Bahan Kuliah IF2211 Strategi Algoritma

Algoritma *Greedy*

(Bagian 3)

Oleh: Rinaldi Munir

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika ITB
2022

9. Kode 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.

Fixed-length code

Misalkan terdapat sebuah pesan yang panjangnya 100.000 karakter dengan frekuensi kemunculan huruf-huruf (hanya *a*, *b*, *c*, *d*, *e*, f) di dalam pesan tersebut adalah sbb:

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

Pesan ‘bad’ dikodekan sebagai ‘001000011’

Pengkodean 100.000 karakter pesan membutuhkan 300.000 bit.

Variable-length code (Huffman code)

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

'bad' dikodekan sebagai '1010111 '

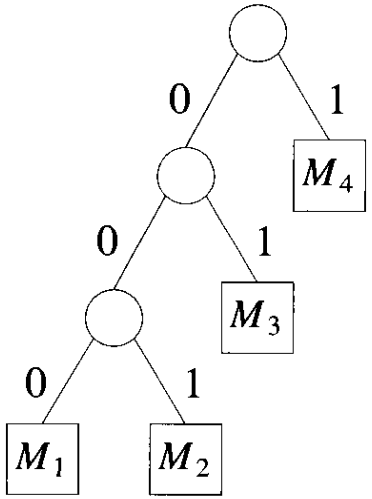
Pengkodean 100.000 karakter membutuhkan

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

Nisbah (ration) pemampatan: $(300.000 - 224.000)/300.000 \times 100\% = 25{,}3\%$

- Algoritma *greedy* untuk membentuk kode Huffman bertujuan untuk meminimumkan panjang kode biner untuk setiap simbol karakter di dalam pesan (M_1, M_2, …, M_n).

- Pembentukan kode Huffman menggunakan pohon biner (*binary tree*) berbobot. Setiap simpul daun pada pohon biner menyatakan simbol karakter di dalam pesan, sedangkan simpul bukan daun (*internal nodes* menyatakan penggabungan simbol-simbol.



- Setiap sisi pada pohon biner diberi nilai (label) 0 atau 1 secara konsisten (misalnya sisi cabang kiri dengan 0, sisi cabang kanan dengan 1)

- Lintasan dari akar ke daun sama dengan barisan bi-bit biner yang merepresentasikan simbol karakter pada daun.

- Meminimumkan panjang kode biner untuk setiap symbol karakter ekivalen dengan dengan meminimumkan panjang lintasan dari akar ke daun.

Langkah-langkah pembentukan pohon biner di dalam Algoritma 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.

4. Selanjutnya, beri label sisi-sisi di dalam pohon biner dengan 0 atau 1 secara konsisten.

5. Lintasan dari akar ke daun merepresentasikan string biner untuk setiap karakter

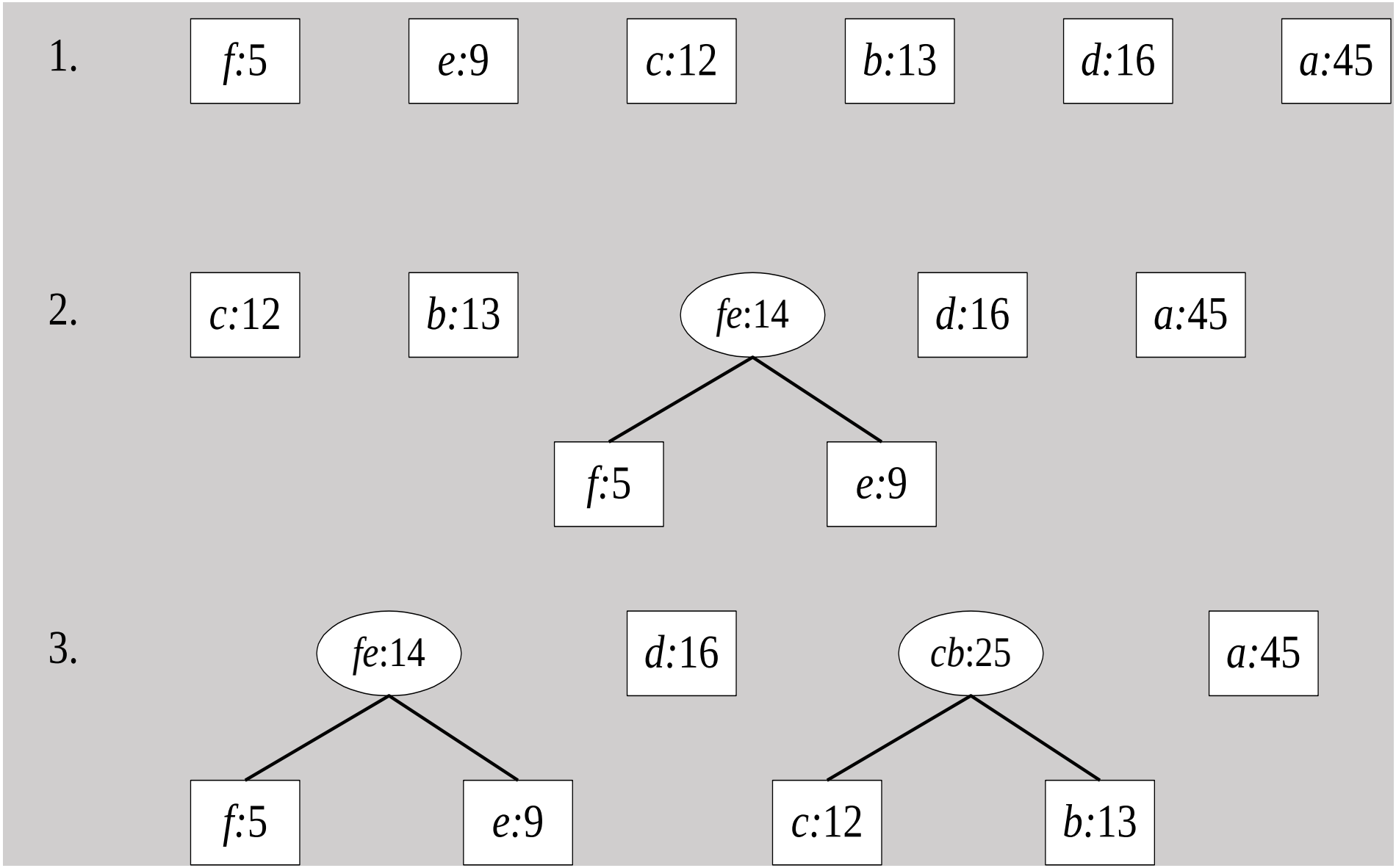
Kompleksitas algoritma Huffman: $O(n \log n)$

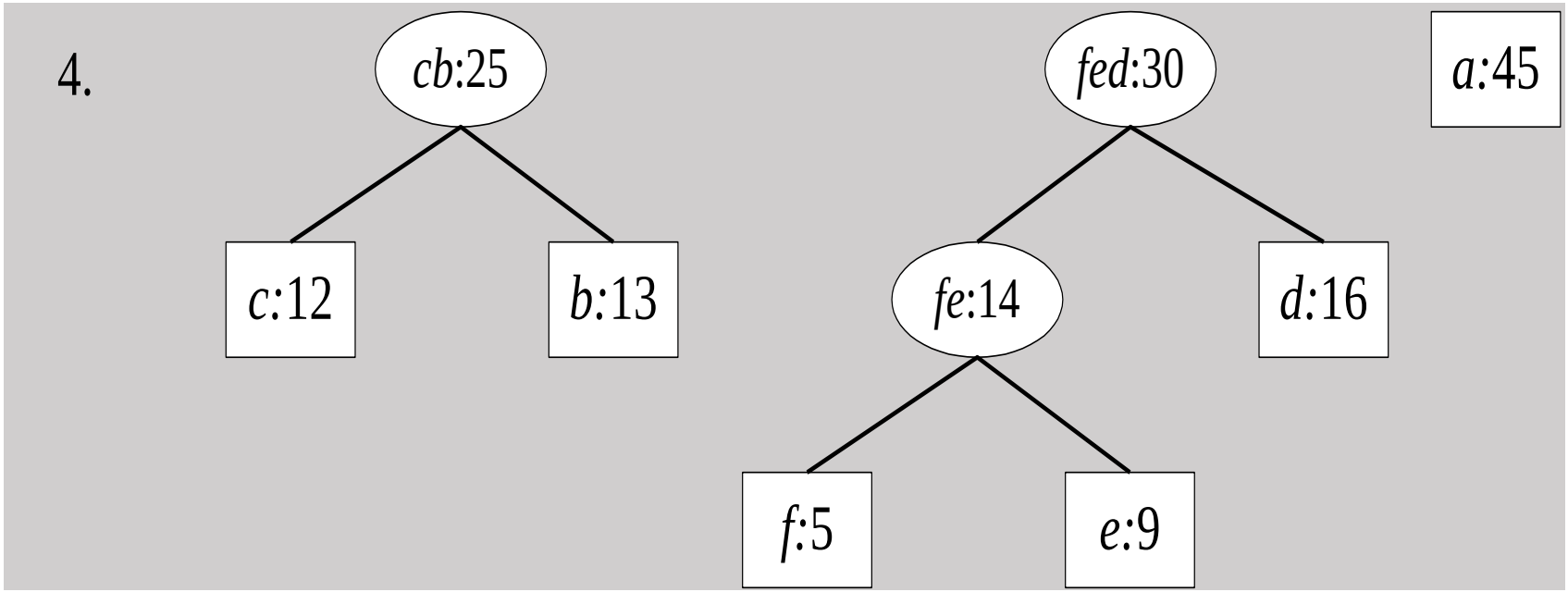
Contoh 15. Misalkan sebuah pesan panjangnya 100 karakter. Pesan hanya disusun oleh huruf-huruf *a*, *b*, *c*, *d*, *e*, *f*. Frekuensi setiap huruf di dalam pesan adalah sebagai berikut:

Karakter	*a*	*b*	*c*	*d*	*e*	*f*
Frekuensi	45	13	12	16	9	5

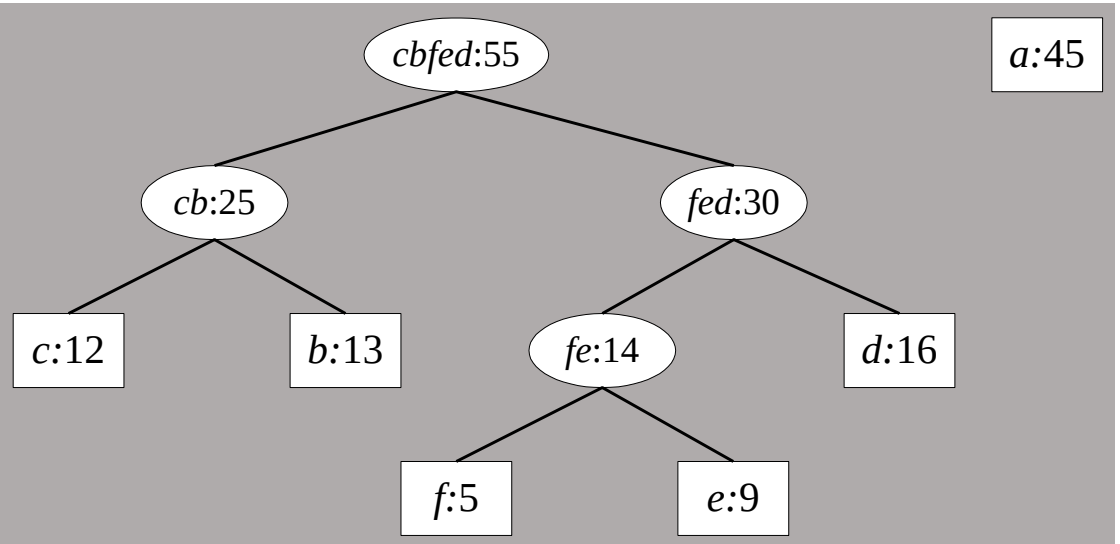
Carilah kode Huffman untuk setiap karakter di dalam pesan tersebut.

Penyelesaian:

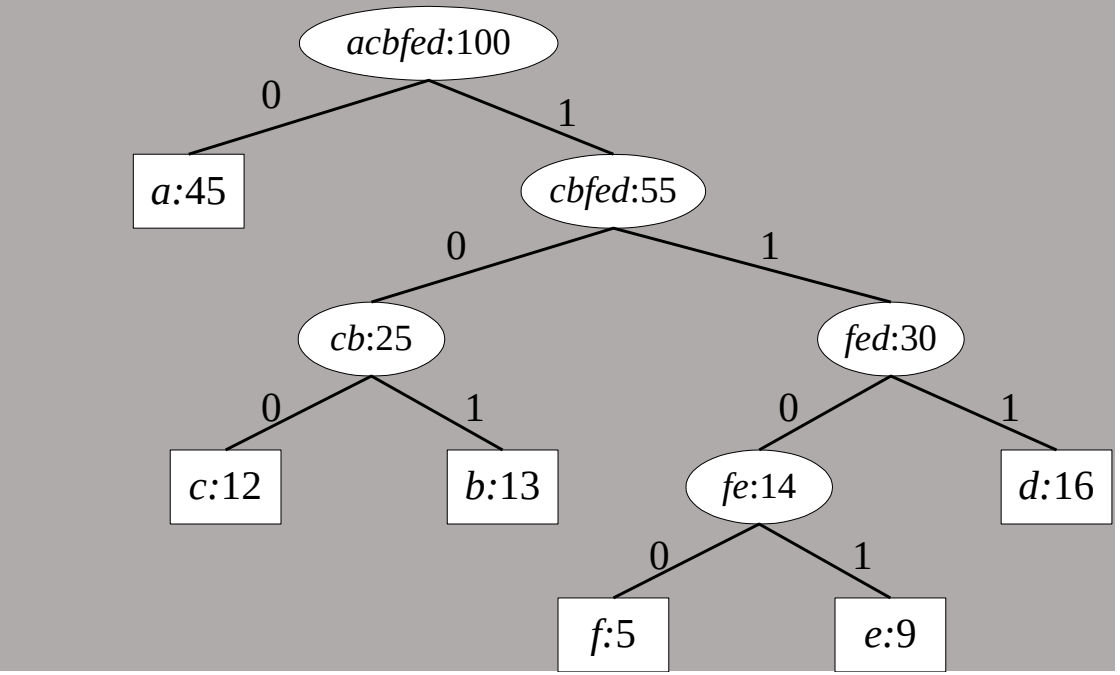



4.
cb:25
fed:30
a:45
c:12
b:13
fe:14
d:16
f:5
e:9

5.



6



Kode Huffman:

a = 0

b = 101

c = 100

d = 111

e = 1101

f = 1100

10. Pecahan Mesir (*Egyptian Fraction*)

Pecahan Mesir adalah pecahan, dalam bentuk p/q, yang dapat didekomposisi pecahan menjadi jumlah dari sejumlah pecahan yang berbeda:

$$\frac{p}{q} = \frac{1}{k_1} + \frac{1}{k_2} + \ldots + \frac{1}{k_n}$$

yang dalam hal ini, $k_1 < k_2 < \ldots < k_n$.

Contoh 16: Beberapa pecahan Mesir

$$\frac{2}{5} = \frac{1}{3} + \frac{1}{15}$$

$$\frac{5}{7} = \frac{1}{2} + \frac{1}{5} + \frac{1}{70}$$

$$\frac{87}{100} = \frac{1}{2} + \frac{1}{5} + \frac{1}{11}$$

- Pecahan yang diberikan mungkin mempunyai lebih dari satu representasi Mesir

  Contoh: 8/15 = 1/3 + 1/5

  8/15 = 1/2 + 1/30

- Persoalannya adalah, bagaimana mendekompoisinya dengan jumlah unit pecahan sesedikit mungkin.

  Contoh: $\frac{2}{5} = \frac{1}{3} + \frac{1}{15}$ $\rightarrow$ didekomposisi menjadi hanya dua unit pecahan

Strategi *greedy*: pada setiap langkah, tambahkan unit pecahan terbesar ke representasi yang baru terbentuk yang jumlahnya tidak melebihi nilai pecahan yang diberikan

Algoritma:

Input: p/q

1. Mulai dengan $i = 1$
2. Jika $p = 1$, maka $k_i = q$. STOP
3. $1/k_i$ = pecahan terbesar yang lebih kecil dari p/q
4. $p/q = p/q - 1/k_i$
5. Ulangi langkah 2.

- Contoh hasil:

  8/15 = 1/2 + 1/30 (optimal)

  5/7 = 1/2 + 1/5 + 1/70 (optimal)

  tetapi, *counterexample* dengan algoritma *greedy*:

  26/133 = 1/6 + 1/35 + 1/3990 (tidak optimal)

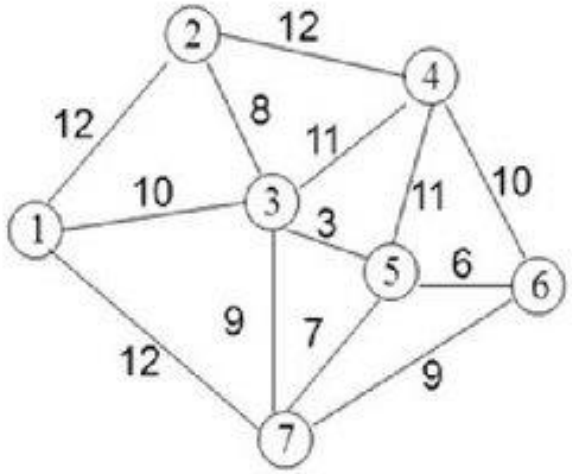
  seharusnya (dengan *brute force*)

  26/133 = 1/7 + 1/19 (optimal)

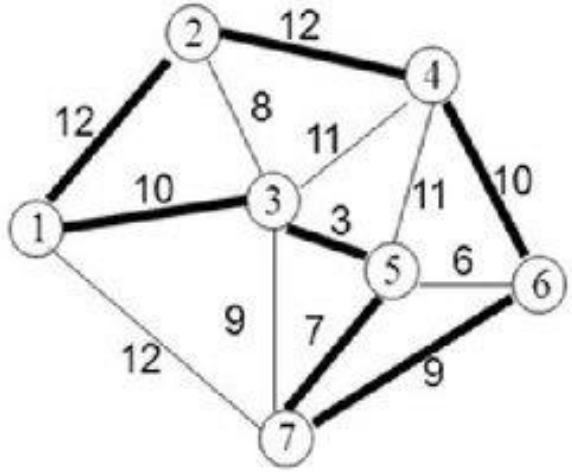
- Kesimpulan: algoritma *greedy* untuk masalah pecahan Mesir tidak selalu optimal

11. Travelling Salesperson Problem (TSP)

Persoalan: Diberikan *n* buah kota serta diketahui jarak antara setiap kota satu sama lain. Temukan perjalanan (*tour*) dengan jarak terpendek yang dilakukan oleh seorang pedagang sehingga ia melalui setiap kota tepat hanya sekali dan kembali lagi ke kota asal keberangkatan.



a) Persoalan TSP



b) Solusi TSP (lintasan yang dicetak tebal)

- Misalkan graf berbobot memiliki *n* buah simpul, v_1, v_2, …, v_n.

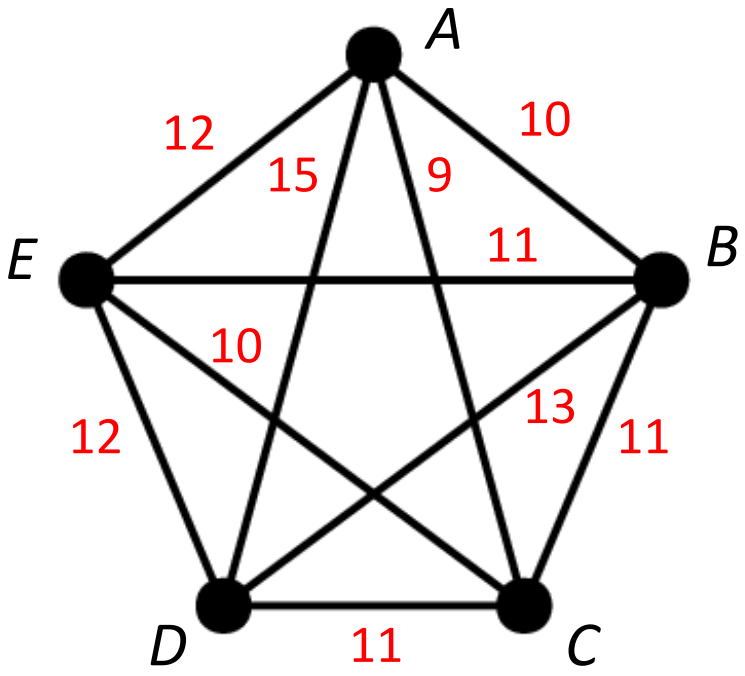
- Misalkan tur dimulai dari simpul v_1, mengunjungi simpul-simpul lainnya tepat sekali dan kembali lagi ke v_1.

- Dari simpul v_1, tur akan memilih $n - 1$ simpul-simpul lainnya yang dikunjungi.

- Dengan algoritma *greedy*, pada setiap langkah kita memilih simpul berikutnya yang akan dikunjungi.

- **Strategi *greedy***: pada setiap langkah *i*, pilih sisi dari v_i ke v_j yang memiliki bobot terkecil

Contoh 17: Diberikan sebuah graf berbobot dengan lima simpul sebagai berikut



\$000's	A	B	C	D	E
A	-	10	9	15	12
B	10	-	11	13	11
C	9	11	-	11	10
D	15	13	11	-	12
E	12	11	10	12	-

Tentukan tur terpendek dari simpul *A*!

Dengan algoritma *greedy*: $A \overset{9}{\rightarrow} C \overset{10}{\rightarrow} E \overset{11}{\rightarrow} B \overset{13}{\rightarrow} D \overset{15}{\rightarrow} A$ Bobot (jarak) = 58

Solusi optimal adalah: $A \overset{10}{\rightarrow} B \overset{11}{\rightarrow} C \overset{11}{\rightarrow} D \overset{12}{\rightarrow} E \overset{12}{\rightarrow} A$ Bobot (jarak) = 56

Kesimpulan: Algoritma *greedy* tidak memberikan solusi optimal untuk persoalan TSP

Aplikasi Algoritma *Greedy* pada Permainan *Othello* (*Riversi*)

Othello

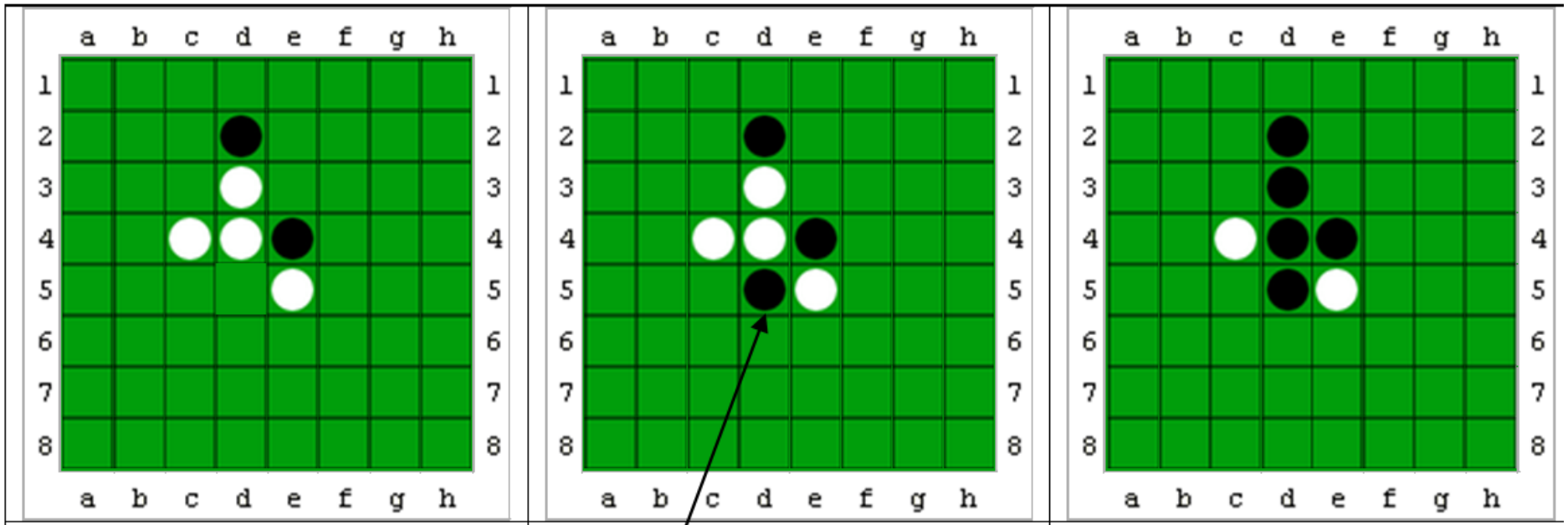
- *Othello* atau *Riversi* adalah permainan yang menggunakan papan (*board game*) dan sejumlah koin yang berwarna gelap (misalnya hitam) dan terang (misalnya putih).

- Ukuran papan biasanya 8 x 8 kotak (grid) dan jumlah koin gelap dan koin terang masing-masing sebanyak 64 buah. Sisi setiap koin memiliki warna yang berbeda (sisi pertama gelap dan sisi kedua terang).

- Pada permainan ini kita asumsikan warna hitam dan putih. Jumlah pemain 2 orang.

- Dalam permainan ini setiap pemain berusaha mengganti warna koin lawan dengan warna koin miliknya (misalnya dengan membalikkan koin lawan) dengan cara “menjepit” atau memblok koin lawan secara vertikal, horizontal, atau diagonal.

- Barisan koin lawan yang terletak dalam satu garis lurus yang diapit oleh sepasang koin pemain yang *current* diubah (*reverse*) warnanya menjadi warna pemain *current*.



- Setiap pemain bergantian meletakkan koinnya. Jika seorang pemain tidak dapat meletakan koin karena tidak ada posisi yang dibolehkan, permainan kembali ke pemain lainnya.

- Jika kedua pemain tidak bisa lagi meletakkan koin, maka permainan berakhir. Hal ini terjadi jika seluruh kotak telah terisi, atau ketika seorang pemain tidak memiliki koin lagi, atau ketika kedua pemain tidak dapat melakukan penempatan koin lagi.

- Pemenangnya adalah pemain yang memiliki koin paling banyak di atas papan.

- Algoritma Greedy dapat diaplikasikan untuk memenangkan permainan.

- Algoritma *greedy* berisi sejumlah langkah untuk melakukan penempatan koin yang menghasilkan jumlah koin maksimal pada akhir permainan.

- Algoritma *Greedy* dipakai oleh komputer pada tipe permainan komputer vs manusia.

Dua strategi greedy heuristik:

1. *Greedy by* jumlah koin

   Pada setiap langkah, koin pemain menuju koordinat yang menghasilkan sebanyak mungkin koin lawan. Strategi ini berusaha memaksimalkan jumlah koin pada akhir permainan dengan menghasilkan sebanyak-banyaknya koin lawan pada setiap langkah.

2. *Greedy by* jarak ke tepi

   Pada setiap langkah, koin pemain menuju ke koordinat yang semakin dekat dengan tepi arena permainan. Strategi ini berusaha memaksimumkan jumlah koin pada akhir permainan dengan menguasai daerah tepi yang sulit untuk dilangkahi koin lawan. Bahkan untuk pojok area yang sulit dilangkahi lawan.

Greedy by Jumlah Koin

1. Himpunan kandidat

   Langkah-langkah yang menghasilkan jumlah koin yang diapit.

2. Himpunan solusi

   Langkah-langkah dari Himpunan kandidat yang memiliki jumlah koin diapit paling besar.

3. Fungsi seleksi

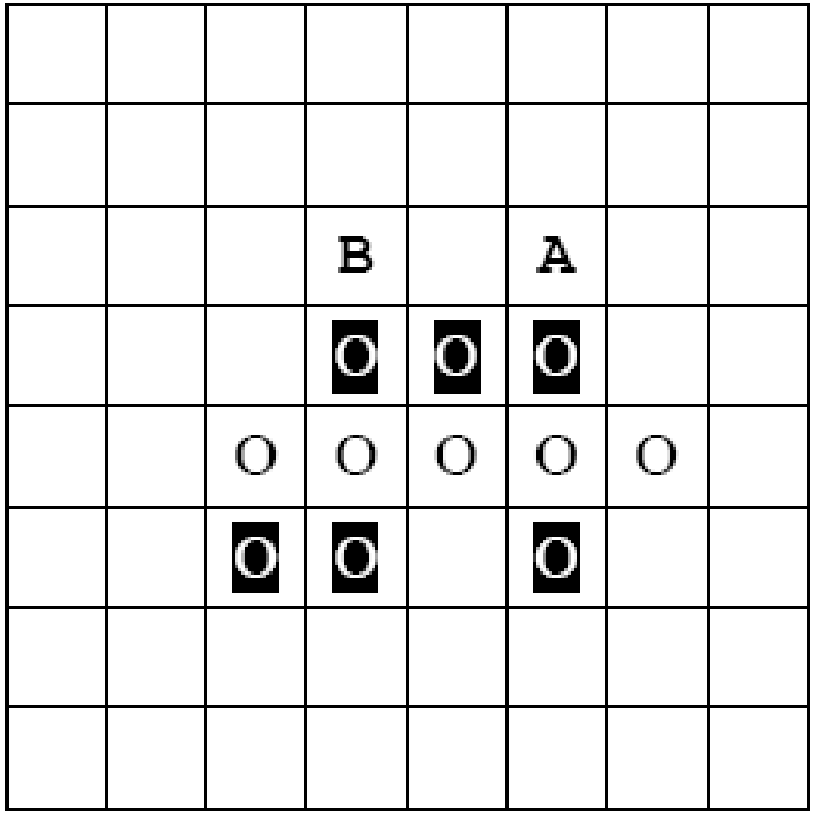
   Pilih langkah yang memiliki jumlah koin diapit paling besar

4. Fungsi kelayakan

   Semua langkah adalah layak

5. Fungsi obyektif

   Maksimumkan jumlah koin lawan


B
A

Pembahasan Soal UTS

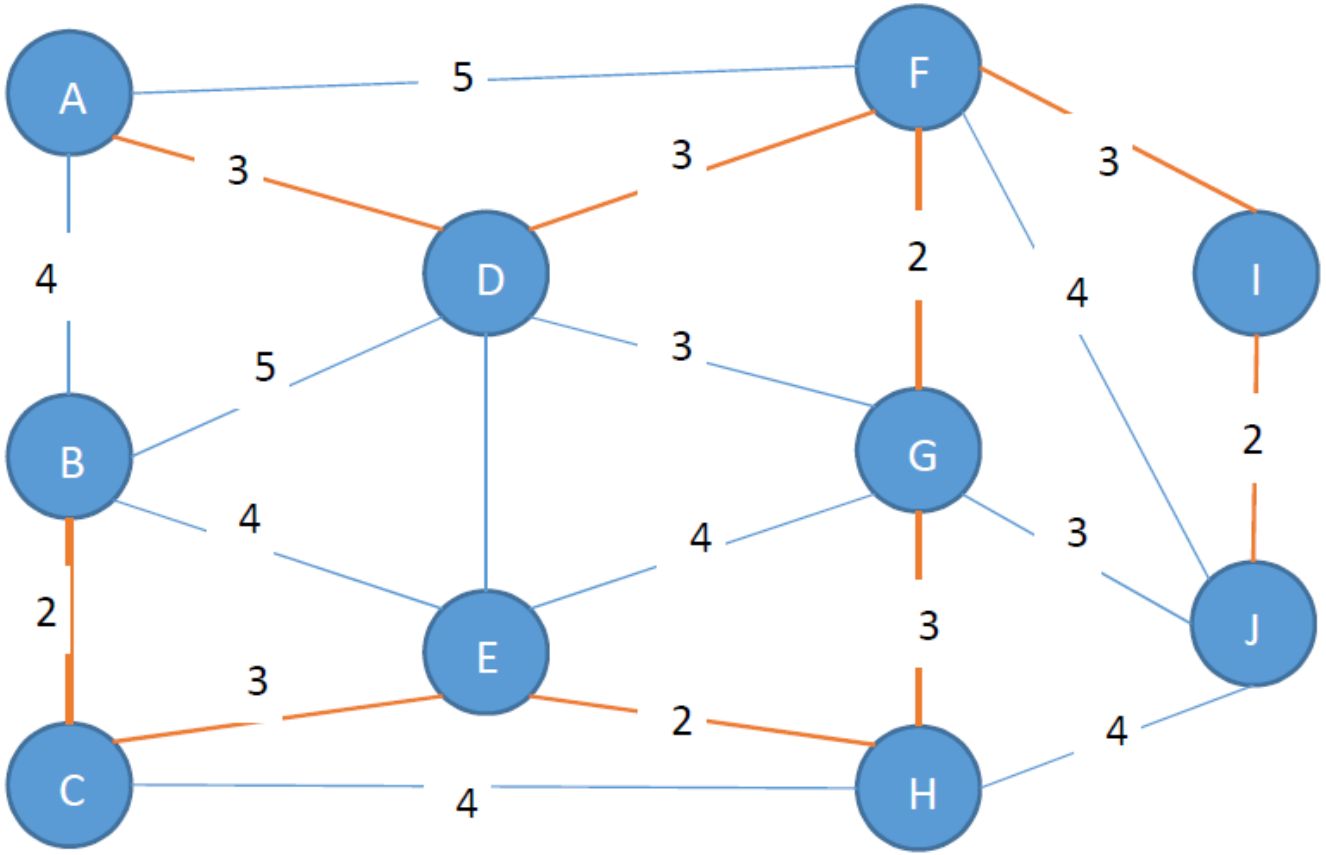
Soal UTS 2018

Pada persoalan *muddy city*, terdapat suatu kota tanpa jalan aspal. Jika hujan badai, jalanan menjadi sangat berlumpur, mobil terjebak di lumpur dan boots orang menjadi kotor. Walikota memutuskan untuk memasang *paving block* lebar, tetapi hanya ingin menghabiskan biaya seminimal mungkin. Terdapat dua kondisi: (1) *Paving block* cukup dipasang sehingga penduduk dapat datang dari rumah mereka ke rumah lain melalui jalan ber-*paving block*, mungkin saja harus melalui rumah lainnya; (2) Jumlah *paving block* seminimal mungkin agar biayanya juga minimal. Bantulah Pak Walikota dengan memberikan solusi jalur *paving block* yang akan dipasang. (Sumber gambar: http://statkclee.github.io/website-csunplugged)

1. a. (Nilai 5) Representasikanlah gambar kota di atas menjadi sebuah graf, dengan simpul merepresentasikan rumah, dan bobot sisi merepresentasikan jumlah *paving block* yang dibutuhkan. Jembatan dihitung sebagai satu *paving block*.
   b. (Nilai 15) Jika diselesaikan dengan *exhaustive search*, jelaskan strateginya seperti apa (tidak perlu pseudo-code), lalu tentukanlah kompleksitas algoritmanya dalam notasi big O.
2. (Nilai 15) Jika diselesaikan dengan *greedy*, rancanglah strategi *greedy* terbaiknya, dan tentukanlah kompleksitas algoritmanya dalam notasi big O.

Jawaban:

1. a. (Nilai 5) Representasikanlah gambar kota di atas menjadi sebuah graf, dengan simpul merepresentasikan rumah, dan bobot sisi merepresentasikan jumlah paving block yang dibutuhkan. Jembatan dihitung sebagai satu paving block.



b. (Nilai 15) Jika diselesaikan dengan exhaustive search, jelaskan strateginya seperti apa (tidak perlu pseudo-code), lalu tentukanlah kompleksitas algoritmanya dalam notasi big O.

Persoalan ini adalah persoalan minimum spanning tree.

Alternatif 1 strategi:

- enumerasi semua spanning tree berupa lintasan hamilton dgn cara membuat semua permutasi semua simpul (telah memenuhi syarat tidak siklik), terdapat (n-1)!
- evaluasi setiap spanning tree dengan menghitung bobot lintasan
- Ambil spanning tree dengan bobot paling minimum

Kompleksitas: O(n.n!), n adalah jumlah simpul

Alternatif 2 strategi:

- enumerasi semua subset sisi, terdapat 2^n, dengan n adalah jumlah sisi.
- evaluasi setiap subset yang memenuhi spanning tree (tidak siklik, melewati semua simpul) dengan menghitung bobot lintasan
- Ambil spanning tree dengan bobot paling minimum

Kompleksitas: $O(n.2^n)$, n adalah jumlah sisi

2. (Nilai 15) Jika diselesaikan dengan greedy, rancanglah strategi greedy terbaiknya, dan tentukanlah kompleksitas algoritmanya dalam notasi big O.

Alternatif 1: dgn menggunakan Kruskal:

1. Sort semua sisi berdasarkan jumlah paving block: BC(2), EH(2), FG(2), IJ(2), AD(3),...
2. Tambahkan sisi secara berurutan untuk membentuk pohon jika tidak membentuk siklik. Jika membentuk siklik, sisi diabaikan. Hal ini dilakukan sampai semua simpul sudah dicapai.

Solusi optimum : BC(2)-EH(2)-FG(2)-IJ(2)-AD(3)-CE(3)-DF(3)-FI(3)-GH(3). Total bobot=23.

Kompleksitas: O(|E|log|E|)

Alternatif 2: dgn menggunakan Prim:

Ambil simpul awal (misalnya A atau simpul dgn bobot terpendek), lalu bangun tree T secara greedy, dengan menambahkan sisi terpendek yang menghubungkan salah satu simpul yang telah terpilih dengan simpul yang belum terpilih.

Solusi optimum: A-D-F-G-H-E-C-B-I-J atau B-C-E-H-G-F-D-A-I-J. Total bobot=23.

Kompleksitas: $O(n^2)$

Soal UTS 2019

Misalkan terdapat sebuah larik $a[1..n]$ dengan n elemen bilangan bulat. Kita ingin menghitung $F = \sum_{i=1}^{n} i * a[i]$ sedemikian sehingga F bernilai maksimum.

(a) Jika diselesaikan secara *brute force/exhaustive search*, bagaimana caranya, dan berapa perkiraan kompleksitas waktunya (dalam notasi Big-Oh)? (5)

(b) Jika diselesaikan secara *greedy*, bagaimana caranya, dan berapa perkiraan kompleksitas waktunya? Contohkan jawaban anda untuk larik a = [3, 5, 6, 1]. (10)

Jawaban:

(a) Cari semua permutasi elemen-elemen larik a[1], a[2], …, a[n], yaitu ada sebanyak n! susunan permutasi. Untuk setiap susunan, hitung $S = \sum_{i=1}^{n} i * a[i]$, lalu pilih susunan permutasi yang memiliki nilai S maksimum.

Menghitung S = $\sum_{i=1}^{n} i * a[i]$ kompleksitsnya adalah O(n), karena ada n perkalian dan n pejumlahan.

Jadi kompleksitas algoritma seluruhnya adalah O(n · n!).

(b) Jika diselesaikan dengan algoritma *greedy*, maka strategi *greedy*-nya adalah "pada setiap langkah, pilih elemen larik yang terbesar lebih dahulu dan kalikan dengan nilai *i* yang terbesar lebih dahulu"

Hal ini dapat dilakukan lebih sangkil dengan mengurutkan larik sehingga terurut menaik (dari nilai terkecil hingga nilai terbesar), sehingga elemen terbesar akan dikalikan dengan indeksnya yang terbesar juga.

Contoh: a = [3, 5, 6, 1] urutkan menjadi [1, 3, 5, 6]

sehingga $S = \sum_{i=1}^{n} i * a[i]$ = 1 * a[1] + 2 * a[2] + 3 * a[3] + 4 * a[4]

= 1*1 + 2*3 + 3*5 + 4*6 = 46

Kompleksitas algoritma, jika larik belum diurutkan, adalah $O(n^2)$, sebab memilih elemen terbesar di dalam larik kompleksitasnya O(n). Jumlah langkah seluruhnya adalah *n* kali, sehingga kompleksitas algoritmanya menjadi $O(n^2)$.

Jika larik diurut terlebih dahulau dan waktu pengurutan larik diperhitungkan, maka kompleksitasnya adalah $O(n^2)$ atau O(n log n) tergantung algoritma pengurutan mana yang digunakan.

Soal UTS 2020

Diberikan n buah job. Setiap job i memiliki (s_i, f_i), s_i = waktu mulai, f_i= waktu selesai, $s_i < f_i$. Tersedia banyak mesin untuk mengerjakan semua job tersebut. Satu mesin dapat mengerjakan job-job secara sekuensial asalkan waktunya tidak bentrok (beririsan). Setiap job dikerjakan dari waktu mulai hingga waktu selesainya. Berapa *minimal* jumlah mesin yang dibutuhkan untuk mengerjakan **semua** job tersebut? Sebagai contoh: $n = 4$, $(s_i, f_i) = [(4,8), (1,3), (3,4), (4,7)]$, dibutuhkan minimal dua mesin, yaitu mesin 1: [(1,3), (4, 7)], mesin 2 : [(3, 4), (4, 8)]. Jika persoalan ini diselesaikan dengan algoritma *greedy*, jelaskan bagaimana strateginya, dan tentukan berapa kompleksitas waktu asimptotiknya. Jelaskan jawaban anda untuk contoh berikut: $n = 8$, $(s_i, f_i) = [(4,7), (2,5), (1,4), (3,7), (7,8), (1,3), (6,9), (5, 8)]$, lalu tentukan berapa jumlah mesin yang dibutuhkan. (15)

Jawaban:

Strategi *greedy*-nya adalah:

1. Urutkan job-job dalam urutan menaik berdasarkan waktu mulainya (s).
2. Mulai dengan mesin $k = 1$.

3. Masukkan job -job ke dalam mesin k yang waktu mulainya lebih besar atau sama dengan waktu selesai job yang telah dipilih sebelumnya.
4. Jika masih ada job yang tersisa, tambahkan mesin baru (k = k+ 1), lalu ulangi langkah 3 sampai seluruh job sudah dikerjakan oleh mesin-mesin.

Contoh: n = 8, (s_i, f_i) = [(4, 7), (2, 5), (1, 4), (3, 7), (7, 8), (1, 3), (6, 9), (5, 8)]

Diurutkan: [(1, 3), (1, 4), (2, 5), (3, 7), (4, 7), (5, 8), (6, 9), (7, 8)]

Mesin k = 1: [(1, 3), (3, 7), (7, 8)]

Mesin k = 2: [(1, 4), (4, 7)]

Mesin k = 3: [(2, 5), (5, 8)]

Mesin k = 4: [(6, 9)]

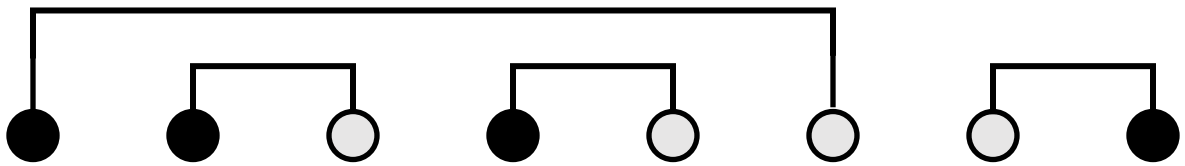
Jadi, dibutuhkan 4 mesin saja

Kompleksita algoritma adalah $O(n^2)$ jika waktu pengurutan diperhitungkan. Jika waktu pengurutan tidak diperhitungkan, maka kompleksitasnya adalah O(n).

Soal-soal Latihan

1. Connecting wires

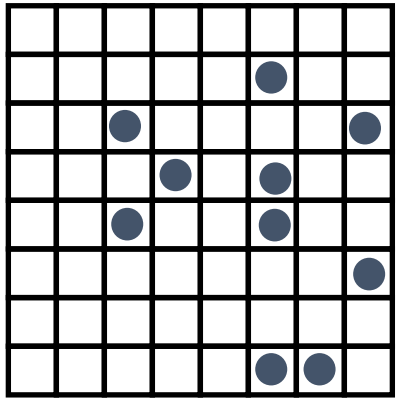
- There are n white dots and n black dots, equally spaced, in a line
- You want to connect each white dot with some one black dot, with a minimum total length of "wire"
- Example:



- Total wire length above is 1 + 1 + 1 + 5 = 8
- Do you see a greedy algorithm for doing this?
- Does the algorithm guarantee an optimal solution?
  - Can you prove it?
  - Can you find a counterexample?

2. Collecting coins

- A checkerboard has a certain number of coins on it
- A robot starts in the upper-left corner, and walks to the bottom left-hand corner
  - The robot can only move in two directions: right and down
  - The robot collects coins as it goes
- You want to collect *all* the coins using the *minimum* number of robots
- Example:



- Do you see a greedy algorithm for doing this?
- Does the algorithm guarantee an optimal solution?
  - Can you prove it?
  - Can you find a counterexample?

3. Persoalan Penugasan (*assignment problem*)

Misalkan terdapat *n* orang dan *n* buah pekerjaan (*job*). Setiap orang akan di-*assign* dengan sebuah pekerjaan. Penugasan orang ke-*i* dengan pekerjaan ke-*j* membutuhkan biaya sebesar *c*(*i*, *j*). Bagaimana algoritma *greedy* untuk melakukan penugasan sehingga total biaya penugasan adalah seminimal mungkin? Buatlah dua algoritma, algoritma pertama meng-assign job dengan orang, dan algoritma kedua meng-assign orang dengan job. Misalkan instansiasi persoalan dinyatakan sebagai matriks *C* sebagai berikut:

$$C = \begin{bmatrix} Job\,1 & Job\,2 & Job\,3 & Job\,4 & \\ 9 & 2 & 7 & 8 & \text{Orang } a \\ 6 & 4 & 3 & 7 & \text{Orang } b \\ 5 & 8 & 1 & 4 & \text{Orang } c \\ 7 & 6 & 9 & 4 & \text{Orang } d \end{bmatrix}$$

Tunjukkan dengan *counterexample* bahwa kedua algoritma tersebut tidak menjamin menghasilkan solusi optimal.

4. Misalkan sebuah mobil menempuh perjalanan sejauh *L* km, dimulai dari titik 0 dan berakhir pada kilometer *L*. Di sepanjang jalan terdapat SPBU pada jarak n_1, n_2, …, n_k km dari titik awal. Tangki bensin penuh hanya cukup untuk berjalan sejauh *d* km. Bagaimana kita menempuh tempat pemberhentian di SPBU agar kita berhenti sesedikit mungkin? Jelaskan strategi greedy-nya dan ilustrasikan jawaban anda dengan contoh L = 100 km, d = 30, dan $n_1 = 10$, $n_2 = 25$, $n_3 = 30$, $n_4 = 40$, $n_5 = 50$, dan $n_6 = 80$.

5. Misalkan terdapat 9 buah job dengan waktu pengerjaan adalah 3, 5, 6, 10, 11, 14, 15, 18, dan 20 menit. Anda memiliki tiga buah prosesor untuk menjalankan job-job tersebut. Bagaimana cara pengaturan job pada setiap prosesor sehingga waktu penyelesaian semua job adalah minimal? Waktu penyelesaian job dihitung dari prosesor yang terlama menyelesaikan job. Misalkan cara pengaturan job adalah sbb:

Waktu penyelesaian job = 18 + 11 + 6 = 35 menit.

TAMAT