AGC 14 解説

yutaka1999

2017 年 5 月 6 日

For International Readers: English editorial starts at page 5.

A : Cookie Exchanges

操作を行える場合、各人の持っているクッキーの個数の集合を $\{A, B, C\}$ とすると、操作後の集合は $\{\frac{A+B}{2}, \frac{B+C}{2}, \frac{C+A}{2}\}$ になる。よって、$A \leq B \leq C$ とすると持っているクッキーの個数の最大と最小の差が操作前は $C - A$、操作後は $\frac{C-A}{2}$ となり、操作を行うと、最大と最小の差がちょうど $\frac{1}{2}$ になると分かる。よって、$A = B = C$ ではない場合は、$M = 10^9$ として、操作を高々 $\log_2 M$ 回しか行わないことになり、愚直にシュミレーションすることで解くことができる。逆に、$A = B = C$ の場合は、どれも偶数のときは無限に操作を行うことができ、どれも奇数の時は一回も操作を行うことができないので、この場合も簡単に解くことができる。以上より、$O(\log M)$ でこの問題を解くことができた。

B : Unplanned Queries

どの頂点においてもクエリの a_i, b_i として現れる回数が偶数回であることが必要十分であることを示す。つまり、全ての木に対して、その条件を満たすとき必ずすべての辺の数が偶数になり、そうでないときある辺の数が奇数になることを示す。

適当な頂点を根として、根付き木にする。この根を r とする。まず、クエリ (a, b) を考えたとき、これを $(r, a), (r, b)$ と分解することができる。これは、(a, b) の LCA を p としたとき、クエリ (a, b) ではパス $a - p, b - p$ に $+1$ しており、クエリ $(r, a), (r, b)$ ではパス $a - p, b - p$ に $+1$、パス $r - p$ に $+2$ するため、mod 2 で考えると同一視できるので明らかである。

よって、クエリ全体は (r, v) を $f(v)$ 回行う、という形に変形でき、mod 2 で考えているため、$f(v) = 0, 1$ としてよく、今示したいことは $f(v) = 0$ が題意が成立するための必要十分条件であることである。これは $f(v) = 1$ なる v で深さが一番深いものを考えると、v の直接の親を p としたとき、$v - p$ に書かれた数が奇数になるため、明らかに成立する。

よって、クエリの a_i, b_i として現れる回数が偶数回であるかどうかを判定すればこの問題を解くことができ、$O(N + M)$ で解くことができる。

C : Closed Rooms

まず、高橋君が今の場所から K 回以内でたどり着ける場所を列挙する。そして、その各場所から以下の操作を行って何回で目標地点にたどり着けられるか求められれば良い。

- まず、閉じられている部屋を K 個まで選び、それらを開いた状態にする。

- その後、K 回まで移動する。

ここで、この操作は以下の操作と同値になることを示す。

- 閉じられている閉じられていないに関わらず、隣接するマスに移動することを K 回まで繰り返す。

先の操作で移動できるならばこの操作で移動できることは明らかなので、その逆を示せばよい。これは、後者の操作で移動するときに通る、閉じられている部屋を開くことで、先の操作でも移動できるから明らか。

よって、後者の操作を行うとしてよく、これは BFS をしたり、1 行目、1 列目、H 行目、W 列目までの距離を計算することで解くことができる。

以上より、$O(HW)$ でこの問題を解くことができる。

D : Black and White Tree

与えられた木に完全マッチングが存在することと後手勝ちであることが同値であることを示す。帰納法でこれを示す。頂点数が $1, 2$ の木の場合は明らかなので、頂点数が $n-2$ のときに成立するならば頂点数 n の時も成立することを示せればよい。

完全マッチングが存在するならば、後手は先手が前に塗った頂点に対応する頂点を塗ることで、どの白の頂点も黒の頂点に隣接するようにできるため、明らかに後手勝ちである。よって、完全マッチングが存在しないときに先手勝ちを示せればよい。

まず、木を適当な頂点を根として根付き木にする。このとき、深さ最大の葉の親は子がすべて葉なので、特に子がすべて葉であるような頂点 v が存在する。ここで、v の子が 2 つ以上あるとき、先手は v を白に塗り、次の手番で先手は v の子の内まだ塗られていない頂点を白に塗れば、最終的にその葉は黒に隣接しないので、先手勝ちとなる。よって、v の子が 1 つしかない場合を考えればよい。その子を l とし、今の木から v, l を取り除いて得られる木を T とする。

このとき、T に完全マッチングが存在するならば、そのマッチングに $v-l$ の組を加えることで、今の木に完全マッチングが存在することになり、これは矛盾。よって、T には完全マッチングは存在せず、特に、T だけでゲームを行った場合、帰納法の仮定より先手勝ちである。

よって、まず先手は v を白に塗り、後手が l を黒に塗らなかったら次の手番で l を白に塗り、後手が l を黒に塗ったら、残りは T だけのゲームと同一視できるため、先手勝ちとなるから、結局このゲームは先手勝ちとなる。

以上より、頂点数が n の時も、与えられた木に完全マッチングが存在することと後手勝ちであることが同値であることが示されたので、帰納法よりこれは成立。

よって、後は与えられた木に完全マッチングがあるかどうかの判定ができればよい。これは先と同様に適当な頂点で根付き木にし、葉から貪欲にマッチングを組んでいくことで判定できる。よって、この問題は $O(N)$ で解くことができる。

E : Blue and Red Tree

まず、青から赤へ木を作り替えられると仮定して、青い辺を取り除かれる順に $b_1, b_2, ..., b_{N-1}$、赤い辺を追加される順に $r_1, r_2, ..., r_{N-1}$ とする。

このとき、任意の $i \in [1, N-1]$ に対して $b_i, ..., b_{N-1}$ からなる森の連結成分集合を $f(i)$ とすると、操作の性質より、r_i, b_i が繋いでいる $f(i+1)$ の 2 つの連結成分は一致する。

よって、$f(N) = \{\{1\}, \{2\}, ..., \{N\}\}$ であることを考慮すると、青から赤へ木を作り替えられるならば、はじめ連結成分集合が $\{\{1\}, \{2\}, ..., \{N\}\}$ という状態から、

異なる 2 連結成分を繋ぐ青い辺と赤い辺を選んで、その 2 つを 1 つの連結成分にまとめる

という操作を繰り返して、連結成分を 1 つにまとめることができる。

逆に、この操作を行うことができるとき、青から赤へ木を作り替えることができることを示す。先の操作で明

らかに同じ辺を 2 回以上使うことがないので、青い辺と赤い辺を使う順に $b_1, b_2, ..., b_{N-1}, r_1, r_2, ..., r_{N-1}$ とすると、$(b_{N-1}, r_{N-1}), (b_{N-2}, r_{N-2}), ..., (b_1, r_1)$ の順で元の問題の操作を行えば、明らかに青から赤へ木を作り替えられる。

よって、異なる 2 連結成分を繋ぐ青い辺と赤い辺を選んで、その 2 つを 1 つの連結成分にまとめる、という操作を繰り返して、連結成分を 1 つにまとめられるかどうかを判定できればよい。言い換えると、同じ 2 頂点間を結んでいる青い辺と赤い辺を選んで、その 2 頂点を縮約することを繰り返して、1 つの頂点にまとめられるかどうかを判定できればよい。同じ 2 頂点間を結んでいるような青い辺と赤い辺があれば貪欲に縮約してよいので、縮約を高速に行い、そのような辺の組を高速に発見できればよい。これは、縮約時に残っている辺の数が少ない方をマージし、マージした際に青い辺と赤い辺が同じ頂点を指していれば、その辺を消して縮約する 2 点として追加することで、全体として $O(N \log^2 N)$ で解くことができる。

よって、この問題を $O(N \log^2 N)$ で解くことができた。これは十分高速である。

F : Strange Sorting

まず、順列の内 i から N の値の部分の動きには 1 から $i-1$ の値の部分が影響しないことに注意し、i から N の部分だけ見たときに、これらがソートされる回数を $T(i)$ で表すことにする。まず、$T(i) = T(i+1)$ or $T(i+1)+1$ であることを示す。

これは、$T(i+1)$ 回目の操作後にはじめて $i+1$ から N の値の部分がソートされ、その時に i が $i+1$ より左にあればその時点で i から N の部分がソートされており、そうでないならもう一回操作を行うことで、i から N の部分がソートされるので、確かに成立する。よって、$T(i)$ が $T(i+1)$ と $T(i+1)+1$ のどちらになるか高速に判定できればよい。

ここで、$T(i+1) = 0$ の時は i が元の順列で $i+1$ より左にあるかどうかだけで判定できるのでよい。よって、$T(i+1) > 0$ の時のみ考えればよい。ここで、$T(i) > 0$ のとき、$T(i)$ 回目の操作を行う時点で i から N の中で先頭にある数を $f(i)$ で表すことにする。このとき、$T(i)$ 回目の操作ではじめて i から N がソートされることより、$f(i) \neq i$ である。ここで、$T(i)$ が $T(i+1)$ と $T(i+1)+1$ のどちらであるかは、元の順列で $i, i+1, f(i+1)$ がどの順番で並んでいるかだけに依存することを示す。

まず、$T(i+1)$ 回目の操作を行う時点での $i, i+1, f(i+1)$ の順番について考えると、この時点で $f(i+1), i, i+1$ の順に並んで入れば $T(i) = T(i+1), f(i) = f(i+1)$ であり、$f(i+1), i+1, i$ もしくは $i, f(i+1), i+1$ の順で並んでいれば $T(i) = T(i+1)+1, f(i) = i+1$ である。(ここで、$f(i+1), i+1$ はこの順に並ぶことに注意する。) よって、$T(i+1)$ 回目の操作時点でのこの 3 つの値の相対順序が元の順列でのこれらの相対順序と一致することを示せれば、判定を $O(1)$ で行うことができ、$f(i)$ についても求めることができる。以下ではこの事実を示す。(ただし、ここでいう相対順序では $(a, b, c), (b, c, a), (c, a, b)$ という 3 つの順序を同一視し、つまり、順列を円環状に並べたときの相対順序を指していることに注意する。)

特に、1 回目から $T(i+1)-1$ 回目までの操作のいずれにおいても、これらの相対順序が変化しないことを示せば十分。そのためにまず、$f(i+1)$ はこれらの操作において、$i+1$ から N だけを見たときに、先頭以外で高い項とならないことを示す。

$i+1$ から N だけを見たときに先頭以外で高い項になったとすると、その範囲だけみたとき、その操作後に $f(i+1)$ の一項前が $f(i+1)$ より小さくなる。このとき、$f(i+1)$ が再び $i+1$ から N だけを見たときに先頭に来ることがないことが示せる。なぜならば、$f(i+1)$ が先頭に来る前に必ず $f(i+1)$ の一項前の数が高い項として選ばれるが、このときその数が $f(i+1)$ よりも小さいため、$f(i+1)$ も高い項として選ばれ、移動してしまう。よって、ひとたび $f(i+1)$ が先頭以外で高い項となれば、次も必ず先頭以外で高い項となってしまうため、先頭として現れることはなくなる。しかし、$f(i+1)$ の定義より、$T(i+1)$ 回目の操作で $i+1$ から N だけを見たときに先頭となるため、これは矛盾。よって、確かに 1 回目から $T(i+1)-1$ 回目までの操作において、$f(i+1)$ は先頭以外で高い項にはならない。

よって、これらの操作の内一つを固定して、その操作での i から N の部分での先頭を k とすると、$k = f(i+1)$ のときは $k > i, i+1$ より明らかに $i, i+1, f(i+1)$ の相対順序は変化せず、$k \neq i, f(i+1)$ の時も、先の性

質より、$i, i+1, f(i+1)$ の相対順序は変化しない。よって、$k=i$ の時だけ考えればよいが、このとき i の次の数を l とすると、$l=i+1, f(i+1)$ のときは明らかに変化せず、$l \neq i+1, f(i+1)$ の時は、$l > i+1$ 及び先の性質より、この場合も相対順序は変化しない。

よって、いずれの場合も変化しないことが示されたので、$T(i+1)$ 回目の操作時点での $i, i+1, f(i+1)$ の相対順序と元の順列での相対順序は変化しない。よって、$T(i), f(i)$ を $T(i+1), f(i+1)$ を用いて $O(1)$ で求めることができるので、特に、$T(1)$ を求めることができ、これが求める答えとなる。

よって、この問題は $O(N)$ で解くことができる。

Atcoder Grand Contest 014 Editorial

writer : yutaka1999

2017 年 5 月 6 日

A : Cookie Exchanges

We prove that the operation stops in at most $O(\log M)$ steps (where $M = \max A, B, C$), except for the special case where $A = B = C$.

Suppose that the three people have A, B, C cookies, respectively. Without loss of generality, we can assume that $A \leq B \leq C$, and the difference between the maximum and the minimum is $C - A$. If an operation can be performed, after an operation, the number of cookies will be $\{\frac{A+B}{2}, \frac{B+C}{2}, \frac{C+A}{2}\}$, respectively. Here, the difference between the maximum and the minimum is $(C - A)/2$. Thus, the operation finishes in $O(\log M)$ unless $A = B = C$.

In case $A = B = C$, the answer is -1 if they are even, and 0 if they are odd. Otherwise, brute force works as described above. This solution works in $O(\log M)$.

B : Unplanned Queries

The answer is "YES" if and only if each vertex appears even number of times in the queries.

In fact, the shape of the tree doesn't matter. Choose an arbitrary vertex as a root, and call it r. If we only consider the parities, the following two things are equivalent:

- Add 1 to the path between a and b.
- Add 1 to the path between r and a, and then add 1 to the path between r and b.

Let p be the loweast common anscestor of a and b. Then, the second operation adds 1 to the path between a and b, and 2 to the path between r and p, so they are equivalent in modulo 2.

Let $f(v)$ be the number of appearances of vertex v in the queries. As we see above, the queries are equivalent to adding $f(v)$ to the path between r and v, for each v.

If $f(v)$ is even for all v, obviously any tree satisfies the condition. If $f(v)$ is odd for some v, consider one of the deepest such vertex. In this case, the edge that connects v and its parent will be labeled with an odd number, so the condition won't be satisfied.

Therefore, we only need to check whether each vertex appears even number of times, and the problem can be solved in $O(N + M)$.

C : Closed Rooms

In the statement, the following operations will be repeated X times, for some X:

- Move to an adjacent (unlocked) room at most K times.
- Then, select and unlock at most K locked rooms.

This is equivalent to the following:

- Move to an adjacent (unlocked) room at most K times.
- Then, repeat the following $X-1$ times: "select and unlock at most K locked rooms, and move to an adjacent (unlocked) room at most K times".

Note that the final unlocking of at most K rooms is irrelevant for reaching the goal.
Furthermore, this is equivalent to the following:

- Move to an adjacent (unlocked) room at most K times.
- Then, move to an adjacent room, this time not necessarily unlocked, at most $(X-1)K$ times.

Now the problem becomes easy. First, compute the set of rooms that can be reached within the distance of K from the initial position, using BFS. From each such room, compute the minimum number of operations of the following type required to reach one of the rooms with an exit:

- Move to an adjacent room, this time not necessarily unlocked, at most K times.

This can be done by a simple division. From (x, y), the minimum number of such operations is $ceil(\min\{x-1, H-x, y-1, W-y\}/X)$.
This solution works in $O(HW)$.

D : Black and White Tree

We prove that the second player wins if and only if there exists a perfect matching.

When a perfect matching exists, let $(a_1, b_1), \ldots, (a_k, b_k)$ be the edges of perfect matching. The second player uses the following strategy: when the first player paints a_i for some i, he paints b_i, and when the first player paints b_i for some i, he paints a_i. This way, exactly one of (a_i, b_i) will be painted black. Thus, after all vertices are colored, each white vertex is adjacent to at least one black vertex, so the second player wins.

Otherwise, the first player follows the following strategy. Consider the tree as a rooted tree. At each his turn, he chooses one of the deepest uncolored vertices, and call it x. Unless x is the only uncolored vertex, x is adjacent to exactly one uncolored vertex. Let's call it y. Then the first player's strategy is to color y. Then, in the next turn, the second player is forced to color x (otherwise the first player colors x in the next turn and he wins).

There are two cases:

- If y has more than one (ucolored) children, the first player wins. For example, suppose that y has two children x and z. Since x is one of the deepest node, both x and z are leaves, and the second player is forced to color them in the next turn. However, he can color at most one of them, and the second player loses.
- If x is the only child of y, after the first player colors y and the second player colors x, we can completely forget about these two vertices and continue the game as if these two vertices don't exist.

By repeating this process, we will either construct a perfect matching or the first player wins. Thus, when a perfect matching doesn't exist, the first player wins.

It's easy to check whether a tree has a perfect matching. We just repeat choosing edges greedily from leaves. This solution works in $O(N)$.

E : Blue and Red Tree

Consider a valid sequence of operations that converts the blue tree to the red tree. Suppose that we cut an edge e in the first operation, and let X and Y be the two connected components of blue edges we get after the removal of edge e. (X and Y are sets of vertices). Then we add a red edge. Let's call it f. Obviously, f must connect one vertex from X and one vertex from Y. After that, whenever we remove a blue edge in X, we must add a red edge that connects two vertices in X, and whenever we remove a blue edge in Y, we must add a red edge that connects two vertices in Y. Thus, only one red edge (that is, f) connects between X and Y.

By similar observation, we get the following. Let $b_1, b_2, ..., b_{N-1}$ be the blue edges in the order of removal, and let $r_1, r_2, ..., r_{N-1}$ be the red edges in the order of addition. Let B_i be the set of connected components obtained by the blue edges $b_i, ..., b_{N-1}$, and define R_i similarly for red edges. Then, for each i, B_i and R_i are the same.

Therefore, when there exists a valid sequence of operations, we can do the following:

- Initially, there is a graph with N vertices and no edges.
- Then, repeat this $N-1$ times: choose a blue edge and a red edge that connect the same pair of connected components, and connect them.

On the other hand, in case we can do this, the answer is "YES". Let $p_1, p_2, \cdots, p_{N-1}$ and $q_1, q_2, \cdots, q_{N-1}$ be blue and red edges used in this process (in the order they are used), respectively. Then, in the original problem, we can perform the opertaions in the order $(p_{N-1}, q_{N-1}), (p_{N-2}, q_{N-2}), \cdots, (p_1, q_1)$.

In summary, we want to the following. We are given a graph with N vertices and $2(N-1)$ edges (we don't need to distinguish blue and red edges). Whenever we find a pair of edges that connects the same pair of vertices, contract those two vertices. Can we contract the entire graph into a single vertex?

Here is one possible implementation. We keep the following information:

- The queue of all pairs of vertices that have at least two edges between them.
- For each pair of vertices, the number of edges between them (for example, map in C++).
- For each vertex, the set of all edges incident to it (for example, multiset in C++).

While the queue is non-empty, pop one element, and call it (x, y). Suppose that $deg(x) \geq deg(y)$. Then, we remove the two edges between x and y, and then for each edge that connects y and some other vertex z, delete it and add an edge connecting x and z. Of course, the information we mentioned above should be kept properly. We repeat this, and if we can perform $N-1$ contractions before the queue becomes empty, the operation is successful.

It is well-known that this way each edge is moved at most $O(\log N)$ times, so this solution works in $O(N \log^2 N)$.

F : Strange Sorting

First, notice that the existence of the element with value 1 doesn't affect the type (high or low) of the other elements. Even if we remove 1 from the original sequence and perform the operations, the behavior is very similar.

Let T be the number of operations required to sort the sequence, when we ignore 1. What happens if we perform T operations on the original sequence (including 1)? The integers $2, 3, \ldots, N$ will appear in the sequence in this order, and 1 will be inserted somewhere. If the integer 1 becomes the first element after T operations, the answer is T. In other cases, we need one more operation to sort the original sequence and the answer is $T + 1$.

How can we check if the answer is T or $T + 1$? The case with $T = 0$ is very easy. Assume that $T > 0$, and consider the state of the sequence after $T - 1$ operations. Let f be the integer that appears first in the sequence among the integers $2, 3, \ldots, N$, after $T - 1$ operations. By the definition of T, we can prove that $f > 2$. (Otherwise the integers $2, 3, \ldots, N$ are sorted in $T - 1$ operations or not sorted in T operations). We can see that after $T - 1$ operations, if 1 appears between f and 2, the answer is T, and otherwise the answer is $T + 1$.

Now, we claim that the "cyclic order" of the integers $1, 2, f$ never changes in the first $T - 1$ operations. Here, in "cyclic order", we assume that the orders $(a, b, c), (b, c, a), (c, a, b)$ are identical. Thus, we can compute the answer by computing the values of T and f, and checking the cyclic order of the integers $1, 2, f$ in the initial sequence. In order to compute T, we can repeat the similar process recursively. The detail will be described later.

Let's prove the claim. First, we prove that f never become high element in the first $T - 1$ operations unless it comes first among the integers $2, 3, \ldots, N$. Suppose that at some point, an integer x is not the first element and x is a high element. Since the first element is always a high element, after one operation x comes directly to the right of another integer y, such that $y < x$. Then, we repeat more operations on this sequence. While x and y are of the same type, these two integers always move together, and x is always directly to the right of y. They can be separated only when x becomes high and y becomes low. However, in this case, x is again a high element at not the first position, so these process will be repeated. Therefore, x never comes first. In particular, when f is not the first among the integers $2, 3, \ldots, N$, f is always a low element.

Now, let's return to the proof. Consider the sequence at some point. Let's verify that the cyclic order of $1, 2, f$ doesn't change in the next opertaion.

- If 1 is the first element,
  - If 2 is the second element, 1 and 2 are high and f is low. The cyclic order doesn't change.
  - If f is the second element, 1 and f are high and 2 is low. The cyclic order doesn't change.
  - Otherwise, 1 is high and 2 and f are low. The cyclic order doesn't change.
- If 2 is the first element, 2 is high and 1 and f are low. The cyclic order doesn't change.
- If f is the first element, f is high and 1 and 2 are low. The cyclic order doesn't change.
- Otherwise, all of $1, 2, f$ are low. The cyclic order doesn't change.

Here is the summary of the solution. Let T_i be the number of operations required to sort the sequence, when we only consider the integers $i, i+1, \cdots, N$. Let f_i be the integer that comes the first after $T_i - 1$ operations, when we only consider the integers $i, i+1, \cdots, N$. (In case $T_i = 0$, f_i is undefined).

First, we compute q_i, the position of i in the initial sequence (i.e., $p_{q_i} = i$). Then, we compute the values T_i, f_i in the order $i = N, N-1, \cdots, 1$, and the answer is T_1. When $i < N$, the values can be computed as follows:

- If $T_{i+1} = 0$,
  - If $q_i > q_{i+1}$, $T_i = 1$ and $f_i = i + 1$.
  - Otherwise, $T_i = 0$ and f_i is undefined.
- Otherwise,
  - If $q_{f_{i+1}}, q_i, q_{i+1}$ are in this cyclic order, $T_i = T_{i+1}$ and $f_i = f_{i+1}$.
  - Otherwise, $T_i = T_{i+1} + 1$ and $f_i = i + 1$.

This way, the problem can be solved in $O(N)$ time.