# BUKLET PEMBAHASAN SOAL


## PENYISIHAN PEMROGRAMAN GEMASTIK 9
## 27 Agustus 2016

Soal-Soal

| Kode | Judul | Penulis |
|---|---|---|
| A | Membangun Menara | William Gozali |
| B | Kode Keras | Alham Fikri Aji |
| C | Menggemaskan dan Cantik | Alham Fikri Aji |
| D | Saklar Lhompat | Ashar Fuadi |
| E | Libur Tahun Baru | Ashar Fuadi |
| F | Kartu Truf | Felix Halim |

## A. Membangun Menara

Oleh: William Gozali
Tema: *sorting*

Ini adalah soal paling mudah pada penyisihan GEMASTIK 9.

Untuk setiap balok yang ada, kita boleh memutar orientasi balok tersebut sesuka hati. Artinya, untuk membentuk menara tertinggi/terendah, kita cukup memutar setiap balok sehingga sisi terpanjang/terpendek balok menjadi tinggi balok tersebut. Dengan kata lain, untuk setiap balok, cari panjang sisi maksimum dan minimumnya.

Kompleksitas: O(N).

## B. Kode Keras

Oleh: Alham Fikri Aji
Tema: *line sweep*, *case study*

Pertama-tama, perhatikan bahwa hanya ada 3 kemungkinan jawaban: -1, 1, dan 2.

- -1 artinya tidak mungkin. Ini terjadi jika interval i berada di dalam interval lain j. Dengan kata lain, terdapat j != i dengan S[j] ≤ S[i] < E[i] ≤ E[j].
- 1 artinya ada waktu X dengan S[i] ≤ X < E[i], dan tidak ada orang lain selain i yang *online* pada waktu X.
- Jika kondisi -1 dan 1 tidak terjadi, maka jawabannya pasti 2. Pembaca dipersilakan untuk membuktikan sendiri kenapa jawaban tidak mungkin lebih dari 2.

Berdasarkan analisis tersebut, kita tinggal mendeteksi apakah setiap interval i termasuk pada kasus -1 atau 1.

**Kasus -1**: Terdapat 86.400 kemungkinan waktu. Untuk setiap waktu X, carilah interval k yang melingkupi X dan memiliki nilai E terbesar. Hal ini dapat dilakukan dengan metode *line sweep* atau *sliding window*. Pada setiap waktu X, asumsikan variabel k adalah suatu interval dengan S[k] ≤ X ≤ E[k], yang mana E[k] adalah terbesar di antara setiap k. Setiap kali kita iterasi nilai X baru, jika itu sama dengan suatu S[i], cek apakah nilai E[i] lebih besar daripada E[k]. Jika iya, perbarui nilai k dengan i. Jika tidak, maka dapat dijamin bahwa interval i tertutup oleh interval k. Artinya, interval i masuk dalam kasus -1. Perlu diperhatikan juga jika ada dua interval yang identik, maka kedua interval tersebut masuk ke dalam kasus -1.

**Kasus 1**: Kita bisa melakukan *line sweep* serupa dengan kasus -1. Pertama-tama, kumpulkan semua titik S[i] dan E[i], kemudian urutkan. Siapkan juga sebuah struktur data *set*. Lalu, iterasi semua titik. Untuk setiap titik, jika itu adalah S[i], masukkan i ke dalam set, dan jika menemukan E[i], keluarkan i dari set. Jika pada suatu waktu X, set tersebut tepat berisi 1 anggota, maka anggota set tersebut masuk ke dalam kasus 1.

## C. Menggemaskan dan Cantik

Oleh: Alham Fikri Aji
Tema: *greedy*

Misalkan {a, b, c} adalah 3 orang dengan nilai kegemasan tertinggi. Maka, {a, b, c} merupakan calon terkuat untuk tim KegeMasan. Namun ada kemungkinan {a, b, c} juga memiliki nilai kecantikan yang tinggi, sehingga mereka juga berkemungkinan menjadi tim KecanTIKan. Dengan demikian, maka kita masih memerlukan {d, e, f}, yakni 3 orang dengan nilai kegemasan tertinggi setelah {a, b, c} sebagai calon terkuat selanjutnya.

Berdasarkan pengamatan di atas, calon solusi adalah 6 orang terkuat berdasarkan nilai kecantikan dan 6 orang terkuat berdasarkan nilai kegemasan. Kumpulkan mereka dalah satu himpunan calon S. Ukuran S paling besar adalah 12. Setelah itu, lakukan *brute force* untuk membentuk tim KegeMasan dan KecanTIKan dari keduabelas orang tersebut.

Kompleksitas: O(mengambil 6 orang terkuat) + O(*brute force* pembagian) (sekitar 3^12).

## D. Saklar Lhompat

Oleh: Ashar Fuadi
Ilustrasi oleh: William Gozali
Tema: *ad-hoc*

Jika kita memiliki K buah stop kontak, apakah mungkin L steker lurus dan B steker bengkok dapat terpasang seluruhnya? Untuk mengetahui ini, kita harus tahu terlebih dahulu bagaimana cara pencolokan steker-steker yang efisien.

Pertama-tama, jika kita ingin menyambungkan stop kontak pada stop kontak lain, akan optimal jika stekernya dicolok pada lubang tengah dari sebuah stop kontak yang sudah ada. Karena jika dipasang pada lubang pinggir, akan memakan tempat yang sebenarnya bisa dipakai untuk steker-steker bengkok. Dengan demikian, untuk K buah stop kontak, kita memiliki:

- 2K buah lubang pinggir
- K+1 buah lubang tengah

Maka, selanjutnya kita tinggal distribusikan L dan B pada lubang-lubang tersebut. Prioritas pertama adalah memasang steker-steker bengkok ke lubang pinggir. Jika lubang tidak cukup, kita pasangkan sisa steker bengkok ke lubang tengah. Jika kita pasang steker bengkok ke lubang tengah, maka setiap saklar akan menghabiskan 2 lubang. Setelah semua steker bengkok terpasang, kita tinggal hitung lubang tersisa, dan cek apakah lubang tersisa ≤ steker lurus.

Dengan cara di atas, kita tinggal mencoba semua nilai K untuk mencari solusi yang terbaik.

Kompleksitas solusi di atas bergantung pada implementasi. Namun dengan implementasi yang efisien, solusi di atas adalah O(N), dengan N adalah jawabannya.

**Catatan penulis 1**: Contoh kasus uji ketiga mengecoh; karena sesungguhnya susunan tersebut (yang mana tidak optimal) kebetulan menghasilkan jawaban yang sama dengan susunan optimal.

**Catatan penulis 2**: Awalnya, batasan 2 ≤ L+B direncanakan sebagai 1 ≤ L+B. Hal ini memungkinkan adanya *tricky case*: jika laptop hanya ada 1, tidak perlu membeli stop kontak; cukup colokkan saja *charger*-nya langsung ke tembok :) Namun, kami merasa ini terlalu "kejam" dan menggantinya.

## E. Libur Akhir Tahun

Oleh: Ashar Fuadi
Tema: *dynamic programming*

Misalkan dp[i][s] adalah besarnya keindahan maksimum jika Budi berangkat dari kota i menuju kota Astik, dengan anggaran sebesar s. Nilai dp[i][s] didefinisikan sebagai berikut:

dp[i][s] =

- 0, jika i = N (sudah sampai di kota Astik).
- max(
  - V[i] + dp[i+1][s-K[i]] (naik kereta; hanya jika s ≥ K[i])
  - max{dp[i+1][s-P], dp[i+2][s-P], …, dp[N][s-P]} (naik pesawat; hanya jika s ≥ P)

  )

Jawabannya adalah dp[1][S].

Jika rekurens di atas langsung diimplementasikan, kompleksitasnya adalah O(N^3), yang tentunya akan mendapatkan TLE. Rekurens tersebut perlu dioptimisasi. Terdapat setidaknya dua cara:

**1. Tambahkan parameter baru: "sedang naik pesawat"**

Sekarang, *state* dp akan menjadi dp[i][s][p]. Variabel p adalah variabel *boolean* menunjukkan apakah sekarang kita sedang naik pesawat atau tidak. Rekurensnya menjadi:

dp[i][s][p] =

- 0, jika i = N (sudah sampai di kota Astik).
- max(
  - V[i] + dp[i+1][s-K[i]][false] (naik kereta; hanya jika s ≥ K[i])
  - dp[i+1][s-P][true] (naik pesawat; hanya jika p = false dan s ≥ P)
  - dp[i+1][s][true] (melanjutkan perjalanan pesawat; hanya jika p = true)

  )

Kompleksitasnya menjadi O(N^2).

**2. Buat *array* bantuan untuk akumulasi DP**

Perhatikan bahwa *bottleneck* perhitungan DP awal kita ada pada perhitungan saat Budi memilih naik pesawat: kita harus *loop* untuk mencoba semua kemungkinan kota akhir tujuan pesawat tersebut. Perhatikan sekali lagi ekspresi berikut ini:

$$\max\{dp[i+1][s-P], dp[i+2][s-P], \ldots, dp[N][s-P]\}$$

Dimensi kedua selalu sama, dan dimensi pertama adalah i+1, i+2, … N. Maka, kita sebenarnya bisa membuat array lain, sebut saja dpMax[][], yang didefinisikan sebagai berikut:

$$dpMax[i][s] = \max\{dp[i][s], dp[i+1][s], dp[i+2][s], \ldots, dp[N][s]\}$$

atau bisa juga ditulis seperti ini:

$$dpMax[i][s] = \max(dp[i][s], dpMax[i+1][s])$$

Sehingga, nilai pilihan untuk naik pesawat bisa disederhanakan menjadi dpMax[i+1][s-P].

Kompleksitasnya juga menjadi O(N^2).

## F. Kartu Truf

Oleh: Felix Halim
Tema: *probability, combinatorics*

Misalkan kartu yang kita keluarkan berjenis S. Misalkan di tangan kita terdapat K buah kartu berjenis S. Artinya, terdapat 12 - K kartu berjenis S yang ada di tangan pemain-pemain selain kita. Kita bisa lakukan *brute force* untuk mencoba semua distribusi 12 - K kartu pada 3 pemain lainnya; yakni terdapat 3^(12 - K) kemungkinan. Jawaban akhirnya adalah total peluang distribusi yang menyebabkan Anda menang.

Untuk menentukan apakah sebuah distribusi menyebabkan kita menang, cukup mudah. Kita akan menang hanya jika setiap pemain lainnya tidak memiliki kartu berjenis S yang bernilai lebih tinggi daripada kartu yang hendak Anda keluarkan.

Yang lebih menantang adalah menghitung peluang sebuah distribusi terjadi. Misalkan C(i, j) = banyaknya cara memilih j benda dari i benda. Pertama-pertama, hitung ada berapa cara pembagian 52 kartu kepada 4 orang, dengan 13 kartu kita sudah ditentukan dari awal. Banyaknya cara adalah:

$$C(39, 13) * C(26, 13) \quad \ldots. (i)$$

Kemudian, misalkan dalam distribusi tersebut orang 1 mendapat A buah kartu berjenis S, orang 2 mendapat B buah kartu berjenis S, dan orang 3 mendapat C buah kartu berjenis S. Banyaknya cara distribusi ini terjadi adalah

$$C(39, 13 - A) * C(12 - K, A) * C(39 - (13 - A), 13 - B) * C(12 - K - A, B) \quad \text{…. (ii)}$$

Peluang sebuah distribusi terjadi adalah hasil pembagian ekspresi (i) dengan (ii).

Catatan: alih-alih *brute-force*, dapat juga dilakukan DP untuk menghitung semua distribusi. Observasinya adalah: untuk setiap pemain lainnya, kita hanya peduli berapa banyaknya kartu yang nilainya lebih rendah dan banyaknya kartu yang lebih tinggi daripada kartu yang kita pegang. Sisanya diserahkan kepada pembaca.