AGC 046 解説

DEGwer

2020/06/20

For International Readers: English editorial starts on page 6.

A: Takahashikun, The Strider

高橋君のいる初期地点を A とし、1 回の移動後にいる地点を B としましょう。$|AO| = |BO|$ かつ角 AOB の大きさが X 度となるような点 O を、高橋君の初期地点から見て左側にあるように取ります。このとき、各移動後に高橋君のいる位置は、すべて O を中心とする半径 $|AO|$ の円周上に載ることがわかります。結局、高橋君の移動は円周上を X 度ぶんだけ移動するものと見做せるので、kX が 360 の倍数となる最小の k が答えになります。

B: Extension

各盤面の塗られ方を、ひとつの操作列に対応させることを考えましょう。一番上の行に黒く塗られたマスが複数ある (もしくは存在しない) 場合、最後の操作は右に列を追加する操作です。逆に、一番上の行に黒く塗られたマスが 1 つだけある場合、一番上の行を取り除いてできる盤面も、初期状態から作ることができることが分かります。よって、初期状態から作ることができる各盤面から、以下のようなアルゴリズムで一意な操作列を復元することができます。

- 一番上の行に黒く塗られたマスが複数ある場合、一番右の列を取り除いてできる盤面を作る操作列を求めたあと、右に列を追加する操作を付け加える
- そうでない場合、一番上の行を取り除いてできる盤面を作る操作列を求めた後、上に行を追加する操作を付け加える

このような操作列は、i 行 j 列のマス目であって一番上の行に黒く塗られたマスが複数あるような/そうでないような盤面の個数をそれぞれ $\mathtt{DP}[i][j][0], \mathtt{DP}[i][j][1]$ としたような動的計画法を用いて $O(CD)$ 時間で数え上げることができます。

C: Shifts

各 0 を文字列の区切りと考えましょう。0 が A 個あるとすれば、0 によって文字列は両端を含めた $A+1$ 個の部分に分割できます。このとき、問題で与えられている操作は、ある区間に存在する 1 を 1 個減らし、それより左の区間に 1 を 1 個増やすものととらえることができます。

$\mathtt{DP}[i][j][k]$ を、後ろから i 個目の区間まで見て、j 回の操作を行い、それより後ろの区間に合計で 1 を k 個増やしてできるような列の個数とすれば、この DP は $O(N^4)$ 時間で更新可能です。各区間に対してまとめて操作を行うのではなく、1 回ずつ分けて行うように気を付けて実装すれば、この DP は $O(N^3)$ 時間に高速化できます。

D: Secret Passage

S の文字の中で、S の先頭の文字として操作を行われたか、あるいは 2 文字目として操作を行われた結果取り除かれたかした文字を「動いた文字」と呼ぶことにします。動いた文字の中で取り除かれなかったものを「挿入された文字」と呼びます。S の動いた文字全体は、S の接頭辞をなします。作れる文字列は、挿入された文字を、動かなかった文字たちの間 (または端) に自由に挿入することで得られるものです。

できる列を T とします。S の接尾辞であって T の部分列になっているようなものの中で最長のものを S' とします。まず、定義より、S' に属していない文字はすべて動いた文字であることが分かります。逆に、S' の文字が動いた文字とならないような操作列であって T をつくるものが存在することが証明できます。実際、S' の先頭 k 文字を追加で動かすことで T を作ることができたとします。このときに挿入された文字の多重集合から S' の先頭 k 文字を除いたものを X とします。 X が挿入された文字の多重集合となるような、S' の文字を動かさない操作列が存在することを証明すればよいです。これは、S' の先頭 k 文字を追加で動かして T を作る場合 S' 以外の挿入される文字の多重集合は X を含まなければならず、X を含む多重集合を挿入される文字の多重集合とするならば、適切に操作を追加することで X を挿入される文字の多重集合ともできることから分かります。

以上より、S' を全通り試し、それぞれの場合の数を足し合わせることで答えを求めることができます。先頭 k 文字を動かすことで挿入される文字の多重集合を 0 を i 個、1 を j 個にできるかどうかを $\mathtt{DP}[k][i][j]$ とする動的計画法によって挿入される文字の多重集合としてありうるものを列挙し、0 を i 個と 1 を j 個挿入することで S の末尾 k 文字から得られる相異なる文字列の個数を $\mathtt{DP}'[k][i][j]$ とする動的計画法によって作れる文字列の個数を数えます。これらを組合わせれば、この問題を解くことができます。どちらの動的計画法も適切な実装によって $O(N^3)$ 時間で動くように実装できるので、時間計算量は $O(N^3)$ 時間となります。

E: Permutation Cover

以下、$[n]$ で集合 $\{1, \ldots, n\}$ を表します。

■ステップ 1. まず、条件を満たす列が存在するための必要十分条件は、A_i たちの中の最大値が A_i たちの中の最小値の 2 倍以下であることです。まずはこれを証明しましょう。まず、A_i たちの中の最大値/最小値を取るような i を 1 つずつとって x, y とします。$A_x > 2A_y$ なら、できる列の x と y だけを取り出してできる列には、x が 3 個 (または端に 2 個) 連続するような箇所が存在します。このような箇所があれば、それらの中の真ん中 (端) の要素はどの順列をなす連続部分列にも含まれず、この条件は必要です。逆に、任意の集合 $S \subseteq [K]$ に対し、S の要素を 2 個ずつ、$[K] \setminus S$ の要素を 1 個ずつ含む列であって、どの項も順列をなすある連続部分列に含まれるようなものが作れます。$A_x \leq 2A_y$ なら、このような列を並べたものとして条件を満たす列を作れるので、この条件は十分条件でもあります。

■ステップ 2.　ステップ 1. の条件が満たされているとして、条件を満たす辞書順最小の列を構成する方法を考えましょう。どの要素も順列を成すある連続部分列に含まれるような列 P を考えます。特に、P の最後 K 項は順列をなします (この順列を Q とします)。また、B_i でこれから追加する必要のある整数 i の個数を表すことにします。 P にまだ追加していない要素を追加することで全体を条件を満たす列にできる必要十分条件は、以下であることが示せます。

- B_i たちの中の最大値を x とし、最小値を y として、$x \leq 2y+1$ である
- $x = 2y+1$ なら、Q の中では、$B_i = x$ なるどの i も $B_i = y$ なる任意の i よりも前に現れる

1 つ目の条件の必要性は、これから追加すべき項からなる列に対してステップ 1. と同様の議論を行うことによって証明できます。2 つ目の条件の必要性も、$B_i = x$ なる i が P で最後に現れる位置以降の列にこれから追加すべき項を追加したものに対して同様の議論を行うことで証明できます。十分性は、$B_i = x$ なる i が Q の先頭 s 文字にすべて現れるような最小の s を取れば、P に Q の先頭 s 項を追加したような列 R を考えれば、R のどの項も順列をなるある連続部分列に含まれ、かつこれから追加すべき項たちの個数はステップ 1. の条件を満たすことから分かります。

■アルゴリズム　現在の列がステップ 2. の条件をみたすとして、これをどの要素も順列をなすある連続部分列に含まれるようなより長い列に拡張することを考えましょう。現在の列を何文字伸ばせばはじめてそのような列になるかを全通り試し、各候補に対してステップ 2. の条件を満たすような伸ばし方が存在するか判定し、存在する場合は辞書順最小のものを求めます (これは簡単な貪欲法で可能です)。こうして求めた伸ばし方のうち、辞書順で最小のものを取って列を拡張すればよいです。この操作を追加すべき項がなくなるまで繰り返すことで、この問題を解くことができます。時間計算量は、A_i たちの合計を S として、$O(SK^2)$ 時間です。

F: Forbidden Tournament

以下、問題文で与えられている禁止部分グラフを H とします。すなわち、H は 4 頂点のグラフであって、そのうち 3 点が三角形をなし、その 3 点すべてから残りの 1 点に向かって辺が出ているようなトーナメントグラフです。また、G が H を誘導部分グラフに持つとき、G が H を **含む** と呼びます。以下、G の構造を、ステップに分けて眺めていきましょう。適宜、グラフの頂点集合とそれが誘導するグラフを同一視します。

■ステップ 1.　トーナメントグラフ G を固定します。G に入次数 0 の頂点 x があるとします。このとき、G が H を含まず入次数の最大値が K 以下であることは、G から x を取り除いたグラフが H を含まず入次数の最大値が $K-1$ 以下であることと同値です。このようなケースは N が 1 小さい場合の問題を解いて求めることにして、以下、G のどの頂点の入次数も 1 以上であると仮定します。

■ステップ 2.　G の頂点 v をひとつ任意に取って固定します。Y を $v \to w$ 辺があるような頂点 w の集合とし、$X = V(G) \setminus Y$ とします ($v \in X$ に注意)。もし X がサイクルを含めば、それは三角形を含み (トーナメントグラフの性質)、その三角形と v を併せて H を得ます。よって、X の頂点は、任意の $i < j$ に対して $x_i \to x_j$ 辺があるように $x_1, \ldots, x_k = v$ と番号づけられます。

■ステップ 3. 入次数 0 の頂点がないので、Y の頂点から X の頂点に出る辺は必ず存在します。このような辺を任意に取り、$w \to u \quad (w \in Y, u \in X)$ とします。$w \to w'$ 辺の存在する頂点 $w' \in Y$ が存在したとします。このとき、$u \to v, v \to w, w \to u, w \to w', v \to w'$ の 5 辺が存在するので、G が H を含まないためには $w' \to u$ 辺が存在する必要があります。この議論を繰り返し行えば、Y 内で w から到達できるすべての頂点 w' について $w' \to u$ 辺が存在する必要があることが分かります。Y 内で w から到達できる頂点たちの集合がサイクルを含めば、それは三角形を含み、その三角形と u を併せて H を得ます。よって、Y 内で w から到達できる頂点たちの集合はサイクルを含みません。逆に、Y 内に w から到達できないようなサイクルがあれば、その中の三角形と w を併せて H を得ます。すなわち、Y もサイクルを含まず、Y の頂点を任意の $i < j$ に対して $y_i \to y_j$ 辺があるように $y_1, \ldots, y_l$ と番号づけることができます。

■ステップ 4. X 内および Y 内の辺の向きはすべて定まったので、以下 X と Y の間の辺の向きを考えていきましょう。ステップ 3. での考察により、$y_j \to x_i$ 辺があるとき、任意の $j' = j, \ldots, l$ に対し、$y_{j'} \to x_i$ 辺が存在することが分かります。このことより、仮に $x_1 \to y_l$ 辺が存在すれば、任意の $j' = 1, \ldots, l$ に対し $x_1 \to y_{j'}$ 辺が存在することになり、x_1 の入次数が 0 になって矛盾します。よって、$y_l \to x_1$ 辺が存在します。

■ステップ 5. $2 \leq i \leq k$ として、$x_i \to y_l$ 辺が存在すると仮定します。このとき、もし $i < i'$ に対して $y_l \to x_{i'}$ 辺が存在すれば、$x_i, x_{i'}, y_l, x_1$ の 4 点が H を誘導します。よって、$x_i \to y_l$ 辺が存在すれば、任意の $i' = i, \ldots, k$ に対して $x_{i'} \to y_l$ 辺が存在します。t を $y_l \to x_t$ 辺が存在するような最大の整数としてとり、$1 \leq i \leq t$ とします。$1 \leq j \leq l-1$ とし、$x_i \to y_j$ 辺が存在するとします。このとき、x_i, y_j, y_l は三角形をなします。特に、$i < i' \leq k$ に対し、G が H を含まないためには $x_{i'} \to y_j$ 辺か $x_{i'} \to y_l$ 辺のどちらかが必要です。ステップ 3. での考察により、$x_{i'} \to y_l$ 辺が存在するなら $x_{i'} \to y_j$ 辺も存在するため、結局 $x_{i'} \to y_j$ 辺が必要なことがわかります。

■ステップ 6. $k \times l$ のグリッドであって、$y_j \to x_i$ 辺があるような (i, j) に対して i 行 j 列目のマスを黒く塗ったものを考えます。ここまでの考察により、このグリッドは以下を満たす必要があることが分かります。

- (i, j) が黒く塗られているなら、$i' < i$ に対して (i', j) も黒く塗られている
- (i, j) が黒く塗られているなら、$j < j'$ に対して (i, j') も黒く塗られている
- $(1, l)$ は黒く塗られている
- 任意の j に対し、(k, j) は黒く塗られていない

逆に、上の 2 つの条件を満たせば G は H を誘導部分グラフに持たないことは、H のサイクルがどこに現れるかで場合分けをすることで示すことができます。よって、黒く塗られていないマスと塗られていないマスの境界は、グリッドの対角線上の 2 頂点を (マンハッタン距離最小で) 結ぶパスとして表すことができます。入次数の条件は、このパスがある 2 つの階段状領域の点を含まないという条件で記述することができます。

■アルゴリズム. 結局、k, l の値を決め打ちすれば、この問題はグリッド上に与えられた 2 点間をマンハッタン距離最小のパスで結ぶ方法の個数を数える問題に帰着されることが分かりました。頂点 $x_1, \ldots, x_k$ と $y_1, \ldots, y_l$ の取り方が $N!$ 通りあり、条件を満たす各グラフ G について v の選び方は N 通り存在するので、このようなパスの個数を $(N-1)!$ 倍すると求める場合の数が得られます。また、このようなパスの個数は動的計画法で $O(kl)$ 時間で求めることができます。k, l の定め方は $O(N)$ 通り、最初に入次数 0 の頂点を取り

除く回数の候補が $O(N)$ 通り存在するので、時間計算量は合計で $O(N^4)$ となります。

A: Takahashikun, The Strider

Let A be Takahashi's initial position, and B be his position after one move. Let O be the point to the left side of Takahashi's initial position such that $|AO| = |BO|$ and the angle AOB is X degrees. Then, we can see that Takahashi's position after every move lies on the circle of radius $|AO|$ centered at O. We can now regard Takahashi's move as moving X degrees on the circle, so the answer is the minimum k such that kX is a multiple of 360.

B: Extension

Let us correspond each way of painting the grid to one sequence of operations. If there are multiple (or no) black squares in the top row, the last operation is the addition of a column to the right end. On the other hand, if there is just one black square in the top row, we can see that the grid without the top row is also reachable from the initial state. Thus, from each state of the grid that is reachable from the initial state, we can restore a unique sequence of operation with the following algorithm:

- If there are multiple black squares in the top row, find the sequence of operations resulting in the grid without the rightmost column and add a column to the right end.
- Otherwise, find the sequence of operations resulting in the grid without the top row and add a row to the top.

With Dynamic Programming where $\texttt{DP}[i][j][0], \texttt{DP}[i][j][1]$ are the number of grids with i rows and j columns where there (are / are not) multiple black squares at the top row, we can count such sequence of operations in $O(CD)$ time.

C: Shifts

Consider each 0 as a separator of the string. If the string has A 0s, they separate the string into $A+1$ parts, including both ends. Then, the operation given in the problem can be seen as removing one 1 from a segment and adding one 1 to a segment to the left.

Let $\texttt{DP}[i][j][k]$ be the number of sequence that can be obtained considering the 1-st, ..., i-th segment from the back, having done j operations, and adding k 1s to the later segments. We can calculate this table in $O(N^4)$ time. We can improve it to $O(N^3)$ time by careful implementation: instead of collectively applying the operation to the segments, we can handle them separately.

D: Secret Passage

We call a character in S a "moved" character if we did the operation when that character was the first character of S, or the second character and got removed. We call a moved but unremoved character an "inserted" character. The set of all the moved characters in S forms a prefix of S. A string we can make

is a string we can obtain by freely inserting the "inserted" characters between unmoved characters (or to the ends of the string).

Let T be an obtainable string. Let S' be the longest suffix of S that is a subsequence of T. First, from the definition, we can see that every character not belonging to S' is a moved character. On the other hand, we can prove that there exists a sequence of operations resulting in T such that no character in S' is a moved character. Assume that we were actually able to make T by additionally moving the first k characters in S'. Let X be the multiset of the inserted characters here except the first k characters in S'. Now we have to prove that there is a sequence of operations without moving the characters in S' such that X will be the multiset of the inserted characters. This can be seen from the following fact: when we additionally move the first k characters in S' to make T, the multiset of the inserted characters not in S' must contain X, and if the multiset of the inserted characters contains X, we can make X the multiset of the inserted characters by properly adding operations.

Therefore, we can try all possible S' and sum the count for each case to find the answer. We will use Dynamic Programming. Let $\mathtt{DP}[k][i][j]$ be whether the multiset of the inserted character can be i 0s and j 1s by moving the first k characters. We will use it to enumerate the possible multisets of inserted characters. Also, let $\mathtt{DP}'[k][i][j]$ be the number of different strings that can be obtained from the last k characters of S by inserting i 0s and j 1s. We use it to count the strings obtained. The problem can be solved by the combination of these two. Both tables can be computed in $O(N^3)$ time when properly implemented, so the time complexity of our solution is $O(N^3)$.

E: Permutation Cover

Below, let $[n]$ denote the set $\{1, \ldots, n\}$.

■Step 1. First, there exists a sequence satisfying the condition if and only if the maximum element among A_i is at most twice the minimum element among A_i. Let us prove it now. Let x and y be the indices i such that A_i is maximum and minimum. If $A_x > 2A_y$, the sequence obtained by extracting only xs and ys from the resulting sequence contains a part where x appears three times in a row (or twice at one end). If such a part exists, the middle one among them (or the one at the end) is not contained in any contiguous subsequence that is a permutation, so this condition is necessary. On the other hand, for any set $S \subseteq [K]$, we can form a sequence containing each element in S twice and each element in $[K] \setminus S$ once such that every term is contained in some contiguous subsequence that is a permutation. If $A_x \leq 2A_y$, we can satisfy the condition by forming such a sequence, so this condition is also sufficient.

■Step 2. Assuming that the condition in Step 1. is satisfied, let us consider the way to construct the lexicographically smallest such sequence. Consider a sequence P where every element is contained in some contiguous subsequence that is a permutation. Notably, the last K terms in P form a permutation (let Q be this permutation). Also, let B_i be the number of times the integer i currently needs to be added. We can show that we can satisfy the condition by adding P the elements that are not yet added if and only if:

- Let x and y be the maximum and minimum among B_i. Then, $x \leq 2y + 1$ holds.
- If $x = 2y + 1$, every i such that $B_i = x$ appears earlier in Q than every i such that $B_i = y$ does.

We can prove the necessity of the first condition by making an argument similar to that in Step 1. for the sequence of terms to be added. We can also prove the necessity of the second condition by making a similar argument for the part of the sequence after the last occurrence of i such that $B_i = x$ in P, with the terms to be added appended at the end. The sufficiency can be seen as follows. Let us take the minimum s such that the first s characters of S contains every i such that $B_i = x$ appear. Then, consider the sequence R obtained by appending the first s terms of Q to P. Now, every term in R is contained in some contiguous subsequence that is a permutation, and the numbers of terms that should be added satisfy the condition in Step 1., so the necessity is shown.

■Algorithm Assuming that the current sequence satisfies the condition in Step 2., let us consider making it a longer sequence such that every term is contained in some contiguous subsequence that forms a permutation. We will try all candidates of the minimum number of characters we need to add to the current sequence so that we obtain a desirable sequence. Then, for each candidate, we will determine whether there is a way to extend the sequence satisfying the condition in Step 2., and find the lexicographically smallest such one if it exists (we can do it with a simple greedy algorithm). Among these ways to extend the sequence, we will use the lexicographically smallest one and extend the sequence. The problem can be solved by repeating this procedure until there are no more terms to be added. The time complexity of our solution is $O(SK^2)$, where S is the sum of A_i.

F: Forbidden Tournament

Sorry, will be ready in a day.