AGC 029 解説

yutaka1999

2018 年 12 月 15 日

A : Irreversible operation

操作として隣合ったオセロの石をともに裏返すのではなくて、それらを裏表そのまま入れ替えると考えて問題ないことに注意します。このとき、各白石の移動した距離の総和が操作回数と一致します。ここで、操作によってもともと同じ色同士であった石の相対順序は入れ替わらず、さらに操作が行えなくなった状態では $W..WB..B$ のようになっていることに注目しましょう。そうすると各白石と相対順序が入れ替わった石の数は、それよりもはじめ左にあった黒石の数と一致します。よって、これ以上操作を行えない状態になったとき、行った操作の回数はその過程によらず $\sum_{S_i=W} \#\{1 \leq j < i | S_j = B\}$ であると分かります。よって、求める最大値は $\sum_{S_i=W} \#\{1 \leq j < i | S_j = B\}$ です。これは $O(N)$ の時間計算量で求めることができます。

B : Powers of two

まず、書いてある数が最大のボールを一つ考えましょう。それを i 番目のボールとします。このとき、i 番目のボールのペアとなるボール j があるとしたら、$A_i < A_i + A_j \leq 2A_i$ より $A_i + A_j$ の値は一意に定まります。(というのも、$A_i + A_j = 2^k$ となっていたとすると、$A_i < 2^k \leq 2A_i$ であるので k は一意に定まります。) よって、$A_i + A_j$ が 2 べきとなるような j があるとしたら、i 番目のボールと j 番目のボールをペアにしてしまってよいことが分かります。これは、ペアの数が最大となる組み方を一つとってきたときに、i 番目のボールがペアとして使用されていれば、それは j 番目のボールと同じ数が書いてあるので j 番目のボールとしてみなしてよく、使用されていない場合は、j 番目のボールが属しているペアを i, j 番目のボールのペアに置き換えてもペアの数が減らないことから示せます。よって、貪欲にペアを作っていって問題ないことが分かるので、シミュレーションをすることでこの問題を解くことができます。時間計算量のボトルネックは、A_i の内最大のものを取ってくるところと $A_i + A_j$ が 2 べきとなる j を探すところで、計算量は $O(N \log N)$ になります。

C : Lexicographic constraints

二分探索を行うことを考えます。つまり、文字を X 種類まで使えるとして、$S_1, ..., S_N$ を構成することができるかどうか考えます。ここで S_0 を空文字列として、各 $1 \leq i \leq N$ に対し S_{i-1} から最適な S_i を順に求めていくことを考えます。長さが A_i で S_{i-1} より辞書順で大きい文字列のうち、辞書順最小な文字列が最適だと分かるので、それを求めればよいです。文字を辞書順に $0, 1, ..., X-1$ とすると求めたい文字列は、$A_{i-1} < A_i$ のときは $A_{i-1}+1, ..., A_i$ 文字目を 0 として S_{i-1} を長さ A_i に拡張して得られる文字列、$A_{i-1} \geq A_i$ のときは S_{i-1} の $A_i + 1, .., A_{i-1}$ 文字目を削除して A_i 文字目を 1 つ大きい文字に置き換えて得られる文字列であることが分かります。(ただし、A_i 文字目が $X-1$ の場合、繰り上がりが生じるので適切に処理する必要があります。) 長さが 10^9 まであるため、実際の文字列の状態を管理するのが難しいように見えますが、ほとんどの文字が 0 であることを考えると、map や配列を適切に用いることで比較的容易に管理することができます。以上より、順に S_i を構成していくことができるので、確かに判定関数を作ることができることが分かります。(X 種類の文字種で構成することができないのは、繰り上がりの際にどの桁も追加できなくなった時です。) よって、$O(N \log N)$ (配列のみを用いると $O(N)$) の時間計算量で二分探索の判定関数を作ることができるので、全体

として $O(N \log N \log \max A)$ の時間計算量で解くことができます。

D : Grid game

高橋君が駒を動かさなかったとき、青木君は自分も駒を動かさずにゲームを終了させるのが最適です。よって、高橋君は動かせる限り駒を動かすことが最適です。このとき、高橋君の行う行動の回数は（高橋君が駒を動かした回数）+1 であるので、高橋君が駒を何回動かせるかを求められればよいです。青木君はゲームを早く終了させたいので、高橋君が駒を動かせなくなるマスにできるだけ早く動かすのが最適です。高橋君は動ける限り必ず動くので、これは青木君一人の選択のみによって定まるゲームとなります。つまり、青木君が駒を (x, y) から $(x+1, y)$ または $(x+1, y+1)$ のいずれかに動かすことができるとき、最小で何回の操作で高橋君が駒を動かせなくなるマスにたどり着けるかを求められればよいです。

$(x, y) \to (x-y, y)$ と座標変換することで、(x, y) から $(x+1, y)$ または $(x, y+1)$ のいずれかに動かせる場合に帰着できます。このとき、高橋君が駒を動かせなくなるマスは $(x+1, y)$ に障害物があるようなマス (x, y) です。（ただし、グリッドの外部にあるマスも障害物とみなします。）よって、そのようなマスの内、障害物にぶつからずにたどり着ける一番近いマスを求められればよいです。障害物にぶつからずにたどり着けるかどうかは、y 座標を小さい順にみていって、障害物にぶつからずにたどり着ける x 座標の範囲を持っておくことで十分高速に判定することができます。よって、一番近いたどり着けるマスを求めることができるので、以上よりこの問題を解くことができます。ボトルネックとなるのは障害物をソートして y 座標の昇順に並べるところであり、$O(N \log N)$ の計算量で解くことができます。

E : Wandering TKHS

便宜上 $c_1 = 1$ とおいて、1 を根として根付きにしたときの、各頂点 $v > 1$ の親を p_v とします。また、p_v から 1 までのパス上にある頂点の番号の最大を m_v とします。（ただし $p_1 = m_1 = -1$）ここで、$c_v - c_{p_v}$ の値を各 $v > 1$ に対して考えます。

$i) v > m_{p_v}$ のとき
$c_v - c_{p_v}$ は（v の子孫（v 含まず）のうち v から m_v 以下の頂点だけを通ってたどり着ける頂点の数）+1 となります。
$ii) v < m_{p_v}$ のとき
$c_v - c_{p_v}$ は v の子孫（v 含まず）のうち v から m_v 以下の頂点だけを通ってたどり着ける頂点の数から、v の子孫（v 含ない）のうち v から m_{p_v} 以下の頂点だけを通ってたどり着ける頂点の数となります。

よって、$Q(v, x) :=$ v の子孫（v 含まず）のうち v から x 以下の頂点だけを通ってたどり着ける頂点の数 が求められればよいことになります。これはデータ構造を使うことでも解けますが、今回は呼ばれるクエリが特殊なので簡潔に解くことができます。まず $Q(v, x) = \sum_c (1 + Q(c, x))$（$c$ は v の子でかつ $c < x$）であるので、この式に従って再帰的に求めることはできます。一般の場合は間に合いませんが、今回は $Q(v, m_v), Q(v, m_{p_v})$ の形式でしか呼び出されないことに注意すると、この漸化式に従ってメモ化再帰を行っても十分高速に動くことが分かります。（各頂点 c に対して、$Q(c, x)$ の x として呼ばれる候補は 1 から c までのパス（c 含まず）の頂点の中で番号上位 3 つのものに絞られます。）よって、$c_v - c_{p_v}$ の値をすべて求めることができるので、すべての c_v を求めることができます。時間計算量は $O(N)$ となります。（メモ化再帰のメモは各頂点に対して行うことで線形になります。）

F : Construction of a tree

まず必要条件を考えます。木の特徴として、どの頂点 v を固定したとしても v を根として根付き木とみなすことで、辺と v 以外の頂点を完全にマッチングさせることができます。この特徴を用いると、任意の頂点 v に対して、v 以外の頂点と E の元の間に完全マッチングが存在することが必要だと分かります。（ただし、ここでは一方が v 以外の頂点、もう一方が辺集合であるような二部グラフであり、(u, E_i) の間には $u \in E_i$ のときに辺が存在するようなものを考えています。）実はこれが十分条件であることが示せます。ある頂点 v での完全マッチングを考え、v 以外の頂点 u に対して u に対応する辺集合を S_u とします。そこで、$a \in S_b$ のときに a から

b に辺を張ってできる N 頂点グラフを考えると、v からすべての頂点にたどり着けることが示せます。というのも、v 以外の任意の頂点 u に対し、u を除いた頂点と辺集合の間の完全マッチングを考えて、その完全マッチングと v を除いた時の完全マッチングの辺を組み合わせて得られる二部グラフを考えます。このとき、v と u のみ次数が 1 なので、v と u を結ぶパスが存在し、そのパスを考えるとそれが先のグラフでの v から u へのパスになっています。よって、v からすべての頂点にたどり着けることが示されました。よって、先のグラフで v から DFS を行って得られる DFS 木を考えると、これが求めたい木となります。以上より、必要十分性が得られました。証明の中で求めたい木の構成が得られているので、判定及び構成がともに可能であることが示されました。ボトルネックとなるのは、ある一つの頂点に対してその頂点を除いたときの完全マッチングを求めるところであり、Dinic 法を用いることにより $O(\sum |E_i|\sqrt{N})$ の時間計算量で解くことができます。

AGC 029 Editorial

yutaka1999

December 15th, 2018

A : Irreversible operation

Note that it makes no difference to regard the allowed operation as swapping two adjacent stones instead of flipping them. Then, the number of operations is the same as the sum of the distance each white stone moves. Here, let's focus on the fact that operations won't change the relative order among stones of the same color, and that the stones should line up like $W..WB..B$ in the end. This means, for each white stone, the number of black stones which switched places with it is the number of black stones on its left in the initial arrangement. Therefore, when you reach a situation where no additional operations can be made, the number of operations you made should be $\sum_{S_i=W} \#\{1 \leq j < i | S_j = B\}$, regardless of how you made them. Hence, the answer is also $\sum_{S_i=W} \#\{1 \leq j < i | S_j = B\}$. This value can be computed in $O(N)$.

B : Powers of two

First, let's think about the ball which has the largest number on it. Assume that the i-th ball has the largest number A_i. Then, if you can pair the i-th ball and the j-th ball, the value of $A_i + A_j$ should be uniquely determined because $A_i < A_i + A_j$ holds: i.e. if there exists k such that $A_i + A_j = 2^k$, k is unique because $A_i < 2^k \leq 2A_i$. Actually this suggests that if you find such j, you can immediately pair the i-th ball and the j-th ball. This can be proved by the fact that, even if the j-th ball is paired with another ball, you can remove that pair and form a pair of i-th ball and j-th ball, with the number of pair unchanged.

Therefore, greedy pairing is sufficient and you can solve this problem just simulating the paring. The time complexity bottleneck is finding the largest A_i and j such that $A_i + A_j$ is a power of 2. The overall complexity is $O(N \log N)$.

C : Lexicographic constraints

Let's consider finding the answer with binary search: is it possible to construct $S_1, ..., S_N$ if you have X characters?

For convenience, let S_0 be an empty string. Let's construct the optimal S_i in turn, based on S_{i-1} for $1 \leq i \leq N$. The optimal S_i is the lexicographically minimal string among strings of length A_i which are lexicographically greater than S_{i-1}. Denote by $0, 1, ..., X-1$ the alphabets ordered from the lexicographically smallest to the largest. Then,

- if $A_{i-1} < A_i$, the optimal S_i is obtained by extending S_{i-1} to length A_i, and assigning 0s to the $A_{i-1}+1, ..., A_i$ -th characters.
- if $A_{i-1} \geq A_i$, the optimal S_i is obtained by deleting the $A_i+1, ..., A_{i-1}$ -th characters, and replacing the A_i-th character with a character bigger by 1. (note that if the A_i-th character is $X-1$, you need to deal with carry.)

Because the length of the strings can be 10^9, you may think it is impossible to maintain and calculate those strings in such a way. However, considering most of the characters are 0, you can see that one can relatively easily maintain them using map or array. This way, S_i can be constructed in order, so the criterion function for binary search can be implemented indeed. (it is impossible to construct $S_1, ..., S_N$ with X characters if you become unable to deal with carry.)

Hence, overall you can solve this problem in $O(N \log N \log \max A)$ as each check is done in $O(N \log N)$ (or $O(N)$ with array),

D : Grid game

If Takahashi didn't make a move, it is optimal for Aoki to finish the game not moving the piece. This means it is optimal for Takahashi to make as many moves as possible without leaving the piece in the same cell. Then, the number of actions Takahashi performs is (the number of moves Takahashi makes) +1, so it suffices to calculate how many times Takahashi can move the piece.

Since Aoki would like to end the game as soon as possible, Aoki should move the piece as soon as possible to cells where Takahashi cannot make a move. Because Takahashi has no other choice but to move the piece, you only need to consider Aoki's choice. That is, you should find the minimum number of actions Aoki need to take, under the condition that Aoki is allowed to move the piece from (x, y) to $(x + 1, y)$ or $(x + 1, y + 1)$.

Transforming the coordinate by $(x, y) \to (x - y, y)$, you can reduce the problem to the case where the destinations are $(x + 1, y)$ and $(x, y + 1)$. In this case, Takahashi cannot make a move from cell (x, y) if $(x + 1, y)$ is occupied by an obstacle, (we consider the coordinates outside of the grid are also occupied by obstacles.) Therefore, you have to find among such cells the nearest cell, which is reachable without hitting obstacles. You can judge efficiently whether some cell is reachable without hitting obstacles, if you check cells in ascending order of y and keep the range of x coordinates of the cells, at which the piece can arrive without facing obstacles. Hence, you can find the nearest cell, and can solve this problem.

The time complexity bottleneck is sorting obstacles in ascending order of y, and the overall complexity is $O(N \log N)$.

E : Wandering TKHS

Let's make the tree rooted at 1. For convenience, let c_1 be 1 and p_v be the parent of node v. We also define m_v as the maximum ID of vertices on the path from p_v to $1(p_1 = m_1 = -1)$, and $Q(v, x)$ as (the number of v's offspring nodes, reachable from v by passing only nodes whose ID is less than or equal to x). Then, let's consider the value of $c_v - c_{p_v}$ for each $v > 1$.

i) The case where $v > m_{p_v}$: $c_v - c_{p_v}$ equals $Q(v, m_v) + 1$.
ii)The case where $v < m_{p_v}$: $c_v - c_{p_v}$ equals $Q(v, m_v) - Q(v, m_{p_v})$.

Therefore, calculating $Q(v, x)$ is enough to find the answer.

This can be done with data structures, but there is an easier way because this time we care only about the particular values of $Q(v, x)$. First, we can see that the values can be calculated recursively because $Q(v, x) = \sum_c (1 + Q(c, x))$(where c is v's child and $c < x$) holds. Generally this takes $O(N^2)$ even with memoization, but considering that the values we need are just $Q(v, m_v), Q(v, m_{p_v})$ of each v, one can see it works faster enough in this case. (for each vertex c, the candidates for x of $Q(c, x)$ are at most 3 vertices on the path between p_c and 1, which have a larger ID than others. Hence, you can find all the values of $c_v - c_{p_v}$ and eventually c_v. The time complexity is $O(N)$. (the linear time can be achieved with the memoization in every vertex.)

F : Construction of a tree

Consider necessary conditions first. Tree has the following characteristic: for any v, edges and vertices except v have a natural matching if we root the tree at v. This characteristic shows that as a necessary condition, for any v, there should be a perfect matching between vertices except v and $\{E_i\}$. (here we consider a bipartite graph which has vertices except v on one side, and the subsets $\{E_i\}$ on the other. The graph has edge (u, E_i) if $u \in E_i$.)

Actually, this condition is also a sufficient condition. Consider the perfect matching obtained for some v. Let S_u be the set of edges on the bipartite graph, one of whose endpoints is u. Next, we construct an extra graph with N vertices by adding edge (a, b) if $a \in S_b$. It can be proved that every vertex in the new graph is reachable from v in the following way: For any $u(\neq v)$, consider the union of two perfect machings obtained by applying the above-mentioned operation to u and v. (this should be a bipartite graph.) Then, there exists a path between v and u because only v and u are the vertices with degree 1, and this path should be also a path in the extra graph. Thus, it is shown that every vertex in the extra graph is reachable from v. This also means when we run DFS from v in the extra graph, the resultant DFS-tree is a tree demanded in this problem. Hence, we find the condition is necessary and sufficient.

As we alredy saw the way of constructing the demanded tree in the proof, it is proved that both of the decision and construction are possible in the problem. The time complexity bottleneck is finding the perfect matching for some vertex, and it can be done in $O(\sum |E_i|\sqrt{N})$ using Dinic's algorithm.