AtCoder Grand Contest 019 解説

writer: tourist

2017 年 8 月 26 日

A: Ice Cream Store

必要なリットル数、N が整数であることに注意します。

もし 0.25 リットルサイズを奇数個買ってしまうと、買う量の合計を 0.5 で割った余りが 0.25 になってしまい、合計が整数リットルになりません。よって、0.25 リットルサイズは必ず偶数個買うべきです。0.25 リットルサイズを二つずつ組にすることで、0.25 リットルサイズ二つを 0.5 リットルサイズ一つとして扱えます。したがって、$H := min(H, 2 \times Q)$ とすることで、0.25 リットルサイズの存在を忘れてもよくなります。

同様の議論により、0.5 リットルサイズも必ず偶数個買うべきです。同じように、 $S := min(S, 2 \times H)$ とすることで、0.5 リットルサイズの存在を忘れてもよくなります。

これで、1 リットルサイズを一つ H 円、2 リットルサイズを一つ D 円で買えるという状況になりました。以下の二つの場合を考えます。

- $2 \times S \leq D$ のとき、2 リットルサイズを買うのは決して最適ではありません。その代わりに 1 リットルサイズ二つを常により安く買えるからです。したがって、この場合の答えは $N \times S$ です。
- $2 \times S > D$ のとき、2 リットルサイズをできるだけ多く買うべきです。2 リットルサイズを $(N/2)$ 個買うべきことが容易にわかり、$N \mod 2 = 1$ の場合はさらに 1 リットルサイズを一つ買います。この場合の答えは $(N/2) \times D + (N \mod 2) \times S$ です。

まとめると、答えは以下のようになります。

$$(N/2) \times min(8 * Q, 4 * H, 2 * S, D) + (N \mod 2) \times min(4 * Q, 2 * H, S).$$

B: Reverse and Compare

$A_i = A_j$ であるような i, j を選んだとします。すると、部分文字列 $i+1..j-1$ を反転した結果は、部分文字列 $i..j$ を反転した結果と同じです。これにより、$A_i = A_j$ であるような組 (i, j) は答えに影響しないため無視することができます。

後ほど、$A_i \neq A_j$ であるような組 $i < j$ のそれぞれから、A のうち部分文字列 $i..j$ を反転することで異なる文字列が得られることを証明します。

答えは ($i < j$ かつ $A_i \neq A_j$ であるような組 (i, j) の個数) $+1$ です。この組の個数を数えるには、すべての組の個数 $n \cdot (n-1)/2$ から $i < j$ かつ $A_i = A_j$ であるような組の個数を引けばよいです。同じ文字の組の個数は、“a” から “z” までのそれぞれの文字 c について $z(c) \cdot (z(c)-1)/2$ を足し合わせた和になります。ここで、$z(c)$ は A に c が出現する回数です。

この解法の計算量は $O(n)$ となります。

上記の証明:

$B(i, j)$ を A のうち部分文字列 $i..j$ を反転して得られる文字列とします。ここで $A_i \neq A_j$ と仮定します。すると、$B(i, j)$ と A で文字が異なる最も左の位置は i です。同様に、$B(i, j)$ と A で文字が異なる最も右の位置は j です。

今度は、$B(i, j) = B(x, y)$ と仮定します。$B(i, j)$ と $B(x, y)$ の A と文字が異なる最も左の位置は同じでなければならないため、$i = x$ を得ます。同様に、$B(i, j)$ と $B(x, y)$ の A と文字が異なる最も右の位置は同じでなければならないため、$j = y$ を得ます。よって、$B(i, j) = B(x, y)$ であれば $i = x$ かつ $j = y$ が成立します。この対偶を取ると、$i \neq x$ または $j \neq y$ であれば $B(i, j) \neq B(x, y)$ となります。ゆえに、$A_i \neq A_j$ であるような部分文字列 $i..j$ を反転して得られる文字列はすべて異なります。

C: Fountain Walk

Consider a path from (x_1, y_1) to (x_2, y_2). Let s be the number of sections it passes through (when we ignore foutains), p be the number of 90-degree turnings at fountains, and q be the number of 180-degree turnings at fountains. For example, in the three samples in the statement, $(s, p, q) = (9, 2, 0), (3, 0, 0), (2, 0, 1)$. The length of this path is:

$$100s - (20 - 5\pi)p + (10\pi - 20)q \tag{1}$$

Thus, we want s and q to be small and p to be big. Since the coefficient of s is by far the largest, we must minimize it (we will give a formal proof later). Without loss of generality, we can assume that $x_1 \leq x_2, y_1 \leq y_2$. Then, $s = (x_2 - x_1) + (y_2 - y_1)$.

What is the largest possible number of fountains we can pass through while minimizing s? Let's call this value k, and suppose that the fountains are at $(a_1, b_1), \ldots, (a_k, b_k)$, in order. This sequence can be a part of the path if and only if the following conditions are satisfied:

- $x_1 \leq a_1 < \cdots < a_k \leq x_2$
- $y_1 \leq b_1 < \cdots < b_k \leq y_2$

k is the longest possible sequence of fountains satisfying these constraints, and it can be computed as a LIS (Longest Increasing Sequence).

Now, ideally, we want to make $p = k$ and $q = 0$. It turns out to be possible except for the special case $k = \min\{x_2 - x_1, y_2 - y_1\} + 1$. (Note that $k \leq \min\{x_2 - x_1, y_2 - y_1\} + 1$ because each street/avenue contains at most one fountain.)

The case with $k < \min\{x_2 - x_1, y_2 - y_1\} + 1$

The case with $k = \min\{x_2 - x_1, y_2 - y_1\} + 1$

D: Shift and Flip

B が 0 のみからなるとき、A のビットを反転することはできません。よって、A も 0 のみからなるときの答えは 0、それ以外の場合の答えは -1 です。以降、B に 1 が一つ以上含まれるとします。

次のような「カウンター」があると想定します。このカウンターの初期値は 0 であり、A を右にシフトするたびに値が 1 増え、左にシフトするたびに値が 1 減ります。ここで、l を操作列の実行中にカウンターが記録した最小の値とします。同様に、r を操作列の実行中にカウンターが記録した最大の値とします。

l, r, d の値が既知であるとします。次の問いを考えましょう:「A を B に変換する操作列は存在するか？存在するなら、そのような最短の操作列の長さは何か？」

A の個別のビット一つ (A_i とします) を考えます。最終的に、このビットは位置 $(i+d) \mod n$ に移動します。よって、$A_i = B_{(i+d) \mod n}$ であれば A_i を反転させる必要はなく、そうでなければどこかのタイミングで反転させる必要があります。操作列の実行中に、ビット A_i は $(i+l) \mod n$ から $(i+r) \mod n$ のすべての位置（両端含む）を訪れます。もし $B_j = 1$ であるような j が存在すれば、A_i が j を訪れたときに反転させることができます。そうでなければ A_i を反転させることはできず、条件を満たす操作列は存在しません。

もし l, r, d の値に対応する操作列が存在することが分かれば、そのような最短の操作列の長さは容易に計算できます。その長さは、反転の回数とシフトの回数の和です。必要な反転の回数は当然、$A_i \neq B_{(i+d) \mod n}$ であるような添字 i の個数です。また、必要なシフトの回数が $2 \cdot (r-l) - |d|$ であることは容易に示せます。例えば、$d \geq 0$ であれば、左に $-l$ 回シフトし、右に $r-l$ 回シフトして左に $r-d$ 回シフトするのが最適です。

ここで、最適解において $|l|, |r|, |d| \leq n$ であることに注意します。実際、例えば右に n 回シフトを行うと、A のすべてのビットがすべての場所を訪れたことになり、また B には少なくとも一つ 1 が含まれるため、A のすべてのビットを反転させる機会がすでにあったのであり、それ以上 A を右にシフトする必要はありません。

簡単な前計算を行うと、これで $O(n^3)$ の解法が得られます。

これを最適化するには、まず d の値を固定します。一般性を失うことなく $d \geq 0$ と仮定します。再び、A の個別のビット一つ (A_i とします) を考えます。もし $A_i = B_{(i+d) \mod n}$ であれば、問題ありません。そうでなければ、A_i を反転する必要があります。もし i と $(i+d) \mod n$ の間に $B_j = 1$ であるような j が存在すれば、A_i が位置 j に来たところで反転させることができるので、他に何もする必要はありません。そうでなければ、A_i が $B_j = 1$ であるような位置 j に来るように A をシフトする必要があります。ここで、x_i を B での位置 i より左側の最も近い 1 までの距離とし、y_i を B での位置 $(i+d) \mod n$ より右側の最も近い 1 までの距離とします。また、$X = -l, Y = r-d$ とします。すると、$X \geq x_i$ と $Y \geq y_i$ のうち少なくとも一方が成立する必要があります。

このようにして複数の組 (x_i, y_i) が得られ、次の部分問題を解くことになります。「条件 ($X \geq x_i$ または $Y \geq y_i$) のもとで、$X+Y$ を最小化する $X \geq 0, Y \geq 0$ を求めよ。」組を x_i の値でソートすると、この問題は単純な平面走査で解けます。これを d の $O(n)$ 個の値について行う必要があるため、全体で $O(n^2 \log n)$ の解法が得られます。ここで、x_i の値が n であることを利用して組のソートを $O(n)$ で行うと $O(n^2)$ の解法となります。

E: Shuffle and Swap

a と b を個別にシャッフルする代わりに、次の同等な手順を考えます。

- まず、a_i と b_i の対応を固定します。対応は $k!$ 通り存在します。
- そして、組 (a_i, b_i) の順序を決定します。順序は $k!$ 通り存在します。

第一ステップがすでに済んだものとして、$A = B$ となるような順序の個数を数えます。n 頂点からなる有向グラフを考えます。各 i に対し、a_i から b_i に辺を描きます。各頂点 i の出次数は A_i, 入次数は B_i です。$A_i = B_i = 0$ であるような i（孤立点）を無視すると、このグラフの各連結成分はパスまたはサイクルです。

サイクルに含まれる辺の順序は関係ありません。それらの辺は 1 と 1 を入れ替えるのに対応しているためです。

しかし、任意のパス $p_1 - > p_2 - > \ldots - > p_t$ については、適切な辺の順序はただ一つです。はじめ、$A_{p_1} = A_{p_2} = \ldots = A_{p_{t-1}} = 1, A_{p_t} = 0$ です。また、$B_{p_1} = 0, B_{p_2} = B_{p_3} = \ldots = B_{p_t} = 1$ です。この状態から、0 を位置 p_t から p_1 に移動させる必要があり、そのような方法はただ一つであることは容易にわかります: まず $A_{p_{t-1}}$ と A_{p_t} を入れ替え（0 を p_{t-1} に移動)、次に $A_{p_{t-2}}$ と $A_{p_{t-1}}$ を入れ替え（0 を p_{t-2} に移動)、…、最後に A_{p_1} と A_{p_2} を入れ替え（0 を p_1 に移動)。

$A_i = B_i = 1$ であるような添字 i の個数を m、$A_i = 1$, $B_i = 0$ であるような i の個数を e とします。A と B に含まれる 1 の個数は等しいため、$A_i = 0$, $B_i = 1$ であるような i の個数も e となります。明らかに、グラフにはちょうど e 本のパス（と何個かのサイクル）があります。グラフの辺の本数は $m + e$ であり、パスまたはサイクルのいずれかに含まれる頂点が m 個あります。

このとき、m 個の頂点を e 本のパスに配分する方法の数を数え、同時に問題文の条件を満たす辺の順序の数を数えるのは、比較的単純な組合せの練習問題のようなものです。定数倍を除くと、求めたいものは次の関数です。

$$f(0,0) = 1,$$
$$f(0,j) = 0 \text{ for } j > 0,$$
$$f(i,j) = \sum_{u=0}^{j} \frac{f(i-1,j-u)}{(u+1)!}.$$

ここで、$f(i,j$ はいずれか j 個の頂点を最初の i 本のパスに配分するのに対応します。

すると、問題の答えは 0 から m までの j に対して $f(e,j)$ を足し合わせた和を $e! \cdot m! \cdot (e+m)!$ 倍したものであることが示せます。これにより、$O(n^3)$ の解法が得られます。

この解法を二段階で最適化することで満点解法が得られます。第一段階は、多項式 $p_i(x) = \sum_{j=0}^{m} f(i,j) \cdot x^j$ を考えることです。すると、$p_i(x)$ から $p_{i+1}(x)$ を高速フーリエ変換により $O(n \log n)$ 時間で計算することができます。これにより、$O(n^2 \log n)$ の解法が得られます。

第二段階は次のようになります。$q(x) = \sum_{j=0}^{m} \frac{x^j}{(j+1)!}$ とします。すると、$p_{i+1}(x) = p_i(x) \cdot q(x)$ （の最初の $m+1$ 項）となります。これと $p_0(x) = 1$ より、$p_e(x) = q(x)^e$ が成立します。よって、繰り返し二乗法により、$O(\log n)$ 回の多項式の掛け算で $p_e(x)$ を計算できます。これで、$O(n \log^2 n)$ の解法が得られました。

F: Yes or No

AtCoder Grand Contest 019 Editorial

writer: tourist

August 26, 2017

A: Ice Cream Store

Note that N, the required number of liters, is integer.

If we buy an odd number of 0.25-liter bottles, we will never get an integer number of liters — remainder of division of our bought volume by 0.5 will be 0.25. Therefore, we must always buy an even number of 0.25-liter bottles. We can arrange 0.25-liter bottles in pairs and consider two 0.25-liter bottles as one 0.5-liter bottle. Thus, we can set $H := min(H, 2 \times Q)$ and forget about 0.25-liter bottles completely.

Now, by a similar argument, we must buy an even number of 0.5-liter bottles. Similarly, we can set $S := min(S, 2 \times H)$ and forget about 0.5-liter bottles as well.

Now we're in a situation where we can only buy 1-liter bottles for S yen each, and 2-liter bottles for D yen each. There are two cases:

- If $2 \times S \leq D$, it's never optimal to buy 2-liter bottles — we can always buy two 1-liter bottles at a better price instead. Therefore, the answer is $N \times S$.
- If $2 \times S > D$, we should buy as many 2-liter bottles as possible. It's easy to see we should buy (N div 2) 2-liter bottles, and if $N \mod 2 = 1$, we should buy an additional 1-liter bottle. The answer is $(N \text{ div } 2) \times D + (N \mod 2) \times S$.

Overall, the answer is:

$$(N \text{ div } 2) \times min(8 * Q, 4 * H, 2 * S, D) + (N \mod 2) \times min(4 * Q, 2 * H, S).$$

B: Reverse and Compare

Suppose we choose such i and j that $A_i = A_j$. Then, reversing substring $i+1..j-1$ results in the same string as reversing substring $i..j$. This means we can safely ignore pairs (i, j) with $A_i = A_j$, since they don't influence the result.

Later, we'll prove that for every pair $i < j$ such that $A_i \neq A_j$, we'll obtain a unique string by reversing substring $i..j$ in A.

The answer is 1 plus the number of pairs (i, j) such that $i < j$ and $A_i \neq A_j$. To count these pairs, we can subtract the number of pairs such that $i < j$ and $A_i = A_j$ from the total number of pairs, $n \cdot (n-1)/2$. The number of pairs of equal characters is the sum of $z(c) \cdot (z(c)-1)/2$ over all characters c from "a" to "z", where $z(c)$ is the number of occurrences of c in A.

The complexity of this solution is $O(n)$.

Proof:

Let $B(i, j)$ be the string obtained by reversing substring $i..j$ in A. Assume $A_i \neq A_j$. Then, the leftmost position where $B(i, j)$ and A differ is position i. Similarly, the rightmost position where $B(i, j)$ and A differ is position j.

Now assume that $B(i, j) = B(x, y)$. Since the leftmost position where $B(i, j)$ and $B(x, y)$ differ from A must be the same, we get that $i = x$. Similarly, the rightmost position where $B(i, j)$ and $B(x, y)$ differ from A must be the same, so we get $j = y$. Thus, if $B(i, j) = B(x, y)$, then $i = x$ and $j = y$. The negation of this statement is: if $i \neq x$ or $j \neq y$, then $B(i, j) \neq B(x, y)$. It follows that all strings obtained by reversing substring $i..j$ where $A_i \neq A_j$ are unique.

C: Fountain Walk

Consider a path from (x_1, y_1) to (x_2, y_2). Let s be the number of sections it passes through (when we ignore foutains), p be the number of 90-degree turnings at fountains, and q be the number of 180-degree turnings at fountains. For example, in the three samples in the statement, $(s, p, q) = (9, 2, 0), (3, 0, 0), (2, 0, 1)$. The length of this path is:

$$100s - (20 - 5\pi)p + (10\pi - 20)q \tag{1}$$

Thus, we want s and q to be small and p to be big. Since the coefficient of s is by far the largest, we must minimize it (we will give a formal proof later). Without loss of generality, we can assume that $x_1 \leq x_2, y_1 \leq y_2$. Then, $s = (x_2 - x_1) + (y_2 - y_1)$.

What is the largest possible number of fountains we can pass through while minimizing s? Let's call this value k, and suppose that the fountains are at $(a_1, b_1), \ldots, (a_k, b_k)$, in order. This sequence can be a part of the path if and only if the following conditions are satisfied:

- $x_1 \leq a_1 < \cdots < a_k \leq x_2$
- $y_1 \leq b_1 < \cdots < b_k \leq y_2$

k is the longest possible sequence of fountains satisfying these constraints, and it can be computed as a LIS (Longest Increasing Sequence).

Now, ideally, we want to make $p = k$ and $q = 0$. It turns out to be possible except for the special case $k = \min\{x_2 - x_1, y_2 - y_1\} + 1$. (Note that $k \leq \min\{x_2 - x_1, y_2 - y_1\} + 1$ because each street/avenue contains at most one fountain.)

The case with $k < \min\{x_2 - x_1, y_2 - y_1\} + 1$

Let $(a_1, b_1), \ldots, (a_k, b_k)$ be one possible sequence of fountains of length k. We can assume that $a_1 > x_1$ (otherwise, $b_1 > y_1$ and the same claim holds). Because of the maximality of k, when we passes through all of them, we don't pass through any other fountains, so we can ignore others.

First, we choose y such that $y_1 \leq y \leq y_2$ and the street y contains no fountains. (As a special case, when $a_k = x_2$, we must choose $y = y_2$.) Then, we enter/leave fountains according to the following rule:

- If $b_i < y$, we enter (a_i, b_i) from $(a_i - 1, b_i)$ and leaves to $(a_i, b_i + 1)$.
- If $b_i > y$, we enter (a_i, b_i) from $(a_i, b_i - 1)$ and leaves to $(a_i + 1, b_i)$.

This way, we can find a path with $p = k$ and $q = 0$.

The case with $k = \min\{x_2 - x_1, y_2 - y_1\} + 1$

We can assume that $y_2 - y_1 \leq x_2 - x_1$. Then, in this case, each street between y_1 and y_2, inclusive, contains one fountain. These fountains splits the rectangle $[x_1, x_2] \times [y_1, y_2]$ into two parts and it is necessary to use one 180-degree turning to go from one part to the other.

On the other hand, $(p, q) = (k - 1, 1)$ is possible: choose arbitrary $k - 1$ fountains and do the same as the previous case.

Thus, in this case, the optimal path has parameters $p = k - 1$ and $q = 1$.

Proof that s should be the smallest

First, we'll prove that the path never goes out of the bounding box $[x1, x2] \times [y1, y2]$.

Suppose that the path goes below the bounding box. Let $y_0 < y_1$ be the minimum y-coordinate of the path. Then, at some point, the path goes from $y_0 + 1$ to y_0, runs along the line $y = y_0$, and return to $y_0 + 1$.

We can change the path by changing y-coordinates of all parts of the path at $y = y_0$ to $y_0 + 1$. This way, s decreases by 2, p decreases by at most 1, and q increases by at most 1, thus we get a shorter path. Thus, we should never go out of the bounding box.

Now, we know that the path goes to $(x_1 + 1, y_1)$ or $(x_1, y_1 + 1)$ from (x_1, y_1). The remaining part of path should be the shortest path from that position to (x_2, y_2), so the same bounding-box argument will prove that s should be smallest possible. (Strictly speaking, we shuold carefully handle the case when the new intersection contains a fountain, but the proof is simlar and we omit details).

D: Shift and Flip

If B consists only of 0's, we can't flip any bits in A. Thus, if A also consists only of 0's, the answer is 0, otherwise it's -1. From now on, assume B contains at least one 1.

Consider a counter which is initialized with zero, increases by 1 whenever we shift A to the right, and decreases by 1 whenever we shift A to the left. Let l be the smallest value ever contained by the counter during executing some sequence of operations. Similarly, let r be the largest value ever contained by the counter, and let d be the resulting value of the counter.

Suppose we know l, r and d. Let's answer the following question: does there exist a sequence of operations which transforms A into B, and if yes, what is the length of the shortest such sequence?

Consider an individual bit in A, say A_i. In the end, it will move to position $(i+d) \mod n$. Then, if $A_i = B_{(i+d) \mod n}$, we don't need to flip A_i. Otherwise, we have to flip it at some moment. During the process, bit A_i visited all positions from $(i+l) \mod n$ to $(i+r) \mod n$, inclusive. If there exists j in this range such that $B_j = 1$, we can flip A_i at the moment it visits position j. Otherwise, we can't flip A_i, so there is no valid sequence of operations.

If we found out that a sequence of operations with corresponding l, r and d exists, we can easily calculate the length of the shortest such sequence. This length is equal to the number of flips plus the number of shifts. We know the required number of flips — it's the number of indices i such that $A_i \neq B_{(i+d) \mod n}$. It's easy to show that the required number of shifts is $2 \cdot (r-l) - |d|$. For example, if $d \geq 0$, it's optimal to make $-l$ shifts to the left, then $r-l$ shifts to the right, and then $r-d$ shifts to the left.

Note that in the optimal solution $|l|, |r|, |d| \leq n$. Indeed, once we make n shifts to the right (for example), it means that every bit in A has been to every single position, and since B contains at least one 1, we've already had a chance to flip every bit in A, so we don't need to shift A to the right anymore.

With some easy preprocessing, this gives an $O(n^3)$ solution.

To optimize it, let's fix the value of d first. Without loss of generality, assume $d \geq 0$. Again, consider an individual bit in A, say A_i. If $A_i = B_{(i+d) \mod n}$, we're fine. Otherwise, we need to flip A_i. If there exists j between i and $(i+d) \mod n$ such that $B_j = 1$, we can flip A_i at the moment it gets to position j, so we don't need to do anything else. Otherwise, we have to perform shifts on A in such a way that A_i gets into some position j with $B_j = 1$. Let x_i be the distance to the closest 1 in B to the left of position i, and y_i be the distance to the closest 1 in B to the right of position $(i+d) \mod n$. Also, put $X = -l$ and $Y = r - d$. Then, we need at least one of $X \geq x_i$ and $Y \geq y_i$ to be true.

Finally, we have several pairs (x_i, y_i), and we need to solve the following subproblem: find $X \geq 0$ and $Y \geq 0$ subject to ($X \geq x_i$ or $Y \geq y_i$) minimizing $X+Y$. After we sort all pairs by x_i, this subproblem can be solved with a simple linear scan. Since we have to do that for $O(n)$ values of d, overall we have an $O(n^2 \log n)$ solution due to sorting, or we can arrive at an $O(n^2)$ solution if we use the fact that x_i doesn't exceed n, so we can sort pairs in $O(n)$.

E: Shuffle and Swap

Instead of shuffling a and b independently, let's perform an equivalent procedure:

- First, fix the matching of a_i and b_i. There are $k!$ matchings.
- Then, fix the order of pairs (a_i, b_i). There are $k!$ orders.

Suppose we've done the first step and want to count the number of orders which result in $A = B$. Consider a directed graph with n vertices. Draw an edge from a_i to b_i for every i. Every vertex i has outgoing degree A_i and ingoing degree B_i. After ignoring i with $A_i = B_i = 0$ (which are isolated vertices), every connected component in this graph is either a path or a cycle.

The order of edges in cycles doesn't matter, since every cycle edge corresponds to swapping 1 with 1.

For any path $p_1 -> p_2 -> \ldots -> p_t$, though, there is only one good order of edges. Initially, $A_{p_1} = A_{p_2} = \ldots = A_{p_{t-1}} = 1$ and $A_{p_t} = 0$. Also, $B_{p_1} = 0$ and $B_{p_2} = B_{p_3} = \ldots = B_{p_t} = 1$. We need to move 0 from position p_t to position p_1, and it's easy to see that there is exactly one way to do that: swap $A_{p_{t-1}}$ with A_{p_t} (moving 0 to position p_{t-1}), then swap $A_{p_{t-2}}$ with $A_{p_{t-1}}$ (moving 0 to position p_{t-2}), ..., then swap A_{p_1} with A_{p_2} (moving 0 to position p_1).

Denote the number of indices i with $A_i = B_i = 1$ by m, and the number of indices i with $A_i = 1$, $B_i = 0$ by e. Since the number of 1's in A and B is equal, the number of indices i with $A_i = 0$, $B_i = 1$ is also e. Clearly, we have exactly e paths in our graph (along with some cycles). The number of edges in the graph is $m + e$, and there are m vertices which might belong either to a path or to a cycle.

Now it's a relatively simple combinatorial exercise to count the number of ways to distribute some of m vertices into e paths, while also counting the number of orders of edges which comply with the problem statement. It can be shown that, up to a constant, we're interested in the following function:

$$f(0,0) = 1,$$
$$f(0,j) = 0 \text{ for } j > 0,$$
$$f(i,j) = \sum_{u=0}^{j} \frac{f(i-1,j-u)}{(u+1)!}.$$

Here, $f(i,j)$ corresponds to distributing some j vertices into the first i paths.

Then, it can be shown the answer to the problem is the sum of $f(e,j)$ over j from 0 to m, multiplied by $e! \cdot m! \cdot (e+m)!$. This gives an $O(n^3)$ solution.

It can be optimized to full solution in two steps. The first step is to consider polynomials $p_i(x) = \sum_{j=0}^{m} f(i,j) \cdot x^j$. Then, we can calculate $p_{i+1}(x)$ from $p_i(x)$ in $O(n \log n)$ using Fast Fourier Transform. This gives an $O(n^2 \log n)$ solution.

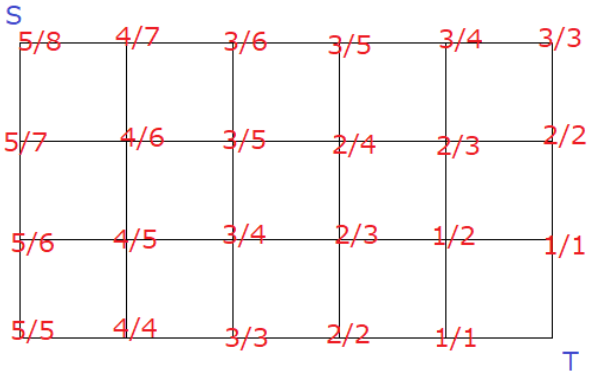
Here is the second optimization. Let $q(x) = \sum_{j=0}^{m} \frac{x^j}{(j+1)!}$. Then, $p_{i+1}(x) = p_i(x) \cdot q(x)$ (cut to the first $m+1$ coefficients). Since $p_0(x) = 1$, it follows that

$p_e(x) = q(x)^e$. We can calculate $p_e(x)$ in $O(\log n)$ polynomial multiplications using exponentiation by squaring. Finally, this gives an $O(n \log^2 n)$ solution.

F: Yes or No

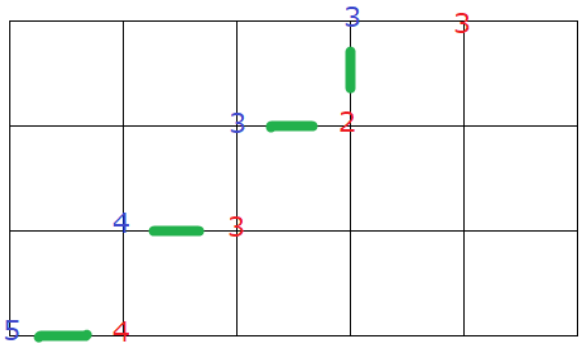
Obviously, in the optimal strategy, if you know that there are A remaining questions with "Yes" and B remaining questions with "No", you should answer "Yes" if $A \geq B$, and "No" otherwise. Thus, in such situation, you can correctly answer the next question with probability A/B.

Let's represent a sequence of "Yes" and "No" as a shortest path from the top-left corner to the bottom-right corner in an $N \times M$ grid.



In the picture above, what we want to compute is the expected sum of numbers on a random path from S to T.

We will handle each diagonal independently. For example, the following picture shows two diagonals:

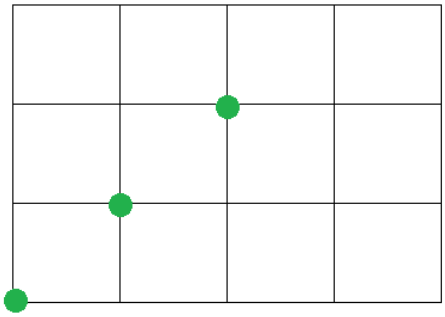


Each path contains exactly one blue number. We should compute the sum of this number over all paths, divide it by 5 (the common denominator) and $(N+M)!/N!M!$ (the number of paths), and add it to the answer. We should do this for all paths.

Let's handle diagonals in the order from bottom-right to top-left. Suppose that we've already computed the sum of red numbers over all paths. How should we compute the sum of blue numbers then?

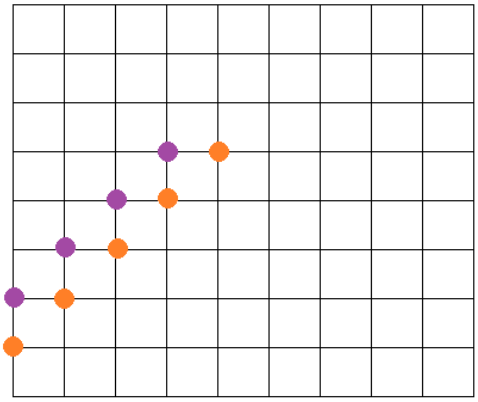
Notice that in any path, (the blue number) minus (the red number) is either zero or one, and it is one when the path passes through a green edge. Thus, (the sum of blue numbers) = (the sum of red numbers) + (the number of paths with a green edge).

We handle horizontal green edges and vertical green edges separately. The number of paths with a horizontal green edge is equal to the number of paths with a green vertex in the following grid (Note that the grid is compressed by one horizontally):



This is hard to compute quickly. However, we can again use the result from previous diagonals.

For the ease of understanding, here's a bigger example:



In this diaglam, we want to compute the number of paths with a purple vertex. However, we already know the number of paths with an orange vertex (during the computation of previous diagonal). Since the difference between those two numbers can be represented as a simple binary coefficient, it can be computed in O(1).