Editorial Babak Penyisihan 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	Angsa Ganas	Zulfahmi
B	Bus Trans Mamminasata	Zulfahmi
C	Counter Rusak	Benediktus Gianto Jarod
D	Dua Empa	Zulfahmi
E	Empang Ternak Lele	Zulfahmi
F	Federasi Desa Kucing	Zulfahmi
G	Game Squirtle	Zulfahmi
H	Hari Ulang Tahun	Benediktus Gianto Jarod
I	Ilmu Kali Kalian	King Ahmad Ilyas
J	Jualan Takjil	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 Penyisihan 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 panitia 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!!!

RECURSION 1.0

Competitive Programming – Editorial

A. Angsa Ganas

Tags : *Data Structure, String*

Dua string disebut anagram jika dan hanya jika jumlah kemunculan setiap huruf pada kedua string tersebut sama. Sebagai contoh, "angsa" dan "ganas" merupakan anagram karena masing-masing huruf muncul dengan frekuensi yang sama, sedangkan "angsa" dan "bebek" bukan.

Misalkan untuk sebuah string S_i, $H(S_i)$ adalah array ukuran 26 yang menyimpan frekuensi dari setiap huruf. Sehingga $H(S_i + S_j)$ sama dengan menambahkan jumlah huruf dari 'a' sampai 'z' pada dua string S_i dan S_j.

Buatlah sebuah map yang berisi array dengan ukuran 26. Kemudian untuk setiap (j, k) dengan $j < k$ carilah S_i dengan $i < j$ sehingga $H(S_i) = H(S_j + S_k)$, simpan kemunculan $H(S_i)$ dalam map.

Solusi Juri : Angsa_Ganas.cpp

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

B. Bus Trans Mamminasata

Tags : *Dynamic Programming, Segment Tree*

Ini adalah soal klasik yaitu Longest Increasing Subsequence.

Definisikan $dp[i]$ sebagai banyaknya transit maksimal yang dapat dilakukan pada jika bus memulai perjalanan dari pemberhentian ke$-i$. Misalkan $bad[i]$ adalah daftar indeks j sehingga (i, j) terdapat pada pengecualian transit langsung. Awalnya $dp[i] = 1$. Kemudian, $dp[i]$ dapat dikalkulasikan sebagai:

$$dp[i] = \left(\max_{j>i, A_j > A_i, j \notin bad[i]} dp[j] \right) + 1$$

Cara ini dapat dilakukan dalam $O(N^2 \log N)$, ini terlalu lambat untuk $N \leq 10^5$.

Untuk mempercepat algoritmanya, gunakan Maximum Segment Tree untuk menghitung nilai maksimum pada suatu rentang dalam $O(\log N)$. Hitung $dp[i]$ berdasarkan A_i terbesar hingga terkecil. Oleh karena itu, urutkan pemberhentian berdasarkan prioritas transit. Simpan urutan aslinya agar bisa mengembalikan hasil ke indeks awal. Untuk mengatasi setiap indeks j pada $bad[i]$, ubahlah $dp[j]$ menjadi 0 dalam $O(\log N)$, kemudian ubahlah ke nilai aslinya setelah menghitung $dp[i]$.

Solusi Juri : Bus_Trans_Mamminasata.cpp

Kompleksitas Waktu: $O\big((N + K) \log N\big)$

C. Counter Rusak

Tags : *Greedy, Math*

Karena *counter* tersebut hanya dapat menampilkan 8 digit, maka *counter* tersebut berperilaku layaknya bilangan basis-8. Ini artinya kita bisa mengkonversi bilangan yang ditampilkan *counter* menjadi bilangan basis-8.

Modifikasi nya dapat dilihat dengan tabel dibawah:

Digit Counter	0	1	2	4	5	6	8	9
Digit Basis-8	0	1	2	3	4	5	6	7

Kemudian bilangan basis-8 yang didapatkan dapat dikonversi menjadi desimal untuk mendapat hitungan sebenarnya.

Sebagai contoh, misalkan *counter* menampilkan 245, maka dengan melihat tabel diatas, bilangan tersebut setara dengan 234_8. Jika dikonversi menjadi desimal:

$234_8 = 2 \times 8^2 + 3 \times 8^1 + 4 \times 8^0$

$234_8 128 + 24 + 4$

$234_8 = 156$

Jadi jika counter menampilkan 245, maka hitungan sebenarnya adalah 156.

Solusi Juri : Counter_Rusak.cpp

Kompleksitas Waktu: $O(\log N)$

D. Dua Empa

Tags : *Brute Force, DFS and Similar*

Lakukan backtracking untuk setiap kemungkinan operasi. Misalkan kartu-kartunya adalah $A_1, A_2, \dots, A_K$, maka kita harus mengecek kemungkinan operasi aritmatika pada $(A_1, A_2), (A_1, A_3), \dots, (A_{K-1}, A_K)$.

Perhatikan bahwa untuk operasi $k = 2, A_i > A_j$ dan untuk operasi $k = 4, A_j \neq 0$ dan $A_i \ (mod\ A_j) = 0$.

Misalkan $N = 4$ adalah jumlah kartu.

Solusi Juri : Dua_Empa.cpp

Kompleksitas Waktu: $\mathcal{O}(N! 4^N)$

E. Empang Ternak Lele

Tags : *Bitmask, Dynamic Programming*

Gunakan *Dynamic Programming With Bitmask.*

Misalkan untuk sebuah empang A, V_A adalah keuntungan yang dihasilkan. Keuntungan yang dihasilkan dari penggabungan empang lele A dan B dapat dihitung sebagai:

$$V_A + V_B - A_1A_M + A_MB_1 + A_1B_M$$

$$V_A + V_B + A_1(B_M - A_M) + A_MB_1$$

Perhatikan bahwa nilai dari $V_A + V_B$ tidak berubah, sehingga kita dapat menghitung V_A secara terpisah untuk masing-masing empang.

Untuk setiap empang ke$-i$, misalkan B_i adalah harga lele pertama dan C_i adalah harga lele terakhir.

Definisikan $dp[mask][i]$ sebagai keuntungan maksimal dari *subset* $\{x \subseteq mask\}$ dalam representasi biner dengan empang ke$-i$ adalah empang terakhir.

Perhatikan bahwa, keuntungan dari penggabungan empang $A_1A_2 \dots A_N$ sama ketika penggabungan empang di rotasikan, dengan kata lain keuntungan dari penggabungan empang $V_{A_1A_2\dots A_N} = V_{A_2A_3\dots A_NA_1} = V_{A_3A_4\dots A_NA_1A_2} = \dots = V_{A_NA_1A_2\dots A_{N-1}}$. Sehingga awalan dari penggabungan empang tidak berpengaruh, ambil awalan dari semua penggabungan empang adalah A_1.

Base case

$$dp[2^i \otimes 1][j] = B_1C_j + B_jC_1, (2 \le j \le N)$$

Transition

$$dp[mask][i] = \max_{j \subseteq mask} \left(dp[mask \otimes 2^j][j] + B_1(C_i - C_j) + B_iC_j\right)$$

Jawabannya adalah $\max_{2 \le j \le N} \left(dp\left[2^N - 1\right][j]\right)$.

Perhatikan bahwa, terdapat edge case ketika $N = 1$.

Solusi Juri : Empang_Ternak_Lele.cpp

Kompleksitas Waktu: $O(N^2 2^N)$.

F. Federasi Desa Kucing

Tags : *Binary Search, Data Structure*

Kita akan menggunakan set (misalkan $index$) untuk menyimpan indeks terbesar dari suatu aliansi dan multiset (misalkan $power$) untuk menyimpan kekuatan dari setiap aliansi. Sebagai contoh, untuk $S =$ "ABBCCC", set akan menyimpan $\{1, 3, 6\}$ dan multiset akan menyimpan $\{1, 2, 3\}$.

1. Query tipe 2 $(2\ i)$ – Kekuatan Desa Kucing

   Kekuatan desa i adalah banyaknya desa yang beraliansi dengannya, yaitu panjang maksimum subarray $[L, R]$ dengan $L < i \leq R$ yang mengandung nilai toleransi yang sama dengan C_i.

   Secara matematis, kekuatan desa i adalah panjang maksimum interval $[L, R]$ dimana $1 \leq L < i \leq R \leq N$ sehingga $C_j = C_i$ untuk setiap $L < j \leq R$.

   Untuk mencari L dan R, kita dapat melakukan binary search pada set index. Untuk mencari R, cukup cari indeks terbesar j sehingga $j \geq i$, pada C++ ini adalah $index.lower\_bound(i)$. Setelah itu L adalah indeks sebelum $index.lower\_bound(i)$. Diperoleh kekuatan dari desa i adalah $R - L$.

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

2. Query tipe 3 (3) – Desa kucing dengan Kekuatan Terbesar

   Untuk melacak desa dengan kekuatan terbesar, kita dapat menggunakan multiset power yang menyimpan semua nilai kekuatan desa.

   Query tipe ke 3 dapat dijawab dalam $\mathcal{O}(\log N)$.

3. Query tipe 1 $(1\ i\ x)$ – Perubahan Nilai Toleransi

   Kita akan mendefinisikan dua fungsi utama, yaitu:

   a. $cut(i)$ – desa i akan memutus aliansi dengan desa $i + 1$.
      Dengan cara yang sama pada query tipe 2, terdapat L dan R dengan $L < i \leq R$ sehingga desa $L + 1$ sampai R beraliansi dengan desa i. Maka, aliansi tersebut akan terbagi menjadi dua kubu, yaitu $[L + 1, i]$ dan $[i + 1, R]$. Akibatnya, terdapat dua kekuatan baru yaitu $i - L$ dan $R - i$. Setelah itu, tambahkan indeks terbesar baru pada kubu $[L + 1, i]$ yaitu i.

   b. $join(i)$ – desa i akan beraliansi dengan desa $i + 1$.
      Hapus i dari index karena desa i akan bergabung dengan desa $i + 1$. Setelah itu, dengan cara yang sama pada query tipe 2, terdapat L dan R dengan $L < i \leq R$ sehingga semua desa L sampai R beraliansi dengan desa i. Aliansi yang awalnya terbagi menjadi dua kubu yaitu $[L + 1, i]$ dan $[i + 1, R]$, akan menjadi satu, akibatnya terdapat dua kekuatan yang dihapus yaitu $i - L$ dan $R - i$, setelah itu kekuatan baru aliansi i adalah $R - L$.

   Ketika desa i merubah nilai toleransinya ke c, lakukan langkah berikut:

   a. $cut(i)$, jika $C_i = C_{i+1}$

b. $cut(i-1)$, jika $C_i = C_{i-1}$
c. $C_i = c$
d. $join(i)$, jika $C_i = C_{i+1}$
e. $join(i-1)$, jika $C_i = C_{i-1}$

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

Solusi Juri : Federasi_Desa_Kucing.cpp

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

G. Game Squirtle

Tags : *DSU, Math*

Definisikan $maxpow(x) = a^b$ dengan a terkecil sehingga $x = a^b$. Misalkan $maxpow(X) = a^b$ dan $maxpow(Y) = a^c$. Untuk sebuah petunjuk (X, Y, i, j), $X^{A_i} = Y^{A_j} \rightarrow a^{bA_i} = a^{cA_j} \rightarrow bA_i = cA_j \rightarrow A_i = \frac{c}{b}A_j$. Perhatikan bahwa, jika $X = 1$ maka $Y = 1$, karena informasi yang diberikan selalu konsisten, sehingga kita tidak perlu memperdulikan kasus tersebut.

Kita dapat menggunakan Weighted DSU untuk menyelesaikan soal ini. Definisikan weight pada edge dari node U ke node V sebagai W_U. Weight tersebut menyatakan bahwa $A_U = W_U \times A_V$. Karena petunjuk (X, Y, i, j) mengindikasikan $A_i = \frac{c}{b}A_j$, maka weight pada edge dari node U ke node V adalah $\frac{c}{b}$.

1. $find(U) - A_U = (?) \times A_{P_U}$
   Misalkan P_U adalah parent dari U dan P_{P_U} adalah parent dari P_U. Kita punya
   - $A_U = W_U A_{P_U}$
   - $A_{P_U} = W_{P_U} A\_P_{P_U}$

   Ketika P_{P_U} menjadi parent dari U, maka:
   $$A_U = W_U A_{P_U}$$
   $$A_U = W_U W_{P_U} A_{P_{P_U}}$$
2. $merge(U, V, b, c) - A_U = \frac{c}{b}A_V$
   Misalkan P_U dan P_V berturut turut adalah *parent* dari U dan V. Kita punya
   - $A_U = \frac{c}{b}A_V$
   - $A_U = W_U A_{P_U}$
   - $A_V = W_V A_{P_V}$

   Ketika P_V menjadi parent dari P_U, maka:
   $$A_U = \frac{c}{b}A_V$$
   $$W_U A_{P_U} = \frac{c}{b}W_V A_{P_V}$$
   $$A_{P_U} = \frac{cW_V}{bW_U}A_{P_V}$$
3. $query(U, V) - \frac{A_U}{A_V} = ?$
   Jika $find(U) \neq find(V)$, maka A_U dan A_V belum memiliki keterkaitan, keluarkan -1. Selain itu, misalkan $P = find(U) = find(V)$. Jawabannya adalah $\frac{A_U}{A_V} = \frac{W_U A_P}{W_V A_P} = \frac{W_U}{W_V}$.

Untuk mencari $maxpow(x)$, terdapat sebuah algoritma bernama Pollard Rho untuk mencari faktorisasi prima dalam $O(\sqrt[4]{x})$. Namun, terdapat sebuah cara yang lebih simpel, gunakan fungsi pow pada C++ atau "**" pada python. Untuk C++ gunakan $pow\left(y, \frac{1.0}{a}\right)$ untuk mencari nilai dari $\sqrt[a]{y}$ dalam $\mathcal{O}(\log y)$. Namun nilai ini tidak presisi jika dibulatkan, cara mengatasinya adalah cek apakah $y^e = x$ untuk setiap e dari $pow\left(y, \frac{1.0}{a}\right) - 1 \leq e \leq pow\left(y, \frac{1.0}{a}\right) + 1$.

Solusi Juri : Game_Squirtle.cpp

Kompleksitas Waktu: $\mathcal{O}(N + M \log(\max(A_i)) + Q)$.

H. Hari Ulang Tahun

Tags : *Greedy, Number Theory*

Karena terdapat $N + 2$ minuman, maka total terdapat $N + 2$ kemungkinan letak racun.

Asumsikan kita menggunakan K detektor.

Karena hasil dari detektor hanya ada dua yakni, beracun dan tidak beracun. Maka, total hanya ada 2^K kemungkinan untuk hasil dari semua detektor.

Mudah dilihat bahwa $N + 2$ harus tidak lebih besar dari 2^K sebab jika tidak, maka ada dua kemungkinan letak racun yang menghasilkan hasil pengujian yang sama untuk K detektor.

$$(N + 2) \leq 2^K \rightarrow K \geq \log_2(N + 2)$$

Jadi, Ciro perlu setidaknya $\lceil \log_2(N + 2) \rceil$ detektor untuk dapat mendeteksi racun dengan tepat.

Strategi Ciro:

Ciro hanya perlu menandai setiap gelas dengan nomor dari 0 hingga $N + 1$. Kemudian mengkonversi nomor-nomor tersebut menjadi basis-2 dengan $\lceil \log_2(N + 2) \rceil$ digit. Kemudian mengambil $\lceil \log_2(N + 2) \rceil$ detektor, yang masing-masing mewakili 1 digit bilangan biner. Detektor tersebut juga di nomor dari 1 hingga $\lceil \log_2(N + 2) \rceil$.

Kemudian, untuk setiap detektor, misalkan detektor tersebut bernomor i, maka detektor tersebut ditetesi setiap minuman yang digit ke-i nya bernilai 1.

3 Jam kemudian, Ciro perlu menyiapkan kertas untuk menuliskan hasil setiap detektor. Kemudian Ciro mengecek detektor satu per satu berurutan dari 1 hingga $\lceil \log_2(N + 2) \rceil$. Jika detektor yang di cek mengeluarkan hasil beracun, Ciro menuliskan angka 1, sebaliknya Ciro menuliskan angka 0.

Setelah semua sudah di cek, Ciro kemudian mendapatkan bilangan biner $\lceil \log_2(N + 2) \rceil$ digit. Bilangan inilah nomor dari gelas beracun.

Solusi Juri : Hari_Ulang_Tahun.cpp

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

RECURSION 1.0

Competitive Programming – Editorial

I. Ilmu Kali Kalian

Tags : *Binary Search, Greedy, Heap*

Langkah yang paling optimal adalah selalu tambahkan 2 pada angka terkecil. Bukti:

$$A_1A_2 \dots A_N \rightarrow (A_1 + 1)A_2 \dots A_N \rightarrow (A_1A_2 \dots A_N) + A_2 \dots A_N$$

Agar $A_2 \dots A_N$ maksimum, maka A_1 haruslah angka terkecil.

Hasil perkalian menghasilkan angka genap jika setidaknya terdapat satu angka genap, jika tidak ada, tambahkan angka terkecil dengan 1.

Selanjutnya lakukan *binary search* untuk mencari x terbesar sehingga banyaknya operasi yang dibutuhkan agar bilangan terkecil pada barisan **setidaknya** x tidak lebih dari K.

Misalkan $good(x)$ menyatakan banyaknya operasi minimum sehingga nilai terkecil dari barisan A **setidaknya** x. Maka untuk setiap $A_i < x$, tambahkan 2 sebanyak $\left\lceil \frac{x-A_i}{2} \right\rceil$. Didapatkan,

$$good(x) = \left(\sum_{i=1}^{N} \max\left(0, \left\lceil \frac{x - A_i}{2} \right\rceil\right) \right) \leq K$$

Kemudian, untuk setiap $A_i < x$, ubah $A_i = x + (x - A_i)\ (mod\ 2)$. Sebagai contoh jika $A = [1, 2]$ dan $k = 2$, didapatkan $x = 3$, namun hasil optimal adalah $A = [3, 4]$ bukan $[3, 3]$.

Selanjutnya, jika K masih tersisa, tambahkan nilai terkecil dari baris angka dengan 2 hingga semua K operasi selesai dilakukan. Ini dapat dilakukan dengan heap dalam $\mathcal{O}(log\ N)$.

Solusi Juri : Ilmu_Kali_Kalian.cpp

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

J. Jualan Takjil

Tags : *Binary Search, Segment Tree*

Misalkan A_i adalah banyaknya penjual yang menjual makanan jenis ke$-i$.

Ketika penjual mulai berjualan di kota, tambahkan 1 pada interval $[L_i, R_i]$.

Ketika penjual pulang kampung, kurangi 1 pada interval $[L_i, R_i]$.

Menghitung banyaknya jenis makanan berbeda yang tersedia di kota Ciro dapat dikalkulasikan dengan

$$\sum_{i=1}^{N} [A_i > 0] = N - \sum_{i=1}^{N} [A_i = 0]$$

Karena A_i pasti selalu positif, kita dapat menggunakan teknik Counting Minimum with Segment Tree. Buatlah segment tree di mana setiap node menyimpan informasi berikut:

- min: nilai minimum pada $[l, r]$, nilai default 0.
- $count$: banyaknya elemen yang sama dengan min dalam rentang $[l, r]$, nilai default $r - l + 1$.
- $lazy$: nilai lazy propagation.

Banyaknya elemen bernilai 0 pada rentang $[1, N]$ adalah $tree[0].count$ jika $tree[0].min = 0$.

Namun, karena N sangat besar $(N \leq 10^9)$, kita tidak dapat menerapkan segment tree secara langsung. Saat menerapkan update pada sebuah segment tree, banyaknya node yang terpengaruh hanyalah $\mathcal{O}(\log N)$. Gunakan sebuah data struktur bernama Dynamic Segment Tree, dimana data struktur ini bekerja seperti Segment Tree pada umunya, namun kita hanya membuat node yang terpengaruh pada sebuah update.

Solusi Juri : Jualan_Takjil.cpp

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