Atcoder Regular Contest 解説

writer : maroonrk

平成 30 年 5 月 26 日

For International Readers: English editorial starts from page 7.

A : Add Sub Mul

整数の足し算、引き算、掛け算はほとんどのプログラミング言語で標準サポートされているでしょう。あとは、3 つの整数の最大値が求められれば良いです。これは if 文による条件分岐で実装出来ますが、プログラミング言語によっては、与えられたいくつかの整数の最大値を求める関数が用意されている場合もあります。

C++による解答コード

B : Cut and Count

文字列を分割する位置を全探索します。英小文字 26 種類すべてについて、分割されたあとの 2 つの文字列の両方に含まれるかの判定を行えば、「X と Y の両方に含まれる文字の種類数」が求まります。あとはその中で最大値を求めれば良いです。

26 種類の文字全てに対して、$if(s[i] == `a`)$, $if(s[i] == `b`)$ $\cdots$ と言ったふうに判定関数を書くのは骨の折れる作業ですが、ASCII コードで、$`a`$ $`b`$ $\cdots$ $`z`$ が並んでいるという性質を利用するとコードが簡単になります。

C++による解答コード

C : Attention

西から i 番目の人がリーダーになった際、向く方向を変える必要のある人は、西から $j(j < i)$ 番目に並んでいて西を向いている人と、西から $j(j > i)$ 番目に並んでいて東を向いている人です。

ここで、「ある範囲に存在する西 (東) を向いている人の人数」は、累積和を使うと、前処理 $O(N)$ 時間、1 クエリあたり $O(N)$ 時間で求められます。

よって、まず最初に $O(N)$ 時間で累積和を求めておくと、各人について、その人がリーダーになった際に向く方向を変える必要のある人の人数を求めることが $O(1)$ 時間ででき、合計 $O(N)$ 時間で全員分求まります。あとはその中で最小値を求めればよいです。

D : Xor Sum 2

整数列 $c_1,\ c_2,\ \cdots,\ c_m$ が、$c_1\ xor\ c_2\ \cdots\ xor\ c_m\ = c_1 + c_2 + \ \cdots\ + c_m$ を満たす条件を考えると、これは、すべての $k(0 \leq k)$ に対して、2 進法表記したときの 2^k の位が 1 であるよう c_i の個数が 1 以下であることと同値です。これは、$a + b - (a\ xor\ b) = 2 \times (a\ and\ b)$（ただしここで and はビットごとの論理積を表す）であることからわかります。

l を固定したときに、(l, r) が条件を満たすような最大の r を $f(l)$ とおきます。すると、f は l に対して広義単調増加です。よって、l を増加させながら尺取り法の要領で $f(l)$ を求めることができ、$O(N)$ で答えを求めることが出来ます。

この他にも、各 bit についてそのビットが立っている位置を保存することで、直接的に $f(l)$ を求める解法も存在します。

E : Range Minimum Queries

取り除いく要素の最小値 X を決めたとします。この条件のもので、乗り除く要素の最大値 Y を最小化することを考えます。

最小値が X という条件を守るためには、X より小さい要素を含むような区間に対して操作をすることは出来ません。そこで、与えられた数列を、X より小さい要素の位置で切ると、いくつかの列に分割されます。分割されたあとの列の中で完結する操作は自由に行なえます。

分割したあとの列の一つを $c_1,\ c_2,\ \cdots,\ c_m$ とします。この列からは、高々$m-K+1$ 個の要素を取り除くことが出来ます。小さい方から取り除くほうが好ましいので、この列から取り除かれる可能性があるのは、$c_1,\ c_2,\ \cdots,\ c_m$ の中で小さい方から $m-K+1$ 個の要素です。

分割したあとの各列について、取り除かれる可能性のある値を列挙すると、それらすべての中で Q 番目に小さいものが、Y の最小値になります。

列の要素の小さい方から何要素かを得るのは、列をソートすれば出来ます。よって、X を決めておくと、Y の最小値は $O(NlogN)$ で求まることになります。

X は N 通りあるので、合計 $O(N^2logN)$ でこの問題は解けます。

余談ですが、実はこの問題は $N \leq 10^5$ で解くことが出来ます。

F : Donation

各頂点 v について、$C_v = max(A_v - B_v, 0)$ と定義します。ここで、問題で行う操作を逆から見ると、次のようなゲームをしていると思うことが出来ます。

> 適当な頂点 s から、スタートする。このときの所持金は好きな値でよい。これから、以下の 2 つの操作を好きな順序で好きなだけ繰り返す。
>
> - 今いる頂点を v としたとき、B_v 円を得る。ただし、今いる頂点をでお金をもらえるのは 1 度だけである。
> - 今いる頂点を v としたとき、所持金が A_v 円以上なら、隣接する頂点から好きなものを選んでそこに移動する。
>
> すべての頂点でお金をもらい、かつ、最後に立っている頂点を t としたときに、所持金が A_t 円以上ならゲームクリアとなる。

このゲームの性質を考えると、今まで訪れたことのない頂点に移動してきた際には、すぐにお金を得るのが最善です。また、一度訪れた頂点は、その後自由に移動できます。以上の考察より、ゲームを更に次のように簡略化出来ます。

> 適当な頂点 s から、スタートする。このときの所持金は C_s 円以上でなくてはならない。これから、以下の操作を、すべての頂点を訪れるまで繰り返す。
>
> - 今まで訪れた頂点に隣接する、まだ訪れていない頂点を一つ選び、移動する。移動先の頂点を v とすると、このときの所持金は C_v 円以上でなくてはならない。そして、移動したあと B_v 円を得る。
>
> すべての頂点を訪れることができればゲームクリアとなる。

このゲームでは、次に移動可能な頂点の中で C_v が最小のものを選び、移動するという戦略が最適だとわかります。

C_v が最大の頂点を z とします。ここで、z をグラフから取り除いたあとの連結成分を P_1, P_2, $\cdots P_L$ とおきます。（ここで L は 0 かもしれません）最適解においてスタートする頂点が z だとします。その場合、ゲームクリアのための所持金の最小値は、C_z に、B_v の総和を足したものになります。最適解においてスタートする頂点が P_i 内にあったとします。その場合、ゲームクリアのための所持金の最小値は、P_i だけでゲームをクリアするための所持金の最小値を Q_i とすると、$max(Q_i, C_z) + P_i$ 以外の頂点の B_vの総和 となります。

これで多項式解法が得られましたが、愚直にグラフを分割していては間に合いません。グラフが分割される様子を逆から見ると、C_v の小さい頂点だけでできるだけ大きな連結成分を作っていると思うことが出来ます。よって、各辺のコストを、端点の C_v の max だと思って、小さいコストの辺から優先的に使っていくと考えれば、Union-Find を使って、グラフが分割される様子を表す根付き木を作ることが出来ます。この木の各頂点は、C_v がある一定の値以下の頂点のみからなるある連結成分に対応することになります。この木の各頂点に対して、対応するグラフ内だけの問題を解いたときの答え、及び B_v の総和を求めると、木 DP によって全体の答えが求まります。

C_v の昇順に頂点を並べたり、Union-Find の操作をするのがボトルネックになり、この問題は $O(NlogN)$ で解けます。

Atcoder Regular Contest Editorial

writer : maroonrk

平成 30 年 5 月 26 日

A : Add Sub Mul

C++ Example

B : Cut and Count

C++ Example

C : Attention

If the i-th (from the west) person is chosen as the leader, the j-th person have to change the direction in the following two cases:

- $j < i$ and the j-th person is facing west.
- $j > i$ and the j-th person is facing east.

First, let's pre-compute prefix sums: for each i, count the number of people among the west-most i people who are facing east. This can be done in $O(N)$ time. After that, we can answer a query of the form "How many people in a given range are facing east (west)?" in $O(1)$ time, thus the two values above can be computed in $O(1)$ after we fix the leader.

This solution works in $O(N)$ time in total.

D : Xor Sum 2

Notice that $a + b - (a\ xor\ b) = 2 \times (a\ and\ b)$. The inequality $a + b \geq a\ xor\ b$ always hold, and the equality holds if and only if $a\ and\ b = 0$. (Intuitively, the equality holds if we get no "carries" when we add a and b as binary numbers.)

Therefore, for a sequence of non-negative integers $c_1,\ c_2,\ \cdots,\ c_m$, the equation

$c_1\ xor\ c_2\ \cdots\ xor\ c_m\ = c_1 + c_2 + \ \cdots\ + c_m$

holds if and only if for all k, the number of elements among $c_1, \ldots, c_m$ that contains the k-th bit (in binary representation) is at most one.

For a fixed l, the interval $[l, r]$ satisfies the condition in the statement if no bit is repeated among $a_l, \ldots, a_r$. Thus, there exists an integer $f(l)$, and the condition is satisfied if and only if $r \leq f(l)$. Since $f(l)$ is monotonously non-decreasing, we can compute the value of $f(l)$ for all l in $O(N)$ time using two-pointers.

It is also possible to directly compute $f(l)$: for each k, list the indices of all elements that contains the k-th bit.

E : Range Minimum Queries

Let's fix the value X: the lower bound of removed integers. Under the condition that we are never allowed to remove integers less than X, let's minimize the value of Y (the upper bound of removed integers). Note that we slightly modified the definition of X: now it is not necessary to remove X.

In order to satisfy the condition, we can never choose an interval that contains an integer less than X. Thus, let's split the entire sequence by integers less than X. Now we get (possibly multiple) sequences that don't contain integers less than X, and we can freely perform operations within each of these sequences.

Let $c_1,\ c_2,\ \cdots,\ c_m$ be one of the sequences we get after the split. We can remove at most $m - K + 1$ elements from this sequence (because before the last operation there must be at least K elements). It is clearly optimal to remove integers from the smallest ones. Thus, the $m-K+1$ smallest elements in this sequence have possibility to be removed.

We do the same thing for each part of splitted sequences, and list all elements that may be removed. If there are Q or more such elements, the minimum possible value for Y is the Q-th smallest element among them.

We can do this in $O(N \log N)$ for a fixed X. If we try all N possibilities for X, this solution works in $O(N^2 \log N)$ time in total.

Note that this problem is solvable even for $N \leq 10^5$. This part is left as an exercise for readers.

F : Donation

Let's modify the problem a bit. Instead of the condition about A_v, we require the following constraint: for each vertex v, whenever you are at vertex v, your money must be at least $C_v := max(A_v - B_v, 0)$ (even after you donate money to the vertex).

It turns out that this problem is equivalent to the original problem. It's clear that a valid sequence of moves in the original problem is also a valid sequence of moves in the modified problem. On the other hand, since it never makes sense to visit a vertex again after you donate for this vertex (in this case, we can "postpone" the donation), a valid sequence of moves in the modified problem can be converted to a valid sequence of moves in the original problem.

Let v be the vertex that maximizes the value of C_v. From the observation above, we can assume that, after we donate for this vertex we never visit this vertex. Let $G_1, \ldots, G_k$ be connected components we get when we remove the vertex v from the graph. Suppose that we end the game inside G_i. Then, after we donate for v, we must move inside G_i.

We claim that (one of) optimal solutions is of the following form:

- We donate for all vertices in $G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_k$.
- Move to the vertex v, and donate for it.
- We donate for all vertices in G_i. We never go out of this subgraph during this phase.

Suppose that there is an optimal solution that is not of this form. Then, there exists a vertex w in G_i such that we donate for w before v. In this case, we can cancel the donation for w, and instead we add a donation for w right after the donation for v, and improve the solution. Therefore, we proved the claim above.

Let's construct a rooted tree as follows:

- v (the vertex that maximizes C_v) is the root of the tree.
- Recursively construct rooted trees for subgraphs $G_1, \ldots, G_k$.
- Add an edge between v and the root of the tree for G_i for each i.

We can construct this tree in $O(N \log N)$ time in the order from leaves to root by handling vertices in the increasing order of C.

Now, we can do a DP on this tree. For each vertex u, define $dp[u]$ as the minimum amount of money required to donate for all vertices in the subtree rooted at u. Then we can fill the DP table in the order from leaves to the root.

This solution works in $O(N \log N)$ time.