Tugas Mandiri 1
Perancangan dan Analisis Algoritma

“Dijkstra’s Algorithm”

Oleh:
Muhammad Asyrof
21343056

Dosen Pengampu:
Randi Proska Sandra, M.Sc

PRODI INFORMATIKA
JURUSAN TEKNIK ELEKTRONIKA
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG
2023

Dijkstra's Algorithm

A. Penjelasan Dijkstra's Algorithm

Algoritma Dijkstra merupakan algoritma yang paling sering digunakan dalam pencarian rute terpendek, sederhana penggunaannya dengan menggunakan simpul-simpul sederhana pada jaringan jalan yang tidak rumit (Chamero, 2006). Adapun nama algoritma Dijkstra sendiri berasal dari penemunya yaitu Edsger Dijkstra. Dalam mencari solusi, algoritma Dijkstra menggunakan prinsip Greedy, yaitu mencari solusi optimum pada setiap langkah yang dilalui, dengan tujuan untuk mendapatkan solusi optimum pada langkah selanjutnya yang akan mengarah pada solusi terbaik.

Algoritma Dijkstra merupakan salah satu varian dari algoritma greedy, yaitu salah satu bentuk algoritma populer dalam pemecahan persoalan yang terkait dengan masalah optimasi. Sifatnya sederhana dan lempang (straightforward). Sesuai dengan artinya yang secara harafiah berarti tamak atau rakus – namun tidak dalam konteks negatif –, algoritma greedy ini hanya memikirkan solusi terbaik yang akan diambil pada setiap langkah tanpa memikirkan konsekuensi ke depan. Prinsipnya, ambillah apa yang bisa Anda dapatkan saat ini (take what you can get now!), dan keputusan yang telah diambil pada setiap langkah tidak akan bisa diubah kembali. Intinya algoritma greedy ini berupaya membuat pilihan nilai optimum lokal pada setiap langkah dan berharap agar nilai optimum lokal ini mengarah kepada nilai optimum global. Misalnya, bila vertices dari sebuah graf melambangkan kota-kota dan bobot sisi (edge weights) melambangkan jarak antara kota-kota tersebut, maka algoritma Dijkstra dapat digunakan untuk menemukan jarak terpendek antara dua kota (Raja, N, & Irwansyah, 2015).

Elemen-elemen penyusun algoritma greedy adalah:

1. Himpunan kandidat, C, Himpunan ini berisi elemen-elemen yang memiliki peluang untuk membentuk solusi. Pada persoalan lintasan terpendek dalam graf, himpunan kandidat ini adalah himpunan simpul pada graf tersebut.
2. Himpunan solusi, S, Himpunan ini berisi solusi dari permasalahan yang diselesaikan dan elemennya terdiri dari elemen dalam himpunan kandidat namun tidak semuanya atau dengan kata lain himpunan solusi ini adalah upabagian dari himpunan kandidat.
3. Fungsi seleksi, Fungsi seleksi adalah fungsi yang akan memilih setiap kandidat yang yang memungkinkan untuk menghasilkan solusi optimal pada setiap langkahnya.
4. Fungsi kelayakan, Fungsi kelayakan akan memeriksa apakah suatu kandidat yang telah terpilih (terseleksi) melanggar constraint atau tidak. Apabila kandidat melanggar constraint maka kandidat tidak akan dimasukkan ke dalam himpunan solusi
5. Fungsi objektif, Fungsi objektif akan memaksimalkan atau meminimalkan nilai solusi. Tujuannya adalah memilih satu saja solusi terbaik dari masing-masing anggota himpunan solusi.

Berikut ini merupakan langkah-langkah umum yang dilakukan oleh Dijkstra's Algorithm:

1. Inisialisasi jarak dari titik awal ke semua titik lainnya sebagai tak terhingga, dan jarak dari titik awal ke dirinya sendiri sebagai 0.
2. Tambahkan titik awal ke antrian prioritas.
3. Ambil titik dengan jarak terpendek dari antrian prioritas.
4. Untuk setiap tetangga yang belum dikunjungi dari titik yang dipilih, hitung jarak ke tetangga tersebut dan perbarui jarak jika jarak yang baru lebih pendek dari jarak sebelumnya.
5. Tambahkan tetangga-tetangga yang belum dikunjungi ke antrian prioritas.
6. Ulangi langkah 3-5 sampai antrian prioritas kosong atau titik akhir telah ditemukan.
7. Kembalikan jarak terpendek dari titik awal ke titik akhir.

Dijkstra's Algorithm memiliki kompleksitas waktu yang relatif efisien, yaitu O (E log V), dimana E adalah jumlah sisi pada grafik dan V adalah jumlah titik. Algoritma ini sering digunakan pada aplikasi pemetaan jalan, navigasi, dan optimasi jaringan lainnya.

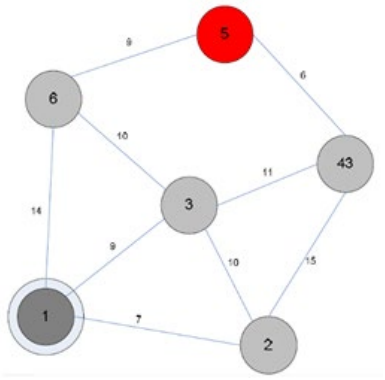

Figure 1.Contoh Gambat Algoritma Dijkstra

Pada Gambar diatas menampilkan contoh dari Algoritma Dijkstra dimana enam lingkaran disebut sebagai node, dan angka disetiap garis disebut juga bobot. Lalu bagaimana menentukan jalur terpendeknya. Dibawah ini merupakan penjelasan langkah demi langkah pencarian jalur terpendek.

1. Semisal kita menentukan node awal 1 dan node tujuan
2. Dijkstra akan melakukan kalkulasi terhadap setiap node yang terhubung dengan node 1, dan hasil yang didapatkan adalah node 2 karena bobot nilai node 2 paling kecil dibandingkan nilai pada node lain, nilai 0+7=7.
3. Node 2 diset sebagai node keberangkatan dan ditandai sebagai node yang telah dihitung. Dijkstra melakukan kalkulasi kembali terhadap nodenode, dan node 3 menjadi node keberangkatan selanjutnya karena bobotnya yang paling kecil 0+9=9.

4. Berlanjut dengan node 3 ditandai menjadi node yang telah terhitung. Dijkstra mengkalkulasi lagi mana node yang paling kecil jumlahnya. 6 memiliki nilai bobot terkeci l yaitu 2.
5. Dijkstra mengkalkulasi kembali dan menemukan node 5 (node tujuan) telah tercapai lewat node 6. Berarti jalur terpendekny adalah 1365 dan bobot yang didapat adalah 9+2+9=20. Bila node tujuan telah tercapai maka kalkulasi dijkstra dinyatakan se lesai.

B. Pseudocode Dijkstra's Algorithm

```
dijkstra(graf, titik_mulai, titik_akhir):
    inisialisasi jarak ke semua titik sebagai tak terhingga
    jarak = {titik: tak_terhingga for titik in graf}
    # jarak ke titik awal diatur sebagai 0
    jarak[titik_mulai] = 0
    # antrian prioritas yang berisi semua titik dengan jarak
terdekat yang diketahui
    antrian = [(0, titik_mulai)]
    while antrian is not empty:
        # ambil titik dengan jarak terdekat dari antrian prioritas
        (jarak_terdekat, titik_saat_ini) = ambil titik teratas
dari antrian
        # jika jarak terdekat ke titik saat ini lebih besar dari
jarak yang diketahui, lewati titik ini
        jika jarak_terdekat > jarak[titik_saat_ini]:
            continue
        # periksa tetangga dari titik saat ini dan update jarak ke
mereka jika lebih pendek
        for tetangga, bobot in graf[titik_saat_ini] di setiap
tetangga:
            jarak_tetangga = jarak_terdekat + bobot
            jika jarak_tetangga < jarak[tetangga]:
                jarak[tetangga] = jarak_tetangga
                tambahkan (jarak_tetangga, tetangga) ke antrian
    # kembalikan jarak terpendek ke titik akhir
    return jarak[titik_akhir]
```

Pseudocode tersebut memerlukan tiga input: graf (yang merupakan daftar tetangga untuk setiap titik pada grafik), titik_mulai (titik awal yang ingin dicari jarak terpendeknya), dan titik_akhir (titik akhir yang ingin dicapai).

Algoritma dimulai dengan menginisialisasi jarak ke semua titik sebagai tak terhingga. Kemudian, jarak ke titik_mulai diatur sebagai 0, karena titik awal harus memiliki jarak nol ke dirinya sendiri.

Selanjutnya, algoritma menggunakan antrian prioritas untuk mengatur titik-titik yang akan diperiksa. Antrian ini diawali dengan titik_mulai, dengan jarak awal 0. Pada setiap iterasi, algoritma mengambil titik dengan jarak terdekat dari antrian prioritas. Jika jarak terdekat ke titik saat ini lebih besar dari jarak yang diketahui, algoritma melewatkan titik ini. Jika tidak, algoritma memeriksa tetangga dari titik saat ini dan memperbarui jarak ke mereka jika lebih pendek.

Algoritma ini terus berjalan hingga tidak ada lagi titik yang harus diperiksa atau titik akhir sudah dicapai. Algoritma kemudian mengembalikan jarak terpendek ke titik akhir.

C. Source Code Dijkstra's Algorithm

Program ini membutuhkan modul heapq untuk mengimplementasikan antrian prioritas.

```
import heapq

def dijkstra(graf, titik_mulai, titik_akhir):
    # Inisialisasi jarak ke semua titik sebagai tak terhingga
    jarak = {titik: float('inf') for titik in graf}
    # Jarak ke titik awal diatur sebagai 0
    jarak[titik_mulai] = 0
    # Antrian prioritas yang berisi semua titik dengan jarak
terdekat yang diketahui
    antrian = [(0, titik_mulai)]
    while antrian:
        # Ambil titik dengan jarak terdekat dari antrian prioritas
        (jarak_terdekat, titik_saat_ini) = heapq.heappop(antrian)
        # Jika jarak terdekat ke titik saat ini lebih besar dari
jarak yang diketahui, lewati titik ini
        if jarak_terdekat > jarak[titik_saat_ini]:
            continue
        # Periksa tetangga dari titik saat ini dan update jarak ke
mereka jika lebih pendek
        for tetangga, bobot in graf[titik_saat_ini].items():
            jarak_tetangga = jarak_terdekat + bobot
            if jarak_tetangga < jarak[tetangga]:
                jarak[tetangga] = jarak_tetangga
                heapq.heappush(antrian, (jarak_tetangga,
tetangga))
    # Kembalikan jarak terpendek ke titik akhir
    return jarak[titik_akhir]

# Contoh penggunaan
graf = {
    'A': {'B': 4, 'C': 2},
    'B': {'C': 3, 'D': 2, 'E': 3},
    'C': {'B': 1, 'D': 4, 'E': 5},
    'D': {'E': 1},
    'E': {}
}
titik_mulai = 'A'
titik_akhir = 'E'
print("Jarak terpendek dari", titik_mulai, "ke", titik_akhir, "adalah",
dijkstra(graf, titik_mulai, titik_akhir))
```

```
PS C:\Users\Acoppz> & C:/Users/Acoppz/AppData/Local/Programs/Python/Python311/python.exe "d:/UNP/Semester 4/Perancangan dan Analisis Algoritma/code/dijkstra's_algorithm.py"
Jarak terpendek dari A ke E adalah 6
PS C:\Users\Acoppz>
```

Program ini mengambil kamus "graf" sebagai input, yang merepresentasikan grafik yang akan dihitung jarak terpendeknya. Setiap kunci dalam kamus mewakili titik dalam grafik, dan nilai kunci adalah kamus lain yang mewakili tetangga-tetangga dari titik tersebut beserta bobot dari setiap sisi yang menghubungkan mereka. Dalam contoh ini, grafik memiliki lima titik (A, B, C, D, E), dengan bobot yang berbeda-beda pada setiap sisi.

Program ini mengambil tiga input yaitu graf (yang merupakan daftar tetangga untuk setiap titik pada grafik), titik_mulai (titik awal yang ingin dicari jarak terpendeknya), dan titik_akhir (titik akhir yang ingin dicapai).

Program ini menggunakan struktur data antrian prioritas dari modul heapq pada Python untuk mengatur titik-titik yang akan diperiksa, dan menjalankan algoritma Dijkstra untuk menghitung jarak terpendek antara titik_mulai dan titik_akhir pada grafik yang diberikan.

Output dari program ini adalah jarak terpendek antara titik_mulai dan titik_akhir.

D. Analisis Kebutuhan Waktu Dijkstra's Algorithm

Ada beberapa kasus pencarian lintasan terpendek yang diselesaikan menggunakan algoritma Dijkstra, yaitu: pencarian lintasan terpendek antara dua buah simpul tertentu (a pair shortest path), pencarian lintasan terpendek antara semua pasangan simpul (all pairs shortest path), pencarian lintasan terpendek dari simpul tertentu ke semua simpul yang lain (single-source shortest path), serta pencarian lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (intermediate shortest path).

1. Analisis waktu Dijkstra's Algorithm berdasarkan operasi/instruksi yang dieksekusi, sebagai berikut:
    a) Memeriksa simpul dengan jarak terpendek yang diketahui dari simpul awal
    b) Menandai simpul tersebut sebagai "terlihat"
    c) Untuk setiap simpul tetangga dari simpul terlihat, hitung jarak total melalui simpul terlihat dan bandingkan dengan jarak terpendek yang saat ini diketahui dari simpul awal ke simpul tetangga tersebut
    d) Jika jarak total lebih pendek dari jarak terpendek yang diketahui saat ini, perbarui jarak terpendek yang diketahui dan catat jalur terpendek yang baru

e) Ulangi proses untuk simpul dengan jarak terpendek yang diketahui sampai sampai semua simpul telah diproses

Waktu eksekusi Dijkstra's Algorithm adalah O(E + V log V), di mana E adalah jumlah tepi dan V adalah jumlah simpul dalam grafik. E + V log V adalah jumlah operasi dasar yang dilakukan dalam algoritma, seperti perbandingan, penambahan, dan pengurangan. Namun, implementasi spesifik dari algoritma dapat memerlukan lebih banyak atau lebih sedikit operasi tergantung pada faktor seperti struktur data yang digunakan untuk merepresentasikan grafik dan simpul-simpulnya.

2. Analisis waktu Dijkstra's Algorithm berdasarkan jumlah operasi abstrak, sebagai berikut:
    a) Inisialisasi: O(V)
        - Menginisialisasi nilai jarak awal dari simpul awal ke semua simpul lainnya ke tak hingga
        - Menandai simpul awal sebagai "terlihat"
    b) Mencari jalur terpendek: O((V+E) log V)
        - Pada setiap iterasi, memilih simpul terlihat dengan jarak terpendek dari simpul awal (O(log V) operasi menggunakan antrian prioritas seperti heap)
        - Memeriksa semua tetangga simpul terlihat (total E tetangga) dan memperbarui jarak terpendek ke tetangga jika diperlukan (O(1) operasi untuk setiap tetangga)
        - Menandai simpul terlihat sebagai "ditemukan"
        - Mengulangi proses ini sampai semua simpul telah ditemukan
    c) Rekonstruksi jalur terpendek: O(E)
        - Memulai dari simpul tujuan dan mengikuti jalur terpendek ke simpul awal

3. Analisis waktu Dijkstra's Algorithm berdasarkan pada pendekatan best-case, worst-case, dan average-case, sebagai berikut:
    a) Best-case: Waktu eksekusi terbaik terjadi ketika simpul tujuan langsung terhubung ke simpul awal. Waktu eksekusi dalam kasus ini adalah O(E log V), di mana E adalah jumlah tepi dan V adalah jumlah simpul dalam grafik.
    b) Worst-case: Waktu eksekusi terburuk terjadi ketika grafik membentuk urutan berurutan. Dalam kasus ini, waktu eksekusi Dijkstra's Algorithm adalah O(V^2), di mana V adalah jumlah simpul dalam grafik.
    c) Average-case: Waktu eksekusi rata-rata tergantung pada struktur grafik dan bisa jauh lebih cepat dibandingkan dengan worst-case. Dalam kasus ini, waktu eksekusi Dijkstra's Algorithm biasanya adalah O((V+E) log V), dengan kompleksitas waktu tergantung pada jumlah simpul dan tepi dalam grafik.

Kesimpulannya, Dijkstra's Algorithm adalah algoritma yang efektif untuk mencari jalur terpendek dalam grafik yang memiliki struktur acak atau tidak teratur. Namun, waktu eksekusi dapat sangat lambat dalam kasus worst-case, sehingga perlu dipertimbangkan secara cermat ketika digunakan dalam grafik yang besar dan kompleks.

E. Referensi

KURNIYANTO, B. (2019). SKRIPSI. *SISTEM INFORMASI PARIWISATA PENCARIAN LOKASI DI WILAYAH BOROBUDUR MENGG PRO UNAKAN ALGORITMA DIJKSTRA*, 7-8.

Novandi, R. A. (2007). MAKALAH IF2251 STRATEGI ALGORITMIK. *Perbandingan Algoritma Dijkstra dan Algoritma Floyd-Warshall dalam Penentuan Lintasan Terpendek (Single Pair Shortest Path)*, 2.

Primadasa, Y. (2015). STUDI KASUS PT.COCA-COLA KOTA PADANG. *PENCARIAN RUTE TERPENDEK MENGGUNAKAN ALGORITMA DIJKSTRA PADA SIG BERBASIS WEB UNTUK DISTRIBUSI MINUMAN* , 47.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

Kleinberg, J., & Tardos, E. (2006). Algorithm design (1st ed.). Addison-Wesl