AGC038 解説

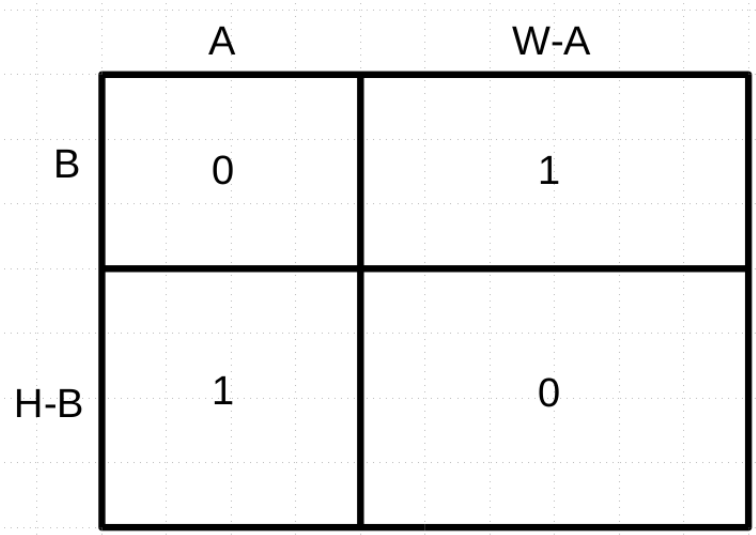
writer : maroonrk

令和元年 9 月 21 日

For International Readers: English editorial starts on page 7.

A : 01 Matrix

以下の図のようなマス目を構築することで、必ず条件をみたすことができます。



解答例

B : Sorting a Segment

区間 $[i, i+K)$ と区間 $[j, j+K)$ $(i<j)$ をソートした結果が同じになることを、i, j が equivalent であると呼ぶことにします。以下、i, j が equivalent になる条件について考えます。

i, j が equivalent である一つの場合は、区間 $[i, i+K)$ と $[j, j+K)$ がどちらももともと昇順に並んでいる場合です。

そうでない場合を考えます。まず、$[i, i+K)$ と $[j, j+K)$ が共通部分を持つ必要があることは明らかです。よって、$j < i+K$ は必要条件です。また、区間 $[i, j)$ は、区間 $[i, i+k)$ を昇順に並び替えたあとでも変化しないことがわかります。これは、区間 $[i, j)$ の値がもともと昇順に並んでおり、かつどの値も $[j, i+k)$ の中にあるどの値よりも小さいことを意味しています。同様にして、区間 $[i+K, j+K)$ の値は、もともと昇順に並んでおり、かつどの値も $[j, i+K)$ の中にあるどの値よりも大きいことになります。

すると、i, j が equivalent なら、任意の x $(i<x<j)$ について、x は i, j と equivalent であることがわかります。特に、i と $i+1$ は equivalent です。

この問題で求めたい答えは、頂点 $0, 1, \cdots, N-K$ のグラフで、i, j が equivalent である場合にのみ i, j 間に辺を張ったものの、連結成分の数となります。先程の考察より、$[i, i+K)$ がもともと昇順に並んでいないものに関しては、$i, i+1$ 間の辺についてのみ考えれば良くなります。$i, i+1$ が equivalent であるかどうかは、$[i, i+K+1)$ の中の最小、最大がそれぞれ P_i, P_{i+K} であるかどうかで判定できます。

スライド最小値を用いれば $O(N)$ 時間で、全ての i について $i, i+1$ が equivalent であるか判定できます。その他の処理も $O(N)$ でできるので、この問題は全体で $O(N)$ で解けます。(なお、std::set を用いれば実装を簡略化でき、計算量は $O(NlogN)$ になりますがこれでも十分高速です)

解答例

C : LCMs

入力される整数の最大値を $V(=1000000)$ とします。次の条件をみたす有理数列 $w_1, w_2, \cdots, w_V$ を考えます。

- 全ての $1 \leq i \leq V$ について、$\sum_{d|i} w_d = 1/i$

このような数列は、i の昇順に定めていくことができます。つまり、$w_1, w_2, \cdots, w_{k-1}$ までの値が定まっている時、$w_k = 1/k - \sum_{d|i, d<i} w_d$ とすれば良いです。この計算量は、$1, 2, \cdots, V$ の約数の個数の総和と同じオーダーであり、$O(VlogV)$ です。

$lcm(x, y) = xy/gcd(x, y)$ です。これはさらに、次のように言い換えられます。

$$xy/gcd(x, y) = xy \times \left(\sum_{d|gcd(x,y)} w_d \right) = xy \times \left(\sum_{d|x, d|y} w_d \right)$$

よって、この問題で求める式は、

$$\sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} lcm(A_i, A_j) = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} A_i A_j \times \left(\sum_{d|A_i, d|A_j} w_d \right)$$

となります。ここで、d に注目してこの式を書き直してみると、以下のようになります。

$$\sum_{1 \leq d \leq V} w_d \left(\sum_{d|A_i, d|A_j, i<j} A_i A_j \right)$$

$\sum_{d|A_i, d|A_j, i<j} A_i A_j$ の部分に注目すると、これは、A の要素のうち d の倍数から 2 つ選んでかけたものの総和となっています。これは、各 x $(1 \leq x \leq V)$ について、$A_i = x$ となる i が何個あるかを予め数えておけば、$O(V$ 以下の d の倍数の個数 $)$ で求められます。これを全ての d について行うと、全体で計算量は $O(VlogV)$ になります。

よって、この問題は $O(N + VlogV)$ で解けました。

解答例

D : Unique Path

$M = N - 1$ の場合、すぬけくんがもらったグラフは木です。その場合、どの頂点ついについても単純パスは 1 個です。以下、$M \geq N$ の場合について考えます。

すぬけくんがもらったグラフを G と呼ぶことにします。G の橋について考えてみます。

すぬけくんがもらったグラフの橋だけを残したグラフを H と呼びます。まず、単純パスが 1 つしか存在しない頂点対 a, b は、H において同じ連結成分に属していなければなりません。(そうでない場合は、G において橋以外を通る a, b パスが存在する。ここで、そのパス上の橋以外の辺を一個消しても G が連結であることから、もう一つの a, b パスが存在することになり、矛盾する) また、単純パスが 2 個以上ある頂点対 a, b は、H において異なる連結成分に属していなければなりません。(そうでない場合は、a, b の属する連結成分の辺を上手く選んで 1 つ削除すると a, b が連結であるようにでき、矛盾する。)

よって、単純パスが 1 つしかない頂点間に辺を張ったグラフ I を考えると、単純パスが 2 つ以上存在する頂点対は、I において異なる連結成分に属する必要があるとわかります。

H の連結成分の個数 k を固定してみます。G において、H の異なる連結成分 x, y 間に張られているような辺を考えます。まず、同じ x, y 間に 2 つ以上の辺が張られていてはいけません。よって、G の辺数の最大値は H の辺数 $+ k(k-1)/2 = N - k + k(k-1)/2 = N + k(k-3)/2$ です。また、G の辺数の最小値を考えると、これは N です。(H の各連結成分から代表点を選び、それらを結ぶサイクルを作れば良いです。) また明らかに、すべての $N \leq i \leq N + k(k-3)/2$ について、G の辺をちょうど i 本にすることができます。

よって、H の連結成分数 k はできるだけ大きい方がよく、これは I の連結成分数そのものであることがわかります。あとは、$M \leq N + k(K-3)/2$ であるかどうかを判定すればよいです。

解答例

E : Gachapon

$A'_i = A_i/S$ とします。

包除原理を使うと、以下の問題が解ければよいことになります。

> $\{0, 1, \cdots, N-1\}$ の空でない部分集合 $s = \{i_0, i_1, \cdots, i_{|s|-1}\}$ すべてについて、次の問題を解き、その答えを $|s|$ の偶奇によってそれぞれ $-1, 1$ の重みをかけて足し合わせよ。
>
> - すべての j $(0 \leq j \leq |s|-1)$ について i_j の登場回数が B_{i_j} 回未満であるような状態が、何回続くか期待値を求めよ。

部分集合 s が固定されている場合を考えます。

整数列 $x_0, x_1, \cdots, x_{|s|-1}$ $(0 \leq x_j < B_{i_j})$ について、i_j の登場回数がちょうど x_j 回であるような状態を、状態 x と呼ぶことにします。状態 x になる回数の期待値を考えます。乱数生成器が $|s|$ の中に含まれる数を生成する確率を P とすると、初めて状態 x になった後、それが維持される回数の期待値は $1/P$ です。よって、一度でも状態 x になる確率を数えれば良いです。これは、次式で求めることができます。ただし、$X = \sum_{i=0}^{|s|-1} x_i$ です。

$$X! \prod_{j=0}^{|s|-1} \left(\frac{A'_{i_j}}{P} \right)^{x_j} \frac{1}{x_j!}$$

求めたいのは、整数列 x としてあり得る全てのものについて上式の値を足し合わせたものです。そしてこれは、DP で求めることができます。(DP のキーとして、今 x の先頭何項目まで決めたか、今まで決めた x の項の総和を持てば良いです)

最後に、ありうる全ての $|s|$ について、上の DP を計算することを考えます。すると、上記の DP のキーの他に、今まで s に追加することに決めた i の A_i の総和をキーとして持てば、計算できることがわかります。

DP の計算量は全体で $O\left(\left(\sum_{i=0}^{N-1} A_i\right)\left(\sum_{i=0}^{N-1} B_i\right)^2\right)$ となり、十分高速です。

解答例

F : Two Permutations

$i \to P_i$ の辺を張ったグラフを考えます。このグラフはいくつかのサイクルからなるグラフです。また、A を作る際には、各サイクルについて、そのサイクルに含まれるすべての i について $A_i = i$ にする、またはすべての i について $A_i = P_i$ とする、のどちらかを選ぶことになります。ここで、前者を選ぶ場合を、そのサイクルに 1 を割り当てることに、後者を選ぶ場合を 0 を割り当てることに対応させます。頂点 i の含まれるサイクルを $x(i)$ と書くことにします。

また、$i \to Q_i$ の辺を張ったグラフを考えます。先ほどと同様に、B を作ることと、各サイクルに 0,1 の値を割り振ることを対応させます。ただし今回は、$B_i = i$ とすることに 0 を、$B_i = Q_i$ とすることに 1 を対応させることにします。また、頂点 i の含まれるサイクルを $y(i)$ と書くことにします。

ある i について、$A_i = B_i$ となるかどうかは、$x(i)$ と $y(i)$ に割り当てられた値によって決定されます。具体的には以下のようになります。ここで、サイクル c に割り当てる値を $v(c)$ と書いています。

- $P_i = Q_i = i$ の時: どのようにしても、$A_i = B_i$
- 上記の条件に当てはまらず、$P_i = i$ の時: $A_i = B_i \Leftrightarrow v(y(i)) = 0$
- 上記の条件に当てはまらず、$Q_i = i$ の時: $A_i = B_i \Leftrightarrow v(x(i)) = 1$
- 上記の条件に当てはまらず、$P_i \neq Q_i$ の時: $A_i = B_i \Leftrightarrow v(x(i)) = 1 \ and \ v(y(i)) = 0$
- 上記の条件に当てはまらない時: $A_i = B_i \Leftrightarrow (v(x(i)) = 1 \ and \ v(y(i)) = 0) \ or \ (v(x(i)) = 0 \ and \ v(y(i)) = 1)$

これは、Project selection problem として定式化できます。すると、必要なのは、$O(N)$ 頂点、$O(N)$ 辺、各辺の容量 1 のグラフの最大流になります。これは、dinic 法を使うと $O(N\sqrt{N})$ で解くことができ、十分高速です。

解答例

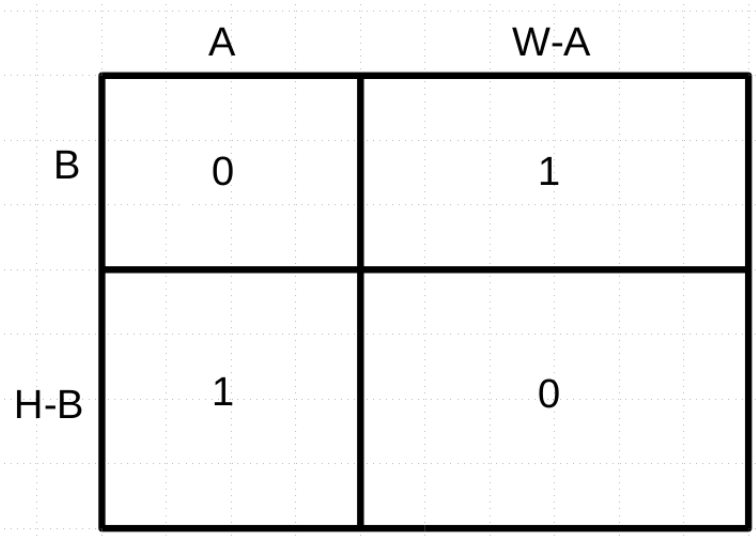
AGC038 Editorial

written by maroonrk

September 21, 2019

A : 01 Matrix

We can always satisfy the conditions by constructing a grid as follows:



Sample Code

B : Sorting a Segment

Let us say i and j are *equivalent* when sorting the interval $[i, i+K)$ and sorting the interval $[j, j+K)$ $(i < j)$ give the same result. Let us consider when i and j will be equivalent.

One case where i and j will be equivalent is the case where the intervals $[i, i+K)$ and $[j, j+K)$ are both already arranged in ascending order.

Let us consider the other cases. First, it is obviously necessary that $[i, i+K)$ and $[j, j+K)$ intersect. Thus, $j < i+K$ is a necessary condition. We also see that sorting the interval $[i, i+k)$ does not change the interval $[i, j)$. This means that the interval $[i, j)$ is already arranged in ascending order, and every element in this interval is smaller than every element in $[j, i+k)$. Similarly, the interval $[i+K, j+K)$ is also already arranged in ascending order, and every element in this interval is larger than every element in $[j, i+k)$.

We then see that, if i and j are equivalent, for any x $(i < x < j)$, x is equivalent to i and j. Particularly, i and $i+1$ are equivalent.

The answer to this problem is the number of connected segments in the graph with vertices $0, 1, \cdots, N-K$ where there is an edge between i and j if and only if i and j are equivalent. From the observation above, we only need to consider edges between i and $i+1$ for the case where $[i, i+K)$ are not initially arranged in ascending order. We can determine whether i and $i+1$ are equivalent by checking if the minimum and maximum elements in $[i, i+K+1)$ are P_i and P_i, P_{i+K}, respectively.

Thus, by applying sliding window minimum algorithm, we can determine whether i and $i+1$ are equivalent for all i in $O(N)$ time. The other computations can also be done in $O(N)$, so the problem can be solved in a total of $O(N)$ time. (With std::set, the implementation will be easier but the complexity becomes $O(NlogN)$, though this is still fast enough.)

Sample Code

C : LCMs

Let $V(= 1000000)$ be the maximum possible value of a given integer. Consider a sequence of rational numbers $w_1, w_2, \cdots, w_V$ that satisfies the following condition:

- For every $1 \leq i \leq V$, $\sum_{d|i} w_d = 1/i$.

We can determine these numbers in ascending order of i, that is, when we have already determined $w_1, w_2, \cdots, w_{k-1}$, we have $w_k = 1/k - \sum_{d|i, d<i} w_d$. The complexity of finding these numbers has the same order as that of the total number of divisors of $1, 2, \cdots, V$, that is, $O(VlogV)$.

We know that $lcm(x, y) = xy/gcd(x, y)$, which can be transformed as follows:

$$xy/gcd(x,y) = xy \times \left(\sum_{d|gcd(x,y)} w_d \right) = xy \times \left(\sum_{d|x, d|y} w_d \right)$$

Thus, the sum in question is equal to:

$$\sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} lcm(A_i, A_j) = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} A_i A_j \times \left(\sum_{d|A_i, d|A_j} w_d \right)$$

Let us focus on d here and rewrite the formula. Then, we have:

$$\sum_{1 \leq d \leq V} w_d \left(\sum_{d|A_i, d|A_j, i<j} A_i A_j \right)$$

Now, focus on the part $\sum_{d|A_i, d|A_j, i<j} A_i A_j$. This is the sum of the products of all possible pairs of two distinct divisors of d. We can compute it in O(The number of divisors of d not greater than V) time by pre-calculating the number of i such that $A_i = x$ for each x $(1 \leq x \leq V)$. By doing this for all d, the total complexity will be $O(VlogV)$.

Therefore, the problem is solved in $O(N + VlogV)$ time.

Sample Code

D : Unique Path

If $M = N - 1$, Snuke got a tree from his mother. In that case, there is one simple path between every pair of vertices. Below, we will consider the case where $M \geq N$.

Let us call the graph Snuke received G, and consider the bridges in G.

Let H be the graph obtained from G by removing all the edges except the bridges. First, a pair of vertices a, b with only one simple path between them must belong to the same connected component in H. (If not, there exists a a, b path in G that traverses non-bridge edges. Here, if we remove a non-bridge edge on that path, G will be still connected, so there exists another a, b path, and contradiction is reached.) Also, a pair of vertices a, b with two or more simple paths between them must belong to different connected components in H. (If not, we can choose an edge in the connected component containing a and b so that a and b will be connected, and contradiction is reached.)

Therefore, we see the following: Let I be the graph where an edge is only spun between pairs of vertices with only one simple path between them. Then, a pair of vertices with two or more simple paths between them must belong to different connected components in I.

Let us fix the number of connected components in H and call it k. In G, consider edges spun between different connected components x and y of H. First, a pair of x and y must not have two or more edges spun between them. Thus, the maximum possible number of edges in G is (the number of edges in H) $+ k(k-1)/2 = N - k + k(k-1)/2 = N + k(k-3)/2$. Also, we can see that the minimum possible number of edges in G is N. (We can choose one vertex from each connected component of H and make a cycle connecting these vertices.) Additionally, for every $N \leq i \leq N + k(k-3)/2$, we can obviously have exactly i edges in G.

Thus, k, that is, the number of connected components in H should be as large as possible, and we can see that it should be the number of connected components of I itself. Now, we only need to determine if $M \leq N + k(K-3)/2$.

Sample Code

E : Gachapon

Let $A'_i = A_i/S$.

After applying inclusion-exclusion principle, we need to solve the following problem:

> For all non-empty subset $s = \{i_0, i_1, \cdots, i_{|s|-1}\}$ of $\{0, 1, \cdots, N-1\}$, solve the following problem, and find the sum of those answered multiplied by -1 or 1 depending on the parity of $|s|$.
>
> - Find the expected number of turns in which, for every j $(0 \leq j \leq |s|-1)$, i_j has appeared less than B_{i_j} times.

Let us consider the case where the subset s is fixed.

For an integer sequence $x_0, x_1, \cdots, x_{|s|-1}$ $(0 \leq x_j < B_{i_j})$, we will represent the state where i_j has appeared exactly x_j times by state x. Consider the expected number of turns until state x is reached. Let P be the probability that the generator generates a number contained in $|s|$. After state x is reached for the first time, the expected number of turns it will last is $1/P$. Thus, we just need to compute the probability that state x is reached, which can be found by the following formula (here, $X = \sum_{i=0}^{|s|-1} x_i$):

$$X! \prod_{j=0}^{|s|-1} \left(\frac{A'_{i_j}}{P}\right)^{x_j} \frac{1}{x_j!}$$

We want to find the sum of the value above over all possible integer sequence x. This can be done by DP. (The "keys" in DP should be the number of elements we have already decided from the beginning, and the sum of those elements).

Finally, consider doing the above computation for all possible $|s|$. This can be done by adding a new key to the above DP, which is the sum of A_i for all i that we have decided to add to s.

The total complexity of the DP is $O\left(\left(\sum_{i=0}^{N-1} A_i\right)\left(\sum_{i=0}^{N-1} B_i\right)^2\right)$, which is fast enough.

Sample Code

F : Two Permutations

Consider a graph with edges $i \to P_i$. This graph consists of some number of cycles. When we make A, for each cycle, we have two options: have $A_i = i$ for all i contained in the cycle, or have $A_i = P_i$ for all i. Let us make the first option correspond to allocating 1 to the cycle, and the second option correspond to allocating 0 to the cycle. Also, let $x(i)$ denote the cycle that contains Vertex i in this graph.

Consider another graph with edges $i \to Q_i$. Similarly to the above, we will make the process of making B correspond to allocating 0 and 1 to the cycle. This time, however, 0 corresponds to $B_i = i$ and 1 corresponds to $B_i = Q_i$. Also, let $y(i)$ denote the cycle that contains Vertex i in this graph.

For a index i, whether $A_i = B_i$ is determined by the value allocated to $x(i)$ and $y(i)$, more specifically, determined as follows ($v(c)$ means the value allocated to cycle c):

- If $P_i = Q_i = i$: always $A_i = B_i$
- If none of the above applies and $P_i = i$: $A_i = B_i \Leftrightarrow v(y(i)) = 0$
- If none of the above applies and $Q_i = i$: $A_i = B_i \Leftrightarrow v(x(i)) = 1$
- If none of the above applies and $P_i \neq Q_i$: $A_i = B_i \Leftrightarrow v(x(i)) = 1$ *and* $v(y(i)) = 0$
- If none of the above applies: $A_i = B_i \Leftrightarrow (v(x(i)) = 1$ *and* $v(y(i)) = 0)$ *or* $(v(x(i)) = 0$ *and* $v(y(i)) = 1)$

This can be formulated as project selection problem. Now, we need to find the maximum flow in a graph with $O(N)$ vertices and $O(N)$ edges, each with capacity 1. With Dinic's algorithm, this can be done in $O(N\sqrt{N})$ time, which is fast enough.

Sample Code