TOKI Regular Open Contest #8 Editorial

(English version is available on page 5.)

Penulis soal

Div 2A	Selamat Datang Bonbon	**vincealek**
Div 2B	Bonbon Dapat Nama	**FaustaAD**
Div 1A	Bonbon dan String	**vincealek**
Div 1B	Bon Appétit	**mosesmayer**
Div 1C	Bonbon Memanjat Pohon	**mosesmayer**
Div 1D	Sampai Jumpa Bonbon	**AMnu**
Div 1E	Oleh-oleh Bonbon	**AMnu**

Div 2A. Selamat Datang Bonbon

Apabila sebuah benda seharga H diberi diskon Y%, maka harga benda tersebut menjadi H * (100-Y)/100. Apabila sebuah benda seharga H dikenakan pajak Z%, maka harga benda tersebut menjadi H * (100+Z)/100.
Maka hasil akhir dari pertanyaan adalah X * (100-Y) * (100+Z) / 10000. Hati-hati dengan tipe data karena perhitungan bisa saja menggunakan lebih dari 32-bit integer.
Kompleksitas waktu: O(1)

Div 2B. Bonbon Dapat Nama

Cara optimal untuk membuat kemunculan “bonbon” sebanyak-banyaknya adalah dengan menyusun string tersebut dengan bentuk “bonbonbon…”. Tentu kita hanya cukup menghitung banyaknya huruf b, o, dan n, dan lihat mana yang paling sedikit (katakan banyaknya x). Jawabannya adalah x-1, atau 0 jika x adalah 0.
Kompleksitas waktu: O(N)

Div 1A. Bonbon dan String

Untuk setiap karakter pada T, hitung banyaknya karakter yang ekuivalen dengan karakter tersebut. Dapat dilihat bahwa banyaknya karakter yang ekuivalen dengan karakter ke-i dari T adalah banyaknya kemunculan karakter ke-i dari S pada S ditambah 1. Setelah kita mengetahui banyaknya karakter ekuivalen dari masing-masing karakter, semua kemungkinan tinggal dikalikan saja.

Kompleksitas waktu: O(N)

Div 1B. Bon Appétit

Kunci dari solusinya adalah *meet-in-the-middle*. Kita hitung semua kemungkinan penjumlahan dari dua lemari pertama, dan hitung semua kemungkinan penjumlahan dari dua lemari terakhir. Sort semua hasil penjumlahan dua lemari pertama, kemudian lakukan binary search, untuk mencari banyaknya bilangan yang lebih kecil dari K - x, untuk x hasil penjumlahan dua lemari terakhir.

Kompleksitas waktu: $O(N^2 \log N)$

Div 1C. Bonbon Memanjat Pohon

Simpan 4 array lazy, sum, size, dan value. $Lazy_i$ adalah jumlah dari semua operasi update di subtree i. Sum_i adalah jumlah value dari subtree i. $Size_i$ adalah ukuran dari subtree i.

Untuk operasi 1 : update semua nilai sum dan value di semua ancestor dari U dan V.
Untuk operasi 2 : update lazy di node U dan sum dari semua ancestor dari U.
Untuk operasi 3 : jumlahkan semua value di shortest path dan lazy dari semua ancestor dari U dan V.
Untuk operasi 4 : jumlahkan sum di node U dan lazy dari semua ancestor dari U.

Kompleksitas waktu: $O(N + Q \times D)$ dengan D tinggi dari pohon

---

Div 1D. Sampai Jumpa Bonbon

Untuk kasus graf lurus berukuran Q+1 dan semua pasang node i dan i-1 terhubung, misalkan P_i adalah peluang dari node i mencapai node Q. Maka $P_i=(P_{i-1}+P_{i+1})/2$, $P_0=0$, dan $P_Q=1$. Bisa didapatkan $P_1=P_2/2$, $P_2=2P_3/3$, …, $P_i=i \times P_{i+1}/(i+1)$ …, $P_{Q-1}=(Q-1) \times P_Q/Q$. Selanjutnya, $P_i=i/Q$.

Untuk kasus “sungai huruf o”, jika kita berada di tengah 2 huruf “o” yang bersebelahan, maka peluang dari tengah ke kanan sama dengan dari tengah ke kiri. Maka pada kasus dua “o” bersebelahan bisa direpresentasikan sebagai garis lurus. Jika “o” bersebelahan dengan garis lurus, maka peluang ke arah “o” adalah P=2X/3 dan X=(P+1)/2, sehingga didapatkan P=½ dengan X peluang dari node 3i+1 di huruf “o”. Sehingga juga bisa direpresentasikan sebagai garis lurus.

Contoh kode:

```
for (int i=0; i<L; i++) {
	if (S[i]=='b') {
		A[3*i+1]=A[3*i]+4;
		A[3*i+2]=A[3*i]+2;
		A[3*i+3]=A[3*i]+6;
	}
	if (S[i]=='o') {
		A[3*i+1]=A[3*i]+1;
		A[3*i+2]=A[3*i]+1;
		A[3*i+3]=A[3*i]+2;
	}
	if (S[i]=='n') {
		A[3*i+1]=A[3*i];
		A[3*i+2]=A[3*i]+2;
		A[3*i+3]=A[3*i]+2;
	}
}
// Jawaban pada node ke-i adalah A[i] / A[N], di mana N panjang sungai
// Perhatikan bahwa A[N] = A[3*L] * M
```

Penjelasan yang lebih detail terdapat di sini.
Kompleksitas waktu: O(L+Q)

Div 1E. Oleh-oleh Bonbon

Misalkan souvenir yang dimaksud adalah string S sepanjang N.
Misalkan pula $cnt_{b,i}$, $cnt_{o,i}$, $cnt_{n,i}$ adalah banyaknya subsequence “...bonbonbon…” sepanjang i pada S yang masing-masing subsequencenya berakhir dengan huruf b, o, dan n.
Misalkan pula $sum_{b,i}$, $sum_{o,i}$, $sum_{n,i}$ adalah jumlah semua value dari subsequence “...bonbonbon…” sepanjang i pada S yang masing-masing subsequencenya berakhir dengan huruf b, o, dan n.

Cara menghitung *cnt* dan *sum* adalah, untuk setiap karakter c pada S:
Jika c == ‘b’, maka untuk semua i = 1 .. N:

- $cnt_{b,i} \mathrel{+}= cnt_{n,i-1}$
- $sum_{b,i} \mathrel{+}= cnt_{n,i-1} * a_i + sum_{n,i}$

Perhitungan yang serupa dilakukan untuk subsequence yang berakhiran dengan huruf o dan n.

Misalkan kita punya string T, yang merupakan gabungan dari banyak string S.
Misalkan $dpcnt_{b,i,j}$ adalah banyaknya subsequence "...bonbonbon…" sepanjang j yang berakhir dengan huruf b, pada string $T_{1..i*N}$, dengan setidaknya satu karakter dari masing-masing $T_{1..N}$, $T_{N+1..2N}$, …, $T_{(i-1)N..i*N}$ merupakan bagian dari subsequence.
Misalkan pula $dp_{sumb,i,j}$ adalah jumlah dari semua value subsequence "...bonbonbon…" sepanjang j yang berakhir dengan huruf b, pada string $T_{1..i*N}$, dengan setidaknya satu karakter dari masing-masing $T_{1..N}$, $T_{N+1..2N}$, …, $T_{(i-1)N..i*N}$ merupakan bagian dari subsequence.

Nilai dari *dpcnt* dan *dpsum* dihitung dengan cara:

$$dp_{cntb,i,j} = \sum_{k=1}^{j-1} dpcnt_{x,i-1,j-k} * dpcnt_{y,1,k}$$

dengan x dan y adalah k+1 dan k karakter sebelum b pada "...bonbonbon…" (sebagai contoh, jika k = 4, maka k karakter sebelum b pada "...bonbonbon…" adalah n).

$$dpsum_{b,i,j} = \sum_{k=1}^{j-1} dpcnt_{x,i-1,j-k} * dpsum_{y,1,k} + dpsum_{x,i-1,j-k} * dpcnt_{y,1,k}$$

dengan x dan y sama seperti perhitungan dpcnt di atas.

Base case-nya adalah $dpcnt_{x,1} = cnt_x$ dan $dpsum_{x,1} = sum_x$
Hal yang serupa dilakukan pula untuk huruf o dan n.

Setiap query kemudian bisa dijawab dengan:
$\sum C(M, i) * dpsum_{n,i,3P}$
Dengan C(n, k) adalah kombinasi n pilih k.

Kompleksitas waktu: $O((N + Q) \times P + P^3)$

TOKI Regular Open Contest #8 Editorial

Problem authors

Div 2A	Welcome Bonbon	**vincealek**
Div 2B	Bonbon Got a Name	**FaustaAD**
Div 1A	Bonbon and String	**vincealek**
Div 1B	Bon Appétit	**mosesmayer**
Div 1C	Bonbon Climbing Tree	**mosesmayer**
Div 1D	Farewell Bonbon	**AMnu**
Div 1E	Bonbon Souvenirs	**AMnu**

Div 2A. Welcome Bonbon

If an item with price H is discounted by Y%, its price became H * (100-Y)/100. If an item with price H is taxed by Z%, its price became H * (100+Z)/100.
The final answer to the problem is X * (100-Y) * (100+Z) / 10000. Be careful with the data type as it may require more than 32-bit integer.
Time complexity: O(1)

Div 2B. Bonbon Got a Name

The optimal way to maximize the number of “bonbon” appearances is to arrange the string with the form “bonbonbon…”. Now we only need to count the number of character b, o, and n, and look for the minimum count (say x). The answer is x-1, or 0 if x is 0.
Time complexity: O(N)

Div 1A. Bonbon and String

For every character in T, count the number of character that is equivalent to it. It can be seen that the number of characters that is equivalent to the i-th character of T is the number of the i-th

character of S on S plus one. Once we know the number of equivalent characters of each character, the number of equivalent strings is simply the product of them.

Time complexity: O(N)

Div 1B. Bon Appétit

The key of this solution is *meet-in-the-middle*. We compute all possible sums from the first two cupboards, and all possible sums from the last two cupboards. Sort all the sums of the first two cupboards, then perform binary search to compute the number of elements smaller than or equal to K - x, where x is the current sum from the last two cupboards.
Time complexity: $O(N^2 \log N)$

Div 1C. Bonbon Climbing Tree

We run DFS on the rooted tree and store the parent of every node. At every node, we store four values: the current value at the node, the subtree sum, the size, and the lazy value used to consider subtree updates.

For operation 1: update all the sum value for all ancestors of U and V.
For operation 2: update the lazy on node U and the sum of all ancestors of U.
For operation 3: sum all the values on the shortest path and the lazy for all ancestors of U and V.
For operation 4 : sum all the subtree sum from node U and lazy for all ancestors of U.
Time complexity: $O(N + Q \times D)$ where D is the height of the tree.

Div 1D. Farewell Bonbon

For the case of line graph of size Q+1 with each pair of node i and i-1 is connected, let P_i be the probaility that node i reach node Q. Then $P_i=(P_{i-1}+P_{i+1})/2$, $P_0=0$, and $P_Q=1$. We can obtain $P_1=P_2/2$, $P_2=2P_3/3$, …, $P_i=i\times P_{i+1}/(i+1)$ …, $P_{Q-1}=(Q-1)\times P_Q/Q$. Hence, $P_i=i/Q$.

For the case “river with letter o”, if we are in between two adjacent letter “o”, then the probability from the center moving to the left is equal to the probability of moving to the right. Therefore the case adjacent “o” can be represented as a line. If “o” is adjacent to a line, then the probability towards “o” is P=2X/3 and X=(P+1)/2, then we obtain P=½ where X is the probability of node 3i+1 on letter “o”. Therefore, this can also be represented as a line.

Example code:
```
for (int i=0; i<L; i++) {
```

```
        if (S[i]=='b') {
                A[3*i+1]=A[3*i]+4;
                A[3*i+2]=A[3*i]+2;
                A[3*i+3]=A[3*i]+6;
        }
        if (S[i]=='o') {
                A[3*i+1]=A[3*i]+1;
                A[3*i+2]=A[3*i]+1;
                A[3*i+3]=A[3*i]+2;
        }
        if (S[i]=='n') {
                A[3*i+1]=A[3*i];
                A[3*i+2]=A[3*i]+2;
                A[3*i+3]=A[3*i]+2;
        }
}
// The answer for the i-th node is A[i] / A[N], where N is the river length
// Notice that A[N] = A[3*L] * M
```

A more detailed explanation is available here.
Time complexity: O(L + Q)

Div 1E. Bonbon Souvenirs

Let the souvenir from the problem be the string S of length N.
Let $cnt_{b,i}$, $cnt_{o,i}$, $cnt_{n,i}$ be the number of subsequences "...bonbonbon…" of length i on S where the subsequences end with the letter b, o, and n respectively.
Let $sum_{b,i}$, $sum_{o,i}$, $sum_{n,i}$ be the sum of all values of subsequences "...bonbonbon…" of length i on S where the subsequences end with the letter b, o, and n respectively.

To compute *cnt* and *sum*, for each character of S:
If c == 'b', the for all i = 1 .. N:

- $cnt_{b,i} \mathrel{+}= cnt_{n,i-1}$
- $sum_{b,i} \mathrel{+}= cnt_{n,i-1} * a_i + sum_{n,i}$

Similar computation are also done for subsequences that end with the letter o and n.

Let the string T be the concatenation of finitely many string S..
Let $dpcnt_{b,i,j}$ be the number of subsequences "...bonbonbon…" of length j that ends with letter b, on string $T_{1..i*N}$, where at least one character from each substring $T_{1..N}$, $T_{N+1..2N}$, …, $T_{(i-1)N..i*N}$ is part of the subsequences.

Let $dp_{sumb,i,j}$ be the sum of all values of subsequences "...bonbonbon…" of length j that ends with letter b, on string $T_{1..i*N}$, where at least one character from each substring $T_{1..N}$, $T_{N+1..2N}$, …, $T_{(i-1)N..i*N}$ is part of the subsequences

The value of *dpcnt* and *dpsum* can be computed as:

$$dp_{cntb,i,j} = \sum_{k=1}^{j-1} dpcnt_{x,i-1,j-k} * dpcnt_{y,1,k}$$

With x and y are the k+1 and k characters before the letter 'b' on "...bonbonbon…" (as an example, if k = 4, then k character before b on "...bonbonbon…" is 'n').

$$dpsum_{b,i,j} = \sum_{k=1}^{j-1} dpcnt_{x,i-1,j-k} * dpsum_{y,1,k} + dpsum_{x,i-1,j-k} * dpcnt_{y,1,k}$$

with x and y defined similarly as the dpcnt above.

The Base case is $dpcnt_{x,1} = cnt_x$ and $dpsum_{x,1} = sum_x$
Same computation are done for letter 'o' and 'n'.

Each query can be answered as
$\sum C(M, i) * dpsum_{n,i,3P}$
where C(n, k) is combination n choose k.

Time complexity: $O((N + Q) \times P + P^3)$