Effective Dart

Mata Kuliah: Visual Programming

Materi Praktikum ke: Week 4

Nama: Deny Wahyudi Asaloei

NIM: 0806022310009

Tanggal Praktikum:

18 Oktober 2024

BAB I

PENDAHULUAN

1.1. Latar Belakang

Dalam konteks pembelajaran algoritma optimasi, diberikan sebuah flowchart yang menggambarkan alur penyelesaian masalah pencarian rute terpendek dengan 5 titik (A, B, C, D, E). Flowchart tersebut menunjukkan pendekatan iteratif untuk menemukan solusi, namun dalam implementasinya, pendekatan Travelling Salesman Problem (TSP) dengan metode Brute Force dipilih sebagai solusi alternatif yang dapat memberikan hasil yang lebih optimal.

TSP sendiri merupakan permasalahan optimasi klasik dalam ilmu komputer yang bertujuan mencari rute terpendek untuk mengunjungi sejumlah titik tepat satu kali dan kembali ke titik awal. Meskipun flowchart yang diberikan menunjukkan pendekatan yang berbeda, penggunaan algoritma Brute Force dalam TSP dapat memberikan solusi yang komprehensif dengan mencoba semua kemungkinan rute yang ada.

1.2. Tujuan

Tujuan dari Praktikum ini adalah:

- Memahami konsep Travelling Salesman Problem (TSP)
- Mengimplementasikan algoritma brute force untuk menyelesaikan Travelling Salesman Problem
- Mengimplementasikan fungsi rekursif untuk menghasilkan semua kemungkinan jalur (permutasi) dalam Travelling Salesman Problem (TSP)

1.3. Tinjauan Pustaka

- Travelling Salesman Problem (TSP)

  Travelling Salesman Problem (TSP) adalah masalah optimasi yang bertujuan menemukan rute terpendek bagi seorang penjual yang harus mengunjungi beberapa lokasi dan kembali ke titik awalnya. TSP pertama kali diperkenalkan pada 1930-an oleh Karl Menger, dengan aplikasi luas dalam logistik dan perencanaan rute.

- Algoritma Brute-force

  Pendekatan Brute-Force, juga dikenal sebagai pendekatan Naive, adalah metode untuk menghitung dan membandingkan semua kemungkinan permutasi rute untuk menemukan solusi rute terpendek. Dengan metode ini, kita perlu menghitung jumlah rute yang mungkin, menggambar semua rute, dan membuat daftar setiap kemungkinan rute. Setelah itu, jarak dari setiap rute dihitung, dan rute terpendek dipilih sebagai solusi terbaik.

- Rekursi dalam Pemrograman

  Rekursi adalah teknik pemrograman di mana fungsi memanggil dirinya sendiri untuk memecahkan masalah secara bertahap. Dalam TSP, rekursi digunakan untuk menghasilkan permutasi rute, memecah masalah besar menjadi sub-masalah kecil hingga kondisi dasar tercapai. Rekursi menyederhanakan implementasi brute-force, meski tetap memerlukan waktu yang signifikan.

BAB II

ALAT DAN BAHAN

2.1.Alat

- Laptop
- Dart SDK versi 3.5.3
- Visual Studio Code atau IDE lainnya yang mendukung Dart
- Github
- Terminal

2.2.Bahan

- Flowchart algoritma sebagai referensi
- Graph dengan 5 vertex (A, B, C, D, E) beserta jaraknya
- Source code implementasi TSP (Bruteforce.dart)

BAB III

PROSEDUR KERJA

Pendekatan Algoritma

3.1.Algoritma Utama Brute Force: Program menggunakan algoritma Brute Force untuk mencari rute terpendek dengan langkah-langkah berikut:

- Program dimulai dengan titik awal A
- Membuat semua kemungkinan permutasi rute untuk titik-titik lainnya (B, C, D, E)
- Menghitung total jarak untuk setiap permutasi rute
- Membandingkan setiap rute untuk menemukan jarak terpendek
- Menyimpan rute dengan jarak terpendek sebagai solusi optimal

3.2.Langkah-langkah Implementasi:

a. Persiapan Data
   - Menyiapkan data jarak antar titik dalam bentuk matriks
   - Mendefinisikan nama titik (A, B, C, D, E)

b. Pembuatan Fungsi Utama
   - calculateDistance: Menghitung total jarak suatu rute
   - generatePermutations: Membuat semua kemungkinan urutan kunjungan
   - tspBruteForce: Menjalankan algoritma dan mencari rute terbaik

c. Proses Pencarian
   - Mulai dari titik A
   - Coba semua kemungkinan urutan kunjungan untuk titik lainnya
   - Bandingkan jarak setiap rute
   - Simpan rute dengan jarak terpendek

BAB IV

HASIL & PEMBAHASAN

4.1.Deklarasi Matriks Jarak dan Daftar Vertix:

```
List<List<int>> graph = [
  [0, 8, 3, 4, 10],
  [8, 0, 5, 2, 7],
  [3, 5, 0, 1, 6],
  [4, 3, 1, 0, 3],
  [10, 7, 6, 3, 0]
];

List<String> vertices = ['A', 'B', 'C', 'D', 'E'];
```

Graph adalah Matriks adjacency yang merepresentasikan jarak antar simpul. Baris dan kolom dari matriks ini merepresentasikan tiap simpul yang ada. Nilai pada graph[i][j] adalah jarak dari simpul i ke simpul j. Contoh, jarak antara A dan B adalah 8, jarak antara B dan D adalah 2, dst. Sedangkan vertices adalah daftar/array yang menyimpan nama-nama dari tiap simpul.

4.2.Fungsi calculateDistance:

```
int calculateDistance(List<int> path) {
  int distance = 0;
  for (int i = 0; i < path.length - 1; i++) {
    distance += graph[path[i]][path[i + 1]];
  }
  distance += graph[path.last][path.first];
  return distance;
}
```

Fungsi calculateDistance digunakan untuk menghitung jarak total dari jalur yang diberikan dalam bentuk daftar indeks vertex (simpul) yang dilalui. Fungsi ini menerima parameter path, yang merupakan daftar urutan indeks vertex yang membentuk jalur. Pertama, variabel distance diinisialisasi dengan nilai 0 untuk menyimpan total jarak. Selanjutnya, loop for digunakan untuk iterasi melalui setiap indeks dalam path, kecuali yang terakhir, untuk menjumlahkan jarak antara setiap pasangan vertex yang berurutan. Jarak antara dua vertex diakses melalui matriks graph, yang menyimpan jarak antar vertex. Setelah loop selesai, jarak dari vertex terakhir kembali ke vertex pertama juga ditambahkan untuk menyelesaikan perjalanan siklus. Akhirnya, fungsi mengembalikan nilai total distance, yang mewakili jarak total dari jalur yang diberikan. Dengan demikian, fungsi ini efektif dalam menghitung jarak yang diperlukan untuk menyelesaikan perjalanan dari satu vertex ke vertex lainnya dalam graf.

4.3.Fungsi generatePermutations:

```
List<List<int>> generatePermutations(List<int> elements) {
  if (elements.length == 1) {
    return [elements];
  }

  List<List<int>> permutations = [];
  for (int i = 0; i < elements.length; i++) {
    var sublist = List<int>.from(elements)..removeAt(i);
    var subPermutations = generatePermutations(sublist);
    for (var perm in subPermutations) {
      permutations.add([elements[i]] + perm);
    }
  }
  return permutations;
}
```

Fungsi generatePermutations bertujuan untuk menghasilkan semua permutasi dari sebuah daftar elemen bertipe List<int>. Fungsi ini dimulai dengan memeriksa apakah panjang daftar elements sama dengan satu, yang merupakan kondisi dasar atau **base case** dari rekursi. Jika panjangnya satu, berarti hanya ada satu elemen yang tersisa, dan fungsi mengembalikan daftar tersebut sebagai satu-satunya permutasi yang mungkin. Hal ini penting karena **base case** mencegah fungsi masuk ke dalam rekursi tak berujung, sehingga memberikan titik akhir pada proses rekursif.

Jika daftar memiliki lebih dari satu elemen, fungsi akan menginisialisasi daftar kosong permutations untuk menyimpan semua permutasi yang dihasilkan. Melalui sebuah loop, fungsi akan melakukan iterasi di setiap elemen daftar. Pada setiap iterasi, elemen ke-i dihapus dari daftar dengan membuat salinan baru menggunakan List<int>.from(elements) dan kemudian menggunakan metode removeAt(i). Selanjutnya, fungsi akan memanggil dirinya sendiri secara rekursif dengan sublist yang telah dimodifikasi, untuk menghasilkan semua permutasi dari elemen yang tersisa.

Hasil permutasi dari sublist ini disimpan dalam variabel subPermutations. Setelah mendapatkan semua permutasi dari sublist, fungsi kemudian menambahkan kembali elemen ke-i ke dalam setiap permutasi yang dihasilkan, membentuk permutasi baru yang akan dimasukkan ke dalam daftar permutations. Proses ini berlanjut hingga semua elemen telah diproses, dan akhirnya fungsi mengembalikan daftar permutations, yang berisi semua kombinasi yang mungkin dari permutasi elemen dalam daftar asli.

4.4.Funsi tspBruteforce:

```
void tspBruteForce() {

  print("1. List of all vertices: ${vertices.join(', ')}");
  List<int> indices = List.generate(vertices.length, (i) => i);
  print("   Indices of vertices: $indices");
  int startVertex = indices.removeAt(0);
```

```
  print("   Start vertex: ${vertices[startVertex]}");

  print("\n2. Generating all permutations starting with A...");
  var permutations = generatePermutations(indices);
  print("   Total permutations generated: ${permutations.length}");
  List<int>? bestPath;
  int minDistance = double.maxFinite.toInt();

  print("\n3. Iterating through all permutations:");
  for (var path in permutations) {
    path.insert(0, startVertex);
    print("\n   Checking path: ${path.map((i) => vertices[i]).join(' -> ')} ->
A");

    int distance = calculateDistance(path);
    print("   4. Calculated distance: $distance");

    if (bestPath == null) {
      minDistance = distance;
      bestPath = path;
      print("   5. First path checked. Updating best path and distance.");
    } else {
      print("   5. Is current distance ($distance) less than previous best
($minDistance)?");
      if (distance < minDistance) {
        minDistance = distance;
        bestPath = path;
        print("      Yes! Updating best path and distance.");
      } else {
        print("      No, keeping previous best path and distance.");
      }
    }
  }

  print("\n6. All permutations checked: Yes");

  print("\n7. Displaying results:");
  if (bestPath != null) {
    print('   Best path: ${bestPath.map((i) => vertices[i]).join(' -> ')} -> A');
  }
  print('   Minimum total distance: $minDistance');
  print("\n8. End of algorithm");
}
```

Fungsi tspBruteForce ini digunakan untuk menyelesaikan masalah Travelling Salesman Problem (TSP) dengan pendekatan brute-force. Fungsi ini dimulai dengan mencetak daftar semua vertex (simpul) pada graf, yang diwakili oleh variabel vertices. Selanjutnya, sebuah daftar indices dihasilkan untuk menyimpan indeks dari setiap vertex, dan vertex pertama (A) dihapus dari daftar untuk dijadikan titik awal. Setelah itu, fungsi memanggil generatePermutations, yang bertugas menghasilkan semua permutasi dari vertex lainnya, sehingga kita dapat mengeksplorasi setiap kemungkinan jalur yang dimulai dari A.

Setelah semua permutasi dihasilkan, algoritma mengiterasi setiap jalur (perm) dan menambahkan kembali vertex A sebagai titik awal. Untuk setiap jalur yang dihasilkan, fungsi calculateDistance dipanggil untuk menghitung jarak total jalur tersebut. Jarak ini diperiksa dan dibandingkan dengan jarak minimum sebelumnya. Jika jalur saat ini lebih pendek, jalur tersebut disimpan sebagai jalur terbaik (best path), dan jarak minimum diperbarui. Jika tidak, jalur terbaik sebelumnya dipertahankan.

Proses ini diulang hingga semua permutasi diperiksa. Setelah semua jalur dicek, hasil terbaik, termasuk jalur terbaik dan jarak minimum, ditampilkan. Fungsi ini memastikan bahwa kita menemukan jalur terpendek yang dimulai dan berakhir di vertex awal (A).

4.5.Fungsi main:

```
void main() {
  tspBruteForce();
}
```

Bagian kode void main() adalah titik masuk utama program. Fungsi ini memanggil tspBruteForce(), yang menjalankan seluruh algoritma brute-force untuk menyelesaikan masalah Travelling Salesman Problem (TSP). Fungsi tspBruteForce() akan melakukan semua langkah yang diperlukan untuk menghitung dan menemukan jalur terpendek yang mengunjungi semua vertex dalam graf, dimulai dan berakhir di vertex awal (A). Dengan memanggil tspBruteForce() dalam main(), program memastikan bahwa algoritma ini dieksekusi segera ketika aplikasi dijalankan.

BAB V

KESIMPULAN

Praktikum ini bertujuan untuk mengimplementasikan algoritma brute-force dalam menyelesaikan masalah Travelling Salesman Problem (TSP) dengan 5 titik simpul (A, B, C, D, E). Penggunaan algoritma brute-force disini bertujuan untuk mencoba semua kemungkinan permutasi rute yang ada agar dapat menemukan solusi rute terpendek.

Dalam pelaksanaannya, saya menggunakan fungsi rekursif untuk mendapatkan semua rute permutasi yang mungkin. Proses ini memungkinkan saya untuk menganalisis setiap jalur dan menghitung total jarak untuk menentukan rute terdekat.

Melalui praktik ini, saya tidak hanya memahami mekanisme kerja dari algoritma brute-force, tetapi juga pentingnya pendekatan ini dalam menyelesaikan masalah optimasi seperti TSP. Meskipun metode brute-force memiliki keterbatasan seperti waktu pemrosesan yang akan bertambah lama jika jumlah titik semakin besar, pendekatan ini tetap menjadi cara paling efektif untuk mendapatkan hasil yang paling optimal pada skala kecil

DAFTAR PUSTAKA

Technology, B. V. (2023, January 24). *Travelling salesman problem : Its definition and implementation*. Bhumi Varta Technology. https://bvarta.com/travelling-salesman-problem-its-definition-and-implementation/

GeeksforGeeks. (2021, March 12). Dart recursion. *GeeksforGeeks*. https://www.geeksforgeeks.org/dart-recursion/

GeeksforGeeks. (2013, November 3). Travelling Salesman Problem using Dynamic Programming. *GeeksforGeeks*. https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/

Kuo, M. (2020, January 2). *Algorithms for the travelling salesman problem*. https://www.routific.com/blog/travelling-salesman-problem