *31*(2), 917.

https://doi.org/10.11591/ijeecs.v31.i2.pp917-924

W3Schools.com. (n.d.).

https://www.w3schools.com/dsa/dsa_ref_traveling_salesman.php