AtCoder Beginner Contest 071 / AtCoder Regular Contest 081 解説

writer: semiexp

2017 年 8 月 20 日

For International Readers: English editorial starts on page 5.

A: Meal Delivery

定義どおりに，すぬけ君の住んでいる位置と，店 A, B の間の距離を求めます．この距離の大小に応じて，より近いほうを出力すればよいです．

- C++ による解答例: `https://abc071.contest.atcoder.jp/submissions/1518940`
- Ruby による解答例: `https://abc071.contest.atcoder.jp/submissions/1518942`
- Rust による解答例: `https://abc071.contest.atcoder.jp/submissions/1516215`

B: Not Found

英小文字 26 種類のそれぞれについて，S の中に現れるかを記録する配列を持っておきます．この配列は最初すべて「現れない」(`false`) で初期化しておきます．S を 1 文字目から順に見ていって，出てきた文字それぞれについて，配列上でその文字に対応する位置を「現れる」(`true`) に更新します．S の最後の文字まで見終わった後には，この配列を見るだけである文字が S の中に現れるかわかるようになります．

最後に，この配列を，`a`, `b`, ..., `z` の順に見ていって，`false` となった最初の文字を出力すればよいです．ただし，`z` まで見終わっても `false` が現れなかった場合は，すべての英小文字が S 中に現れているということなので，`None` を出力します．

- C++ による解答例: `https://abc071.contest.atcoder.jp/submissions/1518956`
- Ruby による解答例: `https://abc071.contest.atcoder.jp/submissions/1518966`
- Rust による解答例: `https://abc071.contest.atcoder.jp/submissions/1516216`

C: Make a Rectangle

$H \times W$ の長方形を作るためには，長さ H, W の棒が 2 本ずつ必要です．ただし，$H = W$ の場合は，長さ H の棒が 4 本必要です．

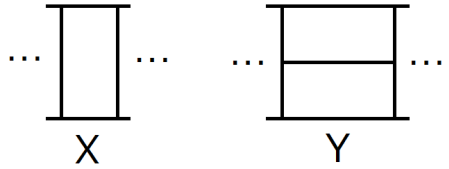
各辺に使う棒は，上の条件のもとで，できるだけ長い棒を貪欲に使えばよいです．$\{A_i\}$ をソートすると，同じ値は連続した区間に集まるため，各長さの棒が何本あるかを知ることができます．なので，長さの長い順に見ていって，次のように辺の長さを決定すればよいです．

- 同じ長さの棒が全部で 4 本以上ある場合: その長さの棒を縦，横両方に使う (すでに片方の辺が決定している場合はそちらを優先)
- 同じ長さの棒が全部で 2 本または 3 本ある場合: その長さの棒を縦，横のうち片方に使う
- 同じ長さの棒が全部で 1 本しかない場合: その長さの棒は使えない

途中で両方の辺が決定した場合は，長方形が作れているので，その長方形の面積 (縦 $\times$ 横) を出力します．最後の辺まで見ても長方形の辺が足りない場合は，長方形を作ることはできないので，0 を出力します．

D: Coloring Dominoes

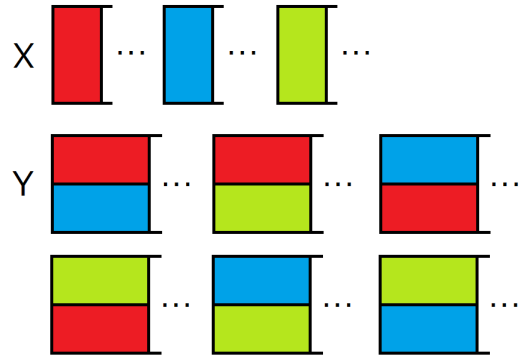
ドミノの並べ方は，縦 2 $\times$ 横 1 のドミノを 1 個並べるか (X) ，縦 1 $\times$ 横 2 のドミノを 2 個縦に並べるか (Y) のどちらかを，横につなげたような並べ方になります．



左から順にドミノを塗ることを考えます．縦に 2 個ドミノが並んでいる箇所については，この 2 個を同時に塗ります．一番左の箇所については，

- X: 3 通り
- Y: 6 通り

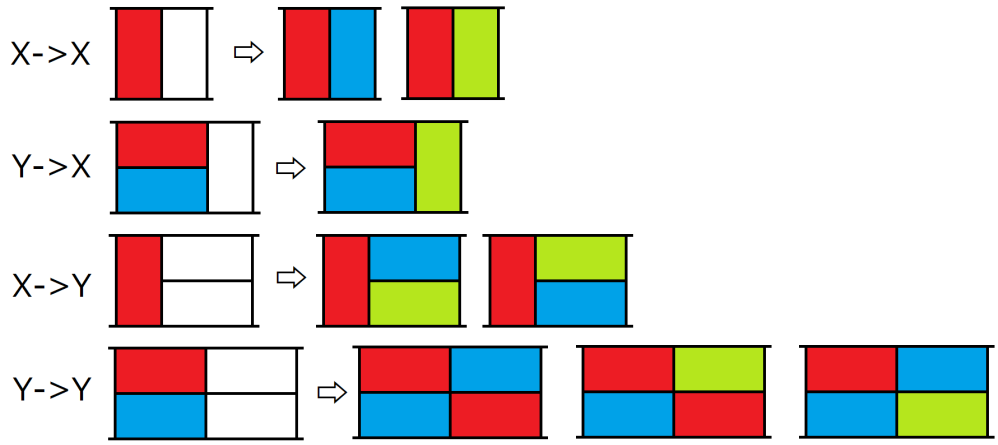
の塗り方があります．



また，それ以外の箇所については，それより左の箇所がすべて塗られているとしたとき，

- X (すぐ左が X): 2 通り
- X (すぐ左が Y): 1 通り
- Y (すぐ左が X): 2 通り
- Y (すぐ左が Y): 3 通り

の塗り方があります．



よって，ドミノの並べ方を判定した上で，左から順に上の規則で並べ方の個数を求めていけばよいです．

E: Don't Be a Subsequence

以下，文字列は英小文字のみからなるもののみを考えます．

文字列 A に対して,「その文字列の suffix であって，すべての英小文字を含むもののうち，最も短いもの」を順次取り除くことを考えます．ただし，そのようなものが存在しなくなったら終了します．この時，suffix を取り除くことができた回数を K とします．また，i 番目に取り除いた suffix を S_{K-1-i}，最後に残った文字列を A' とします．このとき，A は $A', S_1, S_2, \ldots, S_K$ をこの順に並べた文字列になります．

A の部分列でない最短の文字列の長さは $K+1$ になることを示します．まず，長さ K の任意の文字列 B が A の部分列であることを示します．(長さ K 未満の場合は，末尾に適当に文字を加えて長さ K にした場合について示されるので十分です) $S_1, \ldots, S_K$ はいずれもすべての英小文字を含むため，S_i を構成する文字を B の i 文字目に一致する文字 1 個を除いて取り除くことができます．また，A' を構成する文字はすべて取り除きます．すると，S から文字を取り除いて B が得られるため，B は S の部分列です．

次に，A の部分列でない長さ $K+1$ の文字列が存在することを示します．文字列 $C = c_0 c_1 \ldots c_K$ を次で定めます．

- c_0 は，A' に現れない文字のうち 1 つとする (A' からは suffix を取り除けなかったことから，このような文字は必ず存在します)．
- c_i $(i = 1, 2, \ldots, K)$ は，S_i の最初の文字とする．

すると，C は A の部分列ではありません．実際，A を構成する文字をいくつか取り除いて C を得ようとすると，

- c_0 が A の中で初めて現れるのは S_1 の中であるため，A' はすべて取り除かなければなりません．
- c_1 は S_1 の中では 1 文字目にしか現れないため (取り除く suffix の最小性より)，c_0 の次に c_1 を得るためには，S_1 は c_0 を得るために残した文字を除いてすべて取り除かなければなりません．
- ...
- c_K は S_K の中では 1 文字目にしか現れないため，c_{K-1} の次に c_K を得るためには，S_K は c_{K-1} を得るために残した文字を除いてすべて取り除かなければなりません．すると，c_K のために使える文字が A にまったく残らなくなってしまします．

よって，A の部分列でない最短の文字列の長さは $K+1$ になることがわかりました．

上の議論から，suffix を取り除く作業を行うことで，A の任意の suffix についても「部分列でない最短の文字列の長さ」を求めることができます．

すると，1 文字目から貪欲に文字を決定していくことで，辞書順最小の答えを求めることができます．文字 c を使うことができるかどうかは，現在の位置以降の最初の c の現れの位置 p_c を求め，S の p_c 文字目以降についての「部分列でない最短の文字列の長さ」が，現在の位置以降の「部分列でない最短の文字列の長さ」よりちょうど 1 小さくなっているかで判定することができます．p_c は，各文字の現れる位置をソートして持っておくと，二分探索で $O(\log |A|)$ で求めることができます．

各位置に対しては文字種を m として $O(m \log |A|)$ で次の文字を決定することができ，答えの長さは $O(|A|/m)$ なので，以上のアルゴリズムにより，答えを $O(|A| \log |A|)$ で求めることができます．

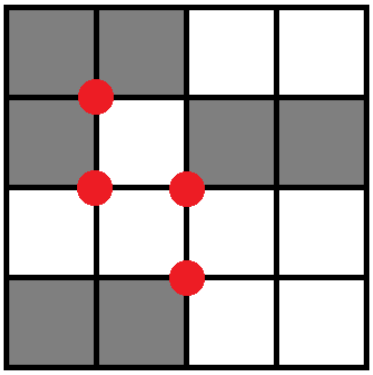
F: Flip and Rectangles

マス目に含まれる 2×2 の部分であって，黒マスを奇数個 (1 個または 3 個) 含むものを「悪い部分」とよぶことにします．うまく操作を行えば長方形 S を選ぶことができることと，S が悪い部分を含まないことは同値です．

実際，S が悪い部分を含むならば，どう操作を行ってもその部分をすべて黒で塗られた状態にすることはできません．

次に S が悪い部分を含まない場合を考えます．このとき，S の一番上の行，一番左の列をすべて黒く塗られた状態にすることができます．ここで，どのような操作を行っても，悪い部分が悪くない部分に変わったり，その逆が起きたりすることはないことに注意します．よって，この状態においても，S は悪い部分を含まないため，S の上から 2 行目，左から 2 列目のマスは黒で塗られています．同様に，上から 2 行目，左から 3 列目，... のマスはすべて黒で塗られていることがわかり，上から 2 行目はすべて黒で塗られていることがわかります．さらに同様にして，上から 3 行目，... のマスもすべて黒で塗られていることがわかり，結果として S 全体が黒で塗られていることがわかります．よって，S 全体を黒で塗られた状態にすることができ，S を選ぶことができることがわかります．

ここで，マス目において．すべての悪い部分の中心の格子点に，印を付けることを考えます．すると，選ぶことができる長方形は，内部に (頂点や辺上にはあってもよい) 印を含まない長方形になります．



この「印を含まない長方形」は，最大長方形のアルゴリズムを用いると $O(HW)$ で求めることができます．

AtCoder Beginner Contest 071 / AtCoder Regular Contest 081 Editorial

writer: semiexp

August 20th, 2017

A: Meal Delivery

- C++ solution: `https://abc071.contest.atcoder.jp/submissions/1518940`
- Ruby solution: `https://abc071.contest.atcoder.jp/submissions/1518942`
- Rust solution: `https://abc071.contest.atcoder.jp/submissions/1516215`

B: Not Found

- C++ solution: `https://abc071.contest.atcoder.jp/submissions/1518956`
- Ruby solution: `https://abc071.contest.atcoder.jp/submissions/1518966`
- Rust solution: `https://abc071.contest.atcoder.jp/submissions/1516216`

C: Make a Rectangle

In order to make an $H \times W$ rectangle, you need two sticks of length H and two sticks of length W. In case $H = W$, you need four sticks of length H.

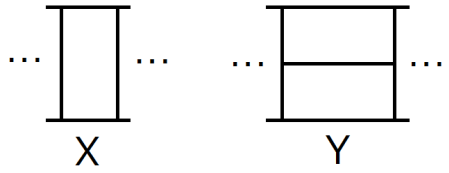
Under the constraints above, you should choose the longest possible sticks greedily. First, you sort $\{A_i\}$ in the decreasing order. Since sticks of the same lengths are grouped together after the sorting, we can count the number of sticks of each length. We check the sticks in the decreasing order of their lengths, and do the following:

- If we haven't chosen any sticks and we find four (or more) sticks of the same length, we choose them and finish the process.
- If we haven't chosen any sticks and we find two or three sticks of the same length, we choose them (and continue the process).
- If we have chosen two sticks and we find two (or more) sticks of the same length, choose two of them and finish the process.
- Otherwise, skip sticks of current length.

If we successfully choose four sticks after this process, print the area of the rectangle. Otherwise, print 0 instead.

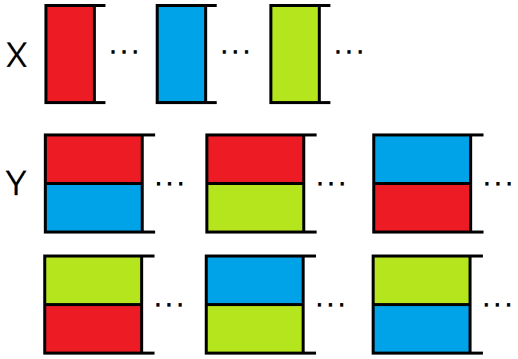
D: Coloring Dominoes

The arrangement of dominoes is a concatenation of the following two patterns:



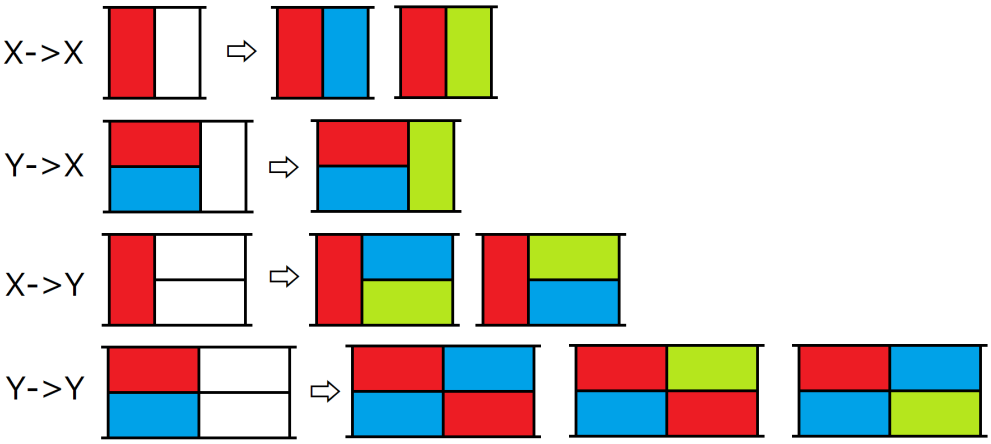
We paint dominoes from left to right.
The number of ways to paint the leftmost part is as follows:

- X: 3
- Y: 6



For other parts, the number of ways to paint a part is as follows (assuming that the left part is already painted):

- X (to the right of X): 2
- X (to the right of Y): 1
- Y (to the right of X): 2
- Y (to the right of Y): 3



The answer is the product of these numbers.

E: Don't Be a Subsequence

First, let's compute the length of the desired string. Later, we'll describe how to extend the solution to compute the lexicographically smallest string.

Let $f(s)$ be the length of the shortest string that is not a subsequence of s. Suppose that t is not a subsequence of s. What t should we choose, assuming that t starts with a c (arbitrary character)?

- If s doesn't contain c, the length of t can be 1: a single c.
- Otherwise, let i be the leftmost occurrence of c in s ($s[i] = c$). t is not a subsequence of s if and only if $t.substr(1)$ is not a subsequence of $s.substr(i + 1)$.

Here, $s.substr(i)$ means the suffix of s from the i-th character.

Therefore, we can compute the length of $f(s)$ using the following DP:

Define $dp[i]$ as the length of $f(s.substr(i))$. Also, precompute $next(i, c)$: the minimum index j such that $j \geq i$ and $s[j] = c$. Then,

$$dp[i] = \min_c \{dp[next(i, c) + 1]\} + 1 \tag{1}$$

holds.

How can we compute the lexicographically smallest string? The first character of the answer should be the lexicographically smallest character c that satisfies:

$$dp[i] = dp[next(i, c) + 1] + 1 \tag{2}$$

The remaining part of the solution should be the lexicographically smallest shortest string that is not a subsequence of $s.substr(next(0, c))$, so we can repeat the same process.

This solution works in $O(|A|m)$, where $m = 26$ is the number of characters.

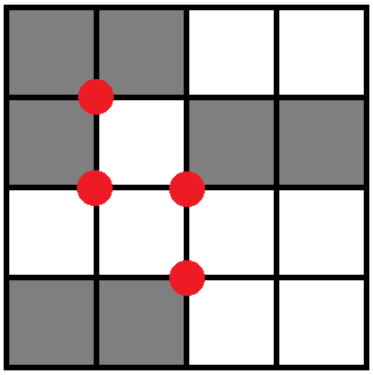
F: Flip and Rectangles

Consider a 2×2 subsquare inside the grid. If it contains odd (1 or 3) number of black squares, we call it "bad". We can prove that, we can choose a rectangle S (after some "invert" operations) if and only if S doesn't contain any bad parts.

If S contains a bad part, the bad part always contain odd number of black squares even after operations, so we can never choose S.

Suppose that S doesn't contain bad parts. By performing operations properly, we can make all squares in the topmost row and the leftmost column of S black. Notice that a good part (not bad part) always remain good after operations. Thus, when all squares in the topmost row and the leftmost column of S are black, the square on the second row of the second column of S must be black. Similarly, we can prove that all squares in S are black.

Now, let's put a token on the center of each bad part. We can choose a rectangle S if and only if S doesn't contain any tokens.



This problem can be solved in the same way as the well-known problem "find the largest rectangle that consists only of black squares".