Single-source Shortest Path Problem

Mata Kuliah: Algoritma & Pemrograman 2

ITDev 4.0 – 22 Mei 2021

Hilman F. Rakhmad
hifra@pemro.id

Definisi

- Bila terdapat sebuah graf dan kita diberi dua node A dan B dari graf tersebut, jalur manakah yang terpendek untuk menghubungkan kedua node tersebut?
- Masalah ini biasa digambarkan menggunakan graf, baik terarah maupun tak terarah

Implementasi di Dunia Nyata

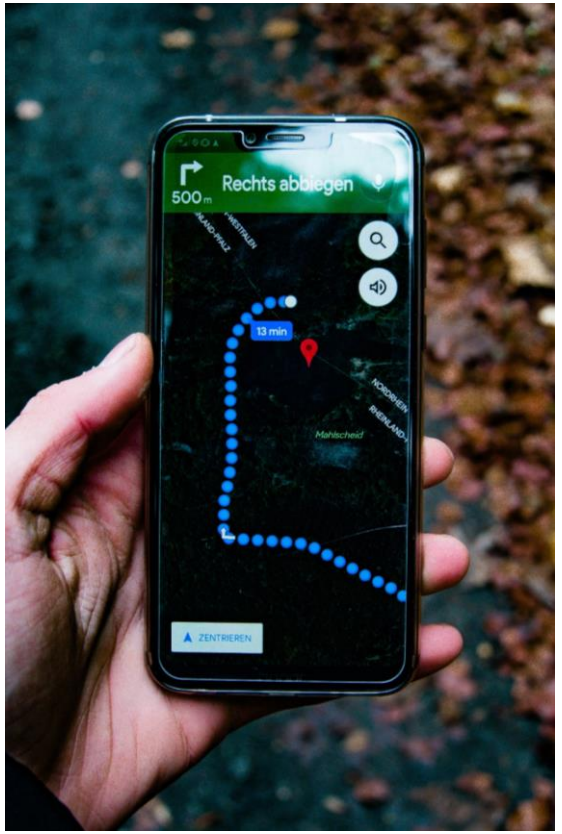


Photo by [Julian Peter](#) from [Pexels](#)

Jenis-jenis Shortest Path Problem

- Single-source
  - Mencari jalur terpendek dari **satu node asal** ke node tujuan.
  - Node tujuan dapat berupa semua node lain yang berada pada satu graf
  - Node tujuan juga dapat berupa satu node lain yang terdapat pada graf
- All-pairs
  - Mencari jalur terpendek dari **setiap node asal** ke **setiap node lain** pada satu graf

Algoritma Pencarian Shortest Path

- **Dijkstra's Algorithm**
- Bellman-Ford Algorithm
- Topological Sort
- Floyd-Warshall Algorithm
- Johnson's Algorithm

Dijkstra’s Algorithm

- Disusun oleh ilmuwan komputer **Edsger W. Dijkstra** pada tahun 1956 dan diterbitkan tiga tahun kemudian.
- Algoritme asli Dijkstra mencari jalur terpendek antara dua node yang diberikan
- Kemudian terdapat varian yang lebih umum yang mencari jalur terpendek dari satu node sumber ke semua node lain dalam graf

Algoritme:

1. Tandai setiap node sebagai belum dikunjungi (**unvisited**)
2. Tentukan node asal (**origin**)
3. Setiap node akan dicatat jarak terpendek dari node asal (**shortest distance from origin**) beserta node sebelumnya yang harus dilewati (**previous node**)
4. Beri setiap node nilai jarak sementara:
   - Beri nilai 0 untuk node asal (**origin**)
   - Beri nilai *infinite* untuk node lainnya
5. Jadikan **origin** sebagai node yang akan dicek (**current node**)

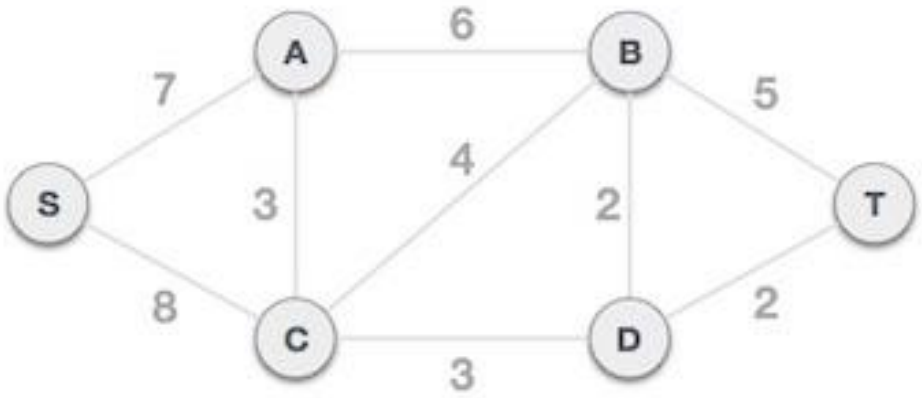
6. Untuk **current node** saat ini, cari semua tetangga (**neighbor**) yang belum pernah dikunjungi (**unvisited**) dan hitung jarak sementaranya bila melewati **current node**, kemudian bandingkan jarak ini dengan jarak yang sudah tercatat pada tabel jarak terpendek. Bila jarak yang saat ini dihitung lebih kecil maka ubah nilai pada tabel dan catat **current node** sebagai **previous node**-nya.

   Sebagai contoh, misal **current node** adalah A dan **neighbor** adalah B. Bila nilai A pada tabel jarak terpendek adalah 6 dan jarak dari A ke B adalah 2, maka nilai jarak sementara B adalah 6 + 2 = 8. Bandingkan dengan nilai B pada tabel jarak terpendek. Bila nilai 8 lebih kecil daripada jarak terpendek saat ini maka ubah nilai B pada tabel dan catat A sebagai **previous node**.

7. Bila semua **neighbor** yang **unvisited** sudah dicari jarak sementaranya, ubah tanda **current node** menjadi **visited**. Node yang **visited** tidak akan dicek lagi.
8. Bila semua node (atau node tujuan, bila hanya mencari satu node tujuan) sudah ditandai **visited** atau nilai sementara terkecil dari semua node **unvisited** adalah *infinite* (terjadi bila node yang belum dikunjungi tidak terhubung dengan graf node asal), maka hentikan proses. Algoritma sudah selesai.
9. Jika tidak, pilih node **unvisited** dengan nilai terkecil sebagai **current node** baru, dan kembali ke langkah 6.

Ilustrasi Dijkstra's Algorithm

- Diketahui sebuah graf sebagai berikut



- Dengan S sebagai node asal, cari jarak terpendek menuju node lainnya!

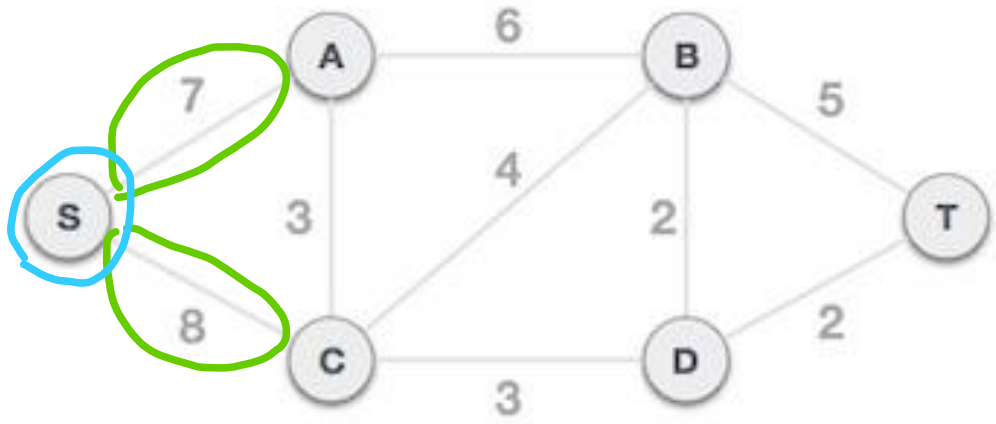
- Siapkan tabel **Unvisited**, **Visited**, dan **Jarak Terpendek**, serta inisiasi nilai awal ke semua elemen pada tabel **Jarak Terpendek**.

Unvisited
S
A
B
C
D
T

Visited

Shortest Distance		
Node	**Distance**	**Previous Node**
S	0	None
A	Inf	None
B	Inf	None
C	Inf	None
D	Inf	None
T	Inf	None

- Pengecekan pertama dilakukan terhadap neighbor dari node S. Karena node yang dicek masih node asal, maka untuk menghitung jarak terpendeknya cukup melihat biaya edge menuju neighbor tersebut



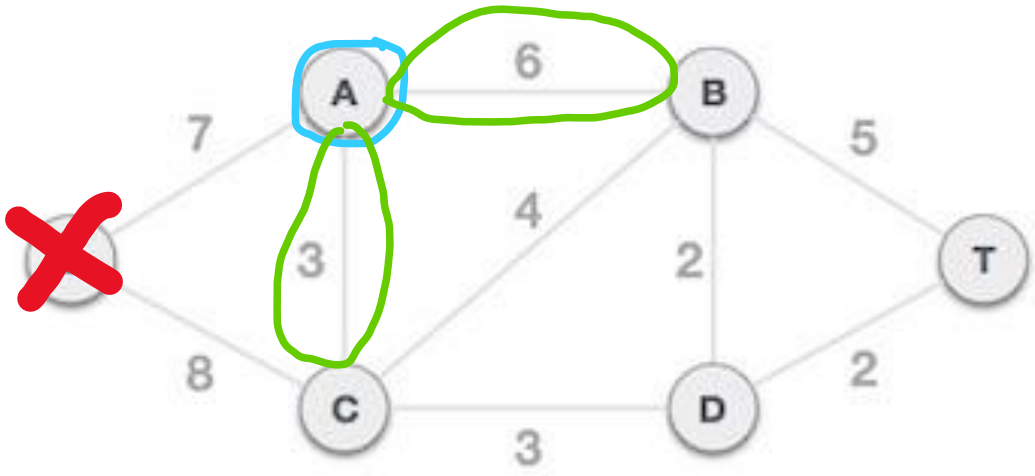
Unvisited
A
B
C
D
T

Visited
S

Shortest Distance		
Node	**Distance**	**Previous Node**
S	0	None
A	7	S
B	Inf	None
C	8	S
D	Inf	None
T	Inf	None

Current node berikutnya diambil dari node **Unvisited** dengan **Distance** terkecil pada tabel **Shortest Distance** yaitu A

- Terdapat dua node yang dapat dicari jaraknya dari A:
  - B (7 + 6 = 13)
  - C (7 + 3 = 10)
- Yang dapat ditambahkan ke tabel hanya node B, karena nilai node C pada tabel lebih kecil (8)



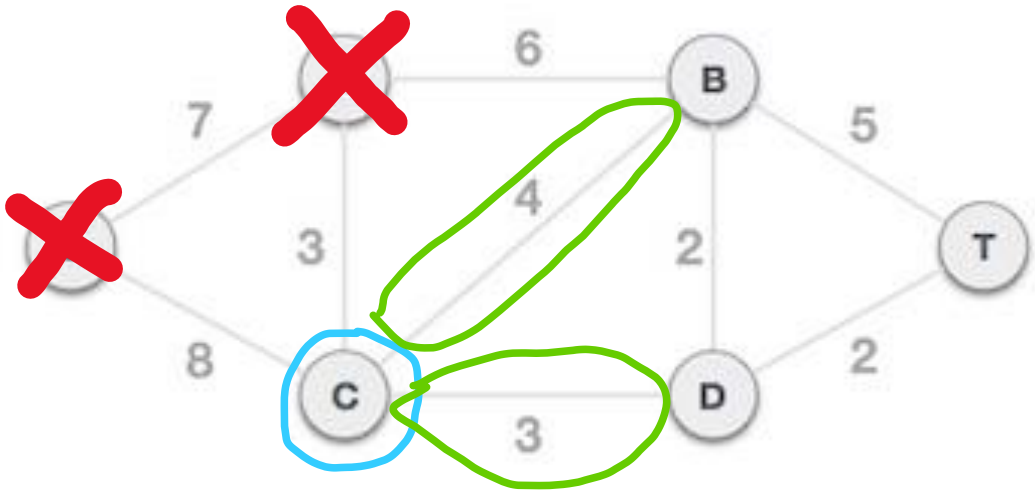
Unvisited
B
C
D
T

Visited
S
A

Shortest Distance		
Node	**Distance**	**Previous Node**
S	0	None
A	7	S
B	13	A
C	8	S
D	Inf	None
T	Inf	None

Current node berikutnya diambil dari node **Unvisited** dengan **Distance** terkecil pada tabel **Shortest Distance** yaitu C

- Terdapat dua node yang dapat dicari jaraknya dari C:
  - B (8 + 4 = 12)
  - D (8 + 3 = 11)
- Node B dapat ditambahkan ke tabel karena nilai B pada tabel lebih besar (13), begitu pun dengan node D (inf)



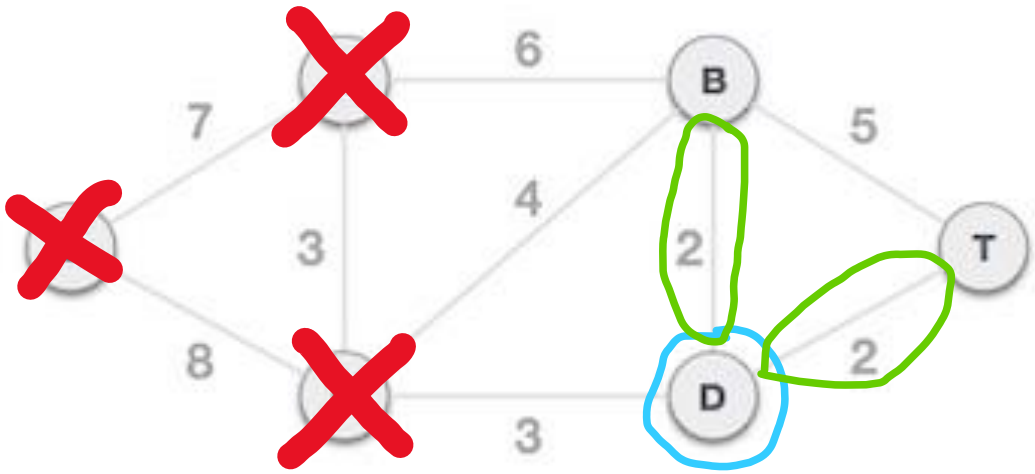
Unvisited
B
D
T

Visited
S
A
C

Shortest Distance		
Node	**Distance**	**Previous Node**
S	0	None
A	7	S
B	12	C
C	8	S
D	11	C
T	Inf	None

Current node berikutnya diambil dari node **Unvisited** dengan **Distance** terkecil pada tabel **Shortest Distance** yaitu D

- Terdapat dua node yang dapat dicari jaraknya dari D:
  - B (11 + 2 = 13)
  - T (11 + 2 = 13)
- Node B tidak dapat ditambahkan ke tabel karena nilai B pada tabel lebih kecil (12)
- Node T dapat ditambahkan ke tabel karena nilai T pada tabel lebih besar (inf)



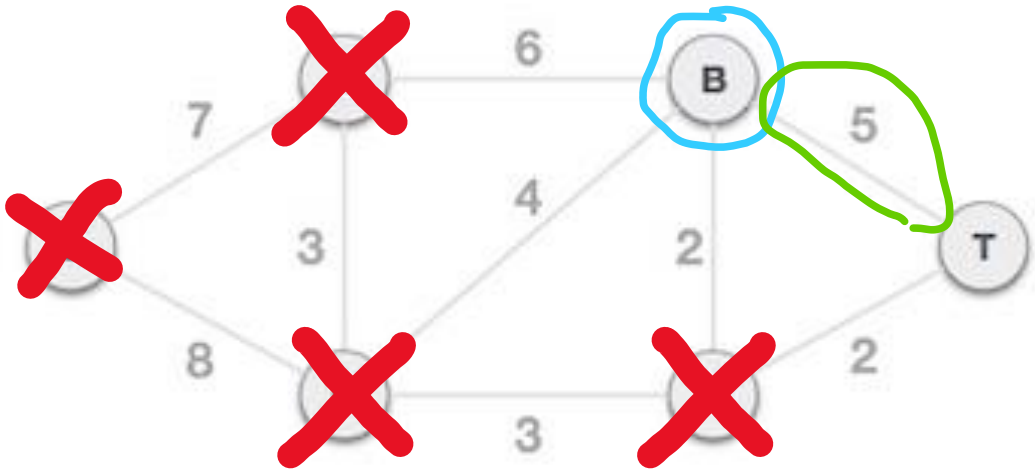
Unvisited
B
T

Visited
S
A
C
D

Shortest Distance		
Node	**Distance**	**Previous Node**
S	0	None
A	7	S
B	12	C
C	8	S
D	11	C
T	13	D

Current node berikutnya diambil dari node **Unvisited** dengan **Distance** terkecil pada tabel **Shortest Distance** yaitu B

- Terdapat satu node yang dapat dicari jaraknya dari B:
  - T (12 + 5 = 17)
- Node T tidak dapat ditambahkan ke tabel karena nilai T pada tabel lebih kecil (13)



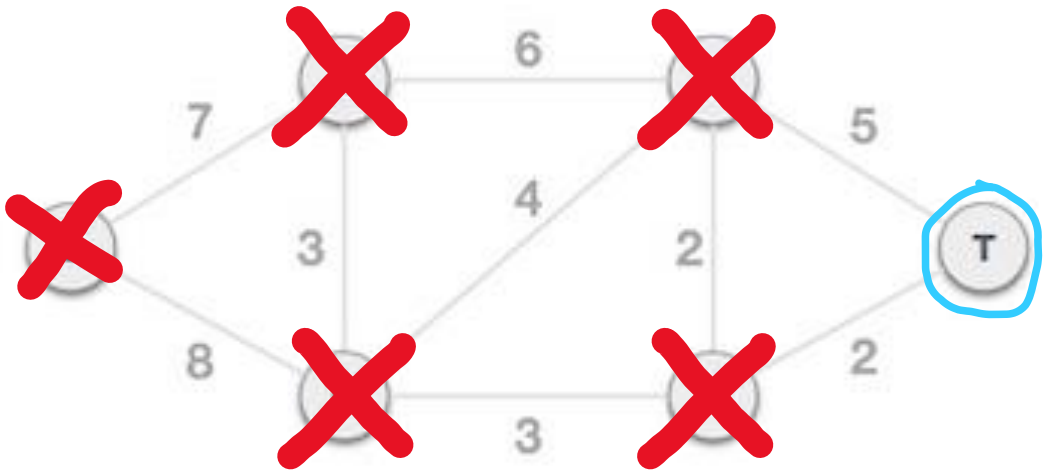
Unvisited
T

Visited
S
A
C
D
B

Shortest Distance		
Node	**Distance**	**Previous Node**
S	0	None
A	7	S
B	12	C
C	8	S
D	11	C
T	13	D

Current node berikutnya diambil dari node **Unvisited** dengan **Distance** terkecil pada tabel **Shortest Distance** yaitu T

- Tidak ada node tetangga T yang dapat dicari jarak sementaranya, karena semua node tetangga T sudah pernah dikunjungi (**visited**)



Unvisited

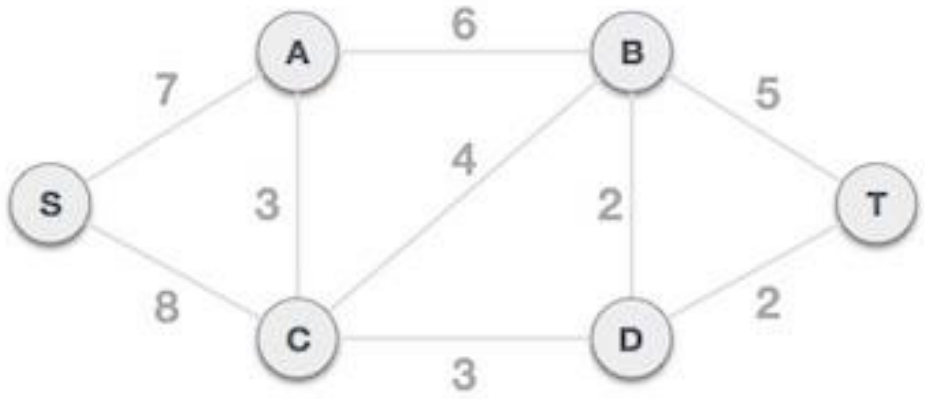
Visited
S
A
C
D
B
T

Shortest Distance		
Node	**Distance**	**Previous Node**
S	0	None
A	7	S
B	12	C
C	8	S
D	11	C
T	13	D

Sudah tidak ada node yang belum dikunjungi (**Unvisited**), maka algoritma ini telah selesai.

- Solusi Shortest Path yang didapatkan, dengan node S sebagai node asal

Shortest Distance		
Node	**Distance**	**Previous Node**
S	0	None
A	7	S
B	12	C
C	8	S
D	11	C
T	13	D



Contoh Kode: Dijkstra’s Algorithm

- https://github.com/hifra01/itdev-2021-algo/blob/main/Single-source%20Shortest%20Path/Dijkstra.py

Referensi

- Benjamin Baka. 2017. Python Data Structures and Algorithms: Improve application performance with graphs, stacks, and queues. Packt Publishing.
- https://brilliant.org/wiki/shortest-path-algorithms/
- https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm/