Minimum Spanning Tree

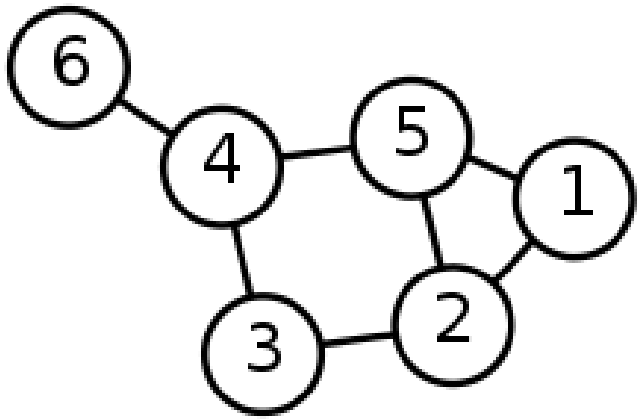
Mata Kuliah: Algoritma & Pemrograman 2

ITDev 4.0 – 22 Mei 2021

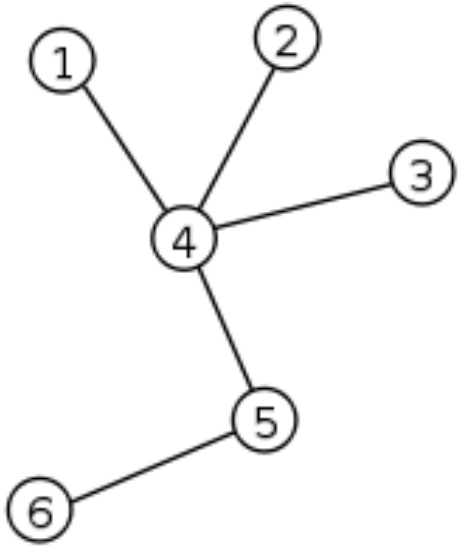
Hilman F. Rakhmad
hifra@pemro.id

Definisi

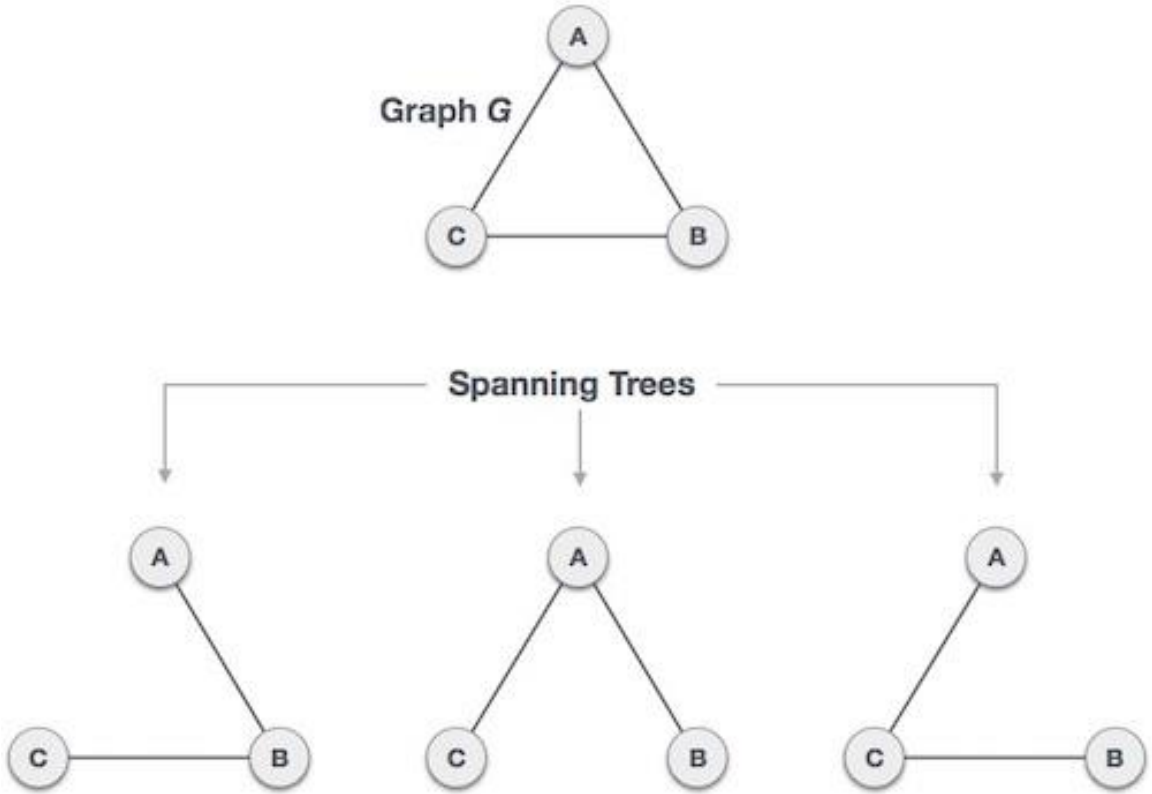
Graf: himpunan yang terdiri dari kumpulan node (vertex) dan edge (link)



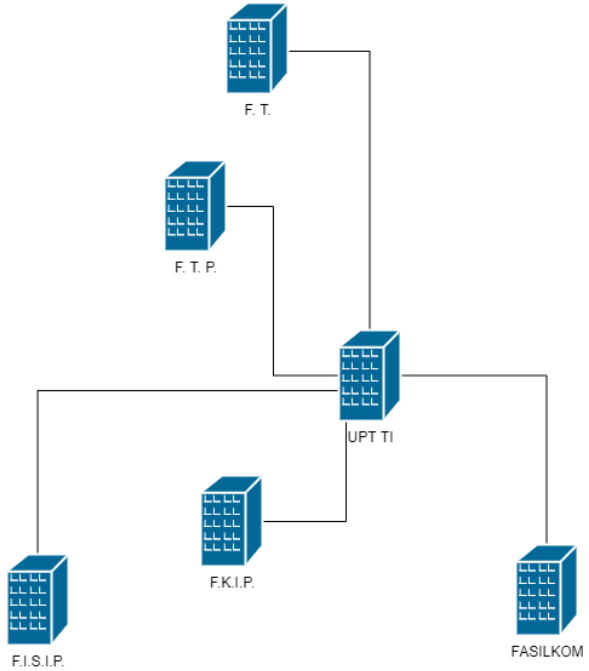
Tree: graf terhubung yang tidak memiliki arah dan tidak mengandung siklis

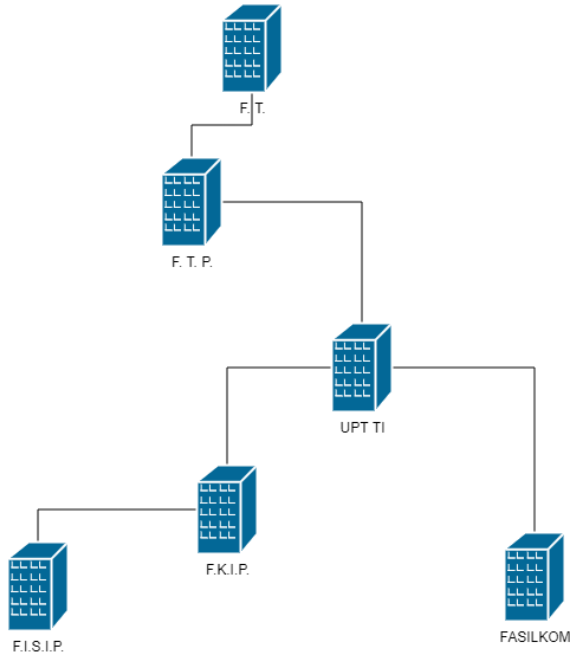


- Spanning Tree: subset dari sebuah graf di mana semua node/vertex terhubung dengan jumlah edge/link seminimal mungkin
- Minimum Spanning Tree: Spanning Tree dengan bobot/biaya paling kecil (minimal)



Implementasi di Dunia Nyata





- Kebutuhan kabel lebih panjang = Lebih mahal
- Biaya perawatan lebih mahal

- Kebutuhan kabel lebih pendek = Lebih murah
- Biaya perawatan lebih murah

Algoritme Penyelesaian

1. Prim’s Spanning Tree Algorithm
2. Kruskal’s Spanning Tree Algorithm

- Kedua algoritme tersebut menerapkan pendekatan greedy
- Pendekatan greedy mencari nilai optimal pada setiap langkahnya untuk mencari nilai optimal keseluruhan

Prim’s Spanning Tree Algorithm

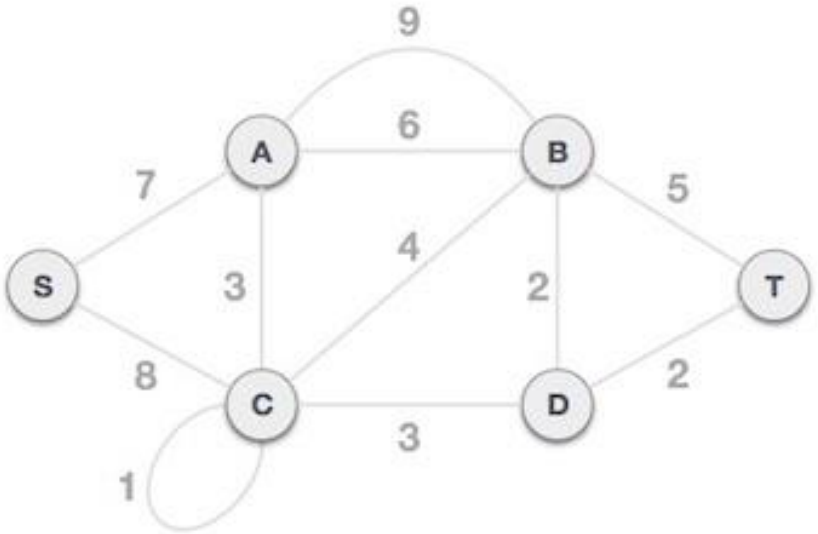
- Ditemukan pada 1930 oleh **Vojtěch Jarník**, dan ditemukan kembali oleh computer scientist **Robert C. Prim** pada 1957 dan **Dijkstra** pada 1959.
- Pencarian dimulai dari satu node akar/sumber, kemudian menambahkan node-node terdekat hingga semua node terpilih

Algoritme:

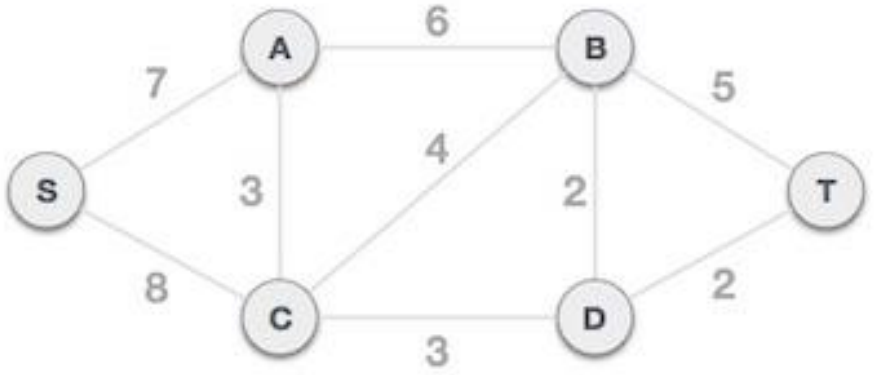
1. Pastikan pada graph tidak ada loop atau edge yang paralel
2. Siapkan daftar node yang belum dikunjungi (**unvisited**) dan yang telah dikunjungi (**visited**), serta daftar **solusi**
3. Masukkan semua node ke dalam daftar **unvisited**
4. Pilih satu node sebagai node akar/sumber (root/source)
5. Pindahkan node akar dari **unvisited** ke **visited**
6. Dari setiap node di daftar **visited**, cari edge menuju node yang belum dikunjungi dengan nilai paling minimal
7. Catat node asal dan tujuan dari edge yang terpilih beserta biayanya ke **solusi**
8. Pindahkan node tujuan ke daftar **visited**
9. Ulangi langkah 6 hingga 8 hingga semua node telah dikunjungi (daftar **unvisited** kosong)
10. Cetak daftar edge solusi beserta biaya total

Ilustrasi Prim’s Algorithm

- Diketahui graf sebagai berikut:



- Bersihkan graf dari loop dan edge paralel sehingga menjadi seperti berikut:



- Pilih node akar/sumber. Di sini kita akan menggunakan node S

- Siapkan daftar/tabel **Unvisited**, **Visited**, dan **Solusi**. Masukkan semua node ke tabel **Unvisited**

Unvisited
S
A
B
C
D
T

Visited

Solusi		
Node Asal	**Node Tujuan**	**Biaya**

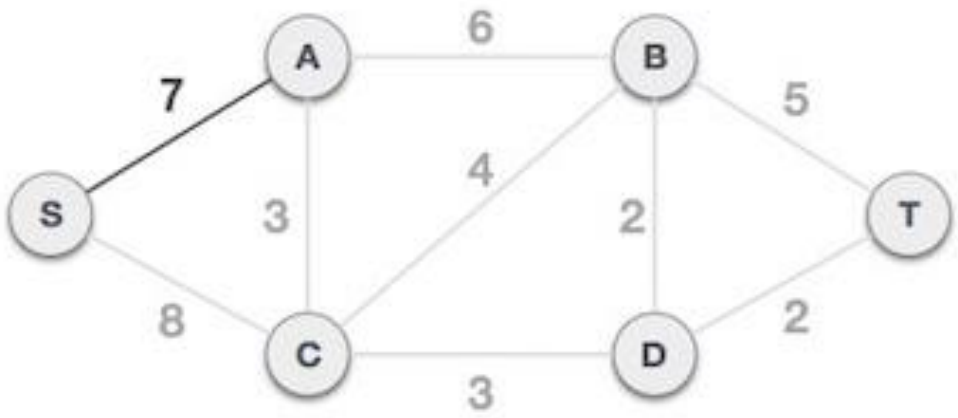
- Pindahkan node S sebagai node akar/sumber ke tabel **Visited**

Unvisited
A
B
C
D
T

Visited
S

Solusi		
Node Asal	**Node Tujuan**	**Biaya**

- Node S memiliki dua edge yang terhubung:
  - S – A dengan biaya 7
  - S – C dengan biaya 8
- Pilih edge dengan biaya minimal, yaitu edge S – A
- Masukkan edge tersebut ke tabel **Solusi**, dan masukkan node A ke tabel **Visited**

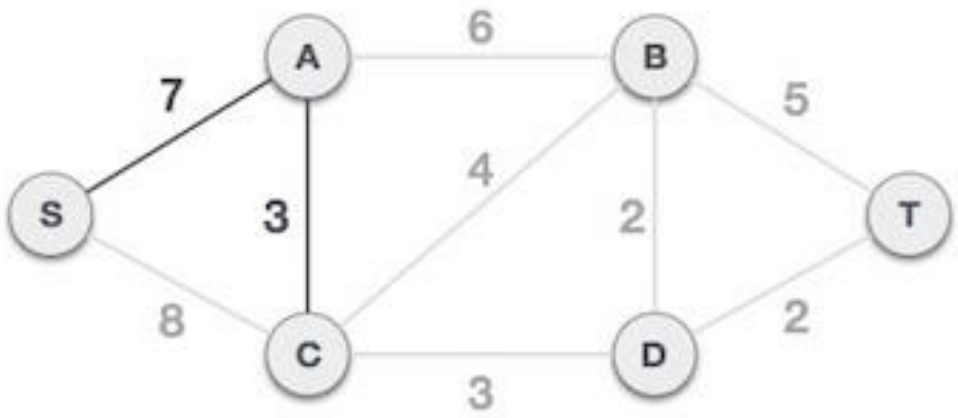


Unvisited
B
C
D
T

Visited
S
A

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
S	A	7

- Dari semua node pada tabel **Visited**, daftar edge yang tersedia dan tidak membentuk siklis:
  - S – C dengan biaya 8
  - A – B dengan biaya 6
  - A – C dengan biaya 3
- Pilih edge dengan biaya minimal, yaitu edge A – C
- Masukkan edge tersebut ke tabel **Solusi**, dan masukkan node C ke tabel **Visited**

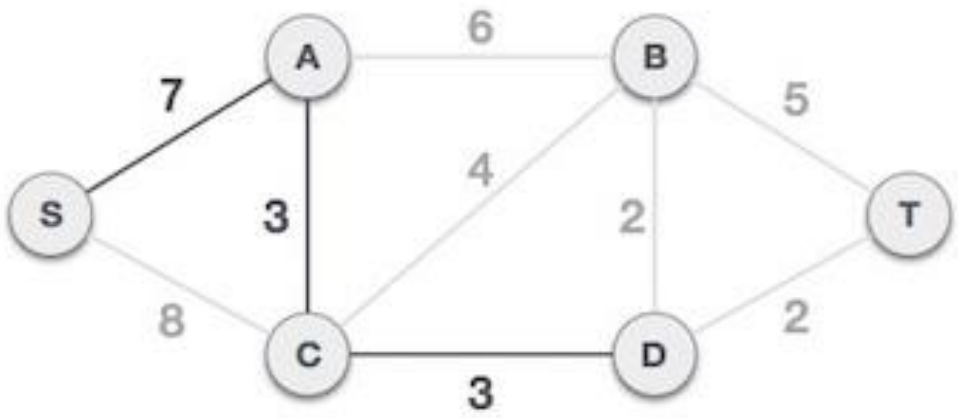


Unvisited
B
D
T

Visited
S
A
C

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
S	A	7
A	C	3

- Dari semua node pada tabel **Visited**, daftar edge yang tersedia dan tidak membentuk siklis :
  - A – B dengan biaya 6
  - C – B dengan biaya 4
  - C – D dengan biaya 3
- Pilih edge dengan biaya minimal, yaitu edge C – D
- Masukkan edge tersebut ke tabel **Solusi**, dan masukkan node D ke tabel **Visited**

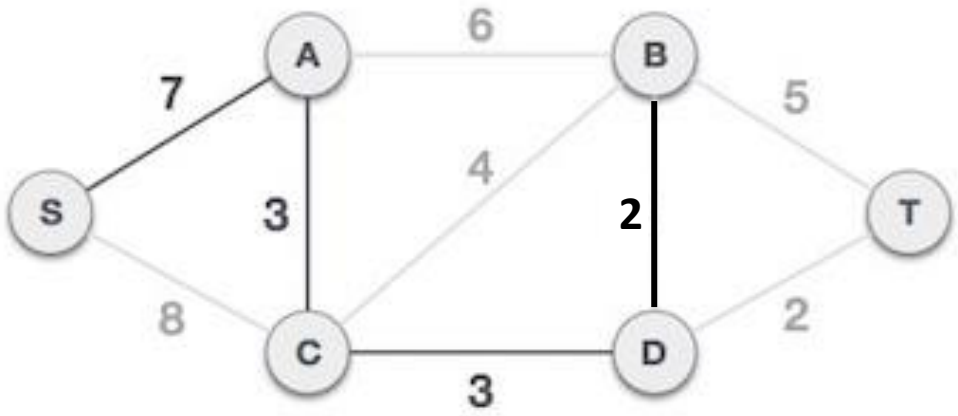


Unvisited
B
T

Visited
S
A
C
D

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
S	A	7
A	C	3
C	D	3

- Dari semua node pada tabel **Visited**, daftar edge yang tersedia dan tidak membentuk siklis :
  - A – B dengan biaya 6
  - C – B dengan biaya 4
  - D – B dengan biaya 2
  - D – T dengan biaya 2
- Pilih edge dengan biaya minimal, yaitu edge D – B
- Masukkan edge tersebut ke tabel **Solusi**, dan masukkan node B ke tabel **Visited**

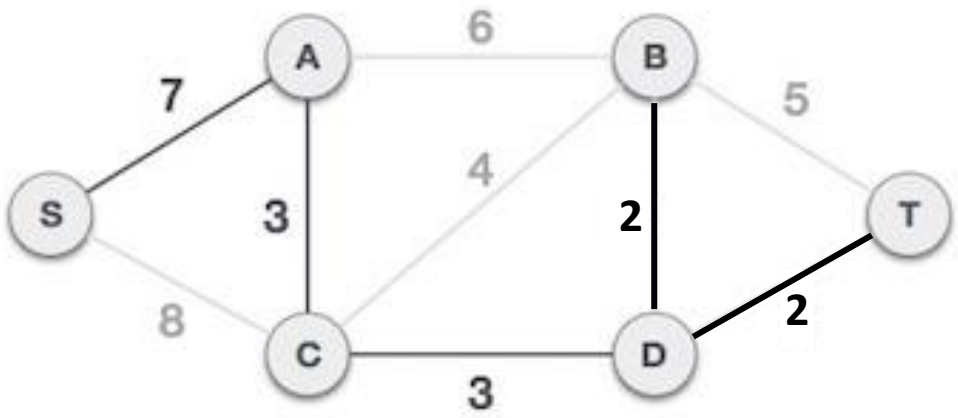


Unvisited
T

Visited
S
A
C
D
B

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
S	A	7
A	C	3
C	D	3
D	B	2

- Dari semua node pada tabel **Visited**, daftar edge yang tersedia dan tidak membentuk siklis :
  - B – T dengan biaya 5
  - D – T dengan biaya 2
- Pilih edge dengan biaya minimal, yaitu edge D – T
- Masukkan edge tersebut ke tabel **Solusi**, dan masukkan node T ke tabel **Visited**

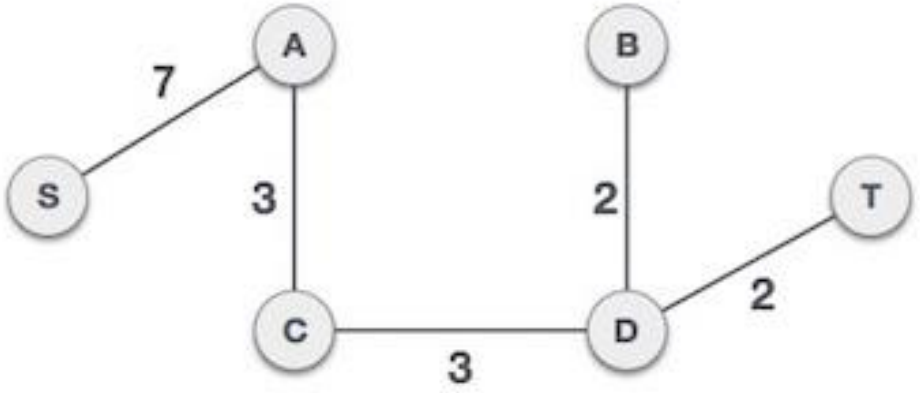


Unvisited

Visited
S
A
C
D
B
T

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
S	A	7
A	C	3
C	D	3
D	B	2
B	T	2

- Karena tabel **Unvisited** sudah kosong, maka program sudah selesai. Hasil yang didapatkan adalah sebagai berikut:



Solusi		
Node Asal	**Node Tujuan**	**Biaya**
S	A	7
A	C	3
C	D	3
D	B	2
B	T	2
Total biaya		17

Contoh Kode: Prim's Algorithm

- https://github.com/hifra01/itdev-2021-algo/blob/main/Minimum%20Spanning%20Tree/Prim.py

Kruskal’s Spanning Tree Algorithm

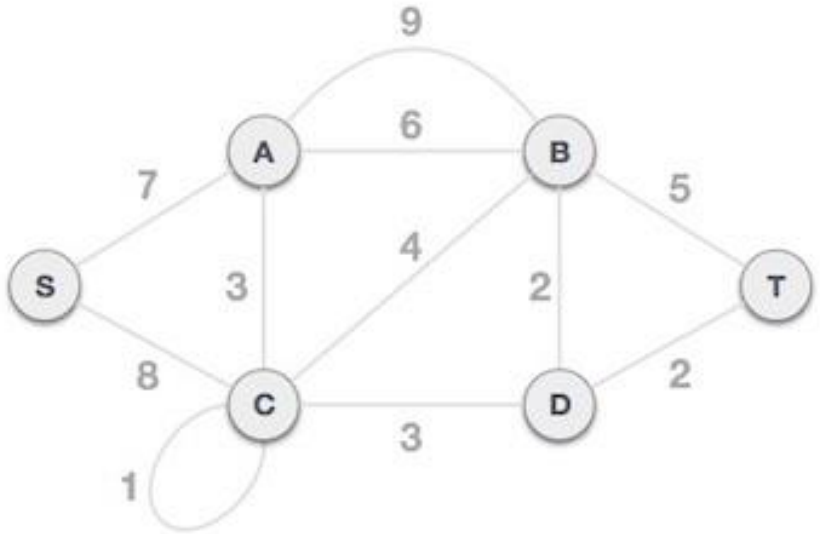
- Pertama kali muncul di Proceedings of the American Mathematical Society, pp. 48–50 pada tahun 1956, ditulis oleh **Joseph Kruskal**
- Pencarian dimulai dengan mengurutkan semua edge yang ada mulai dari yang terkecil hingga yang terbesar, kemudian mengambil edge dengan nilai minimal hingga semua node dikunjungi

Algoritme:

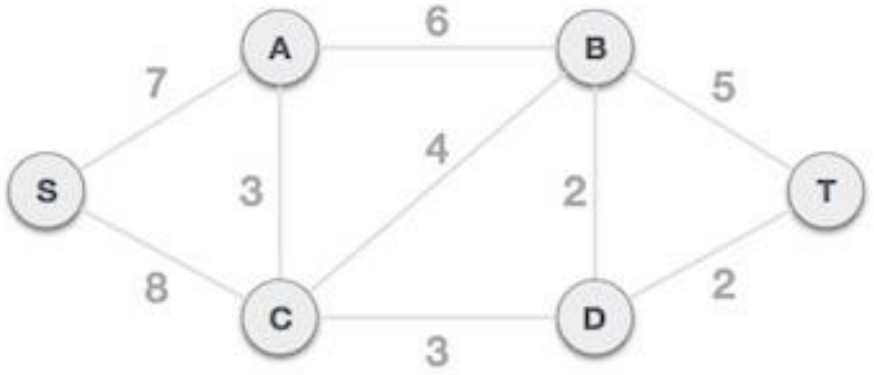
1. Pastikan pada graph tidak ada loop atau edge yang paralel
2. Catat semua edge beserta costnya, urutkan dari cost terkecil ke cost terbesar
3. Lakukan pengecekan pada edge pertama pada daftar
4. Jika edge tersebut tidak membuat siklis masukkan edge tersebut ke solusi, selain itu maka lewati.
5. Jika jumlah edge solusi sudah mencapai (Jumlah_Node – 1), maka algoritma selesai. Jika tidak, kembali ke langkah 4 dengan edge berikutnya untuk dicek
6. Cetak daftar edge solusi beserta biaya total

Ilustrasi Kruskal's Algorithm

- Diketahui graf sebagai berikut:



- Bersihkan graf dari loop dan edge paralel sehingga menjadi seperti berikut:

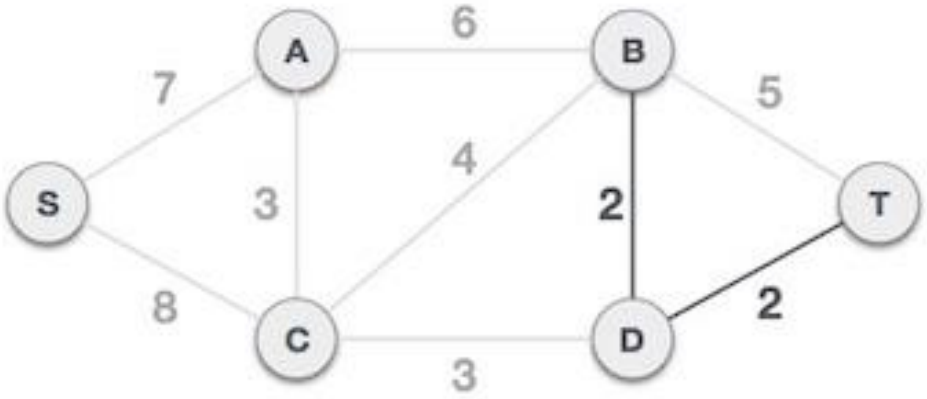


- Siapkan daftar/tabel **Daftar Edge** dan **Solusi**.

Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8

Solusi		
Node Asal	**Node Tujuan**	**Biaya**

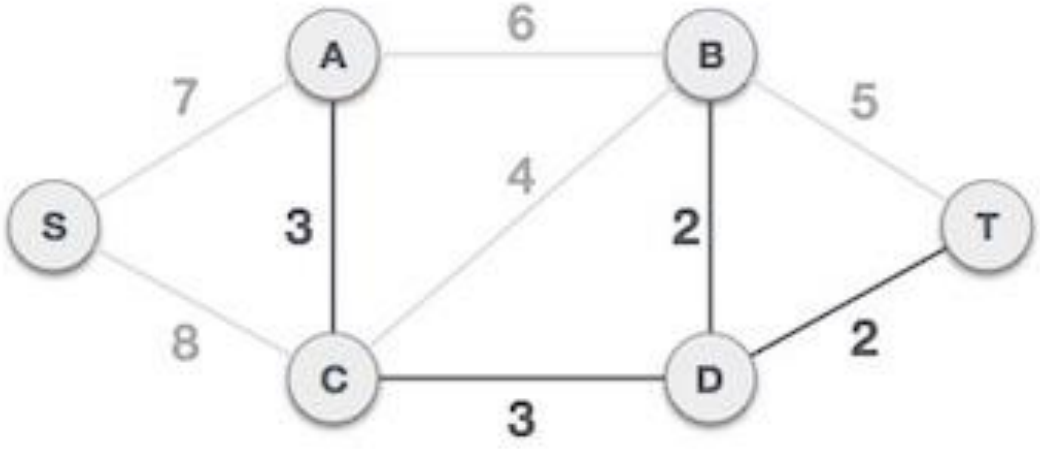
- Edge pertama pada tabel adalah B – D dengan biaya 2. Edge B – D dapat ditambahkan karena belum ada edge solusi lain yang dapat membentuk siklis
- Edge kedua pada tabel adalah D – T dengan biaya 2. Edge D – T juga dapat ditambahkan karena tidak membentuk siklis



Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2

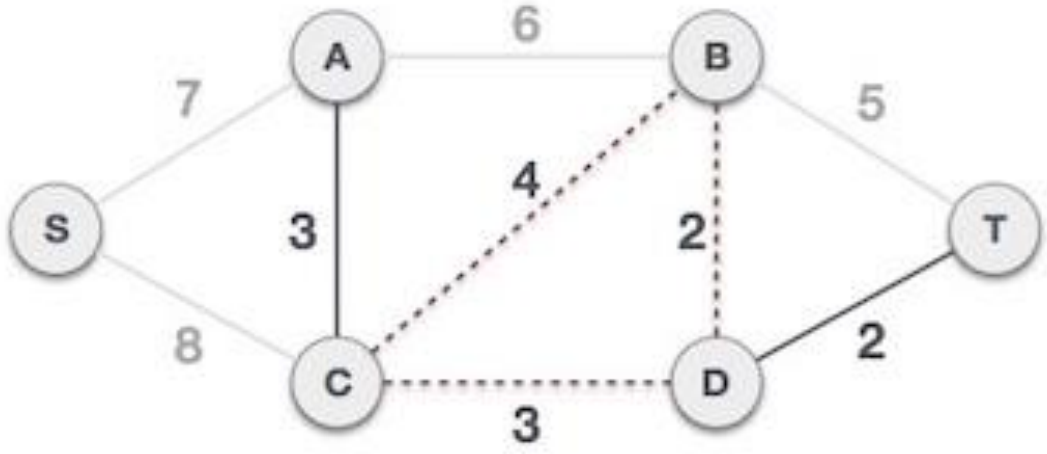
- Berikutnya ada dua edge yang memiliki biaya 3, yaitu A – C dan C – D. Kedua edge ini tidak membentuk siklis dengan edge solusi yang sudah ada sehingga dapat dimasukkan ke tabel solusi



Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3

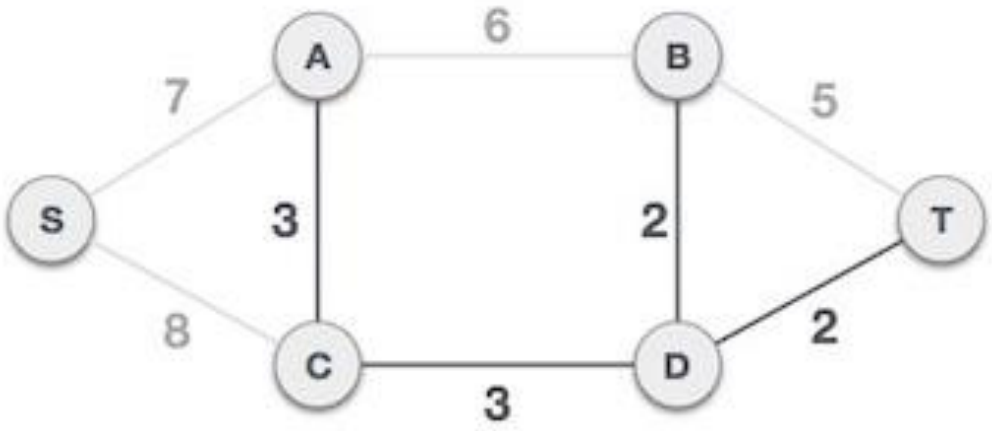
- Kemudian ada edge C – B dengan biaya 4. Edge ini bila ditambahkan sebagai solusi akan terjadi siklis C – B – D, sehingga edge ini tidak dapat ditambahkan sebagai solusi



Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	T	5
A	B	6
S	A	7
S	C	8

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3

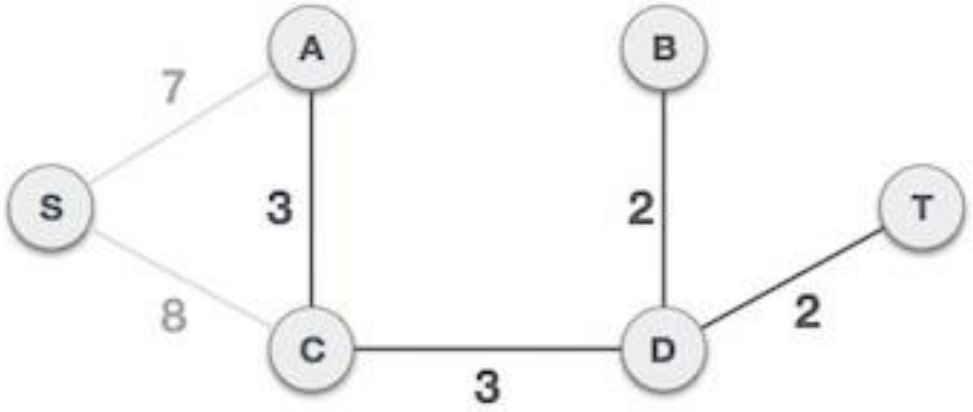
- Edge B – T dengan biaya 5 dan edge A – B dengan biaya 6 juga tidak dapat ditambahkan ke tabel solusi karena akan membentuk siklis



Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
S	A	7
S	C	8

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3

- Tersisa satu node (S) untuk ditambahkan, dan ada dua edge yang terhubung ke node S:
  - S – A dengan biaya 7
  - S – B dengan biaya 8
- Seperti langkah-langkah sebelumnya, biaya terendah akan dipilih terlebih dahulu, sehingga edge yang ditambahkan ke solusi adalah edge S – A.

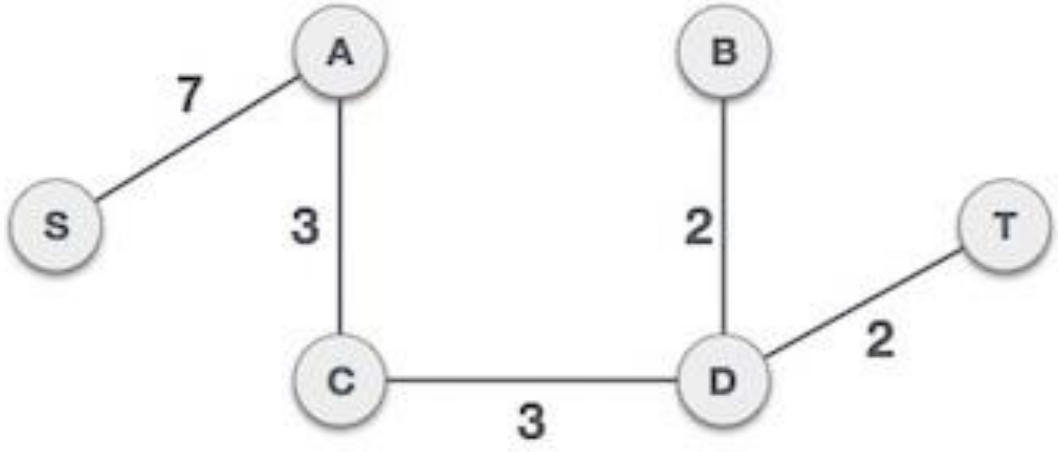


Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
S	A	7

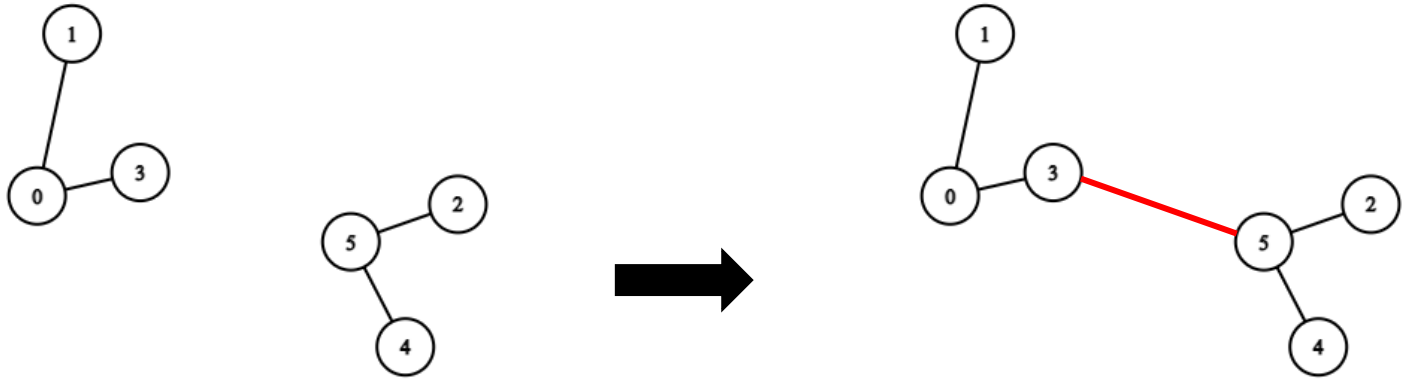
- Karena tidak ada lagi edge yang dapat dicek, maka ditemukanlah solusi minimum spanning tree menggunakan algoritme Kruskal.

Solusi		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
S	A	7



Cara Mencari Siklis pada Algoritme Kruskal

- Pada algoritme Kruskal, akan ada kemungkinan terbentuk dua Tree yang terpisah, sebelum akhirnya kedua Tree tersebut digabung.



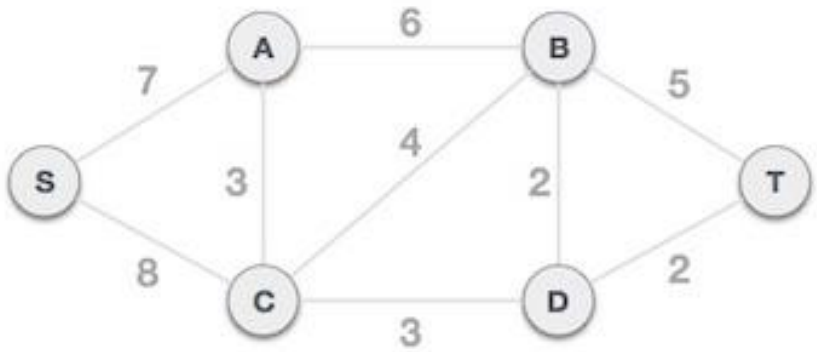
- Untuk mengecek siklis tidak memungkinkan untuk memakai metode **visited-unvisited** seperti Prim, karena 3 dan 5 sama-sama sudah pernah dikunjungi sebelum mereka terhubung

- Solusinya dapat menggunakan algoritme **Find-Union**
- Setiap node memiliki **Parent** dan **Rank**, yang digunakan untuk mencari siklis
- Setiap node awalnya akan dianggap sebagai sebuah graf tersendiri dengan anggota hanya node itu sendiri
- Karena hanya sendiri dan belum terhubung ke node lain, maka **Parent** untuk setiap node adalah **node itu sendiri**.
- Nilai awal rank untuk masing-masing node juga sama, dalam studi kasus ini kita gunakan **0**.

- Proses **Find** akan mencari absolute parent dari sebuah node. Jika sebuah node memiliki parent yang berbeda, maka proses **Find** secara rekursif mencari parent dari parent node tadi, hingga ditemukan node yang merujuk ke diri sendiri sebagai parent
- Proses **Union** akan menggabungkan dua tree yang berbeda, dengan membuat hubungan dari dua node pada edge menjadi **parent and child**, salah satu node akan menjadi parent, dan node lainnya akan menjadi child dengan mengubah parentnya.

- Aturan dalam Algoritme Find-Union:
  - Diketahui node A dan B sebagai ujung dari suatu edge, maka:
    a. Bila **Parent** A dan B sama, maka pasti akan membentuk siklis
    b. Bila **Rank Parent** A lebih besar dari **Rank Parent** B, maka node A menjadi parent dan node B menjadi child
    c. Bila **Rank Parent** A lebih kecil dari **Rank Parent** B, maka node A menjadi child dan node B menjadi parent
    d. Bila **Rank Parent** A **sama dengan Rank Parent** B, maka salah satu node harus menjadi parent dan sisanya menjadi child. Misal Anda memilih A sebagai parent, maka tambahkan **Rank Parent** A sebesar 1 point, dan ubah parent node B menjadi A.

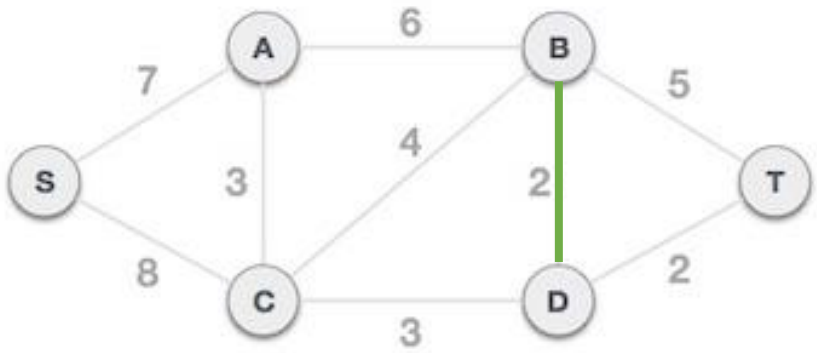
Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8



Node	S	A	B	C	D	T
Parent	S	A	B	C	D	T
Rank	0	0	0	0	0	0

- Edge B – D:
  - Node B memiliki parent B, dan node D memiliki parent D.
  - Karena parent berbeda dan rank sama, maka salah satu harus menjadi parent.
  - Misal D menjadi parent dan B menjadi child, maka:
    - Rank D bertambah satu poin
    - Parent B berubah menjadi D

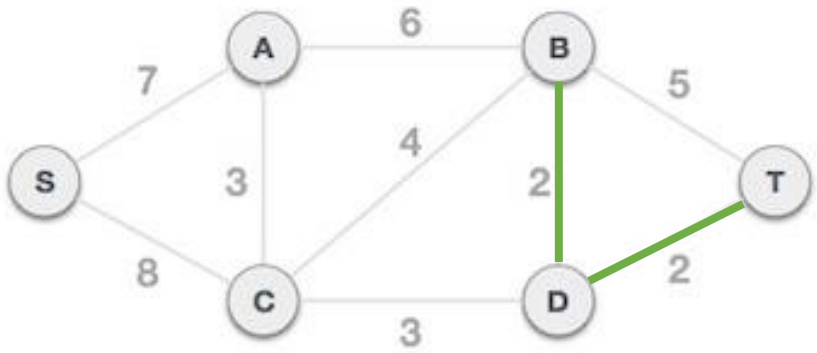
Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8



Node	S	A	B	C	D	T
Parent	S	A	D	C	D	T
Rank	0	0	0	0	1	0

- Edge D – T:
  - Node T memiliki parent T, dan node D memiliki parent D.
  - Karena parent berbeda dan rank D lebih tinggi dari T, maka D menjadi parent dan T menjadi child:
    - Parent T berubah menjadi D

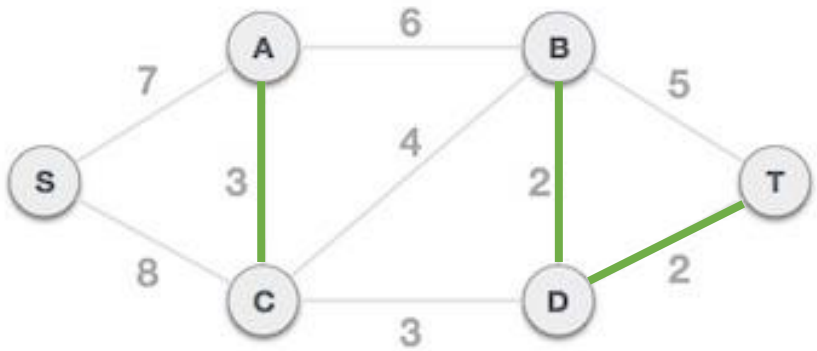
Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8



Node	S	A	B	C	D	T
Parent	S	A	D	C	D	D
Rank	0	0	0	0	1	0

- Edge A – C:
  - Node A memiliki parent A, dan node C memiliki parent C.
  - Karena parent berbeda dan rank sama, maka salah satu harus menjadi parent.
  - Misal C menjadi parent dan A menjadi child, maka:
    - Rank C bertambah satu poin
    - Parent A berubah menjadi C

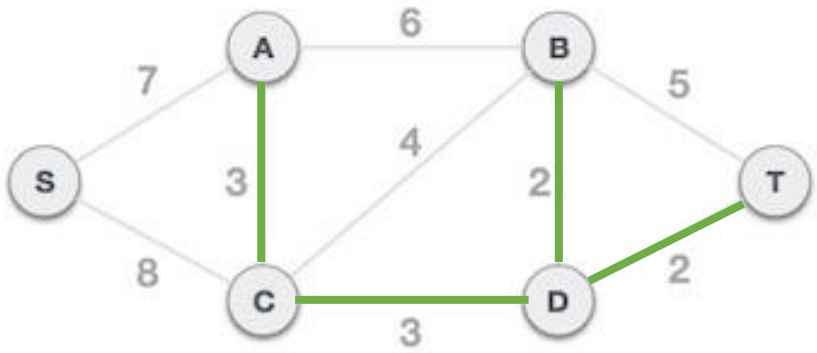
Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8



Node	S	A	B	C	D	T
Parent	S	C	D	C	D	D
Rank	0	0	0	1	1	0

- Edge C – D:
  - Node C memiliki parent C, dan node D memiliki parent D.
  - Karena parent berbeda dan rank sama, maka salah satu harus menjadi parent.
  - Misal D menjadi parent dan C menjadi child, maka:
    - Rank D bertambah satu poin
    - Parent C berubah menjadi D

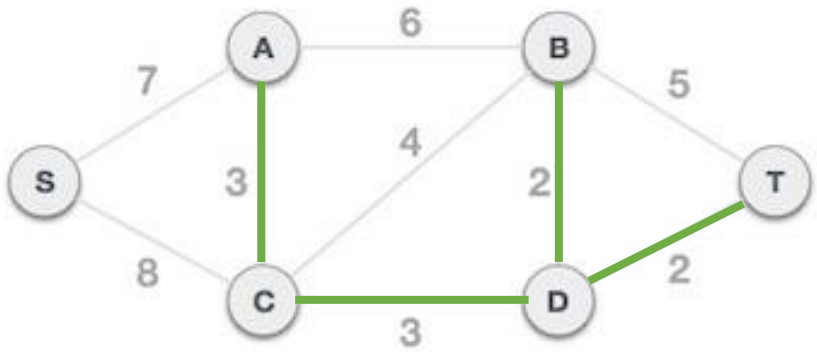
Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8



Node	S	A	B	C	D	T
Parent	S	C	D	D	D	D
Rank	0	0	0	1	2	0

- Edge C – B:
  - Node C memiliki parent D, dan node B memiliki parent D.
  - Karena parent sama, pasti akan membentuk siklis. Maka lewati.
- Edge B – T:
  - Node B memiliki parent D, dan node T memiliki parent D.
  - Karena parent sama, pasti akan membentuk siklis. Maka lewati.

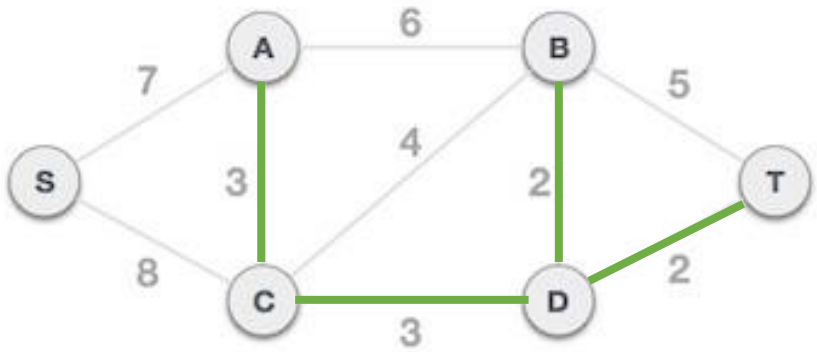
Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8



Node	S	A	B	C	D	T
Parent	S	C	D	D	D	D
Rank	0	0	0	1	2	0

- Edge A – B:
  - Node A memiliki parent C → Node C memiliki parent D → Node D memiliki parent D.
    - Maka Node A memiliki absolute parent (root) D
  - Node B memiliki parent D.
  - Karena parent sama, pasti akan membentuk siklis. Maka lewati.

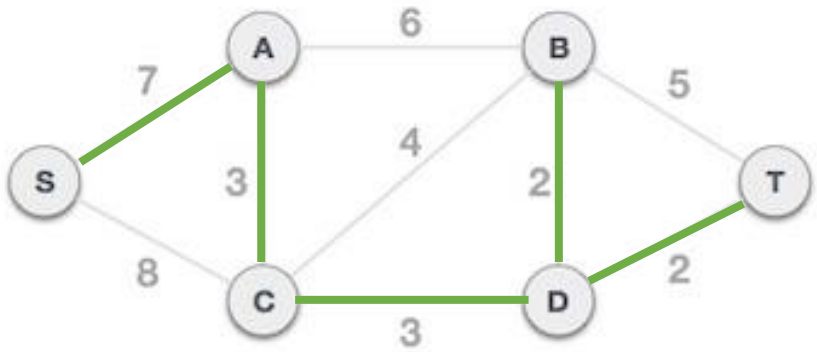
Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8



Node	S	A	B	C	D	T
Parent	S	C	D	D	D	D
Rank	0	0	0	1	2	0

- Edge S – A:
  - Node S memiliki parent S
  - Node A memiliki parent C → Node C memiliki parent D → Node D memiliki parent D.
  - Karena parent beda dan Rank D lebih tinggi dari S, maka S menjadi child dari D
    - Parent S berubah menjadi D

Daftar Edge		
Node Asal	**Node Tujuan**	**Biaya**
B	D	2
D	T	2
A	C	3
C	D	3
C	B	4
B	T	5
A	B	6
S	A	7
S	C	8



Node	S	A	B	C	D	T
Parent	D	C	D	D	D	D
Rank	0	0	0	1	2	0

- Karena jumlah edge sudah 5 dan memenuhi syarat jumlah edge MST (jumlah node – 1), maka algoritme dapat dihentikan

Contoh Kode: Kruskal's Algorithm

- https://github.com/hifra01/itdev-2021-algo/blob/main/Minimum%20Spanning%20Tree/Kruskal.py

Referensi

- https://en.wikipedia.org/wiki/Minimum_spanning_tree
- https://www.tutorialspoint.com/data_structures_algorithms/spanning_tree.htm
- https://www.tutorialspoint.com/data_structures_algorithms/prims_spanning_tree_algorithm.htm
- https://en.wikipedia.org/wiki/Prim's_algorithm
- https://www.tutorialspoint.com/data_structures_algorithms/kruskals_spanning_tree_algorithm.htm
- https://en.wikipedia.org/wiki/Kruskal's_algorithm
- https://www.youtube.com/watch?v=Ub-fJ-KoBQM