9.1 - Algoritma pada Graf (part 1)

[KOMS120403]

Desain dan Analisis Algoritma (2023/2024)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 10 (April 2024)

Daftar isi

- Definisi graf
- Minimum spanning tree (MST)
  - Greedy MST
  - Kruskal algorithm for MST
  - Prim algorithm for MST
  - Euclidean MST

Materi perkuliahan ini diambil dari slide Robert Sedgewick (Analysis of Algorithms)

Graf

Graf adalah struktur data matematis yang terdiri dari himpunan simpul/titik/nodes/vertices yang dihubungkan oleh sisi/edges.

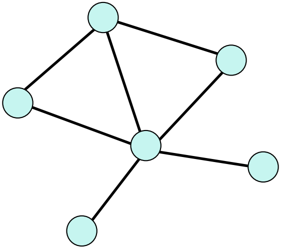


Figure: Sebuah graf tak berarah (*undirected*)

Graf

Simpul dalam graf dapat memiliki orientasi, dan graf tersebut disebut graf berarah.

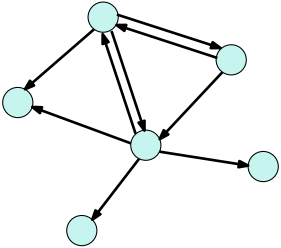


Figure: Sebuah graf berarah (*directed*)

Graf

Terminologi:

- Derajat simpul: banyaknya sisi insiden ke simpul tersebut
- Derajat masuk (*in-degree*) (untuk graf berarah): banyaknya sisi-sisi yang masuk ke simpul
- Derajat keluar (*out-degree*): banyaknya sisi yang keluar dari simpul
- Lintasan (*path*): barisan simpul/sisi dari satu simpul ke simpul lainnya
- Terhubung (*connected*): jika di antara dua simpul, ada lintasan
- sirkuit: lintasan yang dimulai dan diakhiri pada simpul yang sama
- Tree: graf terhubung yang tidak memuat sirkuit (asiklik)

Bagian 1. Minimum Spanning Tree

Graf pohon (*tree*)

Apa itu graf pohon (*tree*)?

Graf pohon adalah graf terhubung yang asiklik (tidak memuat sirkuit).

- Terhubung berarti ada lintasan antara dua simpul dalam graf.
- Asiklis berarti tidak mengandung sirkuit.

Sebuah pohon merentang (*spanning tree*) dari graf tak berarah G adalah subgraf T yang merupakan tree (terhubung, graf asiklis) dan span G (termasuk semua $V(G)$).

Graf pohon merentang (*minimum spanning tree*)

Minimum spanning tree (MST)

Permasalahan: diberikan graf tidak berarah G dengan bobot sisi positif. Temukan pohon merentang dengan berat minimum

Teorema (Cayley, 1889)

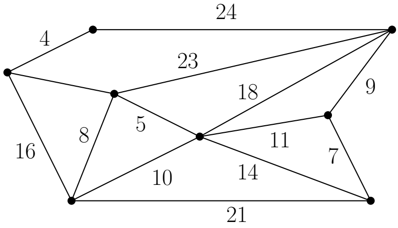
Terdapat n^{n-2} pohon merentang pada graf lengkap pada n simpul.

Minimum spanning tree (MST)

Permasalahan: diberikan graf tidak berarah G dengan bobot sisi positif. Temukan pohon merentang dengan berat minimum

Teorema (Cayley, 1889)

Terdapat n^{n-2} pohon merentang pada graf lengkap pada n simpul.



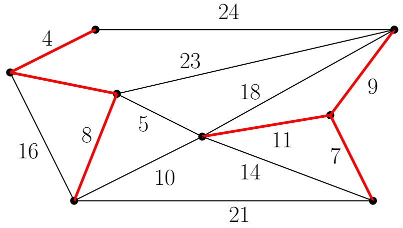
graph G with weighted edges

Minimum spanning tree (MST)

Permasalahan: diberikan graf tidak berarah G dengan bobot sisi positif. Temukan pohon merentang dengan berat minimum

Teorema (Cayley, 1889)

Terdapat n^{n-2} pohon merentang pada graf lengkap pada n simpul.



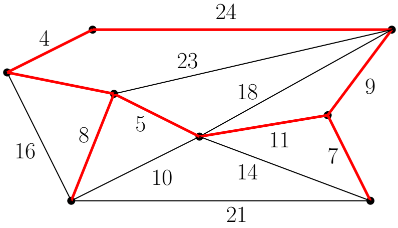
not connected

Minimum spanning tree (MST)

Permasalahan: diberikan graf tidak berarah G dengan bobot sisi positif. Temukan pohon merentang dengan berat minimum

Teorema (Cayley, 1889)

Terdapat n^{n-2} pohon merentang pada graf lengkap pada n simpul.



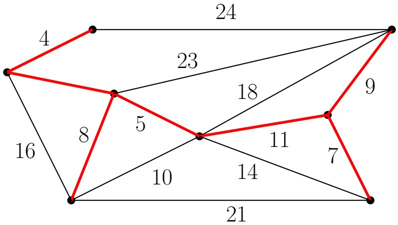
not acyclic

Minimum spanning tree (MST)

Permasalahan: diberikan graf tidak berarah G dengan bobot sisi positif. Temukan pohon merentang dengan berat minimum

Teorema (Cayley, 1889)

Terdapat n^{n-2} pohon merentang pada graf lengkap pada n simpul.



spanning tree T with cost $50 = 4 + 6 + 8 + 5 + 11 + 9 + 7$

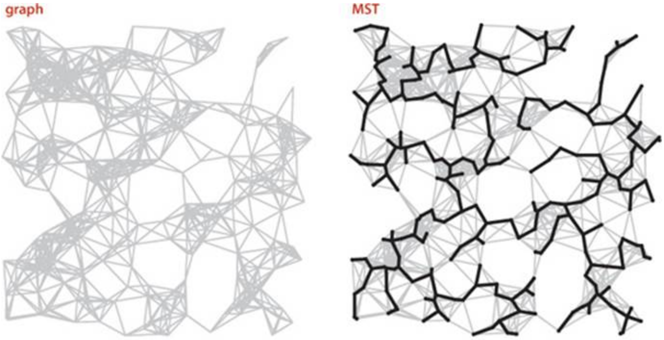
Minimum spanning tree (MST)

Asal mula MST

- Masalah menemukan konstruksi jaringan tenaga listrik yang paling ekonomis.
- Ilmuwan Ceko Otakar Borüvka mengembangkan algoritma pertama yang diketahui untuk menemukan pohon rentang minimum, pada tahun 1926.

MST dalam dunia nyata (1)

Graf dengan banyak simpul (large graph) dan MST-nya

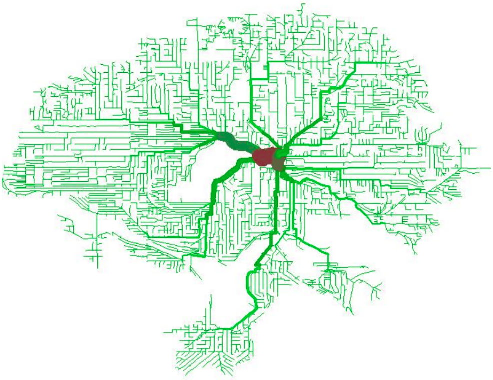


A 250-vertex Euclidean graph (with 1,273 edges) and its MST

source: `https://apprize.best/science/algorithms_2/algorithms_2.files/image091.jpg`

MST dalam dunia nyata (2)

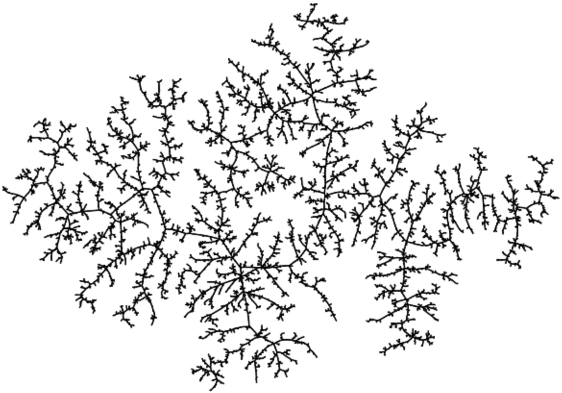
MST dari rute lintasan sepeda di North Seattle



source: http://www.flickr.com/photos/ewedistrict/21980840

MST dalam dunia nyata (3)

MST dari graf random/acak



source: `http://algo.inria.fr/broutin/gallery.html`

Aplikasi MST

- Desain jaringan (*network*)
  - *telepon, listrik, hidrolik, kabel TV, komputer, atau jaringan jalan di satelit*
- Cluster analysis
- Real-time face verification
- Algoritma aproksimasi untuk permasalahan NP-Hard
  - *contoh: Traveling Salesman Problem*

Penyelesaian MST dengan *brute force*

Bagaimana kita memecahkan MST dengan pendekatan brute-force?

Algoritma brute-force:

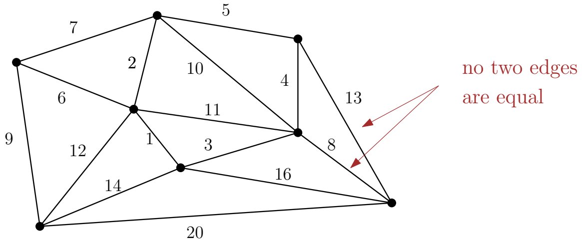
1. Buat daftar semua pohon rentang yang mungkin
2. Hitunglah berat setiap pohon rentang
3. Ambil pohon rentang yang memiliki bobot minimum

Bagian 2. Algoritma greedy MST

Asumsi untuk Minimum Spanning Tree pada graf

- Graf terhubung
- Bobot sisi-sisi berbeda
- Bobot sisinya tidak harus berupa jarak antara dua simpul ujungnya

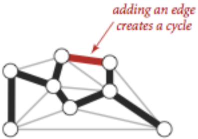
Konsekuensi: dengan asumsi tersebut, maka MST ada dan tunggal. Namun, tanpa asumsi, sebuah graf mungkin saja memiliki lebih dari satu MST (atau tidak ada sama sekali)



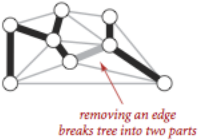
Prinsip dasar pencarian MST

Sifat-sifat berikut berlaku untuk setiap *spanning tree* pada graf

- Menambahkan sisi yang menghubungkan dua simpul dalam sebuah pohon menciptakan tepat sebuah sirkuit.



- Menghapus sisi dari graf pohon memecahnya menjadi dua subpohon yang terpisah.

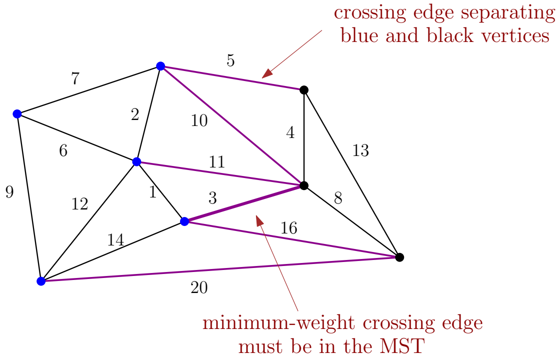


Cut property

Cut property (1)

Sebuah cut di G adalah partisi dari $V(G)$ menjadi dua himpunan (yang tidak kosong) A dan B.

Crossing edge menghubungkan sebuah simpul di A dan sebuah simpul di B.



Cut property (2)

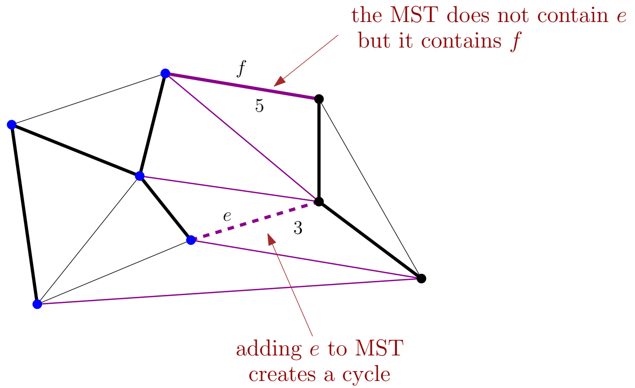
Lemma (Cut property)

Diberikan sebarang cut, crossing edge dengan bobot minimum termuat pada MST.

Proof. Misalkan e adalah *crossing edge* dengan bobot minimum, dan e tidak termuat di MST T.

- Perhatikan bahwa menambahkan e ke T menghasilkan sebuah sirkuit.
- Misal f adalah sisi pada sirkuit tersebut, maka f termuat di T.
- $T' = T - \{e\} + \{f\}$ juga adalah spanning tree, dan $\text{cost}(T') < \text{cost}(T)$.
- Kontradiksi, karena kita mendapatkan spanning tree T' yang memiliki bobot kurang dari T.

Cut property (3)



Algoritma greedy untuk MST

Algoritma greedy untuk MST (1)

Input: G: graf tak berarah, w: fungsi pembobotan sisi
Output: himpunan sisi yang membentuk MST

- Mulailah dengan semua sisi berwarna abu-abu (warna abu-abu menunjukkan bahwa sisi tersebut tidak berada dalam solusi);
- Temukan *cut* tanpa crossing edge bewana hitam; warnai sisi berbobot minimumnya dengan warna hitam (warna hitam menunjukkan bahwa sisi tersebut termasuk dalam solusi);
- Ulangi hingga $|V| - 1$ sisinya berwarna hitam;
- Semua sisi hitam adalah MST.

Algoritma greedy untuk MST (2)

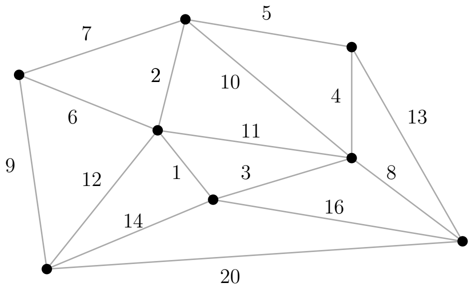


Figure: Input graf G

Algoritma greedy untuk MST (3)

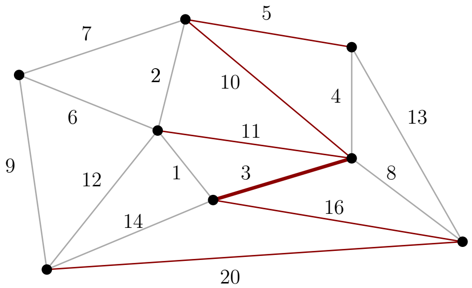


Figure: Sisi merah membentuk potongan G

Algoritma greedy untuk MST (3)

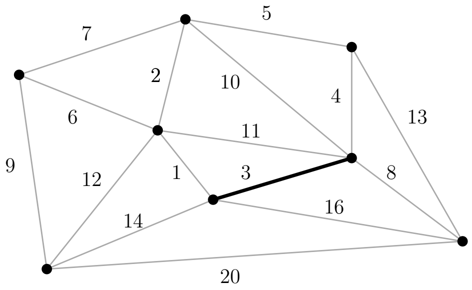


Figure: Ambil sisi berbobot minimum pada potongan dan warnai dengan hitam

Algoritma greedy untuk MST (3)

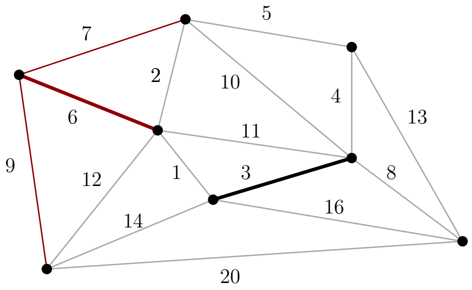


Figure: Pilih *cut* yang lain

Algoritma greedy untuk MST (3)

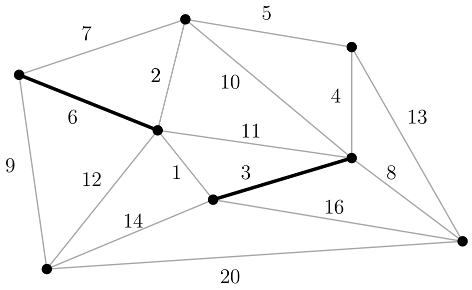


Figure: Ambil sisi berbobot minimum pada potongan dan warnai dengan hitam

Algoritma greedy untuk MST (3)

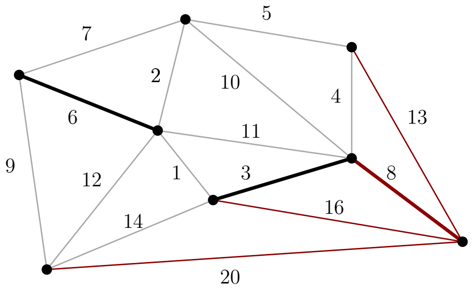


Figure: Pilih *cut* yang lain

Algoritma greedy untuk MST (3)

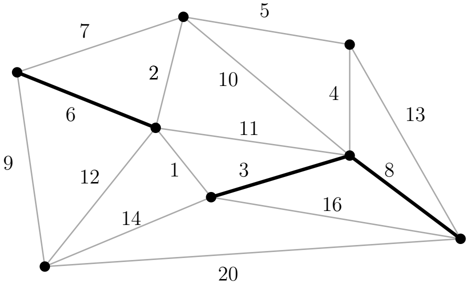


Figure: Ambil sisi berbobot minimum pada potongan dan warnai dengan hitam

Algoritma greedy untuk MST (3)

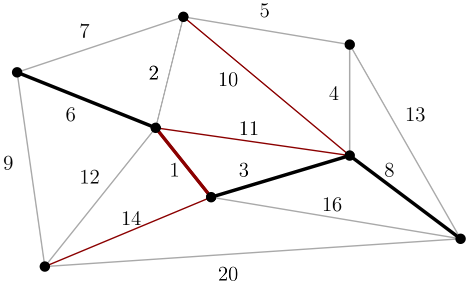


Figure: Pilih *cut* yang lain

Algoritma greedy untuk MST (3)

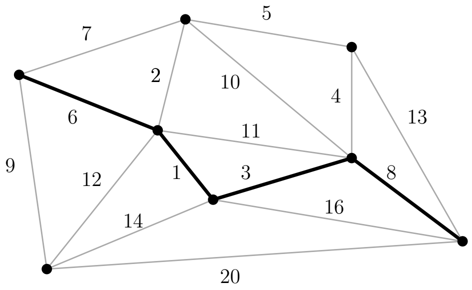


Figure: Ambil sisi berbobot minimum pada potongan dan warnai dengan hitam

Algoritma greedy untuk MST (3)

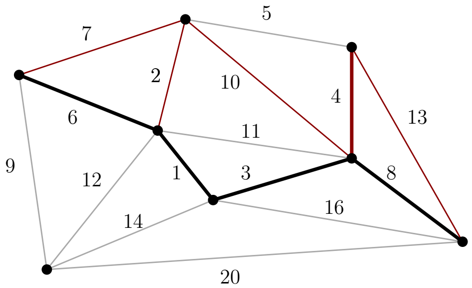


Figure: Pilih *cut* yang lain

Algoritma greedy untuk MST (3)

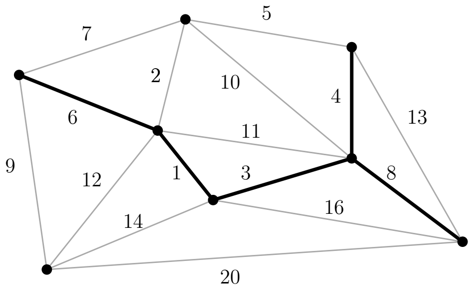


Figure: Ambil sisi berbobot minimum pada potongan dan warnai dengan hitam

Algoritma greedy untuk MST (3)

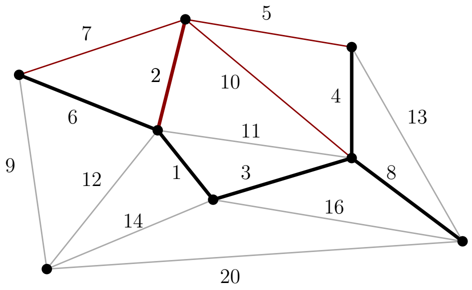


Figure: Pilih *cut* yang lain

Algoritma greedy untuk MST (3)

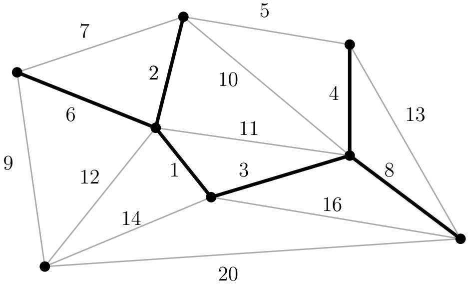


Figure: Ambil sisi berbobot minimum pada potongan dan warnai dengan hitam

Algoritma greedy untuk MST (3)

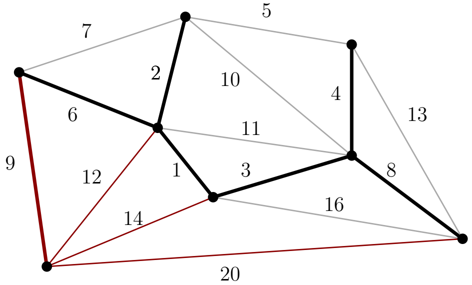


Figure: Pilih *cut* yang lain

Algoritma greedy untuk MST (3)

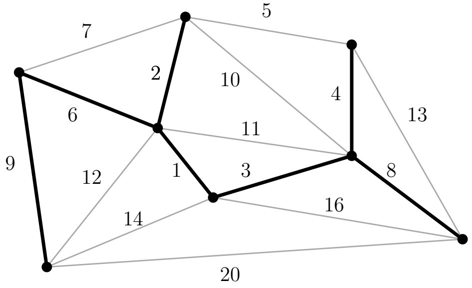


Figure: Ambil sisi berbobot minimum pada potongan dan warnai dengan hitam

Kebenaran algoritma greedy untuk MST

Teorema

Algoritma greedy menghasilkan MST.

Proof.

Kebenaran algoritma greedy untuk MST

Teorema

Algoritma greedy menghasilkan MST.

Proof.

- Berdasarkan *cut property*, setiap sisi yang berwarna hitam ada di MST.

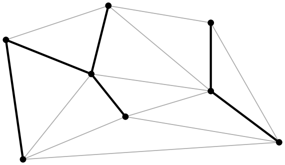
Kebenaran algoritma greedy untuk MST

Teorema

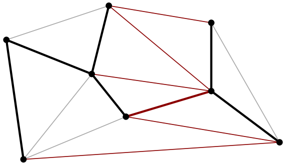
Algoritma greedy menghasilkan MST.

Proof.

- Berdasarkan *cut property*, setiap sisi yang berwarna hitam ada di MST.
- Jika tedapat kurang dari $(|V| - 1)$ sisi hitam, maka terdapat *cut* tanpa crossing edge bewarna hitam (ambil cut yang simpulnya merupakan satu komponen yang terhubung)



fewer than $|V| - 1$ edges colored black



a cut with no black crossing edges

Bagian 3. Algoritma Kruskal dan algoritma Prim

Implementasi yang lebih efisien dari "Greedy MST"

Algoritma greedy yang didasarkan pada "Cut property" sulit diimplementasikan, dan efisiensi algoritma masih dapat ditingkatkan. Kita mengajukan pertanyaan-pertanyaan berikut:

- Bagaimana cara memilih cut secara efisien?
- Bagaimana cara menemukan sisi dengan bobot minimum secara efisien?

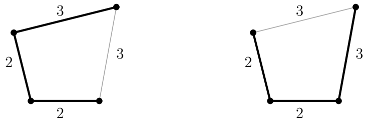
Beberapa algoritma pencarian MST lainnya:

1. Algoritma MST Kruskal
2. Algoritma Prim
3. Borüvka's algorithm (tidak dibahas pada perkuliahan ini)

Jika asumsi penyederhanaan tidak digunakan...

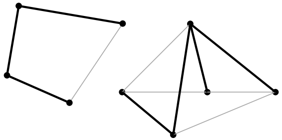
1. Bagaimana jika bobot sisi tidak berbeda?

MST tidak tunggal. Tetapi algoritma Greedy MST masih berfungsi (*coba Anda buktikan!*)



2. Bagaimana jika graf tidak terhubung?

Carilah Minimum Spanning Forest (yakni MST untuk setiap komponen yang terhubung pada graf)



Algoritma MST Kruskal

Algoritma MST Kruskal (1)

Bertujuan untuk menemukan minimum spanning tree dari graf berbobot yang tidak berarah. Dalam hal graf tidak terhubung, maka ini adalah pencarian *minimum spanning forest*).

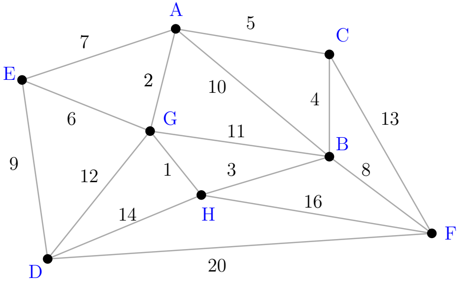
Algoritma ini menggunakan teknik Greedy, dan pertama kali muncul di Proceedings of the AMS, 1956, dan ditulis oleh Joseph Kruskal.

Input: graf berbobot G
Output: himpunan simpul yang membentuk MST dari G

Algorithm.

- Perhatikan sisi dalam urutan bobot yang menaik (*ascending*);
- Tambahkan sisi berikutnya ke pohon T kecuali hal itu akan membentuk sebuah sirkuit.

Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14
FH	16
DF	20

Figure: Sisi-sisi diurutkan berdasarkan bobotnya

Algoritma MST Kruskal (2)

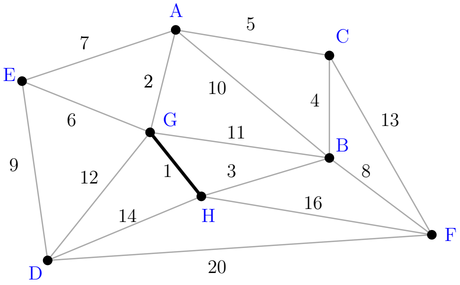
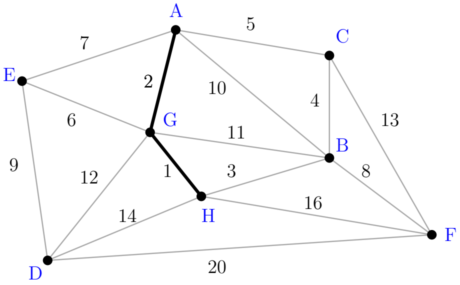


Figure: Sisi *GH* termasuk dalam MST

Edge	**Weight**
GH	1

Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2

Figure: Sisi *AG* termasuk dalam MST

Algoritma MST Kruskal (2)

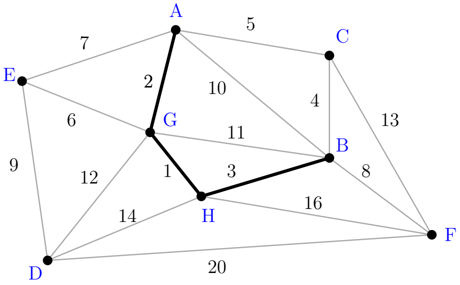
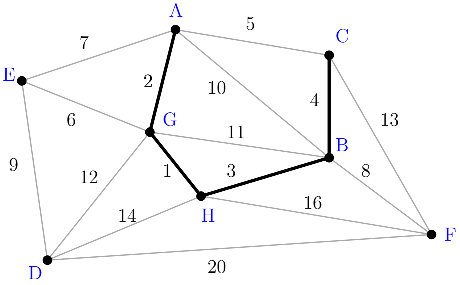


Figure: Sisi *BH* termasuk dalam MST

Edge	**Weight**
GH	1
AG	2
BH	3

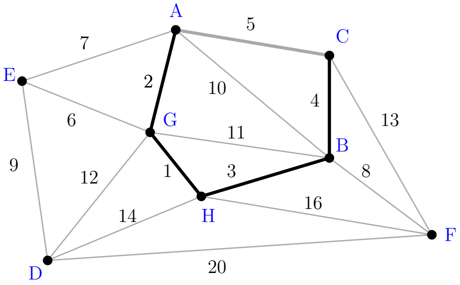
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4

Figure: Sisi *BC* termasuk dalam MSTT

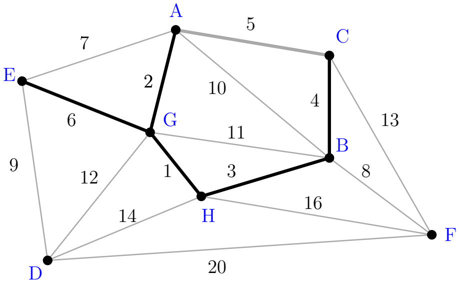
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5

Figure: Sisi *AC* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

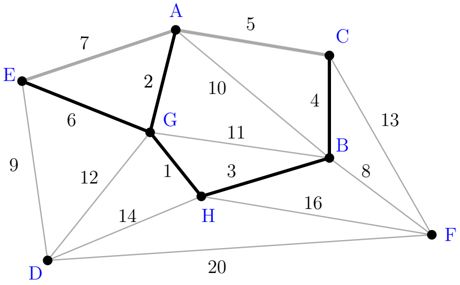
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6

Figure: Sisi *EG* termasuk dalam MST

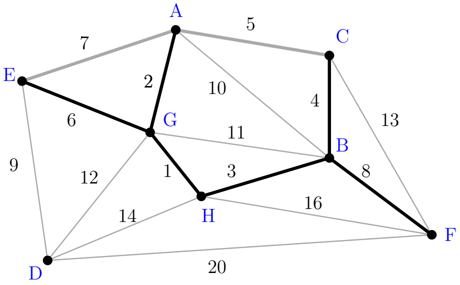
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7

Figure: Sisi *AE* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

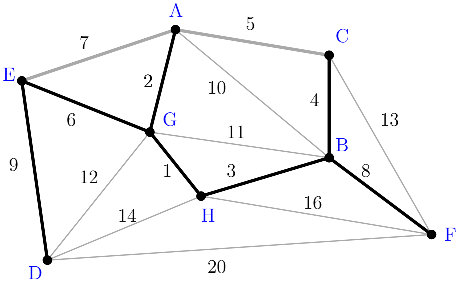
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8

Figure: Sisi *BF* termasuk dalam MST

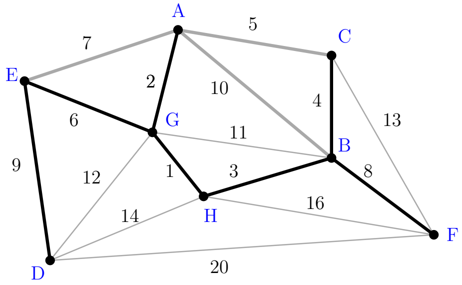
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9

Figure: Sisi *DE* termasuk dalam MST

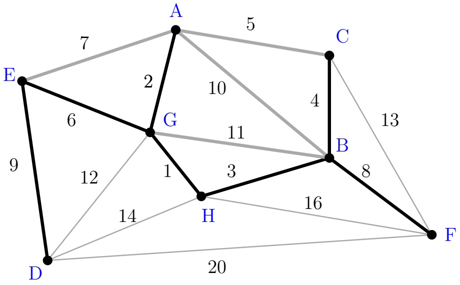
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10

Figure: Sisi *AB* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

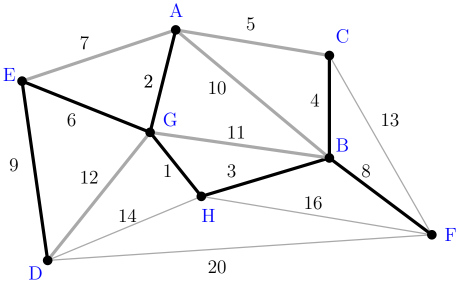
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11

Figure: Sisi *BG* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

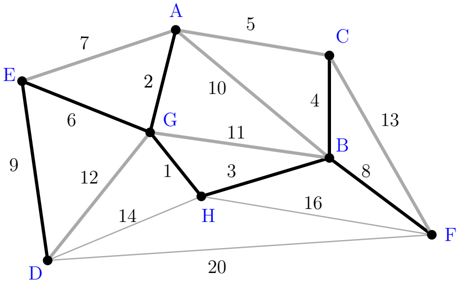
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12

Figure: Sisi *DG* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

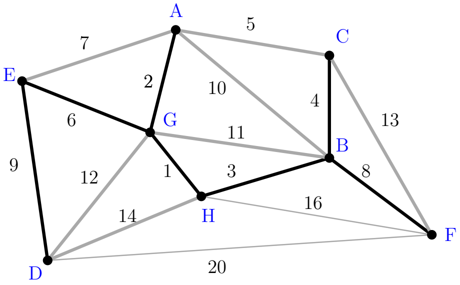
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13

Figure: Sisi *CF* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

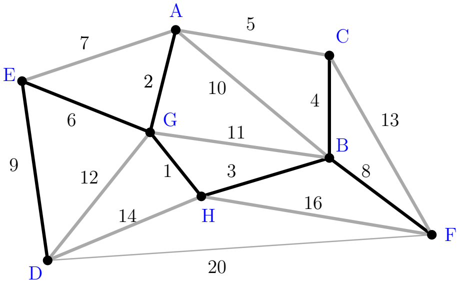
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14

Figure: Sisi *DH* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

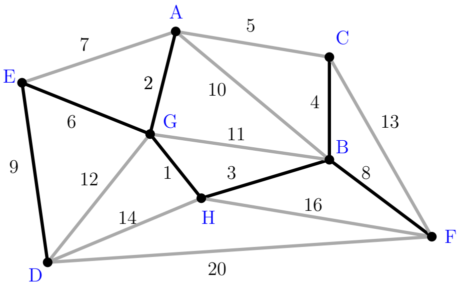
Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14
FH	16

Figure: Sisi *FH* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

Algoritma MST Kruskal (2)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14
FH	16
DF	20

Figure: Sisi *DF* **tidak** termasuk dalam MST, karena akan membektuk sirkuit

Bukti kebenaran algoritma MST Kruskal

Teorema (Kruskal, 1956)

Algoritma Kruskal menghasilkan MST.

Proof. Misalkan T adalah graf yang dihasilkan oleh algoritma Kruskal.

- T adalah sebuah spanning tree. Berdasarkan algoritma, kita tidak memilih suatu sisi jika itu membuat sirkuit dengan sisi yang sudah dipilih sebelumnya. Jadi, T asiklik.

  Selanjutnya perhatikan bahwa T terhubung, sebab sisi dengan bobot terkecil yang melintasi dua komponen T akan dipilih.

- T memiliki bobot minimal di antara *spanning tree* lainnya di graf tersebut.
  Ini merupakan akibat dari **cut property**: diberikan sebarang cut, *crossing edge* dengan bobot minimum termuat di MST.

Bukti kebenaran algoritma MST Kruskal

Pertanyaan: Bagaimana cara memeriksa bahwa menambahkan sisi ke T akan membuat sirkuit?

Alternatif 1: Terapkan algoritma DFS.

- Dibutuhkan waktu sebesar $\mathcal{O}(n)$ untuk memeriksa sebuah sirkuit, dimana $n = |V|$.
- Secara keseluruhan, dibutuhkan waktu $\mathcal{O}(mn)$, dimana $n = |V|$ adn $m = |E|$.

Alternatif 2: Gunakan struktur data union-find*.

- Pertahankan satu himpunan untuk setiap komponen yang terhubung.
- Jika v dan w berada dalam komponen yang sama, maka menambahkan sisi vw akan menghasilkan sebuah sirkuit.
- Untuk menambahkan vw ke T, gabungkan himpunan yang berisi v dan w.

*https://www.wikiwand.com/en/Disjoint-set_data_structure

Algoritma Prim

Algoritma Prim (1)

Algoritma Prim juga berbasis pada algoritma Greedy.

Algoritma

- Dimulai dengan pohon rentang yang kosong;
- Idenya adalah untuk mempertahankan dua himpunan simpul;
- Himpunan pertama berisi simpul yang sudah termasuk dalam MST, himpunan lainnya berisi simpul yang belum disertakan;
- Pada setiap langkah, pertimbangkan semua sisi yang menghubungkan dua himpunan, dan ambil sisi dengan bobot minimum di antara sisi-sisi ini;
- Setelah memilih sisi, pindahkan simpul akhir lainnya dari sisi ke himpunan yang berisi MST.

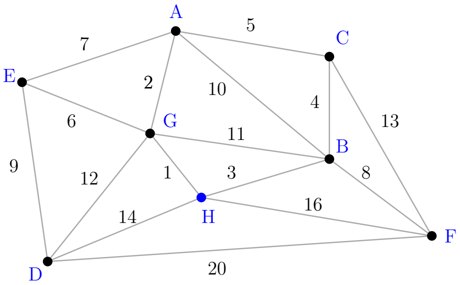
Algoritma Prim (2)

Input: graf berbobot G
Output: himpunan simpul yang membentuk MST dari G

Algoritma:

- Inisialisasi himpunan solusi $T = \emptyset$;
- Mulailah dengan sebarang simpul dan dengan cara greedy kita akan "memperbesar" T;
- Tambahkan ke T sisi bobot minimum dengan tepat pada simpul akhir di T;
- Ulangi sampai T memiliki ukuran $(|V| - 1)$.

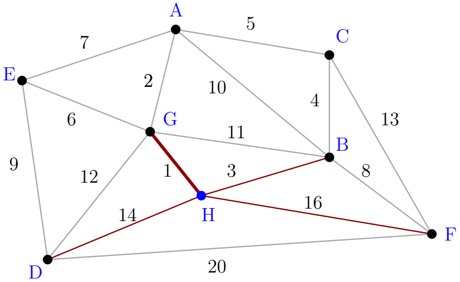
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9
AB	10
BG	11
DG	12
CF	13
DH	14
FH	16
DF	20

Mulailah dengan graf berbobot. Pilih simpul acak, misalnya H.

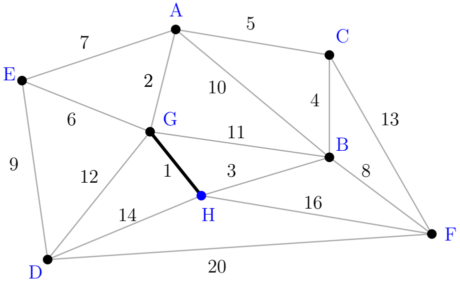
Algoritma Prim (3)



Edge	Weight
GH	1

Perhatikan sisi yang insiden dengan ke H. GH adalah sisi dari bobot minimum.

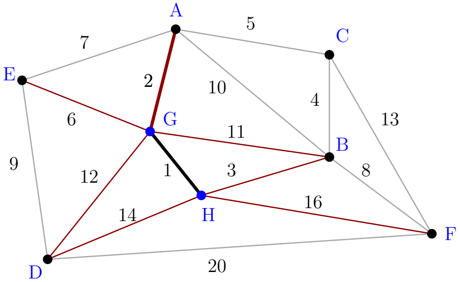
Algoritma Prim (3)



Edge	Weight
GH	1

Sertakan *GH* ke kumpulan solusi.

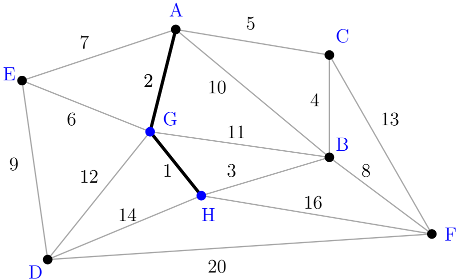
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2

Perhatikan simpul $\{H, G\}$. Perhatikan sisi yang insiden dengan ke G atau H. AG adalah sisi dengan bobot minimum.

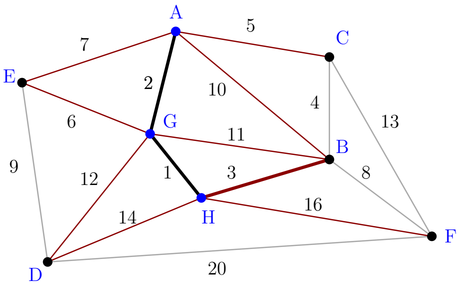
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2

Sertakan *AG* ke kumpulan solusi.

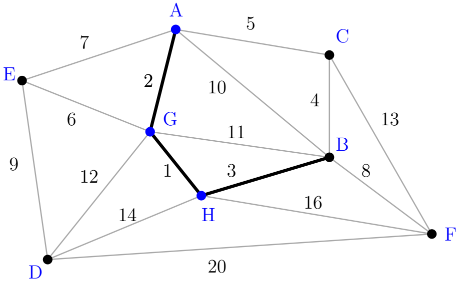
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3

BH adalah sisi berbobot minimum yang insiden dengan $\{H, G, A\}$.

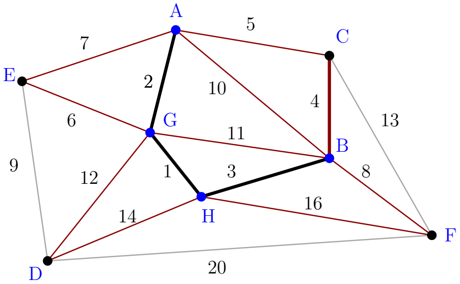
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3

Sertakan *BH* ke kumpulan solusi.

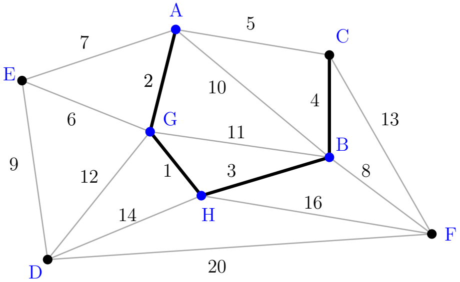
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4

BC adalah sisi berbobot minimum yang insiden dengan $\{H, G, A, B\}$.

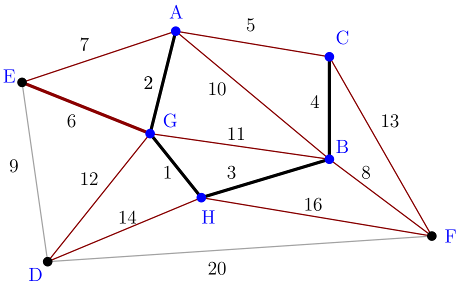
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4

Sertakan *BC* ke kumpulan solusi.

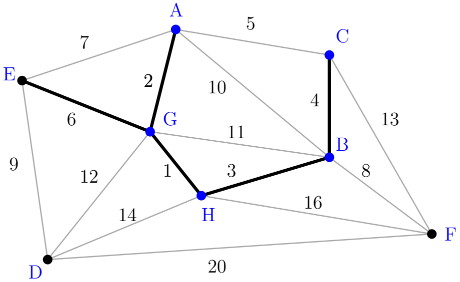
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6

EG adalah sisi berbobot minimum yang insiden dengan $\{H, G, A, B, C\}$
(AC tidak dapat dipilih karena memiliki dua tetangga dalam solusi saat ini)

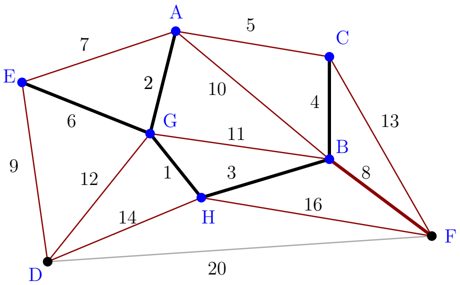
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6

Sertakan *GE* ke kumpulan solusi.

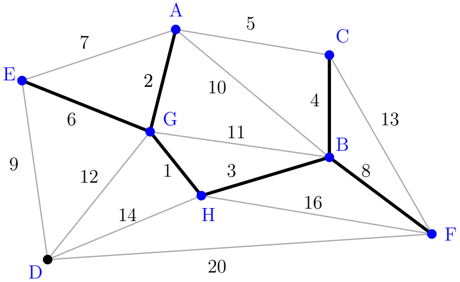
Algoritma Prim (3)



Edge	**Weight**
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8

BF adalah sisi berbobot minimum yang insiden dengan $\{H, G, A, B, C, E\}$ (*AE* tidak dapat dipilih karena memiliki dua tetangga dalam solusi saat ini)

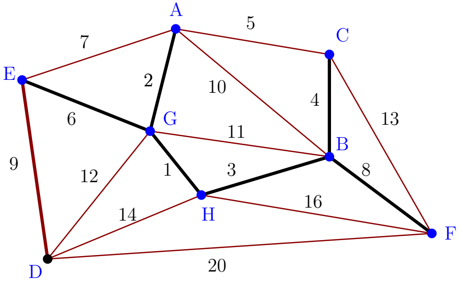
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8

Sertakan *BF* ke kumpulan solusi

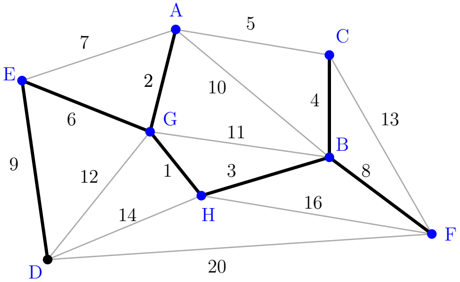
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9

DE adalah sisi berbobot minimum yang insiden dengan $\{H, G, A, B, C, E, F\}$

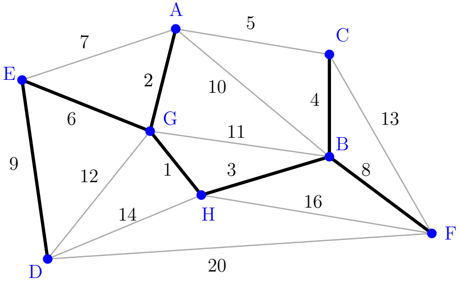
Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9

Sertakan *DE* ke kumpulan solusi

Algoritma Prim (3)



Edge	Weight
GH	1
AG	2
BH	3
BC	4
AC	5
EG	6
AE	7
BF	8
DE	9

$\{GH, AG, BH, BC, EG, BF, DE\}$ adalah solusi dari MST

Algoritma MST Prim: pembuktian kebenaran

Teorema (Jarnik 1930, Djikstra 1957, Prim 1959)

Algoritma Prim menghasilkan MST.

Proof. Misalkan T adalah output graf oleh Algoritma Prim

- T adalah spanning tree. Pada setiap langkah, kita menambahkan sebuah simpul di $G \setminus T$ yang memiliki satu tetangga di T. T terbentang karena T berisi sisi $|V| - 1$, yaitu $|V(T)| = |V(G)|$.
- T adalsh MST. Berdasarkan **cut property**: diberikan sebarang cut, sisi penyeberangan dengan bobot minimum ada di MST.

Kruskal's MST algorithm: pembuktian kebenaran

Pertanyaan. Bagaimana cara memeriksa apakah menambahkan sisi ke T akan membuat sirkuit?

Alternatif. Naive solution: dengan DFS.

- Untuk setiap sirkuit, dibutuhkan waktu $\mathcal{O}(|V|)$.
- Secara keseluruha, dibutuhkan waktu $\mathcal{O}(|E||V|)$.

Pertanyaan. Bagaimana menemukan sisi dengan bobot minimum dengan tepat satu simpul akhir di S?

Alternatif. Brute force: coba semua sisi.

- Dibutuhkan waktu $\mathcal{O}(E)$ untuk setiap sisi pohon rentang.
- Dibutuhkan waktu $\mathcal{O}(|E||V|)$ secara keseluruhan.

Bagian 4. Algoritma MST pada ruang Euclid

Algoritma untuk MST dalam ruang Euclidean

Euclidean MST

Permasalahan: diberikan n titik pada sebuah ruang Euclid, temukan MST yang menghubungkan mereka, di ana *jarak antara pasangan simpul adalah jarak Euclidean mereka*.

Jarak Euclid. Diberikan dua simpul $a = (x_1, y_1)$ dan $b = (x_2, y_2)$,

$$d_{(a,b)} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Algoritma naif: Hitung jarak $\Theta(n^2)$ dan jalankan Algoritma Prim.

Perbaikan. Eksploitasi struktur geometris yag dapat dilakukan dalam waktu $\mathcal{O}(n \log n)$.

Algoritma MST untuk graf di ruang Euclid

Algoritma Euclidean MST

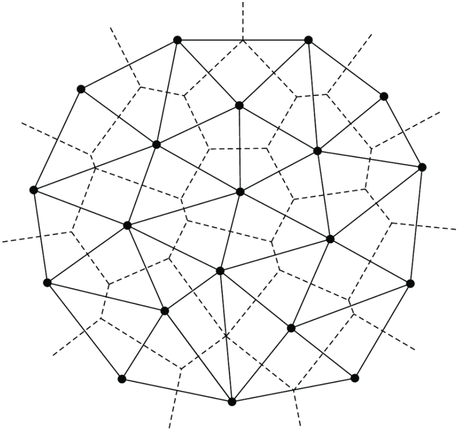
- Hitung diagram Voronoi untuk mendapatkan triangulasi Delaunay.
- Jalankan algoritma MST Kruskal di sisi Delaunay.

Kompleksitas. $\mathcal{O}(n \log n)$

- Triangulasi Delaunay berisi $\leq 3n$ sisi karena planar.
- $\mathcal{O}(n \log n)$ for Voronoi.
- $\mathcal{O}(n \log n)$ for Kruskal

Batas bawah. Setiap algoritma Euclidean MST berbasis perbandingan membutuhkan $\Omega(n \log n)$ perbandingan.

Algoritma MST untuk graf di ruang Euclid



Penerapan Euclidean MST: *k*-clustering

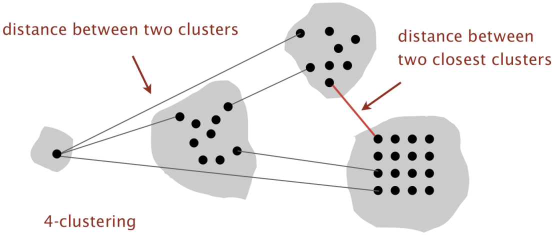
k-Clustering: Bagilah satu set objek yang diklasifikasikan ke dalam k grup yang koheren.

Fungsi jarak: Nilai numerik yang menentukan "kedekatan" dua objek.

Tujuan: Membagi menjadi cluster sehingga objek dalam cluster yang berbeda berjauhan.

Single-link: Jarak antara dua cluster sama dengan jarak antara dua objek terdekat (satu di setiap cluster).

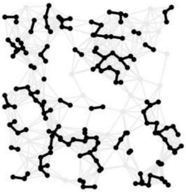
Single-link clustering: Diberi bilangan bulat k, temukan k-clustering yang memaksimalkan jarak antara dua kelompok terdekat.



Single-link clustering algorithm

Algoritma "terkenal" dalam literatur sains untuk *single-link k-clustering*:

- Bentuk $|V|$ cluster masing-masing satu objek.
- Temukan pasangan objek terdekat sedemikian rupa sehingga setiap objek berada di cluster yang berbeda, dan gabungkan kedua cluster.
- Ulangi sampai ada tepat k cluster.



Observasi: Ini adalah algoritma Kruskal (yang berhenti ketika ada k komponen yang terhubung).

Solusi alternatif: Jalankan Algoritma Prim dan hapus sisi berbobot maksimum $k-1$.

to be continued...