

@Design and Analysis Algorithm

Pertemuan 06

Drs. Achmad Ridok M.Kom
Imam Cholissodin, S.Si., M.Kom
M. Ali Fauzi S.Kom., M.Kom
Ratih Kartika Dewi, ST, M.Kom



Contents

1

Greedy Algorithm



Pendahuluan

- Algoritma greedy merupakan metode yang paling populer untuk memecahkan persoalan optimasi.
- Persoalan optimasi (optimization problems):
 - persoalan mencari solusi optimum.
- Hanya ada dua macam persoalan optimasi:
 1. Maksimasi (maximization)
 2. Minimasi (minimization)



- Contoh persoalan optimasi:
(Masalah Penukaran Uang): Diberikan uang senilai A. Tukar A dengan koin-koin uang yang ada. Berapa jumlah minimum koin yang diperlukan untuk penukaran tersebut?
 - Persoalan minimasi
- Contoh 1: tersedia banyak koin 1, 5, 10, 25
 - Uang senilai A = 32 dapat ditukar dengan banyak cara berikut:
$$32 = 1 + 1 + \dots + 1 \quad (32 \text{ koin})$$
$$32 = 5 + 5 + 5 + 5 + 10 + 1 + 1 \quad (7 \text{ koin})$$
$$32 = 10 + 10 + 10 + 1 + 1 \quad (5 \text{ koin})$$
$$\dots \text{ dst}$$
 - Minimum: $32 = 25 + 5 + 1 + 1 \quad (4 \text{ koin})$



Algoritma Greedy

- Algoritma greedy adalah algoritma yang memecahkan masalah langkah per langkah;
- Pada setiap langkah:
 1. mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip “take what you can get now!”)
 2. berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.



- 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 ($\text{Total} = 25$)
 - Langkah 2: pilih 1 buah koin 5 ($\text{Total} = 25 + 5 = 30$)
 - Langkah 3: pilih 2 buah koin 1 ($\text{Total} = 25+5+1+1= 32$)
- Solusi: Jumlah koin minimum = 4 (solusi optimal!)



Elemen Algoritma Greedy

- Elemen-elemen algoritma greedy:
 1. Himpunan kandidat, C.
 2. Himpunan solusi, S
 3. Fungsi seleksi (selection function)
 4. Fungsi kelayakan (feasible)
 5. Fungsi obyektif
- Dengan kata lain:
 - 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.



Elemen Algoritma Greedy

- Pada masalah penukaran uang:
 - Himpunan kandidat: himpunan koin yang merepresentasikan nilai 1, 5, 10, 25, paling sedikit mengandung satu koin untuk setiap nilai.
 - Himpunan solusi: total nilai koin yang dipilih tepat sama jumlahnya dengan nilai uang yang ditukarkan.
 - Fungsi seleksi: pilihlah koin yang bernilai tertinggi dari himpunan kandidat yang tersisa.
 - Fungsi layak: memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar.
 - 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 pengujian, solusi yang terbentuk adalah optimum lokal.
- Pada akhir perulangan 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) **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)



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



Contoh-contoh Algoritma Greedy

1. Masalah penukaran uang

Nilai uang yang ditukar: A

Himpunan koin (multiset): $\{d_1, d_2, \dots, d_n\}$.

Himpunan solusi: $X = \{x_1, x_2, \dots, x_n\}$,

$x_i = 1$ jika di dipilih, $x_i = 0$ jika di tidak dipilih.

Obyektif persoalan adalah

$$\text{Minimisasi } F = \sum_{i=1}^n x_i \quad (\text{fungsi obyektif})$$

$$\text{dengan kendala } \sum_{i=1}^n d_i x_i = A$$



Penyelesaian dengan exhaustive search

- Terdapat 2^n kemungkinan solusi (nilai-nilai $X = \{x_1, x_2, \dots, x_n\}$)
- Untuk mengevaluasi fungsi obyektif = $O(n)$
- Kompleksitas algoritma exhaustive search seluruhnya = $O(n \cdot 2^n)$.



■ Penyelesaian dengan algoritma greedy

- Strategi greedy: Pada setiap langkah, pilih koin dengan nilai terbesar dari himpunan koin yang tersisa

```
function CoinExchange(input C : himpunan_koin, A : integer) → himpunan_koin
{ mengembalikan koin-koin yang total nilainya = A, tetapi jumlah koinnya minimum }

Deklarasi
    S : himpunan_koin
    x : koin

Algoritma
    S ← {}
    while ( $\sum$ (nilai semua koin di dalam S) ≠ A) and (C ≠ {}) do
        x ← koin yang mempunyai nilai terbesar
        C ← C - {x}
        if ( $\sum$ (nilai semua koin di dalam S) + nilai koin x ≤ A then
            S ← S ∪ {x}
        endif
    endwhile

    if ( $\sum$ (nilai semua koin di dalam S) = A then
        return S
    else
        write('tidak ada solusi')
    endif
```



- Agar pemilihan koin berikutnya optimal, maka perlu mengurutkan himpunan koin dalam urutan yang menurun (nonincreasing order).
- Jika himpunan koin sudah terurut menurun, maka kompleksitas algoritma greedy = $O(n)$.
- Sayangnya, algoritma greedy untuk masalah penukaran uang ini tidak selalu menghasilkan solusi optimal (lihat contoh sebelumnya).



Contoh-contoh Algoritma Greedy

2. 0/1 Knapsack

$$\text{Maksimasi } F = \sum_{i=1}^n p_i x_i$$

dengan kendala (*constraint*)

$$\sum_{i=1}^n w_i x_i \leq K$$

yang dalam hal ini, $x_i = 0$ atau 1 , $i = 1, 2, \dots, n$

Penyelesaian dengan exhaustive search

- Sudah dijelaskan pada pembahasan exhaustive search.
- Kompleksitas algoritma exhaustive search untuk persoalan ini = $O(n \cdot 2^n)$.



Penyelesaian dengan algoritma greedy

- Masukkan objek satu per satu ke dalam knapsack. Sekali objek dimasukkan ke dalam knapsack, objek tersebut tidak bisa dikeluarkan lagi.
- Terdapat beberapa strategi greedy yang heuristik yang dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam knapsack:



1. Greedy by profit.

- Pada setiap langkah, pilih objek yang mempunyai keuntungan terbesar.
- Mencoba memaksimumkan keuntungan dengan memilih objek yang paling menguntungkan terlebih dahulu.

2. Greedy by weight.

- Pada setiap langkah, pilih objek yang mempunyai berat teringan.
- Mencoba memaksimumkan keuntungan dengan dengan memasukkan sebanyak mungkin objek ke dalam knapsack.



3. Greedy by density.

- Pada setiap langkah, knapsack diisi dengan objek yang mempunyai π/w_i terbesar.
- Mencoba memaksimumkan keuntungan dengan memilih objek yang mempunyai keuntungan per unit berat terbesar.
- Pemilihan objek berdasarkan salah satu dari ketiga strategi di atas tidak menjamin akan memberikan solusi optimal.



Contoh 1

$w_1 = 6; p_1 = 12; w_2 = 5; p_2 = 15;$

$w_3 = 10; p_3 = 50; w_4 = 5; p_4 = 10$

Kapasitas knapsack $K = 16$

Properti objek				<i>Greedy by</i>			Solusi Optimal
i	w_i	p_i	p_i/w_i	<i>profit</i>	<i>weight</i>	<i>density</i>	
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total bobot				15	16	15	15
Total keuntungan				65	37	65	65

Solusi optimal: $X = (0, 1, 1, 0)$

Greedy by profit dan greedy by density memberikan solusi optimal!



Contoh 2

$w_1 = 100; p_1 = 40; w_2 = 50; p_2 = 35; w_3 = 45; p_3 = 18;$
 $w_4 = 20; p_4 = 4; w_5 = 10; p_5 = 10; w_6 = 5; p_6 = 2$

Kapasitas knapsack $K = 100$

i	Properti objek		<i>Greedy by</i>			Solusi Optimal	
	w_i	p_i	p_i/w_i	<i>profit</i>	<i>weight</i>	<i>density</i>	
1	100	40	0,4	1	0	0	0
2	50	35	0,7	0	0	1	1
3	45	18	0,4	0	1	0	1
4	20	4	0,2	0	1	1	0
5	10	10	1,0	0	1	1	0
6	5	2	0,4	0	1	1	1
Total bobot			100	80	85	100	
Total keuntungan			40	34	51	55	

Ketiga strategi gagal memberikan solusi optimal!



Kesimpulan:

Algoritma greedy tidak selalu berhasil menemukan solusi optimal untuk masalah **0/1 Knapsack.**



Fractional Knapsack

Maksimasi $F = \sum_{i=1}^n p_i x_i$

dengan kendala (*constraint*)

$$\sum_{i=1}^n w_i x_i \leq K$$

yang dalam hal ini, $0 \leq x_i \leq 1$, $i = 1, 2, \dots, n$



Penyelesaian dengan *exhaustive search*

- Oleh karena $0 \leq x_i \leq 1$, maka terdapat tidak berhingga nilai-nilai x_i .
- Persoalan *Fractional Knapsack* menjadi malar (*continuous*) sehingga tidak mungkin dipecahkan dengan algoritma *exhaustive search*.



Penyelesaian dengan algoritma *greedy*

- Ketiga strategi *greedy* yang telah disebutkan di atas dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam *knapsack*.



Contoh 3

$$w_1 = 18; \quad p_1 = 25; \quad w_2 = 15; \quad p_2 = 24$$

$$w_3 = 10; \quad p_3 = 15 \quad \text{Kapasitas knapsack } K = 20$$

Properti objek				<i>Greedy by</i>		
<i>i</i>	w_i	p_i	p_i/w_i	<i>profit</i>	<i>weight</i>	<i>density</i>
1	18	25	1,4	1	0	0
2	15	24	1,6	2/15	2/3	1
3	10	15	1,5	0	1	1/2
Total bobot				20	20	20
Total keuntungan				28,2	31,0	31,5

- Solusi optimal: $X = (0, 1, 1/2)$
- yang memberikan keuntungan maksimum = 31,5



- Strategi pemilihan objek berdasarkan densitas p_i/w_i terbesar akan selalu memberikan solusi optimal.
- Agar proses pemilihan objek berikutnya optimal, maka kita urutkan objek berdasarkan p_i/w_i yang menurun, sehingga objek berikutnya yang dipilih adalah objek sesuai dalam urutan itu.

Teorema 3.2. Jika $p_1/w_1 \geq p_2/w_2 \geq \dots \geq p_n/w_n$ maka algoritma *greedy* dengan strategi pemilihan objek berdasarkan p_i/w_i terbesar menghasilkan solusi yang optimum.



Algoritma persoalan fractional knapsack:

1. Hitung harga p_i/w_i , $i = 1, 2, \dots, n$
2. Urutkan seluruh objek berdasarkan nilai p_i/w_i dari besar ke kecil
3. Panggil FractionalKnapsack



▪ Algoritma fractional knapsack

```
function FractionalKnapsack(input C : himpunan_objek, K : real) → himpunan_solusi
{ Menghasilkan solusi persoalan fractional knapsack dengan algoritma greedy yang menggunakan strategi pemilihan objek berdasarkan density ( $p_i/w_i$ ). Solusi dinyatakan sebagai vektor  $X = x[1], x[2], \dots, x[n]$ .
Asumsi: Seluruh objek sudah terurut berdasarkan nilai  $p_i/w_i$  yang menurun
}

Deklarasi
    i, TotalBobot : integer
    MasihMuatUtuh : boolean
    x : himpunan_solusi

Algoritma:
    for i  $\leftarrow$  1 to n do
        x[i]  $\leftarrow$  0 { inisialisasi setiap fraksi objek i dengan 0 }
    endfor

    i  $\leftarrow$  0
    TotalBobot  $\leftarrow$  0
    MasihMuatUtuh  $\leftarrow$  true
    while (i  $\leq$  n) and (MasihMuatUtuh) do
        { tinjau objek ke-i }
        i  $\leftarrow$  i + 1
        if TotalBobot + C.w[i]  $\leq$  K then
            { masukkan objek i ke dalam knapsack }
            x[i]  $\leftarrow$  1
            TotalBobot  $\leftarrow$  TotalBobot + C.w[i]
        else
            MasihMuatUtuh  $\leftarrow$  false
            x[i]  $\leftarrow$  (K - TotalBobot)/C.w[i]
        endif
    endwhile
    { i  $>$  n or not MasihMuatUtuh }

    return x
```

Kompleksitas waktu algoritma = $O(n)$.



Optimal Storage on Tapes

- Terdapat n program yang akan disimpan pada computer tape dengan panjang L.
- Setiap program i memiliki panjang l_i , $1 \leq i \leq n$
- Semua program di-retrieve secara merata, rata-rata waktu retrieve/ mean retrieval time (MRT) yang diharapkan adalah

$$(1/n) \sum_{1 \leq j \leq n} t_j$$



Contoh

- Misalkan $n = 3$ dan $(l_1, l_2, l_3) = (5, 10, 3)$

Ordering I	$D(I) = \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq j} l_{i_k}$
1,2,3	$5 + 5 + 10 + 5 + 10 + 3 = 38$
1,3,2	$5 + 5 + 3 + 5 + 3 + 10 = 31$
2,1,3	$10 + 10 + 5 + 10 + 5 + 3 = 43$
2,3,1	$10 + 10 + 3 + 10 + 3 + 5 = 41$
3,1,2	$3 + 3 + 5 + 3 + 5 + 10 = 29$
3,2,1	$3 + 3 + 10 + 3 + 10 + 5 = 34$



The Greedy Solution

Buatlah tape = kosong

for i := 1 to n do

 ambil file terpendek berikutnya

 letakkan file tersebut pada tape

- Algoritma greedy mengambil pilihan terbaik jangka pendek tanpa memeriksa untuk melihat apakah kondisi tersebut adalah keputusan jangka panjang terbaik atau tidak.



Optimal Storage on Tapes (cont.)

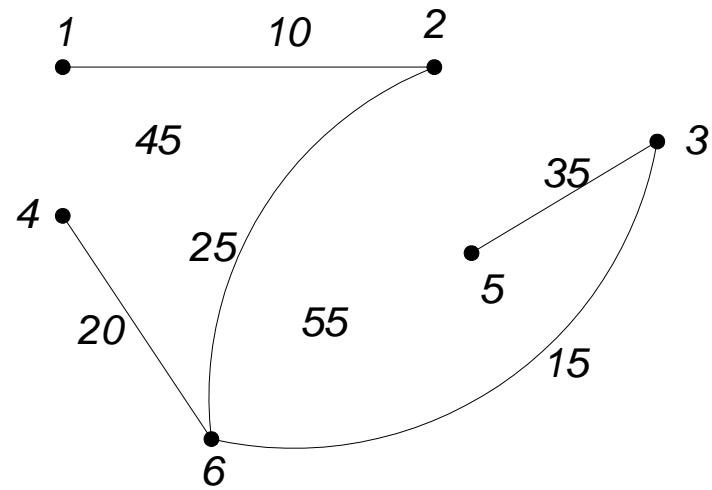
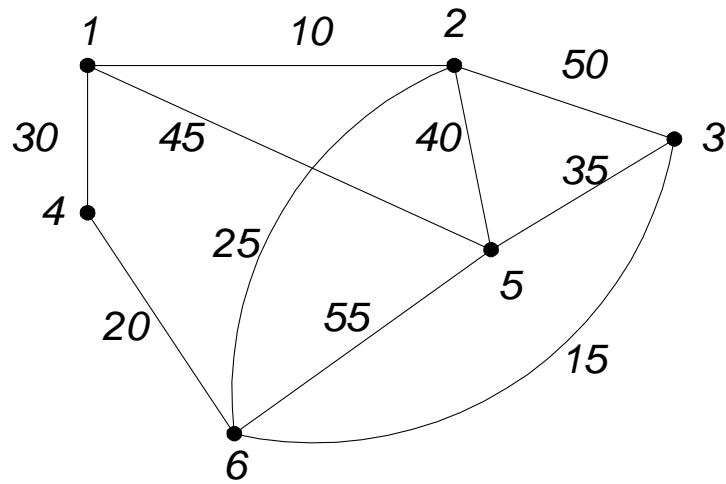
- **Theorem 4.1.** Jika $l_1 \leq l_2 \leq \dots \leq l_n$ maka ordering $i_j = j$, $1 \leq j \leq n$ meminimalkan nilai

$$\sum_{k=1}^n \sum_{j=1}^k l_{i_j}$$

- Berlaku pada semua kemungkinan permutasi dari i_j
- Lihat pembuktiannya pada buku hal.154-155



Pohon Merentang Minimum





Pohon Merentang Minimum

a) Algoritma Prim

- Strategi *greedy* yang digunakan:
Pada setiap langkah, pilih sisi e dari graf $G(V, E)$ yang mempunyai bobot terkecil dan bersisian dengan simpul-simpul di T tetapi e tidak membentuk sirkuit di T .
- Kompleksitas algoritma: $O(n^2)$



Pohon Merentang Minimum

b) Algoritma Kruskal

- Strategi *greedy* yang digunakan:
Pada setiap langkah, pilih sisi e dari graf G yang mempunyai bobot minimum tetapi e tidak membentuk sirkuit di T .
- Kompleksitas algoritma: $O(|E| \log |E|)$



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*).



Lintasan Terpendek (Shortest Path)

Persoalan:

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

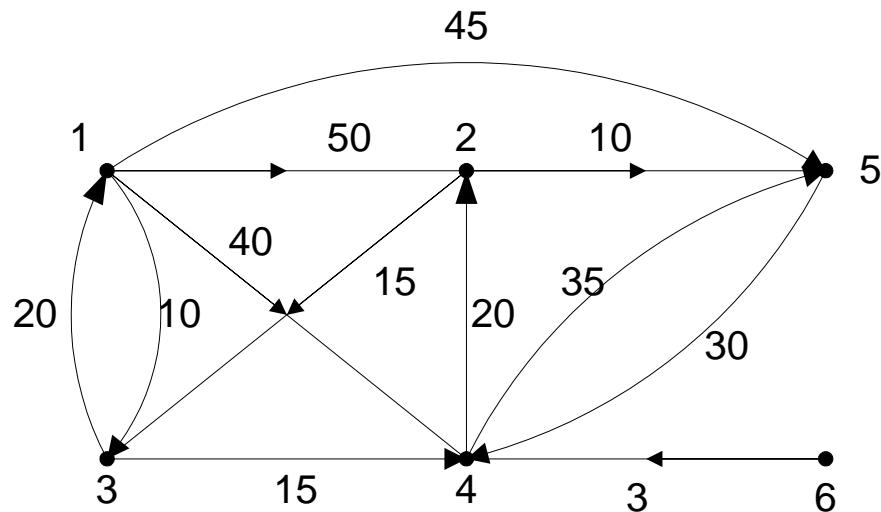
Asumsi yang kita buat adalah bahwa semua sisi berbobot positif.

Strategi *greedy*:

Lintasan dibentuk satu per satu. Lintasan berikutnya yang dibentuk ialah lintasan yang meminimumkan jumlah jaraknya.



Contoh



Simpul asal	Simpul tujuan	Lintasan terpendek	Jarak
1	3	$1 \rightarrow 3$	10
1	4	$1 \rightarrow 3 \rightarrow 4$	25
1	2	$1 \rightarrow 3 \rightarrow 4 \rightarrow 2$	45
1	5	$1 \rightarrow 5$	45
1	6	tidak ada	-



Algoritma Dijkstra

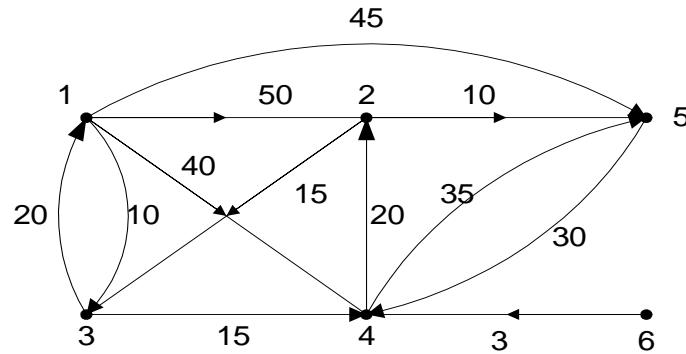
Strategi *greedy*:

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.



Contoh

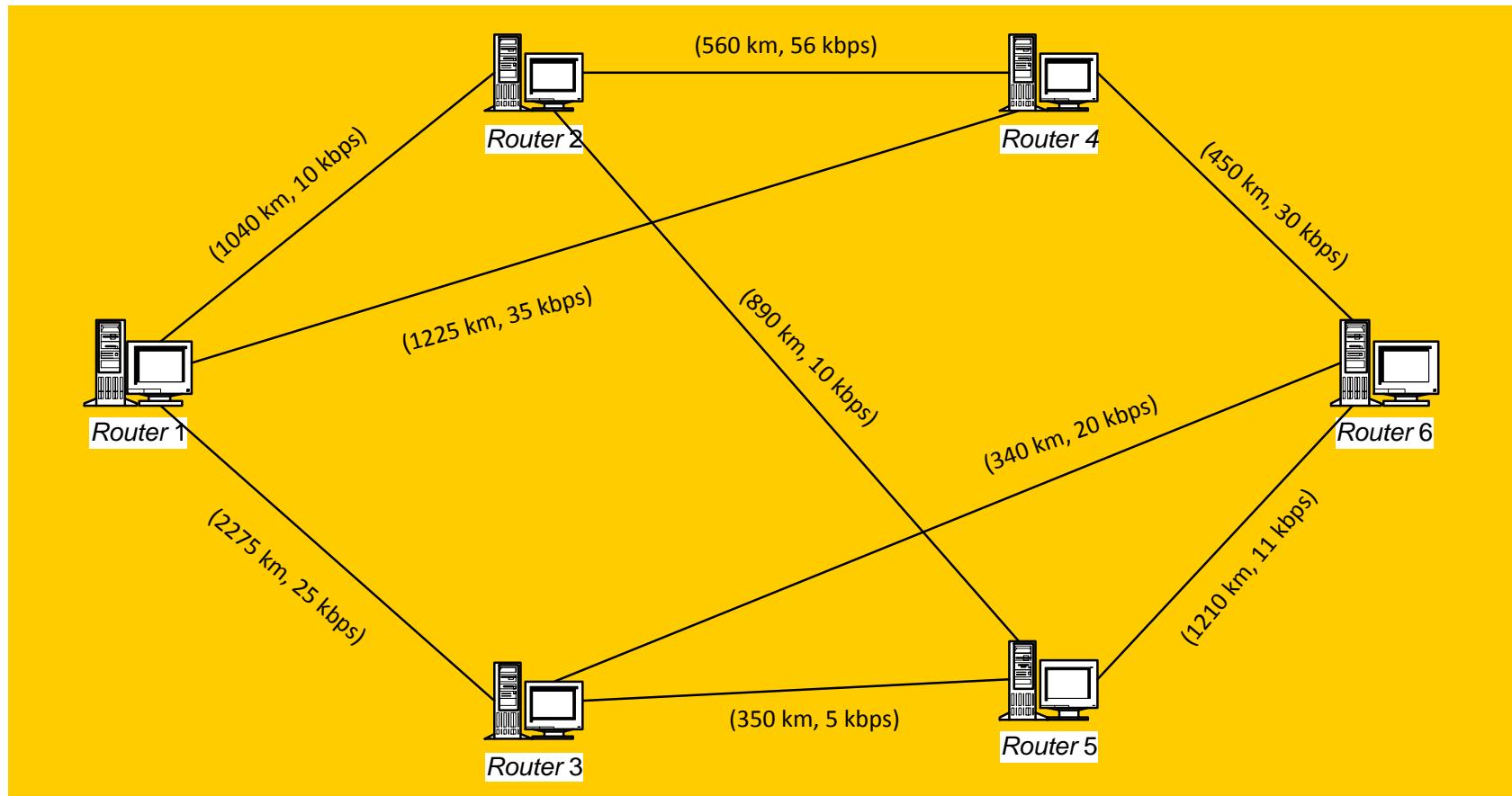


Lelaran	Simpul yang dipilih	Lintasan	S						D					
			1	2	3	4	5	6	1	2	3	4	5	6
Inisial	-	-	0	0	0	0	0	0	0	50	10	40	45	∞
										(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
1	1	1	1	0	0	0	0	0	∞	50	10	40	45	∞
										(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
2	3	1, 3	1	0	1	0	0	0	∞	50	10	25	45	∞
										(1,2)	(1,3)	(1,3,4)	(1,5)	(1,6)
3	4	1, 3, 4	1	0	1	1	0	0	∞	45	10	25	45	∞
										(1,3,4,2)	(1,3)	(1,3,4)	(1,5)	(1,6)
4	2	1, 3, 4, 2	1	1	1	1	0	0	∞	45	10	25	45	∞
										(1,3,4,2)	(1,3)	(1,3,4)	(1,5)	(1,6)
5	5	1, 5	1	1	1	1	1	0	∞	45	10	25	45	∞



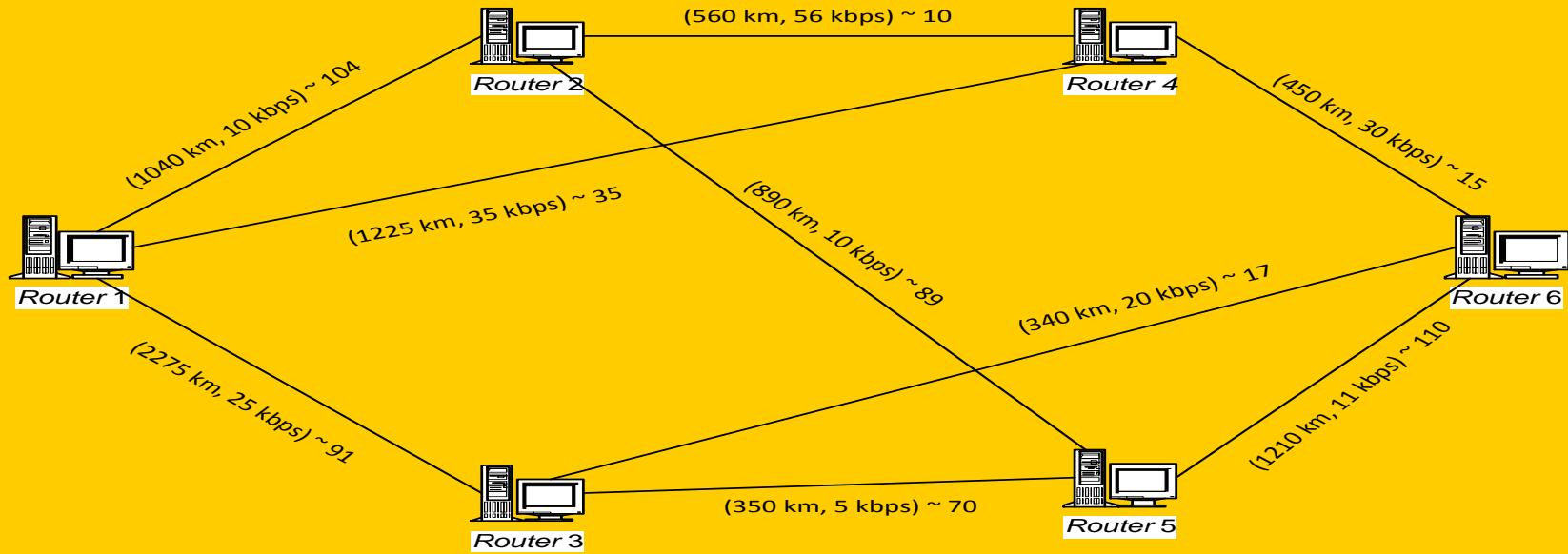
Algoritma Dijkstra

Aplikasi algoritma Dijkstra:
→ *Routing* pada jaringan komputer





Lintasan terpendek (berdasarkan delay):



Router Asal	Router Tujuan	Lintasan Terpendek
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

Router Asal	Router Tujuan	Lintasan Terpendek
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

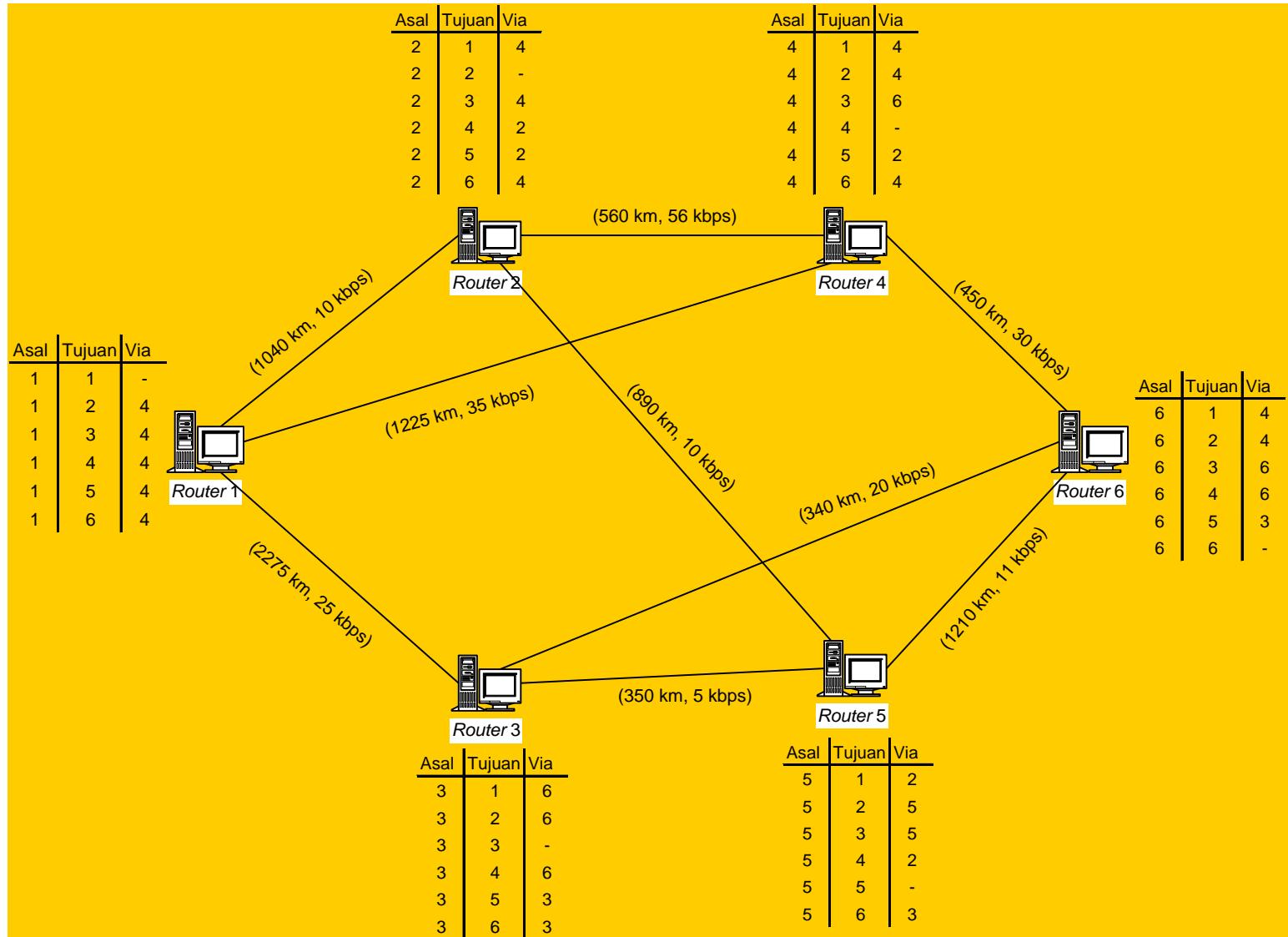


Lintasan terpendek (berdasarkan delay):

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



Lintasan terpendek (berdasarkan delay):





Algoritma Huffman

Pemampatan Data dengan Algoritma Huffman

Prinsip kode Huffman:

- karakter yang paling sering muncul di dalam data dengan kode yang lebih pendek;
- sedangkan karakter yang relatif jarang muncul dikodekan dengan kode yang lebih panjang.



Algoritma Huffman

Fixed-length code

Karakter	a	b	c	d	e	f
<hr/>						
Frekuensi	45%	13%	12%	16%	9%	5%
Kode	000	001	010	011	100	111

‘bad’ dikodekan sebagai ‘001000011’

Pengkodean 100.000 karakter membutuhkan 300.000 bit.



Algoritma Huffman

Variable-length code (Huffman code)

Karakter	a	b	c	d	e	f
<hr/>						
Frekuensi	45%	13%	12%	16%	9%	5%
Kode	0	101	100	111	1101	1100

‘bad’ dikodekan sebagai ‘1010111 ’

Pengkodean 100.000 karakter membutuhkan
 $(0,45 \times 1 + 0,13 \times 3 + 0,12 \times 3 + 0,16 \times 3 + 0,09 \times 4 + 0,05 \times 4) \times 100.000 = 224.000 \text{ bit}$

Nisbah/ratio pemampatan:

$$(300.000 - 224.000)/300.000 \times 100\% = 25,3\%$$



Algoritma Huffman

Algoritma Greedy untuk Membentuk Kode Huffman:

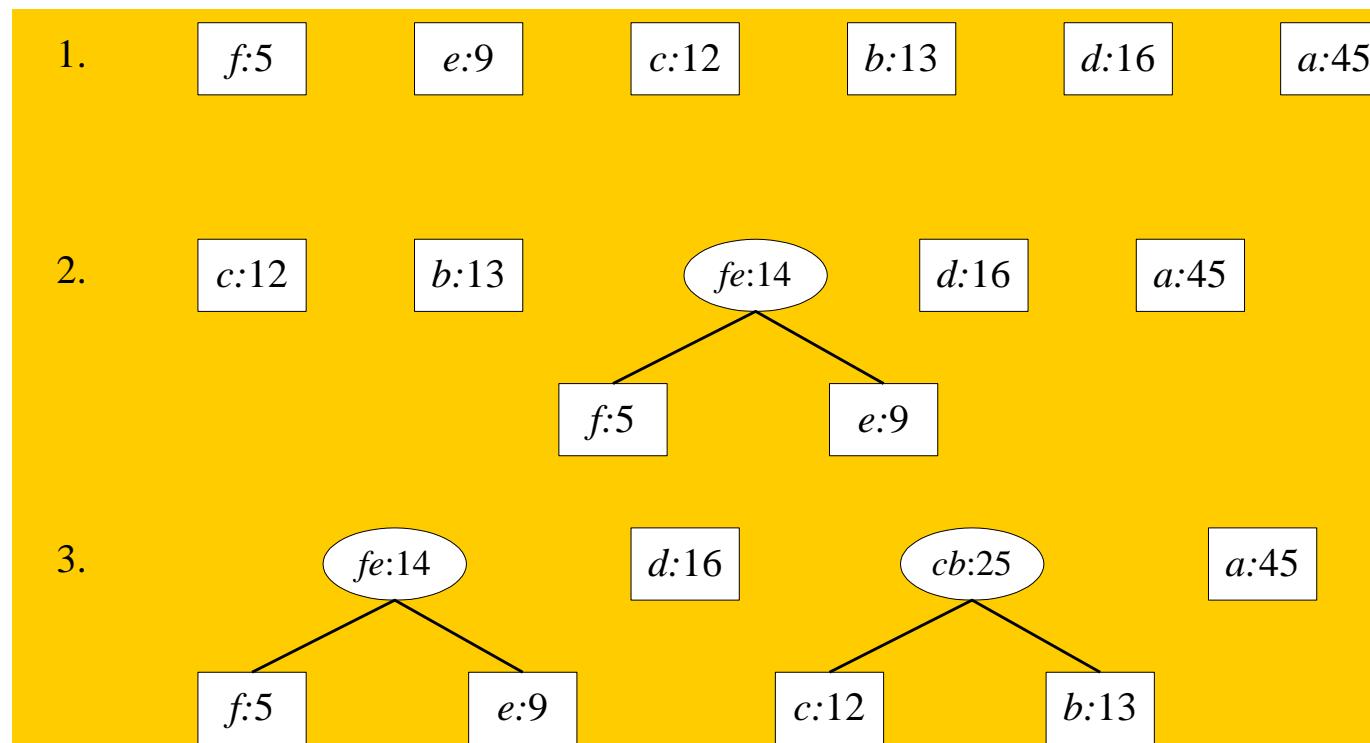
1. Baca semua karakter di dalam data untuk menghitung frekuensi kemunculan setiap karakter. Setiap karakter penyusun data dinyatakan sebagai pohon bersimpul tunggal. Setiap simpul di-assign dengan frekuensi kemunculan karakter tersebut.
2. Terapkan strategi *greedy* sebagai berikut: gabungkan dua buah 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.

Kompleksitas algoritma Huffman: $O(n \log n)$ untuk n karakter.



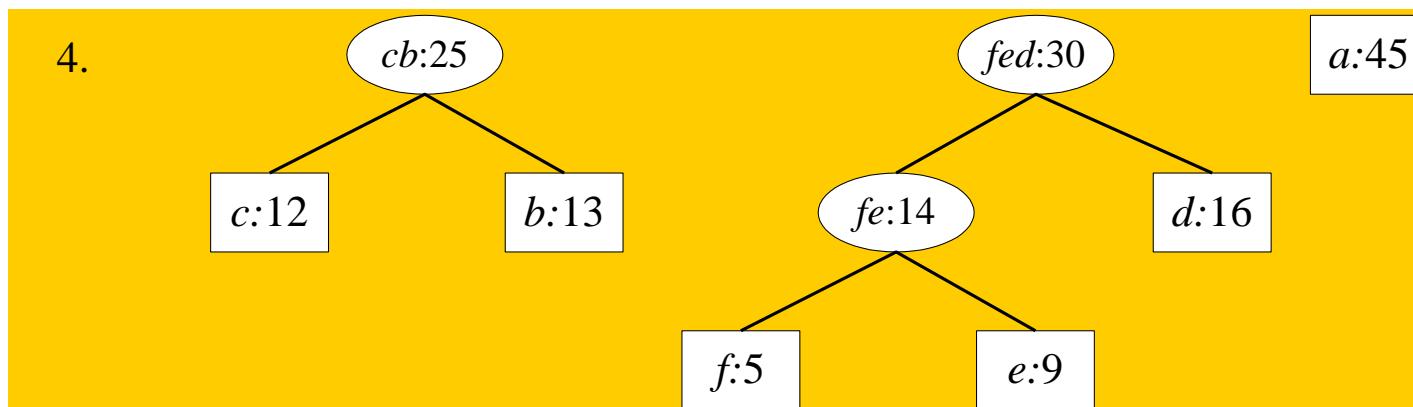
Contoh

Karakter	a	b	c	d	e	f
<hr/>						
Frekuensi	45	13	12	16	9	5



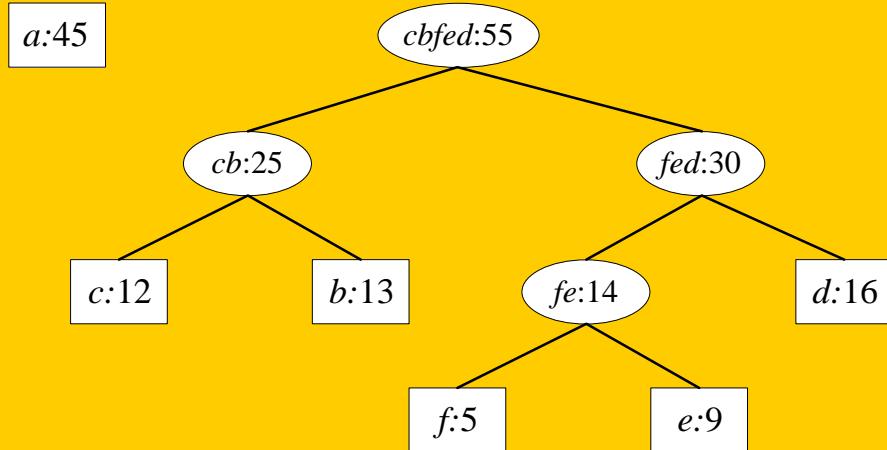


Karakter	a	b	c	d	e	f
<hr/>						
Frekuensi	45	13	12	16	9	5

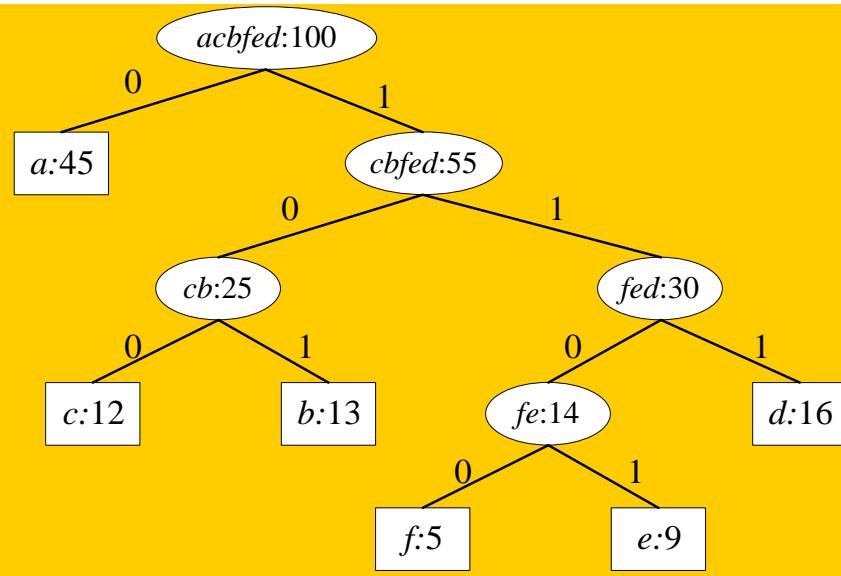




5.



6



Click to edit subtitle style

Thank You !