いろはちゃんコンテスト Day4 解説

この pdf は、いろはちゃんコンテスト Day4 の各問題の解説をまとめたものです。個別の解説はそれぞれの問題ページ下部にあります。

最終日なのでおまけに各問題の FA 情報もつけます

A - あ - rickytheta　12:03

B - さ - TMJN　2:59

C - き - catupper　39:44

D - ゆ - zaki_　33:33

E - め - satashun　19:02

F - み - beet　41:33

G - し - nuip 155:35

H - ゑ - zundamochi_1117　30:48

I - ひ - noshi91　41:39

J - も - 該当者なし

K - せ - nuip　28:34

L - す - 該当者なし

きたむーの独り言。

前書き

コンテスト終了時に起きていられる自信がないので、普段 Twitter に書くことをちょっとだけ書いておきます。

解説は次のページから始まります。これめっちゃ大事。別にこの 1 枚目読まなくていいです。

きたむーの彼女

これはいろはコンですが、彼女はいろはちゃんではありません。

鬼の実行時間制限と実行メモリ制限

Day2 のときに Python が落ちて叩かれてしまったのですが、僕の問題はやたら時間とメモリの制限が厳しいです。実は「あるよるのできごと」ももともと「1 秒、384MB」でした。これは、

・メモ化再帰を使って無駄な処理を省くと実は $O(N^4)$ ($N \leqq 100$)でも 50ms かからないことに気付いてほしい(C++想定解は 34ms、Python でも 1 秒あれば流石に足りると思われる)

・適切な変数型を選べるようになってほしい

というのが理由です。前者は割と汎用的なテクだと思いますし、後者も bool 型の代わりに long long 型を使うのは流石にエレガントじゃないと思うので・・・。

なお、制限変更で、多分ループで DP しても long long 型を使っても通るようになったと思うのですが、この変更がコンテスト 2 日前のことで、解説の修正が間に合わなかったので、この後の解説では bool 型を使いましょうとなっています。

writer 勢の強い人々がそうだったので僕の鬼の制限に疑問のある人はたくさんいると思いますが、僕は所詮青色なので許してください。

あと、もしかしたらこの変更で $O(N^5)$が通るようになっちゃったかもしれません。もうここまでくると問題として破綻してる気がしますが、今更差し替えもできないので仕方がないです。すみません。今後有志コンを開く方は余裕をもって作問をし、余裕をもってテストをすることを強く勧めます。

配点が適正か

適正ではないです。300 ではないと思います。絶対。でも、100 ずつ上がってるのが美しい、みたいな感じで 300 が維持されました。ごめんなさい。

大荒れのいろはコンだった気がしますが、楽しんで頂けたなら幸いです。

来年の GW は 10 日間×10 問の「ちはやふる」コンテストを僕が主催します。Twitter で writer を募集しているので是非。(大嘘)

2019/5/2　きたむー(@Pro_ktmr)

いろはちゃんコンテスト解説

Day4-A:あるよるのできごと

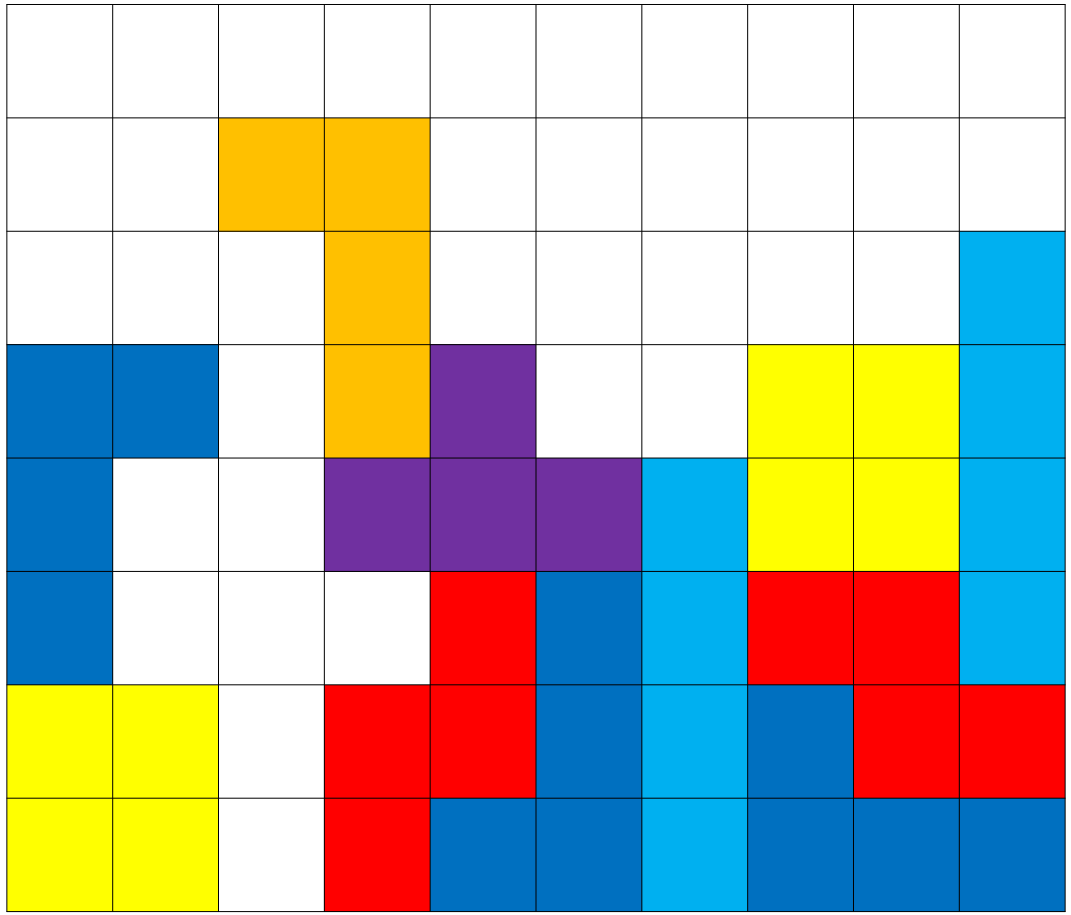
@Pro_ktmr

注意

- 内容の正確性には十分注意していますが、間違った情報が含まれることもあります。何かお気づきのことがあればご連絡ください。
- このスライドのサンプルコードはC++で実装されています。

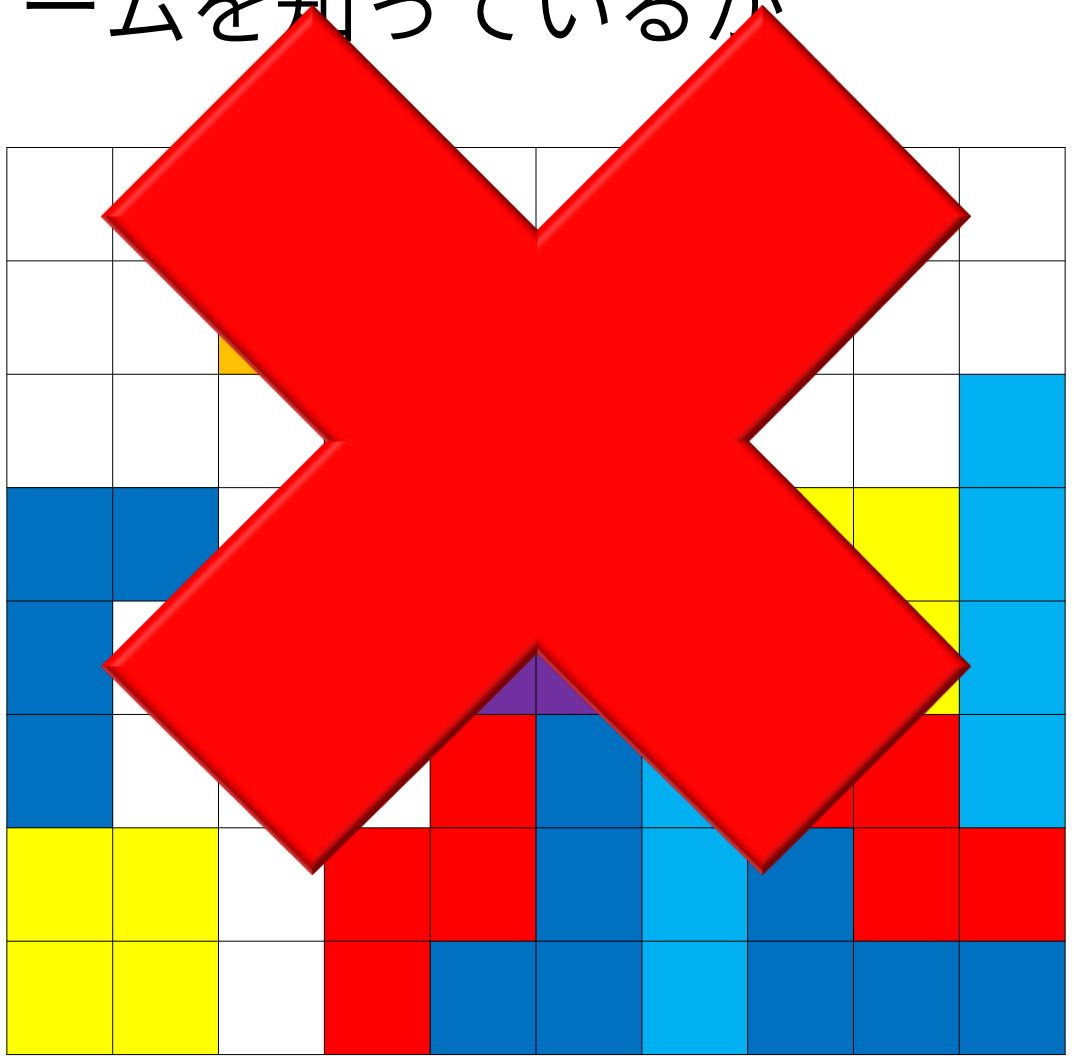
この問題の狙い

- テト〇スというゲームを知っているか



この問題の狙い

- テト〇スというゲームを知っているか



この問題の狙い

- Tricky T〇wersというゲームを知っているか

https://store.steampowered.com/app/437920/Tricky_Towers/?l=japanese

この問題の狙い

- Tricky T〇wersというゲームを知っているか

https://store.steampowe[illegible]om/app/43792[illegible]cky_Towers/?l=japanese

この問題の狙い

- DPを見抜けるか
  JOIをやや意識し、保持するデータ数の多い(DP配列の次元が大きい)DPを用意した

問題概要

- ゲームの得点が入ったログのデータが与えられる
- ログを満たす各ラウンドの対戦結果が有り得るか判定し、有り得るならばそれを出力

- プレイヤー数 : 4人
- ラウンド数N ≦ 100回

時間計算量からの推測

- プレイヤー数 : 4人
- ラウンド数N ≦ 100回

- O(N^4)なら間に合いそう
- 各プレイヤーについて何らかの情報を持たせてDPをすると良さそう

どんな情報を持つか

- 思いつくのは、
  - 今何ラウンド目を考えているか
  - プレイヤーAは何回得点を得ているか
  - プレイヤーBは何回得点を得ているか
  - プレイヤーCは何回得点を得ているか
  - プレイヤーDは何回得点を得ているか
- このままではN^{5}のデータが必要になる

どんな情報を持つか

- 思いつくのは、
  - 今何ラウンド目を考えているか
  - プレイヤーAは何回得点を得ているか
  - プレイヤーBは何回得点を得ているか
  - プレイヤーCは何回得点を得ているか
  - プレイヤーDは何回得点を得ているか
- このままではN^{5}のデータが必要になる？

どんな情報を持つか

- 思いつくのは、
  - 今何ラウンド目を考えているか
  - プレイヤーAは何回得点を得ているか・・・a
  - プレイヤーBは何回得点を得ているか・・・b
  - プレイヤーCは何回得点を得ているか・・・c
  - プレイヤーDは何回得点を得ているか・・・d
- 今何ラウンド目かは、(a+b+c+d)/3すれば求められる

- 他の要素から復元できる値がないか確認するのはDPにおいて非常に重要

実装面の注意

- $N^4=10^8$で、特にスクリプト型言語ではACするか微妙
- ループを回すDPよりも、再帰メモ化+枝切りの方が高速化が可能な場合が多い

- 例えば、a+b+c+dが3の倍数でない状況を満たす対戦結果は絶対に存在しない→処理が3分の1になる
- 特定のプレイヤーにのみ勝利/敗北が偏っている→後々厳しくなるのが目に見えている
- こういった工夫によって定数倍高速化が期待できる

実装面の注意

- $N^4=10^8$で、int型などでDPテーブルを持つとMLE
- bool型を使用する
- 入力を満たす対戦結果を出力するには、経路復元をする
- 経路復元はDP関数とほぼ同じように再帰で実装することができる

実装例

```
#include"bits/stdc++.h"
using namespace std;
int N, A, B, C, D, a[101]={}, b[101]={}, c[101]={}, d[101]={};
int main(){
		scanf("%d%d%d%d%d", &N, &A, &B, &C, &D);
		for(int i=0; i<A; i++) scanf("%d", a+i);
		for(int i=0; i<B; i++) scanf("%d", b+i);
		for(int i=0; i<C; i++) scanf("%d", c+i);
		for(int i=0; i<D; i++) scanf("%d", d+i);

		if((A+B+C+D)%3 == 0 && (A+B+C+D)/3 == N && dp(0, 0, 0, 0)){
				printf("%s¥n", "Yes");
				restore(0, 0, 0, 0);
		}
		else printf("%s¥n", "No");
}
```

- 関数「dp」「restore」は次のスライド

実装例

```
bool visited[101][101][101][101]={}, memo[101][101][101][101]={};
vector<int> v ={ 1, 2, 3 }, tmp;
bool dp(int i, int j, int k, int l){
        if(i==A && j==B && k==C && l==D) return true;
        if(visited[i][j][k][l]) return memo[i][j][k][l];
        tmp.clear();
        tmp ={ a[i], b[j], c[k] }; sort(tmp.begin(), tmp.end());
        if(v == tmp) memo[i][j][k][l] |= dp(i+1, j+1, k+1, l);
        tmp.clear();
        tmp ={ a[i], b[j], d[l] }; sort(tmp.begin(), tmp.end());
        if(v == tmp) memo[i][j][k][l] |= dp(i+1, j+1, k, l+1);
        tmp.clear();
        tmp ={ a[i], d[l], c[k] }; sort(tmp.begin(), tmp.end());
        if(v == tmp) memo[i][j][k][l] |= dp(i+1, j, k+1, l+1);
        tmp.clear();
        tmp ={ d[l], b[j], c[k] }; sort(tmp.begin(), tmp.end());
        if(v == tmp) memo[i][j][k][l] |= dp(i, j+1, k+1, l+1);
        return memo[i][j][k][l];
}
```

```
void restore(int i, int j, int k, int l){
        if(i==A && j==B && k==C && l==D) return;
        tmp.clear(); tmp ={ a[i], b[j], c[k] }; sort(tmp.begin(), tmp.end());
        if(v == tmp && dp(i+1, j+1, k+1, l)){
                printf("4¥n"); restore(i+1, j+1, k+1, l); return;
        }
        tmp.clear(); tmp ={ a[i], b[j], d[l] }; sort(tmp.begin(), tmp.end());
        if(v == tmp && dp(i+1, j+1, k, l+1)){
                printf("3¥n"); restore(i+1, j+1, k, l+1); return;
        }
        tmp.clear(); tmp ={ a[i], d[l], c[k] }; sort(tmp.begin(), tmp.end());
        if(v == tmp && dp(i+1, j, k+1, l+1)){
                printf("2¥n"); restore(i+1, j, k+1, l+1); return;
        }
        tmp.clear(); tmp ={ d[l], b[j], c[k] }; sort(tmp.begin(), tmp.end());
        if(v == tmp && dp(i, j+1, k+1, l+1)){
                printf("1¥n"); restore(i, j+1, k+1, l+1); return;
        }
}
```

Thank you for reading!

いろはちゃんコンテスト **Day4**

叫び声 解説

Writer : ミドリムシ

問題概要

・駅が**M+1**個、電車が**N**個ある
・電車**i**は駅**j+1**には時刻$A_i+B_i \times j$に着く
・走ると駅**j**から駅**j+1**まで**L**単位時間で移動できる
・今は駅**1**にいて時刻**0**、駅**M+1**に着く最速の時刻は？

考察1

電車から他の電車には乗り換えないとしてよい

なぜ？

電車**i**から電車**j**に乗り換えたとする

$B_i > B_j$なら始めから電車**j**に乗っていればいい
それ以外なら乗り換える意味がない

考察2

電車から走行には切り替えないとしてよい

なぜ？

電車**i**から走行に切り替えたとする

B_i>Lなら始めから走っていればよい
それ以外なら切り替える意味がない

考察3

走行から電車には切り替えないとしてよい

なぜ？

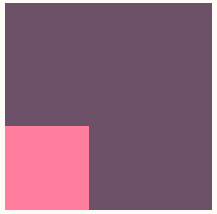
走行から電車**i**に切り替えたとする

$L>B_i$なら始めから電車**i**に乗っていればいい
それ以外なら切り替える意味がない

解法

始めから同じ電車に乗り続ける **or** 走り続ければよい

答えは **$\min(L\times M, A_1+B_1\times M, \ldots A_N+B_N\times M)$** になる



君の力に(原題：暗号化いろはちゃん)解説

Writer : MMNMM

問題概様

01-文字列 s に次の操作を x 回以下行う

左端に '0' を 2 つ追加する

s[i] と s[i+2] の xor をとり、新しい文字列の s[i] とする

与えられた文字列より大きくできるか?

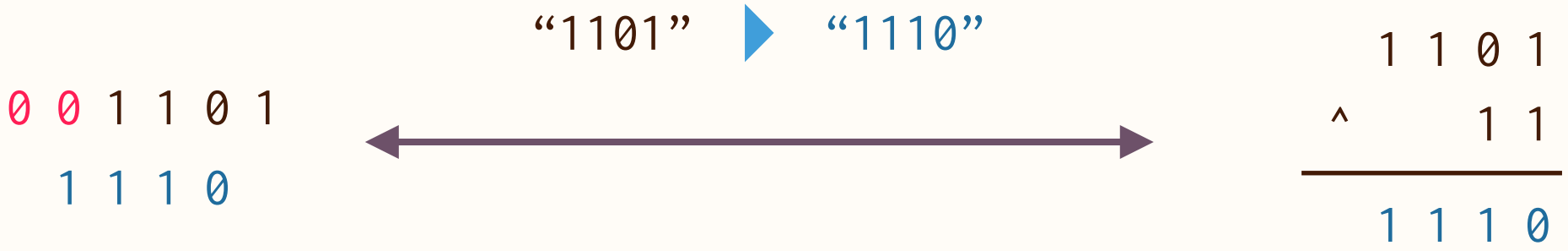
(一般性を失わずに最上位が 1 であるとしてよい)

いいかえる

文字列の長さは変わらない

→ 2 進整数とみて、x回で得られる最大のものを求めればよい

操作も、s を s ^ (s >> 2) とする と言い換えられる



実験

1回操作する → s ^ (s >> 2)

2回操作する → s ^ (s >> 4)

4回操作する → s ^ (s >> 8)

これは...?

回数 n を2べきの和に分解する！

		s >> n							
		0	2	4	6	8	10	12	14
k	0	1	0	0	0	0	0	0	0
	1	1	1	0	0	0	0	0	0
	2	1	0	1	0	0	0	0	0
	3	1	1	1	1	0	0	0	0
	4	1	0	0	0	1	0	0	0
	5	1	1	0	0	1	1	0	0
	6	1	0	1	0	1	0	1	0
	7	1	1	1	1	1	1	1	1

x 回に限らないとき

2 bit ずつのまとまりを上から
貪欲に決めていけば OK

(大きくなればとる そうでなければとらない)

これは操作回数 k を
下の bit から決めていっているということ

x があるとき

よく考えると、さっきのアルゴリズムで k < M を保って
できるだけ操作を続けると最適になっている

文字列を上から、操作回数を下から貪欲に決めているので、

- このビットより上はこれより大きくはならない
- このビットより上がこの状態になるような最小手数が k である

が成り立っている

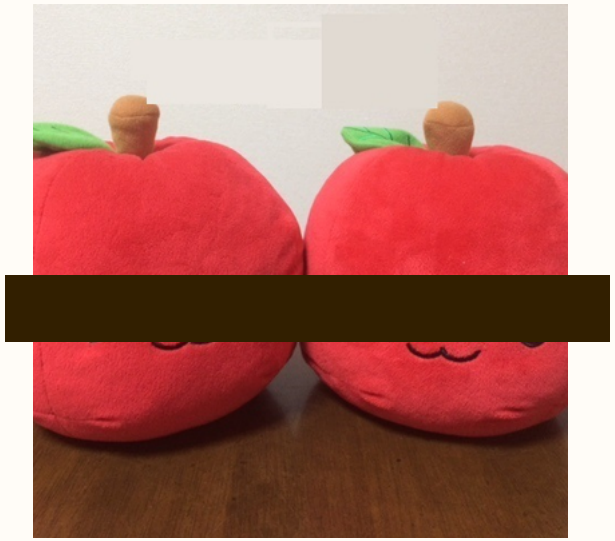
想定解法

上のビットから貪欲に決めていくことで、$\mathcal{O}\left(|S| \log |S|\right)$ で
(文字列, 回数) の増加列を作ることができます

これを元に各クエリに答えると、文字の比較は全体で

$\mathcal{O}\left(\min\left\{|S|, \sum_i |s_i|\right\}\right)$ 回程度 当然間に合います

!?



tester : ?869120氏

この問題、もうちょっと楽に
解くことができますよ

（発言は架空のものです）

シミュレーション

とりあえずどんな行動をするかを考えてみると、

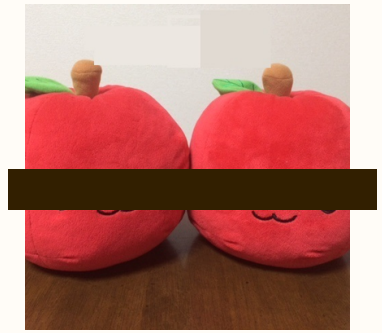
- 倒せるなら、目の前の敵を倒す
- 倒せないなら、魔法をかける

が最適に見える

65536回操作すると `x ^ (x >> 131072)` となり元に戻るため、
65536回以内に倒しきれなかったら諦める

じつはこれでACできる

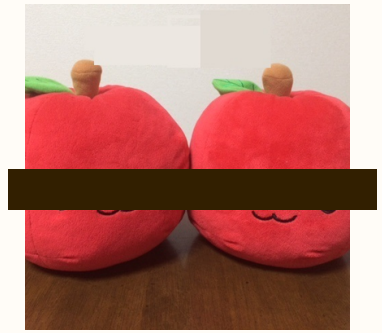
正当性



tester : ?869120氏

シミュレーションをするときに
S ^= S >> 2の回数が多くなりすぎませんか？

std::bitsetで常勝！(|S|^2/64は間に合います)



tester : ?869120氏

勝ち負けの判定が2乗になってしまうのでは…

長く比較する必要があることは稀なので
そんなことはありません

（やりとりは架空のものです）

統計

- First Submission : emthrm (33:59)
- First Acceptance : catupper (39:44)

- Shortest : kotatsugame (804 byte)

- AC / Challenge Rate : 5 /24 ← これはなに

裏話 ~ぽかいこは難易度判定ができない~

(おふろにはいっていたら
ビット系の問題が生えました
投げてみましょう)

Boss Rushに1問、
ビット系の彩りはいかがですか？

いいですね、ちょっと難しめなので...
500点ぐらいだと思います
(注 : 解法を間違えていたらしい)

作問陣

(まあ私も10分くらいで思いついたし、)
それくらいですよね～

差し替えも検討されたが据え置きに

→　大虐殺

いろはちゃんコンテスト**Day4**

揺れる街、増える敵 解説

Writer : ミドリムシ

問題概要

- **1**個以上**L**個以下のマスを持った敵がいる
- 敵は**0**個以上のマスを壊されて分裂している
- 敵の強さは分裂したマスそれぞれの長さの積である
- 敵の強さは2^A以上である
- 敵のもともとのマスの個数は何通りあるか
- ただし、敵は**T**体いるのでそれぞれについて答えなさい
- $1 \leqq T \leqq 300$
- $1 \leqq L \leqq 10^9,\ 0 \leqq A \leqq 10^9$

サンプル1

1番目の敵はL=9, A=3

‘x’はマスが破壊済み
‘o’はそうでないことを表す

長さが7 : ‘ooxoooo’ (強さ8)
長さが8 : ‘ooxooxoo’ (強さ8)
長さが9 : ‘oooxoxooo’ (強さ9)

長さは7, 8, 9の3通り

サンプル1

1番目の敵はL=9, A=3

‘x’はマスが破壊済み
‘o’はそうでないことを表す

長さが7 : ‘ooxoooo’ (強さ8)
長さが8 : ‘ooxoooox’ (強さ8)
長さが9 : ‘ooxoooooxx’ (強さ8)

こうすることもできる

考察0

長さが**k**の場合が構築できたら、末尾に**'x'**を入れると
長さが**k+1**の場合も構築できる

よって、長さが**n**未満の場合は構築できず
長さが**n**以上の時は構築できるような整数**n**が存在する

二分探索！

長長　よ長

二分探索

f(k)=長さが**k**の場合に構築できるなら**1**、できないなら**0**

f(k)が求められれば、構築できる最小の長さが二分探索で求まる

f(k)はどうやって求めたらいいだろうか？

考察1

ただし、以降では**’o’**が**a**個並んでいてその両側に**’x’**または端がある場合、**’o’**が並んでいる部分を長さ**a**の連結成分と呼ぶことにする。

考察1

長さ**6**の連結成分はないとしてよい

長さ**a(≧6)**の連結成分があるとすると、
長さ**2**の連結成分と長さ**a-3**の連結成分に分けても弱くはならないので

考察2

敵の長さが1の場合を除いて、
長さ1の連結成分はないとしてよい

その連結成分の横にある’x’を’o’に変えるとより強くなるので

考察3

長さ5の連結成分は1個以下としてよい

長さ5の連結成分2個は長さ3の連結成分3個に置き換えるとより強くなるので
(‘oooooxooooo’ → ‘oooxoooxooo’)

考察4

長さ**2**の連結成分は**1**個以下としてよい

長さ**2**の連結成分**2**個は長さ**5**の連結成分に置き換えるとより強くなるので
(‘ooxoo’ → ‘ooooo’)

考察5

長さ3の連結成分は4個以下としてよい

長さ3の連結成分5個は長さ4の連結成分4個に置き換えるとより強くなるので
(‘oooxoooxoooxoooxooo’ → ‘ooooxooooxooooxooooxoooo’)

まとめ

・長さ1の連結成分は0個としてよい (敵の長さが1の場合を除く)
・長さ2の連結成分は1個以下としてよい
・長さ3の連結成分は4個以下としてよい
・長さ5の連結成分は1個以下としてよい
・長さ6以上の連結成分は0個としてよい

→長さが4以外の連結成分の個数のパターンを全探索し、余った部分にできるだけ長さが4の連結成分を詰め込めばいい

比較

長さ1, 2, 3, 4, 5の連結成分の個数がそれぞれ決まった後、その連結成分の長さの積が2^Aを超えているか判定する必要がある

まず長さ4以外の連結成分の長さの積を求める

これは$2^1 \times 3^4 \times 5^1 = 810$以下なので、int型に余裕でおさまる

長さ4以外の連結成分の長さの積をp、
長さ4の連結成分の個数をc個とすると、
$p \times 4^c$が2^A以上か判定する問題になる

これにはやり方が2種類ある

比較

やり方1

$p \times 4^c \geqq 2^A$を変形すると$p \geqq 2^{A-2c}$になる

pは810以下なので、A-2cが10以上なら構築不可能
逆に、A-2cが0未満なら構築可能
どちらでもない場合は素直に2^{A-2c}を計算する

やり方2

$p \times 4^c \geqq 2^A$の両辺に$\log_2$をつけると$2c+\log_2 p \geqq A$になる

両辺を計算して比較すればOK

Aは最大で10^9になるので、桁数の少ない浮動小数点数を使う場合は誤差に注意

終局

これで**f(k)**を高速に求められるようになった

二分探索をすると、構築できる長さの最小値が求まる

答えは **L - 構築できる長さの最小値 + 1** となる

計算量は…よくわかんないけど**O(TlogL)**に定数倍が**200**倍程度なので間に合うはず

反省

実は、構築できる長さの最小値は**A**から簡単に求まってしまう

・**A**が**0**の場合は**1**
・それ以外の場合は**floor(A×2.5-0.5)**

しかも、この性質は実験で簡単に分かってしまう

(証明は読者への演習課題とする**)**

ACした参加者の多くはこの解法で通したと思いますが、

実はこれ**全く想定してませんでした！！！**

難しめの**600**になるどころか、とんでもない逆詐称になってしまったことをお詫びします。

E – 芽生え

sheyasutaka

Time Limit について

- Writer解は452msでした
- それとほとんど変わらないはずのTester解(C++)が3604ms
  - なんで？

問題概要

- 各要素が0以上K以下の整数である長さNの数列であって，XORが0でないものは何通りか？
- N<=1000
- K<=10^18

- 以下では余事象を取って，XORが0になるものをかぞえる
- また，「K未満」の形に帰着して考える

解説

- i ビット目以上を比べたときに K よりも真に小さくなる要素のことを，「i-自由な要素」と呼ぶことにする
- dp[i][j] = i ビット目以上だけを見たとき，i-自由な要素が j 個となるものは何通りか
- こうおいたとき，dp[i][j]は(i+1)-自由な要素の個数 k をすべて試すことでO(N)で求まる
  - 漸化式は省略

- dpテーブルの大きさは NlogK なので，O(N^2 logK)で解ける

道なき道を　解説

いろはちゃんコンテスト　**DAY4**

問題概要

- 問題文がかなり編集される可能性があるので、元々の問題文で説明します。
  - 元々の問題文で説明する理由はあとがきをお読みください。

- 問題概要
  - コンテストに N 人の参加者がいます。
  - 順位表によると、合計点で i 位の人は、1 問目 A_i 点、2 問目 B_i 点、3 問目 C_i 点を取ったことになっています。
  - 順位表には $3 * N$ 個のマス目がありますが、無矛盾な順位表にするためには最小で何マスを修正する必要がありますか。ただし各問題の満点は B 点です。

問題概要

Rank	P1	P2	P3	合計点
1	100	31	49	180
2	97	37	49	183
3	100	37	49	186
4	100	37	49	186
5	100	17	100	217

Rank	P1	P2	P3	合計点
1	100	37	49	186
2	100	37	49	186
3	100	37	49	186
4	100	37	49	186
5	100	17	49	166

3 箇所修正！

制約・部分点

- $1 \leq N \leq 200000$
- $1 \leq B \leq 10000000000$

- 部分点の制約において
  - $1 \leq N \leq 3000$ (300 点)

考察 1

- 順位表の上から DP していくことを考える

- 考えられる DP の状態として、
  - dp[今順位表の何人目か][最後の人の合計得点] = 最小訂正個数
  - 例えば、$dp[21][186]$ には 21 位となるべき人が実際の合計点 186 点となるような状態の中での最小訂正個数を持つ

考察 1

- さて i 人目の人が P 点を取るためには 3 問の点数のうち何問の点数を訂正する必要があるか？

考察 1

- さて i 人目の人が合計点 P 点を取るためには 3 問の点数のうち何問の点数を訂正する必要があるか？

- 【例】86 + 18 + 36 = 140 点（順位表の情報）の場合
  - 例えば、合計点を 250 点にしたい場合 2 問の点数を修正する必要あり
    - 例： 86 + 100 + 64 など

点数	300-287	286-223	222-141	140	139-54	53-18	17-0
修正個数	3	2	1	0	1	2	3

つまり各問題につき高々 7 個の区間で表される

愚直 DP を書いてみる

- 取り敢えず dp[何人目][直前の人の点数] を状態として持つと、人 i が j 点を取るために何個新たに修正しなければならないかは $O(1)$ で分かるので、

- $dp[i][j] \to dp[i+1][k]$ $(j \geq k)$ の遷移をそのままやると $O(N(3B)^2)$
- $dp[i+1][k] = \min(dp[i][3*B], dp[i][3*B-1], \ldots, dp[i][k])$ であることを利用して累積 **min** を取って遷移を行うと、$O(N*3B)$

部分点解法 (300 点)

- $B \leq 10000000000$
- 考察 1 より、各人が何点を取るのかは高々 7 個の区間で表される

- そのため、座標圧縮をすると B 種類のうち高々 $7N$ 種類の合計点しか考えなくてよい
  - DP の状態数は高々 $3000 * 21000 = 6.3 * 10^7$
  - 余裕で間に合う

満点解法

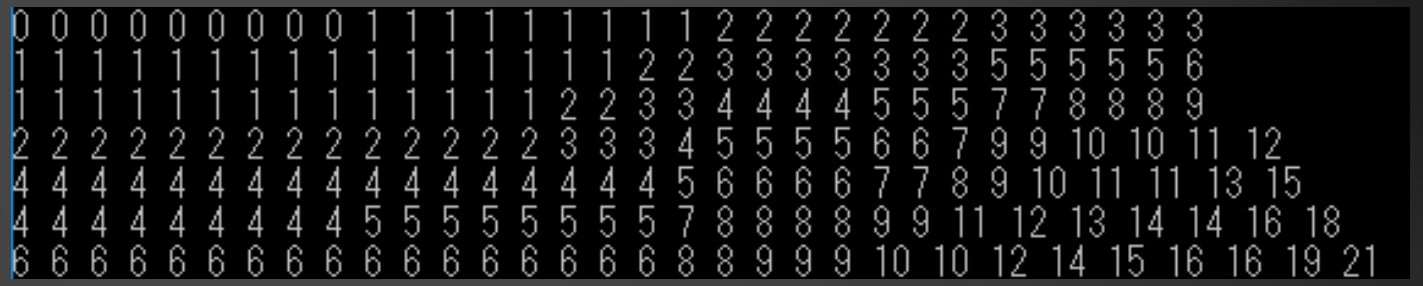
- このまま **DP** の遷移を行うと $O(N^2)$
  - $N \leq 200000$ だと計算時間だけでなくメモリも死んでしまう

- ここで **DP** の遷移を眺めてみることにする
  - 必要なのは DP の累積 **min** なので、**DP** の累積 **min** の遷移を眺める
  - ここでは $r[i][j] = \min(r[i][j], r[i][j+1], \ldots, r[i][3 * B])$ とする

順位表を眺めてみる

- このような場合（$N = 7, B = 10$)

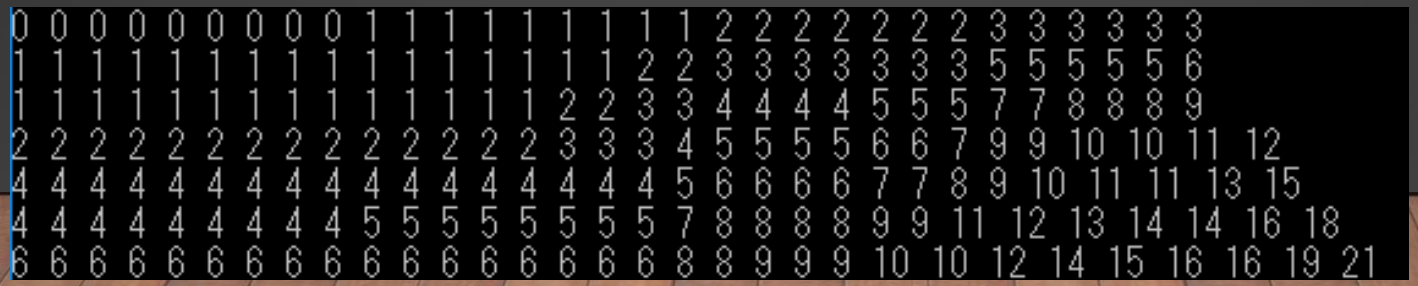
Rank	P1	P2	P3
1	3	1	4
2	1	5	9
3	2	6	5
4	3	5	8
5	9	7	9
6	3	2	3
7	8	4	6



DP の遷移

気づいた事

- $r[i][j] \to r[i+1][j]$ の変化について
  - 0 以上 3 以下の値が足されている気がする
  - ある区間に一定の値が足されている気がする
  - 例えば $r[6][j] \to r[7][j]$ の間の変化だと、
    - [0, 8] だと 2 が足されている
    - [9, 17] だと 1 が足されている
    - [18, 18] だと 0 が足されている
    - [19, 24] だと 1 が足されている（以下略）



考察 2

- 分かった事
  - 実は $r[i] \to r[i+1]$ の変化は区間に分けられており、x が足されている。
  - 分けられている区間は高々 **7** つ

- これは「インライン **DP**」といった **DP** 高速化テクが使えそう
  - $r[i]$ の値を **RangeAdd, RangeMin** などを用いて更新する

満点解法

- 高々 **7** つの区間に対して **RangeAdd** をすれば $r[i] \to r[i+1]$ に遷移できる
  - 最終的な答えは $r[N][0]$

- どの区間に何を足せばいいのかは、二分探索 **RMQ** などを使うと解くことが出来る
  - 二分探索 **RMQ** を使わなくても解ける
  - 実装方針によって計算量が変わるが、$O(NlogN), O(Nlog^2N)$ 両方通る

あとがき (5/3 23:54 追記)

- 僕が解説を書いたのは 5/1 とかなので、その後にストーリー設定と問題文が編集されていましたね。

- Writer の僕からのお願いとして運営はストーリー設定を遅くても 3 日前には完成してほしかったと思いました。
- 解説の訂正が間に合いません。僕はコンテスト当日夜には新幹線に乗っていました。

G-真実の魔法陣(900)

千咲(@Segtree)

はじめに

ーーー

高校二年になった千咲=タプリス=シュガーベルです。

この解説はコンテスト前に書いています。

出揃った問題を見る限り、この問題は飛ばされまくってる未来が見えます。

A~F+(これを飛ばして）Hの7完がたくさんいるんじゃないですか。

たぴゃ...。

意外と解かれてたりするのかな...。

問題

長さ2Nの円環があって、N個のペアを作る。

K個のペアはすでに決まっていて、残りを決める。

何通りの方法で線分の交差を最小化できるかmod10^9+7で求めよ。

1<=k<=N<=300

結論

---

step1.交差を最小化するようなペアの組み方が限定できます

step2.限定した性質を使って区間DPをします

計算量はO(N^3)です

step1

新しくペアにするN-K個の線分どうしは交差しません。

1<=a<b<c<d<=2Nなる整数の組(a,b,c,d)に対して、aとc,bとdがそれぞれ結ばれているとします。

例えば、aとb,cとdを結ぶように変えることで、交差の数が必ず減ります。

二つの線分どうしの交差がなくなるため、交差の数が1減ります。

その他の線分との交差は、減ることはあっても増えることはありません。

線分の端点の場所を全通り試すことで示せます。

step2

ペアが決まっていない点のみを抽出し、座標圧縮して考えます。

前頁の性質を使って、次のようなDPを組みます。

f[l,r):=区間[l,r)の中でペアを完結させたときの、交差の最小値と場合の数

g[l,r):=区間[l,r)の中でペアを完結させた上で、点lと点rが結ばれているときの、交差の最小値と場合の数

として、区間の長さの小さい方から順に求めていきます。

step2

遷移式を簡単にするため、構造体{最小値,場合の数}に対して+と*を定義します。

A+B:=

(A.最小値,A.場合の数+B.場合の数) when A.最小値=B.最小値

(A.最小値,A.場合の数) when A.最小値<B.最小値

(B.最小値,B.場合の数) when B.最小値<A.最小値

A*B:= (A.最小値+B.最小値,A.場合の数*B.場合の数)

step2

---

f[x,x)は乗算の単位元(0,1)に、それ以外のfは加算の単位元(inf,hoge)に初期化しておきます。g[x,x)は使わないのでどうでもいいです。

g[l,r)|l+2<=r は、最小値がf[l+1,r-1)にlとrを結ぶ線分が交差する既存の線分の個数を足したものになり、場合の数はf[l+1,r-1)そのままです。

g[l,r)の値を計算した後で、f[l,r)|l+2<=rに対して以下の遷移を行います。

・for(k=l; k<=r-2 ; k+=2){ f[l,r) += f[l,k) * g[k,r); }

最終的な f[0,2*(n-k)).場合の数 の値が答えです。

step2

---

gはそれぞれのペアの組み方の唯一性を保ちつつ、すべてを正しくfに遷移させるための補助配列です。

すべての場合が遷移されていることは明らかです。重複がないことを示します。

ある区間[l,r)内で完結したペアの組み方であって、複数の方法でf[l,r)に遷移できるものは存在しません。なぜなら、f[l,r)に遷移するためにはg[k,r)を用いる必要がありますが、利用できるkは点r-1とペアにした点のみだからです。

この事実から、重複がないことが帰納的に示せます。

まとめ

---

f[l,r)、g[l,r)の値を区間の長さの小さい方から順に計算していきます。

gの値ひとつに対して、元からあった区間との交差をカウントするのにO(N)、

fの値ひとつに対して、kをlからr-2まで決め打つのにO(N)かかるため、全体でO(N^3)になります。

永遠に(原題:Not Be An Average) 解説

physics0523

問題概要

N*Nのマス目に1~N*Nまでの数を1つづつ書き込む

同じ行または列の部分列に長さ3の等差数列が入ってはいけない

即ち、同じ行または列で好きな2マスを選んで、その間に選んだ2つのマスの数の平均値があったらダメ(∵等差中項の公式、a,b,cがこの順で等差数列の時、a+c=2b、変形してb=(a+c)/2)

3≦N≦300

2次元…の前に1次元

問題では2次元になっている

いきなりこのまま考えてもよいが、1次元で類題を考えられそうなので、次元を落としてみる

2次元以上の問題では、いきなり2次元版を考えるのではなく、1次元に落としてみるのも手

1次元版問題

1~NまでのN個の数を一列に並べる

列のうち好きな2数を選んで、その間に選んだ2数の平均があったらダメ

N=5で

１４２５３ はNG(1と3の間に2がある)

１５３２４ はOK

着想

例えば、
1と3の間…平均が2なので間に2を入れちゃダメ
5と11の間…平均が8なので間に8を入れちゃダメ
2と7の間…平均が4.5なので間に4.5を入れちゃダメ

…4.5???

偶奇分け

ある要素A_iとA_j(i<j)の偶奇が異なれば、この間には何がきてもよい!!!

つまり、左に奇数、右に偶数を寄せる、などすると偶数と奇数の間で問題が起こらないようにできる!

N=8の時の初手で、

１３５７２４６８

とすると7と2の間をまたがって問題が起こらないようにできる

そこからどうするか?

1 2 3 4 5 6 7 8 ->

1 3 5 7 | 2 4 6 8

この数列の右側を考える

ここで、右側の要素を全て2で割ったものを考えると、1~4の数列となり、4数が1 2 3 4である場合に帰着できる(参考までに、そのまま考えるなら、mod4で0or2で左右に分ける)

左側も1を足して2で割ると同様

あとは再帰的に

あとは再帰的に先ほどの問題を分割した列ごとに解くと、分割した数列の長さが1になったタイミングで問題が解ける

1 2 3 4 5 6 7 8

1 3 5 7 | 2 4 6 8

1 5 | 3 7 | 2 6 | 4 8

1 | 5 | 3 | 7 | 2 | 6 | 4 | 8

これで1次元版は解けた

ちなみに

再帰を実装せずに、一次元版の数列を直接求める方法がある

どうするか?

まず、(一旦数列の中に0を含めて)2進表記と最初の偶奇分けを考える

0...000(2) 2...010(2) 4...100(2) 6...110(2)

1...001(2) 3...011(2) 5...101(2) 7...111(2)

これは何をやっているか?

末尾bitが0の数を前、1の数を後に持ってきている

ここから再帰的に考える

0,2,4,6は2で割って0,1,2,3に帰着できる話はさっきした

これを今度は末尾から2番目のbitが0の数を前、1の数を後に持ってくる

0...000(2) 4...100(2) 2...010(2) 6...110(2)

1,3,5,7も同様

すると

解となる数列は以下のような条件を満たすことになる

・末尾bitが0なら前、1なら後

・末尾が同じなら末尾から2bit目が0なら前、1なら後…

・つまり、数列の要素のbitを反転させると辞書順に並ぶ

これは何を意味するか?

bitを反転させた数列が辞書順→その数列は0,1,2,...

逆に0,1,2,...の数列のbitを反転させれば適する列ができる

1次元版の解法

まず、0~511までの9bitの数の数列の要素のbitを反転させる

その後、必要に応じて反転させた後の1~Nまでを取り出した部分列(不連続でもok)を取り出す

すると、長さNの場合の解が得られる(数列全体で条件を満たすので、部分列も当然条件を満たす)

具体的に(3bit,N=5の場合)

0(000),1(001),2(010),3(011),4(100),5(101),6(110),7(111)

bitを反転させると

0(000),4(100),2(010),6(110),1(001),5(101),3(011),7(111)

1≦Ai≦5を満たす部分列を取り出して

４２１５３

これで1次元版は解けた

2次元へ

実はここまでは1次元の問題で、元ネタがあったりする
これをどう2次元に持ち上げるか?

注意しておくこと

1次元版で条件を満たす数列$\{A_i\}$が条件を満たすなら、

A_iの要素を全てk(k≠0)倍したものも条件を満たす

なぜなら、任意の2要素間においてその2要素の平均もk倍されれば間の要素もすべてk倍されるから

同様に、A_iの要素全てに定数Cを加えたものも条件を満たす

一般に、$B_i=kA_i+C(k\neq0)$と書き表されるとき、

「$\{A_i\}$が条件を満たす」⇔「$\{B_i\}$が条件を満たす」

2次元版構築法

結論から言うと、N*Nを構築したいとき、$Ans_{(i,j)}=N(A_i-1)+A_j$でよい(但しAは長さNの1次元版の解)

なぜなら、

縦列をみると$B_i=NA_i+(A_j-N)$(A_jは縦方向にとっては定数)

横列をみると$B_i=A_j+(N(A_i-1))$((N(A_i-1)は横方向にとっては定数)

が成り立っているから

解けた!

これでこの問題は解けました(計算量はO(N^2)です)

ちなみにNを限定すると別の方針もあります(もともと2次元版への拡張のアイデアが「Nを1つ選んで解く」でした)

N=256だと正方形を2*2に分割して左上と右下を最下位bit0,

右上と左下を最下位bit1...という感じで再帰的にやっても解けます

Judge Ver.はO(N^3)で出来ます(考えてみてください)

FA、正解率

FA:zundamochi_1117(30:48)

AC/Trying

14/24(58.3%)

1 – ピンチ

sheyasutaka

問題概要

- 完全グラフの各辺に適当に向きをつけたものがある
  - *“Tournament graph”* というらしいです
  - 向きは与えられない
  - $N \leq 800$
- 辺の activate/deactivate が 666666 回までできる
  - *active*な辺は常に $2N$ 本以下でなければならない
- activeな辺のみをたどってuからvにいけるかの判定が 10000 回までできる

- 最も長いパスを一つ答えよ

実は

- 常に長さ N のパスが構築できる
- 帰納的にやってみよう

頂点 1~$(k-1)$ からなるパスがあるとき

- パス上の頂点の並びを始点から順に$P_1 \sim P_{k-1}$ とおく
- 辺「$k \to P_1$」か辺「$P_{k-1} \to k$」があったらそれをつなげる
- 両方なかったときは？

頂点 1~$(k-1)$ からなるパスがあるとき

- パス上の頂点の並びを始点から順に$P_1 \sim P_{k-1}$ とおく
- 辺「$k \to P_1$」か辺「$P_{k-1} \to k$」があったらそれをつなげる
- 両方なかったときは？

- 辺「$P_x \to k$」がある最大の番号 $x < k$ があって，
  x は $k-1$ でなく辺「$k \to P_{x+1}$」がある
- なので，「$P_x \to P_{x+1}$」を「$P_x \to k \to P_{x+1}$」に張り替えればよい

頂点 1~$(k-1)$ からなるパスがあるとき

- パス上の頂点の並びを始点から順に$P_1 \sim P_{k-1}$ とおく
- 辺「$k \to P_1$」か辺「$P_{k-1} \to k$」があったらそれをつなげる
  - これはやる

- 辺「$P_x \to k$」がある最大の番号 $x < k$ があって,
  x は $k-1$ でなく辺「$k \to P_{x+1}$」がある
- なので,「$P_x \to P_{x+1}$」を「$P_x \to k \to P_{x+1}$」に張り替えればよい
  - これは「X~$(k-1)$は全部 $k \to hoge$ の向き」の条件で二分探索をする.

頂点 1~$(k-1)$ からなるパスがあるとき

- パス上の頂点の並びを始点から順に$P_1 \sim P_{k-1}$ とおく
- 辺「$k \to P_1$」か辺「$P_{k-1} \to k$」があったらそれをつなげる
  - これはやる

- 辺「$P_x \to k$」がある最大の番号 $x < k$ があって,
  x は $k-1$ でなく辺「$k \to P_{x+1}$」がある
- なので,「$P_x \to P_{x+1}$」を「$P_x \to k \to P_{x+1}$」に張り替えればよい
  - これは「X~$(k-1)$は全部 $k \to hoge$ の向き」の条件で二分探索をする.

- トグルと検定はそれぞれ最悪でも $N^2 + 4N$ 回, $N \log N + 2N$ 回なので OK

実は

- 乱択だともっとずっと少ないトグル数で解ける(Tester解)
- ランダムな頂点 v を選び,
  v との間の辺が「v から出る向き」か「v に入る向き」かで残りの頂点を分類
- 2つに分けたそれぞれに関して再帰的にハミルトンパスを構築してから,
  [そいつら] → v → [あいつら] の形に辺を張りなおす
- Tournament graph の次数の出入はそんなに偏らないため,
  トグルも検定も $O(N \log N)$で解ける

いろはちゃんコンテスト Day4 J - も - もう、諦めない 解説

ぽかいこ

パワーポイントで書いていたのですけど、議論が重すぎたので TEX で書くことにしました (悪魔的)

1　部分点

不偏ゲームなので grundy です いつもの おかえり

関数 $imp(a,b)$ を辺長の組から grundy 数を返す関数として定義します。 imp は impartial game の頭文字です。ざっくり言うと

$$imp(a,b) = \begin{cases} (a \text{ の奇数の素因数の重複を含めた個数}) \text{ xor } (b \text{ の奇数の素因数の重複を含めた個数}) & (ab \text{ is odd}) \\ 1 + (a \text{ の奇数の素因数の重複を含めた個数}) \text{ xor } (b \text{ の奇数の素因数の重複を含めた個数}) & (ab \text{ is even}) \end{cases}$$

になります。偶数個に分割したらそれぞれの部分の xor が消えて 0 になるため、2 を素因数としてどちらかが持つ場合常に 1 つ選択肢が増えている状況になります。奇素因数しか持たない場合、a と b のどちらかを選んで奇素因数の個数を 1 つ以上減らすことになるのでこれは 2 山の Nim と同じです。

これで部分点が解けました。

2　非不偏ゲーム：特に、どちらも決められた方向にしか切れない場合

まず、どちらのプレイヤーも片方にしか切ることができない場合を考えます。この場合も、ゲームの状態を整数に変換する関数を考えることにしましょう。

関数 $p(a,b)$ を辺長の組から整数を返す関数として定義します。この関数が満たすべき条件は何でしょうか。次のようなゲームを考えます。

> A さんの前に石が a 個、B さんの前に石が b 個あります。
> 2 人は交互に次の操作をすることができます。
>
> - 目の前にある石の山から、1 個以上の石を選んで捨てる。
>
> 操作ができなくなったほうの勝ちです。

これはニムに似たゲームですが、不偏ゲームではないです。grundy 数と同様にこの問題に帰着することにしましょう。

このゲームに整数を対応させるなら、$a-b$ を対応させるのが自然です。そして、最初の状態でこの値が 0 ならば後手必勝、正ならば A さんの勝ち、負ならば B さんの勝ちであることがただちにわかります。

ゲームの状態 G について、そこから A さん (あなた、右) が手を打つことで移行することができるゲームの状態の集合を G_R、B さん (相手、左) が手を打つことで移行することができるゲームの状態の集合を G_L としましょう。非不偏ゲームを整数に対応させる関数 $hoge$ が持っているとうれしい性質を以下に示します。

(i) $hoge(G)$ は、後手必勝なら 0、A さんの勝ちなら正、B さんの勝ちなら負である。逆も成り立つ。
(ii) $\forall g \in G_R.hoge(g) < hoge(G)$
(iii) $\forall g \in G_L.hoge(g) > hoge(G)$
(iv) $hoge(G) > 0$ なら、$\max_{g \in G_R} hoge(g) + 1 = hoge(G)$
(v) $hoge(G) < 0$ なら、$\min_{g \in G_L} hoge(g) - 1 = hoge(G)$

これが、先ほど言った石のゲームの値と対応しているのは明らかです。ところで、Nim のゲームの合併は xor でした。では、この非不偏石取りゲームの合併は何でしょう？

これは当然単純な和です (山が増えると残機が増えるため)。以上より、上の 5 条件を満たすような関数 p を作れたら、各長方形について値を合計するだけなので、(どちらも片方にしか切れない場合については) 終了であることがわかります。

説明を省いてしまいます (今後増やす予定です) が、次のような p が条件を満たします。

$$t(a,b,c,d,\ldots,e) = 1 + a + ab + \cdots + abcd\ldots e$$

$$p(a,b) = (a \text{ より } b \text{ のほうが重複を含めた素因数の個数が多いとき } (-1) \text{ otherwise } 1)$$

$$\times t(a,b \text{ のうち重複を含めた素因数の個数が多いほうの素因数を、多い分だけ降順にならべたもの})$$

例えば、$p(2,36) = (-1) \times t(3,3,2) = (-1) \times (1+3+3\times3+3\times3\times2) = -31$ などとなります。これで、どちらも片方にしか切れない場合については解くことができました

3 満点への道

まず、ゲームの状態と整数と非負整数の組を対応させるような関数 F を考えます。これは、ゲームを非不偏成分と不偏成分に分離するものと考えられます。なので、次のような性質を満たしていてほしいです。

$$F((0,0,a,b)) = (p(a,b),0) \qquad F((1,1,a,b)) = (0, imp(a,b))$$

$$F(G_1) = (a,b), F(G_2) = (c,d) \text{ とすると、 } F(G_1 + G_2) = (a+c, b \text{ xor } d)$$

このとき (証明はあとで書きますが)、

$$F((1,0,a,b)) = F((1,1,1,b)) + F((0,0,a,1)) \times b$$

が成立します。ここで組 (α,β) と正整数 b の積は、(α,β) を b 回合併するものとして定義します。$F((0,1,a,b)) = F((1,1,a,1)) + F((0,0,1,b)) \times a$ も同様に成り立つため、すべての長方形について組を対応させることができました。

ここで、組の合併演算は群 $(\mathbb{Z},+),(\mathbb{Z}_{\geq 0},\text{xor})$ の直積なので、群をなします。

よって、先頭からの累積和を取っておくことで、区間 $[l,r]$ 内のゲーム全体に対応する組を $\mathcal{O}(1)$ で求めることができます。

ところで、勝敗判定はどのようにするのでしょうか？ これは Nim と非不偏石取りゲームの合併を考えると明らかで、組 (a,b) は、

$$(a,b) = \begin{cases} \text{A さんの勝ち} & a > 0 \\ \text{後手必勝} & a = b = 0 \\ \text{先手必勝} & a = 0, b \neq 0 \\ \text{B さんの勝ち} & a < 0 \end{cases}$$

です。よって、この問題を前計算に素因数列挙 $\mathcal{O}(N \log\log N)$、クエリに $\mathcal{O}(1)$ の計算量で答えることができました。

K-世界線(1300)

ynymxiaolongbao (@Segtree)

はじめに

高校二年になった千咲=タプリス=シュガーベルです。

この解説はコンテスト前に書いています。

この問題の予想AC者数は6人です。当たりましたか？

ところでこの問題見た目がすごく綺麗じゃないですか？

もともとかなり複雑な想定解だったんですが、testerのkobaeさんが簡潔な解法を示してくださいました。配点を詐称してしまった感じです。

1300点には詐称が多いので許してください。

問題

下の操作を繰り返し行って、Nだけの列から目標の順列を作る方法が何通りあるかmod10^9+7で求めよ

・要素をひとつ選んで、それとそのひとつ右の間に、その値より小さい数を挿入する

1<=N<=2*10^5

結論

---

step1.操作について考えます

step2.操作の順番に対して、不等式による制約がかかっている形になります

天才-step3.この制約は独立で、掛け算割り算するだけで答えがわかります

愚者-step3.この制約はよい性質を持っていて、木を頂点とした森ができます

愚者-step4.木の末端から場合の数を調べてマージしていくと答えになります

計算量はO(N)です

step1

数を挿入するのではなく、完成した状態から数をなくしていくふうに捉えます。

操作xを、p[x+1]を消す操作だとします。

操作1,2,..,n-1がそれぞれ一回ずつ行われるはずです。

操作xを行うためには、p[x+1]のすぐ左にそれより大きなものがあるはずです。

この条件を満たす操作の順番の個数が答えです。

step2

操作xを行うためには、p[x+1]のすぐ左にそれより大きなものがあるはずです。

このことは、y<=x+1 かつ p[y]>p[x+1] なる最大の整数yに対して、操作y,y+1,..,x-1が既に行われていることと等しいです。

操作y-1が行われていないなら、p[x+1]の左隣にp[y]があり、OKです。

操作y-1が行われているとき、yから左に向かってはじめてp[y]を越えるところまでの操作は既に行われています。この事実を使って、p[x+1]の左隣にp[x+1]より大きな数があることが帰納的に示せます。

天才-step3

実はこれらの制約はそれぞれ独立であり、(n-1)!をそれぞれのxに対してx-y+1で割っていけば答えになります。

二つの区間[y1,x1)と[y2,x2)が重なることはありません。これは背理法で示すことができます。

よって、区間の長さの小さい方から制約を実行していくと、制約どうしを影響させることなくすべての制約を実行できます。

stackなどでxとyの関係を求めれば、計算量はO(N)です。

天才か？

ふえぇ

---

kobaeさんのコードを見たときすごく焦った。

これ実は良問だったのでは？

独立性のお勉強をしなくちゃ。

凡人には厳しい。

もともとは実家みたいな感じの想定解だった。

これを下にのせておく。

愚者-step3

この不等式たちはよい性質を持っています。

xからy（前頁参照）に辺を張ったグラフを考えます。

このグラフは、深さ２の根付き木（ただし、根には自己辺が張られているため厳密には木ではない）がたくさん集まったかたちをしています。

p[x]>p[x+1]であれば自己辺となり、xは根となります。
p[x]<p[x+1]であれば、xの代わりにx+1に辺が張られるため、親になり得ません。自己辺にもならないのでxは葉となります。

愚者-step3

前頁で示した深さ２の根付き木をクサと呼ぶことにします。
クサどうしの間にもよい性質があります。

直感的には、クサどうしが交差することはありません。
厳密には、1<=a<b<c<d<=N-1なる整数の組(a,b,c,d)に対して、aとbが異なるクサに属しており、aとcが同じクサに属しており、bとdが同じクサに属しているということはありません。

p[a],p[b],p[c],p[d]の大小関係をすべて試すことで示せます。

愚者-step4

交差しない区間の集合が根付き木のかたちになるのは有名です。

今回は、クサの左端（左端は必ず根になります）と右端をそれぞれ区間の左端、右端として、クサの集合を根付き木がいくつか集まったものとして捉えます。

それぞれの根付き木で、末端から順番に、そのクサが被覆する区間内の操作の順番の個数を求めていくことを考えます。

そのクサの頂点（操作）は、左端から順に行われていきます。

そのクサが被覆する頂点たちは、すぐ右のそのクサの頂点より先に行われます。

愚者-step4

前頁の問題は、左側から重複組み合わせなどを使っていけば解けます。

そのクサの頂点たちとそのクサの子たちを混ぜて、頂点の番号でソートします。

クサが交差しないことから、子のクサはどの頂点番号にしても変わりません。

既に見た頂点の個数をtとして、頂点番号の昇順に以下の操作を行います。

そのクサの頂点のとき、tに1を足します。

子のクサのとき、uを子のクサの被覆する頂点数として、tにuを足し、答えに$_tC_u$と子のクサの答えを掛けます。

愚者-まとめ

与えられた順列から、stackなどを使ってクサの構造を明らかにします。

クサの根付き木の末端から順に、コンビネーションを使って答えを求めます。

クサの根付き木それぞれの答えを掛け合わせ、s[i]をi番目のクサの根付き木の被覆する頂点の数として、(sの合計=N)!/s[1]!/s[2]!/../s[m]!を掛ければそれが答えになります。

適切に実装すればO(N)の計算で答えが求まり、十分高速です。

...好きです　解説

いろはちゃんコンテスト DAY4 ～BOSSRUSH～

はじめに

- この問題は…

はじめに

- この問題は…

BossRush のボス

はじめに

- この問題の作問者は **E869120 (79%) + square (21%)** です。
- 私はひらきちにこの問題を出したら **1** 週間考えて解法が分からなかったぽかったので **BossRush** の最後に置かれました。

- でも意外と解いている人は多そうなのですね。（コンテスト前の予想）
  - **Codeforces** に慣れている人は解けやすそう

問題概要

- いろはちゃんは以下の 3 つの操作をします。
  - **追加の術**：座標 x に v 円のコインを置く。
  - **消去の術**：座標 x にあるコインを削除する。
  - **命令の術**：座標 x にひらきちくんを置く。ひらきちくんがコインを 1 つ取る時の「効率」の最大値を求める。ただし「効率」とは、コインの価値をひらきちくんとコインの距離で割ったものである。

- なんだか**命令の術**が一番強そうですよね…（小並感）

問題概要

- $N, Q \leq 300000$
- 実行時間制限 3 秒

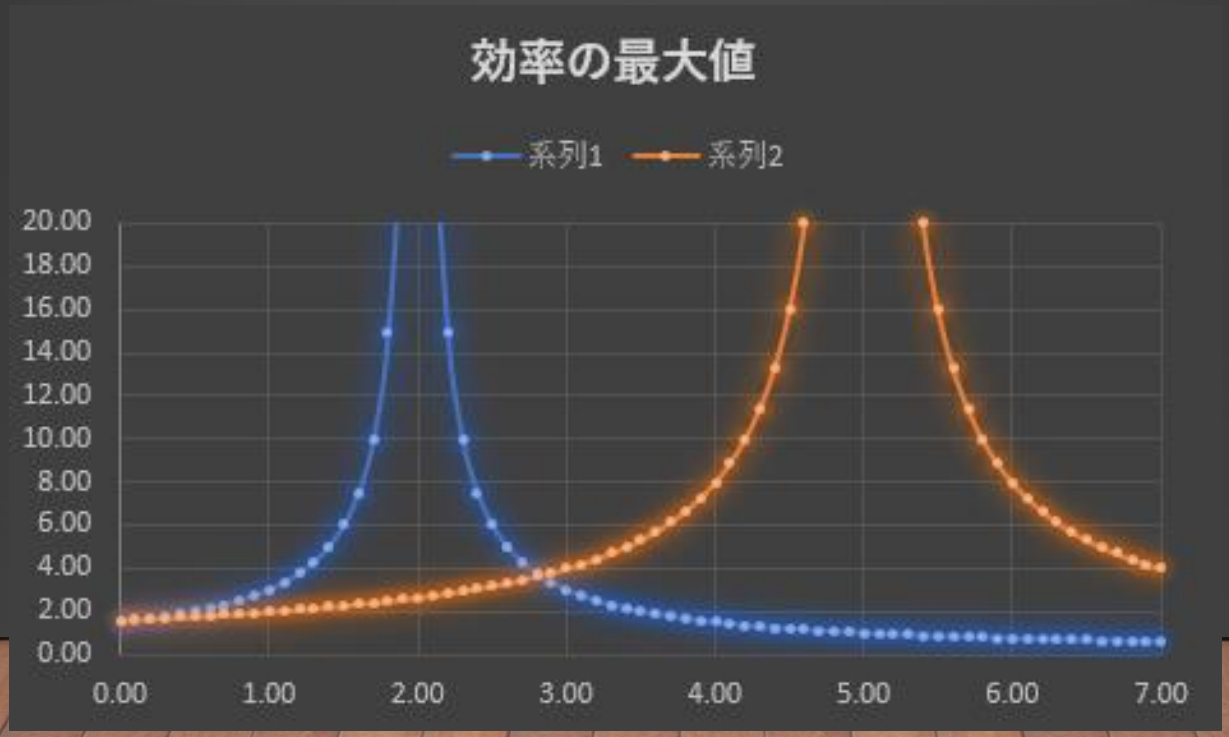
- まず愚直解の $O(NQ)$ は絶対間に合わなさそう
  - 浮動小数点の演算が必要なのにもかかわらず、$9.0 * 10^{10}$ あるいは $2.3 * 10^{10}$ 回の計算が必要

考察 1

- 価値の最大値をグラフで表したい！
- 例えば $(x, v) = (2, 3), (5, 8)$ にコインがある場合…

考察 1

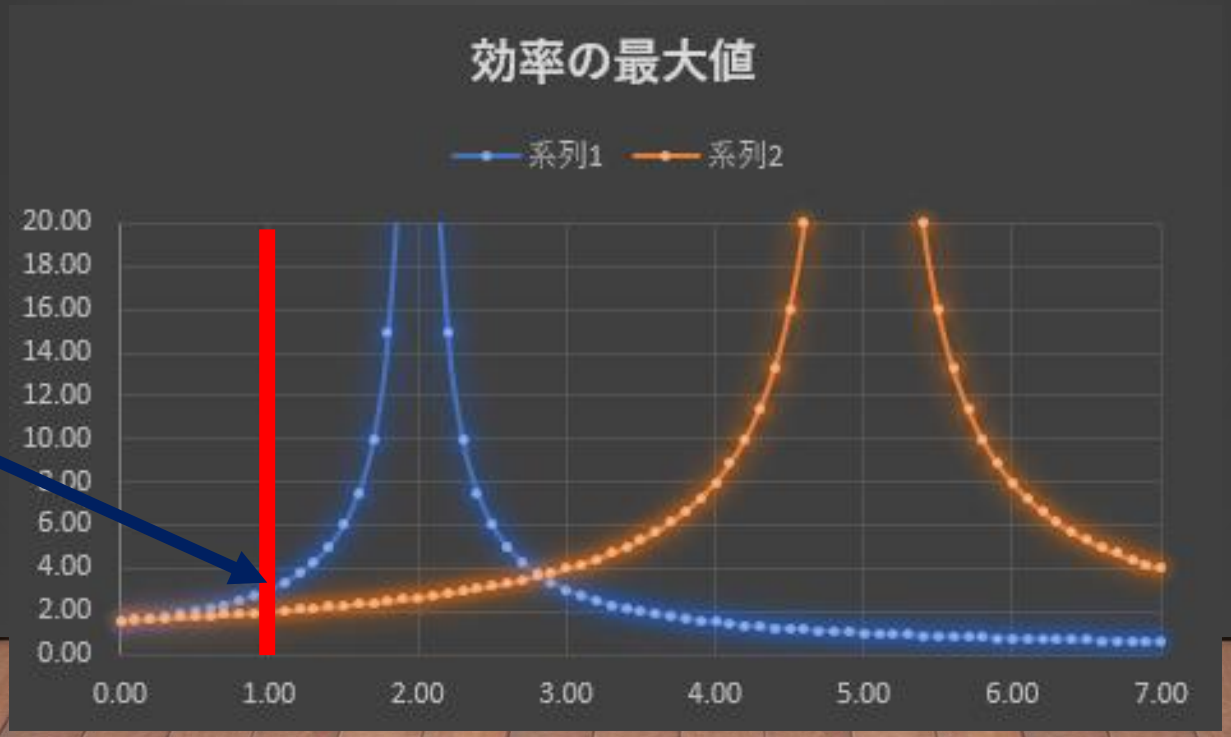
- 効率の最大値をグラフで表したい！
- 例えば $(x, v) = (2, 3), (5, 8)$ にコインがある場合…



考察 1

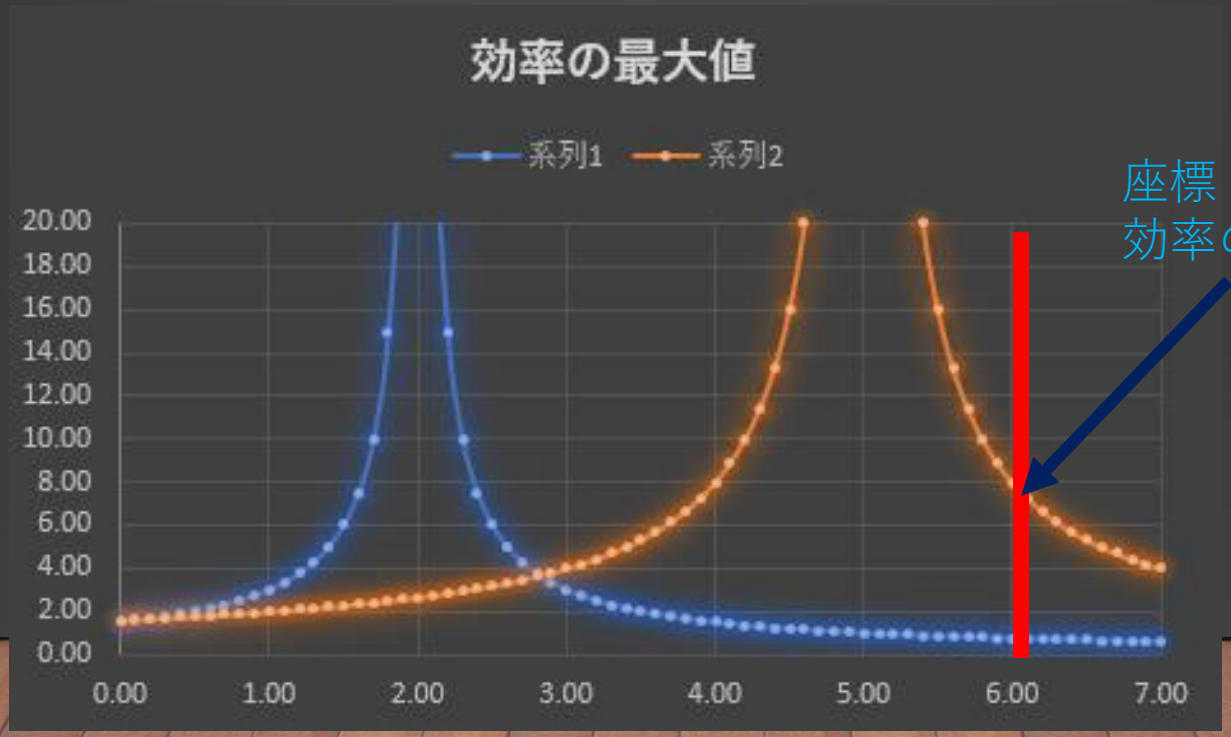
- 効率の最大値をグラフで表したい！
- 例えば $(x, v) = (2, 3), (5, 8)$ にコインがある場合…

座標 1 における
効率の最大値は 3



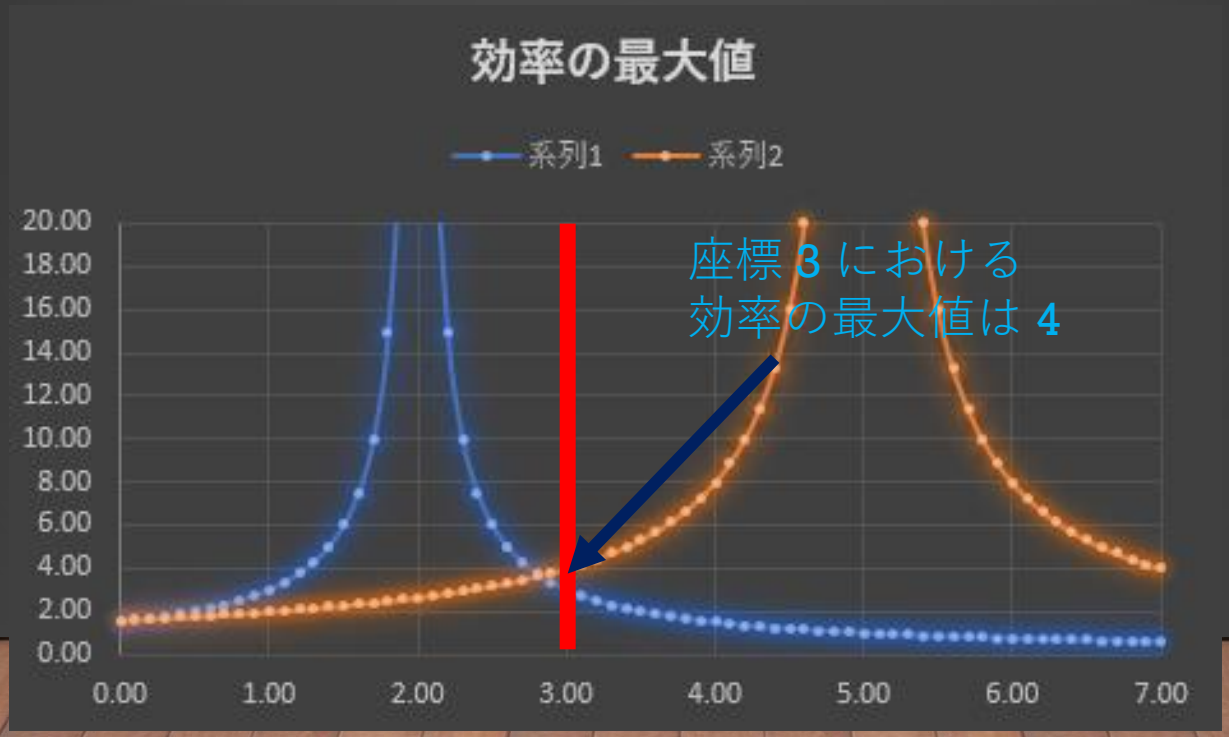
考察 1

- 効率の最大値をグラフで表したい！
- 例えば $(x, v) = (2, 3), (5, 8)$ にコインがある場合…



考察 1

- 効率の最大値をグラフで表したい！
- 例えば $(x, v) = (2, 3), (5, 8)$ にコインがある場合…



考察 1

- つまり以下のような操作を行うことが出来ればよい
  - ① 反比例のグラフを追加する
  - ② 反比例のグラフを削除する
  - ③ ある x 座標における y 座標の最大を求める

- グラフを追加するのは **ConvexHullTrick** と似たクエリだが、直線じゃないので上手くいかなさそう…（しかも削除もある…）
  - **ConvexHullTrick (CHT)** が分からない人は検索するとすぐに出てきます。

考察 2

- 反比例のグラフをどうにかして直線にしたい
  - どうやったら直線にできるのか…？
  - 答えは次のスライド

考察 2

- 反比例のグラフをどうにかして直線にしたい
  - どうやったら直線にできるのか…？
  - 答えは次のスライド

逆数

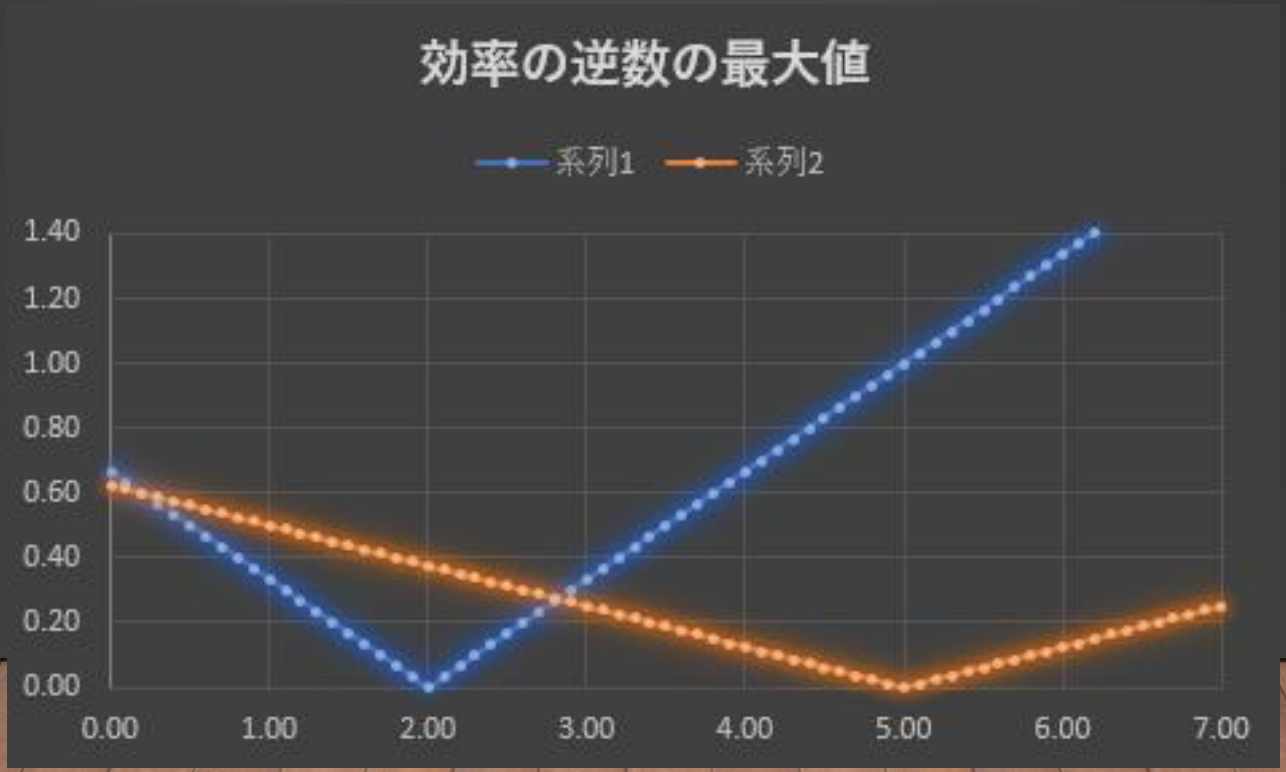
考察 2

- 「効率の最大値」＝「効率の逆数の最小値」
  - 「価値を距離で割った値の最大値」は「距離を価値で割った値の最小値」と等価

- 「距離を価値で割った値」のグラフは直線になりそう
  - $\frac{c-x}{v}$ (x はひらきちくんの座標、c はコインの座標、v はコインの価値)
  - 直線になります！！！！！

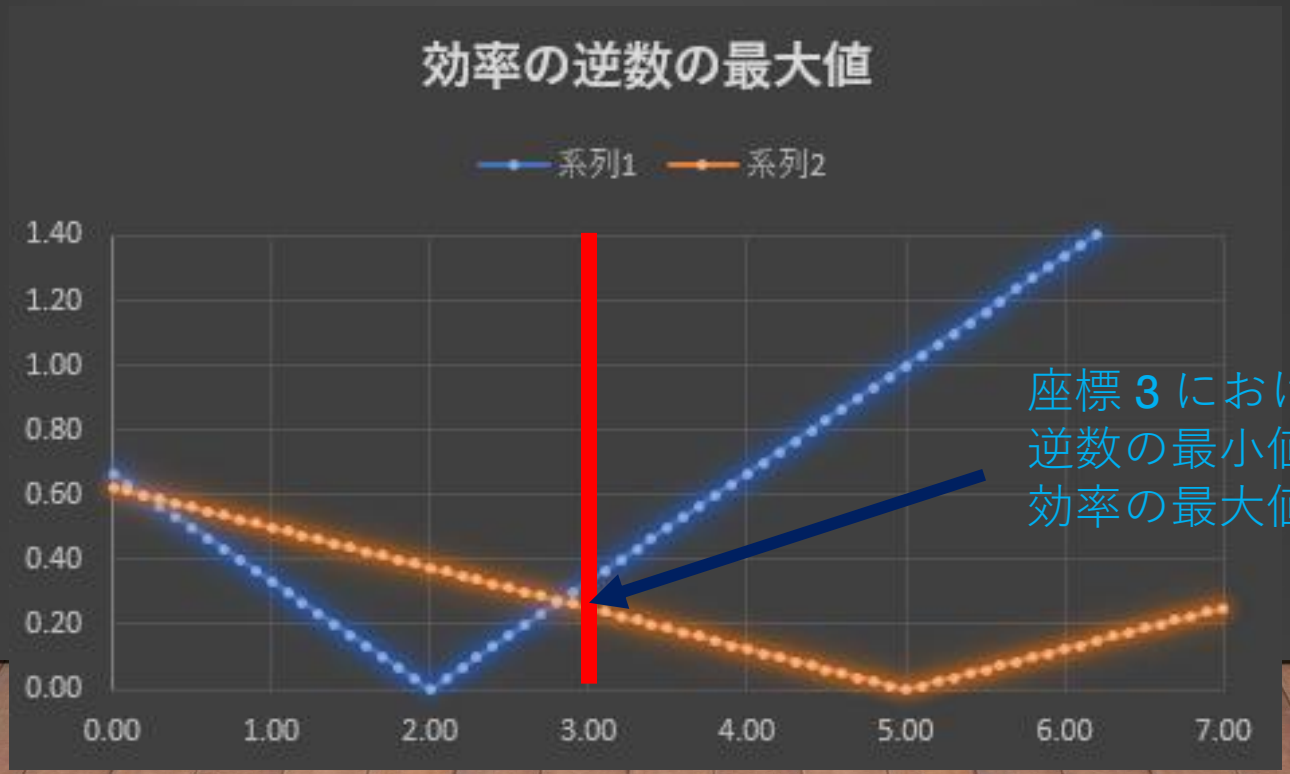
考察 2

- 効率の逆数の最小値をグラフで表したい！
- 例えば $(x, v) = (2, 3), (5, 8)$ にコインがある場合…



考察 2

- 効率の逆数の最小値をグラフで表したい！
- 例えば $(x, v) = (2, 3), (5, 8)$ にコインがある場合…



考察 2

- 以下のようなクエリになった
  - ① 半直線を追加する
  - ② 半直線を削除する
  - ③ ある座標における最小値を求める

- 削除があるので色々と面倒な気がするが…
  - 取り敢えず削除を消すことを考える

考察 2

- 以下のようなクエリになった
  - ① 半直線を追加する
  - ~~② 半直線を削除する~~
  - ③ ある座標における最小値を求める

- これは **Li Chao Tree** というデータ構造を使えば $O(\log^2 N)$ で出来る
  - **Li Chao Tree** は http://smijake3.hatenablog.com/entry/2018/06/16/144548 を参照

実装方針① ～平方分割～

- 削除をどう実装する？
  - 考えられる実装方針の一つとして、クエリ平方分割

- クエリ平方分割では、$[l, r]$ 番目のクエリについて答えを一気に求める感じでやればよい（$r - l = sqrt(N)$ くらい）

実装方針① ～平方分割～

- クエリ平方分割をすると、何が起こる？

- $[l, r]$ 番目に新たに追加／削除される直線に関しては、各命令の術において線形探索をすると上手くいく（そのような直線は高々 $O(r - l)$ 個である）
- l 番目のクエリまでに追加された直線であり、 r 番目のクエリまで残るものは、削除なしの **Li Chao Tree** を使えば最小値が求まる
  - $[l, r]$ 番目を一気に求めるときの計算量は、$O(Q \log^2 Q)$

実装方針① ～平方分割～

- $r-l=B$ と置くときの計算量
  - **Li Chao Tree**： $O(\frac{Q}{B}*Qlog^2Q)$
  - 線形探索： $O(QB)$

- よって、計算量は $O(QB+\frac{Q}{B}*Qlog^2Q)$
  - $B=\sqrt{Q}*logQ$ くらいにすると最適で、全体計算量 $O(Q\sqrt{Q}logQ)$ くらいになる
  - 定数倍が速ければ **1100** 点（**writer** 解だと部分点で **1065ms** くらい）

1100 点

実装方針② ～永続化～

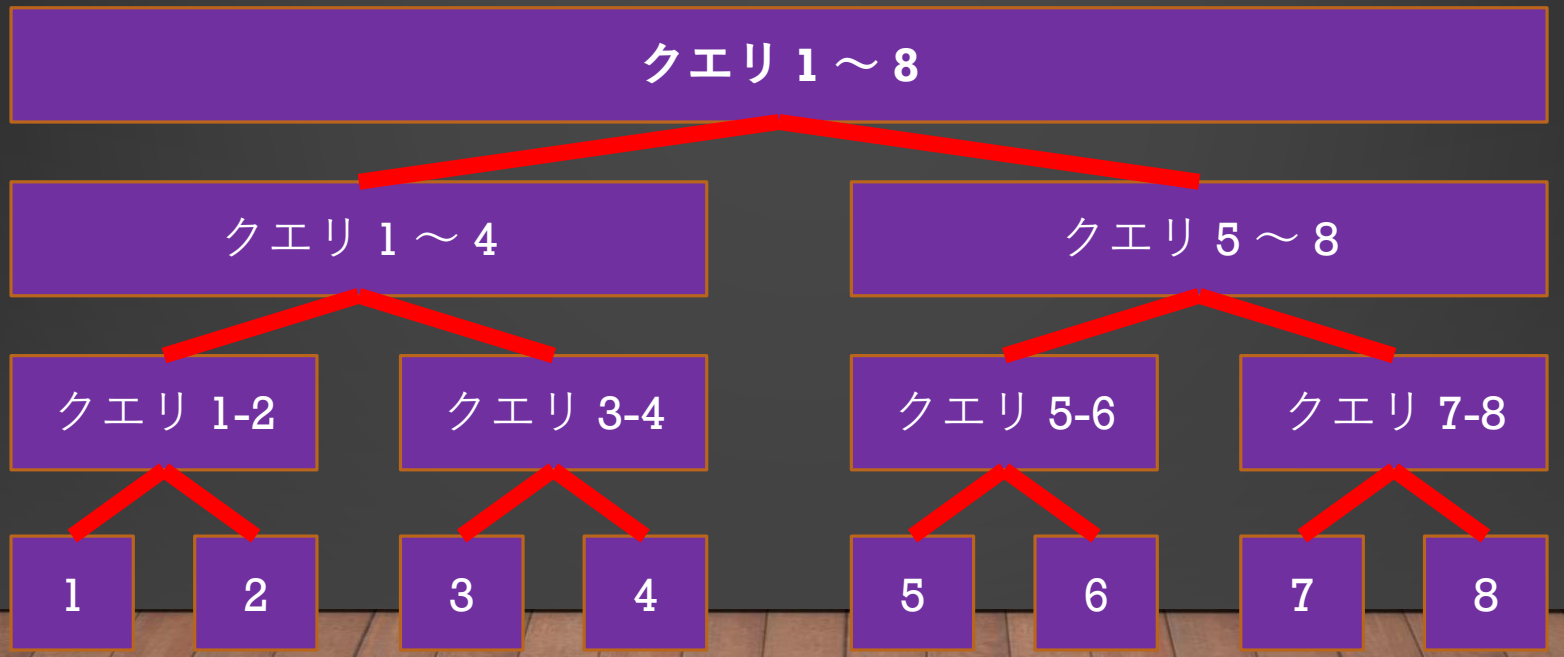
- **Li Chao Tree** を永続化して削除有りにすると上手くいくらしい
  - $O(Qlog^3Q)$ で出来る

- しかし、定数倍が重いため、満点をこの解法で通すのは想定されていない
  - 部分点が通れば、あなたのプログラムは定数倍が速いですと言えるくらい

1100 点

実装方針③ ～セグ木に載せる～

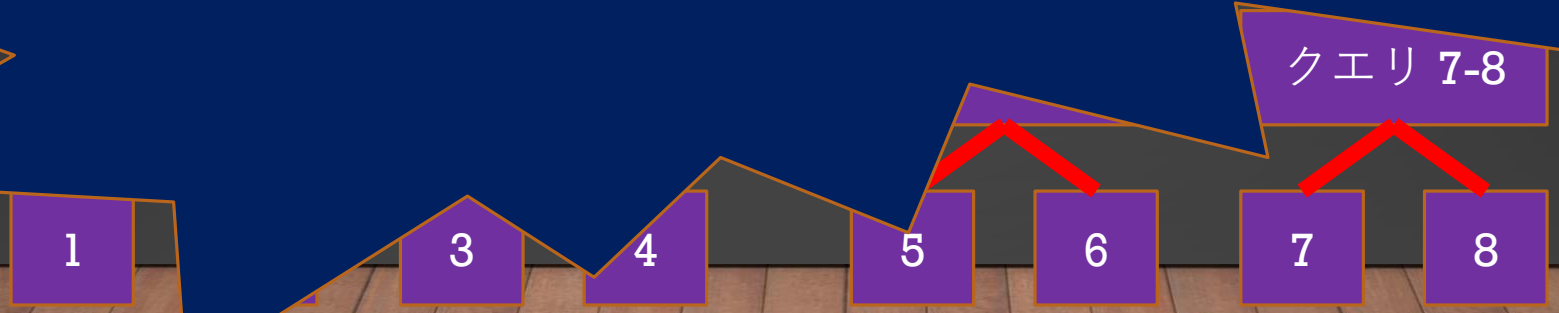
- もう少し計算量を改善するために、セグ木を持つこともできる
  - セグ木のノードをクエリにすることを考える



実装方針③ ～セグ木に載せる～

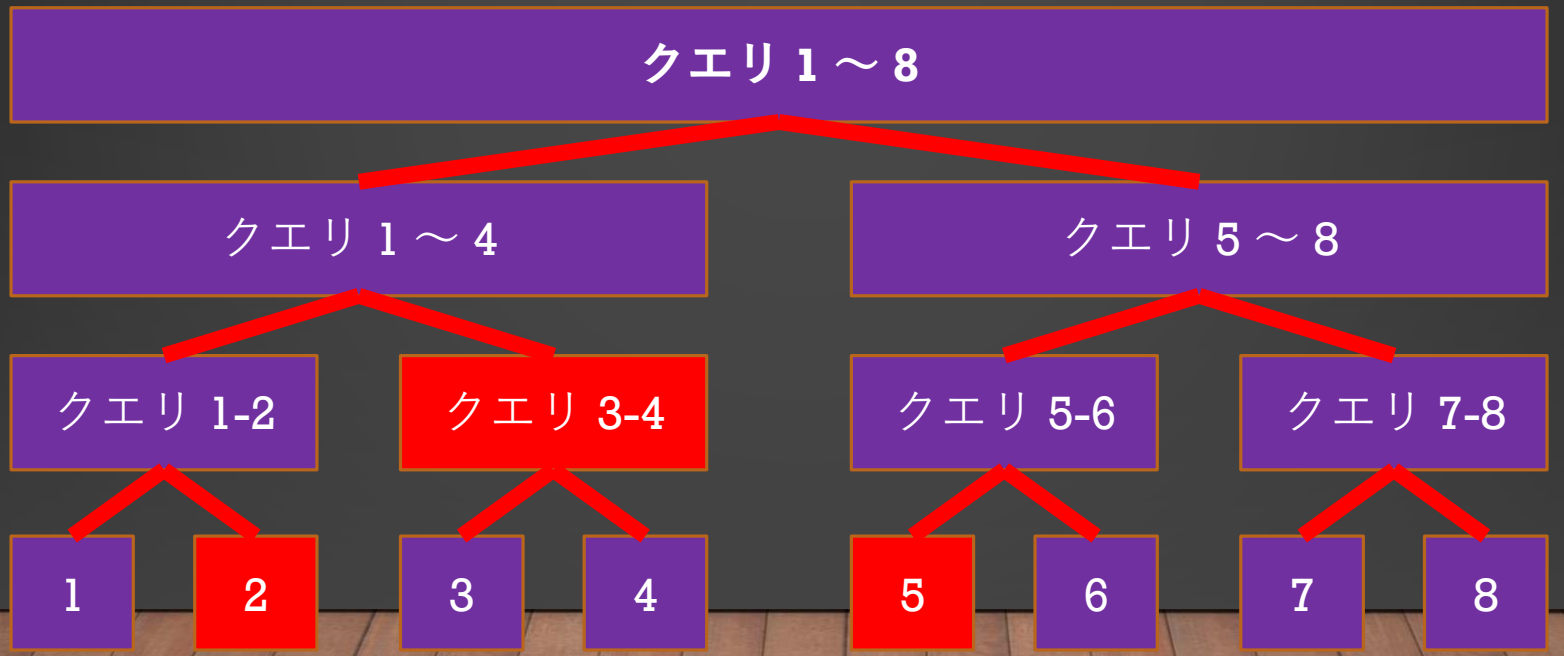
- もう少し計算量を改善するため[illegible]グ木を[illegible]できる
  - セグ木の[illegible]クエリにす[illegible]

ここで重要な事は、
セグ木のノードにに
Li Chao Tree を持つ



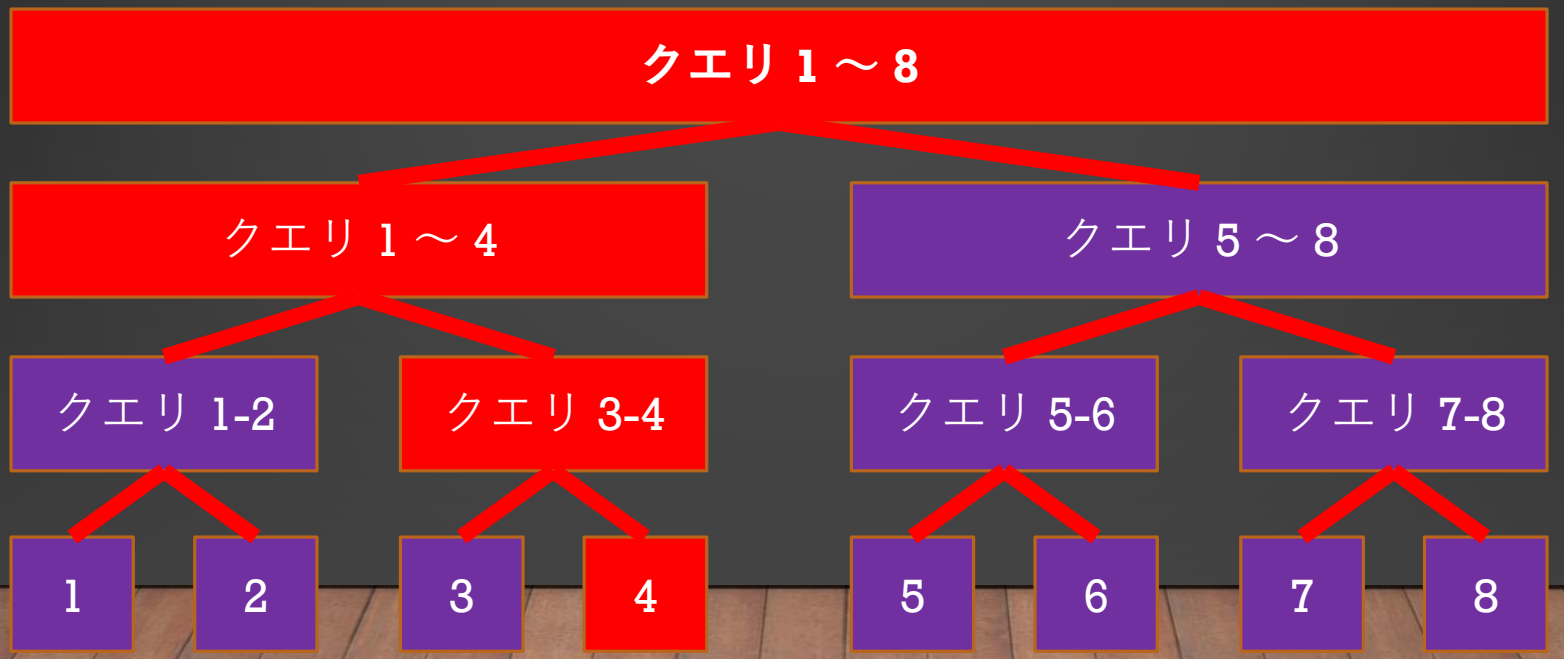
ノードに LI CHAO TREE ってどういうことか？

- 例えば、クエリ 2 に追加されクエリ 5 に削除される直線があるとする
  - この直線がある区間は $[2, 5]$ なので、**赤いノード**の Li Chao Tree に該当直線を追加



ノードに Li Chao Tree ってどういうことか？

- また、クエリ 4 に対する答えを求めたい時
  - 赤いノードにある Li Chao Tree すべてについて、座標 x における最小を求める
  - 答えは、$O(logQ)$ ノードの答えの中の最小値



実装方針③ ～セグ木に載せる～

- このような感じでセグ木に載せれば、削除が必要ない **Li Chao Tree** で出来、永続化も必要ない
- 計算量はどうなるか？
  - **Li Chao Tree** の線分追加の計算量は $O(\log^2 Q)$
  - 各クエリにおいて $O(logQ)$ 個のノードに追加する
  - 全体の計算量は $O(Qlog^3Q)$ であり、定数倍は速め
    - 部分点はほぼ確実に間に合い、満点を取れる可能性も僅かだが存在する

1100 点

考察 3

- この問題は、実際に $O(Qlog^3Q)$ では物足りない
  - 実は $O(QlogQ)$ 解が存在したりする

- 実装方針③ のセグ木に載せる方針を引き継ぐことにするが…
  - 傾きが負の直線がたくさん与えられて、座標 p において $y > 0$ の直線の最小値を求めるクエリをどうにかして $O(MlogM)$ で解きたい
  - ここでは M は半直線の個数
  - **Li Chao Tree** を使えば $O(M \log^2 M)$ だが…。

考察 3

- この問題は、
  - 実は $O(Q)$
- 実装

実は **ConvexHullTrick** の応用で出来る！！！

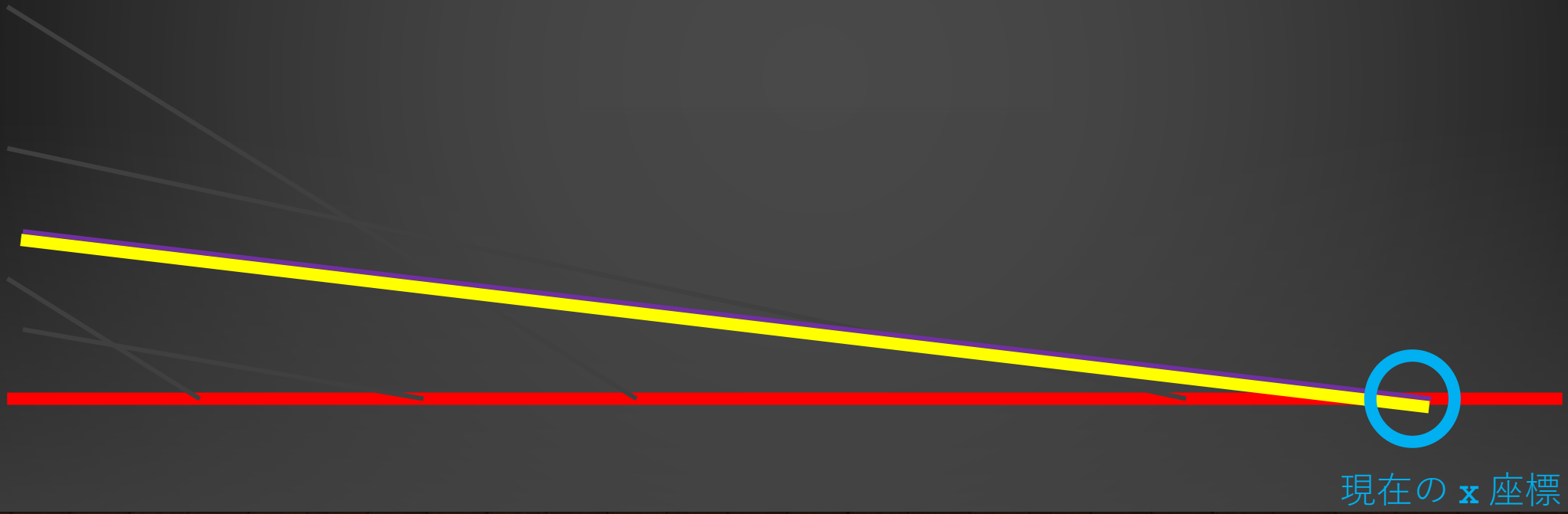
ける最小値を

考察 3

- 例えばこのように直線がたくさん与えられる

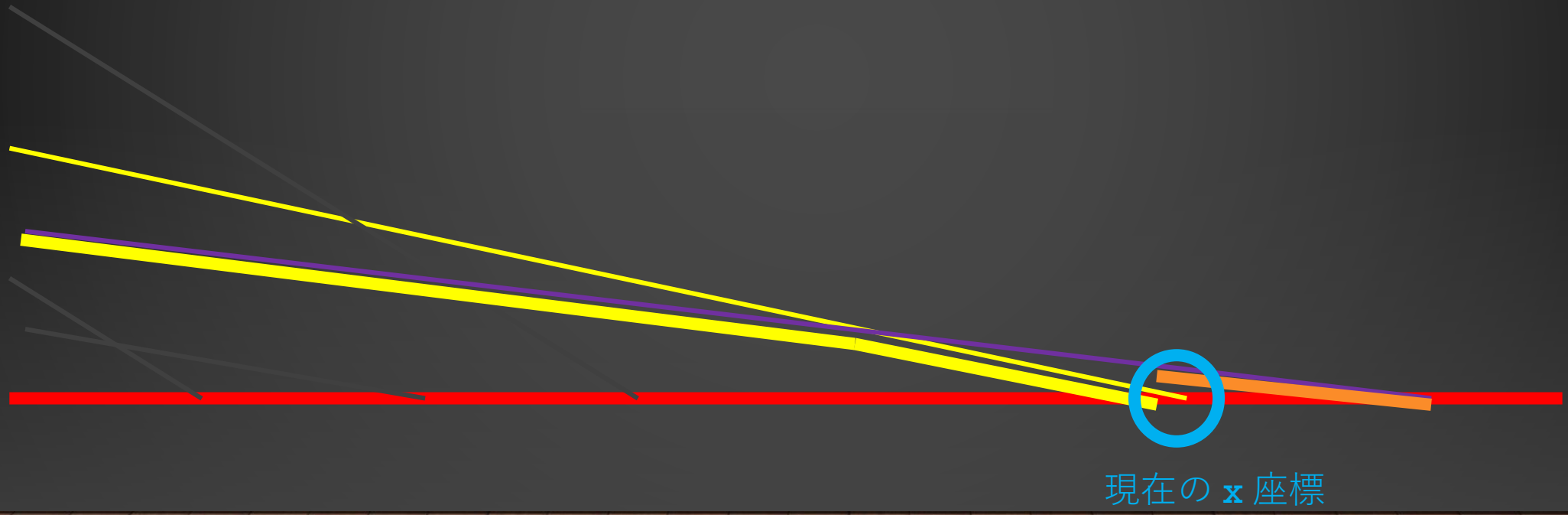
考察 3

- 全て傾きが負なので右から追加することを考える
  - 黄色の太線：暫定の最小値　　橙の太線：確定した最小値



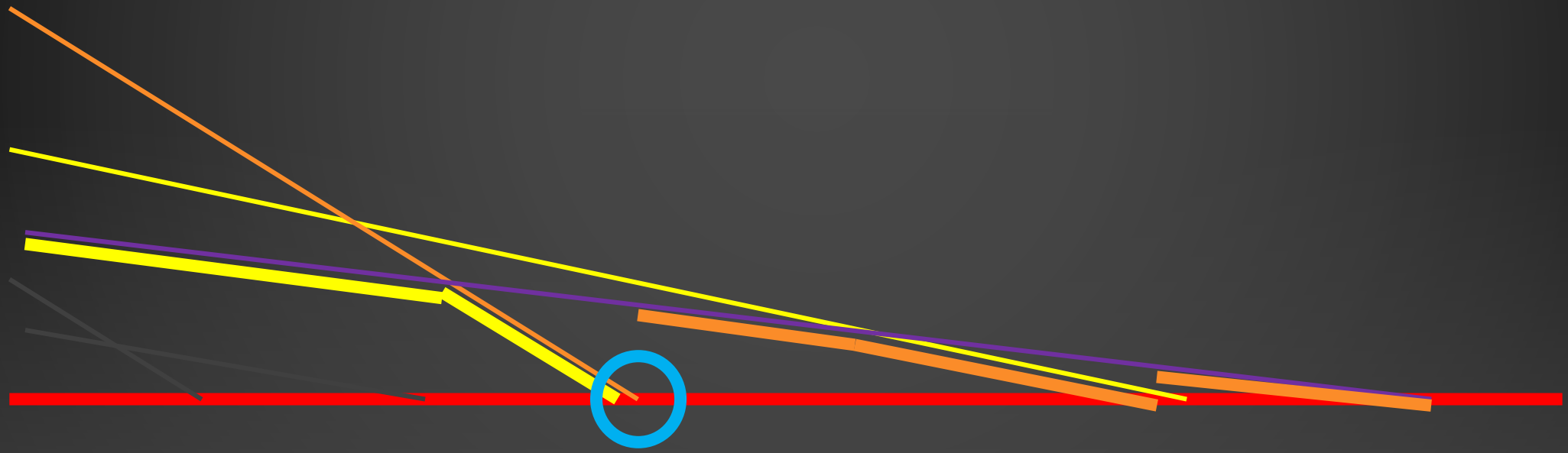
考察 3

- 全て傾きが負なので右から追加することを考える
  - 黄色の太線：暫定の最小値　　橙の太線：確定した最小値



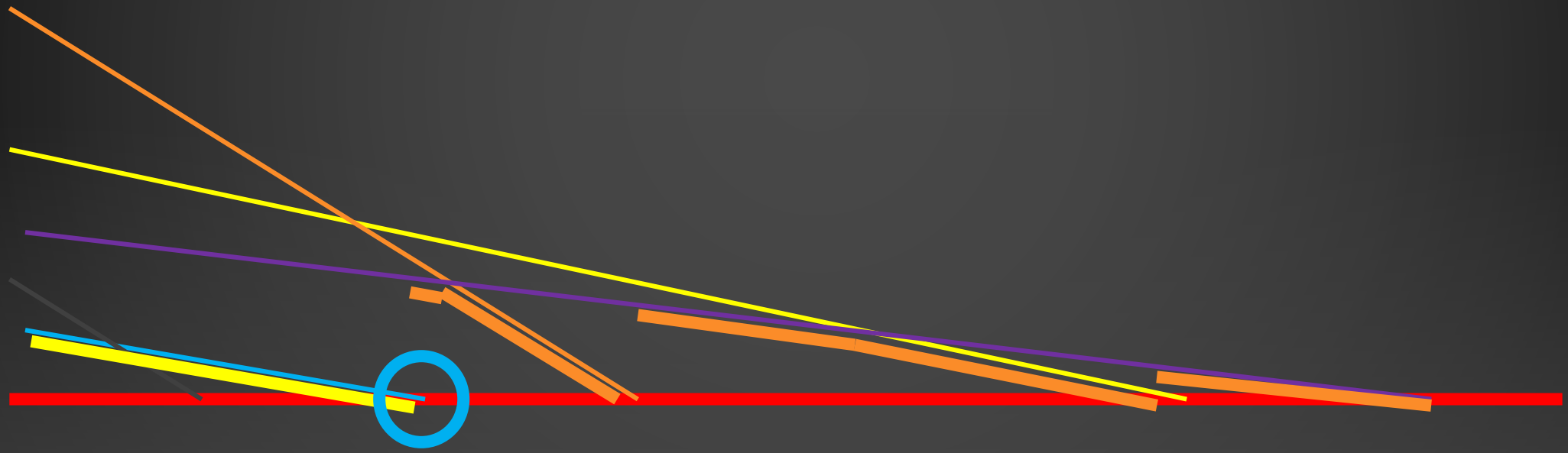
考察 3

- 全て傾きが負なので右から追加することを考える
  - 黄色の太線：暫定の最小値　　橙の太線：確定した最小値



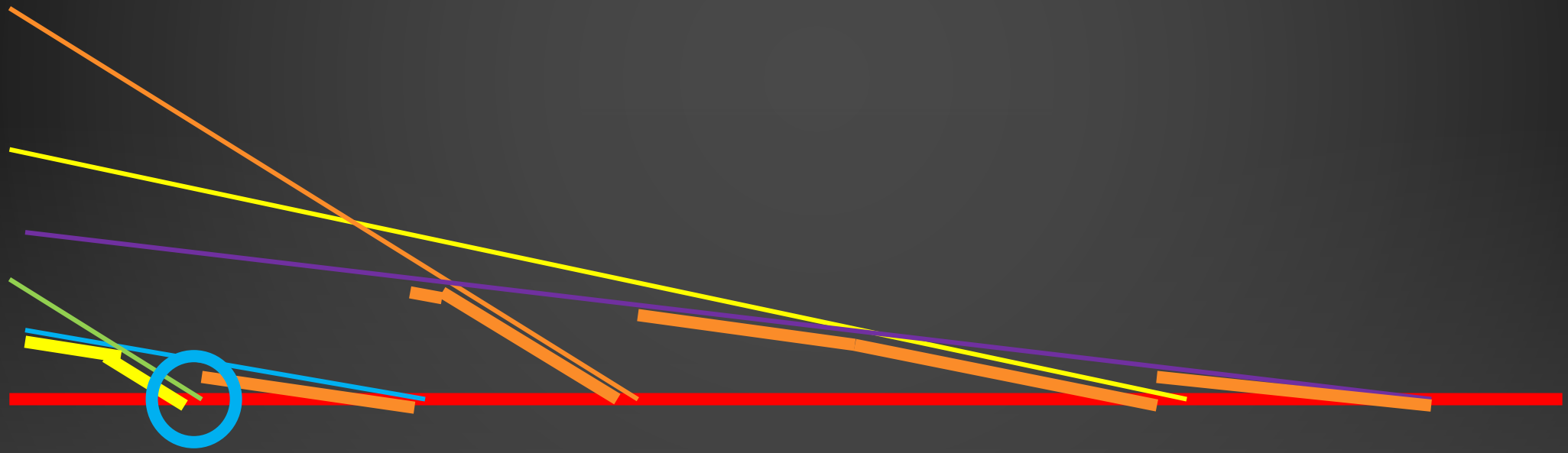
考察 3

- 全て傾きが負なので右から追加することを考える
  - 黄色の太線：暫定の最小値　　橙の太線：確定した最小値



考察 3

- 全て傾きが負なので右から追加することを考える
  - 黄色の太線：暫定の最小値　　橙の太線：確定した最小値



現在の x 座標

考察 3

- このように、x 切片が大きい直線から順に追加していく感じの **ConvexHullTrick** をすると、**stack** などを用いて $y > 0$ 部分における最小の直線区間を $O(M)$ で求めることができる

- 実際には直線を x 切片大きい順に **sort** することが必要なので、$O(MlogM)$ かかる

満点解法

- 全体計算量は以下のように解析できる
  - 追加する直線の数の合計は、セグ木に載せるので $O(QlogQ)$ 個
  - 各直線についての計算量
    - 直線追加： **sort** が一番大きくて $O(logQ)$
    - 命令の術： 二分探索が一番大きくて $O(logQ)$

- 結局 $(Qlog^2Q)$ で解くことができ、満点が得られる
  - **writer** 解は **1373ms** なので **C++** 以外だと厳しい

1400 点

満点解法の先

- 実はこの問題は $O(QlogQ)$ で解くことができる
- log が一個多い部分は **sort** と二分探索だけだが…
  - **sort**：そもそも適当に前処理をして x 座標が大きい順にセグ木に直線を追加すれば **sort** の必要がなくなり問題なし
  - 二分探索： クエリに関しても x 座標が小さい順に行い、尺取り法をすれば二分探索の必要がなくなり問題なし

- 結果、$O(QlogQ) + O(QlogQ) + O(QlogQ) + O(QlogQ) + O(QlogQ) = O(QlogQ)$!!!!!!

最後に

- 実はこの問題、$O(QlogQ)$ で解けるのです。
  - 最初 $O(Qlog^3Q)$ がやっとだったのでまさか **log l** つで解けるのは誰も思いませんよね。
- この問題は普通に難しいので解けなくても安心してください。
  - こどふぉだと **Div1 E** とかに出てもおかしくない難易度です。
- このコンテストに参加していただきありがとうございました！！！