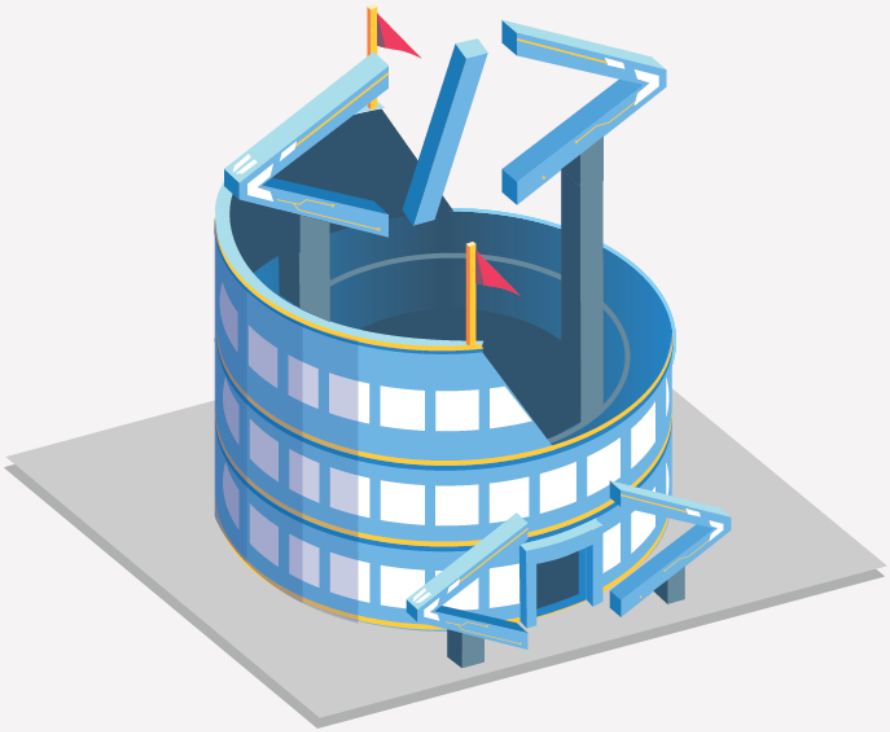
COMPFEST 8

COMPETITION

COMPETITIVE PROGRAMMING CONTEST



COMPFEST 8

Pembahasan Coderclass

SCPC – Minggu 1

Daftar Soal

A. Saluran Televisi
B. Panggilan Akrab
C. Lauk, Lauk Everywhere
D. Toni Vs Tere
E. Euis
F. CompFestSeven
G. Lorong Batu
H. Random Generator

A. Saluran Televisi

Tag

ad hoc

Pembahasan

Kita dapat menggunakan operator modulo (mod pada Pascal, dan % pada C, C++, dan Java) untuk menyelesaikan soal ini. Lebih tepatnya, untuk masukan *next*, maka jawabannya $(X - 1)\ mod\ 100$, dan untuk masukan *prev*, jawabannya $(X + 1)\ mod\ 100$.

Hanya saja, karena implementasi operator modulo, bilangan negatif tidak diubah ke positif. Hal ini menjadi masalah ketika terjadi perhitungan $(0 - 1)\ mod\ 100$. Salah satu cara mengatasinya adalah dengan menambahkan nilai modulonya sebelum dimodulo. Sehingga, pada masukan *next* yang kita hitung menjadi $(X - 1 + 100)\ mod\ 100 = (X + 99)\ mod\ 100$.

Kompleksitas: $O(1)$

B. Panggilan Akrab

Tag

complete search

Pembahasan

Periksa untuk setiap kemungkinan pasangan teman, apakah mereka memiliki panggilan akrab yang sama atau tidak. Ini bisa dilakukan dengan mencoba semua kemungkinan panggilan akrab di salah satu orang, lalu mencarinya di orang yang lain. Jika ada, maka tambahkan jawaban dengan 1.

Berikut *pseudocode* untuk menjalankan algoritma di atas:

```
answer = 0
for i <- 1 .. N-1:
     for j <- i+1 .. N:
          if check(i,j):
               answer = answer + 1

check(i,j):
     for each b nickname of i-th person:
          if b substring of j-th person:
               return true
     return false
```

Kompleksitas: $O(N^2|S|^2)$

C. Lauk, Lauk Everywhere

Tag

parsing, searching

Pembahasan

Observasi yang penting adalah, pemilihan hidangan independen antar tiap kelompok hidangan. Oleh karena itu, kita dapat menggunakan aturan perkalian untuk mencari banyaknya konfigurasi untuk setiap pertanyaan. Misal pertanyaannya berisi Kkelompok hidangan yaitu $G_1, G_2, \ldots, G_K$. Maka, jawabannya adalah $|G_1| \cdot |G_2| \cdot \ldots \cdot |G_K|$.

Salah satu tantangan di soal ini adalah melakukan *parsing* pada masukan soal. Selain itu, kelompok hidangan menggunakan string, yang sedikit mempersulit penggunaan array untuk menghitung. *Time Limit* dibuat besar agar pencarian kelompok hidangan dapat dilakukan secara linier.

D. Toni Vs Tere

Tag

ad hoc

Pembahasan

Jika selisihnya kurang dari 2, maka terjadi voting ulang. Kenapa begitu? Asumsikan $Y < X$, maka $Y = X - d$. Sehingga:

$$X \geq \frac{(X+Y)}{2} + 1$$
$$X \geq \frac{2X - d}{2} + 1$$
$$X \geq X - \frac{d}{2} + 1$$
$$\frac{d}{2} \geq 1$$
$$d \geq 2$$

Oleh karena itu, hanya ada 3 kasus yang bisa terjadi, yaitu:

1. $Y - X \geq 2$, pemenangnya Toni
2. $X - Y \geq 2$, pemenangnya Tere
3. Jika bukan dua kasus di atas, voting ulang

Namun, angka yang diberikan pada masukan sangat besar. Maka, jangan lupa untuk menggunakan **64-bit *unsigned integer***. Selain itu, $X - Y$ atau $Y - X$ memiliki potensi menyebabkan *underflow*, sehingga harus dilakukan pengecekan lagi. Berikut *pseudocode* untuk menyelesaikan soal ini:

```
if(X < Y):
    if(Y - X > 1):
        print "Toni"
    else:
        print "Voting Ulang"
else:
```

```
if(X - Y > 1):
     print “Tere”
else:
     print “Voting Ulang”
```

Kompleksitas: $O(1)$.

E. Euis

Tag

ad hoc

Pembahasan

Perhatikan bahwa cara baca tiap *eu* maupun *e* independen satu sama lain. Oleh karena itu, kita dapat menghitung banyak cara membaca menggunakan aturan perkalian. Lakukan iterasi untuk menghitung kemunculan *eu* dan *e*. Misal kemunculan *eu* adalah a, dan *e* adalah b. Maka, hasil akhirnya adalah $3^a \cdot 2^b$.

Perhatikan bahwa *e* yang merupakan bagian dari *eu* tidak dimasukkan dalam penghitungan a. Selain itu, kesalahan yang sering terjadi adalah menggunakan tipe data 32 bit bertanda dalam menampung jawaban, karena dapat terjadi *overflow* ketika perkalian dengan 3.

Kompleksitas: $O(|S|)$.

F. CompFestSeven

Tag

ad hoc

Pembahasan

Kita cukup melakukan *looping*, dengan pengecekan kondisi di dalamnya. Perhatikan bahwa kita tidak perlu membagi semua kasus secara *hardcode* untuk menuliskan kondisi "CompFestSeven", "CompSeven", dan lainnya. Berikut *pseudocode* untuk solusi tersebut:

```
for i <- 1 .. N:
     if i is divisible by 3:
          print "Comp"
     if i is divisible by 5:
          print "Fest"
     if i is divisble by 7:
          print "Seven"
     if i is not divisible by 3, 5, or 7:
          print i
     if i < N:
          print " "
     else:
          print newline
```

Beberapa kesalahan yang masih sering terjadi adalah kelebihan spasi di belakang, atau lupa memberikan *newline*.

Kompleksitas: $O(N)$.

G. Lorong Batu

Tag

greedy

Pembahasan

Observasi yang penting adalah, tiap batu yang berada di kolom yang sama, atau bersebelahan adalah bertetangga.

Oleh karena itu, kita dapat mengambil batu secara berurutan dari kiri ke kanan. Pada saat mengambil, lakukan secara *greedy*, dengan selalu mengambil 2 batu apabila bisa. Jika terdapat beberapa kemungkinan pengambilan batu, prioritaskan mengambil batu di kolom yang sama. Hal tersebut karena mengambil di kolom berbeda terlebih dahulu bisa tidak optimal, seperti pada kasus berikut:

3
111
100

Kompleksitas: $O(N)$.

H. Random Generator

Tag

kombinatorika, peluang

Pembahasan

Solusi 1: Observasi

Seperti definisi dari *expected value*, kontribusi dari sebuah susunan bit adalah hasil kali dari nilai susunan bit tersebut dan peluang munculnya susunan bit tersebut.

Contoh: susunan "10001" memiliki nilai (16 + 1) dan peluang $(\frac{P}{100})^2 \cdot (\frac{100-P}{100})^3$. Maka kontribusi dari "10001" adalah $(16+1) \cdot (\frac{P}{100})^2 \cdot (\frac{100-P}{100})^3$.

Salah satu strategi yang bisa dipakai adalah mencari kontribusi tiap bit pada *expected value*. Misal, kita ingin mencari kontribusi bit ke-$(N-1)$. Tentunya, bit ke-$(N-1)$ akan memberi kontribusi apabila ia bernilai 1. Oleh karena itu, kontribusinya:

$$2^{N-1} \cdot \frac{P}{100} \cdot \sum_{0}^{2^{N-1}-1} (\frac{P}{100})^{active\ bit\ in\ i} \cdot (\frac{100-P}{100})^{inactive\ bit\ in\ i}$$

Apabila dihitung, ternyata

$$\sum_{0}^{2^{N-1}-1} (\frac{P}{100})^{active\ bit\ in\ i} \cdot (\frac{100-P}{100})^{inactive\ bit\ in\ i}$$

pasti bernilai 1! Penjelasan intuitifnya adalah, apapun nilai bit ke-$(N-1)$, konfigurasi nilai bit sisanya pasti berupa 0/1, dan pasti valid. Hal ini terjadi akibat kita menginginkan *expected value* dari 0 sampai $2^N - 1$. Karena pasti valid, nilainya pasti 1. Sehingga, total kontribusi bit ke-$(N-1)$ adalah $2^{N-1} \cdot \frac{P}{100}$.

Bila diamati lebih lanjut, kontribusi bit ke-$(N-1)$ bisa digeneralisasi untuk semua bit ke-i. Akhirnya, setiap bit ke-i memiliki kontribusi sebesar $2^i \cdot \frac{P}{100}$ untuk setiap $0 \leq i < N$. Jadi, expected valuenya adalah:

$$\sum_{0}^{N-1} 2^i \cdot \frac{P}{100}$$

$$= \frac{P}{100} \cdot \sum_{0}^{N-1} 2^i$$

$$= \frac{P}{100} \cdot (2^N - 1)$$

Solusi 2: Rekursif

Solusi ini menggunakan strategi rekursif dalam penyelesaiannya. Bentuk umum dari rekursif untuk *expected value* adalah F(x) = transisi_i * kemungkinan_i + transisi_{i+1} * kemungkinan_{i+1} + …

Pada soal ini yang dapat diiterasi adalah bitnya. Dan terdapat 2 kemungkinan transisi untuk setiap bit, yaitu 0 dan 1.

Contoh: sekarang di state 10, bila ditambah 1 bit di belakangnya terdapat 2 kemungkinan, yaitu 100 ($(100 - P)$ %) atau 101 (P %).

Maka, kita bisa mendefinisikan $f(x)$ sebagai *expected value* apabila hanya tersisa x bit lagi, dengan pada setiap transisi, kita meletakkan *most significant bit*-nya. Penyelesaiannya adalah sebagai berikut:

$$f(0) = 0$$

$$f(N) = (transisi\ bit\ 1) \cdot \frac{P}{100} + (transisi\ bit\ 0) \cdot \frac{100 - P}{100}$$

$$= (f(N-1) + 2^{N-1}) \cdot \frac{P}{100} + f(N-1) \cdot \frac{100 - P}{100}$$

$$= f(N-1) \cdot \frac{P}{100} + 2^{N-1} \cdot \frac{P}{100} + f(N-1) \cdot \frac{100 - P}{100}$$

$$= f(N-1) + 2^{N-1} \cdot \frac{P}{100}$$

Apabila disimplifikasi, dapat didapatkan lagi rumus berikut:

$$f(N) = \sum_{0}^{N-1} 2^i \cdot \frac{P}{100}$$

$$= \frac{P}{100} \cdot \sum_{0}^{N-1} 2^i$$

$$= \frac{P}{100} \cdot (2^N - 1)$$

Kompleksitas: $O(1)$ atau $O(N)$.

G.