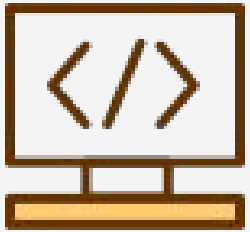


Pembahasan Penyisihan SCPC

COMPFEST 9

Penyisihan SCPC

Soal-soal pada penyisihan SCPC dipersiapkan oleh:

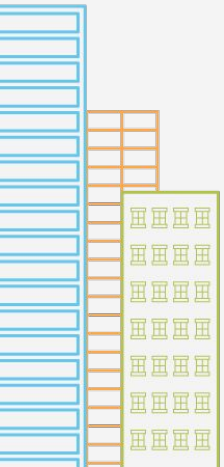
- Muhammad Ayaz Dzulfikar
- Degoldie Sonny
- Firman Hadi Prayoga
- Norman Bintang
- Usama
- Windi Chandra

COMPFEST 9

Penyisihan SCPC

Terima kasih juga pada para tester:

- Agus Sentosa Hermawan
- Ammar Fathin Sabili

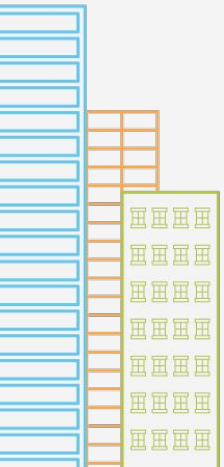


COMPFEST 9

Catatan

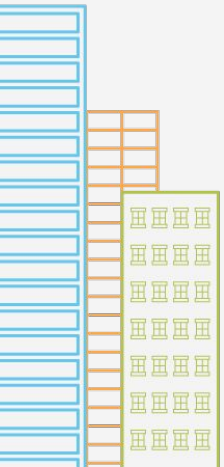
Seluruh statistik pengumpulan pada pembahasan ini mengacu pada pengumpulan sebelum *scoreboard* dibekukan. Kalaupun suatu tim sudah *Accepted* di suatu soal, seandainya masih submit di soal tersebut, submission pasca AC tetap masuk statistik ini.

Pembahasan soal diurutkan dari ekspektasi kesulitan masing-masing soal, dari termudah ke tersulit.



Restoran Chanek

Author: Norman Bintang
Tag: Ad hoc
First Solve: Panitia Compfest Reborn (menit ke-3)



Jumlah AC:	116
Jumlah WA:	206
Jumlah TLE:	6
Jumlah RTE:	35
Jumlah CE:	13

COMPFEST 9

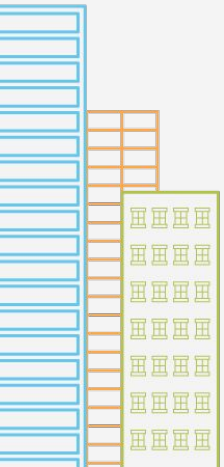
Restoran Chanek (2)

Karena banyaknya tempat masak hanya ada 1, maka setiap koki harus berganti-gantian memasak. Sehingga, total waktu yang dibutuhkan adalah jumlah dari seluruh waktu memasak untuk tiap jenis makanan. Tentunya, urutan memasak tidak ada kaitannya dengan hasil akhir.

Kompleksitas: O(NM)

Palindromisme

Author: M. Ayaz Dzulfikar
Tag: Ad hoc
First Solve: Lodnob Gnale
(menit ke-5)



Jumlah AC:	84
Jumlah WA:	134
Jumlah TLE:	22
Jumlah RTE:	17
Jumlah CE:	24

COMPFEST 9

Palindromisme (2)

Misal S' adalah string hasil akhir.

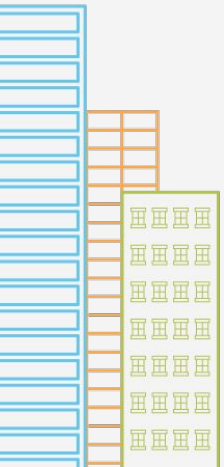
Perhatikan bahwa untuk setiap i ($1 \le i < N$), substring $S'_{i..i+1}$ juga harus merupakan palindrom. Artinya, $S'_i = S'_{i+1}$ untuk setiap i. Lebih lanjut lagi, ternyata $S'_1 = S'_2 = \ldots = S'_N$. Hal ini berarti, S' akan terdiri dari satu jenis karakter.

Ya, versi sederhana dari soal ini sebenarnya "diberikan suatu string, cari pengubahan karakter minimum agar string hasil akhir terdiri dari satu jenis karakter saja!"

Palindromisme (3)

Sehingga, kita dapat mencoba setiap karakter dari ‘a’ sampai ‘z’, dan cari total pengubahan minimum yang mungkin.

Kompleksitas: O(N)

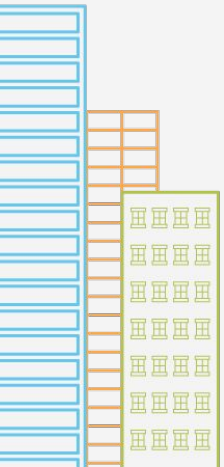


Palindromisme (4)

Catatan:

Jika Anda senang dibohongi seperti pada soal ini, Anda bisa mencoba seri pertama soal bohong-bohongan Author pada tautan berikut:

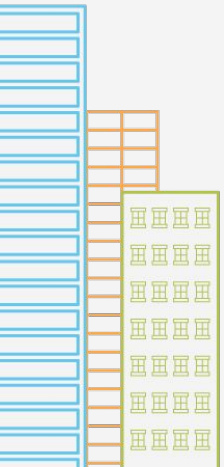
Ramduel



Iri Itu Tidak Baik

Author: Usama
Tag: Greedy
First Solve: Ainge WF
(menit ke-5)

Jumlah AC:	79
Jumlah WA:	222
Jumlah TLE:	65
Jumlah RTE:	78
Jumlah CE:	11



COMPFEST 9

Iri Itu Tidak Baik (2)

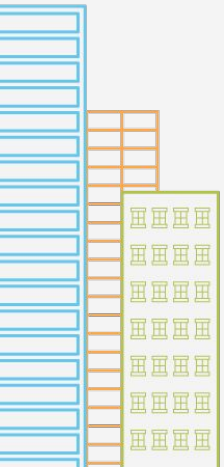
Terdapat 2 observasi yang dapat dibuat:

1. Terdapat solusi optimal yang membuang tepat K orang
2. Misal orang-orang tersebut diurutkan berdasarkan gajinya. Terdapat solusi optimal yang menggunakan N-K orang dengan nilai berurutan. Hal ini karena jika tidak, kita dapat menukar yang dibuang dengan yang tidak dibuang, sedemikian sehingga hasilnya tidak kurang optimal

Iri Itu Tidak Baik (3)

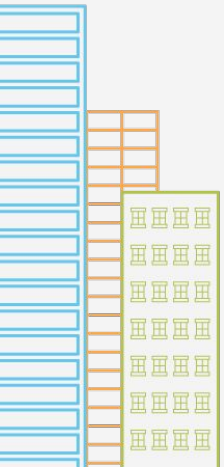
Maka, kita dapat menyelesaikan soal ini dengan mengurutkan orang-orang tersebut berdasarkan gajinya, lalu mencari nilai iri minimum dari setiap subarray dengan ukuran N-K.

Kompleksitas: O(N log N)



Epal Berharga

Author: Ammar Fathin S.
Tag: Brute Force
First Solve: Bersaw
(menit ke-10)



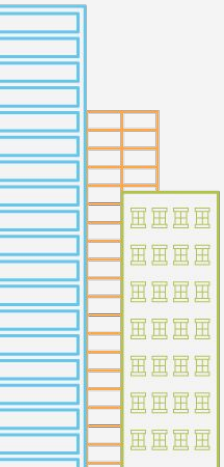
Jumlah AC:	27
Jumlah WA:	284
Jumlah TLE:	34
Jumlah RTE:	30
Jumlah CE:	7

COMPFEST 9

Epal Berharga (2)

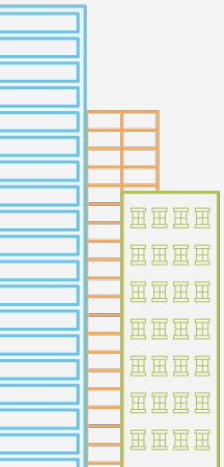
Misal banyak epal minimum yang dimiliki seseorang didefinisikan sebagai mins.

Observasi penting adalah, jawaban yang mungkin bagi soal ini adalah mins, mins+1, atau mins+2. Hal ini karena seserang maksimal mendapatkan 2 epal, masing-masing satu dari orang di kiri dan kanannya.



Epal Berharga (3)

Sehingga, kita bisa memeriksa untuk masing-masing nilai, apakah nilai tersebut bisa menjadi jawaban. Hal ini dapat dilakukan dengan sedikit *greedy*.



Epal Berharga (4)

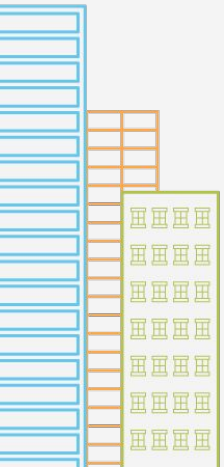
Bagaimana *greedy*-nya? Misal kita ingin memeriksa apakah jawabannya >= x, untuk suatu x. Maka, kita dapat mengiterasi dari 1 sampai N-1, dan ada 2 kemungkinan:

1. Jika epal yang sekarang dipegang orang ke-i lebih dari x, dan dia tidak memberi ke orang ke-(i-1), pasti optimal untuk memberikan satu epal ke orang ke-(i+1)
2. Jika epal yang sekarang dipegang orang ke-i kurang dari x, pasti optimalnya ia mendapat satu epal dari orang ke-(i+1)

Epal Berharga (5)

Di akhir, kita cukup memeriksa apakah epal yang dipegang setiap orang >= x atau tidak. Kompleksitas untuk seluruh langkah pengecekan itu adalah O(N).

Kompleksitas: O(N)



Epal Berharga (6)

Trivia:

Judul asli soal ini, “Epal Babi”, terinspirasi dari suatu video parodi *Snow White* di Youtube.

Kucing Warna-Warni

Author: Muhammad Ayaz D.
Tag: Constructive, Greedy
First Solve: Andi-chan
(menit ke-131)

Jumlah AC:	6
Jumlah WA:	21
Jumlah TLE:	7
Jumlah RTE:	6
Jumlah CE:	1

COMPFEST 9

Kucing Warna-Warni (2)

Pertama-tama, mari kita tentukan batas bawah dan batas atas K yang valid.

Batas bawah jelas adalah N, ketika warna setiap kucing sama.

Batas atas adalah jumlah min(M, i) untuk setiap $1 \le i \le N$. Apa sebenarnya makna min(M, i) ini? Mudahnya, ini adalah banyak maksimum subarray kandang yang warna kucingnya berbeda, yang mana kandang terkanannya berada di i.

Kucing Warna-Warni (3)

Dapat dibuktikan bahwa K berada di antara batas bawah dan batas atas merupakan syarat cukup agar terdapat solusi.

Terdapat berbagai macam solusi untuk soal ini. Solusi yang juri pakai akan dijelaskan pada halaman selanjutnya, yang konstruksinya membuktikan pernyataan di atas.

Kucing Warna-Warni (4)

1. Iterasi i, dari 1 sampai N
2. Apabila min(M, i) + N - i < K, ke langkah 3. Jika tidak, ke langkah 4
3. Isi kandang ke-i dengan kucing warna (i mod m) + 1. Akibatnya, K berkurang sebesar min(M, i). Kembali ke langkah 2

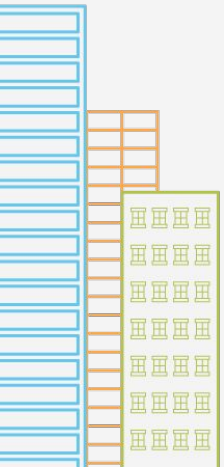
Kucing Warna-Warni (5)

4. Kita dapat mengisi N-i kandang sisanya, sehingga N-i kandang tersebut masing-masing akan berkontribusi 1 dalam mengurangi K. Isi kandang ke-i dengan kucing yang warnanya menyebabkan kandang ke-i dapat “dipanjangkan” ke kiri sebanyak K-(N-i), lalu isi semua kandang setelah i dengan kucing berwarna sama dengan kucing di kandang ke-i. Pengisian kucing berhenti sampai sini.

Kucing Warna-Warni (6)

Perhatikan bahwa konstruksi di halaman-halaman sebelumnya pasti menemukan solusi untuk setiap K yang berada di antara batas bawah dan batas atas.

Sehingga, kompleksitas akhir untuk menyelesaikan soal ini adalah O(N).



Mencari Waifu

Author: M. Ayaz Dzulfikar
Tag: Math, Geometry, Brute Force
First Solve: Andi-chan
(menit ke-33)

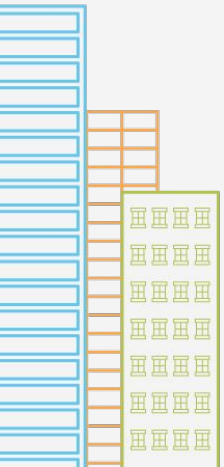
Jumlah AC:	16
Jumlah WA:	32
Jumlah TLE:	20
Jumlah RTE:	16
Jumlah CE:	0

COMPFEST 9

Mencari Waifu (2)

Tentunya, solusi naif mencoba 2001^2 koordinat, lalu mengecek dalam N^2 tidak cukup cepat.

Observasi apa yang bisa didapat dari soal?



Mencari Waifu (3)

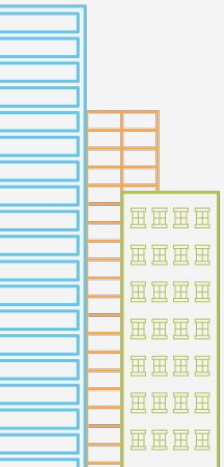
Definisikan MAX_VALUE sebagai 2000.

Pertama, walaupun N bernilai cukup besar, batas koordinat x dan y cukup kecil, yaitu [0, MAX_VALUE].

Selanjutnya, penghitungan jarak manhattan sebenarnya dapat dilakukan terpisah, yakni kita menghitung jumlah selisih pada x, lalu kita jumlahkan dengan jumlah selisih pada y. Cara ini tentunya akan memberikan hasil yang sama.

Mencari Waifu (4)

Mari kita fokus dulu pada penghitungan jumlah jarak manhattan tiap koordinat pada masukan. Pertama, kita selesaikan untuk sumbu x, dan sumbu y dapat diselesaikan dengan cara serupa.



Mencari Waifu (5)

Definisikan $cnt_x[z]$ sebagai banyaknya titik dengan koordinat x bernilai z, dan $cnt_y[z]$ sebagai banyaknya titik dengan koordinat y bernilai z. Maka, jumlah selisih pada x dan y masing-masing dapat dihitung sebagai berikut:

$$sum_x = \sum_{i=0}^{2000} \sum_{j=i}^{2000} cnt_x[i] \times cnt_x[j] \times (j-i)$$

$$sum_y = \sum_{i=0}^{2000} \sum_{j=i}^{2000} cnt_y[i] \times cnt_y[j] \times (j-i)$$

Mencari Waifu (6)

Selanjutnya, kita definisikan $sum\_dist_x(z)$ sebagai jumlah selisih z dengan seluruh koordinat x, dan $sum\_dist_y(z)$ untuk seluruh koordinat y. Maka, kedua fungsi tersebut dapat dihitung sebagai berikut:

$$sum\_dist_x(z) = \sum_{i=0}^{2000} cnt_x[i] \times |(i - z)|$$

$$sum\_dist_y(z) = \sum_{i=0}^{2000} cnt_y[i] \times |(i - z)|$$

Mencari Waifu (7)

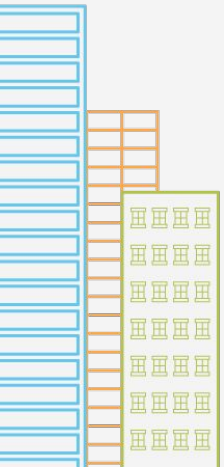
Sekarang, yang kita inginkan adalah mencari titik (a, b) sehingga persamaan di bawah ini berlaku:

$$sum_x + sum_y + sum\_dist_x(a) + sum\_dist_y(b) = K$$

Apabila kita sudah menghitung nilai masing-masing fungsi dalam O(MAX_VALUE^2) secara terpisah, lalu kita tinggal memanfaatkan nilai-nilai tersebut dalam mencari solusi persamaan tersebut.

Mencari Waifu (8)

Maka, kita dapat menyelesaikan soal ini dengan kompleksitas $O(N + MAX\_VALUE^2)$, cukup untuk mendapatkan Accepted.



COMPFEST 9

Mencari Waifu (9)

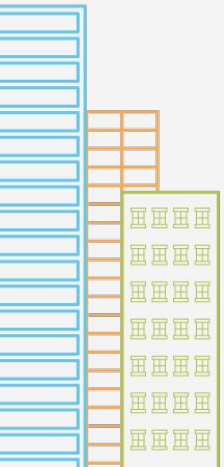
Catatan:

Berikut adalah batasan asli soal sebelum “dipermudah”:

- $1 \le N \le 50.000$
- $0 \le X_i \le 100.000$
- $0 \le Y_i \le 100.000$
- $0 \le K \le 10^{15}$
- Waifu boleh berada di koordinat bilangan bulat manapun

Makan Malam Chanek

Author: M. Ayaz Dzulfikar
Tag: Dynamic Programming
First Solve: Bersaw
(menit ke-30)



Jumlah AC:	14
Jumlah WA:	36
Jumlah TLE:	18
Jumlah RTE:	7
Jumlah CE:	0

COMPFEST 9

Makan Malam Chanek (2)

Pertama-tama, definisikan dist(a, b) sebagai penggeseran minimum apabila seseorang sedang menghadap makanan a, dan ingin menghadap makanan b.

Dapat dilihat bahwa dist(a,b) = min(|a - b|, M - |a - b|).

Makan Malam Chanek (3)

Salah satu solusi *Dynamic Programming* (DP) yang mungkin terbayang adalah dengan state:
$[idx_1][idx_2][pos_1][pos_2]$

Dengan keterangan:

- idx_z = urutan makanan yang ingin dimakan orang z
- pos_z = makanan yang sekarang di hadapan orang z

Untuk transisi, kita coba yang minimal dari menggerakkan agar Pak Chanek makan, atau Dzul makan.

Makan Malam Chanek (4)

DP tersebut memiliki kompleksitas $O(N^2M^2)$, yang tentunya terlalu besar.

Setelah dipikir lagi, state pos_2 sebenarnya dapat diambil dari pos_1 sesuai definisi y pada halaman sebelumnya. Maka, kita bisa membuang state pos_2, sehingga kompleksitas sekarang $O(N^2M)$, yang masih terlalu besar

Makan Malam Chanek (5)

Apabila dipikir lagi, perhatikan bahwa hanya ada 2 kemungkinan pos_1:

1. Jika yang terakhir makan adalah Pak Chanek, pos_1 bernilai A_{idx_1-1}
2. Jika yang terakhir makan adalah Dzul, pos_1 bernilai $(B_{idx\_2-1} + M/2) \bmod M$

Makan Malam Chanek (6)

Sehingga, kita dapat membuang state pos_1, dan cukup mencatat siapa yang terakhir makan. Maka, sekarang state DP-nya adalah:
$[idx_1][idx_2][last]$

Dengan last hanya memiliki dua kemungkinan nilai. Sehingga, kompleksitasnya $O(N^2)$, baik dalam waktu maupun memori.

Makan Malam Chanek (7)

Trivia:

Solusi *greedy* yang selalu membuat orang dengan dist() minimal ke makanan yang dia inginkan yang didahulukan bisa lolos contoh masukan. Hal ini sesungguhnya kebetulan belaka.

Array yang Hilang

Author: Windi Chandra
Tag: Math, Combinatorics
First Solve: Andi-chan
(menit ke-58)

Jumlah AC:	11
Jumlah WA:	37
Jumlah TLE:	27
Jumlah RTE:	25
Jumlah CE:	3

COMPFEST 9

Array yang Hilang (2)

Misal faktorisasi prima dari perkalian isi array A dapat dinyatakan sebagai

$$p_1^{n_1} \times p_2^{n_2} \times \ldots \times p_k^{n_k}$$

Array yang Hilang (3)

Banyak cara mendistribusikan faktor prima ke-i ke tepat N indeks, dengan setiap indeks mendapatkan non-negatif bagian bisa didapat dengan rumus kombinasi sederhana, yaitu

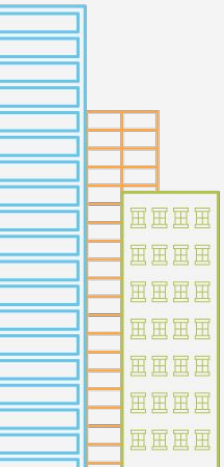
$$C(N + n_i - 1, n_i)$$

C(n, k) menyatakan n kombinasi k

Array yang Hilang (4)

Tentunya, pendistribusian ini bersifat independen antar faktor prima berbeda.

Selanjutnya, berdasarkan aturan perkalian, banyaknya array X yang memenuhi permintaan soal ada sebanyak hasil kali pendistribusian pada halaman sebelumnya, untuk setiap faktor prima berbeda.



Array yang Hilang (5)

Permasalahan selanjutnya adalah menghitung kombinasinya, karena parameternya bisa sangat besar.

Karena modulo-nya prima, 10^9+7, kita bisa memanfaatkan *modular multiplicative inverse* untuk mencari nilai kombinasi.

Array yang Hilang (6)

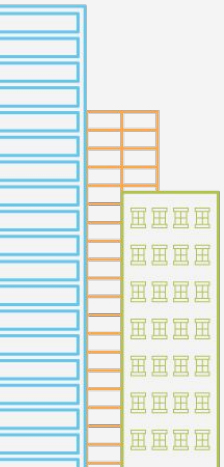
Lakukan prekomputasi faktorial, sehingga membantu penghitungan kombinasi.

Untuk faktorisasi prima, dapat menggunakan algoritma $O(\text{sqrt}(A_i))$ atau prekomputasi menggunakan algoritma sieve dalam $O(A_i \log A_i)$, lalu faktorisasi dapat dilakukan dalam $O(\log A_i)$

Array yang Hilang (7)

Hati-hati dalam prekomputasi faktorial, karena yang dibutuhkan bisa mencapai $N * \log A_i$, sekitar 1.7 juta

Kompleksitas: $O(N \log A_i + A_i \log MOD)$ atau
$O(N \sqrt{A_i} + A_i \log MOD)$



COMPFEST 9

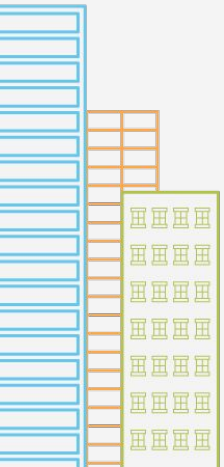
Array yang Hilang (8)

Catatan:

Ada banyak implementasi berbeda untuk soal ini. Ada juga implementasi dengan kompleksitas O(N log A_i + log MOD) yang menghitung *inverse modulo* di akhir. Silahkan bereksperimen untuk mendapatkan kompleksitas yang aneh-aneh dan lucu.

Faktorunesia

Author: M. Ayaz Dzulfikar
Tag: Graph, Greedy
First Solve: Ainge WF
(menit ke-145)

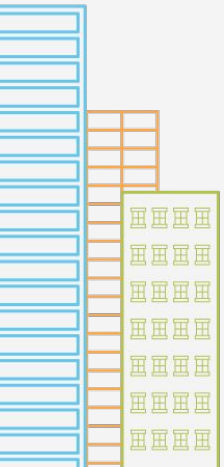


Jumlah AC:	2
Jumlah WA:	15
Jumlah TLE:	12
Jumlah RTE:	21
Jumlah CE:	2

COMPFEST 9

Faktorunesia (2)

Untuk mempermudah, definisikan MAXC sebagai nilai maksimum C_i pada masukan.



Faktorunesia (3)

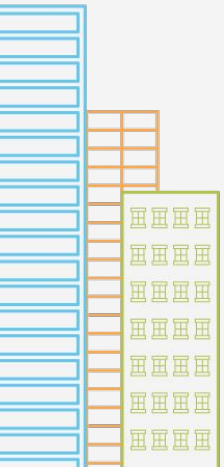
Bagaimana kasus ketika jawaban -1?

Hanya ada 2 hal yang menyebabkan jawaban -1, yaitu:

1. Tidak ada jalan dengan nilai $C_i = 0$ dari kota ke-1
2. Kota ke-1 tidak terhubung ke kota ke-N, melalui jalan-jalan yang ada (grafnya *disconnected*)

Faktorunesia (4)

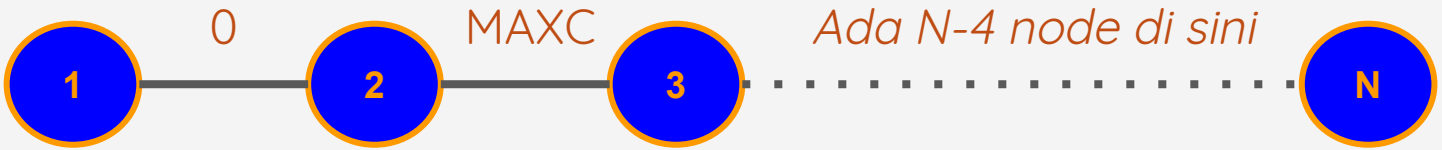
Selanjutnya, berapa langkah minimum yang maksimum, untuk mencapai kota ke-N, untuk suatu graf dengan batasan di soal?



COMPFEST 9

Faktorunesia (5)

Perhatikan graf garis “menyebalkan” di bawah

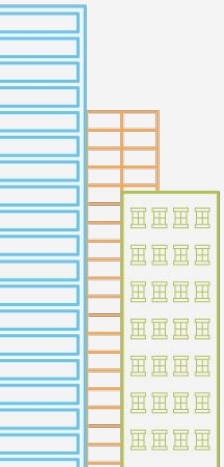


Kita harus berputar-putar di kota ke-1 dan kota ke-2 hingga akhirnya bisa melewati jalan di kota ke-2 dan kota ke-3. Artinya, dibutuhkan O(N + MAXC) langkah untuk mencapai kota ke-N.

Faktorunesia (6)

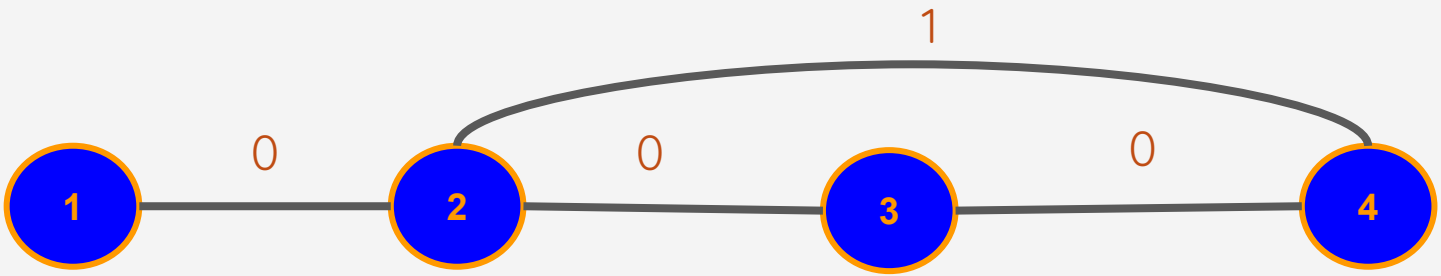
Jika melihat fungsi yang menyatakan panjang jalan, fungsi tersebut seperti basis.

Sehingga, kita bisa membuat observasi bahwa kita harus meminimumkan banyak langkah, karena semakin banyak, panjang jalan akan semakin besar.



Faktorunesia (7)

Sayangnya, itu kurang tepat. Perhatikan graf di bawah



Observasi sebelumnya akan mendapatkan jawabannya 2 dengan rute 1-2-4, padahal jawabannya 0, dengan rute 1-2-3-4.

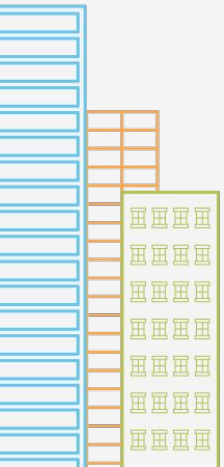
Faktorunesia (8)

Observasinya sedikit rusak karena mungkin adanya *leading zero*. Akan tetapi, bisa dibuktikan bahwa banyaknya langkah pada jawaban optimal dibatasi oleh O(N + MAXC).

Kurang lebih: apabila sudah mencapai MAXC langkah, kita sudah bebas bisa mencapai kota mana saja, dan langkah minimumnya O(N).

Faktorunesia (9)

Ternyata, untuk memperbaiki observasi tersebut cukup mudah.



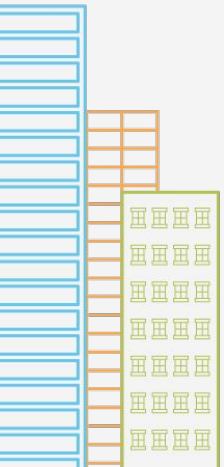
Faktorunesia (10)

Buat suatu *connected component* yang mengandung kota ke-N, memanfaatkan jalan-jalan dengan nilai 0.

Apabila kita sudah mencapai kota-kota tersebut, tentunya kita cukup menggunakan jalan dengan nilai 0 hingga mencapai kota ke-N.

Faktorunesia (11)

Maka, sebenarnya yang perlu kita minimumkan adalah langkah menuju kota-kota pada *connected component* tersebut. Mencarinya dapat menggunakan BFS/DFS sederhana.



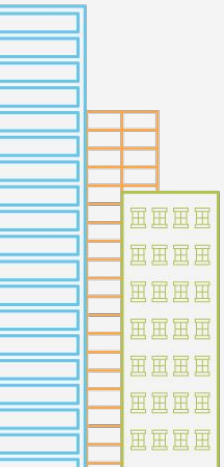
Faktorunesia (12)

Definisikan fungsi check(x, y) sebagai fungsi boolean, yang akan mengembalikan true jika kita bisa mencapai kota ke-x pada langkah ke-y. Fungsi ini dapat dikomputasi juga memanfaatkan BFS/DFS, agar ketika dipanggil bisa O(1).

Faktorunesia (13)

Apa yang perlu kita lakukan selanjutnya?

Dari *connected component* tadi, cari langkah minimum yang diperlukan untuk mencapai komponen itu. Ini bisa memanfaatkan fungsi check().

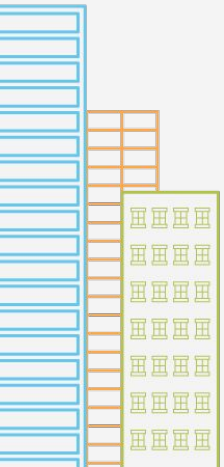


COMPFEST 9

Faktorunesia (14)

Misal langkah minimum itu adalah mins.

Kita iterasi menurun, dari mins hingga 1, untuk mendapatkan nilai-nilai jalan yang kita pakai pada langkah-langkah tersebut.



Faktorunesia (15)

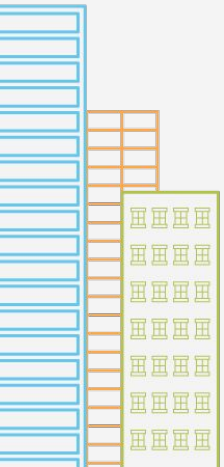
Untuk setiap iterasi, kita pelihara (simpan) kota-kota yang mungkin saja dipakai pada langkah ke-iterasi tersebut.

Kemudian, dari kota-kota tersebut, cari jalan dengan nilai minimum, yang mungkin saja dipakai pada langkah ke-iterasi berikut, serta valid. Dengan kata lain, pada iterasi ke-y, cari jalan dengan nilai kurang dari y dan terhubung dengan kota x, serta check(x, y-1) bernilai true.

Faktorunesia (16)

Lalu, perbarui kota-kota yang tadi dipelihara dengan kota-kota yang terhubung dengan jalan dengan nilai minimum dan memenuhi syarat pada halaman sebelumnya.

Kenapa kita perlu memelihara kota-kota itu? Karena kita sebenarnya belum tahu, dari kota mana kita akan menemukan solusi optimal.



Faktorunesia (17)

Lakukan terus menerus, dan pada akhirnya kita tahu nilai jalan-jalan yang kita pakai pada solusi optimal. Menghitung panjang jalan cukup mudah.

Walaupun menghabiskan paling banyak halaman di pembahasan ini, implementasinya sebenarnya tidak terlalu sulit.

Kompleksitas: O(M * (N + MAXC))

Dua Sejoli

Author: M. Ayaz Dzulfikar
Tag: Graph, Lowest Common Ancestor
First Solve: Ainge WF
(menit ke-177)

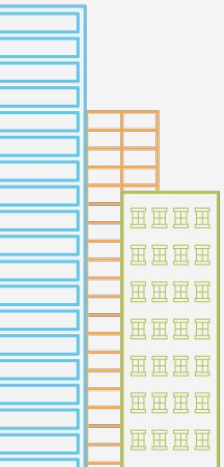
Jumlah AC:	4
Jumlah WA:	31
Jumlah TLE:	7
Jumlah RTE:	7
Jumlah CE:	1

COMPFEST 9

Dua Sejoli (2)

Sebelum masuk ke pembahasan sebenarnya, tahukah Anda bahwa ada 3 kemungkinan seseorang menuliskan 10^9+7 pada *source code* yang ia buat untuk soal ini?

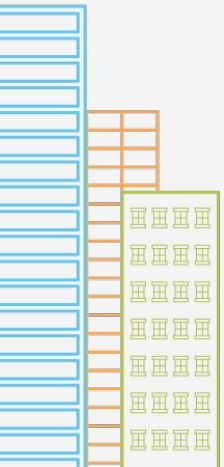
1. Tahu kebenaran, tapi lupa menghapus
2. Tahu solusi, tapi salah hitung
3. Tidak tahu solusi sama sekali



COMPFEST 9

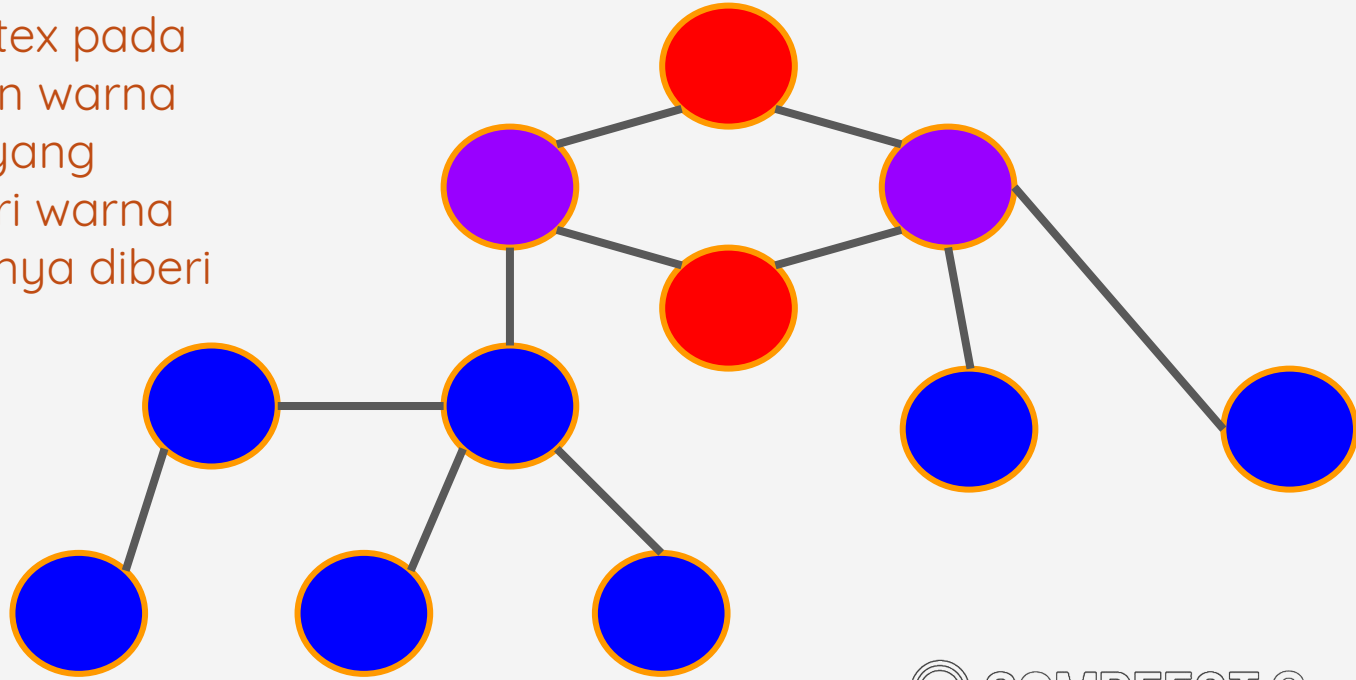
Dua Sejoli (3)

Hanya ada satu kemungkinan bagi graf terhubung dengan N vertex dan N edge, yaitu suatu graf yang dapat dilihat sebagai *cycle* yang “digantungi” beberapa *tree*.



Dua Sejoli (4)

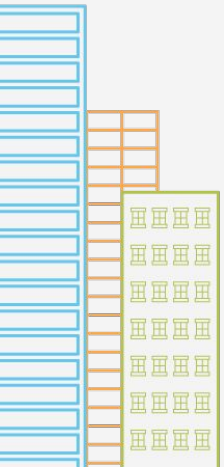
Contoh graf-nya. Vertex pada *cycle* ditandai dengan warna merah, dan *tree* yang "menggantung" diberi warna biru. Campuran keduanya diberi warna ungu.



Dua Sejoli (5)

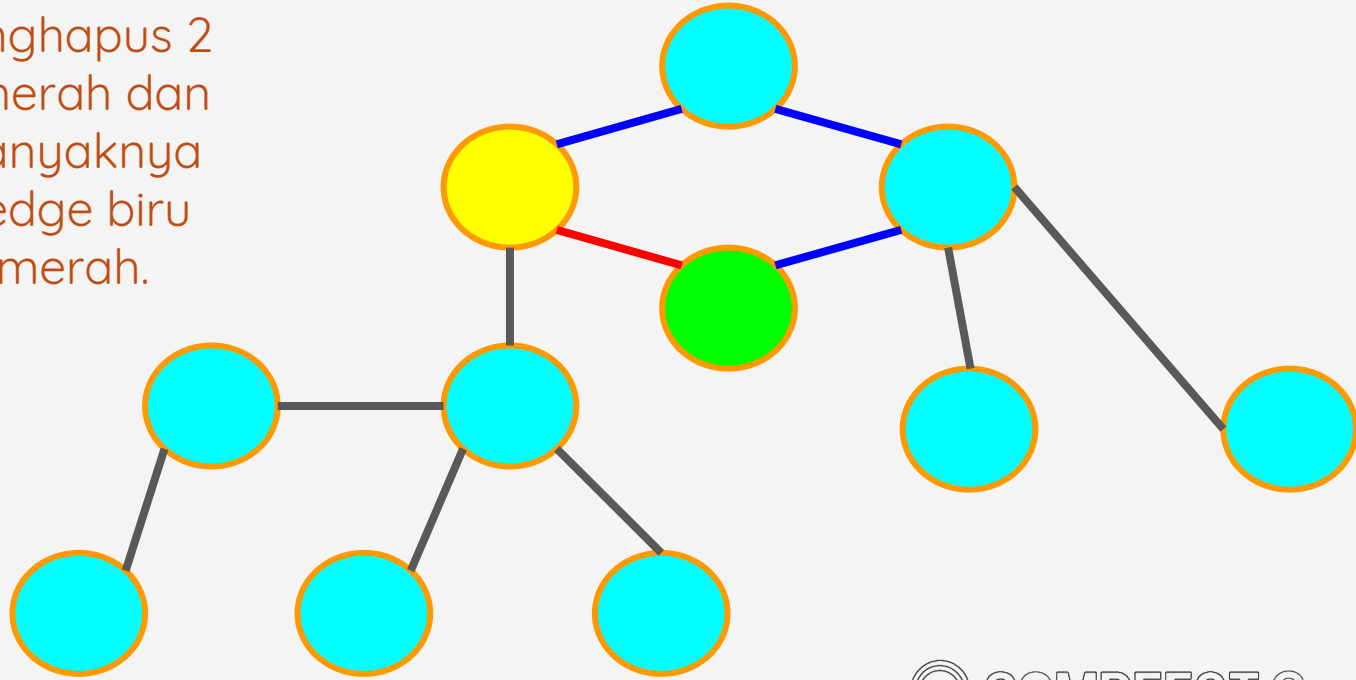
Terdapat 4 kasus yang harus ditangani, yaitu:

1. Kedua vertex berada pada *cycle*
2. Salah satu vertex berada pada *cycle*
3. Kedua vertex tidak di *cycle*, namun pada *tree* berbeda
4. Kedua vertex tidak di *cycle*, dan berada di *tree* yang sama



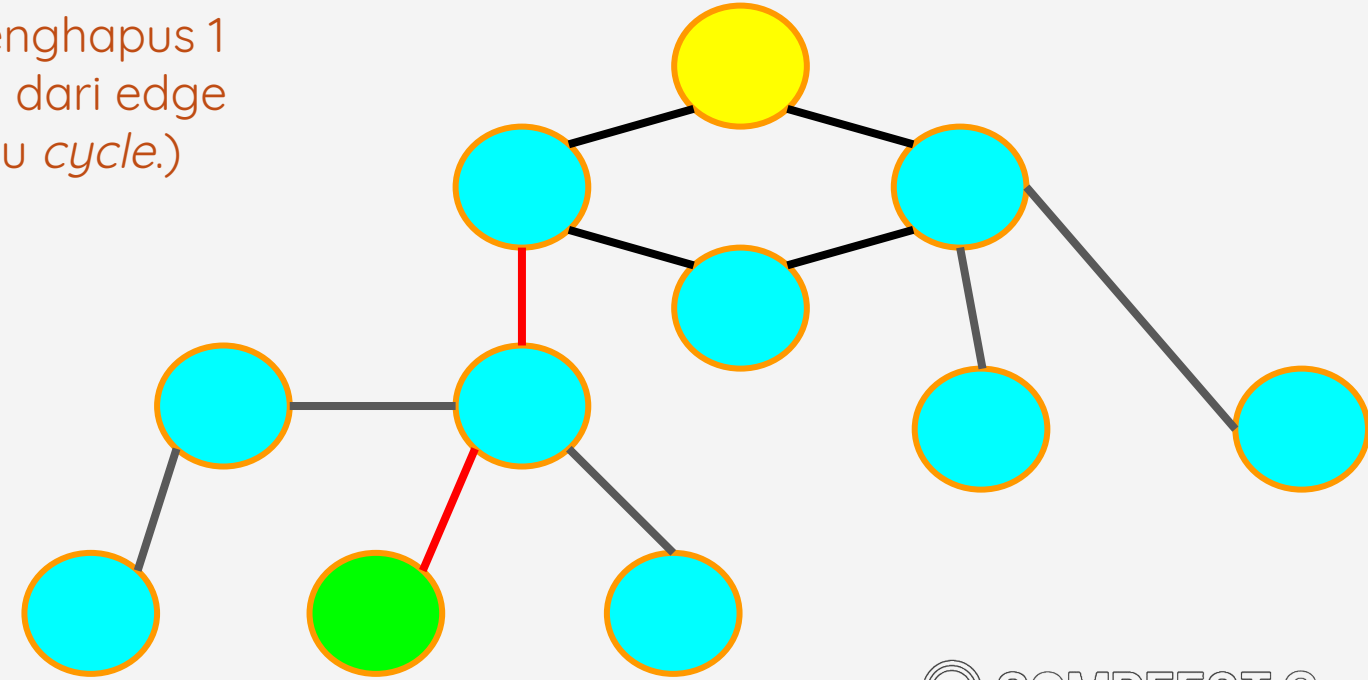
Dua Sejoli (6)

Kasus 1: Kita perlu menghapus 2 edge, satu dari edge merah dan satu dari edge biru. Banyaknya cara adalah banyak edge biru dikali banyak edge merah.



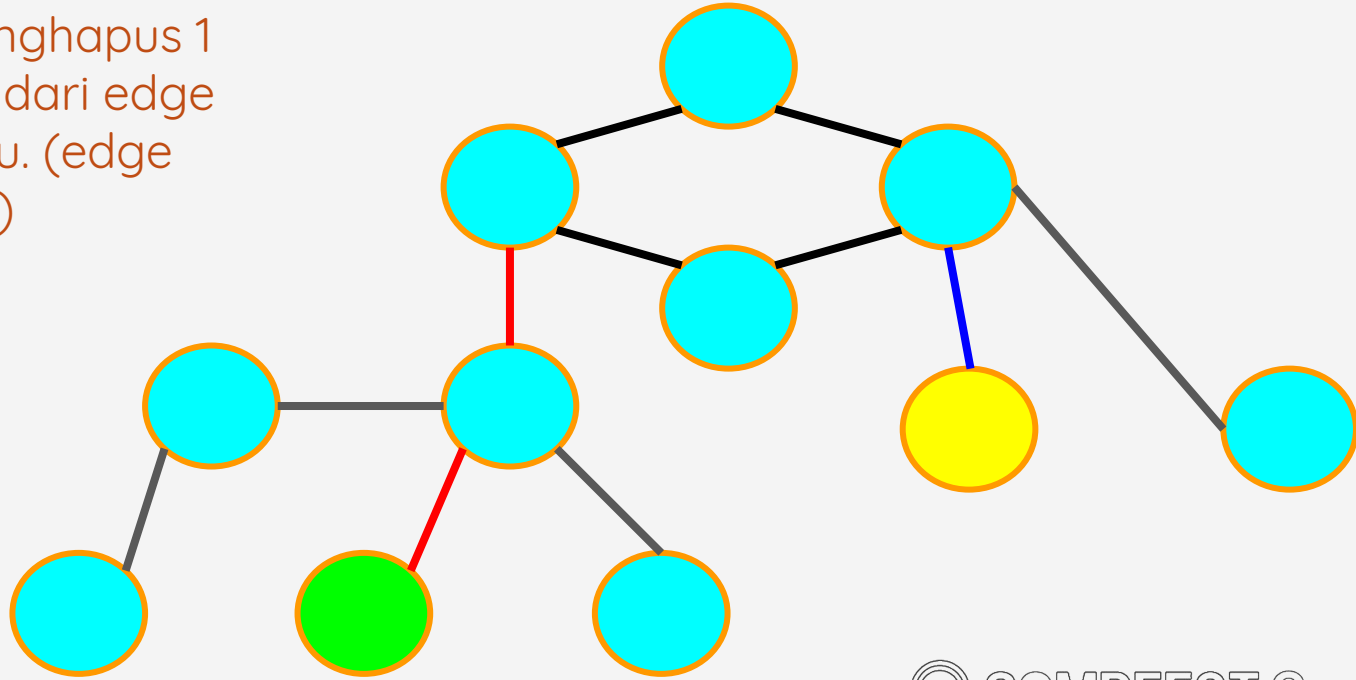
Dua Sejoli (7)

Kasus 2: Kita perlu menghapus 1 edge, yakni salah satu dari edge merah. (edge menuju *cycle*.)



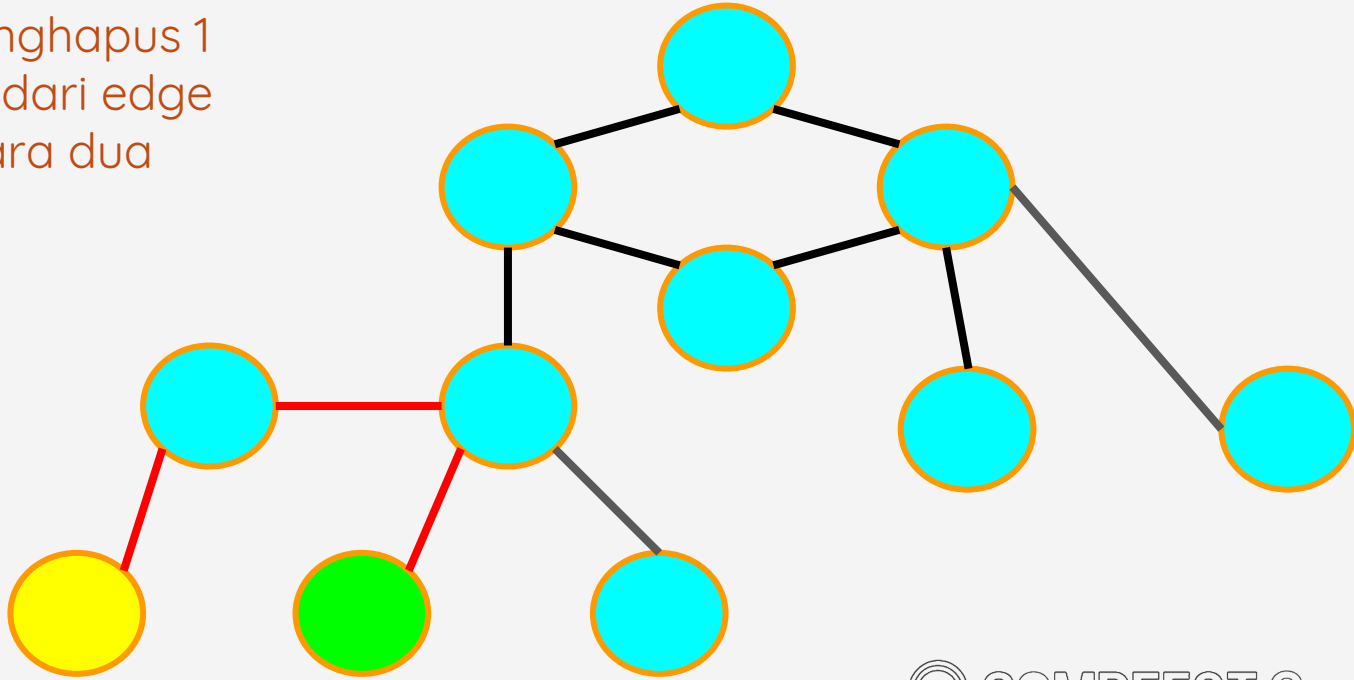
Dua Sejoli (8)

Kasus 3: Kita perlu menghapus 1 edge, yakni salah satu dari edge merah atau edge biru. (edge menuju *cycle*.)



Dua Sejoli (9)

Kasus 4: Kita perlu menghapus 1 edge, yakni salah satu dari edge merah. (path di antara dua vertex.)



Dua Sejoli (10)

Untuk menyelesaikan seluruh kasus, terdapat beberapa langkah yang perlu dilakukan:

1. Mencari *cycle*, lalu membuat urutan vertex pada *cycle*, agar dapat menyelesaikan kasus 1 dengan cepat
2. Mencatat "kedalaman" setiap vertex pada *tree*, untuk menyelesaikan kasus 2 dan 3
3. Membuat struktur data untuk menyelesaikan kasus 4, yang membutuhkan *lowest common ancestor*

Dua Sejoli (11)

Sebagai catatan, ada cara sehingga kita cukup membagi 2 kasus, yakni ketika kedua vertex berada di *cycle* atau tidak. Hal ini dapat dilakukan dengan meng-kontraksikan *cycle* menjadi satu vertex.

Dua Sejoli (12)

Perhatikan bahwa maksimum banyaknya cara adalah 25.000^2, yang kurang dari 10^9+7. Secara teknis, modulo pada soal adalah pembohongan publik.

Dalam implementasi, harus hati-hati karena membutuhkan ketelitian. Perhatikan juga bahwa bisa saja ada multi-edge, seperti pada N = 2.

Kompleksitas: O(N log N + Q log N)