Pembahasan OSN Informatika 2018

Sabili, Ammar Fathin
athin2008@gmail.com

Dzulfikar, Muhammad Ayaz
muhammad.ayaz97@gmail.com

Kolbe, Maximilianus Maria
maxhaibara97@gmail.com

Mudzakir, Muhammad Rais Fathin
rais.fathin38@gmail.com

Pramudita, Febriananda Wida
febriwpramu@gmail.com

August 15, 2018

Di bawah ini adalah solusi resmi yang digunakan oleh Scientific Committee OSN Informatika 2018. 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 : Jembatan Layang

Soal ini ditulis oleh Ammar Fathin Sabili.

1.1 Subsoal 3

Karena banyaknya jembatan layang sama dengan banyaknya lantai, maka setiap lantai yang ada pada gedung lama dan gedung baru pasti terhubung dengan tepat sebuah jembatan layang. Hal ini mengakibatkan hanya terdapat sebuah permutasi yang mungkin jika dan hanya jika semua relokasi sudah memenuhi syarat. Kompleksitas dari solusi ini adalah $O(N)$.

1.2 Subsoal 4

Subsoal ini dapat diselesaikan menggunakan *complete search*. Untuk setiap kemungkinan permutasi yang ada, periksa apakah semua relokasi sudah memenuhi syarat. Kompleksitas dari solusi ini adalah $O(N!)$.

1.3 Subsoal 5

Untuk mempermudah diskusi, mari kita gunakan definisi berikut:

Definisi 1.1. *Banyaknya lantai i pada gedung lama yang terhubung dengan jembatan ke-j yang memenuhi $i > H[j]$ dinyatakan dalam P_j.*

Definisi 1.2. *Banyaknya lantai i pada gedung baru yang terhubung dengan jembatan ke-j yang memenuhi $i > H[j]$ dinyatakan dalam P'_j.*

Definisi 1.3. *Banyaknya lantai i pada gedung lama yang terhubung dengan jembatan ke-j yang memenuhi $i = H[j]$ dinyatakan dalam Q_j.*

Definisi 1.4. *Banyaknya lantai i pada gedung baru yang terhubung dengan jembatan ke-j yang memenuhi $i = H[j]$ dinyatakan dalam Q'_j.*

Definisi 1.5. *Banyaknya lantai i pada gedung lama yang terhubung dengan jembatan ke-j yang memenuhi $i < H[j]$ dinyatakan dalam R_j.*

Definisi 1.6. *Banyaknya lantai i pada gedung baru yang terhubung dengan jembatan ke-j yang memenuhi $i < H[j]$ dinyatakan dalam R'_j.*

Apabila suatu lantai pada gedung lama terhubung dengan jembatan dengan ketinggian yang sama, maka lantai tersebut dapat direlokasi ke semua lantai pada gedung baru yang terhubung dengan jembatan tersebut. Begitu pula apabila suatu lantai pada gedung baru terhubung dengan jembatan dengan ketinggian yang sama, maka lantai tersebut dapat menjadi tujuan relokasi dari semua lantai pada gedung lama yang terhubung dengan jembatan tersebut.

Terdapat 4 buah kasus yang memenuhi aturan relokasi yang ada:

1. $P_i + Q_i = R'_i + Q'_i$ dan $R_i = P'_i$
2. $P_i + Q_i = R'_i$ dan $R_i = P'_i + Q'_i$
3. $P_i = R'_i$ dan $R_i + Q_i = P'_i + Q'_i$
4. $P_i = R'_i + Q'_i$ dan $R_i + Q_i = P'_i$

Perhatikan bahwa apabila $Q_i = 0$ atau $Q'_i = 0$ maka akan terdapat beberapa kasus yang saling ekuivalen.

Dengan menggunakan kaidah penjumlahan, hitung banyaknya permutasi pada masing-masing kasus lalu jumlahkan untuk memperoleh jawaban yang diinginkan. Tetapi perhatikan pula bahwa himpunan penyelesaian kasus-kasus tersebut akan beririsan pada kasus berikut: $P_i = P'_i$, $Q_i = Q'_i$, dan $R_i = R'_i$, yang menyebabkan kasus di atas terhitung 2 kali. Gunakan Prinsip Inklusi-Eksklusi untuk mengatasi kasus tersebut. Kompleksitas dari solusi ini adalah $O(N)$.

1.4 Subsoal 6

Perhatikan bahwa banyaknya cara relokasi berbeda yang melalui suatu jembatan layang tidak dipengaruhi oleh jembatan layang yang lain, sehingga kita dapat menghitung banyaknya cara relokasi berbeda untuk masing-masing jembatan layang kemudian menggunakan kaidah perkalian untuk memperoleh jawaban akhir.

Pada subsoal ini berlaku batasan $Q_i = 0$ dan $Q'_i = 0$. Sehingga untuk setiap jembatan, hanya terdapat 1 buah kasus yang mungkin terjadi dan memenuhi aturan relokasi yang ada, yaitu: $P_i = R'_i$ dan $R_i = P'_i$. Iterasi semua jembatan yang ada, lalu kalikan banyaknya permutasi pada masing-masing jembatan untuk memperoleh jawaban yang diinginkan. Kompleksitas dari solusi ini adalah $O(N)$.

1.5 Subsoal 7

Subsoal ini merupakan gabungan dari Subsoal 5 dan Subsoal 6. Iterasi semua jembatan yang ada, lalu untuk setiap jembatan, banyaknya permutasi yang mungkin dapat dihitung dengan menggunakan solusi Subsoal 5. Kemudian kalikan banyaknya permutasi pada masing-masing jembatan untuk memperoleh jawaban yang diinginkan. Kompleksitas dari solusi ini adalah $O(N)$.

2 1B : Pertahanan Padang

Soal ini ditulis oleh Jonathan Mulyawan Woenardi.

Untuk mempermudah diskusi, mari kita gunakan definisi berikut:

Definisi 2.1. *Banyak subsekuens yang berupa sajak biner bergantian dari suatu sajak biner S dinyatakan dalam (X_S, Y_S), yang mana X_S menyatakan banyaknya subsekuens ganjil dan Y_S menyatakan banyaknya subsekuens genap.*

2.1 Subsoal 3

Perhatikan bahwa panjang maksimum untuk sajak biner S yang memenuhi adalah $A+B$. Oleh karena itu, kita dapat melakukan *brute force* untuk menghasilkan semua sajak biner dengan panjang tidak melebihi $A+B$ dalam $O(2^{A+B})$. Setelah itu, kita dapat memeriksa banyaknya subsekuens sajak biner bergantian dalam $O(2^{A+B})$ juga. Dengan ini, kita dapat mencari sajak biner yang kita inginkan.

Solusi ini memiliki kompleksitas $O(2^{2(A+B)} + Q \times (R[i] - L[i]))$.

2.2 Subsoal 4

Untuk suatu sajak biner S dengan banyak sajak biner bergantian (X_S, Y_S), maka:

- Banyak subsekuens sajak biner bergantian dari $S0$ adalah $(X_S, Y_S + X_S)$
- Banyak subsekuens sajak biner bergantian dari $S1$ adalah $(X_S + Y_S + 1, Y_S)$

Sehingga, penghitungan banyaknya subsekuens sajak biner bergantian dapat dilakukan dalam $O(|S|)$. Untuk menyelesaikan subsoal ini, kita cukup mengganti pencarian banyaknya subsekuens dengan cara ini.

Solusi ini memiliki kompleksitas $O(2^{A+B} \times (A+B) + Q \times (R[i] - L[i]))$.

2.3 Subsoal 5

Menggunakan observasi dari subsoal sebelumnya, kita dapat memeriksa apakah ada sajak biner yang memenuhi dengan menggunakan dynamic programming (DP). Adapun state dari DP tersebut adalah (X_S, Y_S) yang dibahas sebelumnya, dengan transisi mengikuti observasi pada subsoal 4.

Definisikan fungsi $f(X_S, Y_S)$ sebagai fungsi yang memeriksa apakah dengan banyak subsekuens sajak biner bergantian (X_S, Y_S) kita bisa mencapai banyak subsekuens sajak biner bergantian (A, B). Maka:

$$f(X_S, Y_S) = \begin{cases} true & (X_S = A \wedge Y_S = B) \\ false & (X_S > A \vee Y_S > B) \\ f(X_S, Y_S + X_S) \vee f(X_S + Y_S + 1, Y_S) & otherwise \end{cases}$$

Pengecekan dilakukan dengan memanggil $f(1, 0)$.

Mengingat untuk setiap state (X_S, Y_S) berlaku $X_S \leq A$ dan $Y_S \leq B$, maka kompleksitas DP ini adalah $O(AB)$. Untuk mencari sajak biner yang memenuhi, kita dapat melakukan *backtracking* menggunakan DP tersebut.

Solusi ini memiliki kompleksitas $O(AB + Q \times (R[i] - L[i]))$.

2.4 Subsoal 6

Untuk menyelesaikan subsoal ini, perhatikan lema berikut:

Lemma 2.1. *Apabila terdapat sajak biner yang memiliki banyak subsekuens sajak biner bergantian sebanyak (A, B), maka hanya sajak biner tersebut yang memiliki banyak subsekuens sajak biner bergantian sebanyak (A, B).*

Bukti. Pandang pencarian sajak biner yang memenuhi dari belakang. Perhatikan bahwa apabila sajak biner S memiliki banyak subsekuens sajak biner bergantian (X_S, Y_S), maka:

- Apabila $X_S > Y_S$ (lebih banyak ganjil dari genap), maka S pasti berbentuk $T1$, dengan $Y_T = Y_S$ dan $X_T = X_S - Y_S - 1$
- Apabila $X_S \leq Y_S$ (tidak lebih banyak ganjil dari genap), maka S pasti berbentuk $T0$, dengan $X_T = X_S$ dan $Y_T = Y_S - X_S$

Sehingga, dapat dipastikan bahwa jika ada sajak biner yang memenuhi, hanya sajak biner tersebut yang memenuhi. □

Menggunakan lema dan bukti tersebut, kita dapat mencari sajak biner yang memenuhi dari belakang dalam $O(A + B)$, dengan menambahkan karakter demi karakter.

Solusi ini memiliki kompleksitas $O(A + B + Q \times (R[i] - L[i]))$.

2.5 Subsoal 7

Berhubung $Q = 0$, untuk subsoal ini kita cukup memeriksa apakah ada sajak biner yang memiliki banyak subsekuens sajak biner bergantian sebanyak (A, B).

Perhatikan bahwa kita dapat mempercepat solusi sebelumnya dengan memanfaatkan operasi modulo (mirip algoritma Euclid).

Solusi ini memiliki kompleksitas $O(\log(\min(A, B)))$.

2.6 Subsoal 8

Ketimbang menyimpan utuh sajak biner yang memenuhi dalam bentuk string, kita dapat menyimpan pengulangan karakter yang sama. Misalnya, sajak biner 1111001 kita simpan sebagai $[('1', 4), ('0', 2), ('1', 1)]$. Mengingat banyaknya iterasi solusi subsoal 7 hanya sebanyak $O(\log(\min(A, B))$, kita dapat menyimpan sajak biner yang memenuhi dengan hanya menyimpan $O(\log(\min(A, B))$ pengulangan karakter dalam bentuk array. Untuk menjawab setiap pertanyaan, kita dapat mencari karakter di indeks ke-x dengan melakukan iterasi di array yang sudah kita simpan.

Solusi ini memiliki kompleksitas $O(\log(\min(A, B)) + Q \times (R[i] - L[i]) \times \log(\min(A, B))) = O(Q \times (R[i] - L[i]) \times \log(\min(A, B)))$, dan merupakan solusi untuk mendapatkan nilai penuh di soal ini.

3 1C : Ojek Daring

Soal ini ditulis oleh Agus Sentosa Hermawan dan Ashar Fuadi.

3.1 Subsoal 3

Pada subsoal ini, semua panjang jalan, maksimal jarak, dan harga menaiki setiap jenis ojek sama dengan satu. Karena biaya perpindahan dari suatu kota ke kota lain sama dengan satu (apabila terdapat jalan), kita dapat menyelesaikan subsoal ini menggunakan *Breadth-First Search* dengan kompleksitas waktu $O(V + E)$.

3.2 Subsoal 4

Pada subsoal ini, maksimal jarak masing-masing jenis ojek sama dengan satu. Oleh karena itu, harga untuk melintasi sebuah jalan diantara dua kota dapat dibagi menjadi dua kasus:

1. Jalan tidak dikuasai ojek pangkalan Pada kasus ini, harga untuk melintasi sebuah jalan dengan panjang K adalah $K \times min(C_p, C_d)$
2. Jalan dikuasai ojek pangkalan Pada kasus ini, harga untuk melintasi sebuah jalan dengan panjang K adalah $min(K * C_p, C_d + (K - 1) * C_p)$

Perhatikan bahwa perbedaan subsoal ini dan sebelumnya hanya terletak pada harga yang harus dibayarkan untuk melintasi tiap jalan. Untuk dapat mengerjakan varian ini, kita dapat menggunakan *Dijkstra's Algorithm* dengan kompleksitas waktu $O(E \log V)$.

3.3 Subsoal 5

Perhatikan bahwa pada subsoal ini panjang setiap jalan ≤ 10. Oleh karena itu, meskipun kita memecah setiap jalan menjadi jalan-jalan dengan panjang satu (dengan membuat kota-kota dummy di sepanjang jalan), banyaknya node setelah pemecahan hanya $\leq N \times 10 = 1000$. Setelah memecah jalan, perhatikan bahwa mungkin saja kita menggunakan satu ojek untuk melewati beberapa kota sekaligus. Untuk itu, kita harus mencatat jarak sisa ojek yang sedang digunakan dalam *state* Dijkstra kita. Transisi antara satu kota dan kota lain dapat dilakukan dengan hanya mencoba dua kasus:

1. Tetap menggunakan ojek sekarang apabila sisa jarak > 0
2. Menggunakan ojek baru (baik daring maupun pangkalan)

Kompleksitas waktu solusi ini adalah $O(EKM \log(VKM))$ yang mana $M = min(M_d, M_p)$ dan $K = max(K_i)$.

3.4 Subsoal 6

Trik-trik dari subsoal sebelumnya sayangnya tidak dapat digunakan pada subsoal ini karena batasan memori (subsoal 5) atau karena menghasilkan jawaban yang salah (subsoal 1-4). Akan tetapi, kita masih dapat menggunakan sebagian solusi subsoal 5.

Seperti subsoal 5, kita akan mencatat sisa jarak ojek yang sedang digunakan dalam *state* Dijkstra. Karena kita tidak dapat memecah jalan menjadi jalan-jalan dengan panjang satu, transisinya akan menjadi lebih rumit, karena terdapat banyak strategi untuk melewati jalan-jalan tersebut yang memiliki harga dan sisa jarak ojek yang berbeda-beda. Selain itu, perhatikan pula bahwa sisa jarak yang perlu kita perhatikan hanyalah sisa jarak ojek pangkalan, karena kita selalu bisa menggunakan ojek daring baru dari setiap kota.

Perhatikan bahwa kemungkinan sisa jarak ojek $\leq M = min(M_d, M_p)$. Untuk itu, kita harus mencoba semua kemungkinan sisa jarak di kota tujuan dan menghitung harga paling murah untuk mendapatkan sisa jarak tersebut. Untuk menghitung harga paling murah tersebut, terdapat beberapa kasus:

1. Gunakan ojek pangkalan sampai akhir
2. Gunakan ojek daring sampai hampir sampai ke kota tujuan, lalu gunakan ojek pangkalan untuk mendapatkan jarak sisa
3. Gunakan ojek daring sampai kota tujuan apabila sisa jarak sama dengan nol

Kompleksitas waktu solusi ini adalah $O(EM^2 \log(VM))$.

3.5 Subsoal 7

Transisi solusi subsoal 6 dapat dipercepat dengan menggunakan strategi yang berbeda untuk melalui jalan yang dikuasai / tidak dikuasai ojek pangkalan.

1. Jalan yang dikuasai ojek pangkalan

   Pada kasus ini, kita bisa memilih untuk menggunakan atau tidak menggunakan sisa jarak ojek sekarang.

   Apabila kita ingin menggunakan sisa jarak ojek sekarang, jarak sisanya hanya bisa ditempuh menggunakan ojek pangkalan.

   Apabila kita tidak ingin menggunakan sisa ojek sekarang, kita tetap harus mencoba semua kemungkinan sisa jarak di kota tujuan. Akan tetapi, perhatikan bahwa hal ini hanya perlu dilakukan satu kali saat mendapatkan harga termurah menuju setiap kota (karena sisa jarak ojek tidak dipakai). Harga termurah ini bisa didapat saat kita *pop* suatu kota pertama kali dari *heap Dijkstra* kita.

   Total kompleksitas waktu pada kasus ini adalah $O(E_{dikuasai}Mlog(VM))$

2. Jalan tidak dikuasai ojek pangkalan

   Pada kasus ini, M_d menjadi tidak penting lagi (karena kita selalu bisa melanjutkan dengan ojek daring). Sehingga, terdapat dua macam langkah yang bisa kita lakukan *di sepanjang jalan* tersebut:

   (a) Langkah dengan panjang 1, harga = C_d

   (b) Langkah dengan panjang M_p, harga = $min(C_d \times M_p, C_p)$

   Sekarang, perhatikan bahwa kita hanya dapat melakukan langkah (a) $M_p - 1$ kali (dan menggunakan langkah (b) sisanya. Terdapat juga beberapa kasus khusus di ujung jalan yang harus ditangani secara terpisah, namun hal ini tidak mengubah ide besar solusi subsoal ini). Sebagai petunjuk implementasi, kita dapat menggunakan kembali *Heap Dijkstra* untuk mencatat *state* yang menghitung banyaknya langkah (a) yang telah dilakukan.

   Total kompleksitas waktu pada kasus ini adalah $O(E_{tidak\ dikuasai}Mlog(VM))$

Dengan menggabungkan dua kasus diatas, kita dapat menyelesaikan subsoal 7 dengan kompleksitas waktu $O(EM \log(VM))$

4 2A : Hiasan Atap

Soal ini ditulis bersama oleh seluruh Tim SC OSN 2018.

4.1 Subsoal 3

Di subsoal ini $K = 1$, dan $|S| \leq 1000$. Subsoal ini dapat diselesaikan dengan melakukan *brute force* setiap petak, kemudian simpan jawaban untuk setiap kemungkinan $L[i]$ dan $R[i]$. Generate semua kemungkinan jawaban diperlukan waktu $O(N^2)$ dan setiap query membutuhkan waktu $O(1)$ sehingga kompleksitas total adalah $O(N^2 + Q)$.

4.2 Subsoal 4

Berhubung pada subsoal ini $Q = 1$ dan $K = 1$. Kita cukup meng-generate tinggi dari semua petak, dan lakukan iterasi untuk mencari nilai maksimal dari satu-satunya query sehingga kompleksitas total dari solusi untuk subsoal ini adalah $O(N + NQ) = O(NQ)$

4.3 Subsoal 5

Pada subsoal ini $Q = 1$.

Observasi 4.1. *Banyaknya tumpukan atap akan meningkat dari pinggir menuju ke tengah.*

Pola dari penambahan tinggi atap pada setiap kolom pasti akan berulang setiap $|S|$ kali penambahan. Sehingga kita dapat mencari rentang terpendek L' dan R' yang mana kedua bilangan tersebut adalah rentang yang terjadi setelah penambahan ke $k \times |S|$ dimana $0 \leq k \leq K$ dan $L' \leq L \leq R \leq R'$. Kemudian, kita iterasi string S dan mengubah memperkecil rentang dari L' dan R' selama pertidaksamaan $0 \leq k \leq K$ dan $L' \leq R$ dan $L' \leq R$ masih terpenuhi sehingga kompleksitasnya adalah $O(|S|Q)$.

4.4 Subsoal 6

Dalam subsoal ini $|S| \leq 2$. Pengerjaan subsoal ini dapet dibagi menjadi 2 kasus yaitu:

1. $S \in \{"AA", "BB", "A", "B"\}$. Dalam kasus ini, banyaknya atap tiap kolom akan monoton naik atau monoton turun sebanyak 1 dari kolom yang sebelumnya. Sehingga untuk setiap posisi x, banyaknya atap di posisi tersebut adalah x atau $N + 1 - x$. Sehingga kompleksitas dari kasus ini adalah $O(Q)$
2. $S \in \{"AB", "BA"\}$.

   Observasi 4.2. *Jika banyaknya karakter A dan B adalah sama, maka banyaknya atap yang ditambahkan dengan rata kiri dan banyaknya atap yang ditambahkan pada rata kanan sama, maka kolom tertinggi pasti ada di tengah, atau berada di posisi* $\frac{N+1}{2}$

   Sehingga kita dapat membaginya menjadi 2 kasus lagi

   (a) $L \leq \frac{N+1}{2} \leq R$. Dalam kasus ini jawabannya pasti N

   (b) $L > \frac{N+1}{2}$ atau $\frac{N+1}{2} > R$. Dalam kasus iini mirip dengan kasus sebelumnya, namun setiap kolom sebelum kolom selain kolom tertinggi memiliki selisih 2 dari kolom di sebelahnya. Kompleksitas kasus ini juga $O(Q)$.

Total kompleksitas untuk kasus ini adalah $O(Q)$.

4.5 Subsoal 7

Pada subsoal ini, $L = R$. Kita cukup mengetahui berapa tinggi dari suatu posisi. Misalkan $pos = L = R$. Misalkan tinggi atap di kolom ke-i dinyatakan dengan $h[i]$.

Observasi 4.3. *misalkan n_A adalah banyaknya karakter 'A' pada string S, n_B adalah banyaknya karakter 'B' pada string S, dan z adalah posisi kolom tertinggi, maka untuk setiap kolom x, y dimana:*

1. *$x, y \leq z$ dan $y = x + n_B$, maka $h[y] = h[x] + |S|$*
2. *$x, y \geq z$ dan $y = x + n_A$, maka $h[y] = h[x] - |S|$*

Sehingga untuk setiap query, lakukan binary search berapa kali n_A atau n_B yang ditambahkan untuk mencapai posisi tersebut dari posisi awal, kemudian lakukan lagi binary search untuk mencari berapa karakter atap lagi yang ditambahkan untuk mencapai posisi kolom yang dicari. Kompleksitas dari subsoal ini adalah $O(Q \log(K) \log(|S|))$.

4.6 Subsoal 8

Tidak ada batasan tambahan pada subsoal ini. Sama seperti subsoal 7, misalkan tinggi atap di kolom ke-i dinyatakan dengan $h[i]$. Misalkan posisi z adalah kolom tertinggi. Dengan observasi pada subsoal 7 dan subsoal 6, maka untuk mengerjakan subsoal ini dapat dibagi menjadi 3 kasus:

1. $L \leq z \leq R$. Maka jawabannya adalah N.
2. $R \leq z$. Maka jawabannya adalah $h[R]$.
3. $z \leq L$. Maka jawabannya adalah $h[L]$.

sehingga kita cukup mencari $h[L]$, $h[R]$, dan/atau $h[z]$, dan kompleksitas subsoal ini adalah $O(Q \log(K) \log(|S|))$.

5 2B : Permainan Pajel

Soal ini ditulis oleh Ammar Fathin Sabili.

Soal ini berupa soal kreatif di mana program peserta akan dibandingkan dengan program juri untuk penilaiannya.

5.1 Kasus Uji

Terdapat 10 buah kasus uji pada soal ini.

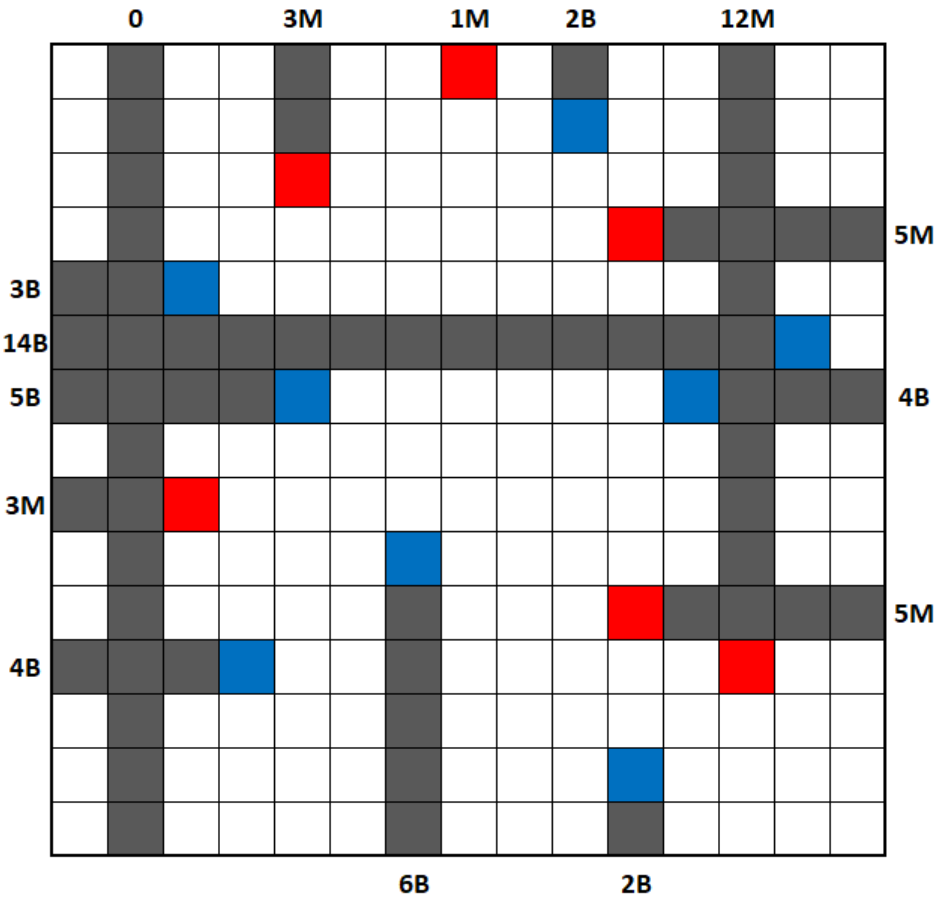
- Kasus uji 1 dan 2 merupakan kasus uji kecil yang dapat diselesaikan dengan tangan.
- Kasus uji 3 dan 4 merupakan kasus uji yang sedikit lebih besar dari kasus uji 1 dan 2 yang juga dapat diselesaikan dengan tangan dan juga dapat digunakan untuk diamati keluarannya apabila digunakan program.
- Kasus uji 5 digunakan untuk observasi *greedy* yang akan dijelaskan pada Solusi Bagian 2.
- Kasus uji 6 dan 8 digunakan untuk melakukan tes untuk Solusi Bagian 3.
- Kasus uji 7 digunakan untuk mengamati beberapa tricky case.
- Kasus uji 9 dan 10 digunakan untuk papan permainan yang besar.

5.2 Solusi Bagian 1: Mendapatkan Nilai di Semua Kasus Uji (30 Poin)

Dimisalkan petak-petak yang kosong dan masih dapat diwarnai disimbolkan dengan petak putih, petak-petak yang harus kosong disimbolkan dengan petak hitam, dan petak-petak yang sudah diwarnai disimbolkan dengan petak biru maupun merah.

Perhatikan bahwa untuk petunjuk ”0”, kita hanya perlu memberi simbol hitam pada baris atau kolom tersebut. Sedangkan untuk petunjuk yang mengacu pada suatu pewarnaan, misalnya ”12M”, maka pada 11 sel pertama baris/kolom tersebut kita dapat beri simbol hitam dan sel ke-12 dapat diberi simbol warna merah.

Berikut merupakan contoh pelabelan simbol-simbol berdasarkan petunjuk yang diberikan.



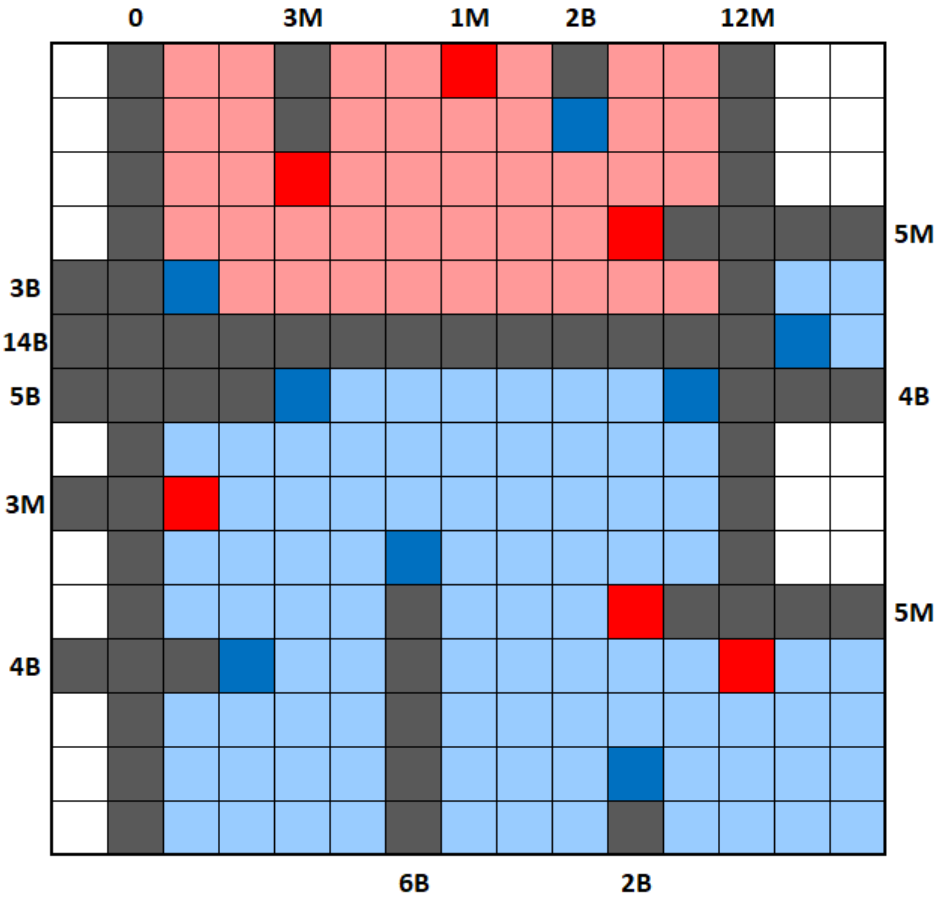
Untuk menghasilkan keluaran dengan nilai di semua kasus uji, kita dapat mengeluarkan keluaran dari pendekatan tersebut dengan simbol hitam maupun putih adalah kosong. Perhatikan bahwa setiap petak berwarna merupakan wilayah-wilayah yang berbeda sehingga terdapat banyak wilayah (dan kita mendapatkan poin yang tidak banyak).

5.3 Solusi Bagian 2: Pendekatan Greedy (60 Poin)

Dari pendekatan sebelumnya, perhatikan bahwa kita dapat membagi petak-petak menjadi beberapa "area" yang dipisahkan oleh petak-petak hitam. Untuk setiap area, kita dapat kerjakan secara terpisah. Hal yang dapat kita lakukan adalah mewarnai simbol-simbol putih dengan merah atau biru sehingga banyaknya wilayah seminimal mungkin.

Salah satu pendekatan yang dapat dilakukan adalah dengan mewarnai seluruh warna putih di suatu area dengan warna merah saja atau biru saja (atau dibiarkan putih apabila tidak ada petak merah maupun biru di area tersebut). Dengan pendekatan ini, seluruh petak berwarna merah atau biru di area tersebut akan terhubung menjadi sebuah wilayah yang besar. Dengan demikian, banyaknya wilayah akan menjadi semakin sedikit.

Berikut merupakan contoh pewarnaan dengan pendekatan greedy tersebut.

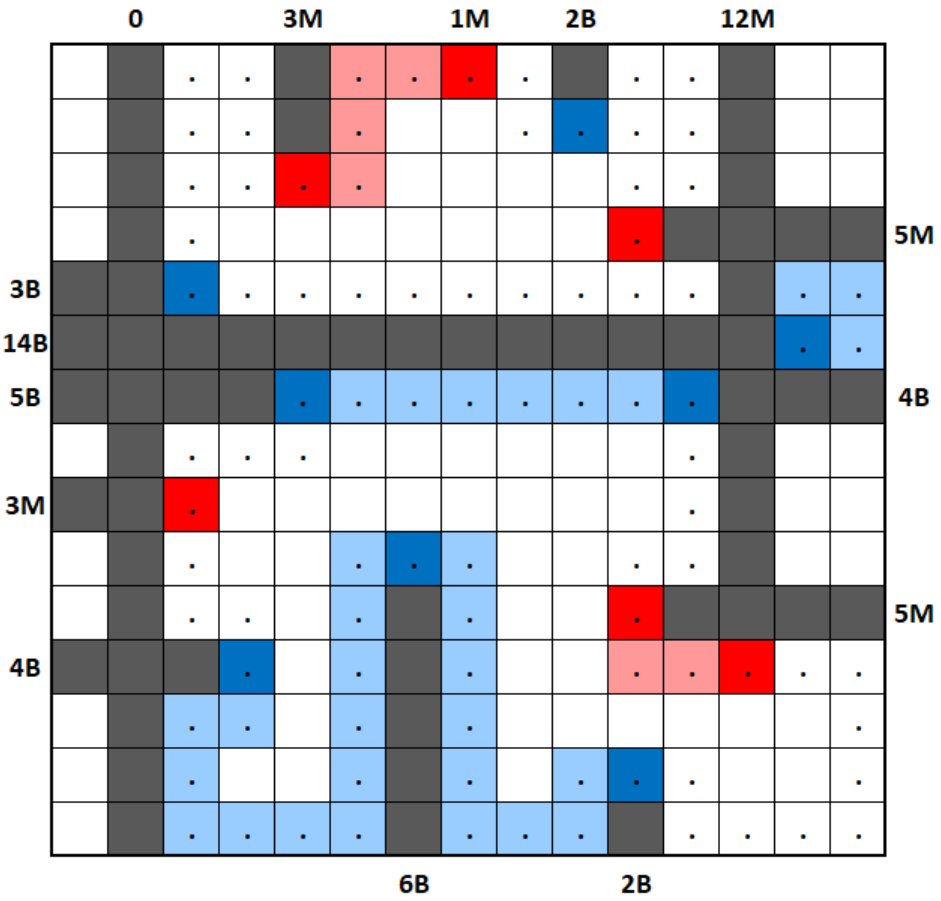


5.4 Solusi Bagian 3: Solusi Optimal (100 Poin)

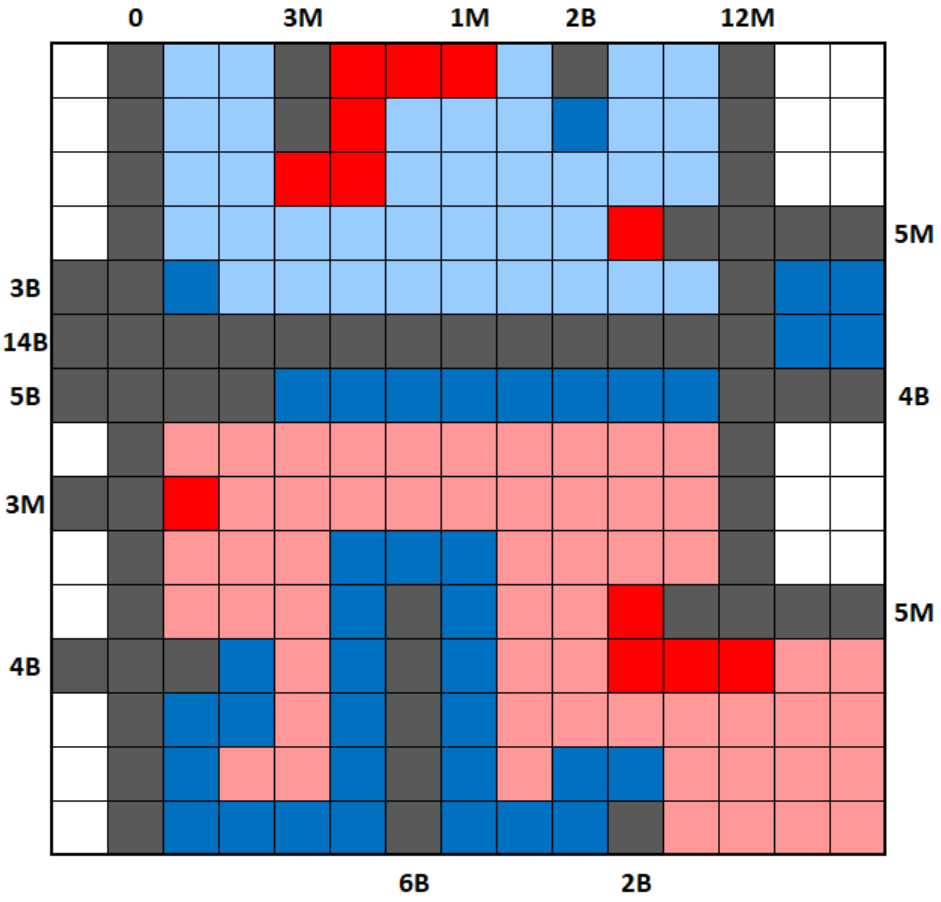
Sebelum beranjak ke solusi optimal, catat bahwa mendapatkan 11 poin di sebuah kasus uji tidaklah mungkin. Terdapat sebuah algoritma deterministik yang dapat digunakan untuk mencari solusi optimal (?) dengan kompleksitas $O(N^2)$.

Dari pendekatan pada Solusi Bagian 1, perhatikan bahwa setiap petak berwarna pasti bersebelahan dengan petak hitam. Dengan kata lain, petak-petak berwarna berada pada sekeliling area. Karena hal tersebut, kita dapat menghubungkan dua petak berwarna sama yang berdekatan melewati sekeliling area. Dengan kata lain, apabila kita memutari suatu area dan mendapati dua petak berwarna yang sama, maka kita dapat mewarnai petak-petak sepanjang jalan yang menghubungkan kedua petak tersebut dengan warnanya.

Berikut merupakan contoh pewarnaan dengan menghubungkan petak-petak berwarna sama yang berdekatan saat memutari suatu area.



Dengan menggabungkan pendekatan ini dan pendekatan greedy (mewarnai sisa petak-petak putih dengan sebuah warna yang sama), didapatlah solusi optimal.



5.5 Lebih Lanjut mengenai Solusi Optimal

Untuk membuktikan bahwa dasar pendekatan pada Solusi Bagian 3 adalah solusi optimal, perhatikan bahwa setelah kita menggabungkan warna-warna yang sama di sekeliling area, kita akan mendapati bahwa sekeliling area terbentuk beberapa wilayah yang secara bergantian adalah merah, biru, merah, biru, dan seterusnya. Dimisalkan terdapat $2N$ wilayah yang berarti terdapat N pasang merah dan biru di sekeliling area. Kita tidak dapat menggabungkan wilayah-wilayah tersebut menjadi N atau kurang wilayah berbeda. Hal ini dikarenakan menggabungkan dua wilayah akan menyebabkan setidaknya sebuah wilayah lain tidak dapat digabungkan dengan wilayah lainnya. Karena paling banyak dilakukan $N - 1$ penggabungan, maka paling sedikit terdapat $2N - (N - 1) = N + 1$ wilayah di area tersebut. Untuk mendapatkan $N + 1$ wilayah ini, dapat digunakan pendekatan greedy yang menggabungkan seluruh warna yang sama menjadi 1 wilayah besar dan meninggalkan N warna lainnya menjadi wilayah masing-masing.

Perhatikan bahwa terdapat beberapa isu teknis dengan solusi ini misalnya apabila saat memutari area terdapat petak yang diwarnai lebih dari sekali dengan warna yang berbeda. Hal ini dapat diatasi dengan memilih warna yang menyebabkan banyaknya wilayah sesedikit mungkin. Kasus ini dimasukkan ke dalam kasus uji 7.

6 2C : Festival FPB

Soal ini ditulis oleh Muhammad Ayaz Dzulfikar.

Untuk menyelesaikan sebagian besar subsoal, kita perlu algoritma untuk mencari faktor persekutuan terbesar (FPB) yaitu Algoritma Euclidean. Untuk mempermudah diskusi, mari kita gunakan beberapa definisi berikut:

Definisi 6.1. *$fpb(x, y)$ menyatakan FPB dari x dan y.*

Definisi 6.2. *$g(l, r)$ menyatakan FPB dari $A[l], A[l+1], \cdots, A[r]$.*

Definisi 6.3. *Suatu pembagian kelompok dinyatakan sebagai $[(L_1, R_1), (L_2, R_2), \cdots, (L_K, R_K)]$, yang mana:*

- *(L_i, R_i) menyatakan bahwa kelompok ke-i terdiri dari rumah ke-L_i, rumah ke-L_{i+1}, hingga rumah ke-R_i*
- *$R_i < L_{i+1}$ untuk setiap $1 \leq i < K$*

6.1 Subsoal 3

Subsoal ini dapat diselesaikan dengan melakukan *complete search*, yang mana untuk setiap rumah, terdapat 3 kemungkinan:

1. Rumah tersebut tidak termasuk kelompok manapun
2. Rumah tersebut merupakan rumah terkiri (memiliki indeks terkecil) pada sebuah kelompok
3. Rumah tersebut berada di kelompok yang sama dengan rumah di kirinya, seandainya rumah di kirinya termasuk dalam suatu kelompok

Kompleksitas dari solusi ini adalah $O(3^N \log(\max(A[i])))$.

6.2 Subsoal 4

Perhatikan bahwa:

$$g(l, r) = \begin{cases} A[l] & l = r \\ fpb(A[r], g(l, r-1)) & l < r \end{cases}$$

Maka, seluruh nilai $g(l, r)$ dapat dihitung dalam $O(N^2 \times \log(\max(A[i]))$.

Subsoal ini dapat diselesaikan dengan memanfaatkan formula tersebut. Perhatikan bahwa karena $K = 1$, jawaban untuk subsoal ini sama saja dengan:

$$\sum_{l=1}^{n} \sum_{r=l}^{n} g(l, r)$$

Kompleksitas dari solusi ini adalah $O(N^2 \log(\max(A[i])))$.

6.3 Subsoal 5

Untuk menyelesaikan subsoal ini, kita perlu memandang penghitungan jawaban menggunakan sudut pandang lain. Ketimbang menghitung jumlahan FPB untuk setiap pembagian kelompok, kita bisa saja menghitung untuk tiap (l, r), ada berapa banyak pembagian kelompok yang memiliki suatu indeks i yang mana $(L_i, R_i) = (l, r)$.

Observasi 6.1. *Banyaknya nilai $g(l, r)$ dihitung di jawaban akhir sama dengan banyak pembagian kelompok yang memiliki suatu indeks i yang mana $(L_i, R_i) = (l, r)$. Definisikan $ways(l, r)$ sebagai banyaknya pembagian kelompok yang memiliki indeks i yang mana $(L_i, R_i) = (l, r)$. Maka, jawaban akhir dapat dihitung sebagai:*

$$\sum_{l=1}^{n} \sum_{r=l}^{n} g(l, r) \times ways(l, r)$$

Subsoal ini dapat diselesaikan dengan memanfaatkan observasi di atas. Perhatikan bahwa:

- Untuk $K = 1$, $ways(l, r)$ selalu bernilai 1
- Untuk $K = 2$, $ways(l, r)$ bernilai $\binom{N-r+1}{2} + \binom{l}{2}$. Nilai tersebut didapat dari kasus $(L_1, R_1) = (l, r)$ dan kasus $(L_2, R_2) = (l, r)$

Selanjutnya, kita cukup menghitung jawaban akhir dengan formula dari observasi.
Kompleksitas dari solusi ini adalah $O(N^2 \log(\max(A[i])))$.

6.4 Subsoal 8

Berhubung seluruh $A[i]$ bernilai sama, maka jumlahan FPB untuk tiap pembagian kelompok pasti $K \times A[1]$. Oleh karena itu, untuk subsoal ini, kita hanya perlu menghitung ada berapa banyak cara pembagian kelompok.

Definisikan $f(n, k, x)$ sebagai banyaknya pembagian n rumah menjadi k kelompok, dengan x sebagai *boolean* yang menyatakan apakah rumah ke-$(n+1)$ termasuk dalam suatu kelompok atau tidak. Maka, kita memiliki formula berikut:

$$f(n,k,x) = \begin{cases} 0 & (n < 0 \wedge k > 0) \\ 1 & (n = 0 \wedge k = 0) \\ f(n-1,k,0) + f(n-1,k-1,1) & x = 0 \\ f(n-1,k,0) + f(n-1,k-1,1) + f(n-1,k,1) & x = 1 \end{cases}$$

Untuk mempercepat penghitungan, kita dapat menyimpan hasil dari tiap $f(n, k, x)$ yang sudah kita pernah kita hitung (teknik ini biasa disebut dynamic programming (DP)). Banyaknya pembagian kelompok yang kita inginkan didapat dari $f(N, K, 0)$. Sehingga, jawaban akhir adalah $K \times A[1] \times f(N, K, 0)$.

Kompleksitas dari solusi ini adalah $O(NK)$.

6.5 Subsoal 6

Subsoal ini dapat diselesaikan dengan menggabungkan solusi subsoal 5 dan subsoal 8. Perhatikan bahwa $f(n, k, x)$ yang digunakan di subsoal 8 dapat digunakan untuk menghitung $ways(l, r)$ yang digunakan di subsoal 5.

Observasi 6.2. *Dalam menghitung $ways(l, r)$, kita dapat melakukan brute force untuk menentukan indeks i, sehingga $(L_i, R_i) = (l, r)$. Maka:*

$$ways(l,r) = \sum_{i=1}^{K} f(l-1, i-1, 0) \times f(N-r, K-i, 0)$$

Maka, $ways(l, r)$ dapat dihitung dalam $O(K)$. Menggabungkan seluruhnya, subsoal ini dapat diselesaikan dalam $O(N^2 log(max(A[i])) + NK + N^2K) = O(N^2 \times (\log(\max(A[i])) + K))$.

Kompleksitas dari solusi ini adalah $O(N^2 \times (K + \log(\max(A[i]))))$.

6.6 Subsoal 7

Kita kembali ke solusi subsoal 4. Perhatikan bahwa terdapat *bottleneck* di penghitungan $g(l, r)$, yang menyebabkan munculnya N^2 di kompleksitas. Untuk menyelesaikan subsoal ini, kita perlu mempercepat penghitungan tersebut.

Lemma 6.1. *Untuk suatu nilai r yang tetap, terdapat $O(\log(A[r]))$ kemungkinan nilai berbeda dari $g(i, r)$.*

Bukti. Untuk suatu nilai x, hasil $fpb(x, y)$ tentunya merupakan x atau faktor dari x. Terdapat $O(\log(x))$ faktor prima dari x, sehingga hasil pemberlakuan $fpb(x, y)$ beberapa kali tentunya menghasilkan paling banyak $O(\log(x))$ nilai berbeda. □

Perhatikan bahwa hanya terdapat $O(Nlog(max(A[i])))$ nilai berbeda dari $g(l, r)$, sehingga apabila nilai-nilai tersebut diketahui beserta muncul berapa kali, kita dapat mendapatkan jawaban akhir dengan lebih cepat.

Observasi 6.3. *Definisikan $S(r)$ sebagai himpunan triplet (v, a, b), yang menandakan bahwa nilai $g(a, r) = g(a+1, r) = \cdots = g(b, r) = v$. Berdasarkan lemma sebelumnya, $|S(r)| = O(\log(A[r]))$. Perhatikan bahwa dengan memanfaatkan $S(r)$, kita bisa mendapatkan $S(r+1)$ dengan mengganti tiap v dengan $fpb(v, A[r+1])$, lalu mengelompokkan nilai-nilai yang sama.*

Selanjutnya, menggunakan definisi $S(r)$, kita dapat memperoleh jawaban akhir melalui formula berikut:

$$\sum_{i=1}^{N} \sum_{(v,a,b) \in S(i)} v \times (b - a + 1)$$

Kompleksitas untuk solusi ini adalah $O(N \log(\max(A[i])))$.

6.7 Subsoal 9

Solusi dari subsoal ini adalah penggabungan dari seluruh subsoal sebelumnya.

Pada subsoal 7, penghitungan kontribusi (v, a, b) dapat dilakukan dengan mudah karena $K = 1$. Untuk $K > 1$, kita menggunakan formula dari subsoal 5, yaitu $g(l, r) \times ways(l, r)$. Dalam hal ini, kontribusinya adalah:

$$\sum_{l=a}^{b} v \times ways(l, r)$$

Mari bongkar formula tersebut:

$$\sum_{l=a}^{b} v \times ways(l, r)$$

$$v \times \sum_{l=a}^{b} ways(l, r) \qquad \text{(Keluarkan v)}$$

$$v \times \sum_{l=a}^{b} \sum_{i=1}^{K} f(l-1, i-1, 0) \times f(N-r, K-i, 0) \qquad \text{(Bongkar ways(l, r))}$$

$$v \times \sum_{i=1}^{K} \sum_{l=a}^{b} f(l-1, i-1, 0) \times f(N-r, K-i, 0) \qquad \text{(Tukar urutan sigma)}$$

$$v \times \sum_{i=1}^{K} (f(N-r, K-i, 0) \times \sum_{l=a}^{b} f(l-1, i-1, 0)) \qquad \text{(Keluarkan f(N - r, K - i, 0))}$$

Bottleneck terdapat di penjumlahan yang paling dalam, yakni $\sum_{l=a}^{b} f(l-1, i-1, 0)$ yang penghitungannya memakan waktu $O(N)$. Namun, perhatikan bahwa parameter hanya berubah di l. Oleh karena itu, kita bisa mempercepat penghitungan dengan membangun prefix sum dari fungsi $f(n, k, x)$ untuk setiap kemungkinan nilai k. Selanjutnya, $\sum_{l=a}^{b} f(l-1, i-1, 0)$ dapat dihitung dalam $O(1)$ dengan memanfaatkan prefix sum tersebut.

Sehingga, solusi penuh untuk soal ini dapat dilakukan dengan:

1. Mencari $S(r)$, untuk setiap $1 \leq r \leq N$
2. Membangun prefix sum dari fungsi $f(n, k, x)$, yang mana fungsi tersebut dihitung menggunakan DP
3. Menghitung jawaban akhir dengan memanfaatkan formula di atas dan prefix sum yang telah dibangun

Kompleksitas solusi ini adalah $O(NK \log(\max(A[i])))$.