Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Solusi Ujian Tengah Semester **IF2211 Strategi Algoritma**
Rabu, 8 Maret 2023
Waktu: 120 menit
Dosen: Rinaldi Munir (K1), Nur Ulfa Maulidevi (K2), Rila Mandala (K3)

Berdoalah terlebih dahulu agar Anda berhasil dalam mengerjakan ujian ini!

Brute force dan Greedy Algorithm

1. Diberikan dua buah larik integer, A[1..n] dan B[1..n]. Kita ingin meminimumkan $\sum_{i=1}^{n} A[i] * B[i]$.
   (a) Jika diselesaikan secara *brute force*, bagaimana caranya dan berapa kompleksitas algoritmanya?
   (b) Jika diselesaikan dengan algoritma *greedy*, bagaimana strateginya? Berapa kompleksitasnya? Ilustrasikan jawaban anda dengan contoh berikut: A = [7, 5, 1, 4] dan B = [6, 17, 9, 3].

(Nilai: 10)

Jawaban:

(a) Untuk setiap susunan elemen-elemen di dalam larik A, kalikan A[i] dengan B[i] untuk i = 1, 2, …, n, jumlahkan semua hasil kalinya, lalu pilih yang paling minimum. Ada n! susunan elemen di dalam larik A dan untuk setiap susunan ada n operasi perkalian elemen larik A dengan elemen larik B, serta n buah operasi penjumlahan untuk setiap perkalian larik A dan B. Jika kompleksitas waktu algoritma brute force dihitung hanya dari operasi perkalian saja, maka T(n) = n · n! = O(n · n!). Jika operasi penjumlahan pada setiap perkalian larik juga diperhitungkan maka terdapat T(n) = n · n! + n · n! = O(n · n!).

(b) Urutkan larik A sehingga terurut menaik dan larik B terurut menurun, lalu kalikan A[i] dengan B[i] untuk i = 1, 2, …, n. Mengurutkan larik A dengan algoritma pengurutan brute force membutuhkan waktu $O(n^2)$, mengurutkan larik B membutuhkan waktu $O(n^2)$, mengalikan elemen-elemen larik membutuhkan waktu O(n), menjumlahkan semua hasil perkalian membutuhkan waktu O(n). Jadi, kebutuhan waktu seluruhnya adalah $O(n^2) + O(n^2) + O(n) + O(n^2) = O(n^2)$. Jika algoritma pengurutan yang digunakan adalah Quicksort/Mergesort, maka kebutuhan waktu seluruhnya adalah O(n log n).

Contoh: A = [7, 5, 1, 4] urutkan menaik → A = [1, 4, 5, 7]

B = [6, 17, 9, 3] urutkan menurun → A = [17, 9, 6, 3]

$\sum_{i=1}^{n} A[i] * B[i]$ = (1 x 17) + (4 x 9) + (5 x 6) + (7 x 3) = 104

2. Terdapat *n* buah mata kuliah yang akan dijadwalkan pada sejumlah ruang kuliah. Setiap mata kuliah *i* memiliki waktu mulai s_i dan waktu selesai f_i. Bagaimana menjadwalkan semua kuliah pada ruang-ruang kuliah sehingga jumlah ruang kuliah yang dipakai seminimal mungkin? Tidak boleh ada dua atau lebih kuliah yang bentrok waktunya (beririsan waktunya) menggunakan ruang kuliah yang sama. Jelaskan strategi *greedy*-nya seperti apa dan berapa kompleksitas waktunya. Ilustrasikan jawaban anda dengan contoh berikut: **(Nilai: 15)**

n = 10, (s_i , f_i) = [(5, 6), (4, 7), (2, 5), (1, 4), (3, 7), (8, 10), (7, 8), (1, 3), (6, 9), (5, 8)]

Jawaban:

Strategi *greedy*-nya adalah:

(i) Urutkan mata kuliah dalam urutan menaik berdasarkan waktu mulainya (s).
(ii) Mulai dengan ruang kuliah ke-k = 1.

(iii) *Assign* kuliah-kuliah ke dalam ruang kuliah k yang waktu mulainya lebih besar atau sama dengan waktu selesai kuliah yang telah dipilih sebelumnya.

(iv) Jika ada kuliah yang tersisa, tambahkan ruang kuliah baru (k = k+ 1), lalu ulangi langkah (iii) sampai seluruh mata kuliah sudah di-*assign* ke ruang-ruang kuliah.

Contoh: $n = 10$, $(s_i , f_i) = [(5, 6), (4, 7), (2, 5), (1, 4), (3, 7), (8, 10), (7, 8), (1, 3), (6, 9), (5, 8)]$

Diurutkan: [(1, 3), (1, 4), (2, 5), (3, 7), (4, 7), (5, 6), (5, 8), (6, 9), (7, 8), (8, 10)]

Ruang kuliah k = 1: [(1, 3), (3, 7), (7, 8), (8, 10)]

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

Ruang kuliah k = 3: [(2, 5), (5, 6), (6, 9)]

Ruang kuliah k = 4: [(5, 8)]

Jadi, dibutuhkan 4 ruang kuliah saja

Meng-*assign* setiap kuliah ke ruang kuliah cukup dilakukan dalam waktu O(n) saja. Kompleksitas algoritma adalah $O(n^2)$ atau O(n log n) jika waktu pengurutan diperhitungkan. Jika waktu pengurutan tidak diperhitungkan, maka kompleksitasnya adalah O(n).

Teorema Master

3. Tentukan notasi notasi Big-Oh untuk kompleksitas waktu dalam bentuk relasi rekurens berikut. Jika teorema master tidak bsia diterapkan jelaskan alasannya: **(Nilai: 5)**

a. $T(n) = 27T(n/3) + n^3$
b. $T(n) = 0.5T(n/4) + n/2$
c. $T(n) = 16T(n/2) – n^2$

Jawaban:

a. $T(n) = 27T(n/3) + n^3 \rightarrow a = 27, b = 3, c = 1, d = 3 \rightarrow 27 = 3^3 \rightarrow a = b^d \rightarrow$ case 2 $\rightarrow O(n^3 \log n)$
b. $T(n) = 0.5T(n/4) + n/2 \rightarrow$ tidak bisa diterapkan Teoema Master karena $a < 1$
c. $T(n) = 16T(n/2) – n^2 \rightarrow$ tidak bisa diterapkan Teorema Master karena $c < 0$

Divide and Conquer

4. Di bawah ini adalah algoritma dengan strategi *divide and conquer* untuk menghitung jumlah daun dari sebuah pohon biner

```
Algoritma LeafCounter( T)
//Menghitung secara rekursif jumlah daun dalam sebuah pohon biner.
//Input: sebuah pohon biner T
//Output: Jumlah daun dalam pohon T
if  T = ∅
   return 0
else
  return LeafCounter(TL)+ LeafCounter (TR)
```

a. Apakah algoritma di atas benar atau salah (bukan secara sintaksis) untuk semua kemungkinan pohon biner ? Jika anda menjawab benar, maka jelaskan mengapa algoritma di atas benar. Jika anda menjawab salah, maka berikan sebuah contoh data (berupa pohon biner) dimana algoritma itu akan memberikan jawaban yang salah. **(Nilai : 5)**
b. Berdasarkan notasi algoritma di atas, buatlah dengan notasi di atas, algoritma untuk menghitung jumlah level yang ada pada sebuah pohon biner sbb **(Nilai : 5)** :

Algoritma *Levels*(*T*)
//Input: pohon biner T
//Output: jumlah level pohon biner T

Jawaban:

a. Algoritma ini salah, karena tidak akan berlaku jika pohon biner nya hanya berisi satu simpul/node. Jika pohon biner nya hanya berisi satu simpul/node maka hasilnya adalah 0 bukan 1. Algoritma yang benar adalah sbb :

```
Algoritma LeafCounter( T)
//Menghitung secara rekursif jumlah daun dalam sebuah pohon biner.
//Input: sebuah pohon biner T
//Output: Jumlah daun dalam pohon T
if T = ∅ return 0
else if TL = ∅ and TR = ∅return 1
else return LeafCounter(TL)+ LeafCounter (TR)
```

b. algoritma untuk menghitung jumlah level yang ada pada sebuah pohon biner sbb :

```
Algorithm Levels(T)
//Input: pohon biner T
//Output: jumlah level pohon biner T
if T = ∅ return 0
else return max(Levels(TL)+ LeafCounter (TR)) + 1 ;
```

Decrease and Conquer

5. Diberikan sebuah array yang dimulai dengan indeks 0 yang isinya adalah n bilangan integer yang sudah terurut dan perbedaan diantara 2 bilangan yang berurutan adalah selalu konstan. Diketahui bahwa diantara ***n*** bilangan tersebut ada 1 buah bilangan yang sengaja dihilangkan. Carilah sebuah bilangan yang hilang dalam urutan ***n*** bilangan tersebut.

   Sebagai contoh jika diberikan array sbb : int nums[] = { 5, 7, 9, 11, 15 }; maka *missing* elemen nya adalah 13.

   a. Selesaikan persoalan di atas (boleh *pseudo-code* atau memakai kata-kata sehari-hari namun harus jelas) dengan menggunakan strategi algoritma Brute Force dan hitunglah kompleksitas waktu nya. **(Nilai : 5)**

   b. Selesaikan algoritma tersebut dengan menggunakan pseudo-code (C-like) dengan menggunakan algoritma *Decrease and Conquer* yang harus lebih efisien daripada metode brute force di bagian ***a*** (harus ada komentar untuk setiap instruksinya, di instruksi itu melakukan apa) dengan membuat body dari fungsi di bawah ini :

```
int findMissingTerm(int nums[], int n)
{
}
```

   Dimana `nums[]` adalah array yang sudah berisi bilangan integer, dan ***n*** adalah jumlah bilangan yang ada di array nums[]. **(Nilai : 10)**

   c. Jalankan step-step dari algoritma yang sudah anda buat di bagian ***b*** pada array { 5, 7, 9, 11, 15 } **(Nilai : 5)**

d. Tentukan kompleksitas waktu algoritma yang anda buat tersebut. **(Nilai : 5)**

Jawaban:

a. Brute force : Hitung konstanta pembeda tiap elemen array. Misalkan ***d.*** Mulai dari elemen pertama, bandingkan nums[k] dengan nums[k+1], dengan k mulai 0 sampai n-2. Jika ada nums[k+1] dikurangi nums[k] tidak sama dengan ***d*** maka elemen yang hilang adalah nums[k]+d.
Kompleksitas waktunya adalah O(n).

b. Algoritmanya memakai binary search sbb :

```
// Fungsi untuk menemukan sebuah elemen
int findMissingTerm(int nums[], int n)
{
// ruang pencariannya adalah nums[low...high]
  int low = 0, high = n - 1;

  // menghitung perbedaan di antara 2 elemen yang berturutan
  int d = (nums[n - 1] - nums[0]) / n;

  // looping sampai ruang pencariannya habis
  while (low <= high)
  {
    // cari indeks tengah
    int mid = high - (high - low) / 2;

    // memeriksa perbedaan antara elemen array di tengah dg elemen tetangganya di kanan
    if (mid + 1 < n && nums[mid + 1] - nums[mid] != d) {
      return nums[mid + 1] - d;
    }

    // memeriksa perbedaan antara elemen array di tengah dg elemen tetangganya di kiri
    if (mid - 1 >= 0 && nums[mid] - nums[mid - 1] != d) {
      return nums[mid - 1] + d;
    }

    // Jika elemen array yang hilang ada di  subarray kiri , maka reduksi
    // ruang pencarian ke subarray kiri nums[low...mid-1]
    if (nums[mid] - nums[0] != (mid - 0) * d) {
      high = mid - 1;
    }

    // Jika elemen array yang hilang ada di  subarray kanan , maka reduksi
    // ruang pencarian ke subarray kanan nums[mid+1...high]
    else {
      low = mid + 1;
    }
  }
}
```

c. Jalannya steps dari algoritma yang sudah anda buat di bagian ***b*** pada array { 5, 7, 9, 11, 15 }

D = (15-5)/5 = 2
Low = 0, high = 4
Mid = 4-(4-0)/2 = 2
Nums[mid] = nums[2] = 9
Nums[mid+1] – nums[mid] = 2
(nums[mid] - nums[0] = 4) != ((mid - 0) * d) = 2*2) ini adalah false maka

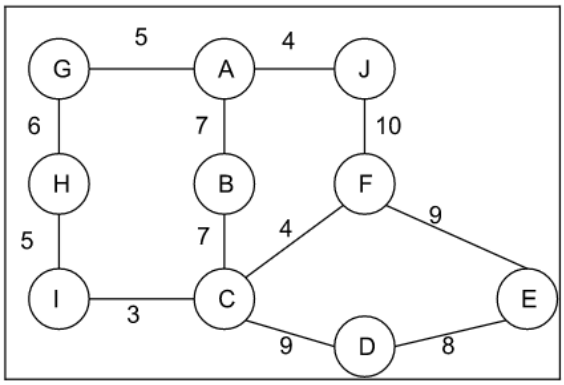
Low = mid+1 = 2 + 1 = 3
High = 4

Mid = 4-(4-3)/2 = 3
Nums[mid] = nums[3] = 11
Nums[mid+1] – nums[mid] = 15 – 11 = 4 tidak sama dengan 2
Maka return nums[mid+1] – 2 = 13.
SELESAI

d. Karena ini pada prinsipnya adalah binary search, maka kompleksitas waktunya adalah ***O(log n)***

Brute Force, DFS dan BFS

Terdapat graf sebagai berikut. Simpul merepresentasikan kota, dan sisi merepresentasikan jalan yang menghubungkan antar kota dan jaraknya. **Prioritas pemilihan simpul berdasarkan urutan abjad**.



Gambar 1. Graf Terhubung

6. Jika ingin dicari jalur terpendek dari A menuju ke I. **(Nilai 10)**
   a. Jelaskan langkah-langkah untuk mendapatkan jalur terpendek dengan memanfaatkan Brute Force.
   b. Jelaskan dengan singkat kompleksitas waktu untuk jawaban butir (a) dalam Big O.

Jawaban:
Jalur terpendek dengan Brute Force

a. Langkah-langkah:

- Enumerasi semua kemungkinan jalur dari simpul A ke simpul I, paling banyak kemungkinan jalur adalah permutasi dari n-2 dengan n adalah banyaknya simpul;
- Setiap kemungkinan jalur dievaluasi total jaraknya, simpan jalur dengan jarak terpendek saat itu;
- Jika pencarian berakhir (semua kemungkinan sudah dievaluasi), kembalikan jalur dengan jarak terpendek
-

b. Kompleksitas algoritme:

- Enumerasi semua kemungkinan jalur dari simpul A ke simpul I, adalah permutasi dari n-2 dengan n adalah banyaknya simpul; jadi banyaknya kemungkinan yang perlu diperiksa: O((n-2)!)
- Setiap kemungkinan jalur perlu diperiksa total jaraknya, dengan kompleksitas: O(n)

Jadi total kompleksitas: O(n.(n-2)!)

7. Ingin ditelusuri semua simpul pada gambar 1 menggunakan pendekatan BFS dan DFS. **(Nilai 10)**
   a. Tuliskan urutan penelusuran semua simpul pada Gambar 1 yang dimulai dari simpul A dengan BFS.

b. Tuliskan urutan penelusuran semua simpul pada Gambar 1 yang dimulai dari simpul A dengan DFS.

Jawaban:
Penelusuran semua simpul pada Gambar 1.

a. BFS: A - B - G - J - C - H - F - D - I - E

b. DFS: A - B - C - D - E - F - J - I - H - G

8. Jika ingin dicari jalur dari A menuju ke I menggunakan BFS dan DFS. **(Nilai 15)**
   a. Tuliskan proses pencarian jalur menggunakan BFS menggunakan contoh tabel 1.
   b. Tuliskan proses pencarian jalur menggunakan DFS menggunakan contoh tabel 1.
   c. Bandingkan jalur hasil pencarian (jarak) dari A menuju I menggunakan Brute Force, BFS, dan DFS.
   Catatan: Untuk BFS dan DFS, simpul yang sudah pernah di ekspan tidak diperiksa lagi. Pencarian berhenti ketika simpul yang diperiksa adalah simpul goal (simpul I).

Tabel 1. Proses Pencarian Jalur dari Simpul A ke Simpul I

Iterasi	Simpul Ekspan	Simpul Hidup
1	A	B_A, G_A, J_A
2	…	…
…	…	…
Jalur Hasil:	…	
Jarak:	…	

Jawaban:

(a) Tabel 1. Proses Pencarian Jalur dari Simpul A ke Simpul I BFS

Iterasi	Simpul Ekspan	Simpul Hidup
1	A	B_A, G_A, J_A
2	B_A	G_A, J_A, C_{BA}
3	G_A	J_A, C_{BA}, H_{GA}
4	J_A	C_{BA}, H_{GA}, F_{JA}
5	C_{BA}	H_{GA}, F_{JA}, D_{CBA}, F_{CBA}, I_{CBA}
6	H_{GA}	F_{JA}, D_{CBA}, F_{CBA}, I_{CBA}, I_{HGA}
7	F_{JA}	D_{CBA}, F_{CBA}, I_{CBA}, I_{HGA}, E_{FJA}

8	D_{CBA}	F_{CBA}, I_{CBA}, E_{FJA}, E_{DCBA}
9	F_{CBA}	I_{CBA}, E_{FJA}, E_{DCBA}, E_{FCBA}
10	I_{CBA}	Goal state, berhenti
Jalur Hasil:	A - B - C - I	
Jarak:	17	

(b) Tabel 2. Proses Pencarian Jalur dari Simpul A ke Simpul I DFS

Iterasi	Simpul Ekspan	Simpul Hidup
1	A	B_A, G_A, J_A
2	B_A	C_{BA}, G_A, J_A
3	C_{BA}	D_{CBA}, F_{CBA}, $I_{CBA,}$ G_A, J_A
4	D_{CBA}	E_{DCBA}, F_{CBA}, $I_{CBA,}$ G_A, J_A
5	E_{DCBA}	F_{EDCBA}, F_{CBA}, $I_{CBA,}$ G_A, J_A
6	F_{EDCBA}	J_{FEDCBA}, F_{CBA}, $I_{CBA,}$ G_A, J_A
7	J_{FEDCBA}	F_{CBA}, $I_{CBA,}$ G_A, J_A
8	F_{CBA}	$I_{CBA,}$ G_A, J_A
9	I_{CBA}	Goal state, berhenti
Jalur Hasil:	A - B - C - I	
Jarak:	17	

c. Dengan brute force (exhaustive search), maka didapatkan jalur terpendek dari A ke I adalah 16, dan hal ini tidak tercapai saat melakukan pencarian dengan BFS maupun DFS, dengan prioritas simpul berdasarkan abjad.