AtCoder Beginner Contest 081 / AtCoder Regular Contest 086 解説

writer: semiexp 並びに anonymous

2017 年 12 月 9 日

For International Readers: English editorial starts on page 7.

A: Placing Marbles

それぞれの文字が 1 か 0 かを判定して 1 の数を数えればよいです．マスの数は高々 3 つなので，for 文を使わなくても簡単に実装できます．

```
#include <iostream>
#include <string>
using namespace std;
int ans;
string s;
int main(){
    cin >> s;
    if(s[0]=='1') ans++;
    if(s[1]=='1') ans++;
    if(s[2]=='1') ans++;
    cout << ans << endl;
}
```

B: Shift only

黒板に書いてある整数を管理しておいて,「書かれている整数がすべて偶数である限り」「書かれている整数すべてを 2 で割る」をシミュレートすればよいです.

あるいは，ほとんど同じことですが，最初書かれている整数それぞれに対して「最大で何回 2 で割れるか」を求め，その最小値をとってもよいです.

- C++ による解答例: `https://abc081.contest.atcoder.jp/submissions/1842477`
- Python による解答例: `https://abc081.contest.atcoder.jp/submissions/1842030`

C: Not so diverse

「できるだけ少ない数のボールの整数を書き換える」というのは,「できるだけ多くのボールの整数を，変更しないままにしておく」というのと同じです．ここで，変更を行わないボールたちに書かれている整数は K 種類以下でなければなりません．逆に，これが K 種類以下なら，他のボールに書かれている整数も，変更しないでおくボールに書かれている整数から選んで書き換えることで，全体として書かれている整数を K 種類以下にすることができます．

よって，問題は「整数を K 個選んで，それらのうちのどれかが書かれているボールの数を最大化する」と言い換えられます．選ぶ整数としては，できるだけそれが書かれているボールの数が多いようなものから順に選んでいくのが最適です．

最初書かれている整数として考えられるものは 1 以上 N 以下なので，各整数に対して「その整数が書かれているボールは何個あるか」を容易に求めることができます．この個数分布をソートして，大きいほうから K 個の和をとることで，書き換えないボールの個数の最大値がわかります．あとは N からこの値を引くと，求める値を計算することができます．

D: Non-decreasing

簡単な場合から考えていきます．a が非負整数のみからなる場合を考えてみます．このとき，a の先頭からの累積和を取った数列は明らかに条件を満たします．よって，$a_2 += a_1, a_3 += a_2, \ldots, a_N += a_{N-1}$ という順番で操作をすればそのような数列を作ることができ，$N-1$ 回の操作で条件を満たすことができます．a が 0 以下の整数のみからなる場合もほぼ同様の方法で $N-1$ 回の操作で条件を満たすことができます．

a に負の数も正の数も含まれるとき，N 回以下の操作で a が非負整数のみからなるように，あるいは 0 以下の整数からなるようにすることができるかどうかを考えてみます．$\mathrm{MAX} := \max(a), \mathrm{MIN} := \min(a)$ とします．このとき，$|\mathrm{MAX}| \geq |\mathrm{MIN}|$ ならば，すべての数に MAX を足せば a を非負整数のみからなる数列にすることが可能です．同様に，$|\mathrm{MAX}| < |\mathrm{MIN}|$ ならば，すべての数に MIN を足せば a を 0 以下の整数のみからなる数列にすることが可能です．

以上より $2N$ 回以下の操作で条件を満たすことができることが分かりました．これは $O(N)$ で実行可能です．

E: Smuggling Marbles

同じ頂点の上に複数のビー玉が乗ったとき，木の上から取り除かれるという手順について考えます．ある 2 つの頂点に置かれたビー玉が同じ頂点の上に乗るためには，これらの頂点の根からの距離が等しい必要があります．よって，根からの深さ d の頂点にのみビー玉を置くとき，最終的に箱にビー玉が移動するようなビー玉の初期配置がいくつあるかを考えていくことにします．

$dp[i][j] =$ 頂点 i を根とする部分木に着目したとき，頂点 i に $j(0 \leq j \leq 1)$ 個のビー玉が置かれるような頂点 0 からの距離が d の頂点へのビー玉の配置の数，として DP をすることで $O(N)$ で数え上げることができます．なお，DP の計算途中では頂点にビー玉が置かれる個数が 2 個以上になりえますが，最終的に取り除かれるため 2 個として扱っても問題ありません．こうして得られた結果は，深さ d 以外の頂点の配置を考慮していないので，$dp[0][1] \times 2^{N-\text{深さ } d \text{ にある頂点の個数}}$ が求めるべき答えです．これを各深さごとに行えば $O(N^2)$ で数え上げることができ部分点を獲得できます．

ここで，この DP には無駄な遷移が多く含まれていることに着目します．例えば，ある頂点に複数個のビー玉が置かれるためにはその頂点の直接の子が 2 つ以上ある必要がありますが，先程の DP では子が 1 つしかないような場合でも毎回計算を行っていました．これらをうまく無視して計算する方法を考えます．

$dp[i][d][j] =$ 頂点 i に j 個のビー玉が置かれるような，頂点 i から深さ d の頂点へのビー玉の初期配置の数，とします．この DP の d についてデックを用いて動的に管理をします．$d = 0$ については，頂点 i のみ考えればよいため，最後にデックの先頭に付け加えることにします．それ以外の深さの更新については以下の図 1 のように子頂点どうしの 2 つのデックのうち長さの短い方を長い方へとマージするような要領で更新をしていきます．このようにすることで全体として $O(N)$ で数え上げることができ，満点を獲得できます．

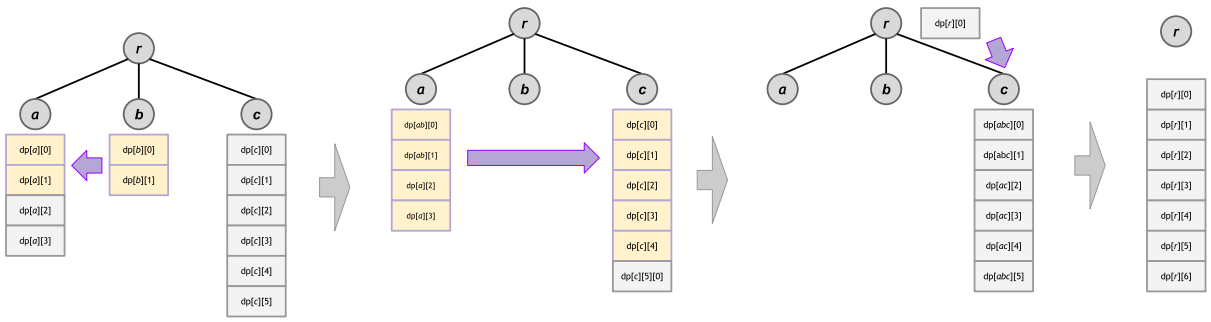


図 1: 頂点 r を根とする DP の更新の模式図

最後にこれが $O(N \log N)$ ではなく $O(N)$ になることの概略を述べておきます．全ての頂点には 0 が書かれているとします．根から距離 d に a_d 個の頂点があるとします．まずこれら a_d 個の頂点たちを行きがけ順に並べます．その後，隣り合う頂点のペア (u, v) について，それぞれ u と v の LCA(最小共通先祖) であるような頂点に 1 を加算します．このとき，各頂点に書かれた整数の数が，その頂点において，根から深さ d の頂点たちに対する DP の更新を行う必要がある (あるいはデックのマージを行う) 回数になります．これらの和は明らかに $a_d - 1$ 回となります．よって，更新の回数は $O(\sum_{d=0}^{N} \max(0, a_d - 1)) = O(N)$ となります．

F: Shift and Decrement

操作 B を行う回数を k とします．入力で与えられる整数は $10^{18} < 2^{60}$ 以下なので，$0 \leq k \leq 60$ として問題ありません (60 回 B を行うとすべての整数が 0 になるため，61 回目を行う意味はありません)．$i = 0, 1, \ldots, k$ に対して，i 回目の操作 B と $i+1$ 回目の操作 B の間で行う操作 A の回数を p_i で表すことにします (ただし，p_0 は「最初の操作 B を行う前に行う操作 A の回数」，p_k は「最後の操作 B を行った後に行う操作 A の回数」とします)．

p_i を $2d$ 減らして p_{i+1} を d 増やしても，黒板上の整数の書かれ方は変わらないことに注意します．このことから，$i = 0, 1, \ldots, k-1$ に対して，$p_i = 0$ または $p_i = 1$ としてよいということがわかります．(なぜならば，$p_i \geq 2$ であれば，p_i を 2 減らして p_{i+1} を 1 増やすことで，最終的な黒板上の整数の書かれ方を保ちつつ，操作回数を減らすことができるからです．) また，p_i を 1 減らして p_0 を 2^i 増やしても，黒板上の整数の書かれ方は変わらないことも従います．よって，$\{p_i\}$ に対応する一連の操作は，「操作 A を $P = \sum_{i=0}^{k-1} 2^i p_i$ 回行い，操作 B を k 回行い，最後に操作 A を p_k 回行う」一連の操作と等価です．$p_i = 0, 1$ としたので，$0 \leq P \leq 2^k - 1$ が成り立ちます．

A_i を 2^k で割った商を B_i，余りを C_i $(0 \leq C_i \leq 2^k - 1)$ とします．また，C_i $(i = 1, \ldots, N)$ から重複を除去し，ソートしたものを $C'_1 \leq C'_2 \leq \cdots \leq C'_M$ とします．$0 \leq P \leq 2^k - 1$ であることから，一連の操作によって A_i は次の値になることがわかります．

- $P \leq C_i$ ならば，$B_i - p_k$
- $C_i \leq P$ ならば，$B_i - p_k - 1$

よって，すべての i に対して「$P \leq C_i$ かどうか？」が同じであれば，異なる P に対しても同じ整数の書かれ方が得られることがわかります．ゆえに，考えるべき P は次のように区分されます．

- 0 以上 C'_1 以下
- $C'_1 + 1$ 以上 C'_2 以下
- $\ldots$
- $C'_{M-1} + 1$ 以上 C'_M 以下

ここで，P ごとに実際に行われる操作 A の回数は，2 進表記したときに現れる 1 の個数と等しいです．この操作回数をできるだけ少なくすることを考えると，P としては「区分の範囲内で，最も 2 進表記における 1 の個数が少なくなるもの」を用いるべきであることがわかります (具体的方法は後述します)．なお，$C'_M + 1$ 以上については，$P = 0$ として p_k に 1 を加えるのと同じであることから，考える必要がありません．

よって，各 k に対して考えるべき P の候補が $M \leq N$ 通りにまで絞られました．それぞれの候補に対して，最後の操作 B を行うまでの操作回数を計算することができるので，p_k の範囲を求めることができます．(ここで，負の整数が現れてはいけないことに注意します．) 黒板に書かれているすべての整数に同じ整数を加減することで移り変わるような整数の書かれ方同士を同じグループとみなすことにすると，p_k を変更しても所属するグループは変わりません．なので，考えるべきグループは全体で (k, P のとり方に対応して) $61 \times N$ 個以下です．それぞれのグループごとに，「A_1 は最終的に何に書き換えられるか」としてありうる範囲の集合を求めることができます．このような範囲も合計で $61 \times N$ 個以下です．あとは，各グループごとに，範囲の重複を除いて可能なパターン数を求めればよいです．

このアルゴリズムの計算量は，グループおよび範囲が全部で $O(\log A_{\max} \cdot N)$ 個あり，グループのインデックスは $N-1$ 要素の配列で管理できることから，$O(\log A_{\max} \cdot N^2 \log(\log A_{\max} \cdot N))$ になります．

最後に，X より大きく Y 以下の整数で $(X < Y)$，最も 2 進表記における 1 の個数が少なくなるものを求める方法を述べておきます．2 進表記において X, Y で異なる値になる桁が存在するので，そのうち最も上位のものを l ビット目とします．すると，l ビット目より上位のビットについては，X, Y と同じにするほかありません．一方，l ビット目およびそれより下位のビットをすべて 0 にすることはできない (X 以下になるため) ので，少なくともこの範囲で一つは 1 が必要です．ところで，l ビット目のみを 1 にし，より下位のビットはすべて 0 にすると，X より大きく Y 以下となります．よって，このようにして得られた整数が，最も 2 進表記における 1 の個数が少なくなるものになります．

AtCoder Beginner Contest 081 / AtCoder Regular Contest 086 Editorial

writer: semiexp and anonymous

December 10th, 2017

A: Placing Marbles

```
#include <iostream>
#include <string>
using namespace std;
int ans;
string s;
int main(){
    cin >> s;
    if(s[0]=='1') ans++;
    if(s[1]=='1') ans++;
    if(s[2]=='1') ans++;
    cout << ans << endl;
}
```

B: Shift only

- C++ implementation: `https://abc081.contest.atcoder.jp/submissions/1842477`
- Python implementation: `https://abc081.contest.atcoder.jp/submissions/1842030`

C: Not so diverse

For each integer x between 1 and N, count the number of occurrences of x in the input, and sort it. The answer is N minus the sum of K greatest integers in the frequency list.

D: Non-decreasing

In case all numbers in a are non-negative, the sequence of prefix sums (i.e., $a_1, a_1 + a_2, a_1 + a_2 + a_3, \cdots$) is non-decreasing. Thus, the sequence of operations "$a_2 \mathrel{+}= a_1, a_3 \mathrel{+}= a_2, \ldots, a_N \mathrel{+}= a_{N-1}$" satisfies the condition. Similarly, in case all numbers are non-positive, "$a_{N-1} \mathrel{+}= a_N, a_{N-2} \mathrel{+}= a_{N-1}, \ldots, a_1 \mathrel{+}= a_2$" satisfies the condition.

In general, there are two cases. Let $\mathrm{MAX} := \max(a), \mathrm{MIN} := \min(a)$. If $|\mathrm{MAX}| \geq |\mathrm{MIN}|$, we first add MAX to all elements of a, and a will be a sequence of non-negative numbers. Then, we can make it non-decreasing as we described above. Similarly, if $|\mathrm{MAX}| < |\mathrm{MIN}|$, we first add MIN to all elements of a, and a will be a sequence of non-positive numbers.

In both cases, we need $2N - 1$ operations. This solution works in $O(N)$.

E: Smuggling Marbles

When multiple marbles are on the same vertex, they will be removed. This happens only when the marbles come from vertices of the same depth. Thus, we can handle each depth independently. For each d, consider all subsets of vertices of depth d, and count the number of subsets that will end up with a single marble at the root.

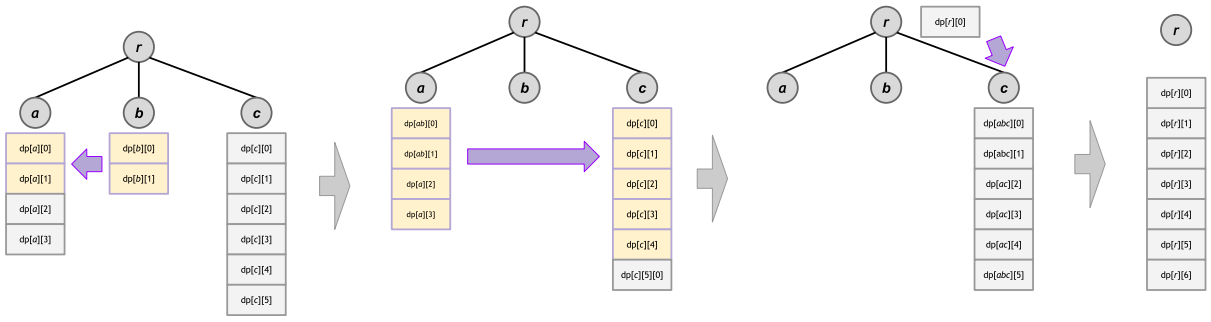
How can we solve this problem for a fixed d? For simplicity, from now on, we use probabilities - each vertex initially contains a marble with probability $1/2$. Let $dp[i][j]$ be the probability that vertex i contains $j (0 \leq j \leq 1)$ marble at the end, when we only consider the subtree rooted at i (and initially we put marbles only on vertices of depth d). The answer is sum of $dp[0][1] \times 2^N$ over all depths. This solution works in $O(N)$ per depth, and $O(N^2)$ in total. You can get partial score.

To make it faster, we compute these values for all depths at once. Let $dp[i][d][j]$ be the following value: Consider a subtree rooted at vertex i. We put marbles to some vertices of depth d (i.e., the distance to i is d), with probability $1/2$ for each such vertex. What is the probability that vertex i contains j marbles at the end?

Let's call a tuple of three doubles $dp[i][d] = (dp[i][d][0], dp[i][d][1], dp[i][d][2])$ "state". Here, $j = 2$ means that a vertex contains two or more marbles (it happens during the computation, as described below). Then, $dp[i]$ is a sequence of states $(dp[i][0], dp[i][1], ...)$ and its length is (the depth of subtree rooted at i) plus one. We represent it as a deque of states.

The figure below shows how to compute $dp[r]$ when it has three children a, b, c. While r has two or more children, we "merge" the shortest deque into the second shortest deque. For example, if $dp[a][i]$ contains x marbles and $dp[b][i]$ contains y marbles, the new deque $dp[ab][i]$ will contain $min(x + y, 2)$ marbles. It can be done in $O(min(len(dp[a]), len(dp[b])))$. Make sure that you don't create a new deque - if $len(dp[a]) > len(dp[b])$, reuse $dp[a]$, and rewrite its first $len(dp[b])$ elements.

Then, if r has only one children, you just need to push a state $(1/2, 1/2, 0)$ to the front. Don't forget to convert $j = 2$ to $j = 0$ at the end.



Why is it $O(N)$ (and not $O(NlogN)$)? Suppose that initially all vertices contain 0. Let a_d be the number of vertices of depth d. First, we sort these vertices in dfs pre-order. For each consecutive pair of two vertices (u, v) (of depth d), add 1 to the integer written on LCA of u and v. Then, the integers written on each vertex represents the number of merges that comes from vertices of depth d. The sum of those integers is $a_d - 1$. Thus, the total number of merges is $O(\sum_{d=0}^{N} \max(0, a_d - 1)) = O(N)$.

F: Shift and Decrement

Suppose that we perform operation A k times. Since $10^{18} < 2^{60}$, we can assume that $0 \leq k \leq 60$ (after 60 operations all numbers will be zero). For each $i = 0, 1, \ldots, k$, let p_i be the number of operation Bs we perform between i-th A and $i+1$-th A. (p_0 and p_k are defined naturally).

An operation A maps an integer x to $floor(x/2)$, and an operation B maps an integer x to $x-1$. Notice that we don't need to take floor after each opertaion A - instead, we compute numbers as doubles, and only at the end of all operations we apply the floor function. Thus, the sequence of operations is equivalent to the following:

- Perform operation B $P = \sum_{i=0}^{k-1} 2^i p_i$ times.
- Perform operation A k times.
- Perform opertaion B p_k times.

Also, we can assume that $0 \leq P \leq 2^k - 1$ (otherwise, we can decrease P by 2^k and increment p_k by one, and decrease the total number of operations).

Let $A_i = 2^k \times B_i + C_i$ ($0 \leq C_i \leq 2^k - 1$). Let $C'_1 \leq C'_2 \leq \cdots \leq C'_M$ be a sorted list of integers that appear in C (after removing duplicates).

After all operations, the value of A_i will be following:

- If $P \leq C_i$, $B_i - p_k$
- If $C_i \leq P$, $B_i - p_k - 1$

Thus, the only important thing for P is "Is $P \leq C_i$?" for each C_i. P will be in one of the following intervals:

- $[0, C_1]$
- $[C_1 + 1, C_2]$
- $\ldots$
- $[C_{M-1} + 1, C_M]$

The number of opertaion Bs in the first phase is $popcount(P)$ (the number of 1s in the binary representation of P). Thus, in each interval, we should choose P that minimizes this value (see below for the way to compute P).

Now, for each k, we have only $M \leq N$ candidates for P. For each candidate, we can compute the valid range for p_k (to make the total number of opertaions at most K, make sure that it won't be negative).

We consider two sequences of integers to be in the same group if we can convert one of them to the other by performing operation Bs. Since the value of p_k doesn't affect its group, we need to consider at most $61 \times N$ groups (at most one group per each (k, P) pair). For each group, compute the possible range for A_1 (the total number of those ranges is also at most $61 \times N$). Then, for each group, we compute the number of possible values for A_1 (after remoging duplicates).

There are $O(\log A_{\max} \cdot N)$ groups and ranges, and since each group can be represented as a sequence of length $N-1$, this solution works in $O(\log A_{\max} \cdot N^2 \log(\log A_{\max} \cdot N))$ time.

Now, we describe how to compute an integer between $X+1$ and Y ($X < Y$) that minimizes its popcount. Let l be the largest integer such that the l-th digits in X and Y differ. Since the integer we choose must be between $X+1$ and Y, all digits above l-th must be the same as X, Y. If we make the l-th digit one and the remaining digits zero, the number will be between $X+1$ and Y. This is optimal because it only adds one to popcount (and we can't make all digits up to l-th zeroes because the number must be at most $X+1$).