ABC #082 / ARC #087 Editorial

writer : sugim48

2017 年 12 月 16 日

For International Readers: English editorial starts on page 5.

A: Round Up the Mean

- C++ のコード例 : https://beta.atcoder.jp/contests/abc082/submissions/1872729
- Java のコード例 : https://beta.atcoder.jp/contests/abc082/submissions/1872732
- Python 3 のコード例 : https://beta.atcoder.jp/contests/abc082/submissions/1872734

B: Two Anagrams

s の文字を昇順にソートし，t の文字を降順にソートした後，$s < t$ を判定すればよいです．

- C++ のコード例 : https://beta.atcoder.jp/contests/abc082/submissions/1872811
- Java のコード例 : https://beta.atcoder.jp/contests/abc082/submissions/1872812
- Python 3 のコード例 : https://beta.atcoder.jp/contests/abc082/submissions/1872814

C: Good Sequence

それぞれの正の整数 x について，a 中の値 x の個数を n_x とします．$n_x < x$ の場合，a 中の値 x はすべて取り除かなければならないので，n_x 回の操作が必要です．$n_x \geq x$ の場合，a 中の値 x の個数を x にすればよいので，$n_x - x$ 回の操作が必要です．すべての x についてこれらの操作回数を合計したものが答えです．

実際には，すべての x ではなく，a 中の値 x のみ調べればよいです．x の値は最大で 10^9 と大きいので，普通の配列の代わりに連想配列を用いて各 n_x を数えればよいです．

- C++ のコード例 : https://beta.atcoder.jp/contests/abc082/submissions/1872852

D: FT Robot

s の長さを N とし，目標の座標を (x_t, y_t) とします．

まず思いつく解法は，$\mathrm{dp}_{i,x,y,k} :=$ (i 文字目の命令後に座標 (x, y) に向き $k \in \{$ 左, 上, 右, 下 $\}$ でいられるか)

と定義し，DP をする解法です．しかし，この DP の状態数は $\mathrm{O}(N^3)$ で，遷移は $\mathrm{O}(1)$ なので，全体の時間計算量は $\mathrm{O}(N^3)$ となり，TLE してしまいます．

ロボットの動き方をよく観察することで，DP の状態数を減らしましょう．命令列を `T` を区切り文字として分割したとき，i 番目の区間に含まれる `F` の個数を d_i とします．このとき，ロボットの動き方は

- 右に距離 d_1 だけ動く
- 上または下の好きな向きに距離 d_2 だけ動く
- 左または右の好きな向きに距離 d_3 だけ動く
- 上または下の好きな向きに距離 d_4 だけ動く
- 左または右の好きな向きに距離 d_5 だけ動く
- 以下同様

となっています．よって，x 軸方向の移動と y 軸方向の移動を独立に考えることができます．例えば，x 軸方向の移動については，

- 右に距離 d_1 だけ動く
- 左または右の好きな向きに距離 d_3 だけ動く
- 左または右の好きな向きに距離 d_5 だけ動く
- 以下同様

という移動を終えた後，x 座標が x_t になっているようにできるか判定すればよいです．

これは次のような DP で計算できます．

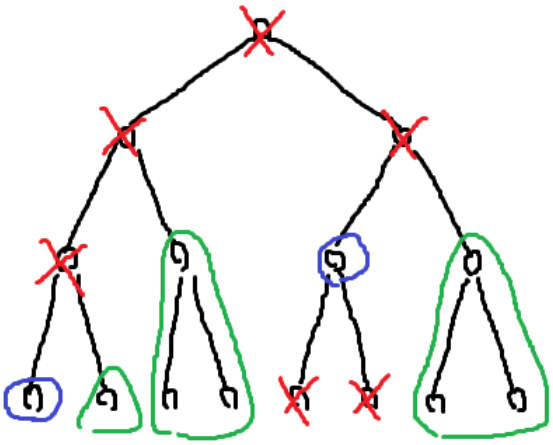
- 定義: $\mathrm{dp}_{i,x} :=$ (i 回目の x 軸方向の移動の後に x 座標が x になっているようにできるか).
- 初期化: 1 回目の x 軸方向の移動距離を d_1 とすると，$\mathrm{dp}_{0,d_1} = \mathrm{true}$.
- 漸化式: i ($i \geq 2$) 回目の x 軸方向の移動距離を d_i とすると，$\mathrm{dp}_{i,x} = \mathrm{dp}_{i-1,x-d_i} \vee \mathrm{dp}_{i-1,x+d_i}$.

この DP の状態数は $\mathrm{O}(N^2)$ で，遷移は $\mathrm{O}(1)$ なので，全体の時間計算量は $\mathrm{O}(N^2)$ となり，十分に高速です．y 軸方向の移動についても，同様の DP で計算できます．

E: Prefix-free Game

長さ L 以下の `0`, `1` のみからなる文字列全体からトライ木を作ると，これは高さ $L+1$ の完全二分木となります．ただし，完全二分木の高さは，根から葉までのパスに含まれる頂点の個数と定義します．この完全二分木を用いると，ゲームは次のように言い換えられます．

> 高さ $L+1$ の完全二分木がある．最初，いくつかの頂点には駒が置かれている．ただし，どの駒どうしも先祖-子孫の関係ではない．Alice と Bob は交互に，駒のない頂点をひとつ選んで駒を置く．ただし，どの駒どうしも先祖-子孫の関係ではないという条件を保つようにする．先に駒を置けなくなった方が負けである．



上図は初期状態の例 ($L = 3$) です．青いマル印は駒を表しており，赤いバツ印は駒を置けない頂点を表しています．このとき，駒を置ける頂点の集合は，いくつかの完全二分木 (緑の三角形) の和になっています．これらの完全二分木は先祖-子孫の関係ではないので，独立に考えることができます．また，完全二分木のある頂点に駒を置くと，その完全二分木はいくつかの完全二分木に分裂します．具体的には，高さ i の完全二分木の深さ j $(0 \leq j \leq i-1)$ の頂点に駒を置くと，高さ $i-1, i-2, ..., i-j$ の完全二分木 1 個ずつに分裂します．

以上より，ゲームはさらに次のように言い換えられます．

> 最初，いくつかのアイテムがある．各アイテムには正整数のレベルが設定されている．Alice と Bob は交互に次の操作を行う．
>
> - アイテムを 1 個選ぶ．選んだアイテムのレベルを i とする．選んだアイテムを取り除き，整数 k $(1 \leq k \leq i)$ を選んでレベル $i-1, i-2, ..., k$ のアイテムを 1 個ずつ追加する．
>
> 先に操作を行えなくなった方が負けである．

このようなゲームの勝敗判定には Grundy 数が有用です．レベル i $(i \geq 1)$ のアイテムの Grundy 数を g_i とします．すると，g_i は

- 0
- g_{i-1}
- $g_{i-1} \oplus g_{i-2}$
- $\vdots$
- $g_{i-1} \oplus g_{i-2} \oplus \cdots \oplus g_1$

に含まれない最小の非負整数です．実験すると，$(g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, \ldots) = (1, 2, 1, 4, 1, 2, 1, 8, \ldots)$ となっています．実験結果から分かるように，$g_i =$ (i を割り切る最大の 2 の冪) であることが示せます．各アイテムの Grundy 数の総 XOR が 0 でないならば Alice が勝ち，0 ならば Bob が勝ちます．

元の問題に戻ると，レベル i のアイテムは高さ i の平衡二分木に対応します．よって，初期状態におけるそれぞれの完全二分木の高さを求められればよいことになります．これは，入力文字列から作られるトライ木を辿り，子の個数がちょうど 1 個の頂点を見ていくことで可能です．

F: Squirrel Migration

まず，移動距離の総和を最大化するような移動方法がどのようなものか考えます．そのために，移動距離の総和の最大値がどのような値か考えます．まずは，移動距離の総和の上限を見積もってみましょう．ある移動方法に対して，各辺 e について e を通過したリスの個数を s_e とします．すると，移動距離の総和は $\sum_e s_e$ となります．ここで，辺 $e=(v,w)$ の v 側の頂点数を n_v とし，w 側の頂点数を n_w とすると，$s_e \le 2\min\{n_v, n_w\}$ です．よって，移動距離の総和の上限はこれらの総和となります．実は，この上限を達成するような移動方法が常に存在します．それは次のような移動方法です．

木の重心に注目します．重心は 1 個または 2 個です．重心が 1 個の場合，それを c とします．すると，どのリスの移動経路にも c が含まれるとき，かつそのときに限り，移動距離の総和は上限を達成することが分かります．また，重心が 2 個の場合，それらを c_1, c_2 とします．すると，どのリスの移動経路にも 辺 (c_1, c_2) が含まれるとき，かつそのときに限り，移動距離の総和は上限を達成することが分かります．

以上の移動方法を数え上げます．重心が 2 個の場合の方が簡単なので，先に考えます．重心を c_1, c_2 とします．さらに，c_1 側の部分木を T_1 とし，c_2 側の部分木を T_2 とします．このとき，木の頂点数 N は偶数であり，T_1, T_2 の頂点数はともに $N/2$ です．T_1 の各リスは，T_2 のどの頂点へも移動できます．同様に，T_2 の各リスは，T_1 のどの頂点へも移動できます．よって，移動方法は $(N/2)! \times (N/2)!$ 通りです．

次に，重心が 1 個の場合を考えます．重心を c とします．さらに，c の周りの部分木を $T_1, T_2, ..., T_K$ とします．このとき，c のリスはどの頂点へも移動できます．また，各 i について，T_i のリスは T_i 以外のどの頂点へも移動できます．この移動方法を $\mathrm{O}(N^2)$ 時間で直接数え上げるのは難しいです．

そこで，包除原理を用います．$T := T_1 \cup T_2 \cup \cdots \cup T_K$ とします (すなわち，T は木の頂点集合から c を除いたもの)．すると，答えは $\sum_{S \subseteq T} (-1)^{|S|} f_S$ となります．ただし，$f_S :=$ (「S 中の各リスは自分の部分木の頂点へ移動する」という条件を満たす移動方法の通り数) です．各 $S_1 \subseteq T_1$, ..., $S_K \subseteq T_K$ について，次が成り立ちます．

$$f_{S_1 \cup \cdots \cup S_K} = \left(N - \sum_{k=1}^{K} |S_k|\right)! \prod_{k=1}^{K} \binom{|T_k|}{|S_k|}^2 |S_k|!$$

よって，答えは

$$\sum_{S_1 \subseteq T_1, \ldots, S_K \subseteq T_K} (-1)^{\sum_{k=1}^{K} |S_k|} \left(N - \sum_{k=1}^{K} |S_k|\right)! \prod_{k=1}^{K} \binom{|T_k|}{|S_k|}^2 |S_k|!$$

となります．$\binom{|T_k|}{|S_k|}^2 |S_k|!$ の部分は，各 k ごとに独立に計算できることに注目します．また，$(-1)^{\sum_{k=1}^{K} |S_k|} \left(N - \sum_{k=1}^{K} |S_k|\right)!$ の部分は，$\sum_{k=1}^{K} |S_k|$ さえ分かっていれば計算できることに注目します．

以上の考察をもとに DP を設計します．

- 定義: $\mathrm{dp}_{i,x} := \left(\sum_{k=1}^{i} |S_k| = x \text{ のとき } \prod_{k=1}^{i} \binom{|T_k|}{|S_k|}^2 |S_k|! \text{ の総和}\right)$.
- 初期化: $\mathrm{dp}_{0,0} = 1$.
- 遷移: 各 $0 \le y \le |T_i|$ について，$\mathrm{dp}_{i,x+y}$ に $\mathrm{dp}_{i-1,x} \times \binom{|T_i|}{y}^2 y!$ を足し込む．

x の状態数は $\mathrm{O}(N)$ で，遷移は $1 \le i \le K$ の合計で $\mathrm{O}(N)$ なので，全体の時間計算量は $\mathrm{O}(N^2)$ となり，十分に高速です．最終的に，$\sum_{x=0}^{N-1} (-1)^x (N-x)! \times \mathrm{dp}_{K,x}$ が答えです．

ABC #082 / ARC #087 Editorial

writer : sugim48

December 16, 2017

A: Round Up the Mean

- C++ Code Example : https://beta.atcoder.jp/contests/abc082/submissions/1872729
- Java Code Example : https://beta.atcoder.jp/contests/abc082/submissions/1872732
- Python 3 Code Example : https://beta.atcoder.jp/contests/abc082/submissions/1872734

B: Two Anagrams

Sort s, sort t, reverse t, and check if $s < t$.

- C++ Code Example : https://beta.atcoder.jp/contests/abc082/submissions/1872811
- Java Code Example : https://beta.atcoder.jp/contests/abc082/submissions/1872812
- Python 3 Code Example : https://beta.atcoder.jp/contests/abc082/submissions/1872814

C: Good Sequence

Let n_x be the number of occurrences of x in a. If $n_x < x$, add n_x to the answer.
If $n_x \geq x$, add $n_x - x$ to the answer.

- C++ Code Example : https://beta.atcoder.jp/contests/abc082/submissions/1872852

D: FT Robot

Let N be the length of s and (x_t, y_t) be the coordinates of the goal.

Let's separate the sequence of operations by `T`, and let d_i be the number of `F` in the i-th part. Then, the movement of the robot can be described as follows:

- Move d_1 to right.
- Move d_2 upward or downward.
- Move d_3 to left or right.
- Move d_4 upward or downward.
- Move d_5 to left or right.
- And so on.

Thus, the x-coordinate and the y-coordinate can be handled independently. For example, for the x-coordinate, the robot will perform the following operations:

- Move d_1 to right.
- Move d_3 to left or right.
- Move d_5 to left or right.
- And so on.

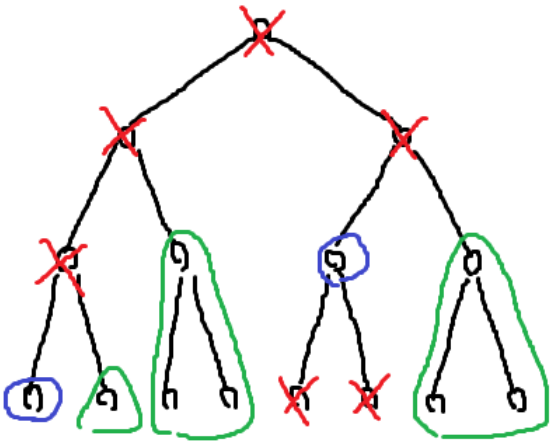
We want to check if x can be x_t after all operations. This can be done by a straightforward DP in $O(N^2)$. The y-coordinate can be handled similarly.

E: Prefix-free Game

Let's construct a trie of all strings of length L. It will be a complete binary tree. On this tree, the game can be described as follows:

> Initially, some vertices contain tokens. Alice and Bob alternately put tokens. Here, no two tokens can be put on two vertices with "ancestor - descendant" relation. The player who fails to put a token loses.

For example, the picture below shows the case with $L = 3$ and two strings 000 and 10.



Blue vertices contain tokens, red vertices are forbidden, and you can put tokens on green vertices. The green vertices will be a disjoint union of complete binary trees, and each part (complete binary tree) can be handled independently. Thus, we can use Grundy Numbers.

Let g_i be the Grundy Number of a complete binary tree of height i (i.e., a complete binary tree with i layers). If you perform an operation on a complete binary tree of height i, it will be splitted into complete binary trees with heights $i-1$, $i-2$, ..., $i-j$ for some j $(0 \leq j \leq i-1)$. Thus, g_i is the smallest integer that doesn't appear in the following:

- 0
- g_{i-1}
- $g_{i-1} \oplus g_{i-2}$
- $\vdots$
- $g_{i-1} \oplus g_{i-2} \oplus \cdots \oplus g_1$

If you perform an experiment, you get $(g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, \ldots) = (1, 2, 1, 4, 1, 2, 1, 8, \ldots)$. In general, g_i is the maximum power of two that divides i. (We omit the proof, it is straightforward.) If the XOR of all Grundy Numbers is 0, Bob wins, otherwise Alice wins.

To get the set of green binary trees, you should construct a trie for all input strings, and check all nodes with exactly one child.

F: Squirrel Migration

Consider an edge $e = (v, w)$ in the tree. Let n_v, n_w be the sizes of two subtrees we get when we cut the tree by e. Then, the number of squirrels that pass through this edge (call it s_e) is st most $2\min\{n_v, n_w\}$.

The total distance defined in the statement is the sum of s_e, thus it is at most $\sum 2\min\{n_v, n_w\}$. It turns out that this value is always achievable. There are two different cases:

The tree has two centroids

Call them c_1, c_2. The maximum can be achieved when all squirrels pass through an edge between c_1 and c_2. There are $(N/2)! \times (N/2)!$ such ways.

The tree has one centroid

Call it c. The maximum can be achieved when all squirrels pass through the vertex c.

Let T_1, T_2, ..., T_K be the subtrees we get when we remove c from the tree. We want to count the number of permutations such that for each i, no squirrel in T_i moves to a vertex in T_i.

We use inclusion-exclusion principle. Let $T := T_1 \cup T_2 \cup \cdots \cup T_K$ (i.e., the entire vertex set minus c). The answer is $\sum_{S \subseteq T} (-1)^{|S|} f_S$, where f_S is the number of permutations of squirrels such that each squirrel in S moves to a vertex in the same group.

Now, for each k, we want to compute the sum of f_S for all S such that $|S| = k$.

Let g_k be the number of ways to choose k squirrels and there destinations, such that all chosen squirrels will travel to a vertex in the same group. Formally, g_k is the number of pairs of two k-tuples of vertices $(s_1, \ldots, s_k), (t_1, \ldots, t_k)$ such that:

- $s_1 < s_2 < \cdots$ (the order of chosen squirrels doesn't matter)
- For each i, s_i and t_i are in the same subtree (and they are not c).
- t_i are pairwise distinct.

Since we can freely choose destinations for unchosen squirrels, the sum of f_S for all S such that $|S| = k$ is $g_k(N-k)!$. Therefore, the answer is $\sum_{k=0}^{N-1} g_k(N-k)!(-1)^k$. Since g_k can be easily computed in $O(N^2)$ by a simple DP, we can also get the answer in $O(N^2)$.