diverta 2019 Programming Contest 解説

camypaper

2019 年 5 月 11 日

For International Readers: English editorial starts on page 8.

A: Consecutive Integers

連続する K 個の整数の最小値が 1 以上 $N-K+1$ 以下ならば、条件を満たします。よって $N-K+1$ を出力すればよいです。

C++ での解答例: https://atcoder.jp/contests/diverta2019/submissions/5322859

B: RGB Boxes

r と g を固定してみると $b = \frac{N-rR-gG}{B}$ と一意に定まることが分かります。これが非負整数となるかどうかを判定すればよいです。$N \leq 3000$ より、$0 \leq r, g \leq N$ の範囲で愚直に全探索が可能です。以上より、$O(N^2)$ で実行可能です。

C++ での解答例: https://atcoder.jp/contests/diverta2019/submissions/5323803

C: AB Substrings

それぞれの文字列中に登場する AB は予め計算できます。文字列を並び替えたときの影響だけを考えましょう。

文字列を並び替えた結果、答えに影響するのは先頭と末尾の 2 箇所です。すると、N 個の文字列は以下の 4 種類のいずれかに分類できます。

1. 先頭が B、末尾が A
2. 先頭が B、末尾が A でない
3. 先頭が B でなく、末尾が A
4. 先頭が B でなく、末尾が A でない

種類 4 が答えに影響することはないため、これは存在しないことにしても構いません。

種類 $1, 2, 3$ の個数をそれぞれ c_1, c_2, c_3 とします。

$c_1 = 0$ のとき、$\min(c_2, c_3)$ 個追加で AB を作ることができます。$c_1 \neq 0$ のとき、$c_2 + c_3 > 0$ ならば $c_1 + \min(c_2, c_3)$ 個追加で AB を作ることができます。$c_2 + c_3 = 0$ の場合は例外で $c_1 - 1$ 個追加で AB を作ることができます。

これらの処理は $O(\sum |s_i|)$ で実行可能で、十分高速です。

D: DivRem Number

問題文中の条件を冷静に観察すると $N = k(m+1)$ と表せる必要があることが分かります。即ち、N の約数から 1 引いた数に限り、お気に入りの整数になりえます。よって、N の約数を全て列挙し、実際に判定を行えばよいです。これは $O(\sqrt{N})$ 程度で実行可能であり、十分高速です。

E: XOR Partitioning

b を A の累積 XOR 和を取った数列、即ち $b_i = \oplus_{j=1}^{i} A_i$ なる数列とします。
問題文中の条件を満たす分割は以下のように言い換えられます。

> $0, 1, \ldots, N$ の番号がついた N 個のマスが左から右に向かって並んでいる。マス i には整数 b_i が書かれている。すぬけ君ははじめに好きな整数 X を任意に選んだのち、駒をマス 0 に置く。その後、以下のルールで駒を動かす。
>
> - 現在駒が置かれているマスより、番号が大きいマスにしか動かせない
> - 現在駒が置かれているマスに 0 が書かれているなら、X が書かれているマスに駒を移動させなくてはならない
> - 現在駒が置かれているマスに X が書かれているなら、0 が書かれているマスに駒を移動させなくてはならない
>
> 最終的に駒がマス N に置かれているように駒を動かす方法は何通りあるか？

簡単のため b_N が 0 でない場合を考えます。このときは X が一意に定まります。よって、$dp(i, x)$ をマス i まで処理したのち、現在駒が置かれているマスに書かれた数が x であるような部分列の選び方、とした動的計画法を行えばよいです。具体的な漸化式は以下のように表せます。

$$dp(i,0) = \begin{cases} 1 & (i = 0) \\ dp(i-1,0) + dp(i-1,X) & (b_i = 0) \\ dp(i-1,0) & (\text{otherwise}) \end{cases}$$

$$dp(i,X) = \begin{cases} 0 & (i = 0) \\ dp(i-1,0) + dp(i-1,X) & (b_i = X) \\ dp(i-1,X) & (\text{otherwise}) \end{cases}$$

以降は b_N が 0 であることを仮定します。$X = 0$ の場合は特殊ですが、容易に計算可能です。$X \neq 0$ の場合を考えましょう。

単に X を全探索して、先程の動的計画法を実施するだけではうまくいきません。これは 0 が書かれたマスはどの X においても処理する必要があるためです。0 が書かれたマスの影響をうまくまとめて計算することを考えましょう。

ここで、b を X と 0 以外の数を取り除いたのち、ランレングス圧縮を行った数列における 0 の出現回数は b における X の出現回数で抑えられることが重要です。この要領で、$b_i = 0$ なる i に関する遷移をうまく遷移をまとめると、先程の動的計画法が O(b における X の出現回数) で出来ます。全体として $O(N)$ となって、十分高速です。

F: Edge Ordering

まず、最小全域木に含まれる辺の重みの総和を求めるのではなく、良い割り当ての個数を求めることにします。

さらに、簡単のため G の最小全域木に現れる辺を重みの小さい順に並べたとき、辺 $1, 2, \ldots, N-1$ の順に現れることを仮定します。このとき、問題は以下のように言い換えられます。

> M 個の整数 $1, 2, \ldots, M$ を左から右に横一列に並べることを考えます。ただし、$2 \leq i \leq M$ なる i は x_i を並べた直後から並べることができます。ここで、$1 \leq x_i \leq N-1$ が成立し、さらに $2 \leq i \leq N-1$ なる i については $x_i = i-1$ が成立します。
>
> ありうる並べ方は何通りありますか？

これは、以下の図 1 のようなグラフをトポロジカルソートする方法の個数で表せます。

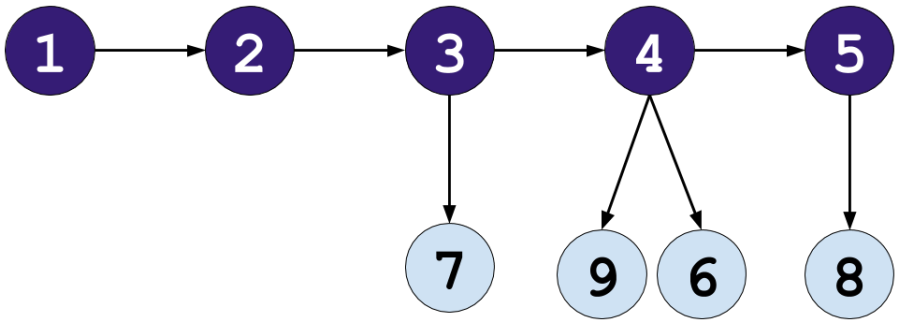

図 1　$N = 6, M = 9$ での例

$1 \leq i \leq N-1$ なる i について、頂点 i から生えている辺のうち、N 以上の頂点に向かって生えている辺の本数を a_i として、問題をさらに言い換えます。

> すぬけ君は色 $1, 2, \ldots, N-1$ のボールをそれぞれ $a_i + 1$ 個持っている。すぬけ君は持っているボールがなくなるまで、左から右に並べることにした。
>
> $2 \leq i \leq N-1$ なる i について、色 i のボールは色 $i-1$ のボールを並べた直後から並べることができる。
>
> ありうるボールの色の並びに $\prod_{i=1}^{N-1} a_i!$ をかけた値を求めよ。

b_i を色 $i, i+1, \ldots, N-1$ であるボールの個数とします。ありうるボールの色の並びは b を用いて、以下

のように表せます。

$$\prod_{i=1}^{N-1} \binom{a_i + b_{i+1}}{a_i}$$

これは色 $N-1, N-2, \ldots, 1$ の順に見て、a_i 個の色 i のボールを挿入したのち、最後に色 i のボールを先頭に置くという操作を繰り返すことと対応しています。

これにより、辺 $1, \ldots, N-1$ に割り当てられた辺の重みの順序が分かっている場合には $O(N+M)$ で解けることが分かりました。辺の重みの順序を全探索すると $O(N!)$ かかりますが、bitDP でまとめることで $O(2^N N)$ で計算可能です。なお、a_i を高速に計算するためには、森を指定していくつかの連結成分に分解したときに、同じ連結成分どうしをつなぐ辺の個数を予め計算しておく必要があります。これも高速ゼータ変換を用いて $O(2^N N)$ で計算できます。

さて、元の問題では、ありうる全ての並びについて最小全域木のコストを計算する必要がありました。まず、最小全域木のコストを求める代わりに、それぞれの辺ごとに重み w が割り当てられるような並べ方の個数を求めることを考えます。ここで、辺 i に重み w が割り当てられたとき、w 以下の重みが割り当てられた辺は w 本あるという事実に着目します。すると、ありうる全ての並びについて、以下の条件を満たす (i, j) の個数を計算すればよいことがわかります。

- j は $1, 2, \ldots, N-1$ のいずれか
- 辺 i に割り当てられた重みは辺 j に割り当てられた重み以下である

ほぼ先ほどと同様の DP により、$O(2^N N)$ で計算することが可能であり、十分高速です。

diverta 2019 Programming Contest Editorial

camypaper

May 11, 2019

A: Consecutive Integers

For each integer i between 1 and $N - K + 1$ (inclusive), there is one way to choose K integers such that the minimum of the chosen integers is i, so we should print $N - K + 1$.

Sample solution in C++: https://atcoder.jp/contests/diverta2019/submissions/5322859

B: RGB Boxes

Since N can be up to 3000, we cannot check all triples (r, g, b) such that $0 \leq r, g, b \leq N$ in time. However, when the values of r and g are fixed, b can be uniquely determined as $b = \frac{N - rR - gG}{B}$. Thus, we can check all possible pairs of r and g such that $0 \leq r, g \leq N$ and test if b will be a non-negative integer, in $O(N^2)$ time.

Solution in C++: https://atcoder.jp/contests/diverta2019/submissions/5323803

C: AB Substrings

We can calculate the number of `AB`s within each string beforehand. Let us concentrate on the change of the number of `AB`s that extend over two strings when the strings are rearranged.

The only characters that matter in each string are its first and last characters. Let us classify the strings according to those characters into four categories, as follows:

1. A string that begins with `B` and ends with `A`
2. A string that begins with `B` but does not end with `A`
3. A string that does not begin with `B` but ends with `A`
4. A string that does not begin with `B` or end with `A`

Category 4 is irrelevant to our interest and can be ignored. Let c_1, c_2 and c_3 be the number of strings of category $1, 2$ and 3, respectively.

When $c_1 = 0$, we can make additional $\min(c_2, c_3)$ `AB`s. When $c_1 \neq 0$, we can make additional $c_1 + \min(c_2, c_3)$ `AB`s if $c_2 + c_3 > 0$, and additional $c_1 - 1$ `AB`s if $c_2 + c_3 = 0$.

These computations can be done fast enough in $O(\sum |s_i|)$ time.

D: DivRem Number

When $[N/m] = Nmodm = k$, N must be $km + m = k(m + 1)$. Thus, for m to be a favorite number, it is necessary that $N = k(m + 1)$ for some integer k. In other words, a favorite number needs to be a divisor of N, minus one. Thus, we should list all divisors of N and check if each divisor minus one is a favorite number. This can be done fast enough in around $O(\sqrt{N})$ time.

E: XOR Partitioning

Let b be a sequence such that $b_i = \oplus_{j=1}^{i} A_i$.
The problem is equivalent to the following:

> There are N squares numbered $0, 1, \ldots, N$ arranged from left to right. An integer b_i is written on Square i. Snuke first choose an integer X freely and place a piece on Square 0. Then, he moves the piece according to the following rules:
>
> - A piece can only be moved to a square to the right.
> - When 0 is written on the current square occupied by the piece, the piece must be moved to a square where X is written.
> - When X is written on the current square occupied by the piece, the piece must be moved to a square where 0 is written.
>
> In how many ways can Snuke move the piece to Square N?

First, let us consider the case where b_N is not 0. In this case, the possible value of X is uniquely determined. Thus, we can solve this case by dynamic programming. Let $dp(i, x)$ be the number of the possible sequences of squares visited by the piece up to Square i such that x is written on the square currently occupied by the piece. Then, the following recurrence relation holds:

$$dp(i,0) = \begin{cases} 1 & (i = 0) \\ dp(i-1,0) + dp(i-1,X) & (b_i = 0) \\ dp(i-1,0) & (\text{otherwise}) \end{cases}$$

$$dp(i,X) = \begin{cases} 0 & (i = 0) \\ dp(i-1,0) + dp(i-1,X) & (b_i = X) \\ dp(i-1,X) & (\text{otherwise}) \end{cases}$$

From now on, we assume that b_N is 0. The special situation $X = 0$ can be easily handled, so let us consider the situation where $X \neq 0$.

Just running the above dynamic programming for all possible values of X will not be fast enough, because we need to process the squares with 0s for every X. Let us consider how we can process those squares altogether.

The important fact is that, if we remove all numbers except X and 0 from b and apply run-length encoding, the number of times 0 occurs in the resulting RLE sequence is at most the number of occurrences of X in b, plus 1. Thus, the above dynamic programming can be run in $O(\text{the number of occurrences of } X \text{ in } b)$ time by properly combining the transitions for i such that $b_i = 0$. The total time complexity is $O(N)$, which is fast enough.

F: Edge Ordering

First, let us find the number of good allocations, not the sum of the total weights of the edges.

We will simplify the problem again by assuming that sorting the edges in the minimum spanning tree results in the order Edge $1, 2, \ldots, N-1$. For example, consider the following graph:

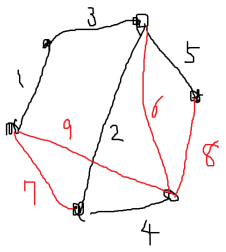

Fig1　An example of a graph where $N = 6, M = 9$

In this case, in order for the first five edges to be an MST (and their costs satisfy $cost(1) < cost(2) < \ldots$), for example, the cost of the edge 7 must be greater than the cost of edge 3. We can rephrase the condition as the topological sort of the following graph:

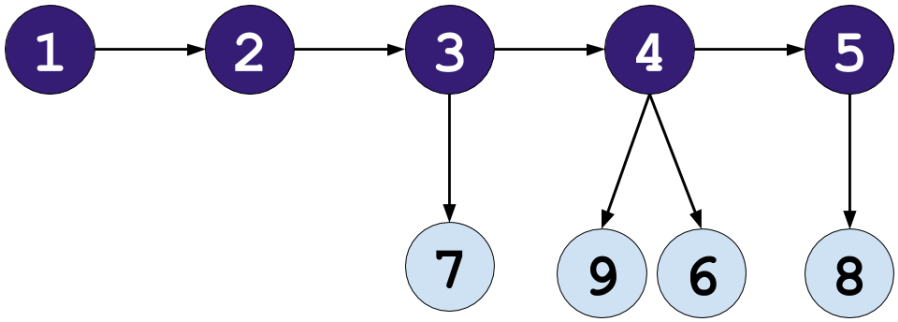

Fig2　An example of a graph where $N = 6, M = 9$

Let a_i $(1 \leq i \leq N-1)$ be the number of edges from Vertex i to a vertex indexed N or greater in the graph above. Let us rephrase the problem once again as follows:

Snuke has an empty sequence of balls. He recieves balls one by one, in the following order: a_N white balls, one black ball, a_{N-1} white balls, one black ball, $\cdots$. (In the example above, it corresponds to the order $5, 8, 4, 9, 6, 3, 7, 2, 1$).

- When he recieves a black ball, he inserts it at the beginning of the sequence.
- When he recieves a white ball, he inserts it to arbitrary place in the sequence.

Find the number of the possible sequences of balls (assuming that the balls are distinguishable even if they have the same color).

Now it's easy to count the number of topological sortings: it's simply the product of number of choices when he receive white balls.

How to count the sum of weights of MSTs in this case? For each sequence of balls we get, we compute the sum of positions of black balls, and we want to get the sum of these values over all possible sequences.

Suppose that we have c sequences of balls of length n (and b of the balls are black). Let s be the sum of (sum of positions of all black balls) over all c sequences. Now we define the state of the set of sequences as the tuple (n, b, c, s). When we recieves a black ball, the state changes to $(n+1, b+1, c, s+(b+1)c)$. When we recieves a white ball, the state changes to $(n+1, b, (n+1)c, (n+2)s)$. When we recieves k white balls one by one, the state changes to $(n+k, b, (n+1)\cdots(n+k)c, (n+2)\cdots(n+k+1)s)$. Thus, by keeping this state, we can do transitions in $O(1)$.

We have solved the problem in $O(N)$ time given the order of weights assigned to Edge $1, \ldots, N-1$ (assuming that the values a_i are pre-computed). The full problem can be solved in $O(N!)$ time if we try all possible orders of the weights of the edges, but this can be improved to $O(2^N N)$ by dynamic programming on bits. Here, in order to compute a_i efficiently, we need to calculate beforehand the number of edges connecting vertices in the same connected component when a specified forest is divided into some connected components, which can also be done in $O(2^N N)$ time by fast zeta transform.