**PEMBAHASAN AGRICODE IPBPC MINGGU 2**

1. MINGGU DUA

hanya simulasikan sesuai perintah pada soal

kode (c++) : <http://ideone.com/2rSoxY>

1. Up Up Up Flying away

Cari nilai x[i] terkecil dari data kemudian selesaikan persamaan yang ditanyakan

**2\*x[i]+y>=k**

**Y>=k-2\*x[i]**

Sebelum output jawaban, cek apakan y<0 atau tidak jika y<0 maka outputkan 0 selain itu output y , perhatikan bahwa ini merupakan soal *multiple testcase* , cukup banyak submission yang salah disini.

Kode (c++) : <http://ideone.com/vyiCGc>

1. Paint Reverse

Simulasikan apa yang diperintahkan soal sesuai warna yang ditanyakan.

Kode (c++) : <http://ideone.com/N3fXFh>

1. FPB BESAR

Solusinya yaitu perhatikan nilai *a* , jika a==b maka fpbnya yaitu 2a selain itu fpb nya 1 , selain itu tinjau nilai *a* , a maksimal 63 , dengan kata lain kita menggunakan tipe data unsigned long long untuk dapat menampung nilai hingga 263 .

Kode (c++) : <http://ideone.com/WFeIt8>

1. Baloooooon

Solusi untuk soal ini yaitu *counting sort,*  counting terhadap jawaban yang bersangkutan, untuk setiap nilai *i* jika count[i]>=k maka hasil+=i\*k selain itu hasil+=i\*count[i].

Kode(c++) : <http://ideone.com/e778Ag>

1. Tidak Ada Deskripsi Untuk Soal Ini

Ini merupakan soal bonus, yang hanya perlu dilakukan yaitu blok kolom contoh input output , kemudia keluarkan hasilnya

Kode (c++) : <http://ideone.com/l5PUhi>

1. Tangga Mangga

Ini merupakan satu dari dua soal yang membutuhkan sedikit bumbu algoritma, untuk dapat solve soal ini kita tidak dapat melakukan searching linear per query karena time limitnya yang begitu kecil (50ms), setidaknya ada dua cara agar dapat solve soal ini :

1. Binary search, untuk mendapatkan batas bawah, kita dapat melakukan teknik binary search , binary search per query akan mereduksi kompleksitas cukup drastis ( O(log(n)) per query)
2. Karena *k* nya yang kecil (1<=k<=10000) maka kita dapat melakukan precompute jawaban untuk setiap nilai *k* , setelah lakukan precompute ,untuk setiap query outputkan *data[k]* .untuk nilai k yang besar (misalnya 2 milyar) ,kita tidak dapat melakukan precompute karena membutuhkan memori yang besar dan time limit yang tidak memungkinkan.

Kode (c++) (binary search) : <http://ideone.com/WHsKlc>

1. Apel Boni.

Ini merupakan soal *decider problem* untuk minggu ini, banyak sekali peserta yang terjebak dengan cara *greedy* (mengambil angka terbesar-> max(data[i],data[i+1]) ) , cara ini tidak bekerja karena tidak terbukti kebenaran solusinya (banyak contoh testcase yang membuat gagal, tetapi hanya diberikan 1 testcase yang kuat , sehingga masih dapat mencapai nilai 80an) . untuk menyelesaikan soal ini setidaknya kita perlu tahu tentang paradigma *dynamic programming*.

Mengapa solusi *greedy* tidak bekerja ? contoh testcase:

-1 -2 -2 -2 -2 1.

Jika kita melakukan *greedy*, maka angka yang terpilih yaitu -1-2-2+1=-4 (untuk setiap langkah pilih max(data[i],data[i+1])), sedangkan solusi optimalnya yaitu -2-2+1=-3.

Inti kunci dari dynamic programming yaitu *“melihat ke belakang” ,* sebagai penggambaran , *“kita akan tahu benda ini bernama x ketika kita sebelumnya sudah diberi tahu terlebih dahulu bahwa benda tersebut bernama x “* . nah, di soal ini kita harus dapat mendefinisikan *f(x)* yang tepat untuk mendapatkan jawabannya. Berdasarkan soal , secara sederhana kita dapat mendapatkan solusi *f(x)* secara rekursif sebagai berikut :

\*data[x] merupakan nilai apel pada posis ke *x*

\*f(x) merupakan banyaknya jumlah apel maksimal dari posisi ke 1 hingga posisi ke-x

Setelah dapat mendefinisikan fungsi tersebut ada beberapa hal yang perlu diperhatikan , jika kita hanya menjalankan fungsi rekursif tersebut secara mentah, maka akan didapatkan verdict *time limit excedded* ,Karena fungsi akan berjalan eksponensial , f(x) akan berjalan berkali-kali (terdapat overlapping subproblem). Maka diperlukannya optimasi yaitu setidaknya terdapat dua cara dalam solve soal dynamic programming :

1. Top - Down

Kita menjalankan fungsi rekursif diatas dimulai dari *f(n) ,* hanya saja , kita harus menyimpan nilai dari setiap fungsi *f(x)* di dalam array, sehingga jika terjadi pemanggilan fungsi >1 kali , kita hanya perlu *return f[x]* (f[x] pada awalnya kita set nilai yang tidak memungkinkan untuk jawaban, contoh: 2 milyar,jika f[x]!=2milyar tandanya f(x) sudah pernah dikunjungi).

1. Bottom – Up

Jika top down dibutuhkan nya rekursif dan dijalankan mulai dari *f(n)* maka bottom - up tidak diperlukannya proses rekursif, melainkan iterative , yaitu kita mulai proses perhitungan dari f[1] ,f[2] kemudian untuk setiap f[x] , jawabannya yaitu f[x]=max(data[x]+f[x-1],data[x]+f[x-2]). Jawaban akhirnya yaitu f[n]

Kode (c++) (Bottom-Up) : <http://ideone.com/ZJKLg5>