

Teori Dijkstra

2. 1 Pengertian

Algoritma Dijkstra adalah metode untuk menemukan jalur terpendek dalam graf dengan bobot non-negatif. Metode ini ditujukan untuk menghitung jarak terpendek dari satu simpul awal ke semua simpul lainnya. Dijkstra dikategorikan sebagai algoritma yang bersifat greedy karena ia selalu memilih simpul dengan jarak sementara paling kecil pada setiap langkahnya.

2. 2 Dasar Teori

Sebuah graf terdiri dari simpul dan sisi yang masing-masing memiliki bobot tertentu. Bobot ini dapat mewakili jarak, biaya, atau waktu. Tujuan dari algoritma Dijkstra adalah untuk meminimalkan jumlah bobot dari sumber ke simpul yang dituju.

Konsep inti:

- Relaksasi: Proses memperbaharui jarak minimum ketika ada jalur yang lebih pendek ditemukan.
- Nod yang sudah dikunjungi: Node yang jaraknya telah ditentukan dan final.
- Nod yang belum dikunjungi: Node yang jaraknya masih bisa diperbarui.

1. 3 Prinsip Kerja

1. Atur semua jarak menjadi tak terhingga, kecuali untuk node awal yang diatur menjadi 0.
2. Pilih node dengan jarak terkecil yang belum dikunjungi.
3. Tandai node tersebut sebagai sudah dikunjungi.
4. Lakukan relaksasi untuk semua tetangganya.
5. Terus ulangi proses ini sampai semua node dikunjungi atau tujuan tercapai.

1. 4 Struktur Data

- Antrian Prioritas (Min-Heap): digunakan untuk memilih node dengan jarak terkecil secara efisien.
- Array distance[]: menyimpan jarak minimum yang saat ini.
- Array parent[]: menyimpan rute yang dilalui untuk membentuk jalur.

1. 5 Kompleksitas

- Tanpa min-heap: $O(V^2)$
- Menggunakan min-heap: $O((V + E) \log V)$
- Dengan Fibonacci Heap: $O(E + V \log V)$

Penggunaan min-heap adalah praktik standar dalam implementasi modern karena efisiensi dan kemudahan penggunaannya.

1. 6 Kelebihan dan Kekurangan

Kelebihan

- Cepat dan efisien untuk graf yang besar.
- Tidak memerlukan perhitungan yang rumit.
- Memiliki determinisme (hasilnya selalu sama).

Kekurangan

- Tidak dapat diterapkan pada bobot negatif.
- Kinerja menurun jika graf sangat padat tanpa adanya optimasi.

1. 7 Aplikasi

- Google Maps atau GPS
- Routing di jaringan komputer (contohnya OSPF)
- Perencanaan logistik
- Penemuan jalur dalam permainan
- Simulasi serta navigasi robotik

2. Outline Ringkas (1 Halaman)

- Definisi graf dan jalur terpendek
- Dijkstra untuk graf dengan bobot non-negatif
- Prinsip greedy
- Langkah-langkah algoritma: inisialisasi \rightarrow pilih jarak terendah \rightarrow relaksasi \rightarrow ulangi
- Struktur data utama: antrian prioritas

- Kompleksitas: $O((V+E) \log V)$
- Kelebihan dan kekurangan
- Contoh aplikasi

3. Pseudocode Dijkstra

```

1  function Dijkstra(graph, source):
2      for each vertex v in graph:
3          distance[v] = ∞
4          parent[v] = null
5          distance[source] = 0
6
7      priorityQueue = min-heap berisi (source, 0)
8
9      while priorityQueue tidak kosong:
10         (u, dist_u) = extract-min(priorityQueue)
11
12         for setiap tetangga v dari u:
13             if distance[u] + weight(u, v) < distance[v]:
14                 distance[v] = distance[u] + weight(u, v)
15                 parent[v] = u
16                 decrease-key(priorityQueue, v, distance[v])
17
18     return distance, parent
19

```

4. Contoh Ilustrasi Sederhana

Misalkan grafik:

- A → B (4)
- A → C (2)
- B → C (5)
- B → D (10)
- C → E (3)
- E → D (4)

Tujuan: temukan jalur terpendek dari A.

Iterasi:

- Mulai: dist[A]=0; yang lainnya=∞
- Pilih A
 - o perbarui B=4, C=2
- Pilih C
 - o perbarui E=2+3=5
- Pilih B
 - o D sementara = 4+10 = 14
- Pilih E

o perbarui $D = 5 + 4 = 9 \rightarrow$ lebih kecil dari 14

- Selesai

Hasil:

- Jalur menuju D = $A \rightarrow C \rightarrow E \rightarrow D$
- Jarak total = 9

5. Implementasi Python

```
1  import heapq
2
3  def dijkstra(graph, start):
4      # graph = {node: [(tetangga, bobot), ...], ...}
5
6      distances = {node: float('inf') for node in graph}
7      distances[start] = 0
8      parent = {node: None for node in graph}
9
10     pq = [(0, start)] # (jarak, node)
11
12     while pq:
13         current_dist, current_node = heapq.heappop(pq)
14
15         if current_dist > distances[current_node]:
16             continue
17
18         for neighbor, weight in graph[current_node]:
19             distance = current_dist + weight
20
21             if distance < distances[neighbor]:
22                 distances[neighbor] = distance
23                 parent[neighbor] = current_node
24                 heapq.heappush(pq, (distance, neighbor))
25
26     return distances, parent
27
```

```
29 # Contoh graf
30 graph = {
31     'A': [('B', 4), ('C', 2)],
32     'B': [('C', 5), ('D', 10)],
33     'C': [('E', 3)],
34     'D': [],
35     'E': [('D', 4)]
36 }
37
38 distances, parent = dijkstra(graph, 'A')
39
40 print("Jarak terpendek:", distances)
41 print("Parent:", parent)
42
```

LANDASAN TEORI

2. 1 Graf

Graf adalah struktur data yang berfungsi untuk merepresentasikan hubungan antara objek. Struktur ini terdiri dari dua elemen utama, yaitu simpul yang mewakili objek dan sisi yang melambangkan hubungan antar objek. Graf digunakan di berbagai sektor, termasuk jaringan komputer, sistem transportasi, navigasi, dan untuk menentukan jalur terpendek.

2. 1. 1 Komponen Graf

Simpul (Node)

Merupakan titik yang melambangkan objek, lokasi, atau entitas dalam suatu sistem.

Sisi (Edge)

Menghubungkan dua simpul. Sisi dapat dibedakan menjadi:

Graf Berarah: hubungan yang memiliki arah.

Graf Tak Berarah: hubungan yang bersifat dua arah.

Bobot

Merupakan nilai yang terhubung pada setiap sisi. Bobot bisa berupa jarak, waktu, biaya, kapasitas, atau parameter lain yang dianggap relevan.

2. 1. 2 Jenis-Jenis Graf

Graf Berbobot

Setiap sisi memiliki bobot tertentu.

Graf Tidak Berbobot

Sisi tidak memiliki bobot sama sekali.

Graf Berarah

Setiap sisi memiliki arah tertentu.

Graf Tak Berarah

Hubungan yang bersifat dua arah antara simpul.

2. 1. 3 Representasi Graf

Graf dapat direpresentasikan dengan dua metode utama:

Daftar Ketetanggaan

Setiap simpul menyimpan daftar tetangga beserta bobot hubungan yang ada. Representasi ini efisien untuk graf yang jarang.

Matriks Ketetanggaan

Menyajikan dalam bentuk matriks $N \times N$ yang menunjukkan apakah dua simpul berhubungan. Metode ini menggunakan lebih banyak memori, tetapi memberikan akses yang mudah.

2. 2 Algoritma Dijkstra

Algoritma Dijkstra adalah metode klasik yang digunakan untuk menemukan jalur terpendek dari satu simpul sumber ke simpul lain dalam graf berbobot yang tidak memiliki bobot negatif. Algoritma ini diciptakan oleh Edsger W. Dijkstra pada tahun 1956 dan tetap menjadi dasar penting dalam komputasi graf.

2. 2. 1 Tujuan Algoritma

Menentukan jarak terpendek dari simpul sumber ke semua simpul dalam graf.

Memastikan jalur terpendek yang dihasilkan akurat dan efisien.

2. 2. 2 Prinsip Kerja

Algoritma Dijkstra beroperasi dengan pendekatan greedy, memilih solusi terbaik di setiap tahap untuk mencapai hasil optimal. Dalam setiap langkah, algoritma memilih simpul dengan jarak sementara terkecil dan melakukan proses relaksasi terhadap tetangganya.

2. 2. 3 Konsep Dasar Dijkstra

Array Jarak

Menyimpan nilai jarak terpendek yang didapat dari sumber ke setiap simpul.

Set Diketahui

Menandai simpul yang jaraknya sudah pasti.

Relaksasi

Proses memperbarui jarak jika jalur yang baru ditemukan lebih pendek daripada yang sebelumnya:

```
"jarak_baru"="jarak[u]"+"bobot(u,v)"
```

Antrian Prioritas (Min-Heap)

Digunakan untuk memilih simpul dengan jarak terkecil secara efisien.

2. 2. 4 Langkah-Langkah Algoritma Dijkstra

Inisialisasi semua jarak dengan nilai ∞ , kecuali sumber yang bernilai 0.

Tambahkan simpul sumber ke dalam antrian prioritas.

Ambil simpul dengan jarak terkecil dari antrian.

Lakukan relaksasi terhadap semua tetangga dari simpul ini.

Jika ditemukan jarak baru yang lebih kecil, perbarui dan masukkan kembali ke antrian.

Ulangi proses sampai semua simpul diproses atau antrian kosong.

2. 2. 5 Kompleksitas Waktu

Tanpa menggunakan struktur data heap: $O(V^2)$

Menggunakan min-heap: $O((V + E) \log V)$

Implementasi yang berbasis heap adalah yang paling umum digunakan.

2. 2. 6 Kelebihan dan Kekurangan

Kelebihan:

Cepat dan efisien untuk graf yang besar.

Cocok untuk menemukan rute dalam sistem navigasi.

Tidak membutuhkan perhitungan yang rumit.

Kekurangan:

Tidak dapat diaplikasikan pada graf dengan bobot negatif.

Kinerjanya menurun pada graf yang sangat padat jika tanpa optimasi.

2. 2. 7 Aplikasi Algoritma Dijkstra

Sistem navigasi GPS seperti Google Maps dan Waze.

Routing dalam jaringan komputer contohnya OSPF.

Pathfinding dalam permainan.

Meniru pergerakan suatu robot.

Perencanaan rute dan distribusi.

NEW

LANDASAN TEORI

2. 1 Graf

Graf merupakan suatu bentuk struktur data yang digunakan untuk menggambarkan hubungan antar objek. Struktur ini terdiri dari dua elemen pokok, yaitu simpul yang merepresentasikan objek dan sisi yang menunjukkan hubungan antara objek tersebut. Graf memiliki berbagai penerapan di banyak bidang, termasuk dalam jaringan komputer, sistem transportasi, navigasi, dan untuk mencari rute terpendek.

2. 1. 1 Komponen Graf

Simpul (Node)

Adalah titik yang melambangkan objek, lokasi, atau entitas dalam suatu sistem.

Sisi (Edge)

Menghubungkan dua simpul. Sisi dapat dibedakan menjadi:

Graf Berarah: hubungan yang memiliki arah tertentu.

Graf Tak Berarah: hubungan yang bersifat dua arah.

Bobot

Adalah nilai yang terasosiasi pada setiap sisi. Bobot ini bisa merepresentasikan jarak, waktu, biaya, kapasitas, atau parameter lainnya yang dianggap berhubungan.

2. 1. 2 Jenis-Jenis Graf

Graf Berbobot

Setiap sisi memiliki nilai bobot tertentu.

Graf Tidak Berbobot

Sisi tidak memiliki bobot sama sekali.

Graf Berarah

Setiap sisi memiliki arah yang spesifik.

Graf Tak Berarah

Menunjukkan hubungan yang bersifat dua arah antara simpul.

2. 1. 3 Representasi Graf

Graf dapat divisualisasikan dengan dua cara utama:

Daftar Ketetanggaan

Pada setiap simpul terdapat daftar tetangga beserta bobot hubungan yang ada. Representasi ini efisien untuk graf yang jarang.

Matriks Ketetanggaan

Menyajikan informasi dalam bentuk matriks $N \times N$ yang menampilkan apakah dua simpul saling berhubungan. Metode ini menggunakan lebih banyak ruang memori, tetapi memberikan akses yang lebih gampang.

2. 2 Algoritma Dijkstra

Algoritma Dijkstra merupakan metode yang klasik untuk menentukan jalur terpendek dari satu simpul sumber menuju simpul lain dalam graf berbobot yang tidak memiliki bobot negatif. Algoritma ini ditemukan oleh Edsger W. Dijkstra pada tahun 1956 dan tetap menjadi referensi penting dalam komputasi graf.

2. 2. 1 Tujuan Algoritma

Menemukan jarak terpendek dari simpul awal ke seluruh simpul dalam graf.

Memastikan bahwa jalur terpendek yang dihasilkan adalah akurat dan efisien.

2. 2. 2 Prinsip Kerja

Algoritma Dijkstra bekerja dengan pendekatan greedy, memilih pilihan terbaik pada setiap langkah untuk mencapai hasil yang optimal. Di setiap tahapan, algoritma ini memilih simpul dengan jarak sementara terkecil dan melakukan proses relaksasi terhadap tetangganya.

2. 2. 3 Konsep Dasar Dijkstra

Array Jarak

Menyimpan nilai jarak terpendek yang diperoleh dari sumber menuju setiap simpul.

Set Diketahui

Menandai simpul yang jaraknya sudah ditentukan.

Relaksasi

Proses memperbarui jarak ketika jalur yang baru ditemukan lebih pendek dibandingkan yang sebelumnya:

"jarak_baru"="jarak[u]"+"bobot(u,v)"

Antrian Prioritas (Min-Heap)

Dipakai untuk dengan efisien memilih simpul yang memiliki jarak paling kecil.

2. 2. 4 Langkah-Langkah Algoritma Dijkstra

Inisialisasi semua jarak dengan nilai ∞ , kecuali untuk sumber yang bernilai 0.

Masukkan simpul sumber ke dalam antrian prioritas.

Ambil simpul dengan jarak terpendek dari antrian.

Lakukan relaksasi pada semua tetangga dari simpul tersebut.

Jika ada jarak baru yang lebih kecil, perbarui dan masukkan kembali ke antrian.

Ulangi proses ini hingga semua simpul diproses atau antrian kosong.

2. 2. 5 Kompleksitas Waktu

Tanpa menggunakan struktur data heap: $O(V^2)$

Menggunakan min-heap: $O((V + E) \log V)$

Implementasi berbasis heap adalah yang paling umum diterapkan.

2. 2. 6 Kelebihan dan Kekurangan

Kelebihan:

Cepat dan efisien untuk graf berukuran besar.

Cocok digunakan untuk mencari rute dalam sistem navigasi.

Tidak memerlukan perhitungan yang rumit.

Kekurangan:

Tidak dapat digunakan pada graf yang memiliki bobot negatif.

Kinerjanya berkurang pada grafik yang sangat padat apabila tidak dilakukan optimasi.

2. 2. 7 Penerapan Algoritma Dijkstra

Sistem navigasi satelit seperti Google Maps dan Waze.

Pengaturan jalur dalam jaringan komputer seperti OSPF.

Penemuan jalur dalam video game.

Meniru gerakan sebuah robot.

Perencanaan rute dan distribusi.