Pembahasan OSN Informatika 2017

Dzulfikar, Muhammad Ayaz
muhammad.ayaz97@gmail.com

Arsadjaja, Alfonsus Raditya
raditya1710@gmail.com

Sabili, Ammar Fathin
athin2008@gmail.com

Gunawan, Jonathan Irvin
jonathanirvingunawan@gmail.com

Johanna, Stacia Edina
edinastacia@gmail.com

December 31, 2017

Di bawah ini adalah solusi resmi yang digunakan oleh Scientific Committee OSN Informatika 2017. Perhatikan bahwa mungkin saja terdapat lebih dari satu solusi untuk beberapa subsoal. Perhatikan juga bahwa subsoal pada diskusi ini mungkin saja tidak terurut untuk kemudahan diskusi.

Karena tujuan utama editorial ini adalah untuk memberikan ide umum untuk menyelesaikan setiap subsoal, kami melewatkan beberapa detil (termasuk detil implementasi) pada diskusi ini untuk latihan para pembaca.

1 1A : Pertahanan Pekanbaru

Soal ini ditulis oleh Alham Fikri Aji.

1.1 Subsoal 3

Subsoal ini dapat diselesaikan menggunakan *complete search*. Salah satu cara yang bisa digunakan adalah menggunakan rekursi; pada setiap lantai, coba semua kemungkinan pilihan yang tersedia, yaitu kabur melewati penjaga atau mengalahkan penjaga, lalu cari jumlah waktu minimal. Kompleksitas dari solusi ini adalah $O(2^N)$.

1.2 Subsoal 4

Subsoal ini dapat diselesaikan menggunakan dynamic programming (DP). State dari DP tersebut adalah (posisi lantai, banyak lantai yang mana penjaganya sudah dikalahkan). Untuk mengisi nilai dari DP, cari hasil minimum dari kedua kemungkinan pilihan untuk setiap lantai. Kompleksitas dari solusi ini adalah $O(N^2)$.

1.3 Subsoal 5

Karena stamina dari Pak Dengklek, sang makhluk jahat, dan para penjaga tidak perlu diperhatikan, maka untuk setiap lantai, pilih nilai yang paling minimum di antara $L[i]$ dan $K[i]$. Kompleksitas dari solusi ini adalah $O(N)$.

1.4 Subsoal 6

Subsoal ini dapat diselesaikan dengan bantuan *sorting*. Untuk setiap lantai ke-i, jika nilai $L[i] \leq K[i]$, maka $L[i]$ pasti dipilih. Selain itu, anggap awalnya kita memilih $K[i]$. Misal S'_d menyatakan stamina Pak Dengklek saat ini. Jika saat selesai $S'_d < S_m$, maka kita harus mengganti beberapa $K[i]$ dengan $L[i]$. Dalam memilih yang diganti, prioritaskan yang memiliki nilai $K[i] - L[i]$ paling besar. Hal ini bisa dicapai dengan bantuan *sorting*. Selanjutnya, kurangi total waktu dengan dengan $S_m - S'_d$ nilai $K[i] - L[i]$ terbesar (bisa dilihat bahwa ini sama dengan menukar $K[i]$ dengan $L[i]$). Kompleksitas dari solusi ini adalah $O(NlogN)$.

1.5 Subsoal 7

Subsoal ini dapat diselesaikan dengan bantuan *priority queue*. Solusi untuk subsoal ini tidak jauh berbeda dengan solusi subsoal 6, namun simpan $K[i] - L[i]$ dalam *priority queue*. Untuk setiap lantai ke-i, selama $S'_d < P[i]$, maka cari nilai terbesar dari *priority queue* tersebut dan kurangi total waktu dengan nilai tersebut. Lakukan hal yang sama selama S'_d saat Pak Dengklek berada di lantai N masih lebih kecil dari S_m. Kompleksitas dari solusi ini adalah $O(NlogN)$.

2 1B : Menukar Kotak

Soal ini ditulis oleh Jonathan Irvin Gunawan.

Untuk kemudahan berdiskusi, mari kita sebut kandang-kandang yang berisi bebek yang berulang tahun sebagai kandang spesial, sedangkan kotak-kotak permen yang pada akhirnya diletakkan pada kandang spesial sebagai kotak spesial. Mari kita juga sebut M sebagai banyaknya kandang spesial.

2.1 Subsoal 3

Subsoal ini dapat diselesaikan menggunakan *complete search*. Untuk setiap penukaran, kita mencoba pasangan kotak mana yang ingin kita tukar. Kompleksitas dari solusi ini adalah $O((N-1)^K)$.

2.2 Subsoal 4

Subsoal ini dapat diselesaikan menggunakan *complete search* yang lebih pintar. Kita dapat mencoba subhimpunan kotak-kotak mana saja yang menjadi kotak spesial. Tentu saja, banyaknya kotak pada setiap subhimpunan ini harus sama dengan banyaknya kandang spesial. Untuk setiap subhimpunan kotak, kita dapat menghitung banyaknya penukaran minimum yang dibutuhkan agar pada akhir penukaran, kotak-kotak pada subhimpunan ini terletak pada kandang spesial.

Penghitungan ini dapat dilakukan dengan menggunakan algoritma *greedy*: jika A_i adalah kandang spesial ke-i (jika diurutkan dari kiri), dan B_i adalah kotak spesial ke-i (jika diurutkan dari kiri menggunakan posisi awal), maka pada akhirnya, kita ingin meletakkan kotak B_i ke kandang A_i. Dengan kata lain, pada saat proses penukaran, kita tidak akan menukarkan dua kotak spesial. Jika banyaknya penukaran tidak lebih dari K, maka subhimpunan tersebut dikatakan sah. Dari seluruh subhimpunan kotak yang sah, kita dapat mencari subhimpunan kotak yang mana jumlah permen di dalam nya semaksimal mungkin. Kompleksitas dari solusi ini adalah $O(2^N \times N)$.

2.3 Subsoal 5

Subsoal ini dapat diselesaikan menggunakan dynamic programming (DP).

Mari definisikan sebuah fungsi $f(a, b, c)$ sebagai berikut: asumsikan kita hanya memiliki kotak pertama sampai kotak ke-a, dan kandang-kandang yang berisi bebek yang berulang tahun hanya kandang spesial pertama sampai kandang spesial ke-b, dan kita hanya dapat menukarkan c pasang kotak, maka berapakah jumlah permen maksimum yang bisa kita letakkan pada kandang-kandang yang berisi bebek yang berulang tahun?

Fungsi ini dapat dihitung dengan mencoba kotak mana yang ingin diletakkan ke kandang spesial ke-b, lalu kita memanggil fungsi ini lagi untuk mencoba kotak mana yang ingin diletakkan ke kandang spesial dengan indeks yang lebih kecil. Dengan kata lain, fungsi ini dapat dihitung dengan menggunakan rekursi di bawah ini:

$$f(a, b, c) = \max_{1 \le i \le a} f(i-1, b-1, c - dist(i, C[b])) + P[i]$$

yang mana $C[b]$ adalah posisi dari kandang spesial ke-b dan $dist(i, C[b])$ adalah banyaknya penukaran agar kotak pada posisi i dapat diletakkan pada kandang ke-$C[b]$. Dengan kata lain, $dist(i, C[b]) = |i - C[b]|$.

Jawaban yang kita inginkan adalah nilai $f(N, M, K)$. Karena banyaknya state dari fungsi ini adalah $O(NMK)$ dan penghitungan satu nilai $f(a, b, c)$ membutuhkan waktu $O(N)$, kompleksitas dari solusi ini adalah $O(N^2MK)$. Karena $M = O(N)$, kompleksitas dari solusi ini bisa disederhanakan menjadi $O(N^3K)$.

2.4 Subsoal 6

Perhatikan bahwa jika $K > \binom{N}{2}$, maka terdapat solusi optimal yang hanya membutuhkan tidak lebih dari $\binom{N}{2}$ penukaran. Maka, solusi untuk subsoal ini diadaptasi dari solusi untuk subsoal 5, hanya saja pada awalnya kita menguji apakah $K > \binom{N}{2}$. Jika benar, maka kita dapat mengganti nilai K dengan $\binom{N}{2}$.

Kompleksitas dari solusi ini adalah $O(N^3\binom{N}{2}) = O(N^5)$.

2.5 Subsoal 7

Mengadaptasi solusi dari subsoal 5, perhatikan bahwa

$$
\begin{aligned}
f(a-1,b,c) &= \max_{1\le i\le a-1} f(i-1,b-1,c-dist(i,C[b])) + P[i] \\
f(a,b,c) &= \max_{1\le i\le a} f(i-1,b-1,c-dist(i,C[b])) + P[i] \\
&= \max(\max_{1\le i\le a-1} f(i-1,b-1,c-dist(i,C[b])) + P[i], f(a-1,b-1,c-dist(a,C[b])) + P[a]) \\
&= \max(f(a-1,b,c), f(a-1,b-1,c-dist(a,C[b])) + P[a])
\end{aligned}
$$

Setiap penghitungan satu nilai $f(a,b,c)$ sekarang membutuhkan waktu $O(1)$. Sehingga, kompleksitas dari solusi ini menjadi $O(N^2K)$. Dengan menggabungkan observasi pada subsoal 6, kompleksitas dari solusi ini menjadi $O(N^4)$.

2.6 Subsoal 8

Untuk menyelesaikan subsoal ini, kita membutuhkan fungsi lain. Mari kita definisikan fungsi $g(a,b,l)$ sebagai nilai terkecil c yang memenuhi $f(a,b,c) \ge l$, atau ∞ jika tidak ada nilai c yang memenuhi.

Dengan kata lain, $g(a,b,l)$ dapat didefinisikan sebagai berikut: asumsikan kita hanya memiliki kotak pertama sampai kotak ke-a, dan kandang-kandang yang berisi bebek yang berulang tahun hanya kandang spesial pertama sampai kandang spesial ke-b, dan kita menginginkan jumlah permen yang berada pada kandang-kandang spesial setidaknya l, maka berapakah banyaknya penukaran minimum?

Fungsi ini dapat dihitung secara rekursi sebagai berikut

$$g(a,b,l) = \min(g(a-1,b,l), g(a-1,b-1,l-P[a]) + dist(a,C[b])]$$

yang mana $C[b]$ adalah posisi dari kandang spesial ke-b dan $dist(i,C[b])$ adalah banyaknya penukaran agar kotak pada posisi i dapat diletakkan pada kandang ke-$C[b]$. Dengan kata lain, $dist(i,C[b]) = |i - C[b]|$.

Karena kita ingin memaksimalkan jumlah permen yang berada pada kandang-kandang spesial, namun hanya boleh menukarkan kotak paling banyak K kali, maka jawaban yang kita inginkan adalah nilai terbesar c yang memenuhi $g(N,M,c) \le K$.

Karena banyaknya state dari fungsi ini adalah $O(N \times M \times 9N)$ dan penghitungan satu nilai $g(a,b,c)$ membutuhkan waktu $O(1)$, kompleksitas dari solusi ini adalah $O(N^2M)$. Karena $M = O(N)$, kompleksitas dari solusi ini bisa disederhanakan menjadi $O(N^3)$.

3 1C : Daratan dan Es

Soal ini ditulis oleh Ammar Fathin Sabili.

3.1 Subsoal 3

Subsoal ini dapat diselesaikan dengan mengunjungi seluruh petak, salah satunya dengan cara berikut.

Kita selalu dapat mengasumsikan petak daratan dengan nilai 1 berada pada baris pertama dan kolom pertama. Kita dapat menggerakkan pemain ke KANAN sebanyak 5 kali untuk mendapatkan nilai-nilai pada seluruh petak daratan di baris tersebut lalu kembali lagi ke kolom pertama. Selanjutnya, kita dapat menggerakkan pemain ke BAWAH kemudian kembali melakukan pergerakan ke KANAN sebanyak 5 kali untuk mendapatkan nilai-nilai pada baris kedua. Pergerakan ini dilakukan hingga baris terakhir.

Dengan cara tersebut, kita dapat mengetahui nilai-nilai pada seluruh petak untuk setiap baris sebagai salah satu dunia yang kompatibel, dengan melakukan pergerakan sebanyak $5 \times 5 + 4 = 29$ kali.

Kompleksitas dari subsoal ini adalah $O(N)$.

3.2 Subsoal 4

Mulai dari subsoal ini, perlu dilakukan *depth-first-search* (DFS) untuk mengetahui petak-petak yang dapat dicapai dari suatu petak i menggunakan salah satu dari keempat kemungkinan gerakan (ATAS, BAWAH, KIRI, KANAN). Informasi tersebut dapat kita simpan dalam sebuah adjacency matrix. DFS ini hanya dilakukan sekali di awal, dan memerlukan pergerakan sebanyak $4 \times N$ kali, cukup dengan batasan yang ada untuk seluruh subsoal.

Apabila $N = 25$, maka dapat diselesaikan dengan cara seperti pada subsoal sebelumnya. Untuk $N = 24$, perlu sedikit kerja ekstra lagi. Pertama-tama, kita tetapkan bahwa angka 1 berada di petak pada baris pertama dan kolom pertama. Selanjutnya, kita lakukan *complete search* untuk meletakkan petak es di salah satu dari 24 petak yang tersisa. Selanjutnya, isi nilai-nilai dari petak satu per satu, dengan memanfaatkan *adjacency matrix* dan petak es yang sudah kita letakkan. Lakukan hingga mendapatkan pengisian yang sah.

Kompleksitas dari subsoal ini adalah $O(N \times 24 \times 25)$. Karena $N \approx 25$, maka sama saja dengan $O(N^3)$.

3.3 Subsoal 5

Untuk subsoal ini, kita dapat melakukan *complete search* $O(N \times 25^N)$ untuk meletakkan petak-petak daratan. Lakukan hingga mendapatkan peletakan yang sah.

Alternatifnya, setelah mendapatkan *adjacency matrix*, tangani semua kemungkinan kasus secara manual. Untuk $N \leq 4$, terdapat tidak lebih dari 30 kasus yang unik.

Kompleksitas dari subsoal ini adalah $O(N \times 25^N)$ (*complete search*) atau $O(N)$ (menangani kasus secara manual).

3.4 Subsoal 6

Anggap urutan kolom pada baris yang terisi penuh sudah ditetapkan. Akibatnya, kolom dari suatu petak bisa ditetapkan secara pasti, dengan berpatokan kepada baris yang sudah penuh tersebut. Untuk mempermudah, tetapkan bahwa baris yang terisi penuh adalah baris ke-1. Selanjutnya, kita akan menentukan urutan baris-baris sisanya.

Perhatikan bahwa untuk dua baris berbeda, terdapat dua kemungkinan:

1. Kedua baris tersebut memiliki petak daratan di kolom yang sama. Akibatnya, urutan relatif dari kedua baris tersebut dapat ditentukan melalui urutan petak daratan pada kolom tersebut. Hal ini karena kita sudah menetapkan bahwa baris yang terisi penuh adalah baris ke-1.

2. Kedua baris tersebut tidak memiliki petak daratan di kolom yang sama. Untuk kasus ini, urutan kedua baris tersebut bisa saling lepas, atau ditentukan melalui kolom lain sesuai kemungkinan 1.

Maka, untuk subsoal ini, yang diperlukan hanyalah mengurutkan baris-baris tersebut sesuai dua kemungkinan di atas.

Kompleksitas dari subsoal ini adalah $O(N + 25log5) = O(N)$

3.5 Subsoal 7

Setelah didapatkan *adjacency matrix* untuk setiap angka, buat himpunan baris yang masing-masing barisnya berisi kumpulan urutan angka yang bersebelahan dalam baris tersebut. Himpunan dibuat dengan *pseudocode* sebagai berikut:

```
set_baris = []
visited = [False] * N
for i in [1..N]:
    if not visited[i]:
        visited[i] = True
        cur_baris = []
        cur_baris.append(i)

        X = kanan[i]
        while(X != i):
            visited[X] = True
            cur_baris.append(X)
            X = kanan[i]
        set_baris.append(cur_baris)
```

Lakukan hal yang sama dengan kolom, yaitu dengan membuat himpunan kolom; hanya saja *kanan* diganti dengan *bawah*.

Kemudian, lakukan *complete search* dengan mempermutasikan urutan untuk himpunan baris. Dalam setiap permutasi tersebut, permutasikan urutan pada kolom. Hal ini dilakukan untuk menentukan baris dan kolom untuk setiap angka. Setiap angka dijamin memiliki baris dan kolom yang berbeda. Setelah mendapatkan baris dan kolom untuk setiap angka, cek apakah konsisten dengan *adjacency matrix* yang telah dibuat di awal. Apabila konsisten, keluarkan jawabannya.

Kompleksitas dari subsoal ini adalah $O(N \times (\sqrt{N}!)^2)$.

4 2A : Laskar Bebek

Soal ini ditulis oleh Anthony. Untuk kemudahan diskusi, mari kita sebut B sebagai barisan yang merupakan A setelah diurutkan. Lalu, kita sebut operasi dengan waktu 1 (mengurutkan 1 elemen) sebagai operasi *dummy*.

4.1 Subsoal 3

Subsoal ini dapat diselesaikan dengan mencoba semua kemungkinan pasangan indeks yang dipilih untuk setiap operasi, lalu mencari total waktu minimum dari semua kemungkinan yang sah. Terdapat $O(N^2)$ kemungkinan pasangan indeks, dan untuk mengurutkan bebek-bebek pada barisan diperlukan waktu $O(NlogN)$. Karena terdapat K buah operasi, maka kompleksitas untuk subsoal ini adalah $O((N^3logN)^K)$.

4.2 Subsoal 4

Berhubung pada subsoal ini K hanya bernilai 1, maka kita harus mengurutkan A hanya dalam 1 operasi. Terdapat 2 kasus yang harus ditangani:

1. Apabila $A = B$ (atau dengan kata lain, barisan A sudah terurut), maka total waktu yang diperlukan adalah 1. (Ingat bahwa kita harus menggunakan tepat K operasi).
2. Selain itu, maka terdapat setidaknya dua buah bilangan bulat i dan j, sedemikian sehingga $A[i] \neq B[i]$ dan $A[j] \neq B[j]$. Definisikan x sebagai nilai minimum yang mana $A[x] \neq B[x]$ dan y sebagai nilai maksimum yang mana $A[y] \neq B[y]$. Maka, kita hanya perlu mengurutkan $A[x \dots y]$. Sehingga, total waktu yang diperlukan adalah $y - x + 1$.

Sehingga, kompleksitas untuk subsoal ini adalah $O(NlogN + N) = O(NlogN)$

4.3 Subsoal 5

Seperti subsoal 4, terdapat 2 kasus yang harus ditangani:

1. Apabila $A = B$ (barisan A sudah terurut), maka total waktu yang diperlukan adalah K.
2. Selain itu, maka barisan A akan berbentuk sebagai berikut:
   $1, 1, 1, 1, \dots, 2, \dots, 1, \dots, 2, 2, 2$
   Dengan kata lain, A dapat dilihat sebagai terdiri dari 3 bagian, yaitu bagian awal yang terdiri dari 1 saja, bagian akhir yang terdiri dari 2 saja, dan bagian tengah yang belum terurut. Misal bagian tengah bermula di indeks x dan berakhir di indeks y. Maka, salah satu solusi yang optimal adalah mengurutkan bagian tengah dengan total waktu $y - x + 1$, lalu melakukan operasi *dummy* untuk $K - 1$ operasi sisanya. Sehingga, total waktu yang diperlukan untuk kasus ini adalah $K - 1 + y - x + 1$.

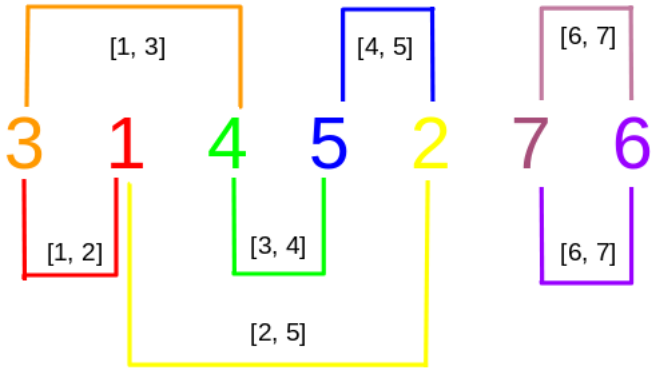
Sehingga, kompleksitas untuk subsoal ini adalah $O(NlogN + N) = O(NlogN)$

4.4 Subsoal 6

Observasi pada subsoal ini merupakan kunci untuk mendapatkan solusi subsoal 7 dan 8.

Untuk menyelesaikan subsoal ini, diperlukan suatu observasi tambahan. Kita definisikan $P[i]$ sebagai posisi $A[i]$ pada B. Apabila terdapat beberapa nilai $A[i]$ yang sama, maka nilai $P[i]$ untuk elemen-elemen tersebut ditentukan oleh urutan relatifnya. Selanjutnya, definisikan $L[i] = \min(i, P[i])$ dan $R[i] = \max(i, P[i])$.

Untuk setiap nilai i, kita bisa menganggap $[L[i], R[i]]$ sebagai kemungkinan (bisa saja salah) interval terkecil yang harus diurutkan agar elemen ke-i berada di posisi yang benar. Sebagai contoh, berikut visualisasi untuk contoh masukan 1:



Selanjutnya, kita dapat membuat beberapa observasi berikut:

Observasi 4.1. *Dua buah interval $[L[i], R[i]]$ dan $[L[j], R[j]]$ dapat digabungkan menjadi satu interval $[L[i], R[j]]$ jika dan hanya $R[i] \geq L[j]$. Kemudian, penggabungan tersebut menghasilkan interval yang lebih benar.*

Observasi 4.2. *Penggabungan dari observasi sebelumnya akan menghasilkan interval dengan total waktu yang sama atau lebih singkat.*

Bukti. Anggap kita harus menggunakan 2 operasi. Sebelum penggabungan, total waktu yang dibutuhkan adalah $(R[i]-L[i]+1)+(R[j]-L[j]+1)$. Setelah penggabungan, total waktu yang dibutuhkan adalah $(R[j]-L[i]+1)+1$ (gunakan 1 operasi *dummy*). Bisa dilihat bahwa

$$(R[j] - L[i] + 1) + 1 + (R[i] - L[j]) = (R[i] - L[i] + 1) + (R[j] - L[j] + 1)$$

Karena $R[i] \geq L[j]$, maka

$$(R[j] - L[i] + 1) + 1 \leq (R[i] - L[i] + 1) + (R[j] - L[j] + 1)$$

□

Observasi 4.3. *Misal setelah tidak ada interval yang bisa digabung, interval-interval yang tersisa dinyatakan sebagai $[L'[1], R'[1]], [L'[2], R'[2]], \ldots, [L'[M], R'[M]]$. Maka, untuk mengurutkan seluruh barisan A, kita hanya perlu mengurutkan sesuai interval-interval tersebut.*

Bukti. Akibat definisi $L[i]$ dan $R[i]$, serta penggabungan interval, maka $[L'[j], R'[j]]$ menyatakan interval terkecil yang harus diurutkan agar tiap elemen ke-i, $L'[j] \leq i \leq R'[j]$, berada di posisi yang benar. □

Observasi 4.4. *Dari observasi sebelumnya, kita hanya perlu mengurutkan semua interval dengan ukuran lebih dari 1 untuk mengurutkan seluruh barisan A.*

Definisi 4.1. *M adalah banyaknya interval yang sudah tidak bisa digabung lagi serta berukuran lebih dari 1.*

Observasi 4.5. *Apabila $M \leq K$, maka gunakan M operasi untuk mengurutkan seluruh barisan A, lalu gunakan $K - M$ operasi sisanya sebagai operasi dummy.*

Observasi 4.6. *Apabila $M > K$, maka kita lakukan penggabungan interval sebanyak $M - K$ kali. Dalam menggabungkan interval, pilih 2 interval yang jaraknya paling kecil. Setelah itu, akan tersisa tepat K interval, dan kita cukup mengurutkan K interval tersebut.*

Berdasarkan seluruh observasi tersebut, kita memerlukan suatu struktur data yang mendukung dua operasi berikut:

1. Menggabungkan dua himpunan (interval) yang beririsan.
2. Mencari interval-interval yang tidak beririsan.

Kita dapat menggunakan struktur data disjoint-set. Inisialisasi *disjoint-set* agar awalnya, *root* dari himpunan yang mengandung i adalah i. Pelihara *disjoint-set* agar *root* dari tiap himpunan merepresentasikan $R'[j]$. Selanjutnya, mencari interval-interval yang tidak beririsan dapat dilakukan dengan memanfaatkan *disjoint-set* tersebut.

Untuk subsoal ini, perhatikan bahwa A merupakan suatu permutasi dari 1 sampai N. Maka, $P[i] = A[i]$. Untuk setiap i, gabungkan himpunan-himpunan yang mengandung $L[i], L[i] + 1, \ldots, R[i]$. Ini dapat dilakukan dengan iterasi secara $O(N)$. Apabila sudah selesai, maka kita dapat memanfaatkan *disjoint-set* untuk mencari interval-interval yang tidak beririsan, lalu menghitung total waktu minimum yang diperlukan dengan menggunakan observasi-observasi yang telah didapat.

Kompleksitas untuk subsoal ini adalah $O(N \times N + NlogN) = O(N^2)$

4.5 Subsoal 7

Subsoal ini mirip dengan subsoal 6. Namun, A bukan merupakan suatu permutasi, yang berarti dalam menentukan $P[i]$, kita harus memperhatikan urutan relatifnya juga. Selain itu, semua sama dengan subsoal 6.

Kompleksitas untuk subsoal ini adalah $O(N \times N + NlogN) = O(N^2)$

4.6 Subsoal 8

Untuk subsoal ini, iterasi secara naif untuk menggabungkan himpunan-himpunan tidak dapat dilakukan. Namun, perhatikan bahwa sebenarnya iterasi ini dapat dioptimasi lagi. Dengan memanfaatkan *disjoint-set* dan definisi *root* himpunan pada subsoal 6, kita cukup melakukan iterasi sebagai berikut:

```
x = find_root(L[i])
while x < R[i]:
    join_set(x, x+1)
    x = find_root(x)
```

`find_root` menyatakan operasi mencari *root*, sedangkan `join_set` menyatakan operasi menggabungkan himpunan. Dengan iterasi seperti itu, total penggabungan dijamin adalah $O(N)$.

Sehingga, kompleksitas untuk subsoal ini adalah $O(N + NlogN) = O(NlogN)$

5 2B : Labirin Dengklek

Soal ini ditulis oleh Alham Fikri Aji.

5.1 Subsoal 3

Subsoal ini dapat diselesaikan dengan menggunakan *complete search*. Terdapat $N \times M$ kemungkinan petak untuk karakter pertama pada S. Setiap karakter pada string memiliki 4 kemungkinan arah untuk karakter yang akan dicek selanjutnya. Terdapat 2 kasus pengecekan string:

1. Apabila string yang dicek merupakan irisan antara S dan T, maka panjang maksimum yang mungkin dicapai adalah $S + T - 1 \approx S + T$. Banyaknya path maksimum yang mungkin dicapai dari setiap petak awal adalah $4^{|S|+|T|}$, sehingga kompleksitas untuk kasus ini adalah $O(N \times M \times 4^{|S|+|T|})$.
2. Apabila string yang dicek bukan merupakan irisan antara S dan T, maka akan ada pengecekan tambahan dari karakter terakhir S ke karakter pertama T. Terdapat $N \times M$ kemungkinan petak untuk karakter pertama T, sehingga kompleksitas untuk kasus ini adalah $O(N^2 \times M^2 \times 4^{|S|+|T|})$.

Kompleksitas dari subsoal ini adalah $O(N^2 \times M^2 \times 4^{|S|+|T|})$.

5.2 Subsoal 4

Perhatikan bahwa $|S| = |T| = 1$. Hal ini mereduksi persoalan menjadi mencari jarak terpendek antara dua karakter. Terdapat $N \times M$ kemungkinan posisi untuk karakter di S, dan $N \times M$ kemungkinan posisi untuk karakter di T. Jarak antara dua karakter adalah *manhattan distance* dari dua karakter tersebut.

Kompleksitas dari subsoal ini adalah $O((NM)^2)$.

5.3 Subsoal 5

Persoalannya mirip seperti subsoal 4, tetapi $N, M \geq 200$ sehingga tidak dapat dilakukan *complete search* untuk setiap petak yang mungkin. Solusi yang mungkin adalah melakukan *breadth-first-search* (BFS) secara serentak dari seluruh petak yang berisi karakter S ke seluruh petak yang berisi karakter T.

Kompleksitas dari subsoal ini adalah $O(NM)$.

5.4 Subsoal 6

Aturan $|S| = 1$ tidak lagi berlaku. Karena itu, persoalan ini hanya dapat direduksi menjadi persoalan mengecek kemunculan sebuah string dalam suatu grid. Hal ini dapat diselesaikan dengan cara BFS dengan state [baris][kolom][indeks pada string]. Setelah mendapatkan semua petak akhir yang mungkin dari string prefiks, dapat dilakukan pengecekan jarak terdekat dengan karakter pada T seperti pada subsoal 4. Kompleksitas BFS adalah $O(NM|S|)$, dan kompleksitas pengecekan setiap pasang petak adalah $O((NM)^2)$.

Kompleksitas dari subsoal ini adalah $O(NM|S| + (NM)^2)$.

5.5 Subsoal 7

Sama seperti subsoal 6, tetapi pengecekan terhadap karakter pada T dapat dihitung dengan BFS seperti pada subsoal 5.

Kompleksitas dari subsoal ini adalah $O(NM|S|)$.

5.6 Subsoal 8

Untuk masing-masing S dan T, carilah seluruh kemunculan kedua string yang mungkin dengan menggunakan BFS seperti subsoal 6.

Pada subsoal ini, aturan $|T| = 1$ tidak lagi berlaku. Terdapat beberapa observasi tambahan dari subsoal sebelumnya. Misal terdapat string X yang akan divalidasi. Ada 2 kemungkinan kasus untuk persoalan ini:

1. $|X| < |S| + |T|$. X memiliki irisan antara sufiks S dan prefiks T. Lakukan pencarian pada grid dengan menggunakan seluruh X yang mungkin dengan BFS seperti pada subsoal 7.
   Kompleksitasnya adalah $O(N \times M \times (|S| + |T|)^2)$.

2. $|X| \geq |S| + |T|$. Simpan semua petak yang terdapat karakter akhir yang mungkin untuk S (seperti pada subsoal 6), dan semua petak yang terdapat karakter awal yang mungkin untuk T. Pencarian karakter awal untuk T dilakukan dari belakang ke depan. Kemudian, cari jarak minimum untuk setiap pasangan (karakter akhir S, karakter awal T) dengan menggunakan BFS dari semua petak yang mungkin seperti pada subsoal 5. Anggap saja jarak ini D_{min}; sehingga panjang string minimum adalah $|S|+|T|+D_{min}-1$. Kompleksitasnya adalah $O(N \times M \times (|S| + |T|))$.

Kompleksitas dari subsoal ini adalah $O(N \times M \times (|S| + |T|)^2)$.

5.7 Subsoal 9

Terdapat *bottleneck* pada kasus $|X| < |S| + |T|$ diatas. Perhatikan bahwa kita tidak perlu untuk mengecek setiap string yang beririsan, tetapi dapat dicek sekali saja dengan melakukan BFS pada **masing-masing** string S dan T seperti kasus kedua pada subsoal 8 diatas.

Definisi 5.1. *Sebuah string X dikatakan **sah** apabila memiliki prefiks S dan sufiks T.*

Definisi 5.2. *Sebuah petak (x, y) dikatakan **sah** sebagai karakter ke-$i(1 \leq i \leq |S|)$ dari S apabila state $[x][y][i]$ pada BFS terhadap S bernilai **true**.*

Definisi 5.3. *Sebuah petak (x, y) dikatakan **sah** sebagai karakter ke-$i(1 \leq i \leq |T|)$ dari T apabila state $[x][y][i]$ pada BFS terhadap T bernilai **true**.*

Apabila terdapat suatu petak (x, y) yang sah sebagai karakter ke-$i(1 \leq i \leq |S|)$ dari S dan juga sah sebagai karakter pertama dari T (ingat bahwa BFS untuk string T dilakukan dari belakang), maka terdapat string X sah yang memiliki panjang $|X| = i + |T| - 1$. Perhatikan bahwa $i \leq |S|$, sehingga $|X| < |S| + |T|$.

Kompleksitas untuk BFS adalah $O(N \times M \times (|S|+|T|)$. Banyaknya *state* yang harus dicek adalah $N \times M \times |S|$, sehingga kompleksitas untuk kasus $|X| < |S| + |T|$ menjadi $O(N \times M \times (|S| + |T|))$.

Kompleksitas dari subsoal ini adalah $O(N \times M \times (|S| + |T|))$.

6 2C : Aritmetika Bebek

Soal ini ditulis oleh Jonathan Irvin Gunawan.

Definisi 6.1. *Pasangan array (A', B') dikatakan sebuah solusi jika A' dan B' membentuk barisan aritmatik.*

Definisi 6.2. *Fungsi $dif(S, S')$ adalah fungsi yang memiliki dua barisan sebagai parameter dan mengembalikan banyaknya indeks i yang mana $S[i] \neq S'[i]$.*

Definisi 6.3. *Fungsi $opt(A', B')$ adalah banyaknya operasi (seperti didefinisikan pada soal) minimum yang dibutuhkan untuk mengubah A menjadi A' dan B menjadi B'.*

Definisi 6.4. *Fungsi $optA(A')$ adalah banyaknya operasi (hanya menggunakan operasi jenis pertama) minimum yang dibutuhkan untuk mengubah A menjadi A'. Dengan kata lain, $optA(A') = dif(A, A')$. Dengan definisi serupa, fungsi $optB(B')$ adalah banyaknya operasi (hanya menggunakan operasi jenis kedua) minimum yang dibutuhkan untuk mengubah B menjadi B'. Dengan kata lain, $optB(B') = dif(B, B')$.*

Definisi 6.5. *Pasangan array (A', B') dikatakan sebuah solusi optimal jika (A', B') merupakan sebuah solusi dan tidak terdapat pasangan array (A'', B'') yang merupakan sebuah solusi dan $opt(A'', B'') < opt(A', B')$.*

Observasi 6.1. *Jika $A = B$, maka terdapat solusi optimal (A', B') yang memenuhi $A' = B'$.*

Corollary 6.1. *Jika $A = B$, maka terdapat solusi optimal yang dapat diraih dengan hanya menggunakan operasi ketiga.*

Observasi 6.2. *Jika $A = B$ dan $N > 1$, maka terdapat solusi optimal (A', B') yang mana $A' = B'$ dan terdapat dua indeks i dan j berbeda yang memenuhi $A'[i] = A[i]$ dan $A'[j] = A[j]$.*

Bukti. Marilah kita membuat array A'' dan B'' dengan konstruksi sebagai berikut:

- $A''[1] = B''[1] = A[1]$,
- $A''[2] = B''[2] = A[2]$, dan
- $A''[i] = B''[i] = A[1] + (A[2] - A[1]) \times (i - 1)$

Maka, (A'', B'') adalah sebuah solusi dan $opt(A'', B'') \leq N - 2$. Karena (A', B') adalah solusi optimal, maka $opt(A', B') \leq opt(A'', B'') \leq N - 2$. Dengan demikian, terdapat setidaknya dua indeks yang nilainya tidak diubah sama sekali. □

6.1 Subsoal 3

Subsoal ini dapat diselesaikan dengan mencoba semua pasangan bilangan bulat (i, j) yang mana $A[i]$ dan $A[j]$ tidak diganti pada solusi optimal. Untuk setiap pasangan bilangan bulat, kita dapat menghitung ada berapa banyak indeks k yang mana $A[k]$ harus diganti dan kita akan mencari yang minimum dari seluruh (i, j). Dengan kata lain, mari kita definisikan $change(i, j)$ sebagai banyaknya k yang memenuhi $A[k] \neq A[i] + (k-i) \times \frac{A[j]-A[i]}{j-i}$. Maka, kita ingin mencari nilai dari

$$\min_{1 \leq i < j \leq N} change(i, j)$$

Kompleksitas dari subsoal ini adalah $O(N^3)$.

6.2 Subsoal 4

Subsoal ini dapat diselesaikan dengan mencoba sebuah bilangan bulat i yang mana $A[i]$ tidak diganti pada solusi optimal. Untuk setiap bilangan bulat i, definisikan $f(j)$ untuk $i \neq j$ sebagai $\frac{A[j]-A[i]}{j-i}$.

Observasi 6.3. *$change(i, j)$ adalah banyaknya k yang memenuhi $f(j) \neq f(k)$.*

Bukti. Jika nilai j dan k memenuhi $A[k] \neq A[i] + (k - i) \times \frac{A[j]-A[i]}{j-i}$, maka

$$\begin{aligned} A[k] &\neq A[i] + (k - i) \times \frac{A[j] - A[i]}{j - i} \\ A[k] - A[i] &\neq (k - i) \times \frac{A[j] - A[i]}{j - i} \\ \frac{A[k] - A[i]}{k - i} &\neq \frac{A[j] - A[i]}{j - i} \\ f(k) &\neq f(j) \end{aligned}$$

□

Jika kita definisikan $f'(D)$ sebagai banyaknya j yang memenuhi $f(j) = D$, maka kita ingin mencari nilai D yang memiliki nilai $f'(D)$ terbesar. Dengan kata lain, kita ingin memilih nilai j sedemikian sehingga terdapat banyak nilai $k \neq j$ yang memenuhi $f(j) = f(k)$.

Untuk mencari nilai D yang memiliki nilai $f'(D)$ terbesar, kita dapat menyimpan nilai $f'(D)$ untuk setiap D dalam sebuah hash table. Untuk setiap indeks $j \neq i$, kita dapat menambahkan satu pada nilai $f'(f(j))$.

Kompleksitas dari subsoal ini adalah $O(N^2)$.

6.3 Subsoal 5

Untuk menyelesaikan subsoal ini, terdapat dua kasus yang harus kita perhatikan:

1. $optsame(A, B)$ akan menghasilkan solusi (A', B') yang mana $A' = B'$, dan $opt(A', B') \leq opt(A'', B'')$ untuk seluruh solusi (A'', B'') yang memenuhi $A'' = B''$. Jika ada lebih dari satu solusi, $optsame(A, B)$ dapat menghasilkan solusi yang mana saja.

2. $optdiff(A, B)$ akan menghasilkan solusi (A', B') yang mana $A' \neq B'$, dan $opt(A', B') \leq opt(A'', B'')$ untuk seluruh solusi (A'', B'') yang memenuhi $A'' \neq B''$. Jika ada lebih dari satu solusi, $optdiff(A, B)$ dapat menghasilkan solusi yang mana saja.

6.3.1 Menghitung $opt(optsame(A, B))$

Bagian ini dapat dihitung dengan hanya menggunakan operasi jenis ketiga. Marilah kita membuat array baru X berukuran N dengan konstruksi sebagai berikut:

$$X[i] = \begin{cases} A[i], & \text{jika } A[i] = B[i] \\ -1, & \text{jika } A[i] \neq B[i] \end{cases}$$

Maka, $optsame(A, B)$ dapat diraih menggunakan solusi pada subsoal 4 dengan menggunakan X sebagai masukan. Namun, jika $X[i] = -1$, maka i tidak boleh merupakan indeks yang tidak diubah.

6.3.2 Menghitung $opt(optdiff(A, B))$

Katakan $optdiff(A, B) = (A', B')$.

Observasi 6.4. *Untuk meraih solusi $optdiff(A, B)$, maka operasi jenis ketiga hanya digunakan paling banyak sekali*

Bukti. Asumsikan operasi jenis ketiga digunakan pada indeks i dan j. Maka, $A'[i] = B'[i]$ dan $A'[j] = B'[j]$. Maka, $A'[k] = B'[k]$ juga berlaku untuk semua $1 \leq k \leq N$. Maka, $A' = B'$. Kontradiksi. □

Observasi 6.5. $optA(A') + optB(B') - 1 \leq opt(A', B') \leq optA(A') + optB(B')$

Bukti. Jika (A', B') diraih tanpa menggunakan operasi jenis ketiga, maka $opt(A', B') = optA(A') + optB(B')$. Jika (A', B') diraih dengan menggunakan operasi jenis ketiga tepat sekali, maka $opt(A', B') = optA(A') + optB(B') - 1$. □

Observasi 6.6. $opt(A', B') = optA(A') + optB(B') - 1$ *jika dan hanya jika terdapat solusi optimal* (A', B') *dan sebuah indeks* i *yang mana* $A[i] \neq A'[i]$, $B[i] \neq B'[i]$, *dan* $A'[i] = B'[i]$.

Bukti. Daripada menggunakan operasi pertama dan kedua pada indeks i untuk mengganti $A[i]$ menjadi $A'[i]$ dan $B[i]$ menjadi $B'[i]$, kita dapat menggunakan operasi jenis ketiga pada indeks i karena $A'[i] = B'[i]$. Melakukan hal ini akan menghemat satu operasi. □

Untuk menguji apakah terdapat solusi optimal (A', B') dan sebuah indeks i yang mana $A[i] \neq A'[i]$, $B[i] \neq B'[i]$, dan $A'[i] = B'[i]$, kita dapat mencoba semua pasangan empat nilai (iA, jA, iB, jB) yang memenuhi $change(iA, jA) = optA(A')$ pada A dan $change(iB, jB) = optB(B')$ pada B yang berarti $A'[iA] = A[iA]$, $A'[jA] = A[jA]$, $B'[iB] = B[iB]$, dan $B'[jB] = B[jB]$. Lalu untuk setiap pasangan nilai, kita menguji apakah terdapat nilai **bulat** i yang memenuhi:

- $1 \leq i \leq N$
- $iA \neq i$
- $jA \neq i$
- $iB \neq i$

- $jB \neq i$
- $A[iA] + (i - iA) \times \frac{A[jA]-A[iA]}{jA-iA} = B[iB] + (i - iB) \times \frac{B[jB]-B[iB]}{jB-iB}$ (ekuivalen dengan $A'[i] = B'[i]$)
- $A[iA] + (i - iA) \times \frac{A[jA]-A[iA]}{jA-iA} \neq A[i]$ (ekuivalen dengan $A'[i] \neq A[i]$)
- $B[iB] + (i - iB) \times \frac{B[jB]-B[iB]}{jB-iB} \neq B[i]$ (ekuivalen dengan $B'[i] \neq B[i]$)

Jawaban yang kita inginkan adalah $\min(opt(optsame(A, B)), opt(optdiff(A, B)))$. Kompleksitas dari subsoal ini adalah $O(N^5)$.

6.4 Subsoal 6

Untuk menyelesaikan subsoal ini dan subsoal selanjutnya, kita membutuhkan solusi yang lebih cepat untuk menghitung $opt(optdiff(A, B))$.

Untuk setiap pasangan empat nilai (iA, jA, iB, jB) yang memenuhi $change(iA, jA) = optA(A')$ pada A dan $change(iB, jB) = optB(B')$ pada B, kemungkinan nilai i yang sah adalah nilai yang memenuhi:

$$\begin{aligned}
A[iA] + (i - iA) \times \frac{A[jA] - A[iA]}{jA - iA} &= B[iB] + (i - iB) \times \frac{B[jB] - B[iB]}{jB - iB} \\
A[iA] + i \times \frac{A[jA] - A[iA]}{jA - iA} - iA \times \frac{A[jA] - A[iA]}{jA - iA} &= B[iB] + i \times \frac{B[jB] - B[iB]}{jB - iB} - iB \times \frac{B[jB] - B[iB]}{jB - iB} \\
i \times \left(\frac{A[jA] - A[iA]}{jA - iA} - \frac{B[jB] - B[iB]}{jB - iB} \right) &= -A[iA] + iA \times \frac{A[jA] - A[iA]}{jA - iA} + B[iB] - iB \times \frac{B[jB] - B[iB]}{jB - iB} \\
i &= \frac{-A[iA] + iA \times \frac{A[jA]-A[iA]}{jA-iA} + B[iB] - iB \times \frac{B[jB]-B[iB]}{jB-iB}}{\frac{A[jA]-A[iA]}{jA-iA} - \frac{B[jB]-B[iB]}{jB-iB}}
\end{aligned}$$

Dengan demikian, terdapat hanya **paling banyak** satu nilai i yang harus diuji untuk setiap pasangan empat nilai (iA, jA, iB, jB).

Definisi 6.6. *$cnt(A)$ adalah banyaknya pasangan nilai (a_0, D) sedemikian sehingga terdapat pasangan bilangan bulat (iA, jA) yang memenuhi $change(iA, jA) = optA(A')$ pada A, $\frac{A[jA]-A[iA]}{jA-iA} = D$, dan $A[iA] - D \times iA = a_0$.*

Definisi 6.7. *$cnt(B)$ adalah banyaknya pasangan nilai (a_0, D) sedemikian sehingga terdapat pasangan bilangan bulat (iB, jB) yang memenuhi $change(iB, jB) = optB(B')$ pada B, $\frac{B[jB]-B[iB]}{jB-iB} = D$, dan $B[iB] - D \times iB = a_0$.*

Kompleksitas dari subsoal ini adalah $O(cnt(A) \times cnt(B))$. Karena $cnt(A) = O(N^2)$ dan $cnt(B) = O(N^2)$, maka kompleksitas dari subsoal ini adalah $O(N^4)$.

6.5 Subsoal 7

Subsoal ini dapat diselesaikan dengan mengiterasi nilai i ($1 \leq i \leq N$). Untuk setiap i, kita ingin menguji apakah terdapat nilai V sedemikian sehingga:

- $A[i] \neq V$
- $B[i] \neq V$
- terdapat empat nilai bilangan bulat (iA, jA, iB, jB) yang memenuhi:
  - $A[iA] + (i - iA) \times \frac{A[jA]-A[iA]}{jA-iA} = V$
  - $change(iA, jA) = optA(A')$
  - $B[iB] + (i - iB) \times \frac{B[jB]-B[iB]}{jB-iB} = V$
  - $change(iB, jB) = optB(B')$

Jika nilai V memenuhi syarat-syarat di atas, maka kita sebut nilai V sebagai nilai yang sah untuk i.

Mari kita definisikan sebuah himpunan V_A. $V \in V_A$ jika dan hanya jika $A[i] \neq V$ dan terdapat dua nilai bilangan bulat (iA, jA) yang memenuhi $A[iA] + (i - iA) \times \frac{A[jA]-A[iA]}{jA-iA} = V$ dan $change(iA, jA) = optA(A')$. Mari kita juga definisikan sebuah himpunan V_B. $V \in V_B$ jika dan hanya jika $B[i] \neq V$ dan terdapat dua nilai bilangan bulat (iB, jB) yang memenuhi $B[iB] + (i - iB) \times \frac{B[jB]-B[iB]}{jB-iB} = V$ dan $change(iB, jB) = optB(B')$.

Maka, nilai V yang sah untuk i akan memenuhi $V \in V_A$ dan $V \in V_B$. Dengan kata lain, akan terdapat nilai V yang sah untuk i jika dan hanya jika $V_A \cap V_B \neq \emptyset$.

V_A dapat dihitung dalam $O(N^2)$ untuk setiap nilai i dengan mengiterasi pasangan nilai (iA, jA) yang memenuhi $change(iA, jA) = optA(A')$ pada A. V_B juga dapat dihitung dengan cara serupa. Maka, kompleksitas dari subsoal ini adalah $O(N^3)$.

6.6 Subsoal 8

Lemma 6.1. *$cnt(A) \leq \frac{N^2}{N - optA(A')}$.*

Corollary 6.2. *Jika $\frac{N}{2} - 1 \leq optA(A') \leq \frac{N}{2} + 1$, maka $cnt(A) = O(N)$.*

Lemma 6.2. *Jika $optA(A') < \frac{N}{2} - 1$, maka $cnt(A) = 1$.*

Bukti. Asumsikan $cnt(A) > 1$, dan (a_0, D) dan (a'_0, D') adalah dua pasangan bilangan bulat berbeda yang memenuhi:

- Terdapat pasangan bilangan bulat (iA, jA) yang memenuhi $change(iA, jA) = optA(A')$ pada A, $\frac{A[jA] - A[iA]}{jA - iA} = D$, dan $A[iA] - D \times iA = a_0$.
- Terdapat pasangan bilangan bulat (iA', jA') yang memenuhi $change(iA', jA') = optA(A')$ pada A, $\frac{A[jA'] - A[iA']}{jA' - iA'} = D'$, dan $A[iA'] - D' \times iA' = a'_0$.

Kita akan membuat dua array baru S dan S' dengan konstruksi sebagai berikut:

- $S[i] = a_0 + i \times D$
- $S'[i] = a'_0 + i \times D'$

Karena $optA(A') < \frac{N}{2} - 1$, terdapat lebih dari $\frac{N}{2} + 1$ indeks i yang memenuhi $S[i] = A[i]$. Selain itu, terdapat lebih dari $\frac{N}{2} + 1$ indeks i yang memenuhi $S'[i] = A[i]$. Maka, terdapat setidaknya dua indeks i dan j yang memenuhi $S[i] = S'[i]$ dan $S[j] = S'[j]$. Maka, $(a_0, D) = (a'_0, D')$. Kontradiksi. □

Teorema 6.1. *Jika $optA(A') + optB(B') \leq N$, maka $cnt(A) \times cnt(B) = O(N^2)$.*

Bukti. W.L.O.G. asumsikan bahwa $optA(A') \leq optB(B')$. Maka $optA(A') \leq \frac{N}{2}$.

- Jika $optA(A') < \frac{N}{2} - 1$, maka $cnt(A) \times cnt(B) = 1 \times O(N^2) = O(N^2)$.
- Jika $\frac{N}{2} - 1 \leq optA(A') \leq \frac{N}{2}$, maka $\frac{N}{2} \leq optB(B') \leq \frac{N}{2} + 1$. Maka $cnt(A) \times cnt(B) = O(N) \times O(N) = O(N^2)$.

□

Karena teorema di atas, jika $optA(A') + optB(B') \leq N$ kita dapat menggunakan solusi subsoal 6 yang memiliki kompleksitas $O(cnt(A) \times cnt(B))$ untuk menghitung nilai $opt(optdiff(A, B))$. Jika tidak, maka kita bisa mengabaikan nilai $opt(optdiff(A, B))$ dan langsung mengambil nilai $opt(optsame(A, B))$ karena dijamin $opt(optsame(A, B)) \leq N$.

Kompleksitas dari solusi ini adalah $O(N^2)$.