AGC 033 解説

yutaka1999

2019 年 05 月 05 日

For International Readers: English editorial starts on page 9.

A : Darker and Darker

一回操作を行ったときに、どのようなマスが黒色になるのかを考えてみましょう。問題文に書かれている操作を言い換えているだけですが、次のように考えると分かりやすいかもしれません。

- 各々の黒マスに対して、隣接するマスをすべて黒色にする。

このように言い換えることで、高速にシミュレーションを行うことができます。元の操作でシミュレーションを行おうとすると、毎回すべての白マスを見る必要があります。しかしこのように言い換えると、見る必要があるマスは黒マスに変わり、しかもある操作でその黒マスの周囲を黒色に変えたならば、それ以降の操作ではその黒マスを見る必要がありません。つまり、各マスを操作で見る回数というのは高々 1 回になります。

実装する際には、幅優先探索などと同様に queue を用いるのがよいでしょう。そもそも先の操作自体が、グリッドグラフ上で黒マスから幅優先探索をしているとみなすことができるので、これは自然な発想でしょう。計算量としては $O(HW)$、つまりマス目の大きさに関して線形な時間計算量で解くことができます。

B : LRUD Game

まず、縦と横が独立であることに注目します。つまり、一次元の問題を解けばよいことが分かります。問題を書きなおしてみると次のようになります。

- 長さ L の棒上に駒が置いてある。
- 先手と後手は N 回交互に駒を動かす。各ターンごとに以下のいずれかを行うことができる。
  - 動かす場合は左のみに動かせる。
  - 動かす場合は右のみに動かせる。
  - この回は動かせない。
- 先手は棒から落としたいが、後手は棒から駒を落としたくないです。
- どちらが勝つでしょう。

このようにすると考えやすくなるでしょう。あとは i 回目にどの位置に駒が置いてあれば後手が勝てるかを i が大きい方から計算していけばよいです。

実際は縦と横が独立であることに気づかなくとも、後手が勝てる駒の位置というのが長方形になるので、同様に後ろから計算していけばよいです。計算量は線形です。

C : Removing Coins

コインが残っている頂点がなす部分木を考えてみましょう。そうすると、操作を行うごとにその部分木は以下のようにして変化します。

- 選んだ頂点を根として木を根付き木にする。
- 葉の頂点をすべて消す。1 頂点しかない場合は根を消す。

ここで、この木の直径に注目してみましょう。操作を行ったあと、直径が 1 または 2 しか減少しないのは明らかです。ここで、直径の端点を選んで操作をすることで、直径はちょうど 1 減らすこともできます。さらに、直径が 2 以上の場合は、葉でない頂点が存在するので、その頂点を選んで操作をすることで、直径をちょうど 2 減らすこともできます。よって、これは以下のようなゲームと見なすことができます。

- 直径 L が与えられる。
- 先手と後手はそれぞれ L を負にならないように 1 または 2 減らすことができる。
- $L = 0$ とした人が負けのとき、どちらが勝つか求めよ。

これにより、よくあるゲームに帰着できました。L を 3 で割ったあまりが 1 のとき後手が勝ち、それ以外の場合は先手の勝ちとなります。木の直径は $O(N)$ で求めることができるので、この問題は $O(N)$ で解くことができます。

D : Complexity

1×1 のマス目は複雑さ 0 であることに注目しましょう。すると、だいたい $\log_2 H + \log_2 W$ 回ぐらいの分割で複雑さ 0 のマス目に分割できるので、マス目の複雑さは $O(\log H + \log W)$ であることが分かります。ここで、以下のようなデータを計算することを考えます。

- 複雑さ i の長方形のうち、左上のマスが (r, c)、左下のマスが (r', c) のものの横の長さの最大
- 複雑さ i の長方形のうち、左上のマスが (r, c)、右上のマスが (r, c') のものの縦の長さの最大

これはデータ量としては $O(HW(H + W) \log HW)$ なので、すべて $O(1)$ で計算できれば時間制限には間に合います。i が小さいほうから計算していくと、実際に $O(1)$ でこれは計算することができます。

E : Go around a circle

まず各円弧が赤と青の 2 色で塗られているとして、その塗り分けが条件を満たすかどうか判定する方法を考えましょう。

□ S が一色のみからなる場合

S が R のみからなるとします。このとき、円周上に青色の弧が連続していると明らかに条件を満たしません。逆に、すべての点が R に隣接していれば十分です。よって、この場合は青色の弧が連続しないことが必要十分条件になります。

□ S が R と B をともに含む場合

S の最初の文字が R だとします。先同様、円周上に青色の弧が連続していると明らかに条件を満たさないので、青色の弧が連続しないことが必要です。また、S に B が含まれるので、青色の弧は 1 つ以上なければいけません。しかしこの場合はそれでは十分ではありません。ある偶数個の連続する円弧が赤色で、かつその隣の円弧は青色であるとしましょう。S の先頭 k 文字が R でその次の文字が B だとします。このとき、その偶数長の円弧に含まれるどの点に最初駒を置いたとしても、k 回赤色の円弧を移動したのちに、青色の円弧で移動できる点、つまり円弧の端の点にいなければなりません。しかし、k が奇数ならば円弧の端の点からはじめ、k が偶数ならばその隣の点からはじめると、そのように駒を移動させることができないことが分かります。このことから、赤色の円弧が連続する個数も奇数でないといけないです。また、同様のことを考えると、赤色の円弧が連続する個数に上限があると分かります。簡単のため、S の最後の文字は B であるとします。(末尾に続く R は無視してよいです。) 先頭 k 文字が R だとしたとき, 赤色の円弧は、k が偶数のとき $k+1$ 個まで、k が奇数のとき k 個までしか連続することができないことが分かります。また、その他で R が k' 文字連続するときは、直前に B で移動していることから、駒がある位置は B に隣接する場所のみだと分かるので、k' が偶数のときは常に移動可能で、k' が奇数のときはどの赤色の円弧も k' までしか連続しないことが必要になります。以上をまとめると、以下が必要です。これが必要十分であることは簡単に確認できます。

- 青色は存在し、かつ連続しない
- 赤色は奇数個連続している
- 赤色が連続する個数は高々 L 個 (L は S から定まるある整数)

以上により条件の言い換えができました。では数え上げるためにはどうすればよいでしょう。S が一色の場合も明らかではないですが、R と B をともに含む場合と同様に解くことができるので、ここでは R と B をともに含む場合のみを扱います。

まず、いずれの色も連続して奇数個でしか現れないので、円弧は偶数個でないといけな

いです。また、青色の円弧の番号の偶奇は等しい必要があります。その偶奇を固定したときに条件を満たす個数は同じであるので、以降青色は偶数番目の円弧にのみ塗るとして構いません。すると奇数番目の円弧はすべて赤色で塗る必要があるので、考えなくてもよさそうです。実際、以下の問題を解けばよいことが分かります。

- $\frac{N}{2}$ 個の円周を青と赤で塗る。
- 青を一か所以上塗る必要がある。
- 赤は連続して L 個までしか塗ることはできない。
- 以上の条件を満たすものを数える。

いきなり円周上で考えるのは難しいので、直線の場合を考えてみましょう。つまり以下の問題を考えてみます。

- X 等分されている棒のそれぞれの部分を赤と青で塗る。
- 両端は青で塗る必要がある。
- 赤は連続して L 個までしか塗ることはできない。
- 以上の条件を満たすものを数える。

これは累積和を持ちながら dp を行うことで $O(X)$ で解くことができます。$X = i$ のときの答えを $dp[i]$ とすると、$dp[i] = \sum_{j=max(1,i-L-1)}^{i-1} dp[j]$ であるので、$rdp[i] = \sum_{j=1}^{i} dp[j]$ とすると $dp[i] = rdp[i-1] - rdp[\max(0, i-L-2)]$ となります。よって、$dp[i]$ と $rdp[i]$ を $i = 1, \ldots, X$ の順に更新していくことで、$O(X)$ の計算量でこの問題が解けます。

では元の問題を考えてみましょう。実は先の $dp[i]$ を用いることで簡単に答えは求まります。円周上の適当な位置で円周を切ってみます。すると数えるべき塗り分け方は、青を一か所以上塗り、どの赤も L 個までしか連続せず、さらに両端に連続する赤の個数の合計が L 個以下のものです。これは、一番左の一番右の青の間の距離を定めれば、後は適当な係数をかけるだけで求まります。さらに、その距離を j とすると、一番左の青と一番右の青の間を塗る方法は $dp[j+2]$ 通りです。よって、この問題が線形時間で解けました。

F : Adding Edges

G に辺を加えていくのは難しいので、辺を短くしていくことを考えます。つまり、G に対して以下の操作を行うことを考えます。

- G に辺 ab,ac があり、a, b, c がこの順にあるパス上にあるとき、辺 ac を消して辺 bc を加える。

この操作を行ったあとに G に対して辺を加えていっても、最終的に生成できる辺の集合は変わりません。そこで、G に対してこの操作を繰り返し行い、これ以上操作ができなくなった状態というのを考えることにします。このとき、そのような G に対して元の操作を繰り返して生成できる辺というのは、以下のようにかけることが示せます。

- 辺 xy が生成できることは、ある頂点の列 $x = a_0, a_1, \ldots, a_k = y$ があって、$a_i a_{i+1}$ が G に含まれており、かつ a_i, a_{i+1}, a_{i+2} がこの順にあるパス上にあることと同値である。

Proof. 上の条件が十分であることは明らかです。ここでは必要であることを示します。証明するためには、先の条件を満たすような 2 つの辺 ab, ac があったときに、b, a, c があるパス上にあるならば、それらから生成される辺 bc も先の条件を満たすことを示せばよいです。b, a, c がこの順にパス上にあるときはよいです。そうでない場合は、辺 ab, ac に対して条件を満たす頂点の列をとると、G に対して辺を短くする操作がこれ以上行えないことから、ある辺 ab, ac 上の a と異なる頂点 x があって、辺 xb, xc が条件を満たして生成される辺であることが分かります。これを繰り返し行うことで、x が b または c と一致するので、辺 bc がこの命題の条件を満たして生成されることが示されます。 □

以上より、G の辺をこれ以上短くできない場合、BFS 等を用いることにより $O(NM)$ でこの問題を解くことができます。ではどうすれば G の辺を短くしていけるかを考えてみましょう。一つの方法は、各頂点ごとにその頂点を根としたときの euler-tour を計算しておき、set などを用いて短くなる辺を効率よく探す方法です。しかし、これは計算量に log がつきますし、実装も少し大変です。ここでは、$O(NM)$ で辺を短くしていく方法を紹介します。

辺を一つずつ追加していき、徐々に短くしていくことを考えます。各頂点ごとにその点を端点に持つ辺は、もう一方の端点が○○の頂点ならば短くできる、という情報を持っておきます。辺を追加するとき、一方の端点について先の情報から短くできるならば、その辺を短くして、もう一回やり直します。どちらの端点を見ても短くできない場合は、それ

ぞれの端点に対して、辺を短くできる頂点の集合というのが増えます。よって、適当に短くできる頂点の集合を増やし、短くできる辺を見つけたら短くすることで、短縮を効率よく行えます。

上には概略を述べましたが、詳しくは writer のコードを確認してもらえると、分かりやすいと思います。全体で $O(N^2 + NM)$ の計算量で解くことができます。

AGC 033 Editorial

yutaka1999

May 5, 2019

A : Darker and Darker

Let us consider what squares will become black after one operation. In different words from the statement, our operation is as follows:

- For each black square, paint the squares adjacent to it black.

By rephrasing this way, we can quickly simulate the process. We needed to observe all white squares every time if we are to literally simulate the process, but now we only need to observe each square once: we only need to observe a square when it is being painted black, and after we paint the squares adjacent to it black, we never need to observe it again.

In the implementation, we recommend using a queue in the similar way to breadth-first search. Since the rephrased operation can be regarded as performing breadth-first search from the black squares in the grid graph, this approach is quite natural. The time complexity of this solution is $O(HW)$: linear on the size of the grid.

B : LRUD Game

First, notice that the two directions in this game, horizontal and vertical, are independent from each other. That is, we just need to solve the one-dimensional version of this problem, as follows:

- There is a piece on a bar of length L.
- Two players alternately performs N moves each. Each turn is in one of the following forms:
  - The player can move the piece to the left or do nothing.
  - The player can move the piece to the right or do nothing.
  - The player cannot move the piece this time.
- The player who goes first wants to drop the piece from the bar, while the player who goes second wants to prevent it.
- Which player will win?

Now, the problem should be easier. We just need to calculate the area such that the second player can win if the piece is in that area in the i-th turn, in the descending order of i.

If you missed the first observation that the two directions are independent, you can still solve the problem similarly, since the area from which the second player can win is a rectangle. The complexity of the solution is linear.

C : Removing Coins

Let us consider the subtree formed by the vertices that still contain coins. In each operation, it is changed as follows:

- Make the tree rooted by designating the chosen vertex as the root.
- Erase all the leaf vertices. Erase the root if it is the only vertex.

Now, let us pay attention to the diameter of this tree. Obviously, the diameter decreases by 1 or 2 in each operation. We can always choose one end of the diameter to reduce the diameter by 1. Also, when the diameter is at least 2, the tree has a non-leaf vertex and we can always choose it to reduce the diameter by 2. Thus, we can consider the game to be played as follows:

- Given is the diameter L.
- Each of the two players can reduce L by 1 or 2 without making it negative.
- The player who lets $L = 0$ loses. Find which player will win.

This is a quite common game. The second player will win if $L \equiv 1 \mod 3$, and the first player will win otherwise. Since the diameter of a tree can be found in $O(N)$ time, the problem can be solved in $O(N)$ time.

D : Complexity

Note that a 1×1 subgrid has a complexity of 0. The grid will be divided into such subgrids after approximately $\log_2 H + \log_2 W$ divisions, so we can see that the complexity of the grid is $O(\log H + \log W)$. Now, let us consider computing the following values:

- The maximum horizontal length of a subgrid with complexity i whose top-left and bottom-left squares are (r, c) and (r', c)
- The maximum vertical length of a subgrid with complexity i whose top-left and top-right squares are (r, c) and (r, c')

There are $O(HW(H+W)\log HW)$ values, so it will run in time if we can compute each of them in $O(1)$, which is actually possible by computing them in ascending order of i.

E : Go around a circle

First, assume that each arc is already painted red or blue, and let us think of the way to determine if that coloring satisfies the condition.

□ If S consists of a single color

We assume that S consists only of R. If there are two consecutive blue arcs along the perimeter, the condition is obviously violated. On the other hand, it is sufficient if every point is neighboring to a red arc. Thus, the necessary and sufficient condition here is that there is no two consecutive blue arcs.

□ If S contains both R and B

We assume that S begins with R. Similarly to the above, two consecutive blue arcs obviously violate the condition, so it is necessary that there is no two consecutive blue arcs. Additionally, as S contains a B, there must be at least one blue arc. It is not sufficient this time, though. Assume that some even number of consecutive arcs are red, and the arcs next to them are blue. Let the first k characters in S be Rs and the next character is B. Then, wherever along the even number of red arcs we place the checker in the beginning, after k moves along a red arc we must be at the end of the sequence of red arcs, where we can make a move along a blue arc. However, this cannot be achieved if k is odd and we begin from an end of the red sequence, or k is even and we begin from a point next to that end. Therefore, the number of the consecutive red arcs must be odd. Similarly, we see that there is an upper limit to the number of the consecutive red arcs. For simplicity, assume that S ends with B. (Trailing Rs can be ignored.) Assuming that the first k characters are Rs, there can be at most $k+1$ consecutive red arcs if k is even, and at most k consecutive red arcs if k is odd. If there are k' consecutive Rs at other than the beginning of S, the piece is guaranteed to be next to a blue arc when we make the moves corresponding to those Rs since we have just moved along a blue arc. Thus, if k' is even, the moves corresponding to those Rs are always possible. If k' is odd, it is necessary that there are at most k' consecutive red arcs. To sum it up, the necessary conditions, which can be easily verified to be necessary and sufficient, are as follows:

- There is a blue arc, but not two in a row.
- The length of every sequence of consecutive red arcs is odd.
- There is at most L consecutive red arcs, where L is some integer derived from

S.

We have now rephrased the condition, but how do we count the colorings? It is not trivial even if S consists of a single color, but the solution is similar to the case with two colors, so here we will only handle the latter.

First, since the length of every sequence of arcs of the same color is odd, the number of arcs must be even. Also, the parities of the indices of all blue arcs must be the same. We can safely assume that blue is only used on the arcs with even indices, since fixing it to odd indices results in the same number of colorings. Then, all arcs with odd indices must be painted red, and it seems that we do not explicitly consider them. Actually, the problem is equivalent to the following:

- We paint $\frac{N}{2}$ arcs blue or red.
- At least one arc must be painted blue.
- There can be at most L consecutive red arcs.
- Find the number of colorings under the above conditions.

It is difficult on a circle, so let us consider the linear version, that is, the following problem:

- There is a bar divided into X segments of equal length. We paint each segment red or blue.
- The both ends must be painted blue.
- There can be at most L consecutive red segments.
- Find the number of colorings under the above conditions.

This can be solved in $O(X)$ time by dynamic programming with prefix sums. Let $dp[i]$ be the answer for the case $X = i$. Then, $dp[i] = \sum_{j=max(1,i-L-1)}^{i-1} dp[j]$ holds, so if we let $rdp[i] = \sum_{j=1}^{i} dp[j]$, $dp[i] = rdp[i-1] - rdp[\max(0, i-L-2)]$. Thus, by updating $dp[i]$ and $rdp[i]$ in the order $i = 1, \ldots, X$, the problem can be solved with the time complexity $O(X)$.

Now, let us get back to the original problem. It turns out that we can easily found the answer by using $dp[i]$ above. Let us cut the perimeter at some point. Then, we want to count the number of colorings such that there is at least one blue arc, at most L consecutive red arcs and the total number of the consecutive red arcs at both ends

is at most L. If we fix the distance between the leftmost and rightmost blue arcs to be j, the number of ways to color the arcs between them is $dp[j+2]$. By multiplying it by some coefficient, the problem is now solved in linear time.

F : Adding Edges

Instead of adding edges, which is difficult, let us consider "shortening" edges, that is, performing the following operation on G:

- When G has edges ab and ac and the vertices a, b, c in T lie on some path in this order, erase the edge ac and add an edge bc.

After this operation, the final set of edges in G that can be generated by adding edges does not change, so let us consider the situation where we have performed this operation repeatedly until we can no longer do it. In this situation, it can be shown that we can represent an edge that can be generated as follows:

- An edge xy can be generated if and only if there exists a sequence of vertices $x = a_0, a_1, \ldots, a_k = y$ such that G contains the edges $a_i a_{i+1}$ and the vertices a_i, a_{i+1}, a_{i+2} in T lie on some path in this order.

Proof. The sufficiency of the above condition is obvious, so here we will only show the necessity. We can prove it by showing that, if there are two edges ab, ac satisfying the condition and the vertices b, a, c in T lie on some path, the edge bc generated from these two edges also satisfy the condition. If the vertices b, a, c lie on the path in this order, the condition is obviously satisfied. Otherwise, consider the sequences of vertices for the edges ab, ac that the condition requires. From the fact that we can no longer perform the operation of shortening an edge in G, we can see that there exists a vertex x ($x \neq a$) along the generated edges ab, ac such that the edges xb, xc satisfy the condition to be generated. By applying this procedure repeatedly, x will finally coincide with b or c, which shows that the edge bc satisfies the condition to be generated. □

Thus, when we cannot shorten any edge in G, we can solve this problem by BFS or similar methods in $O(NM)$ time. Now, let us consider how we can shorten the edges in G. One way to do this is: for each vertex, we compute the euler-tour when that vertex is considered as the root, and efficiently search the edges to be shortened with `set` or similar structures. However, we will have log in the complexity, and it is a little tough to implement. Here is an $O(NM)$ approach to shorten the edges:

We will add the edges from the input one by one to the graph and gradually shorten

them. For each vertex v, we maintain the set of vertices such that an edge from v can be shortened if the other endpoint is in that set. When we add an edge, if it can be shortened according to the information above, we will shorten it and do this again. If we cannot shorten the edge after considering both endpoints, the set above for each of the endpoints will have new elements. For more detail, see the writer's code (link). The total complexity of the solution is $O(N^2 + NM)$.