

Algoritma *Greedy*

Oleh :
Aris Sugiharto

**Departemen Ilmu Komputer
FSM – Universitas Diponegoro**

OUTLINE

- A. Permasalahan Optimisasi
- B. Konsep Algoritma Greedy
- C. Contoh-contoh Algoritma Greedy
- D. Kesimpulan

A. Permasalahan Optimasi

Permasalahan optimasi (*optimization problems*): Persoalan yang menuntut pencarian solusi optimum.

Permasalahan optimasi ada dua macam:

1. Maksimasi (*maximization*)
2. Minimasi (*minimization*)

- Solusi optimum (terbaik) adalah solusi yang bernilai minimum atau maksimum dari sekumpulan alternatif solusi yang mungkin.
- Elemen persoalan optimasi:
 1. kendala (*constraints*)
 2. fungsi objektif(atau fungsi optimasi)
- Solusi yang memenuhi semua kendala disebut **solusi layak (*feasible solution*)**.
- Solusi layak yang mengoptimalkan fungsi optimasi disebut **solusi optimum**.

Contoh Persoalan Optimasi

Persoalan Penukaran Uang :

Diberikan uang senilai A . Tukar A dengan koin-koin uang yang ada. Berapa jumlah minimum koin yang diperlukan untuk penukaran tersebut?

Ini merupakan persoalan ***minimasi***

Contoh 1: tersedia banyak koin 1, 5, 10, 25

- Uang senilai $A = 32$ dapat ditukar dengan banyak cara berikut:

$$32 = 1 + 1 + \dots + 1 \quad (32 \text{ koin})$$

$$32 = 5 + 5 + 5 + 5 + 10 + 1 + 1 \quad (7 \text{ koin})$$

$$32 = 10 + 10 + 10 + 1 + 1 \quad (5 \text{ koin})$$

... dst

- Minimum: $32 = 25 + 5 + 1 + 1 \quad (4 \text{ koin})$

B. Konsep Algoritma Greedy

- ▣ *Greedy* = rakus, tamak, loba, ...
- ▣ Prinsip *greedy*: “*take what you can get now!*”.
- ▣ Algoritma *greedy* membentuk solusi langkah per langkah (*step by step*).
- ▣ Pada setiap langkah, terdapat banyak pilihan yang perlu dieksplorasi.
- ▣ Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan.

- Pada setiap langkah, kita membuat pilihan **optimum lokal** (*local optimum*)
- Dengan harapan bahwa langkah sisanya mengarah ke solusi **optimum global** (*global optimum*).

- Tinjau masalah penukaran uang:

Strategi *greedy*:

Pada setiap langkah, pilihlah koin dengan nilai terbesar dari himpunan koin yang tersisa.

- Misal: $A = 32$, koin yang tersedia: 1, 5, 10, dan 25
Langkah 1: pilih 1 buah koin 25 (Total = 25)
Langkah 2: pilih 1 buah koin 5 (Total = $25 + 5 = 30$)
Langkah 3: pilih 2 buah koin 1 (Total = $25 + 5 + 1 + 1 = 32$)
- Solusi: Jumlah koin minimum = 4 (solusi optimal!)

Skema Umum Algoritma *Greedy*

Algoritma greedy disusun oleh elemen-elemen berikut:

- 1. Himpunan kandidat.**

Berisi elemen-elemen pembentuk solusi.

- 2. Himpunan solusi**

Berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.

- 3. Fungsi seleksi (*selection function*)**

Memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.

Skema Umum Algoritma *Greedy*

4. Fungsi kelayakan (*feasible*)

Memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (*constraints*) yang ada.

Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.

5. Fungsi obyektif, yaitu fungsi yang memaksimumkan atau meminimumkan nilai solusi (misalnya panjang lintasan, keuntungan, dan lain-lain).

Dengan demikian :

algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S , dari himpunan kandidat, C ;

yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.

Pada masalah penukaran uang:

1. **Himpunan kandidat:** himpunan koin yang merepresentasikan nilai 1, 5, 10, 25, paling sedikit mengandung satu koin untuk setiap nilai.
2. **Himpunan solusi:** total nilai koin yang dipilih tepat sama jumlahnya dengan nilai uang yang ditukarkan.
3. **Fungsi seleksi:** pilihlah koin yang bernilai tertinggi dari himpunan kandidat yang tersisa.
4. **Fungsi layak:** memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar.
5. **Fungsi obyektif:** jumlah koin yang digunakan minimum.

Skema umum algoritma *greedy*:

```
function greedy(input C: himpunan_kandidat) → himpunan_kandidat
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy
  Masukan: himpunan kandidat C
  Keluaran: himpunan solusi yang bertipe himpunan_kandidat
}
Deklarasi
  x : kandidat
  S : himpunan_kandidat

Algoritma:
  S ← {}    { inisialisasi S dengan kosong }
  while (not SOLUSI(S)) and (C ≠ {} ) do
    x ← SELEKSI(C)      { pilih sebuah kandidat dari C }
    C ← C - {x}         { elemen himpunan kandidat berkurang satu }
    if LAYAK(S ∪ {x}) then
      S ← S ∪ {x}
    endif
  endwhile
  { SOLUSI(S) or C = {} }

  if SOLUSI(S) then
    return S
  else
    write('tidak ada solusi')
  endif
```

- Pada akhir setiap iterasi, solusi yang terbentuk adalah optimum lokal.
- Pada akhir kalang while-do diperoleh optimum global.

- ▣ *Warning*: Optimum global belum tentu merupakan solusi optimum (terbaik), tetapi *sub-optimum* atau *pseudo-optimum*.
- ▣ Alasan:
 1. Algoritma *greedy* tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada (sebagaimana pada metode *exhaustive search*).
 2. Terdapat beberapa fungsi SELEKSI yang berbeda, sehingga kita harus memilih fungsi yang tepat jika kita ingin algoritma menghasilkan solusi optimal.
- ▣ Jadi, pada sebagian masalah algoritma *greedy* tidak selalu berhasil memberikan solusi yang optimal.

Contoh 2:

Tinjau masalah penukaran uang.

(a) Koin: 5, 4, 3, dan 1

Uang yang ditukar = 7.

Solusi *greedy*: $7 = 5 + 1 + 1$ (3 koin) \square tidak optimal

Solusi optimal: $7 = 4 + 3$ (2 koin)

(b) Koin: 10, 7, 1

Uang yang ditukar: 15

Solusi *greedy*: $15 = 10 + 1 + 1 + 1 + 1 + 1$ (6 koin)

Solusi optimal: $15 = 7 + 7 + 1$ (hanya 3 koin)

(c) Koin: 15, 10, dan 1

Uang yang ditukar: 20

Solusi *greedy*: $20 = 15 + 1 + 1 + 1 + 1 + 1$ (6 koin)

Solusi optimal: $20 = 10 + 10$ (2 koin)

- Untuk sistem mata uang dollar AS, euro Eropa, dan *crown* Swedia, algoritma *greedy* selalu memberikan solusi optimum.
- Contoh: Uang \$6,39 ditukar dengan uang kertas (*bill*) dan koin sen (*cent*), kita dapat memilih:
 - Satu buah uang kertas senilai \$5
 - Satu buah uang kertas senilai \$1
 - Satu koin 25 sen
 - Satu koin 10 sen
 - Empat koin 1 sen

$$\begin{array}{l} \$5 + \$1 + 25c + 10c + 1c + 1c + 1c + 1c = \\ \$6,39 \end{array}$$

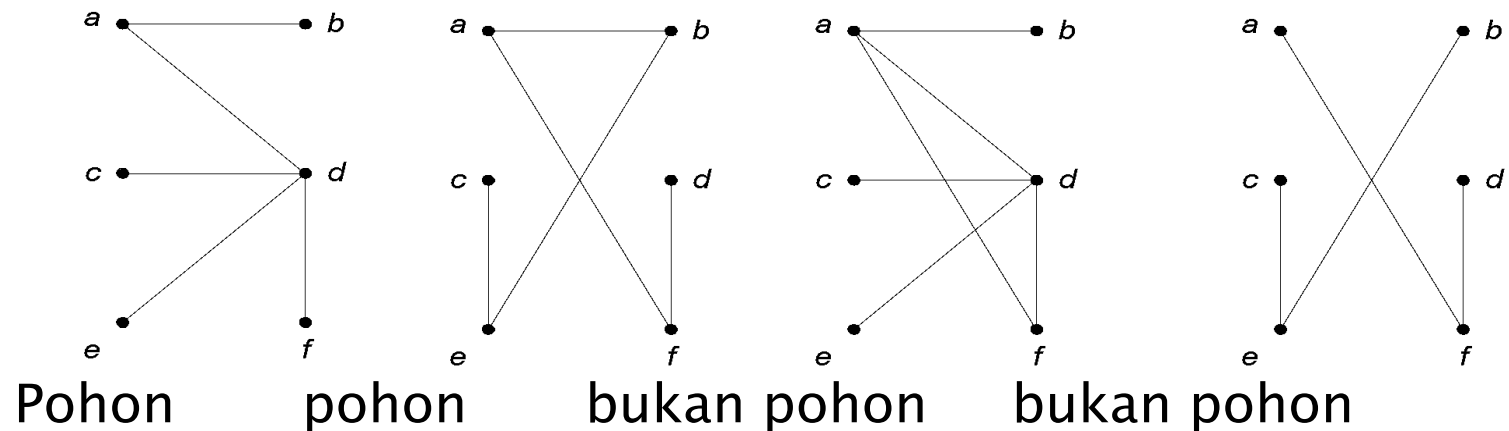
- Jika jawaban terbaik mutlak tidak diperlukan, maka algoritma *greedy* sering berguna untuk menghasilkan solusi hampiran (*approximation*),
- daripada menggunakan algoritma yang lebih rumit untuk menghasilkan solusi yang eksak.
- Bila algoritma *greedy* optimum, maka keoptimalannya itu dapat dibuktikan secara matematis

C. Contoh-contoh Algoritma Greedy

1. **Masalah penukaran uang**
2. **Minimisasi Waktu di dalam Sistem (Penjadwalan)**
3. ***An Activity Selection Problem***
4. ***Integer Knapsack, Fractional Knapsack***
5. **Penjadwalan *Job* dengan Tenggat Waktu (*Job Schedulling with Deadlines*)**
6. **Pohon Merentang Minimum**
7. **Lintasan Terpendek (*Shortest Path*)**
8. **Pemampatan Data dengan Algoritma Huffman**

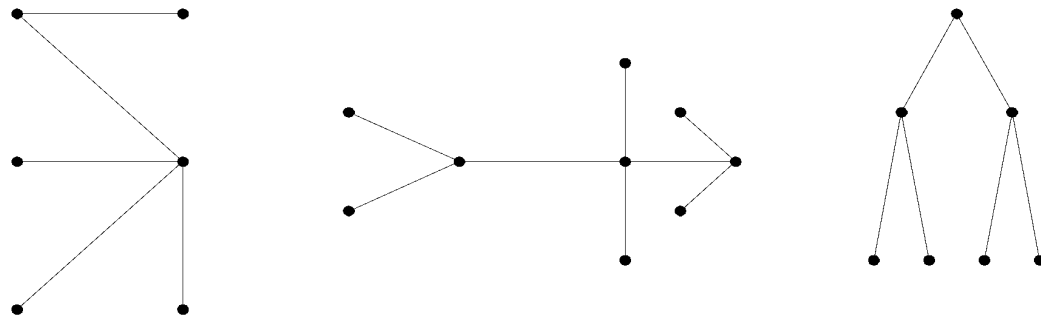
6. Pohon

Merupakan graf tak-berarah terhubung yang tidak mengandung sirkuit



Hutan (*forest*)

- kumpulan pohon yang saling lepas
- graf tidak terhubung yang tidak mengandung sirkuit. Setiap komponen di dalam graf terhubung tersebut adalah pohon.



Hutan yang terdiri dari tiga buah pohon

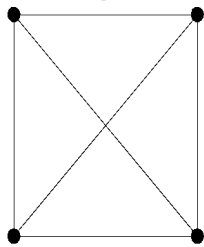
Sifat-sifat Pohon

Misalkan $G = (V, E)$ adalah graf tak-berarah sederhana dan jumlah simpulnya n . Maka, semua pernyataan di bawah ini adalah ekuivalen:

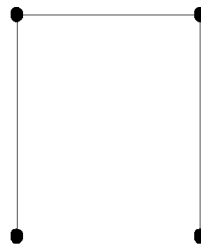
1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan intasan tunggal.
3. G terhubung dan memiliki $m = n - 1$ buah sisi.
4. G tidak mengandung sirkuit dan memiliki $m = n - 1$ buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan.

Pohon Merentang (*spanning tree*)

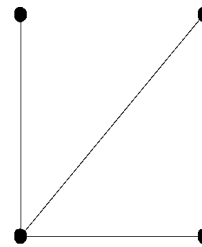
- Pohon merentang dari graf terhubung adalah upagraf merentang yang berupa pohon.
- Pohon merentang diperoleh dengan memutus sirkuit di dalam graf.



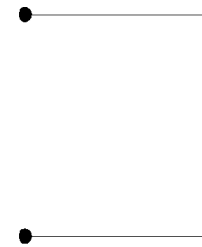
G



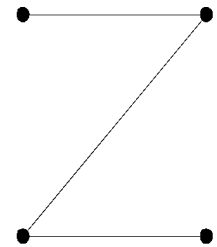
T_1



T_2



T_3



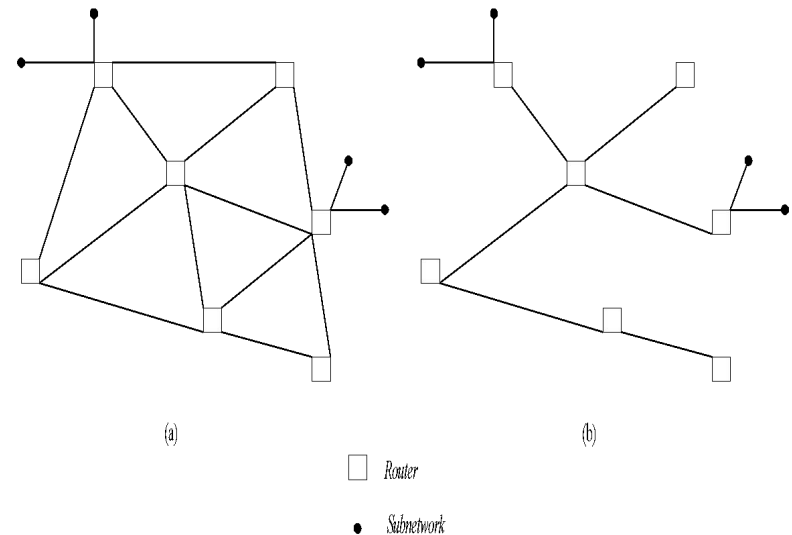
T_4

- Setiap graf terhubung mempunyai paling sedikit satu buah pohon merentang.
- Graf tak-terhubung dengan k komponen mempunyai k buah hutan merentang yang disebut hutan merentang (*spanning forest*).

Aplikasi Pohon Merentang

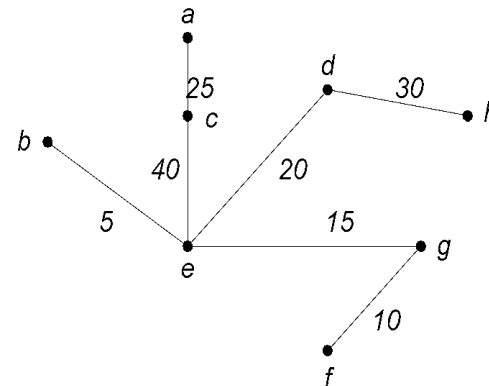
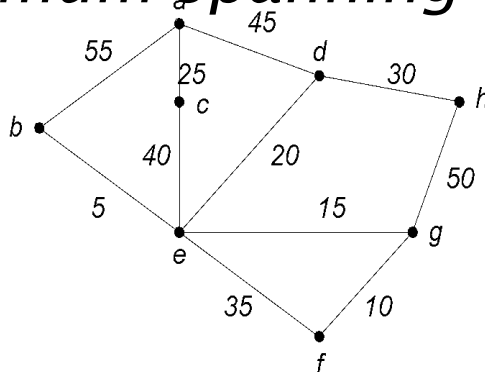
- Jalan-jalan seminimum mungkin yang menghubungkan semua kota sehingga setiap kota tetap terhubung satu sama lain.
- Perutean (*routing*) pesan pada jaringan komputer.

□ Contoh



Pohon Rentang Minimum

- Graf terhubung-berbobot mungkin mempunyai lebih dari 1 pohon merentang.
- Pohon rentang yang berbobot minimum –dinamakan **pohon merentang minimum** (*minimum spanning tree*).

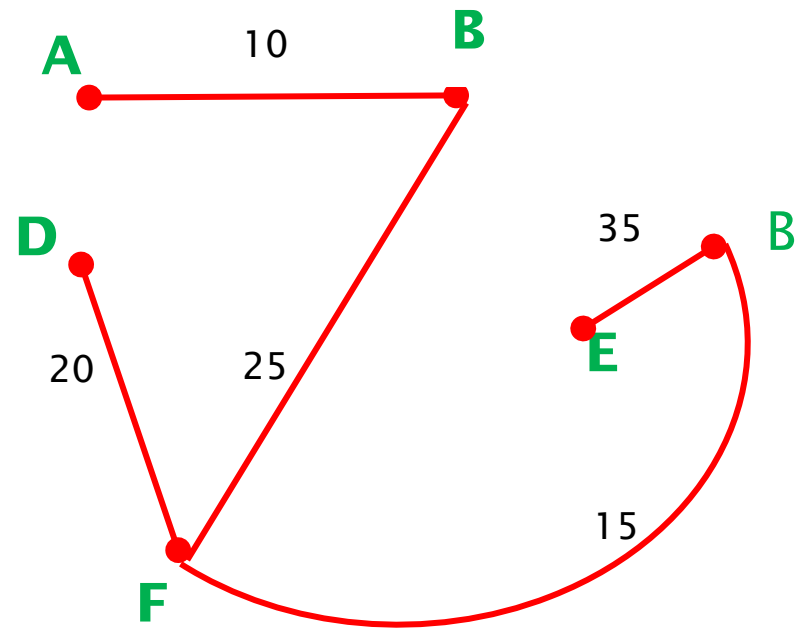
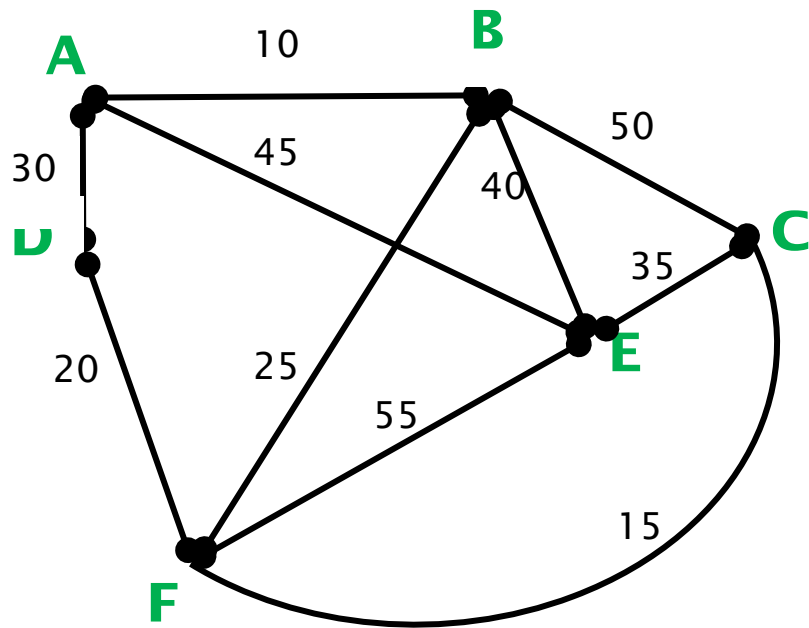


Minimum Spanning Tree/ Pohon Merentang Minimum

- Adalah sebuah **spanning tree yang terhubung**, graph tidak berarah. Menghubungkan **seluruh vertices** (simpul) secara bersama dan memiliki total **minimal** bobot dari seluruh edge (tepi) nya.
- Graph terdiri dari vertices dan edge $G=(v,e)$

Contoh Graph

(a) Graph $G=(V,E)$ (b) Minimum Spanning Tree



Algoritma Untuk Mendapatkan Minimum Spanning Tree

Ada 2 yaitu :

1. Algoritma Prim
2. Algoritma Kruskal

Algoritma **Prim** Untuk Minimum Spanning Tree

- Strategi **Greedy** yang digunakan:
 - *Pada setiap langkah, pilih sisi E dari graf $G(V,E)$ yang mempunyai **bobot minimum** dengan syarat sisi E tersebut tetap terhubung dengan pohon merentang minimum T yang telah terbentuk.*

Pseudocode **Algoritma Prim**

Procedure **Prim**(input G : graf, output T : pohon)

{ Membentuk minimum spanning tree T dari graf terhubung G .

Masukan: graf-berbobot terhubung $G = (V, E)$, dimana $|V| = n$

Keluaran: pohon rentang minimum $T = (V, E')$ }

Deklarasi

i, p, q, u, v : integer

Algoritma

Cari sisi (p,q) dari E yang berbobot terkecil

$T \leftarrow \{(p,q)\}$

for $i \leftarrow 1$ to $n-2$ **do**

 Pilih sisi (u,v) dari E yang bobotnya terkecil namun bersisian dengan suatu simpul di dalam T

$T \leftarrow T \cup \{(u,v)\}$

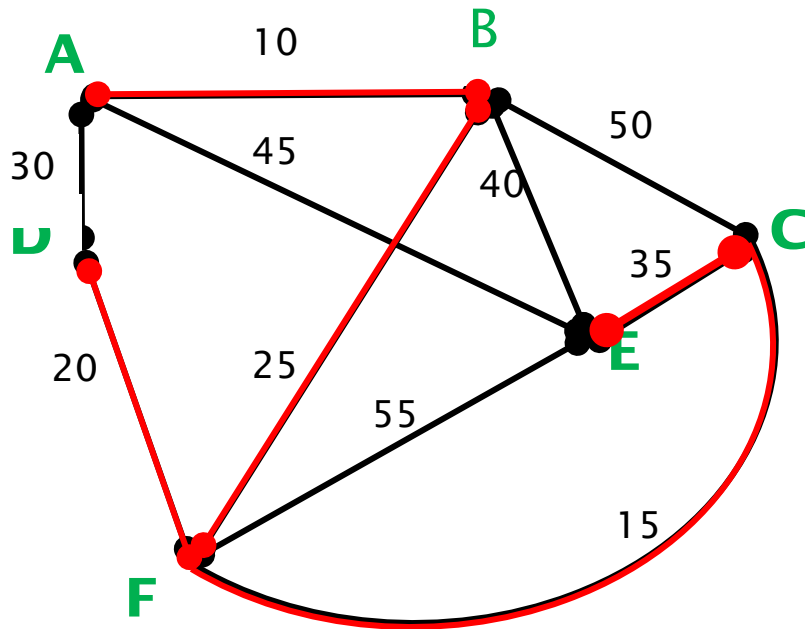
endfor

Algoritma Prim

- Pada algoritma prim, **dimulai** pada vertex V yang mempunyai sisi E dengan **bobot terkecil**.
- Sisi yang dimasukkan ke dalam himpunan T adalah sisi graph G yang bersisian dengan sebuah simpul di T, sedemikian sehingga T adalah Tree (pohon). Sisi dari Graph G ditambahkan ke T **jika ia tidak membentuk cycle**.
- **Prosedur** Algoritma Prim:
 1. Ambil sisi (edge) dari graph yg berbobot minimum, masukkan ke dalam T
 2. Pilih sisi (edge) (i,j) yg berbobot minimum dan bersisian dengan simpul di T, tetapi (i,j) tidak membentuk cycle di T. tambahkan (i,j) ke dalam T
 3. Ulangi prosedur no 2 sebanyak (n-2) kali

Kompleksitas nya **$O(n^2)$**

Minimum Spanning Tree dengan Algoritma Prim



Langka h	Edge	Bobot
1	(A,B)	10
2	(B,F)	25
3	(F,C)	15
4	(F,D)	20
5	(C,E)	35

Total :
105

Algoritma **Kruskal** Untuk Minimum Spanning Tree

- Strategi **Greedy** yang digunakan:
 - Pada setiap langkah, pilih *sisi dari graf G yang mempunyai **bobot minimum** tetapi sisi tersebut tidak membentuk sirkuit di T .*

Algoritma Kruskal

- Pada algoritma kruskal, sisi (edge) dari Graph diurut terlebih dahulu berdasarkan bobotnya dari kecil ke besar.
- □ Sisi yang dimasukkan ke dalam himpunan T adalah sisi graph G yang sedemikian sehingga T adalah Tree (pohon). Sisi dari Graph G ditambahkan ke T jika ia tidak membentuk cycle.

- **Prosedur** Algoritma Kruskal:

1. T masih kosong
2. Pilih sisi (i,j) dengan bobot minimum
3. Pilih sisi (i,j) dengan bobot minimum berikutnya yang tidak membentuk cycle di T, tambahkan (i,j) ke T
4. Ulangi langkah 3 sebanyak (n-2) kali.
5. Total langkah (n-1) kali

Kompleksitas nya $O(|E| \log |E|)$ dimana E adalah jumlah sisi dalam Graph

Pseudocode **Algoritma** **Kruskal**

Procedure **Kruskal**(input G : graf, output T : pohon)

{ Membentuk pohon merentang minimum T dari graf terhubung G .

Masukan: graf-berbobot terhubung $G = (V, E)$, yang mana $|V| = n$

Keluaran: pohon rentang minimum $T = (V, E')$ }

Deklarasi

i, p, q, u, v : integer

Algoritma

{Asumsi: sisi-sisi dari graf sudah diurut menaik berdasarkan bobotnya }

$T \leftarrow \{\}$

while jumlah sisi di dalam $T < n-1$ **do**

Pilih sisi (u,v) dari E yang bobotnya terkecil

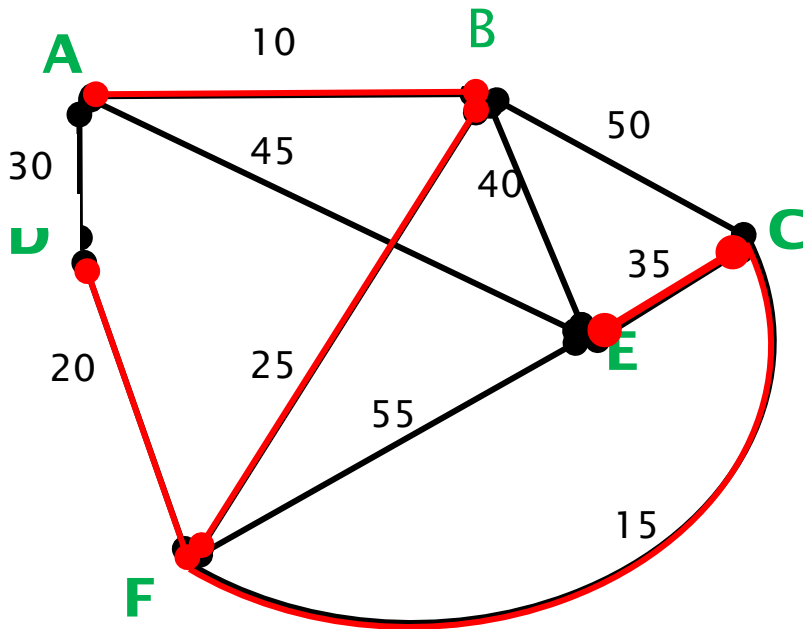
if (u,v) tidak membentuk siklus di T **then**

$T \leftarrow T \cup \{(u,v)\}$

endif

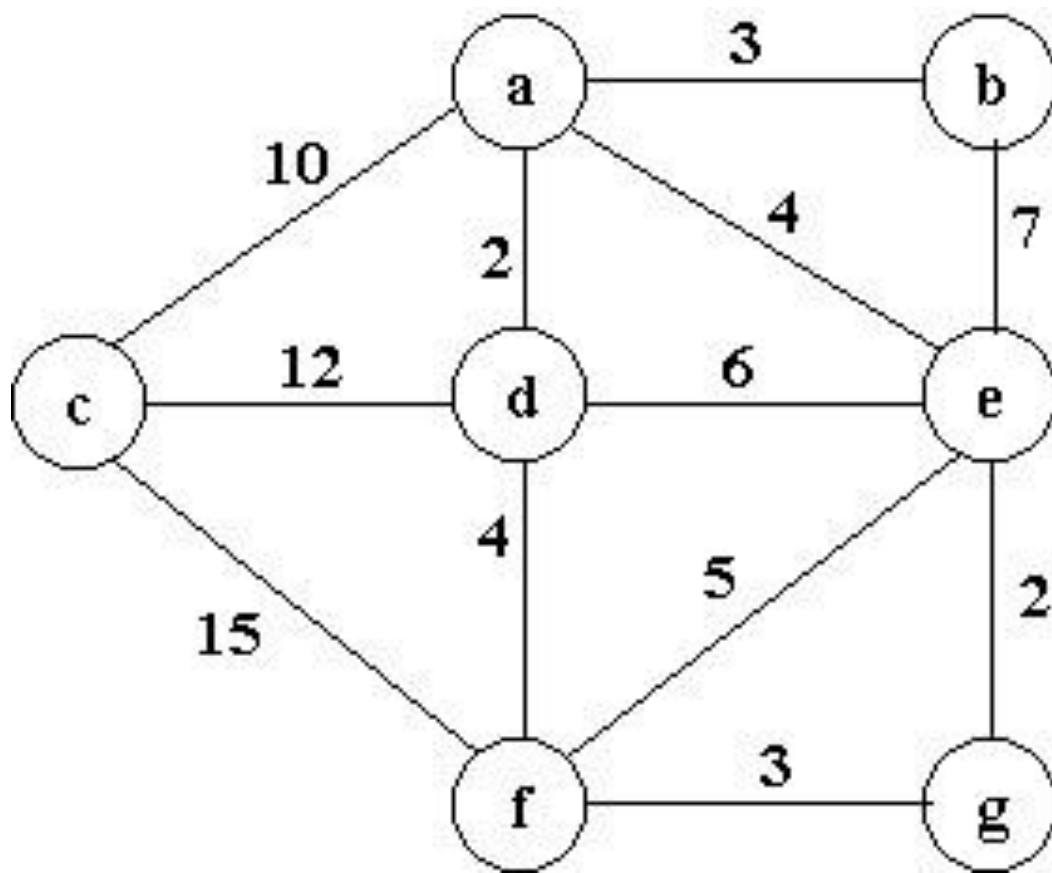
endwhile

Minimum Spanning Tree dengan Algoritma Kruskal

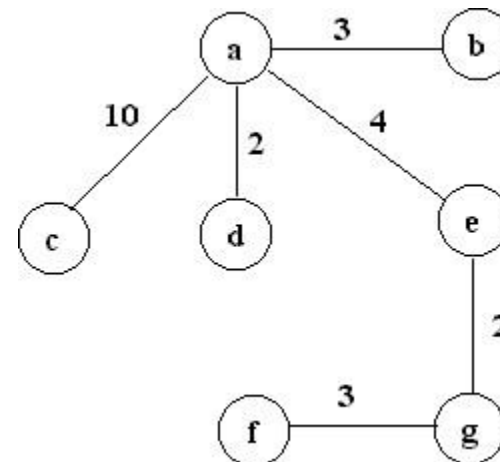
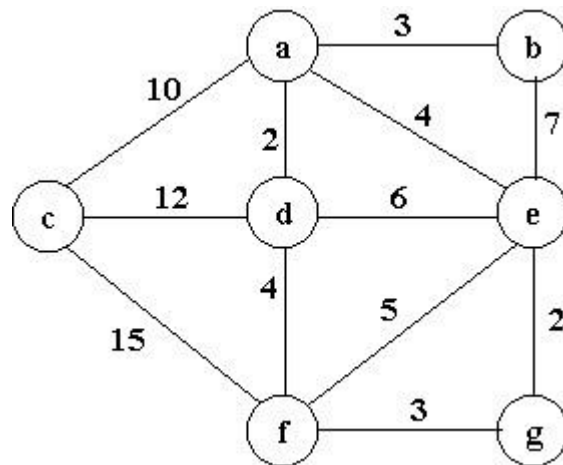


Langka h	Edge	Bobot
1	(A,B)	10
2	(C,F)	15
3	(F,D)	20
4	(B,F)	25
5	(C,E)	35

Total :
105

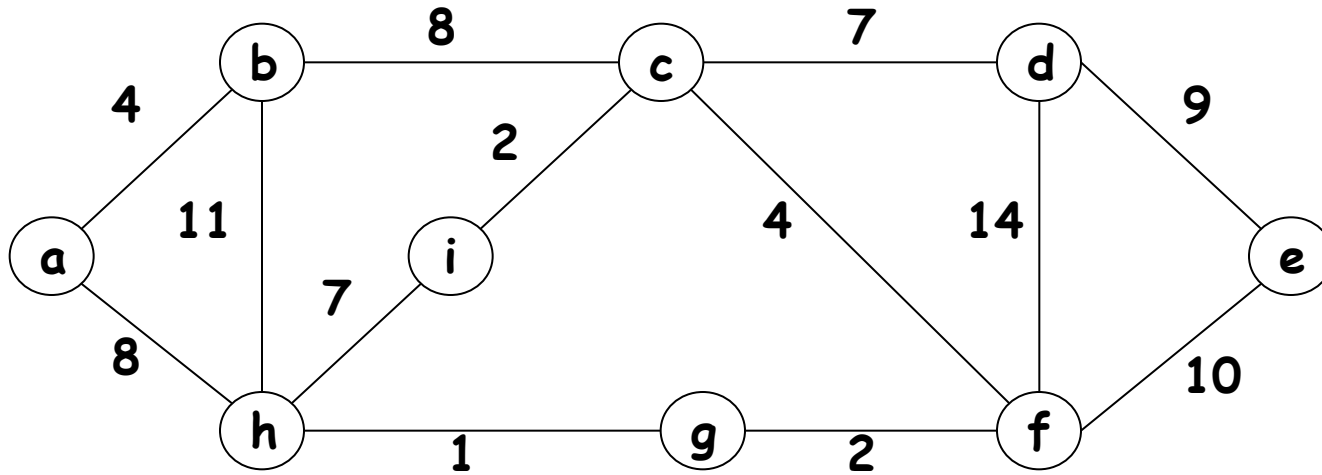


edge	ad	eg	ab	fg	ae	df	ef	de	be	ac	cd	cf
weight	2	2	3	3	4	4	5	6	7	10	12	15
insertion status	✓	✓	✓	✓	✓	x	x	x	x	✓	x	x
insertion order	1	2	3	4	5					6		



The minimum cost is: 24

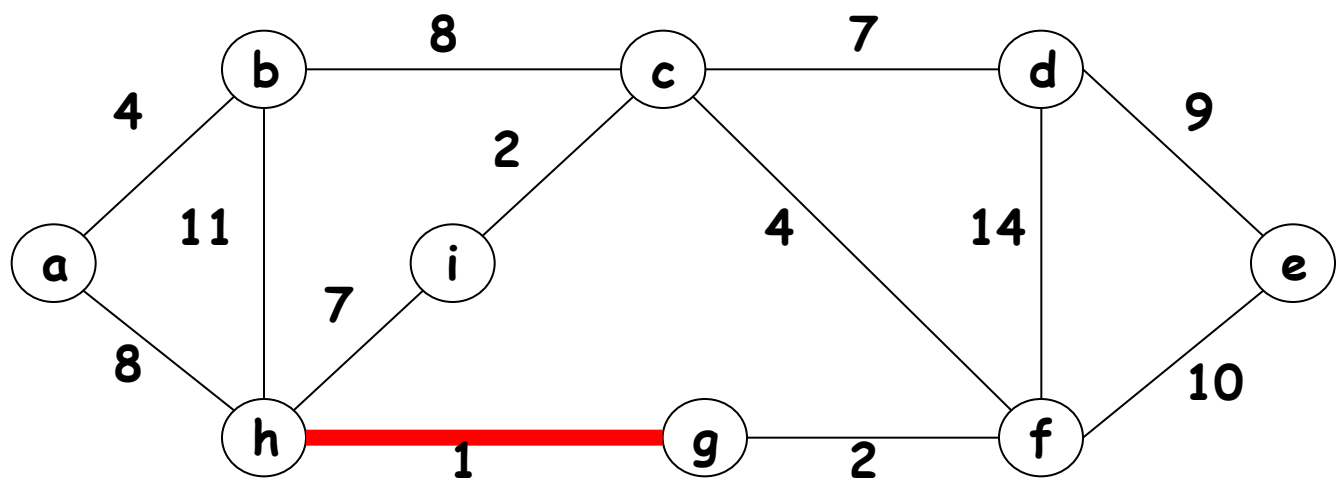
Kruskal's algorithm - MST



Arrange edges from smallest to largest weight

(h,g)	1
(i,c)	2
(g,f)	2
(a,b)	4
(c,f)	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

Kruskal's algorithm - MST

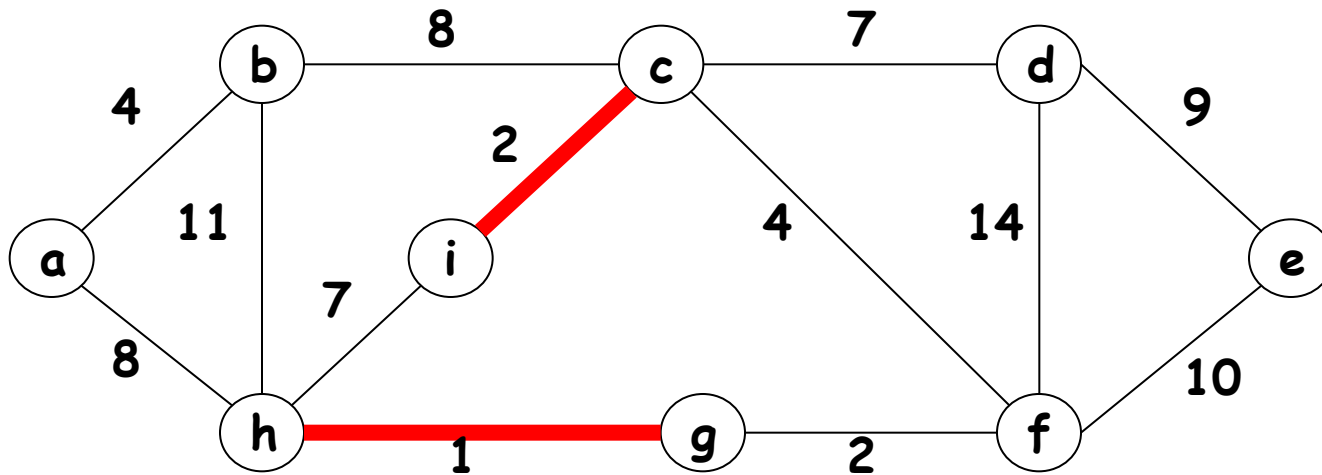


Choose the minimum weight edge

<i>(h,g)</i>	<i>1</i>
(i,c)	2
(g,f)	2
(a,b)	4
(c,f)	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

Kruskal's algorithm - MST

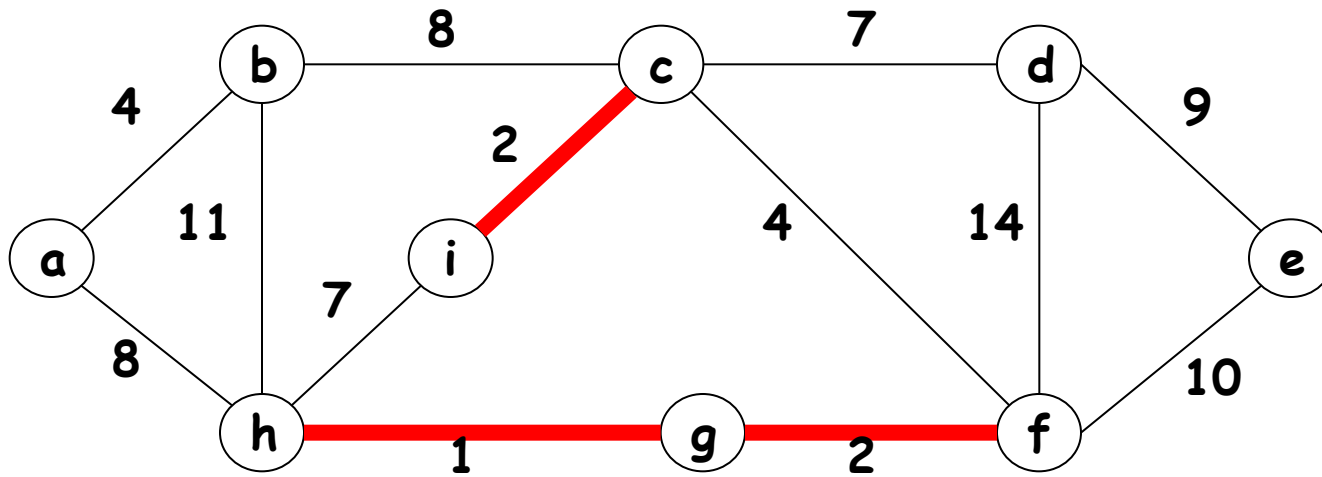


Choose the next minimum weight edge

<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
(g,f)	2
(a,b)	4
(c,f)	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

Kruskal's algorithm - MST



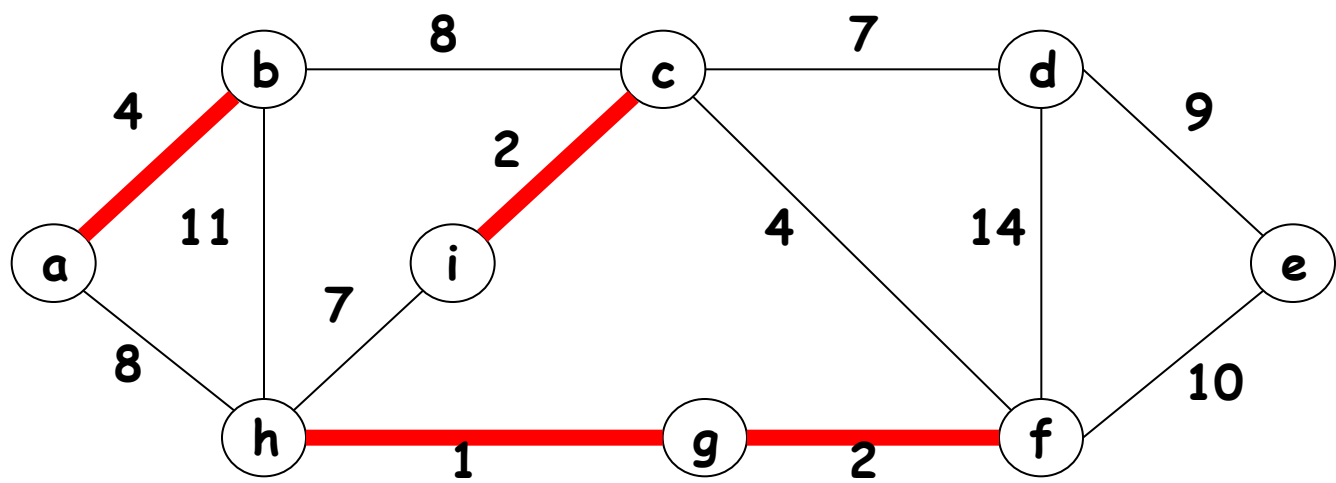
Continue as long as no cycle forms

<i>(h,g)</i>	1
<i>(i,c)</i>	2
<i>(g,f)</i>	2
(a,b)	4
(c,f)	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

42
(G
re
ed
y)

Kruskal's algorithm - MST



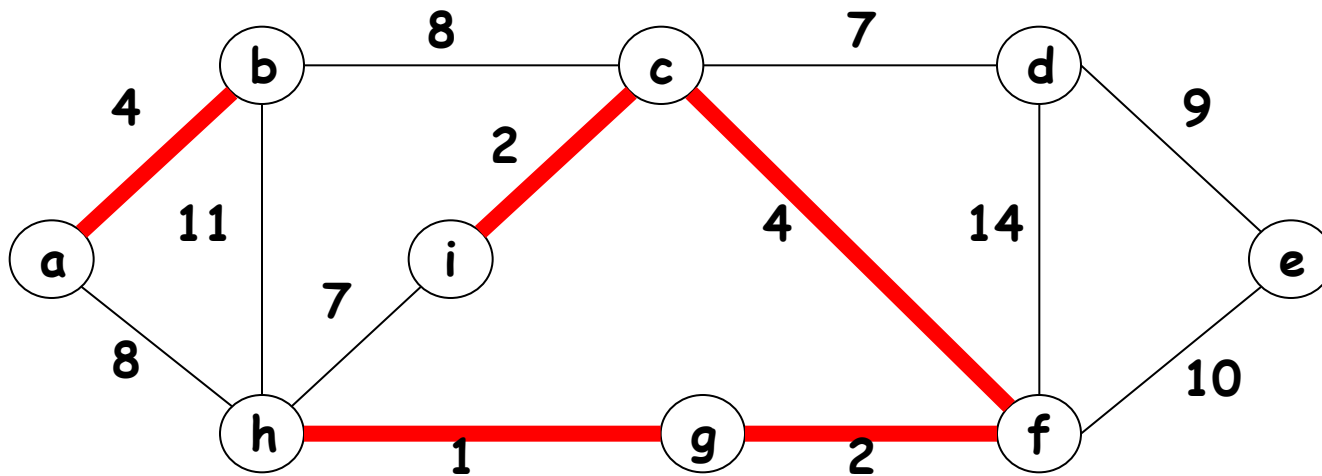
Continue as long as no cycle forms

<i>(h,g)</i>	1
<i>(i,c)</i>	2
<i>(g,f)</i>	2
<i>(a,b)</i>	4
(c,f)	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

43
(G
re
ed
y)

Kruskal's algorithm - MST

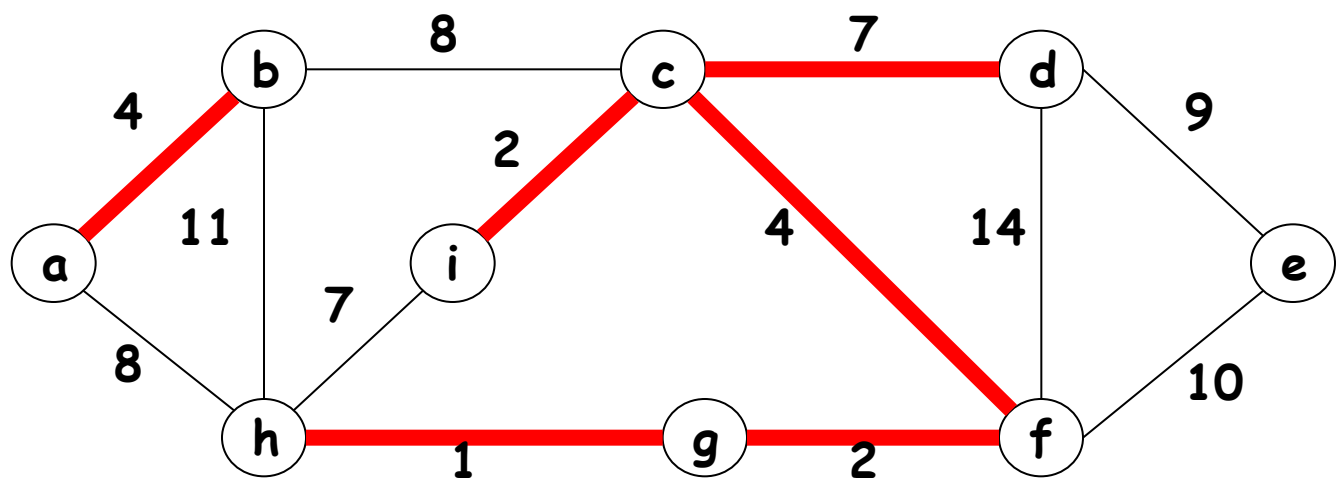


Continue as long as no cycle forms

<i>(h,g)</i>	1
<i>(i,c)</i>	2
<i>(g,f)</i>	2
<i>(a,b)</i>	4
<i>(c,f)</i>	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

Kruskal's algorithm - MST



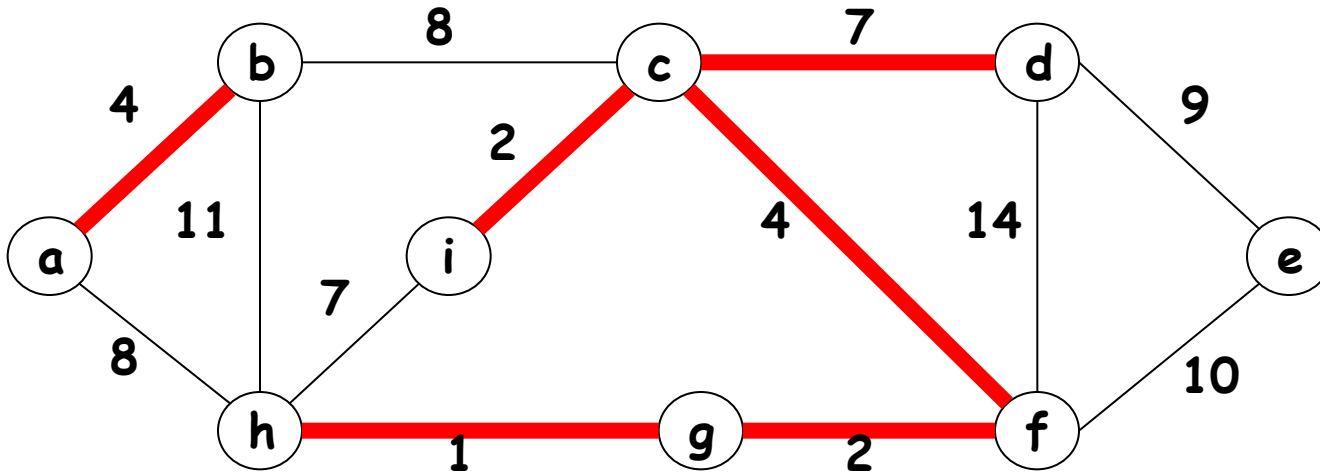
Continue as long as no cycle forms



<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
<i>(g,f)</i>	<i>2</i>
<i>(a,b)</i>	<i>4</i>
<i>(c,f)</i>	<i>4</i>
<i>(c,d)</i>	<i>7</i>
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

Kruskal's algorithm - MST

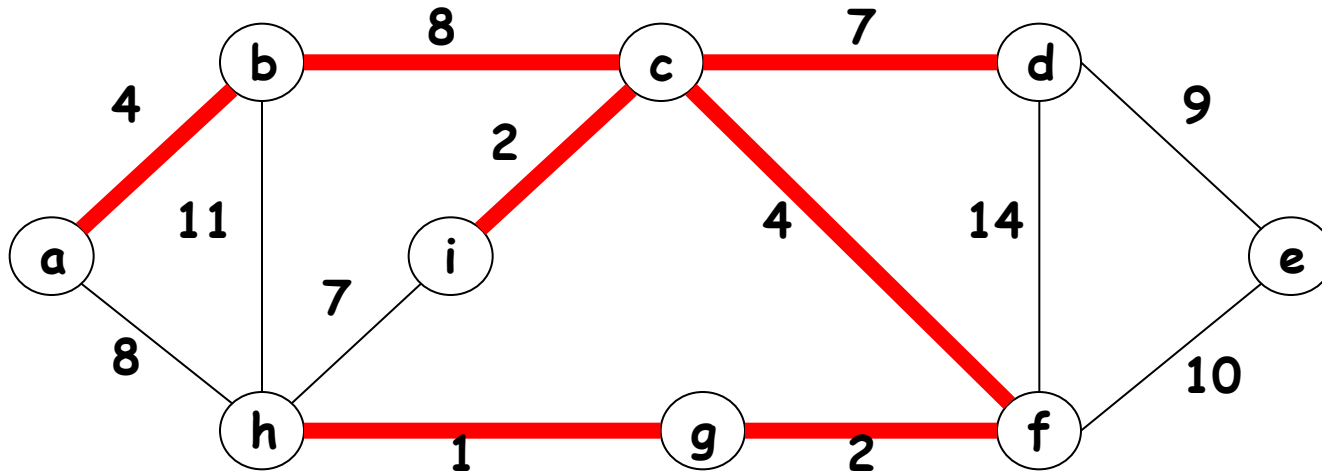


(h,i) cannot be included, otherwise, a cycle is formed

<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
<i>(g,f)</i>	<i>2</i>
<i>(a,b)</i>	<i>4</i>
<i>(c,f)</i>	<i>4</i>
<i>(c,d)</i>	<i>7</i>
<i>(h,i)</i>	<i>7</i>
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

Kruskal's algorithm - MST

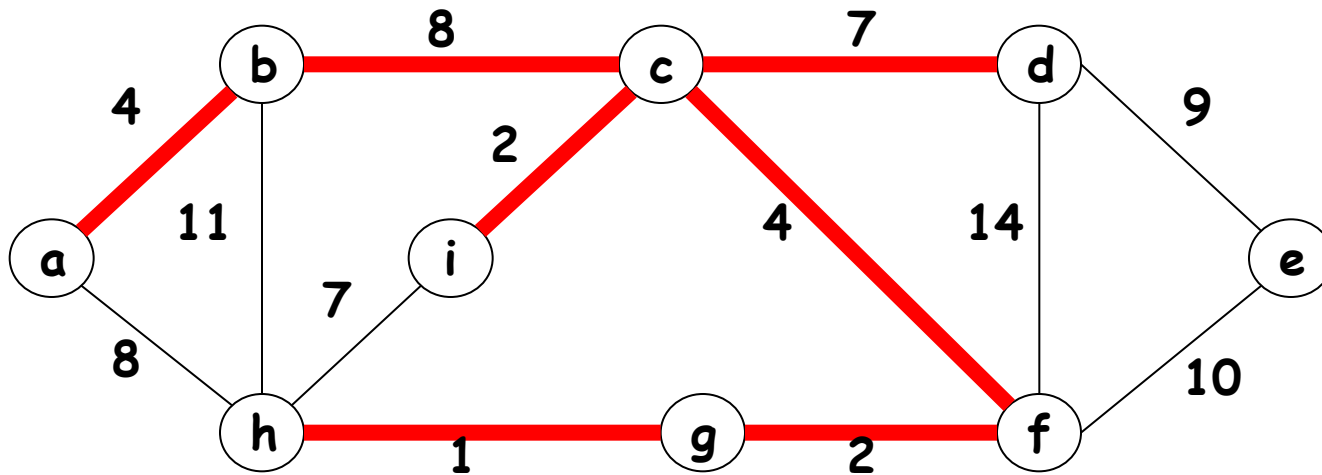


Choose the next minimum weight edge

<i>(h,g)</i>	1
<i>(i,c)</i>	2
<i>(g,f)</i>	2
<i>(a,b)</i>	4
<i>(c,f)</i>	4
<i>(c,d)</i>	7
<i>(h,i)</i>	7
<i>(b,c)</i>	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

Kruskal's algorithm - MST



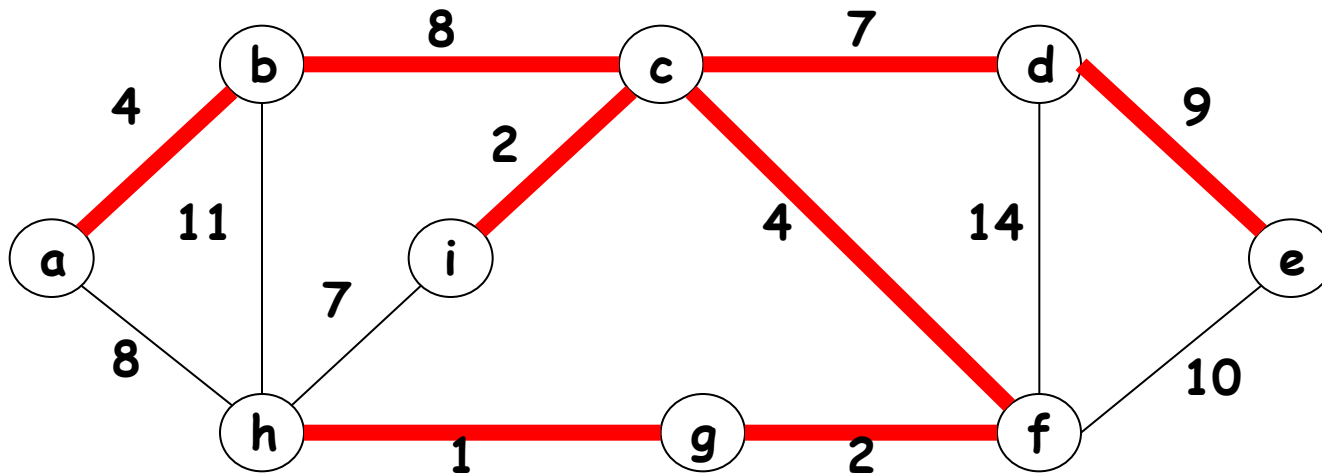
(a,h) cannot be included, otherwise, a cycle is formed

<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
<i>(g,f)</i>	<i>2</i>
<i>(a,b)</i>	<i>4</i>
<i>(c,f)</i>	<i>4</i>
<i>(c,d)</i>	<i>7</i>
(h,i)	7
<i>(b,c)</i>	<i>8</i>
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

48
(G
re
ed
y)

Kruskal's algorithm - MST



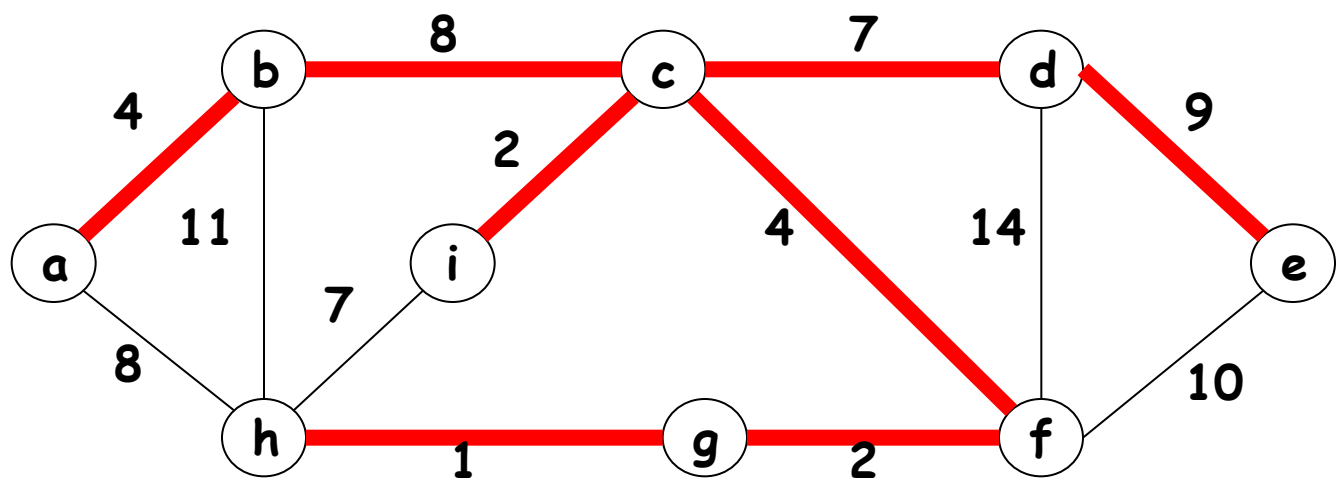
Choose the next minimum weight edge

<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
<i>(g,f)</i>	<i>2</i>
<i>(a,b)</i>	<i>4</i>
<i>(c,f)</i>	<i>4</i>
<i>(c,d)</i>	<i>7</i>
(h,i)	7
<i>(b,c)</i>	<i>8</i>
(a,h)	8
<i>(d,e)</i>	<i>9</i>
(f,e)	10
(b,h)	11
(d,f)	14



italic: chosen

Kruskal's algorithm - MST



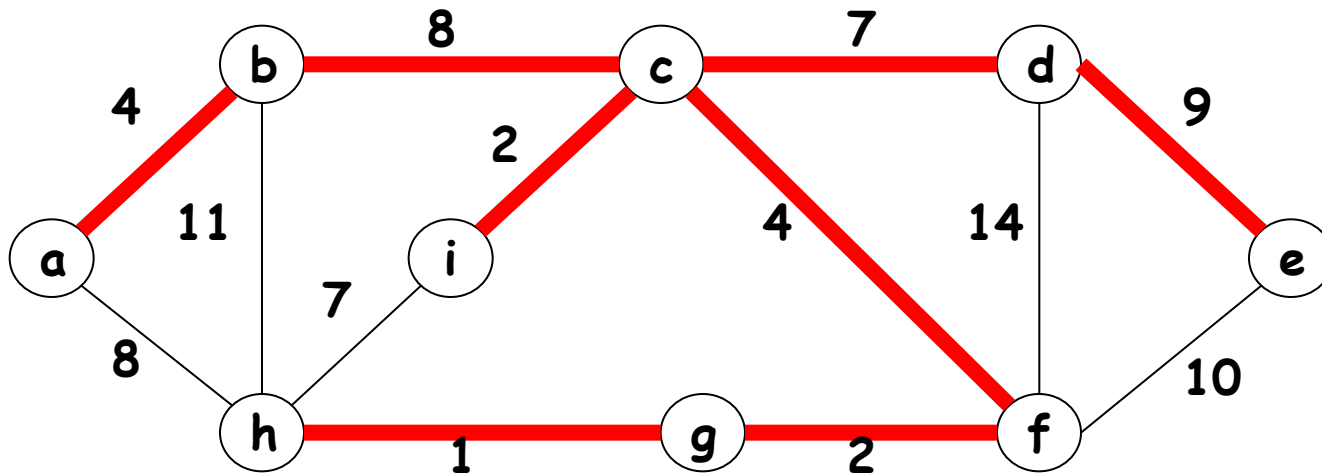
(f,e) cannot be included, otherwise, a cycle is formed

<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
<i>(g,f)</i>	<i>2</i>
<i>(a,b)</i>	<i>4</i>
<i>(c,f)</i>	<i>4</i>
<i>(c,d)</i>	<i>7</i>
(h,i)	7
<i>(b,c)</i>	<i>8</i>
(a,h)	8
<i>(d,e)</i>	<i>9</i>
(f,e)	10
<i>(b,h)</i>	<i>11</i>
<i>(d,f)</i>	<i>14</i>



italic: chosen

Kruskal's algorithm - MST



(b,h) cannot be included, otherwise, a cycle is formed

<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
<i>(g,f)</i>	<i>2</i>
<i>(a,b)</i>	<i>4</i>
<i>(c,f)</i>	<i>4</i>
<i>(c,d)</i>	<i>7</i>
(h,i)	7
<i>(b,c)</i>	<i>8</i>
(a,h)	8
<i>(d,e)</i>	<i>9</i>
(f,e)	10
(b,h)	11
<i>(d,f)</i>	<i>14</i>

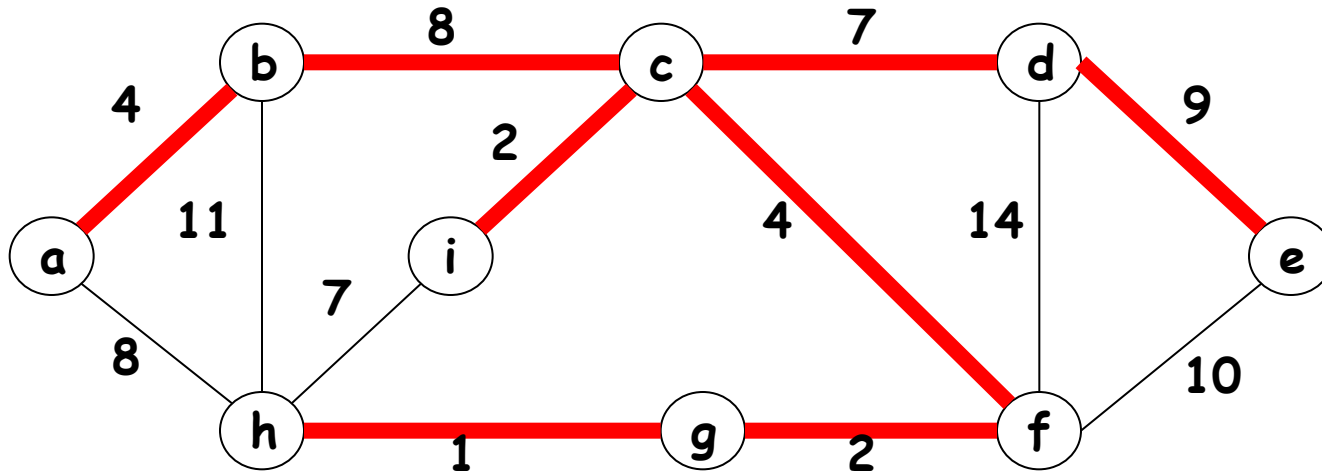


italic: chosen

51

(G
re
ed
y)

Kruskal's algorithm - MST



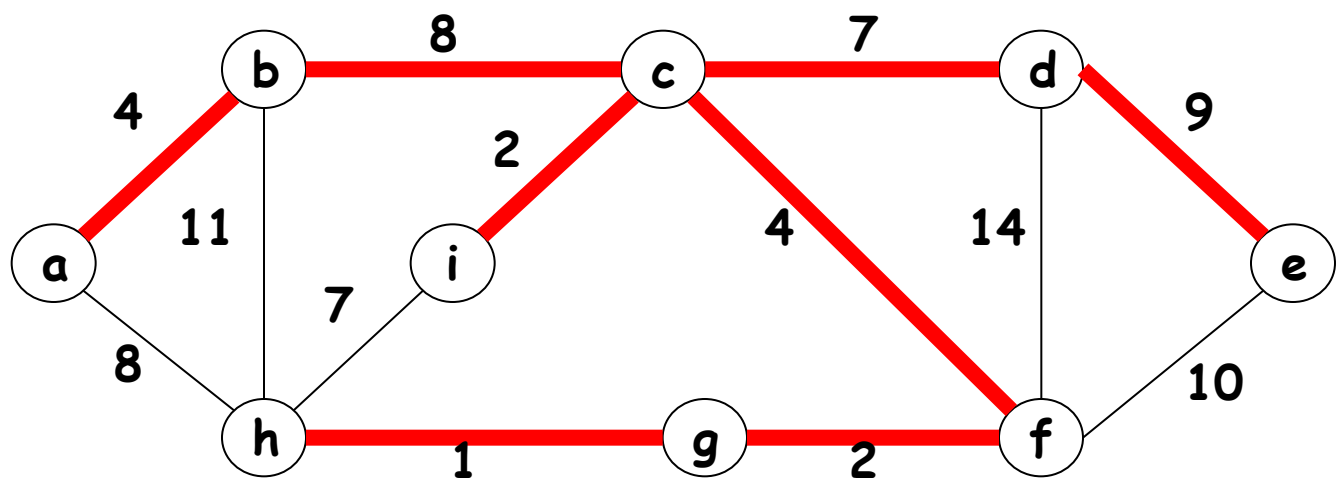
(d,f) cannot be included, otherwise, a cycle is formed

<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
<i>(g,f)</i>	<i>2</i>
<i>(a,b)</i>	<i>4</i>
<i>(c,f)</i>	<i>4</i>
<i>(c,d)</i>	<i>7</i>
(h,i)	7
<i>(b,c)</i>	<i>8</i>
(a,h)	8
<i>(d,e)</i>	<i>9</i>
(f,e)	10
(b,h)	11
(d,f)	14



italic: chosen

Kruskal's algorithm - MST



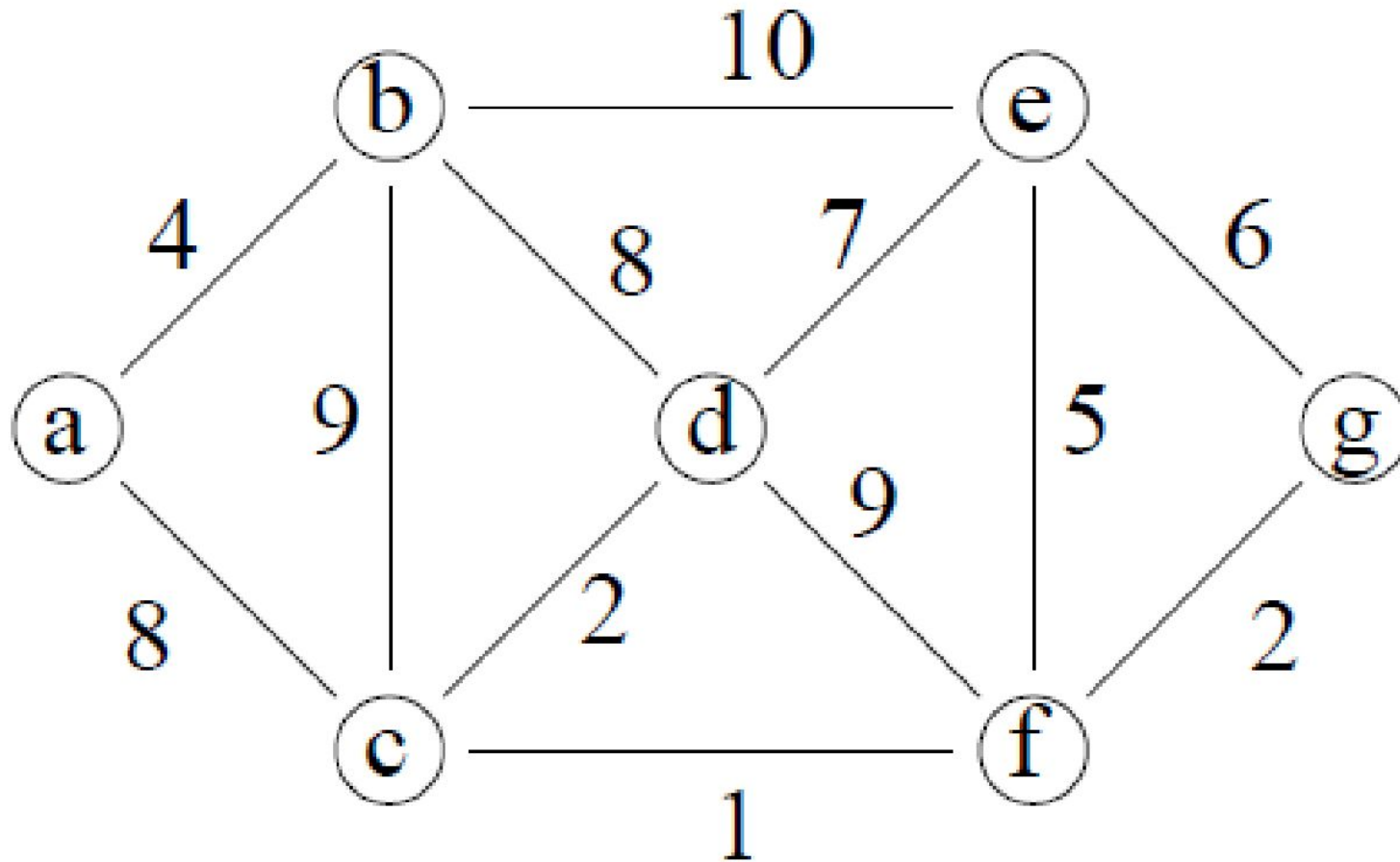
MST is found when all edges are examined

<i>(h,g)</i>	<i>1</i>
<i>(i,c)</i>	<i>2</i>
<i>(g,f)</i>	<i>2</i>
<i>(a,b)</i>	<i>4</i>
<i>(c,f)</i>	<i>4</i>
<i>(c,d)</i>	<i>7</i>
(h,i)	7
<i>(b,c)</i>	<i>8</i>
(a,h)	8
<i>(d,e)</i>	<i>9</i>
(f,e)	10
(b,h)	11
(d,f)	14

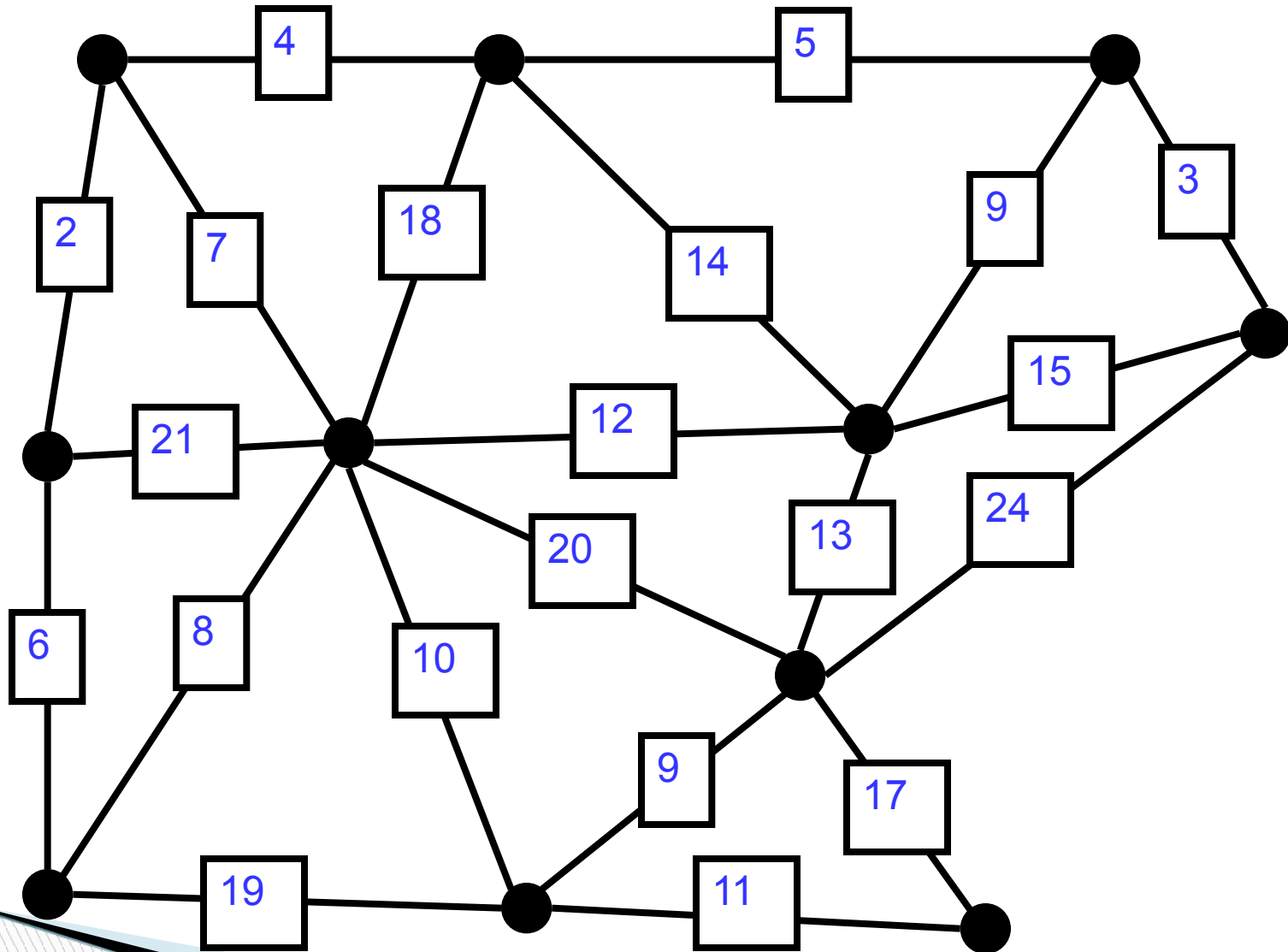
italic: chosen

Tentukan Minimum Spanning Tree dari Graph
(1 dan 2) dengan algoritma Prim dan Kruskal

SOAL 1.



SOAL 2



Kesimpulan Prim dan Kruskal

- Baik algoritma Prim maupun algoritma Kruskal, keduanya **selalu berhasil** menemukan minimum spanning tree (pohon merentang minimum).

7. Lintasan Terpendek (*Shortest Path*)

Beberapa macam persoalan lintasan terpendek:

- Lintasan terpendek antara dua buah simpul tertentu (*a pair shortest path*).
- Lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).
- Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shortest path*).
- Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

Persoalan:

Diberikan graf berbobot $G = (V, E)$. Tentukan lintasan terpendek dari sebuah simpul asal a ke setiap simpul lainnya di G .

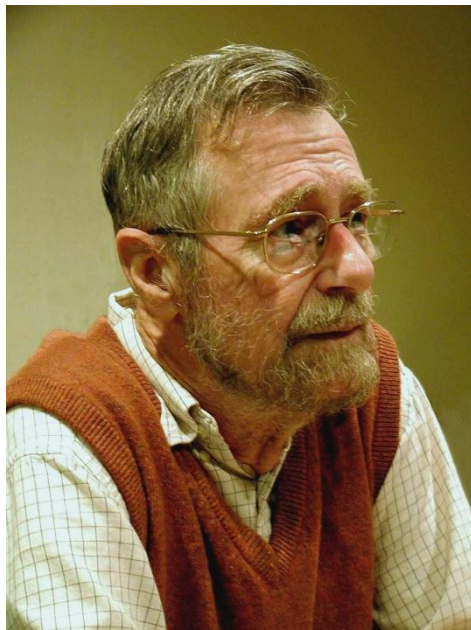
Asumsi yang kita buat adalah bahwa semua sisi berbobot positif.

Algoritma Dijkstra

Strategi *greedy*:

Pada setiap langkah, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang sudah terpilih dengan sebuah simpul lain yang belum terpilih.

Lintasan dari simpul asal ke simpul yang baru haruslah merupakan lintasan yang terpendek diantara semua lintasannya ke simpul-simpul yang belum terpilih.



Edsger W. Dijkstra (1930–2002)

- Edsger Wybe Dijkstra was one of the most influential members of computing science's founding generation. Among the domains in which his scientific contributions are fundamental are
- algorithm design
- programming languages
- program design
- operating systems
- distributed processing

In addition, Dijkstra was intensely interested in teaching, and in the relationships between academic computing science and the software industry. During his forty-plus years as a computing scientist, which included positions in both academia and industry, Dijkstra's contributions brought him many prizes and awards, including computing science's highest honor, the ACM Turing Award.

Sumber: <http://www.cs.texas.edu/users/EWD/>

procedure Dijkstra (**input** G: weighted_graph), **input** a: intial_vertex)
Deklarasi:

S : himpunan simpul solusi

L : array[1..n] of real { *L(z) berisi panjang lintasan terpendek dari a ke z* }

Algoritma

for $i \leftarrow 1$ **to** n

$L(v_i) \leftarrow \infty$

end for

$L(a) \leftarrow 0$; $S \leftarrow \{ \}$

while $z \notin S$ **do**

u \leftarrow simpul yang bukan di dalam S dan memiliki L(u) minimum

$S \leftarrow S \cup \{u\}$

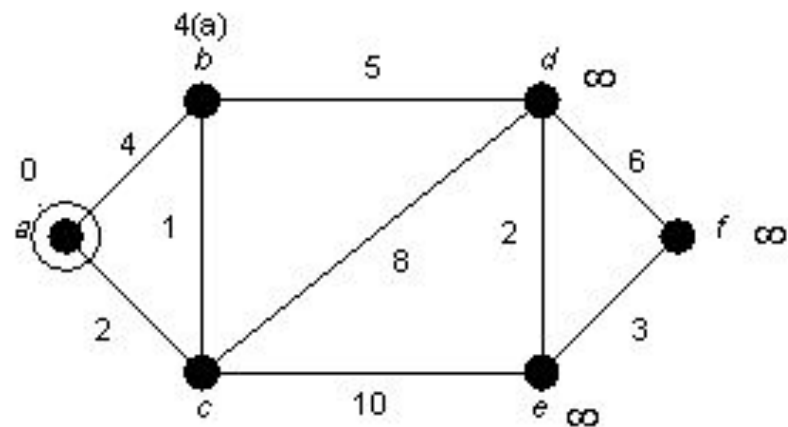
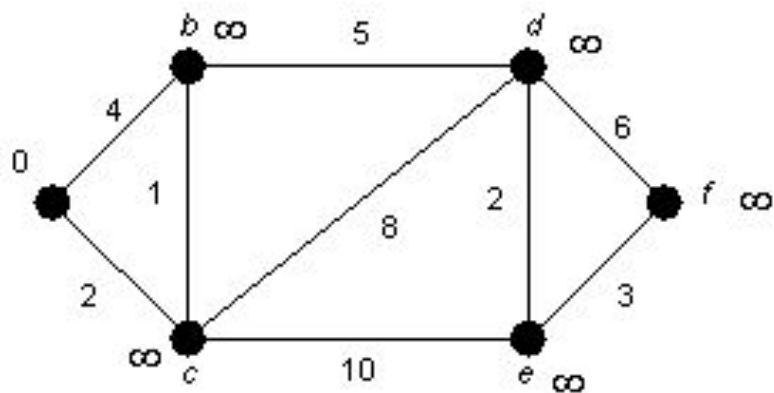
for semua simpul v yang tidak terdapat di dalam S

if $L(u) + G(u,v) < L(v)$ then $L(v) \leftarrow L(u) + G(u,v)$

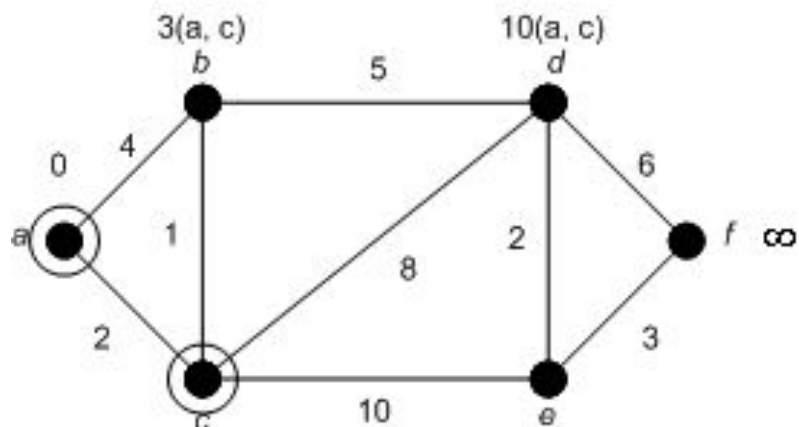
end for

end while

Algoritma Dijkstra memiliki kompleksitas = $O(V \log V + E)$

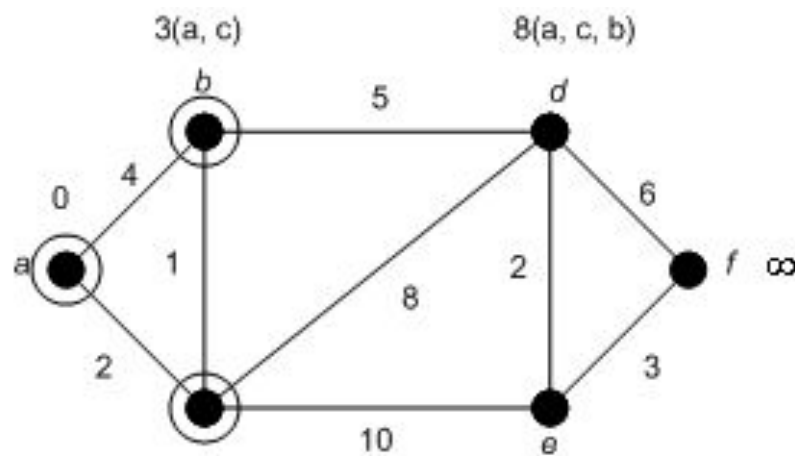


2(a)



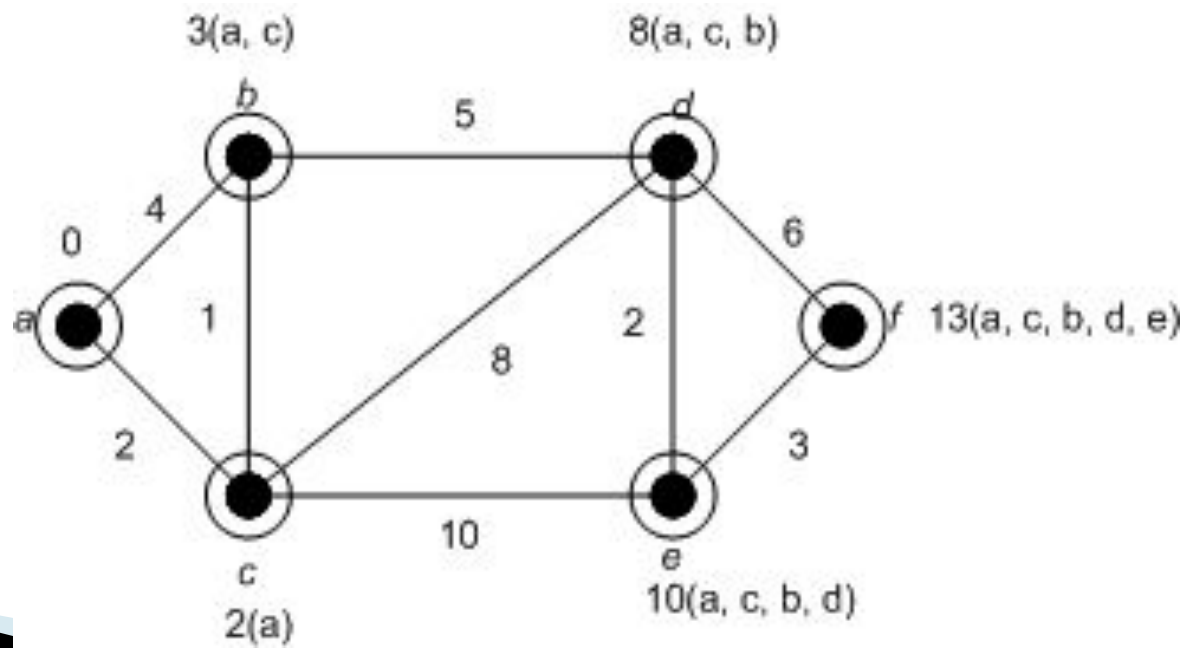
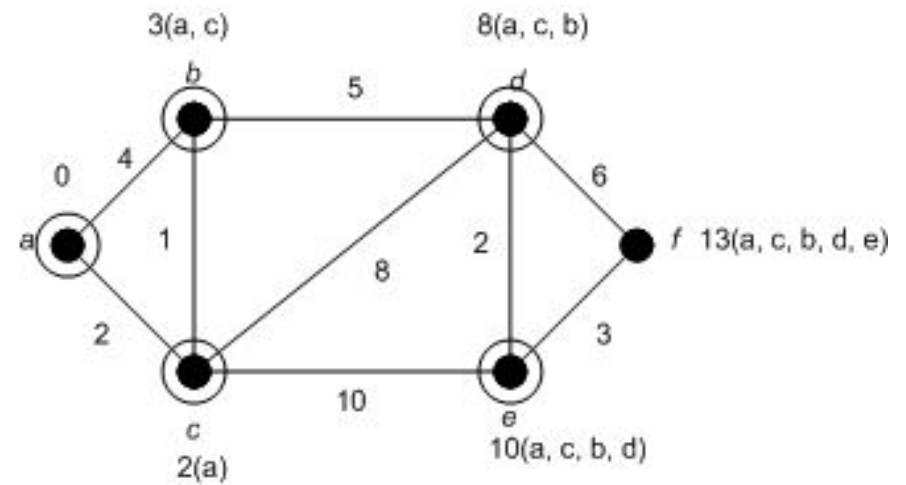
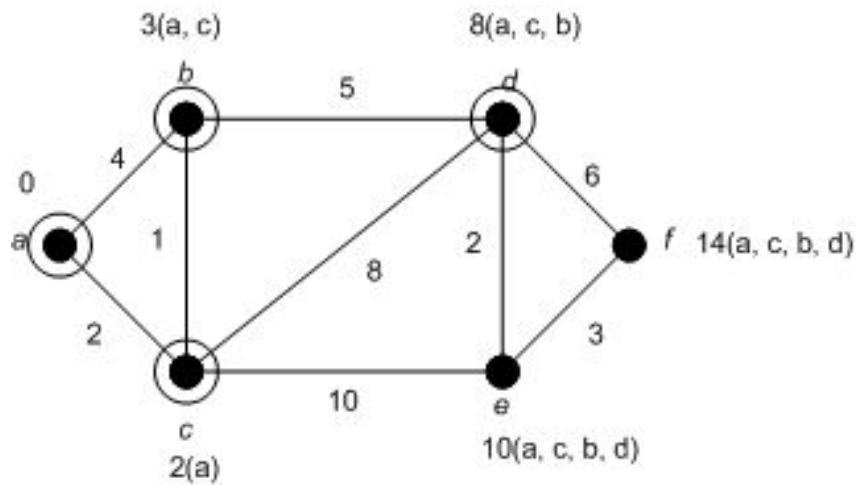
2(a)

12(a, c)



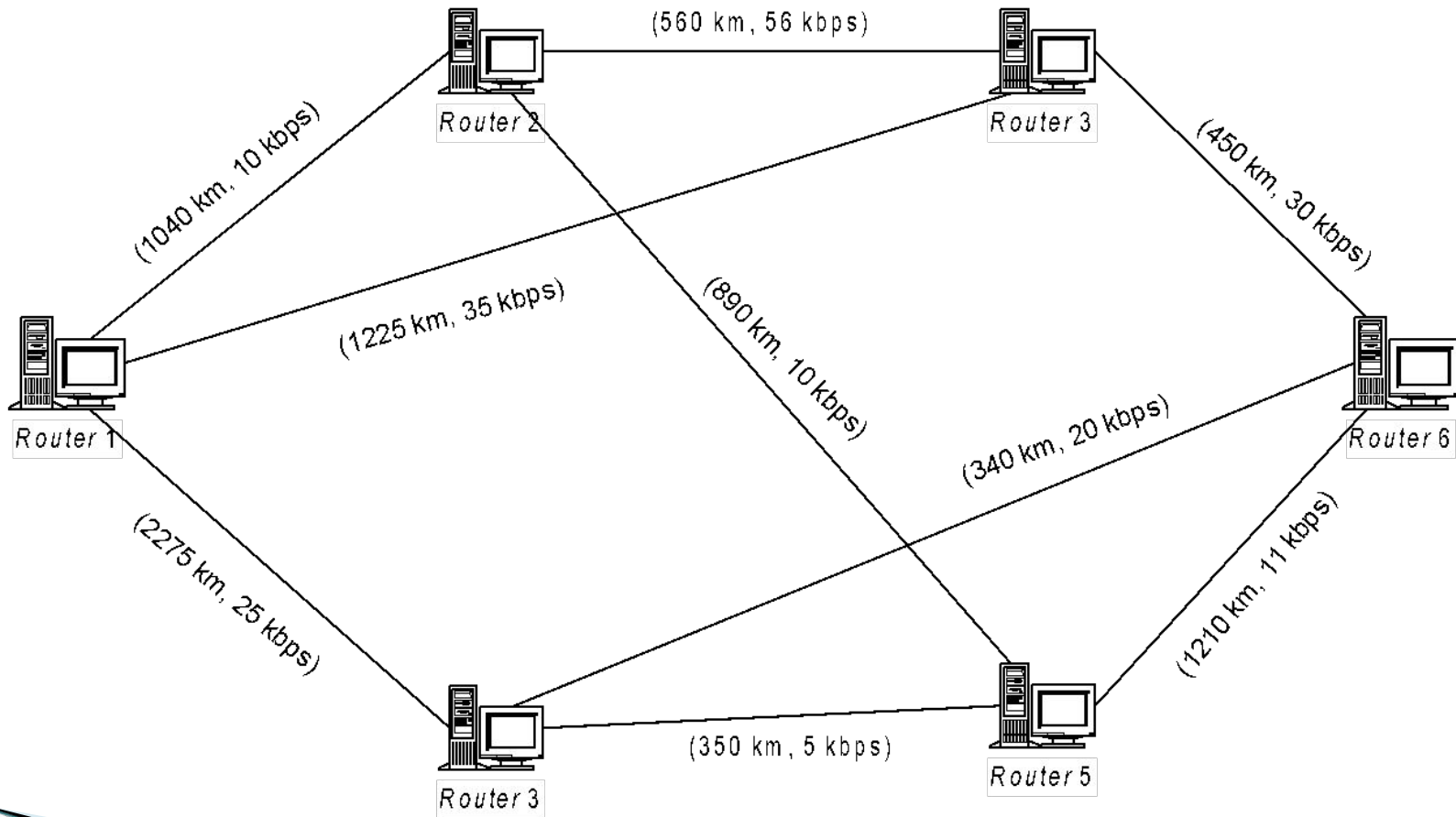
2(a)

12(a, c)



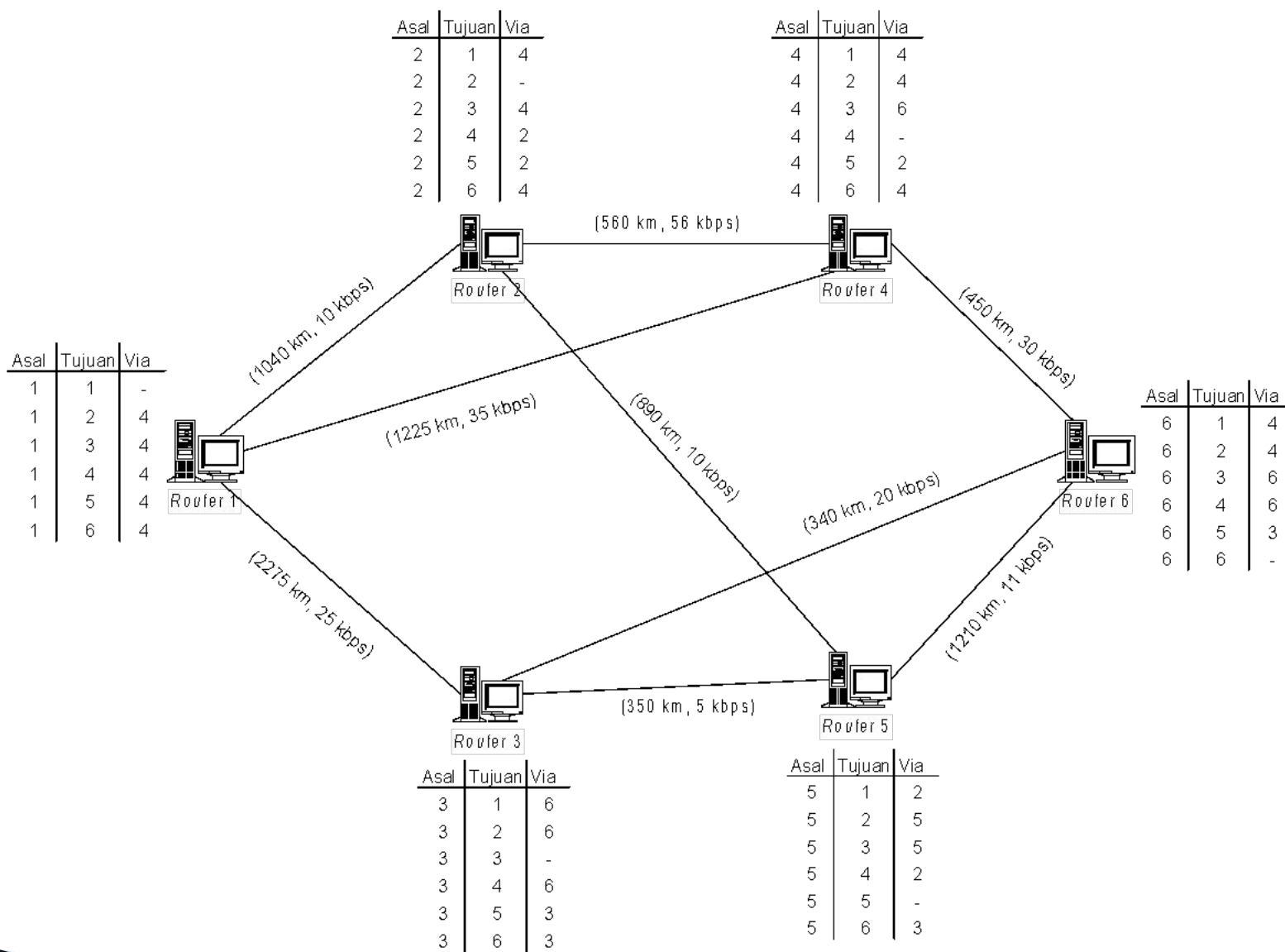
Aplikasi algoritma Dijkstra:

▣ *Routing* pada jaringan komputer



Lintasan terpendek (berdasarkan delay):

<i>Router Asal</i>	<i>Router Tujuan</i>	<i>Lintasan Terpendek</i>
1	1	-
	2	1, 4, 2
	3	1, 4, 6, 3
	4	1, 4
	5	1, 4, 2, 5
	6	1, 4, 6
2	1	2, 4, 1
	2	-
	3	2, 4, 6, 3
	4	2, 4
	5	2, 5
	6	2, 4, 6
3	1	3, 6, 4, 1
	2	3, 6, 4, 2
	3	-
	4	3, 6, 4
	5	3, 5
	6	3, 6
4	1	4, 1
	2	4, 2
	3	4, 6, 2
	4	4, 6, 3
	5	4, 2, 5
	6	4, 6



8. Pemampatan Data dengan Algoritma Huffman

- Kode Huffman mengkompresi data **sangat efektif**.
- Dapat melakukan kompresi **20% sd 90%** tergantung pada karakteristik data yang akan dikompresi.

Contoh Kompresi

- Perhatikan Tabel Berikut:

	a	b	c	d	e	f
Frekuensi (dalam ribuan)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

- Terdapat 100.000 karakter yang akan disimpan.
- Karakter a muncul 45.000 kali.
- Maka untuk **fixed-length codeword** membutuhkan $100.000 \times 3 \text{ bit} = 300.000 \text{ bit}$ untuk menyimpan datanya

Contoh Kompresi

- Dengan menggunakan **variable length codeword (Huffman code)** maka yang dibutuhkan adalah :

$$(45*1)+(13*3)+(12*3)+(16*3)+(9*4)+(5*4)*1000 = \mathbf{224.000 \text{ bit}}$$

- Kompresi / pemampatan yang diperoleh adalah $(300.000-224.000)/300.000 = \mathbf{25.3\%}$.

Bagaimana Membuat Huffman Code ?

□ **Prinsip** kode Huffman:

1. Karakter yang **paling sering muncul** di dalam data dengan **kode yang lebih pendek**;
2. Karakter yang **relatif jarang muncul** dikodekan dengan **kode yang lebih panjang**.

Pada contoh :

karakter a yg frekuensinya terbanyak memiliki panjang 1 bits
karakter f yg frekuensinya paling sedikit memiliki panjang 4 bits.

Algoritma **Greedy** Untuk Membuat Kode Huffman

1. Baca semua karakter di dalam data untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun data dinyatakan sebagai **pohon bersimpul tunggal**. Setiap **simpul** **di-assign** dengan frekuensi kemunculan karakter tersebut.
2. Terapkan strategi *greedy* sebagai berikut: pada setiap langkah **gabungkan** dua buah pohon yang mempunyai frekuensi **terkecil** pada sebuah akar. Akar mempunyai frekuensi yang merupakan jumlah dari frekuensi dua buah pohon penyusunnya.
3. Ulangi langkah 2 sampai hanya tersisa satu buah pohon Huffman.

Running Time **Greedy** Untuk Membuat Kode Huffman

- Q asumsinya menggunakan **binary min-heap**, untuk sekumpulan C dari n karakter, waktu yang dibutuhkan dengan menggunakan procedure BUILD-MIN-HEAP adalah **$O(n)$** .
- Untuk loop dari baris 3-8 yang dieksekusi $n-1$ kali, dengan asumsi setiap heap membutuhkan waktu **$O(\lg n)$** , maka loop akan memiliki running time $O(n \lg n)$. Sehingga **total running** time yang dibutuhkan adalah **$O(n \lg n)$** .
- Maka kompleksitas untuk pembentukan kode huffman dengan algoritma greedy adalah **$O(n \lg n)$**

Contoh Pembentukan Kode Huffman Dengan Algoritma

Greedy

□ Contoh Kasus:

	a	b	c	d	e	f
Frekuensi (dalam ribuan)	45	13	12	16	9	5

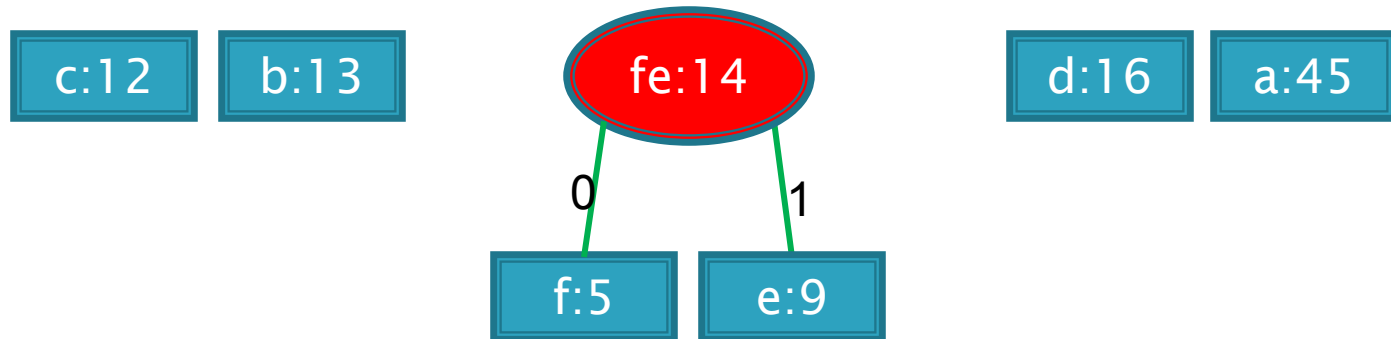
- Langkah 1 : Initial set dengan $n=6$ nodes (karakter a sd f)
- Langkah 2 : Bentuk Q dengan binary-min-heap (urutkan dari frekuensi terkecil)

a)

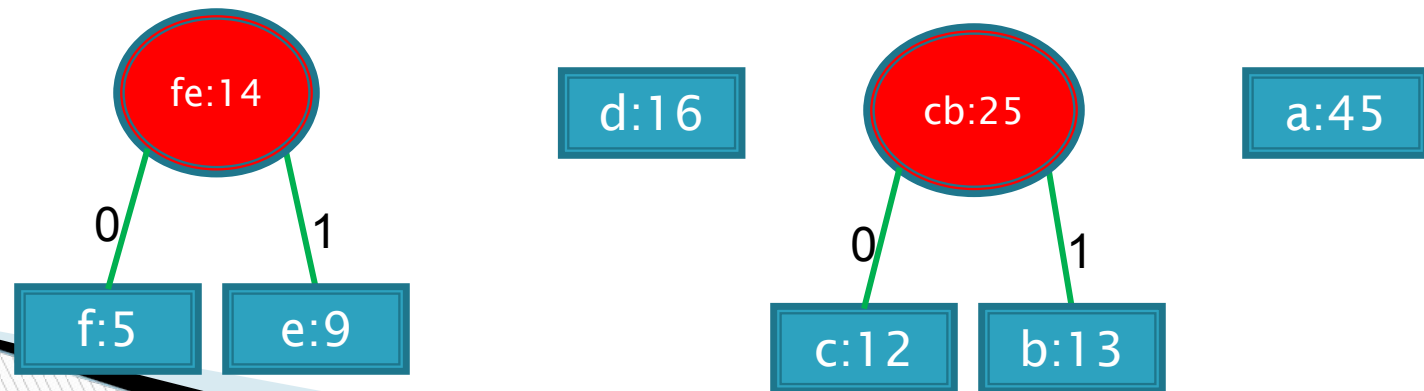
f:5	e:9	c:12	b:13	d:16	a:45
-----	-----	------	------	------	------

- Langkah 3-9 : setiap kondisi mewakili 1 loop (b sd e)

b)

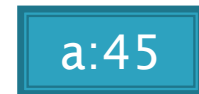
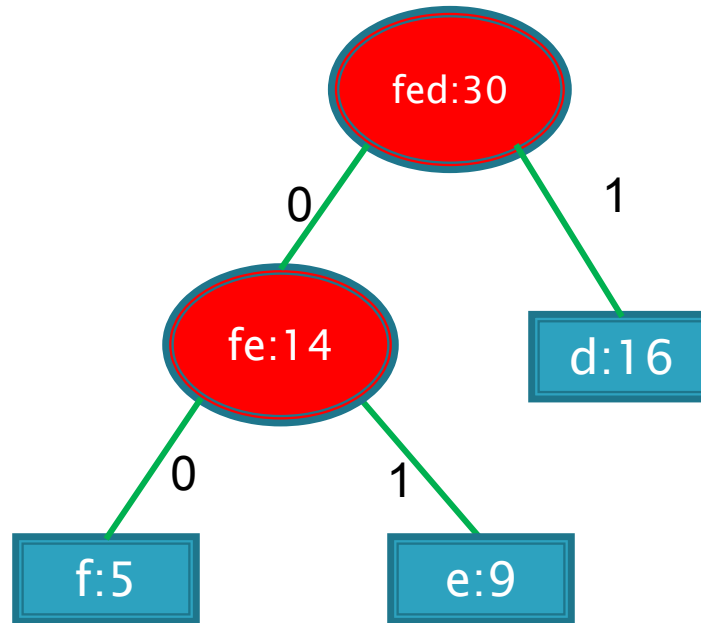
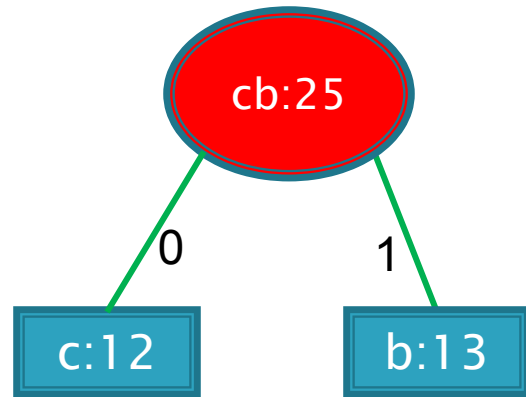


c)



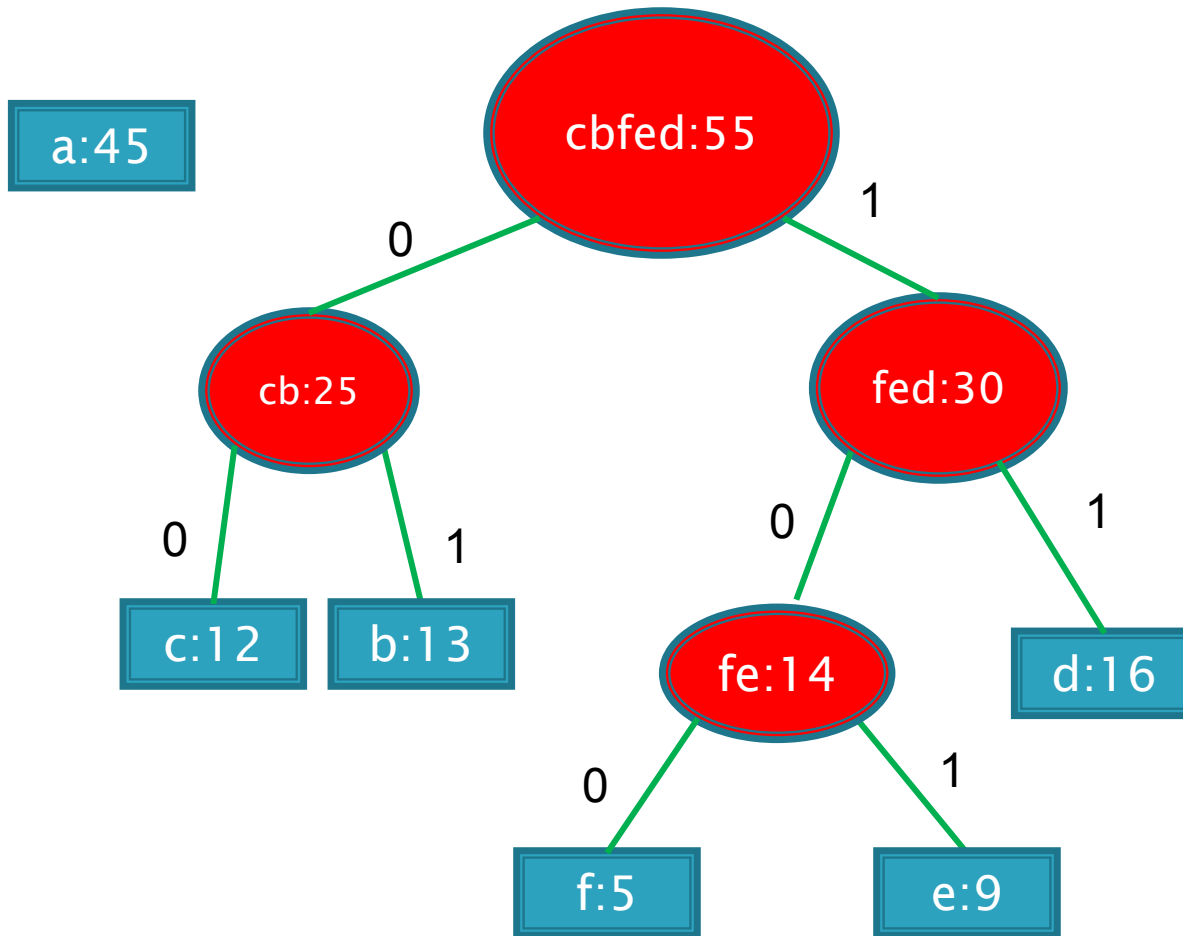
Kondisi d

d)



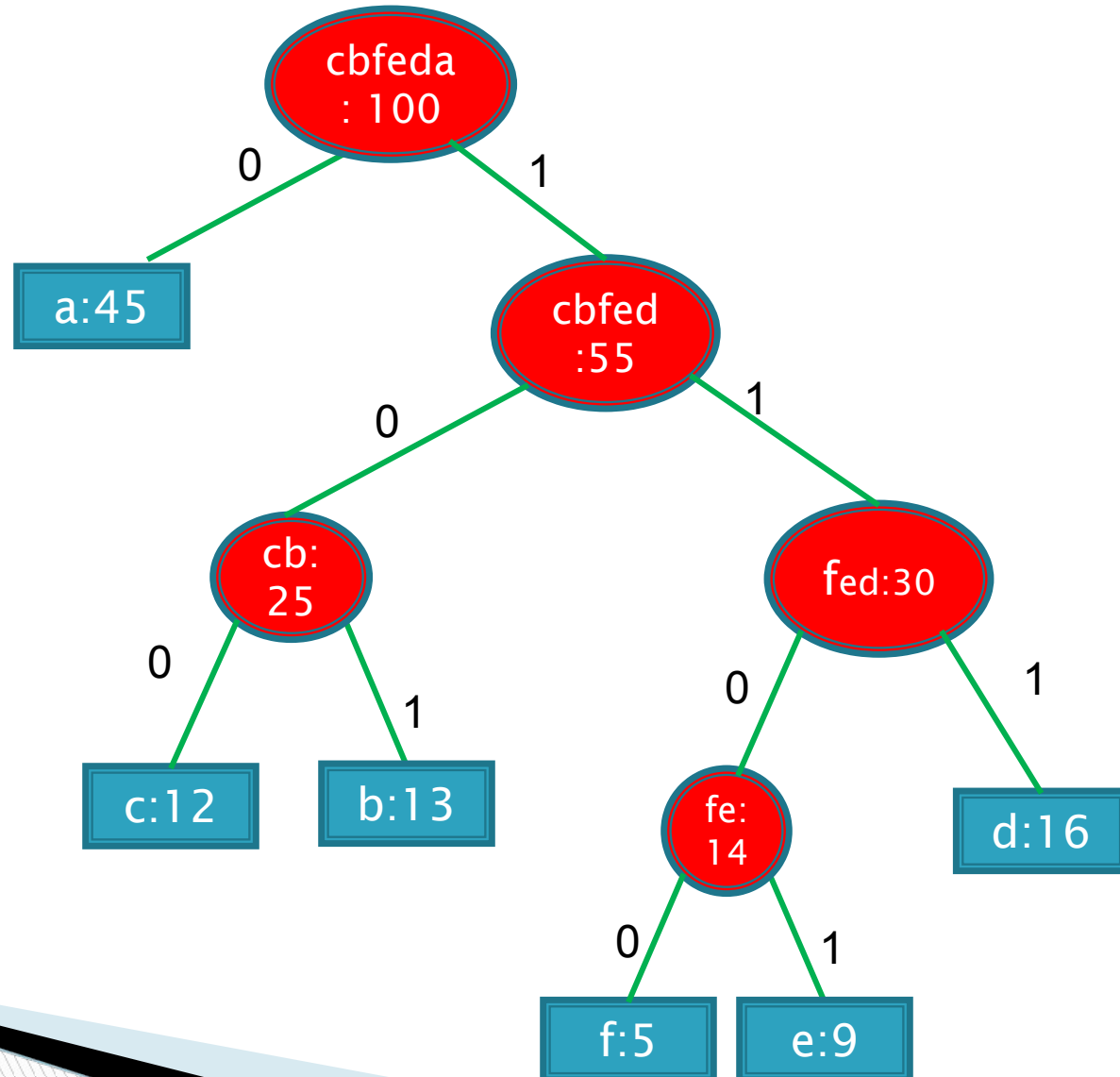
Kondisi e

e)



Kondisi Final f

f)



Pengkodean Huffman


- Pengkodean huffman dimulai dari root, sehingga :
 - karakter a : dikodekan dengan 0 (1 bit),
 - karakter b: dikodekan dengan 101 (3bit)
 - karakter c: dikodekan dengan 100 (3 bit)
 - karakter d: dikodekan dengan 111 (3 bit)
 - karakter e: dikodekan dengan 1101 (4 bit)
 - karakter f: dikodekan dengan 1100 (4 bit)
- Untuk karakter dengan frekuensi lebih banyak dikodekan dengan panjang bit yg lebih sedikit.

Kasus Kompresi Citra

Image / Citra

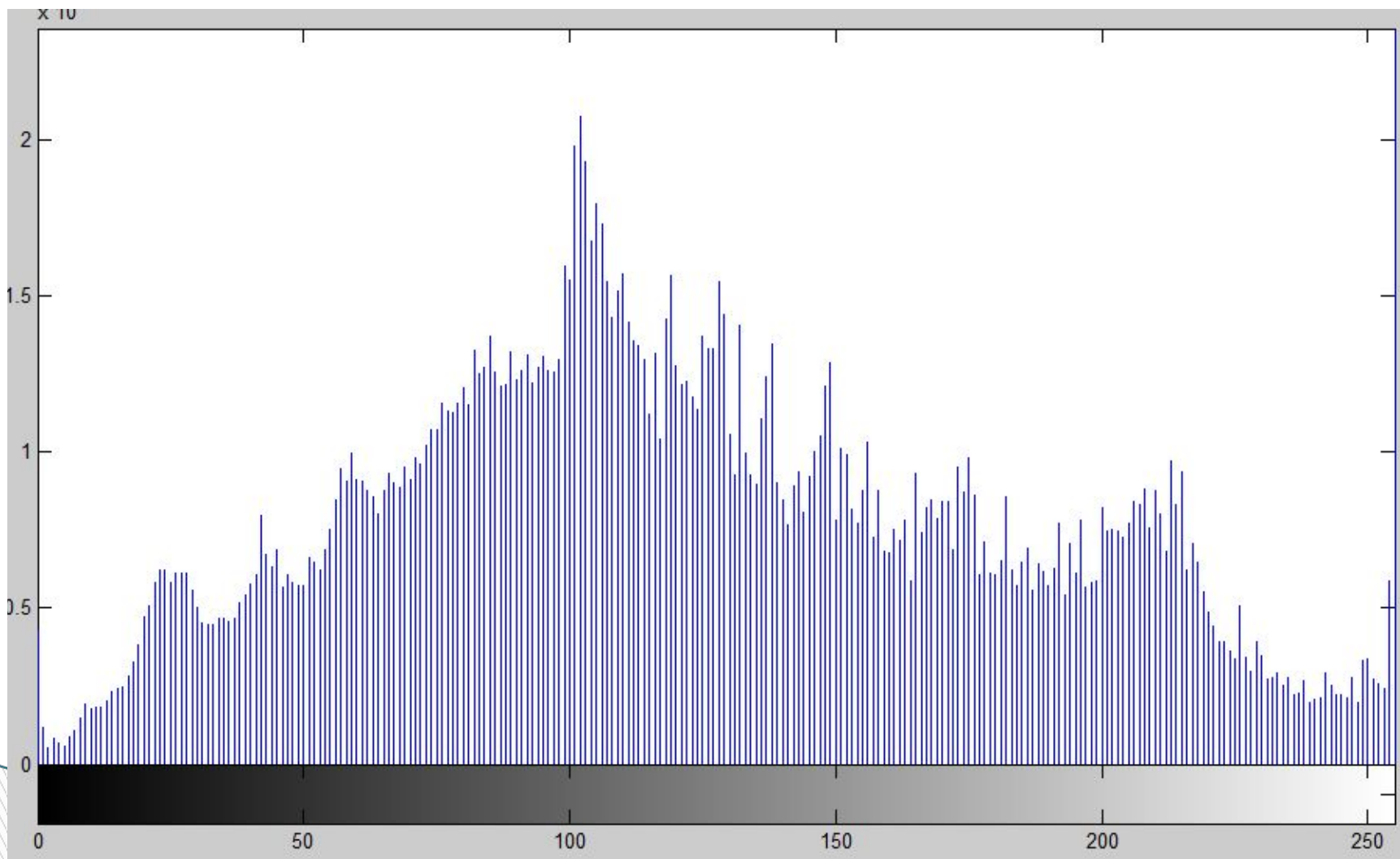


Piksel Citra

Workspace				
Name ▲	Value	Min	Max	
 x	<1080x1920 uint8>	<Too ...	<T...	

	292	293	294	295	296	297	298	299	300	301	302	303	
328	240	238	236	243	253	251	253	214	123	88	155	230	
329	243	247	239	237	244	253	227	110	59	124	204	248	
330	232	230	239	251	253	244	160	43	39	113	159	168	
331	230	231	249	253	247	179	105	54	87	107	86	80	
332	237	242	242	214	177	79	71	118	168	133	82	91	
333	239	232	205	163	131	130	150	223	253	219	194	204	
334	238	219	189	173	175	194	207	253	253	238	250	253	
335	240	227	212	219	236	212	219	253	249	240	253	253	
336	243	241	237	240	246	231	225	248	242	231	231	196	
337	242	251	242	244	245	247	253	229	245	215	152	129	
338	245	251	240	238	231	213	243	245	243	175	136	162	
339	240	237	219	208	187	179	160	171	192	173	205	229	
340	230	216	193	175	140	100	79	129	171	168	231	248	
341	230	214	199	184	140	52	57	130	172	153	194	187	
342	240	230	230	223	174	81	86	119	151	138	150	110	
343	244	237	243	240	185	143	159	141	141	118	105	76	
344	241	229	235	229	169	229	229	151	133	126	111	86	
345	226	240	242	228	165	251	231	165	146	152	124	102	
346	220	235	226	215	181	197	159	101	87	109	100	116	
347	222	252	245	196	141	96	69	54	38	43	30	92	
348	251	253	224	138	96	53	40	67	56	44	14	88	

Histogram



Kasus Kompresi Citra

- Terdapat Matriks Citra

100	100	100	100	100
100	200	200	200	100
250	100	200	100	250

- Hitung frekuensi

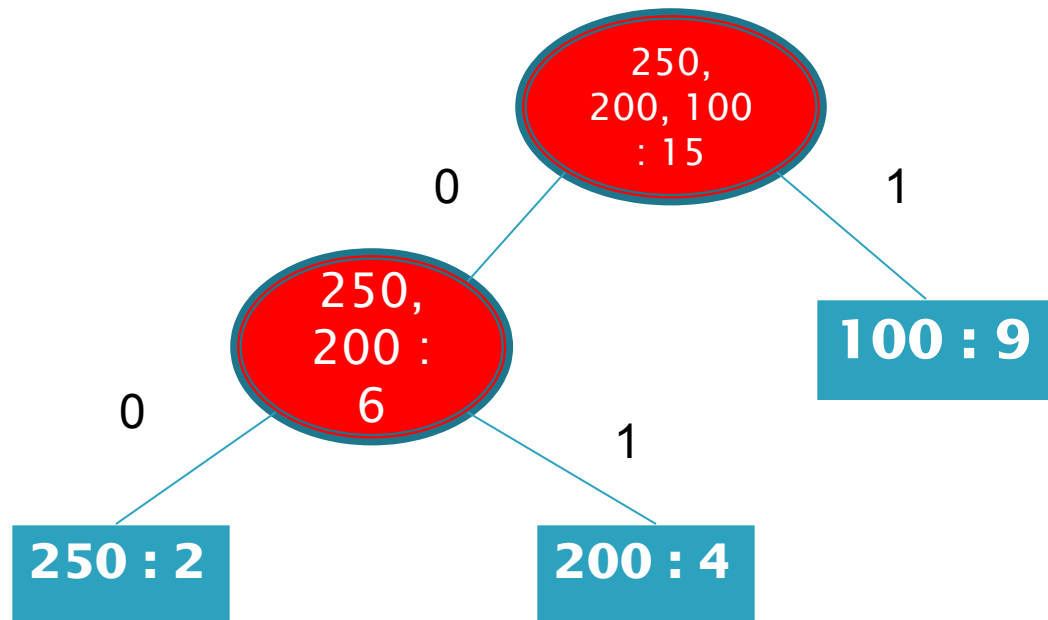
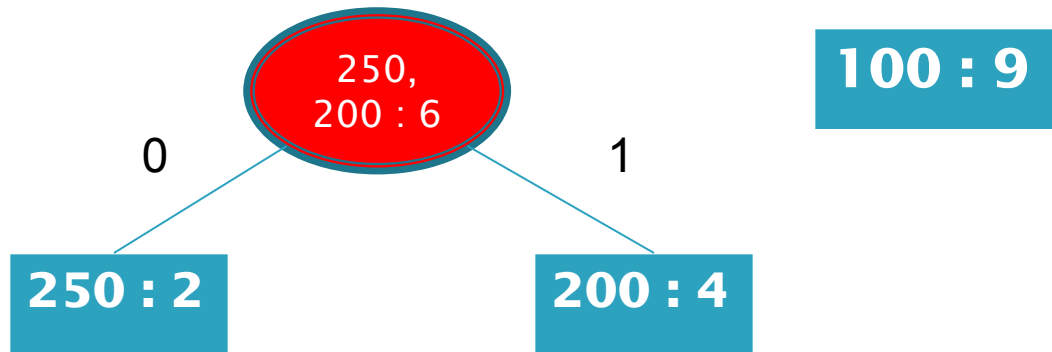
100	200	250
9	4	2

- Langkah 1 : Initial set dengan $n=3$ nodes (Piksel 100, 200, 250)
- Langkah 2 : Bentuk Q dengan binary-min-heap (urutkan dari frekuensi terkecil)

250 : 2

200 : 4

100 : 9



Rasio Kompresi

Piksel	100	200	250
Jumlah	9	4	2

- ❑ Ukuran Citra Sebelum Kompresi
 $15 \times 8 \text{ bit} = 120 \text{ bit}$
- ❑ Ukuran Citra Setelah Kompresi Huffman
 $(1 \times 9) + (2 \times 4) + (2 \times 2) = 21 \text{ bit}$
 $\text{Rasio} = (1 - (21/120)) \times 100\% = 82,5\%$

D. KESIMPULAN

1. Algoritma Greedy memiliki prinsip : “*take what you can get now!*”
2. Algoritma Greedy membentuk solusi yang terdiri dari beberapa langkah dan setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan.
3. Pada setiap langkah, dibuat pilihan **optimum lokal (*local optimum*)** , dengan harapan sisa langkahnya dapat mengarah ke solusi **optimum global (*global optimum*)**.
4. Tidak seluruh penyelesaian Greedy dapat menghasilkan hasil optimum, seperti pada kasus penukaran uang dan juga 1/0 Knapsack.

Latihan-1

Terdapat dokumen dengan frekuensi kemunculan karakternya (dalam ribuan) sebagai berikut :

a	c	g	k	m	n	o	p
15	7	3	8	5	10	12	1

Tentukan :

1. Pohon Huffman
2. Kode biner masing-masing karakter
3. Rasio Kompresi.

Latihan-2

Terdapat data piksel sebuah gambar (8x8) sbb:

100	0	100	0	100	0	0	0
130	0	0	0	100	0	100	0
0	130	100	45	45	255	255	150
200	100	200	45	255	255	255	255
100	0	0	130	100	255	100	255
200	200	100	130	100	255	255	255
200	0	0	130	100	45	255	100
0	200	0	255	255	255	128	150

Tentukan :

1. Pohon Huffman
2. Kode biner masing-masing piksel
3. Rasio Kompresi.