Editorial Babak Final CP Recursion 1.0

Editorial ini disusun dengan tujuan untuk memberikan penjelasan menyeluruh terhadap solusi dari tiap soal. Beberapa pendekatan yang dijelaskan mungkin berbeda dari solusi peserta, namun itulah bagian dari kekayaan dunia competitive programming. Tiap soal dirancang dengan variasi tingkat kesulitan, dari yang easy, medium, hingga hard secara konsep dan implementasi. Kami berharap editorial ini tidak hanya menjadi kunci jawaban, namun juga dapat menjadi bahan belajar bagi pembaca yang ingin meningkatkan skill problem solving-nya.

Judul Soal		**Penulis**
A	Anomali	Zulfahmi
B	Bilangan Baik	Benediktus Gianto Jarod
C	Ciro dan Roti	Zulfahmi
D	Dadu dan Keberuntungan	Benediktus Gianto Jarod
E	Es Pisang Ijo	Zulfahmi
F	FT Unhas 09	Zulfahmi
G	Gado Gado	Zulfahmi
H	Himmel Pasti Akan Melakukan Hal Yang Sama	Zulfahmi
I	Indomi Soto	Zulfahmi
J	Jalangkote	King Ahmad Ilyas
K	Krisis Air	Benediktus Gianto Jarod
L	Lembaga Desa Kucing	Zulfahmi

Terima kasih kami ucapkan kepada seluruh peserta yang telah berpartisipasi, serta kepada **Abdul Malik Nurrokhman** yang telah berkontribusi sebagai tester dalam memberikan kritik dan saran serta memverifikasi kualitas terhadap soal – soal Babak Final Competitive Programming Recursion 1.0.

Terima kasih yang sebesar besarnya juga kami ucapkan terhadap anggota Divisi Competitive Programming Recursion 1.0, yaitu:

- **Zulfahmi**
- **King Ahmad Ilyas**
- **Benediktus Gianto Jarod**.

Dan kepada seluruh panitisa Recursion 1.0 yang bertugas sehingga acara dapat terlaksana dengan baik.

Akhir kata, semoga editorial ini bermanfaat dan bisa menjadi sumber belajar bagi para pembaca. Sampai jumpa di Recursion 2.0!!!

Competitive Programming – Editorial

A. Anomali

Tags : *Dynamic Programming, Tree, Number Theory*

Misalkan untuk sebuah tree T dan bilangan M, $C(T, M)$ menyatakan banyaknya cara memilih nilai pada setiap nodes sehingga $F(1, u) = M$ untuk setiap $u \in leaf$.

Misalkan faktorisasi prima $M = p_1^{e_1} \times p_2^{e_2} \times \ldots \times p_K^{e_K}$, maka $A_u = p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_K^{a_K}$ dengan $0 \leq a_i \leq e_i$, karena faktor prima selain $p_1, p_2, \ldots, p_k$ tidak akan berkontribusi pada jawaban. Sehingga didapatkan

$$C(T, M) = F\left(C, p_1^{e_1}\right) \times C\left(T, p_2^{e_2}\right) \times \ldots \times C\left(T, p_K^{e_K}\right)$$

Kita akan mencari $C(T, p^e)$ secara terpisah untuk setiap faktorisasi prima dengan $A_u = 1, p, p^2, \ldots, p^e$. Misalkan T_u menyatakan subtree pada root u dan $dp[u][a] = C(T_u, p^a)$.

Awalnya $dp[u][a] = 0$ dengan $1 \leq a \leq e$ untuk setiap $1 \leq u \leq N$.

Base case: $dp[u][a] = 1$ dengan $1 \leq a \leq e$ untuk setiap $u \in leaf$. Selain itu,

1. Jika $A_u = p^a$, maka bilangan pada setiap subtree T_u kecuali u, dapat berupa apapun di antara nilai $1, p, p^2, \ldots, p^a$, maka
$$dp[u][a] = dp[u][a] + (a+1)^{sub[u]-1}$$
dimana $sub[u]$ adalah banyaknya nodes dalam T_u.
2. Jika $A_u = 1, p, p^2, \ldots, p^{a-1}$, maka kita harus menambahkan $F(T_v, a)$ untuk $v \in child(u)$,
$$dp[u][a] = dp[u][a] + a \times \left(\prod_{v \in child(u)} dp[v][a] \right)$$

Dengan $M = p_1^{e_1} \times p_2^{e_2} \times \ldots \times p_K^{e_K}$, Jawabannya adalah

$$\prod_{e \in (e_1, e_2, \ldots, e_K)} dp[1][e]$$

Kompleksitas Waktu : $O(N \log M)$

B. Bilangan Baik

Tags : *Binary Search, Number Theory*

Perhatikan bahwa $148 = 2^2 \times 37$. Misalkan y merupakan bilangan baik. Agar y habis di bagi 148, maka y harus berbentuk:

$$y = 2^{q_1} \times 37^{q_2} \times {p_3}^{q_3} \times {p_4}^{q_4} \times {p_5}^{q_5} \times \ldots \times {p_n}^{q_n}$$

Dimana p_i adalah bilangan prima untuk $1 \leq i \leq n$ dengan $q_1 \geq 2$ dan $q_2 \geq 1$

Mudah dilihat bahwa banyak faktor dari y adalah $(q_1 + 1) \times (q_2 + 1) \times (q_3 + 1) \times \ldots \times (q_n + 1)$ atau

$$\prod_{i=1}^{n} (q_i + 1)$$

Karena y memiliki tepat 148 faktor maka $\prod_{i=1}^{n} (q_i + 1) = 148$.

Asumsikan $n \geq 4$, karena $q_1 \geq 2$, maka $q_1 + 1 \geq 3$. Perhatikan bahwa 148 hanya memiliki 6 faktor, yakni $1, 2, 4, 37, 74, 148$. Jelas, bahwa $y < 74$ sebab $2^{73} \times 37 > 10^{18}$. Ini artinya, hanya ada 2 kemungkinan untuk q_1, yakni $q_1 = 3$ dan $q_1 = 36$.

1. Kasus 1: $q_1 = 3$

   $$(q_2 + 1) \times (q_3 + 1) \times .. \times (q_n + 1) = 37$$

   Karena 37 merupakan bilangan prima dan $n \geq 4$, maka tidak ada solusi untuk y.
2. Kasus 2: $q_1 = 36$

   $$(q_2 + 1) \times (q_3 + 1) \times .. \times (q_n + 1) = 4$$

   Karena $n \geq 4$, maka $q_i + 1 \geq 2$ untuk $2 \leq i \leq 4$, artinya, $(q_2 + 1) \times (q_3 + 1) \times .. \times (q_n + 1) \geq 8$. Yang dimana adalah suatu kontradiksi.

Jadi, $n \geq 4$ tidak memiliki solusi untuk y, dan kemungkinan yang tersisa adalah $n = 2$ dan $n = 3$.

1. Kasus 1: $n = 2$

   seperti yang sudah diperlihatkan tadi bahwa hanya ada dua kemungkinan untuk q_1.

   a) Subkasus 1: $q_1 = 3$

   $$q_2 + 1 = 37$$
   $$q_2 = 36$$
   $$y = 2^3 \times 37^{36}$$
   $$y > 10^{18} \; (Tidak\ ada\ solusi)$$

   b) Subkasus 2: $q_1 = 36$,

   $$q_2 + 1 = 4$$
   $$q_2 = 3$$
   $$y = 2^{36} \times 37^3$$
   $$y = 2480847655108608$$

   Jadi, untuk $n = 2$, diperoleh 1 solusi

2. Kasus $n = 3$

   seperti yang sudah diperlihatkan tadi bahwa hanya ada dua kemungkinan untuk q_1

   a) Subkasus 1: $q_1 = 3$

   $$(q_2 + 1) \times (q_3 + 1) = 37$$

   Karena 37 merupakan bilangan prima serta $q_2 \geq 1 \; dan \; q_3 \geq 1$, maka tidak ada solusi.

   b) Subkasus 2: $q_1 = 36$

   $$(q_2 + 1) \times (q_3 + 1) = 4$$

   Karena $q_2 \geq 1 \; dan \; q_3 \geq 1$, maka hanya ada satu kemungkinan yakni $q_2 = 1$ dan $q_3 = 1$. Sehingga, $y = 2^{36} \times 37 \times p_3$ dimana $p \geq 3$

   $$y < 10^{18}$$

   $$2^{36} \times 37 \times p_3 < 10^{18}$$

   $$p_3 < 393296$$

   Jadi untuk mencari solusi y, kita hanya perlu mencari semua bilangan prima yang lebih dari 3 dan kurang dari 393296.

Kita bisa menggunakan algortima sieve of Eratosthenes, untuk mencari bilangan prima dari 1 hingga 393296. Kemudian membuat prefix sum untuk mencari jumlah bilangan baik dari 1 hingga $2^{36} \times 37 \times p_3$, lalu untuk setiap nilai K kita hanya perlu melakukan binary search untuk mencari p_3 yang memenuhi, dan jika K melewati 2480847655108608, kita hanya perlu menambahkan 2480847655108608 pada

Kompleksitas Waktu : $O(N \log N)$

C. Ciro dan Roti

Tags : *Dynamic Programming, Hashing*

Definisikan $dp[x][y]$ sebagai panjang terkecil dari $S = A_{i_1} + \cdots + A_{i_x}$ dengan y adalah *suffix* dari S. Kita dapat mengecek y menggunakan *Hashing*. Simpan nilai *hash* dari *prefix* dan *suffix* setiap *string*, karena $1 \leq |A_i| \leq 100$, maka banyaknya nilai *hash* berbeda yang tersimpan adalah $200N = 200.000$. Kemudian definisikan dua hal berikut:

- $pref[i][j]$ sebagai nilai *hash* dari *prefix* A_i dengan panjang j
- $suff[i][j]$ sebagai nilai *hash* dari *suffix* A_i dengan panjang j

Base case dari $dp[x][y]$ adalah $dp[1][suff[i][j]] = \min(dp[1][suff[i][j]], |A_i|)$.

Untuk setiap i $(1 \leq i \leq N)$, misalkan $m = |A_i|$. Lakukan hal berikut:

$$x = \min_{1 \leq k \leq m} dp[j - 1][pref[i][k]] + m - k$$

$$dp[j][suff[i][k]] = x, 1 \leq k \leq m$$

Sehingga panjang terkecil dari $A_{i_1} + \cdots + A_{i_j}$ adalah $\min_{y \in hash} dp[j][y]$.

Kompleksitas Waktu : $\mathcal{O}(NK \max(|A_i|)$.

D. Dadu dan Keberuntungan

Tags : *Combinatoric, Math*

Misalkan keadaan ciro berhenti saat banyaknya lemparan dadu adalah ganjil disebut keadaan menang. Dan sebaliknya, jika ciro berhenti saat banyaknya lemparan dadu adalah genap disebut keadaan kalah.

Misalkan $f(x)$ adalah peluang menang jika saat ini ciro sudah mendapat angka 1 sebanyak x kali beruntun dan banyaknya lemparan dadu yang sudah dilakukan ciro adalah genap.

Misalkan $g(x)$ adalah peluang menang jika saat ini ciro sudah mendapat angka 1 sebanyak x kali beruntun dan banyaknya lemparan dadu yang sudah dilakukan ciro adalah ganjil.

Mudah dilihat bahwa $f(N) = 0$ dan $g(N) = 1$.

Jawaban dari soal ini adalah $f(0)$, karena pada keadaan awal, ciro belum melempar apapun, artinya ciro belum mendapat angka 1 $(x = 0)$, dan ciro sudah melempar dadu sebanyak 0 kali (genap).

Mudah dilihat bahwa $f(x) = \frac{1}{6}g(x+1) + \frac{5}{6}g(0)$.

Karena $f(x)$ dan $g(x)$ simetris, maka $f(x) + g(x) = 1$.

Misalkan $a = f(0)$ dan $b = 1 - \frac{5}{6}a$.

$$f(x) = \frac{1}{6}g(x+1) + \frac{5}{6}g(0)$$

$$f(x) = \frac{1}{6}\big(1 - f(x+1)\big) + \frac{5}{6}\big(1 - f(0)\big)$$

$$f(x) = 1 - \frac{1}{6}f(x+1) - \frac{5}{6}f(0)$$

$$f(x) = b - \frac{1}{6}f(x+1)$$

Setelah itu, untuk mencari nilai a dan b,

$$a = f(0)$$

$$a = b - \frac{1}{6}f(1)$$

$$a = b - \frac{1}{6}\left(b - \frac{1}{6}f(2)\right)$$

$$a = b - \frac{1}{6}b + \frac{1}{6^2}f(2)$$

$$a = b - \frac{1}{6}b + 1/6^2(b - \frac{1}{6}f(3))$$

$$a = b - \frac{1}{6}b + \frac{1}{6^2}b - \frac{1}{6^3}f(3)$$

...

$$a = b - \frac{1}{6}b + \frac{1}{6^2}b - \frac{1}{6^3}b + \cdots + \frac{1}{6^{N-1}}b \pm \frac{1}{6^N}f(N)$$

$$a = b(\frac{1}{6} - \frac{1}{6^2} + \frac{1}{6^3} - \frac{1}{6^4} + \cdots + \frac{1}{6^{N-1}})$$

$$a = b(\frac{\left(-\frac{1}{6}\right)^N - 1}{-\frac{7}{6}})$$

$$a = b\frac{1 - \left(-\frac{1}{6}\right)^N}{\frac{7}{6}}$$

$$a = b(\frac{6^N + (-1)^{N+1}}{7 \times 6^{N-1}})$$

Misalkan $k = \left(\frac{6^N + (-1)^{N+1}}{7 \times 6^{N-1}}\right)$ maka,

$$a = (1 - \frac{5}{6}a)k$$

$$a = k - \frac{5}{6}ak$$

$$a\left(1 + \frac{5}{6}k\right) = k$$

$$a\left(\frac{6 + 5k}{6}\right) = k$$

$$a = \frac{6k}{5k + 6}$$

$$a = \frac{6^{N+1} + 6 \times (-1)^{N+1}}{5 \times 6^N + 5 \times (-1)^{N+1} + 7 \times 6^N}$$

$$a = \frac{6^{N+1} + 6 \times (-1)^{N+1}}{2 \times 6^{N+1} + 5 \times (-1)^{N+1}}$$

Jadi jawabannya adalah

$$\frac{6^{N+1} + 6 \times (-1)^{N+1}}{2 \times 6^{N+1} + 5 \times (-1)^{N+1}}$$

Kompleksitas Waktu : $\mathcal{O}(\log N)$

E. Es Pisang Ijo

Tags : *Dynamic Programming, Number Theory*

Misalkan $C(B) = FPB(B_1, B_2, \dots, B_M) \times (B_1 + B_2 + \cdots + B_M) = FPB(B) \times SUM(B)$.

Definisikan $dp[i][g]$ dan $cnt[i][g]$ berturut turut sebagai jumlah $SUM(B)$ dan banyaknya subsequence sehingga $FPB(B) = g$ dari seluruh subsequence hingga indeks ke$-i$. Maka untuk setiap $g \in dp[i-1]$, ketika bertransisi dari $dp[i-1]$ ke $dp[i]$ lakukan:

1. Misalkan $g' = FPB(g, A[i])$
2. Jika kita tidak memakai $A[i]$, maka:
$$dp[i][g'] = dp[i][g'] + dp[i-1][g']$$
$$cnt[i][g'] = cnt[i][g'] + cnt[i-1][g']$$
3. Jika kita memakai $A[i]$, maka:
$$dp[i][g'] = dp[i][g'] + dp[i-1][g] + A[i] \times cnt[i-1][g]$$
$$cnt[i][g'] = cnt[i][g'] + cnt[i-1][g]$$

Diperoleh transisi akhir:

$$dp[i][g'] = dp[i-1][g'] + dp[i-1][g] + A[i] \times cnt[i-1][g]$$

$$cnt[i][g'] = cnt[i-1][g'] + cnt[i-1][g]$$

Karena jumlah maksimum pembagi dari suatu bilangan $X \leq 10^9$ adalah $\mathcal{O}\left(X^{\frac{1}{3}}\right)$ atau sekitar 1344 (https://wwwhomes.uni-bielefeld.de/achim/highly.txt), maka jumlah transisi maksimal dari setiap indeks dp adalah $\mathcal{O}\left(N \max(A_i)^{\frac{1}{3}}\right)$.

Kompleksitas Waktu : $\mathcal{O}\left(N^2 \max(A_i)^{\frac{1}{3}}\right)$.

F. FT Unhas 09

Tags : *Segment Tree, Heavy Light Decomposition*

Mari kita selesaikan jika Ciro diberikan sebuah array daripada sebuah tree. Kita akan menggunakan *Lazy Segment Tree* untuk menyelesaikan soal berikut.

Untuk suatu rentang $[l, r]$ dalam array, misalkan $S(a, d)$ menandakan bahwa rentang $[l, r]$ akan ditambahkan $a, a + d, \dots, a + (r - l)d$. Apa hasil dari $S(a_1, d_1)$ kemudian $S(a_2, d_2)$?

$$a_1, a_1 + d_1, \dots, a_1 + (r - l)d_1$$

$$a_2, a_2 + d_2, \dots, a_2 + (r - l)d_2$$

$$\downarrow$$

$$(a_1 + a_2), (a_1 + a_2) + (d_1 + d_2), \dots, (a_1 + a_2) + (r - l)(d_1 + d_2)$$

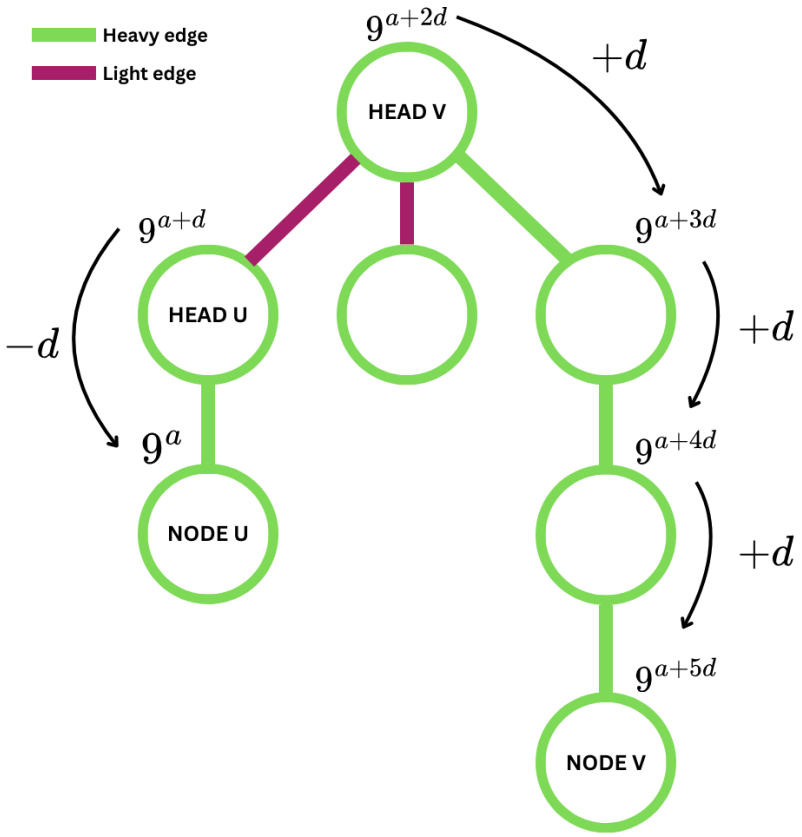
Diperoleh hasilnya sama dengan $S(a_1 + a_2, d_1 + d_2)$.

Kita dapat memakai Data Struktur Heavy Light Decomposition untuk mengaplikasikan *Segment Tree* pada sebuah *path* dari *tree.* Hal yang menjadi tantangan adalah update yang bersifat non komutatif, dimana update dari node $u \to v$ tidak sama dengan $v \to u$. Definisikan

- pos_u – posisi dari node u pada segment tree
- $head_u$ – index dari node pertama pada heavy path yang memiliki node u
- $depth_u$ – kedalaman node u dalam tree
- $parent_u$ – parent dari node u

Definisikan $update(u, v, a, d)$ menandakan sebuah update dari node $u \to v$ sehingga $A_u = A_u \times 9^a, \dots, A_v = A_v \times 9^{a+(k-1)d}$ dengan k adalah jarak dari node u ke node v, k dapat dicari menggunakan *HLD*. Awalnya $au = a$ dan $av = a + (k - 1)d$.

- Jika $depth_{head_u} \geq depth_{head_v}$, maka kita akan melakukan update dari node $head_u \to u$. Ini berarti $A_u = A_u \times 9^{au}, \dots, A_{head_u} = A_{head_u} \times 9^{au+(pos_u - pos_{head_u})d}$, sehingga kita harus melakukan:
  1. $update(head_u, u, au + (pos_u - pos_{head_u})d, -d)$
  2. $au = au + (pos_u - pos_{head_u} + 1)d$
  3. $u = parent_{head_u}$
- Jika $depth_{head_u} < depth_{head_v}$, maka kita akan melakukan update dari node $head_v \to v$. Ini berarti $A_v = A_v \times 9^{av}, \dots, A_{head_v} = A_{head_v} \times 9^{av-(pos_v - pos_{head_v})d}$, sehingga kita harus melakukan:
  1. $update(head_v, v, av - (pos_v - pos_{head_v})d, d)$
  2. $av = av - (pos_v - pos_{head_v} + 1)d$
  3. $v = parent_{head_v}$



Kompleksitas Waktu : $\mathcal{O}\left((N+Q)\log^2 N\right)$

G. Gado Gado

Tags : *Dynamic Programming, Matrix Exponentiation*

Dengan satu operasi, Ciro harus melakukan dua hal berikut:

1. Mengubah $[A_1, A_2, \dots, A_N] \rightarrow [A_1, A_1 + A_2, \dots, A_1 + A_2 + \cdots + A_N] = [B_1, B_2, \dots, B_N]$
2. Mengubah $[B_1, B_2, \dots, B_N] \rightarrow [B_1 + B_2 + \cdots + B_N, \dots, B_{N-1} + B_N, B_N] = [C_1, C_2, \dots, C_N]$

Untuk setiap elemen pada array B, berlaku

$$B_i = \sum_{j=1}^{i} A_j$$

Array B dapat diperoleh menggunakan matriks, yaitu sebagai berikut:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & & 1 & 1 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{N-1} \\ A_N \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_{N-1} \\ B_N \end{bmatrix}$$

Untuk setiap elemen pada array C, juga berlaku

$$C_i = \sum_{j=i}^{N} A_j$$

Array C dapat diperoleh dengan cara yang sama, yaitu sebagai berikut:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & & 1 & 1 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & & 0 & 1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_{N-1} \\ B_N \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_{N-1} \\ C_N \end{bmatrix}$$

Ketika digabungkan, diperoleh

$$\begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & & 1 & 1 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & & 0 & 1 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_{N-1} \\ B_N \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_{N-1} \\ C_N \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & & 1 & 1 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & & 1 & 1 \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{N-1} \\ A_N \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_{N-1} \\ C_N \end{bmatrix}$$

Hasil dari array A dapat diperoleh menggunakan *matrix exponentiation* dalam $\mathcal{O}(N^3 \log K)$, sehingga total kompleksitas waktunya adalah $\mathcal{O}(N^3 Q \log K)$, ini terlalu lambat. Misalkan

$$M = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & & 1 & 1 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & & 1 & 1 \end{bmatrix}$$

Cara yang optimal adalah melakukan prekomputasi matriks $M^1, M^2, M^4, M^8, \ldots$ dalam $\mathcal{O}(N^3 \log K)$.

Setelah prekomputasi, untuk menghitung hasil dari array A, alih alih kita mengalikan dua buah matriks, kita dapat mengalikan array A saat ini dengan setiap matriks M^k sehingga bit ke$-k$ dari K sama dengan 1 yang diprekomputasi sebelumnya. Karena A berukuran $1 \times N$ dan M^k berukuran $N \times N$, maka perkalian A dan M^k dapat dilakukan dalam $\mathcal{O}(N^2)$. Diperoleh total kompleksitas waktu per query adalah $\mathcal{O}(N^2 \log K)$.

Kompleksitas Waktu : $\mathcal{O}(N^2 Q \log K)$

H. Himmel Pasti Akan Melakukan Hal Yang Sama

Tags : *Brute Force, DFS and Similar, Number Theory*

Urutan dari pemilihan bilangan tidak berpengaruh, sehingga kita dapat mengurutkan A terlebih dahulu. Untuk sebuah nilai K, kita akan mencari semua solusi $(B_1, B_2, \dots, B_K)$ sehingga:

1. $B_1 + \cdots + B_K = KPK(B_1, \dots, B_K)$
2. $B_1 \leq \cdots \leq B_K$
3. $FPB(B_1, \dots, B_K) = 1$

Alasan kita hanya mencari solusi sehingga $FPB(B_1, \dots, B_K) = 1$, karena untuk setiap konstan C, berlaku $CB_1 + \cdots + CB_K = KPK(CB_1, \dots, CB_K) \rightarrow C \times (B_1 + \cdots + B_K) = C \times KPK(B_1, \dots, B_K)$.

Misalkan $K = 4$ dan $M = KPK(B_1, \dots, B_K)$. Perhatikan simulasi penyelesaian berikut,

$$B_4 < B_1 + \cdots + B_4 \leq 4B_4$$

$$B_4 < M \leq 4B_4$$

Misalkan kita ambil $M = 3B_4$,

$$B_1 + \cdots + B_4 = 3B_4 \rightarrow B_1 + \cdots + B_3 = 2B_4$$

$$B_1 + \cdots + B_4 = M \rightarrow 2B_1 + \cdots + 2B_4 = 2M \rightarrow 3B_1 + \cdots + 3B_3 = 2M$$

Dengan cara yang sama, diperoleh

$$3B_3 < 2M \leq 9B_3$$

Karena $B_1, \dots, B_K \mid M$, maka $2M = 4B_3, 6B_3, 8B_3$, misalkan kita ambil $2M = 8B_3$,

$$3B_1 + \cdots + 3B_3 = 8B_3 \rightarrow 3B_1 + 3B_2 = 5B_3 \rightarrow 9B_1 + 9B_2 = 15B_3$$

$$3B_1 + \cdots + 3B_3 = 2M \rightarrow 15B_1 + \cdots + 15B_3 = 10M \rightarrow 24B_1 + 24B_2 = 10M$$

Dengan cara yang sama, diperoleh

$$24B_2 < 10M \leq 48B_2$$

Karena $B_1, \dots, B_K \mid M$, maka $10M = 30B_2, 40B_2$, misalkan kita ambil $10M = 40B_2$,

$$24B_1 + 24B_2 = 40B_2 \rightarrow 24B_1 = 16B_2 \rightarrow 3B_1 = 2B_2 \rightarrow 60B_1 = 40B_2 \rightarrow 60B_1 = 10M \rightarrow 6B_1 = M$$

Didapatkan $M = 6B_1 = 4B_2 = 4B_3 = 3B_4 \rightarrow M = KPK(6, 4, 4, 3) = 12$

Diperoleh $(B_1, B_2, B_3, B_4) = (2, 3, 3, 4)$. Ketika diuji, $(2 + 3 + 3 + 4) = KPK(2, 3, 3, 4) \rightarrow 12 = 12$.

Misalkan $S_K = B_1 + \cdots + B_K$. Dari simulasi penyelesaian tersebut, definisikan $F(k, x, y, z)$ sebagai solusi dari $(B_1, \dots, B_k)$ sehingga $M = zB_k$ dan $xS_K = yM$. Untuk mencari $F(k-1, x, y, z)$ dapat dilakukan dengan cara berikut:

$$xS_k = yM$$

$$xS_k = yzB_k$$

$$xS_{k-1} = (yz - x)B_k$$

$$B_k = \frac{xS_{k-1}}{yz - x}$$

$$xB_k = \frac{x^2 S_{k-1}}{yz - x}$$

Substitusi xB_k pada xS_k, diperoleh:

$$xS_k = yM$$

$$xS_{k-1} + \frac{x^2 S_{k-1}}{yz - x} = yM$$

$$xS_{k-1}\left(1 + \frac{x}{yz - x}\right) = yM$$

$$xS_{k-1}\left(\frac{yz}{yz - x}\right) = yM$$

$$xyzS_{k-1} = (yz - x)yM$$

$$xzS_{k-1} = (yz - x)M$$

Untuk mencari z, dapat dicari dalam batasan

$$xzB_{k-1} < (yz - x)M \leq kxzB_{k-1}$$

$$\frac{xzB_{k-1}}{yz - x} < M \leq \frac{kxzB_{k-1}}{yz - x}$$

Diperoleh $F(k, x, y, z) \to F\left(k - 1, xz, yz - x, \frac{jxz}{yz-x}\right)$ untuk setiap $1 < j \leq k$ dengan $(yz - x) \mid jxz$.

Untuk mencari semua solusi dari $(B_1, \ldots, B_K)$ membutuhkan waktu sekitar $\mathcal{O}(K^{K+1})$ dengan jumlah solusi valid $\mathcal{O}(K^K)$. Terakhir kita hanya perlu mencari semua solusi menggunakan map.

Kompleksitas Waktu: $\mathcal{O}(NK^K \log N)$

Competitive Programming – Editorial

I. Indomie Soto

Tags : *Treap*

Mari kita buat segment tree untuk mengatasi permintaan tipe kedua dan ketiga. Setiap node akan menyimpan:

1. $first$ – Elemen pertama pada suatu rentang
2. $last$ – Elemen pertama pada suatu rentang
3. inc – Apakah suatu rentang terurut secara tidak menurun?
4. dec – Apakah suatu rentang terurut secara tidak menaik?

Transisi dari $pull\ node$ dari $left$ dan $right$ ialah

1. $node.first = left.first$
2. $node.last = right.last$
3. $node.inc = (left.inc)\ \&\ (right.inc)\ \&\ (left.last \leq right.first)$
4. $node.dec = (left.dec)\ \&\ (right.dec)\ \&\ (left.last \geq right.first)$

Hati – hati ketika melakukan transisi $pull\ node$ ketika $left = NULL$ atau $right = NULL$.

Untuk mengatasi permintaan tipe pertama, gunakan data struktur Implicit Treap yang dapat dilakukan dalam $\mathcal{O}(\log N)$. Dengan metode *Split* dan *Merge*, permintaan ini sangat mudah diatasi. Simpan variabel rev pada setiap node yang menandakan apakah suatu rentang akan dibalik atau tidak, mirip dengan metode lazy propagation.

Kompleksitas Waktu: $\mathcal{O}\big((N+Q)\log N\big)$

J. Jalangkote

Tags : *Math*

Cara terbaik untuk Alvin memenangkan permainan adalah dengan membagi isian jalangkote serata mungkin untuk setiap jenis isian, sehingga satu jenis isian jalangkote dengan jumlah terbanyak akan menjadi:

$$\left\lceil \frac{N}{X} \right\rceil$$

Sehingga, King harus mengubah semua jalangkote selain satu jenis dengan jumlah yang paling besar agar dapat memenangkan pertandingan, jadi jika:

$$N - \left\lceil \frac{N}{X} \right\rceil > Y$$

Maka Alvin akan MENANG, jika kondisi tidak memenuhi, Alvin akan KALAH.

Kompleksitas Waktu : $\mathcal{O}(1)$

K. Krisis Air

Tags : *Greedy, Number Theory*

Misalkan $B = [B_1, B_2, B_3, \ldots, B_N]$ dimana $B_i = A_i \ (mod\ N)$.

Jika diperhatikan setiap kertas Ajaib yang dibakar oleh Ciro, hanya ada dua kemungkinan perubahan untuk A_i yakni, $A_i \rightarrow A_i + 1$ atau $A_i \rightarrow A_i - (N-1)$.

Karena array B merupakan hasil modulo N dari array A dan $(N-1)\ (mod\ N) \equiv 1\ (mod\ N)$, maka, untuk setiap gerakan, seluruh elemen dari array B akan bertambah 1.

Jika kita ingat lagi, Ciro dapat menyelamatkan desanya hanya apabila setidaknya $N-1$ menara memiliki puncak sejajar dengan tanah, dengan kata lain ketinggiannya adalah 0 meter, ini artinya setidaknya $N-1$ anggota array B bernilai 0. Karena sifat yang ada pada teks yang di beri warna merah, artinya sedari awal array B harus memiliki setidaknya $N-1$ anggota yang bernilai sama. Dan jika syarat ini tidak terpenuhi, Ciro tidak akan pernah dapat menyelamatkan desanya.

Jika terdapat tepat $N-1$ anggota array B yang bernilai sama, maka anggota yang berbeda sendirian akan menjadi satu-satunya menara yang memiliki ketinggian di atas tanah. (Kasus 1)

Jika seluruh anggota array B sama, mudah dilihat bahwa memilih menara tertinggi untuk menjadi satu-satunya menara yang memiliki ketinggian di atas tanah adalah strategi terbaik Ciro. (Kasus 2)

Jumlah kertas Ajaib yang di butuhkan Ciro untuk menyelamatkan desanya adalah:

$$\left(\sum_{i=1}^{N} A_i\right) - A_j$$

Untuk kasus 1, B_j adalah anggota array B yang berbeda sendirian.

Untuk kasus 2, menara ke-j adalah menara tertinggi,

Dengan menghitung jumlah kertas Ajaib yang dibutuhkan Ciro, kita dapat memperkirakan apakah kertas Ciro cukup untuk menyelamatkan desanya.

Kompleksitas Waktu : $\mathcal{O}(N)$

L. Lembaga Desa Kucing

Tags : *Segment Tree*

Banyaknya toleransi berbeda untuk setiap desa kucing adalah 26. Oleh karena itu, kita dapat menggunakan 26 *Lazy Segment Tree*. Misalkan $seg[i]$ adalah *Segment Tree* ke−i, digunakan untuk menyimpan banyaknya desa kucing yang memiliki toleransi i dalam suatu rentang $[l, r]$. Setiap node akan menyimpan:

1. $pref$ → Panjang maksimum prefix pada rentang $[l, r]$ yang seluruh nilainya bernilai 1.
2. $suff$ → Panjang maksimum suffix pada rentang $[l, r]$ yang seluruh nilainya bernilai 1.
3. $whole$ → Apakah seluruh nilai pada rentang $[l, r]$ bernilai 1?
4. sum → Subarray terpanjang pada rentang $[l, r]$ yang seluruh nilainya bernilai 1.
5. $value$ → Total kekuatan pada rentang $[l, r]$.
6. $lazy$ → Lazy propagation.

Transisi dari $pull\ node$ dari $left$ dan $right$ ialah:

1. $node.pref = left.pref + (left.whole\ ?\ right.pref\ :\ 0)$
2. $node.suff = right.suff + (right.whole\ ?\ left.suff\ :\ 0)$
3. $node.whole = left.whole\ \&\ right.whole$
4. $node.sum = \max(left.sum, right.sum, left.suff + right.pref)$
5. $node.value = left.value + right.value + 2 \times left.suff \times right.pref$
6. $node.lazy = -1$

Untuk operasi 5, total kekuatan desa kucing adalah total kuadrat dari setiap aliansi, sebagai contoh untuk ABBCCC total kekuatannya adalah $1^2 + 2^2 + 3^2$. Pembuktian operasi 5, $left.suff$ desa akan beraliansi dengan $right.pref$ desa, sehingga total kekuatan yang awalnya $(left.suff)^2 + (right.pref)^2$ akan menjadi $(left.suff + right.pref)^2$, didapatkan total transisinya adalah $(left.suff + right.pref)^2 - (left.suff)^2 - (right.pref)^2 = 2 \times left.suff \times right.pref$.

1. Query tipe 1 $(1\ l\ r\ x)$ – Perubahan Nilai Toleransi

Lakukan update pada $seg[j]$ untuk setiap 'A' $\leq j \leq$ 'Z', $j \neq c$ menjadi 0 pada rentang $[l, r]$ dan $seg[c]$ menjadi 1 pada rentang $[l, r]$.

Query tipe 1 dapat dilakukan dalam $\mathcal{O}(\log^2 N)$.

2. Query tipe 2 $(2\ l\ r)$ – Total Kekuatan Desa Kucing

Total dari $D_l + \cdots + D_r$ adalah total $value$ pada $seg[j]$ untuk setiap 'A' $\leq j \leq$ 'Z' dalam rentang $[l, r]$, kita misalkan sebagai Y. Namun ini masih belum cukup untuk menghitung total kekuatan desa kucing. Sebagai contoh untuk kasus ABBCCC dan kita ingin mengetahui kekuatan $D_3 + D_4$, maka $Y = 1^2 + 1^2 = 2$, ini karena segment tree hanya menghitung total kekuatan dari BC, sehingga lakukan hal berikut:

1. Jika $node.pref > 0$ pada rentang $[l, r]$ dalam $seg[j]$, maka terdapat desa lain yang beraliansi dengan desa l yang memiliki toleransi C_j, total desanya adalah $left.suff$ pada rentang $[1, l-1]$. Dalam $seg[j]$, total kekuatannya akan bertambah sebanyak $node.pref \times left.suff$.
2. Jika $node.suff > 0$ pada rentang $[l, r]$ dalam $seg[j]$, maka terdapat desa lain yang beraliansi dengan desa r yang memiliki toleransi C_j, total desanya adalah $right.pref$ pada rentang $[r+1, N]$, total kekuatannya akan bertambah sebanyak $node.suff \times right.pref$.

Query tipe 2 dapat dilakukan dalam $\mathcal{O}(\log^2 N)$

3. Query tipe 3 (3) – Kekuatan desa kucing dengan Kekuatan Terbesar

Kekuatan desa kucing dengan kekuatan terbesar yaitu sebuah subarray yang bernilai 1 pada rentang $[l, r]$ dalam sebuah segment tree, itu adalah nilai maksimal dari $node.sum$ pada rentang $[1, N]$ untuk setiap $seg[j]$ dengan 'A' $\leq j \leq$ 'Z'.

Query tipe 3 dapat dilakukan dalam $\mathcal{O}(\log^2 N)$

Kompleksitas Waktu : $\mathcal{O}(N \log N + Q \log^2 N)$