JOI 2025/2026 セミファイナル 6問目
奇妙な機械 (Strange Machine) 解説

解説: 蜂矢 倫久 (Mitsubachi)

Step 0

問題概要

0 問題概要

- 表裏が白か黒の色をしたタイルが N 枚あります
- 表, 裏の色の情報は文字列 S, T で与えられます

- 2 つのタイルをマージして 1 つのタイルにする操作ができます

0 問題概要

- 表裏が白か黒の色をしたタイルが N 枚あります
- 表, 裏の色の情報は文字列 S, T で与えられます

- 2 つのタイルをマージして 1 つのタイルにする操作ができます
- タイル a, b からタイル c を作れます

- c の**表**の色は a の**裏**の色と b の**表**の色が同じなら黒
- そうでないなら白

- c の**裏**の色は a の**表**の色と b の**裏**の色が同じなら黒
- そうでないなら白

0 問題概要

- 表裏が白か黒の色をしたタイルが N 枚あります
- 表, 裏の色の情報は文字列 S, T で与えられます

- 2 つのタイルをマージして 1 つのタイルにする操作ができます
- タイル a, b からタイル c を作れます

- Q クエリ解きたいです
- 変更クエリと判定クエリがあります

0 問題概要

- Q クエリ解きたいです
- 変更クエリと判定クエリがあります

- 変更クエリ: 1 枚のタイルを別のものに変更
- 判定クエリ: 範囲 [L, R] のタイルを取り出して並べる
- 先ほどの操作を繰り返して表面が白色のタイルの枚数を M 枚にできるか判定

0 問題概要

- Q クエリ解きたいです
- 変更クエリと判定クエリがあります

- 変更クエリ: 1 枚のタイルを別のものに変更
- 判定クエリ: 範囲 [L, R] のタイルを取り出して並べる
- 先ほどの操作を繰り返して表面が白色のタイルの枚数を M 枚にできるか判定

- S, T のある範囲について判定するというもの

0 問題概要

- Q クエリ解きたいです
- 変更クエリと判定クエリがあります

- 変更クエリ: 1 枚のタイルを別のものに変更
- 判定クエリ: 範囲 [L, R] のタイルを取り出して並べる
- 先ほどの操作を繰り返して表面が白色のタイルの枚数を M 枚にできるか判定

- S, T のある範囲について判定するというもの
- S, T のある RANGE について ...

0 問題概要

- Q クエリ解きたいです
- 変更クエリと判定クエリがあります

- 変更クエリ: 1 枚のタイルを別のものに変更
- 判定クエリ: 範囲 [L, R] のタイルを取り出して並べる
- 先ほどの操作を繰り返して表面が白色のタイルの枚数を M 枚にできるか判定

- S, T のある範囲について判定するというもの
- S, T のある RANGE について ...

0 問題概要

- Q クエリ解きたいです
- 変更クエリと判定クエリがあります

- 変更クエリ: 1 枚のタイルを別のものに変更
- 判定クエリ: 範囲 [L, R] のタイルを取り出して並べる
- 先ほどの操作を繰り返して表面が白色のタイルの枚数を M 枚にできるか判定

- S, T のある範囲について判定するというもの
- S, T のある RANGE について ...

STRANGE

0 制約

- N ≦ 300 000
- Q ≦ 300 000
- タイルの色や範囲に特殊な制約はなし

0 小課題

小課題番号	N	Q	クエリ	配点
1	6			6 点
2	100		判定のみ	10 点
3	500		判定のみ	9 点
4	1700		判定のみ	8 点
5	10000	10000		23 点
6	100000	100000		14 点
7				30 点

Subtask 1

$N \leqq 6$

1 全探索

- タイルの色は 4 通り
- 長さ n の列の状態は 4^n = 2^2n 通り

- n <= 6 ならこれは 4096 以下
- 全部の状態を列挙できないか？

1 全探索

- タイルの色は 4 通り
- 長さ n の列の状態は 4^n = 2^2n 通り

- n <= 6 ならこれは 4096 以下
- 全部の状態を列挙できないか？

- できる
- 頂点数 O(4^N) で，辺数 O(N 4^N) の有向グラフの到達判定などとみなせる

1 グラフにする

- 状態の遷移を表すグラフを作ろう
- タイルの列の状態を数字などにしたい

- 4 進数 or 5進数(カードがないことを表す数字をつける) で解釈すると楽

- いずれにでせよ O(N 4^N) でグラフが作れる

1 グラフにする

- 状態の遷移を表すグラフを作ろう
- タイルの列の状態を数字などにしたい

- 4 進数 or 5進数(カードがないことを表す数字をつける) で解釈すると楽

- いずれにでせよ O(N 4^N) でグラフが作れる
- 前準備で全頂点間の到達可能性を調べておく
- DFS とか BFS とか お好みな方法で

1 グラフにする

- クエリはこれを使えば O(1) とかで答えられる
- 全体で O(N 4^2N + Q) など
- 色々定数倍が軽くなると思うので実装が酷すぎなければ通ると思われる

1 グラフにする

- クエリはこれを使えば O(1) とかで答えられる
- 全体で O(N 4^2N + Q) など
- 色々定数倍が軽くなると思うので実装が酷すぎなければ通ると思われる
- 2 ページ後にある実装テクニックを参照
- どうしても TLE するならグラフを対応する状態のタイルの枚数で層にして分割するとかがある
- 多分小課題 2 よりも実装が大変なのでやめた方がいいです

1 別解

- 判定クエリにおいて操作の仕方は O((N - 1)!) 通りになる

- 全部試せばいい
- O((N - 1)! Q) などで解ける

- 特に，状態 s から表が白の枚数を m にしたいという組 (s, m) が 2 回以上出てきたらその後はサボるという高速化ができる
- **メモ化再帰**という名前がついています

1 実装テクニック

- これらの解法においてタイルの状態の列を持つ必要がある

- タイルの各状態を 0 から 3 の整数の 4 通りに対応したい
- でもどうやって？

- 2 * (表の色が黒なら 0, 白なら 1) + 1 * (表の色が黒なら 0, 白なら 1) という表し方が楽

1 さらに考察

- タイルの各状態を 0 から 3 の整数の 4 通りに対応したい
- 2 * (表の色が黒なら 0, 白なら 1) + 1 * (表の色が黒なら 0, 白なら 1) という表し方が楽

- この表し方で操作を解釈する

- c の**表**の色は a の**裏**の色と b の**表**の色が同じなら黒
- そうでないなら白

- c の**裏**の色は a の**表**の色と b の**裏**の色が同じなら黒
- そうでないなら白

1 さらに考察

- タイルの各状態を 0 から 3 の整数の 4 通りに対応したい
- 2 * (表の色が黒なら 0, 白なら 1) + 1 * (表の色が黒なら 0, 白なら 1) という表し方が楽

- この表し方で操作を解釈する

- c の**表**の色は a の**裏**の色と b の**表**の色の **XOR**

- c の**裏**の色は a の**表**の色と b の**裏**の色の **XOR**

1 さらに考察

- c の**表**の色は a の**裏**の色と b の**表**の色の **XOR**
- c の**裏**の色は a の**表**の色と b の**裏**の色の **XOR**

- XOR は排他的累積和のこと
- (この問題の制約では) 足し算して 2 で余りを取ると思って OK

1 以降の前提

- これ以降は黒を 0 に，白を 1 に対応させます

- c の**表**は a の**裏**と b の**表**の **XOR**
- c の**裏**は a の**表**と b の**裏**の **XOR**

1 以降の前提

- これ以降は黒を 0 に，白を 1 に対応させます

- 特に #-@ のように書いたら表が # で裏が @ のタイルと思ってください
- 例えば 1-0 なら表が白で裏が黒のタイルという意味

- * は 0, 1 のどちらも ok という意味
- 例えば 0-* なら表が黒で裏はなんでも OK のタイル

- 判定クエリは 1-* の数を M 枚にしてねというもの

Subtask 2

N ≦ 100, 判定クエリのみ

2 判定クエリのみ

- ありうる全ての判定クエリについて答えを求めておこう

- dp[i][j][k] := 区間 [i, j) から 1-* を k 枚にできるか？
- いわゆる区間 DP

- 遷移を考えると区間を 1 枚にする場合に困る
- これも補助的に区間 DP をしよう
- dp'[i][j][k] := 区間 [i, j) から k に対応したタイル 1 枚にできるか？

2 DP

- dp’[i][j][k] := 区間 [i, j) から k に対応したタイル 1 枚にできるか？
- これは O(N^3) とかでできる

- dp[i][j][k] := 区間 [i, j) から 1-* を k 枚にできるか？
- dp’ があれば O(N^4) とかでできる

- 全体で O(N^4 + Q)

Subtask 3

$N \leqq 500$, 判定クエリのみ

3 前書き

- 小課題 2 で区間 DP を考えた
- この方針は同じ

- 高速化のアイデアとして以下の 2 つがある

- 一般的に使えるもの
- この問題固有の性質を使うもの

- 1 つ実装すると小課題 3 が解けて，2 つ実装すると小課題 4 が解けるはず
- 定数倍にもかなり拠ると思います

3 前書き

- 高速化のアイデアとして以下の 2 つがある

- 一般的に使えるもの
- この問題固有の性質を使うもの

- まずは「一般的に使えるもの」の方を書きます

3 DP の定義

- DP の定義を思い出す

- dp’[i][j][k] := 区間 [i, j) から k に対応したタイル 1 枚に　できるか？
- dp[i][j][k] := 区間 [i, j) から 1-* を k 枚にできるか？

- いずれもできるか？の形
- つまり true / false の bool の形

3 bitset 高速化

- こういうbool の形の DP は bitset で高速化できることが多い

- ワードサイズを w として 1/w がオーダーにつく
- 大体 w = 64 であることが多い

- bitset 高速化をすると O(N^4/w + Q) になる

Subtask 4

$N \leqq 1700$，判定クエリのみ

4 前書き

- 高速化のアイデアとして以下の 2 つがある

- 一般的に使えるもの
- この問題固有の性質を使うもの

- ここでは「この問題固有の性質を使うもの」の方を書きます

4 考察

- この問題の判定クエリが解けるなら，1-* の最大化や最小化も解けそう

- 最大化を考えてみよう！
- まず，操作によって 1-* の数の差分はどうなる？

4 差分に着目

- この問題の判定クエリが解けるなら，1-* の最大化や最小化も解けそう

- 最大化を考えてみよう！
- まず，操作によって 1-* の数の差分はどうなる？

- 減る方は -2, -1 がありえる
- 増える方は +1 がありえる

- 差分は -2 から +1 の間をとる

4 性質 01

- **性質 01:**
- 現在の値と最大値の間は全部取れる

- これより上の性質が成り立つことがいえる
- 補題として「差分は -2 から +1 の間をとる」を用いる

- 数学が得意な人は「中間値の定理」と思うとピンと来るかも

4 証明 01

- **性質 01:**
- 現在の値と最大値の間は全部取れる

- **証明 01:**
- 補題「差分は -2 から +1 の間をとる」を用いる
- 現在の 1-* を x として 1-* を最大化する操作列を 1 つ取る
- 最終結果が x’ になったとする (x <= x’)

- x < z < x’ で z 枚にできないような z が存在するとする
- このとき z - 1 枚以下から z + 1 枚以上に 1 回の操作で変化
- これは補題に矛盾 よって示された

4 最大化

- 現在の値を求めるのは簡単 (累積和とかなんでも)

- 最大値を求めよう
- まず，最初に 1-* であるタイルは触らなくて良い

- タイルの列を区間に分割して，それぞれを 1 枚のタイルにすると捉えると，1-* であるタイルを含む区間からは 1-* は最大 1 枚しか得られない
- それは最初に達成されてるので，区間から消して残りで作ることで損をしない

4 最大化

- 現在の値を求めるのは簡単 (累積和とかなんでも)

- 最大値を求めよう
- まず，最初に 1-* であるタイルは触らなくて良い

- 残りは 0-* のタイルのみからなる区間での最大化
- それぞれのタイルは 0-0 か 0-1

4 最大化

- 残りは 0-* のタイルのみからなる区間での最大化
- それぞれのタイルは 0-0 か 0-1

- [[0-1 * n] + [0-0 * m]] のブロックがたくさん並んでいると思える
- ここで，n, m は非負

- 0-0 を左に置く操作は何も意味しないと思うとこの順番になる

4 最大化

- 残りは 0-* のタイルのみからなる区間での最大化
- それぞれのタイルは 0-0 か 0-1

- [[0-1 * n] + [0-0 * m]] のブロックがたくさん並んでいると思える
- ここで，n, m は非負

- 各ブロックについて最大値は (n + min(m, 1)) / 2 (切り捨て)
- 左から 2 つずつ操作をしてけば達成可能

4 最大化

- [[0-1 * n] + [0-0 * m]] のブロックがたくさん並んでいると思える
- ここで，n, m は非負

- 各ブロックについて最大値は (n + min(m, 1)) / 2 (切り捨て)
- 左から 2 つずつ操作をしてけば達成可能

- 実は全体の最大値も各ブロックの最大値の総和になる
- ブロックを跨ぐ操作をすると 0-0 と 0-1 で操作することになるがこれは意味がないため

4 最大化

- [[0-1 * n] + [0-0 * m]] のブロックがたくさん並んでいると思える
- ここで，n, m は非負

- 各ブロックについて最大値は (n + min(m, 1)) / 2 (切り捨て)
- 左から 2 つずつ操作をしてけば達成可能

- 実は全体の最大値も各ブロックの最大値の総和になる
- ブロックを跨ぐ操作をすると 0-0 と 0-1 で操作することになるがこれは意味がないため
- 長さについての帰納法でも示せると思う

4 最大化

- 結局のところ，最大値は O(N) で求めることができる
- 最大化はまともそう
- 最小化を考える

4 最小化へ

- 結局のところ，最大値は O(N) で求めることができる
- 最大化はまともそう
- 最小化を考える
- まず先ほど書いた区間 DP テーブルを観察🔍
- すると大体 true になってる

4 最小化

- 最小化を考える 区間 DP テーブルを観察🔍

- 大体 true になってる
- 逆にそうではないケースは true / false が交互になっている感じ

- 「そうではないケース」を観察すると 0-0 や 1-1 がほとんど
- 「そうであるケース」では 3 以上はできそう

4 最小化

- 最小化を考える 区間 DP テーブルを観察🔍

- 大体 true になってる
- 逆にそうではないケースは true / false が交互になっている感じ

- 「そうではないケース」を観察すると 0-0 や 1-1 がほとんど
- 「そうであるケース」では 3 以上はできそう

- 実は成り立つ
- 証明をしよう

4 初手に着目

- まず初手に着目する

- 1-0 + 0-* => 0-*
- 1-0 + 1-* => 1-*
- 0-1 + 1-* => 0-*
- 0-1 + 0-* => 1-*

- の操作は 1-* の数が 1 だけ増減する
- 上 3 つが -1, 下 1 つは +1 になる

4 初手に着目

- 1-0 + 0-* => 0-*
- 1-0 + 1-* => 1-*
- 0-1 + 1-* => 0-*
- 0-1 + 0-* => 1-*

- これらができないとき，0-1, 1-0 は存在するなら右端になる
- 操作ができないというのは操作しても保たれる

- このようなとき 1-* の数の偶奇は不変で，減らす方は 2 ずつ減っていく
- これらの操作のいずれかができるとしよう

4 初手に着目

- 1-0 + 0-* => 0-*
- 1-0 + 1-* => 1-*
- 0-1 + 1-* => 0-*
- 0-1 + 0-* => 1-*

- これらができるとして，そのような操作ができる 2 枚を取り出す
- 列は 左側 L + (その 2 枚) + 右側 R
- 左側からの最小値を m_L, 右側からの最小値を m_R とする
- ここで，これらは共に 1 以下 (1 枚になるまで操作すれば OK)

4 初手に着目

- これらができるとして，そのような操作ができる 2 枚を取り出す
- 列は 左側 L + (その 2 枚) + 右側 R
- 左側からの最小値を m_L, 右側からの最小値を m_R とする
- ここで，これらは共に 1 以下 (1 枚になるまで操作すれば OK)

- 今 L, 2 枚, R に 1-* が n_L, n_M, n_R 枚あるとする

4 初手で -1 ができるとき

- 今 L, 2 枚, R に 1-* が n_L, n_M, n_R 枚あるとする

- (1) -1 の操作ができるとき
- 2 枚での操作をしないことで n_L + n_M + n_R から m_L + n_M + m_R への操作列が得られる
- 最初にすることで n_L + n_M + n_R - 1 から m_L + n_M + m_R - 1 への操作列が得られる

- 1-* の数の差分が -2 から -1 であることを考えると， m_L + n_M + m_R - 1 から n_L + n_M + n_R までの全てが作れる

4 初手で -1 ができるとき

- 今 L, 2 枚, R に 1-* が n_L, n_M, n_R 枚あるとする

- (1) -1 の操作ができるとき
- 1-* の数の差分が -2 から -1 であることを考えると， m_L + n_M + m_R - 1 から n_L + n_M + n_R までの全てが作れる

- 証明: 背理法

- m_L + n_M + m_R - 1 <= 1 + 2 + 1 - 1 = 3 である

4 初手で +1 ができるとき

- 今 L, 2 枚, R に 1-* が n_L, n_M, n_R 枚あるとする
- (2) +1 の操作ができるとき
- (1) と同様に考えると， m_L + n_M + m_R から n_L + n_M + n_R までの全てが作れる
- 証明: 背理法
- m_L + n_M + m_R <= 1 + 0 + 1 = 2 である

4 初手で -1 か +1 ができるとき

- 結局 -1 か +1 ができるなら 3 以上は作ることができる

- つまり dp[i][j][k] := 区間 [i, j) から 1-* を k 枚にできるか？
- の k として考えるべき範囲は 0 <= k <= 2 のみで良い！

- これにより DP の計算量を落とせる

- 全体で O(N^3/w + Q) などで解ける
- これ単体だと O(N^3 + Q) で小課題 3 が解けます

Subtask 5

$N \leqq 1\,000, Q \leqq 1\,000$

5 振り返り

- 区間から作れる最大値は O(N) でできる
- 減らす方は 0, 1, 2 が作れるかの判定となる
- この判定をなんとか高速化したい

5 実験

- やっぱり先ほどと同様に実験をしてみる
- するといくつかルールが見えてくる
- oo ができない というのは「1-* が oo 枚になるような操作ができない」という意味を指します

5 性質 02

- **性質 02:**
- 0 ができないことの必要十分条件は 0-1, 1-0 が右 2 つ以外ないことである

5 性質 02

- **性質 02:**
- 0 ができないことの必要十分条件は 0-1, 1-0 が右 2 つ以外ないことである

- **証明 02:**
- いくつかの区間に分け，各区間が 1 つのタイルになるがそれが全て 0-* の形になってほしい
- 右 2 枚を除けば 1-1 の枚数の偶奇のみに依存

- 左の 1-1 が偶数 => 右 2 枚に帰着

5 性質 02

- **性質 02:**
- 0 ができないことの必要十分条件は 0-1, 1-0 が右 2 つ以外ないことである

- **証明 02:**
- 左の 1-1 が偶数 => 右 2 枚に帰着
- できる: [0-0, 0-*], [0-1, *-*], [1-0, 0-*], [1-1, 1-*]
- できない: otherwise

- 左の 1-1 が奇数 => 1-1 と右 2 枚に帰着
- できる: 1-1 + ([0-0, 1-*], [0-1, 0-*], [1-0, *-*], [1-1, 0-*])
- できない: otherwise

5 性質 03

- **性質 03:**
- 1 ができないことの必要十分条件は [特性 1](1-* が偶数) と (1-0, 0-1 が右端以外ない) である

5 性質 03

- **性質 03:**
- 1 ができないことの必要十分条件は (1-* が偶数) と (1-0, 0-1 が右端以外ない) である

- **証明 03:**
- 長さの帰納法を用いる 長さ 1 では明らか
- まず，上の特性が成り立つ列が一手後も成り立つことを示す
- 列は [0-0 or 1-1] * (L - 1) + [*-*] と書ける
- *-* が絡まなければ成り立つ
- *-* が絡んでも成り立つことが場合わけで示される

5 性質 03

- **性質 03:**
- 1 ができないことの必要十分条件は (1-* が偶数) と (1-0, 0-1 が右端以外ない) である

- **証明 03:**
- 上の特性が成り立たない列が一手後で成り立たないようにできる or 1 が作れることを示す
- 右 2 つ以外に 0-1, 1-0 があると可能なのでそうでないとする
- [0-0 or 1-1] * (L - 2) + [0-1 or 1-0] + [*-*] と書ける
- (1) 右から 2 つめが 0-1 とする
- (1 - i) L - 2 の中の 1-1 が偶数とする

5 性質 03

- **性質 03:**
- 1 ができないことの必要十分条件は (1-* が偶数) と (1-0, 0-1 が右端以外ない) である

- **証明 03:**
- [0-0 or 1-1] * (L - 2) + [0-1 or 1-0] + [*-*] と書ける
- (1) 右から 2 つめが 0-1 とする
- (1 - i) L - 2 の中の 1-1 が偶数とする
- 右端が 0-* なら最初に右 2 つを merge すれば良い
- 右端が 1-* なら L - 2 を全部 0-0 にすれば 1-* が作れる

5 性質 03

- **性質 03:**
- 1 ができないことの必要十分条件は (1-* が偶数) と (1-0, 0-1 が右端以外ない) である

- **証明 03:**
- [0-0 or 1-1] * (L - 2) + [0-1 or 1-0] + [*-*] と書ける
- (1) 右から 2 つめが 0-1 とする
- (1 - ii) L - 2 の中の 1-1 が奇数とする
- 右端が 0-* なら右端以外をまとめて 1-0 を作れば良い
- 右端が 1-* なら L - 2 をまとめると 1-* になり，右 2 つをまとめると 0-* となり，可能

5 性質 03

- **性質 03:**
- 1 ができないことの必要十分条件は (1-* が偶数) と (1-0, 0-1 が右端以外ない) である

- **証明 03:**
- [0-0 or 1-1] * (L - 2) + [0-1 or 1-0] + [*-*] と書ける
- (2) 右から 2 つめが 1-0 とする
- (2 - i) L - 2 の中の 1-1 が偶数とする
- 右端が 0-* なら右端以外をまとめて 1-0 を作れば良い
- 右端が 1-* なら全てまとめると 1-* ができる

5 性質 03

- **性質 03:**
- 1 ができないことの必要十分条件は (1-* が偶数) と (1-0, 0-1 が右端以外ない) である

- **証明 03:**
- [0-0 or 1-1] * (L - 2) + [0-1 or 1-0] + [*-*] と書ける
- (2) 右から 2 つめが 1-0 とする
- (2 - ii) L - 2 の中の 1-1 が奇数とする
- 右端が 0-* なら右 2 つをまとめれば良い
- 右端が 1-* なら右端以外を 0-1 にまとめれば良い

5 性質 03

- **性質 03:**
- 1 ができないことの必要十分条件は (1-* が偶数) と (1-0, 0-1 が右端以外ない) である

- **証明 03:**
- ここまでの議論を総合すると [0-0, 1-1] * (L - 1) + [*-*] を考えれば良い
- 1-* が奇数: 右 2 つを merge すれば良い
- 1-* が偶数: 仮定に矛盾

- 以上より帰納法が回る

5 性質 04

- **性質 04:**
- 2 ができないことの必要十分条件は (作れる最大値が 2) かつ (1-* が奇数) と (0-1 が右端以外ない) である

5 性質 04

- **性質 04:**
- 2 ができないことの必要十分条件は (作れる最大値が 2 未満) かつ (1-* が奇数) と (0-1 が右端以外ない) である

- **証明 04:**
- 最大値 2 以下については自明
- 0, 1 なら無理だし，2 なら明らかにできる

- 最大値 3 以上として，(1-* が奇数) と (0-1 が右端以外ない) に議論を絞る

5 性質 04

- **性質 04:**
- 2 ができないことの必要十分条件は (作れる最大値が 2 未満) かつ (1-* が奇数) と (0-1 が右端以外ない) である

- **証明 04:**
- (1-* が奇数) と (0-1 が右端以外ない) のとき
- [0-0 or 1-1] * (L - 1) + [0-1 or 1-1] と書ける
- 1-* の偶奇は不変なので良い

- そうでないケースを考える
- 今の 1-* が 2 以下なら最大値に向かう途中でできる

5 性質 04

- **性質 04:**
- 2 ができないことの必要十分条件は (作れる最大値が 2 未満) かつ (1-* が奇数) と (0-1 が右端以外ない) である

- **証明 04:**
- (1-* が奇数) と (0-1 が右端以外ない) でないケースを考える
- 今の 1-* が 2 以下なら最大値に向かう途中でできる
- 今の 1-* が 3 以上とする
- [0-1, 0-*] or [0-1, 1-*] が連続部分列に含まれてれば 2 が作れる
- この 2 枚を触らない操作列とこれを先に触る操作列を考えれば良い (小課題 4 の「初手に着目」と同様の考え方)

5 性質 04

- **性質 04:**
- 2 ができないことの必要十分条件は (作れる最大値が 2 未満) かつ (1-* が奇数) と (0-1 が右端以外ない) である

- **証明 04:**
- (1-* が奇数) と (0-1 が右端以外ない) でないケースを考える
- [0-1, 0-*] or [0-1, 1-*] が連続部分列にないとする

- [1-*] * ?, [0-0] * ?, [1-*] * ?, ..., [0-1] * (0 or 1) みたいなブロックの並びになる

5 性質 04

- **性質 04:**
- 2 ができないことの必要十分条件は (作れる最大値が 2 未満) かつ (1-* が奇数) と (0-1 が右端以外ない) である

- **証明 04:**
- [1-*] * ?, [0-0] * ?, [1-*] * ?, ..., [0-1] * (0 or 1) みたいなブロックの並びになる
- (1) 1-0 がないケース
- 1-* が 1-1 のみとなり，1-* の偶奇が不変量となるため成立
- (2) 1-0 があるケース
- 0-1 があるかないかで場合分け

5 性質 04

- **性質 04:**
- 2 ができないことの必要十分条件は (作れる最大値が 2 未満) かつ (1-* が奇数) と (0-1 が右端以外ない) である

- **証明 04:**
- [1-*] * ?, [0-0] * ?, [1-*] * ?, ..., [0-1] * (0 or 1) みたいなブロックの並びになる
- (2) 1-0 があるケース
- (2 - i) 0-1 がある場合
- 1-0 が右端以外にあれば 2 が作れ，0-1 が右端より良い

5 性質 04

- **性質 04:**
- 2 ができないことの必要十分条件は (作れる最大値が 2 未満) かつ (1-* が奇数) と (0-1 が右端以外ない) である

- **証明 04:**
- [1-*] * ?, [0-0] * ?, [1-*] * ?, ..., [0-1] * (0 or 1) みたいなブロックの並びになる
- (2) 1-0 があるケース
- (2 - ii) 0-1 がない場合
- 1-0 が右端のみのケースを考えると [0-0 or 1-1] * (L - 1) + [1-0] となり，1-* の偶奇が不変量となるため成立

5 解法

- 0-1, 1-0 は存在するなら右端かの判定も O(N) でできる
- 0, 1, 2 ができるかは O(N) で判定できる
- 最大値も O(N) で判定できる
- よってクエリあたり O(N) で解ける
- 全体で O(NQ) となり解ける

Subtask 7

追加制約なし

7 振り返り

- 定数倍が悪いか遅い言語だと小課題 6 までだと思います
- 基本的には小課題 7 ということで，これを説明

- 小課題 5 の解法を振り返る

- 0-1, 1-0 は存在するなら右端かの判定も O(N) でできる

- 0, 1, 2 ができるかは O(N) で判定できる

- 最大値も O(N) で判定できる

7 Segment Tree へ

- 0-1, 1-0 は存在するなら右端かの判定も O(N) でできる
- 0, 1, 2 ができるかは O(N) で判定できる
- 最大値も O(N) で判定できる
- とあったが，実はこれらは Segment Tree に乗る

7 解法

- 0-1, 1-0 は存在するなら右端かの判定

- これは 0-0, 0-1, 1-0, 1-1 が区間に何個あるか分かれば良い
- 一点更新 / 区間和 の Segment Tree になる

- 4 本持ってもいいし，モノイドに 4 つの個数の情報を載せても良い
- 4 つの情報を持たせるときは array を使うと定数倍が良くなる

- 0, 1, 2 ができるかの判定もこの Segment Tree でできる

7 解法

- 最大値の計算
- ブロックごとに計算できるので，ブロックの情報を持たせたい
- マージ (ACL の op) を考えると，次のような情報が欲しい
- 左端を含むブロックの情報
- 右端を含むブロックの情報
- 今の区間のブロックがちょうど 1 つであるか

7 解法

- 最大値の計算
- ブロックごとに計算できるので，ブロックの情報を持たせたい
- マージ (ACL の op) を考えると，次のような情報が欲しい
- 左端を含むブロックの情報
- 右端を含むブロックの情報
- 今の区間のブロックがちょうど 1 つであるか
- 後はこれらで頑張るとマージできる (実装は重いです)

7 解法

- 結局 Segment Tree で全て処理できる形になった
- 2 種類あるので，抽象化しておくと実装が楽になる
- AtCoder 環境では ACL が使えるので楽
- (抽象化できなくとも)ソラ書きできるとファイナルでは役に立ちます

7 解法

- 0-1, 1-0 は存在するなら右端か
- 0, 1, 2 ができるか
- 最大値

- これらは Segment Tree に乗る
- O(log N) で一点変更や区間積が計算できる

- 全体で O(N + Q log N) で解ける

7 解法

- 0-1, 1-0 は存在するなら右端か
- 0, 1, 2 ができるか
- 最大値

- これらは Segment Tree に乗る
- O(log N) で一点変更や区間積が計算できる

- 全体で O(N + Q log N) で解ける

- ちなみに，　0-1, 1-0 が右端以外にある場合，1, 2 は必ずできるので実装上は 0 のみで良い

Step 8

得点分布

8 得点分布

- 理想:

点数	1	2	3	4	5	6	人数	累計
100	o	o	o	o	o	o	170	170

- 選抜という意味では差がつかないな

8 得点分布

- 現実:

点数	1	2	3	4	5	6	人数	累計
100	o	o	o	o	o	o	1	1
25	o	o	o	-	-	-	1	2
10	-	o	-	-	-	-	3	5
6	o	-	-	-	-	-	9	14
0	-	-	-	-	-	-	156	170