Convex Hull Trick

えび (rsk0315)

HCPC 春の勉強会 @ April 4, 2019

最近知ったこと (1/3)

最初に，えびちゃんが最近お勉強したことの紹介をします．
興味を持ったものがあったら実装してみてね．

- フラクショナルカスケーディング
  - 長さ k の sorted 配列 n 個に対して x を探す問題を考える．
  - 愚直にやると $O(n \log k)$ のところを $O(n + \log k)$ にできる．
- 双対 LP
  - ある種の最適化問題を別の最適化問題に言いかえる．
  - 最短路とか最小費用流とかに帰着されたりする．

最近知ったこと (2/3)

- 非再帰セグ木
  - 区間の不等号がどうこう，みたいなバグとばいばいできる．
  - 配列長を 2 べきにしなくてもよい．
  - $a[i]$ の取得を $O(1)$ でできる．
  - 遅延伝播もできる．
  - かっこいい．
- HL 分解
  - 木のパス上のクエリを高速に処理できる．
- Borůvka法 (ブルーフカ)
  - 最小全域木を求めるアルゴリズム．
  - 各連結成分で最小コストの辺を選ぶのを繰り返す．

最近知ったこと (3/3)

- Lagrange 補間
  - n 次多項式 f について $f(x_0), f(x_1), \ldots, f(x_n)$ が既知とする.
  - このとき，任意の x に対して $f(x)$ を計算できる.
  - $x_i = i$ としておくと高速化が楽.
- Tonelli–Shanks アルゴリズム
  - a, p に対して $x^2 \equiv a \pmod p$ なる x を求める.
  - $p = (2q-1) \cdot 2^S$ に対して $O(S \log S / \log \log S)$.
- Convex hull trick
  - 直線群に関する特定のクエリを処理する.
  - 今日はこれを紹介します.

今日の流れ

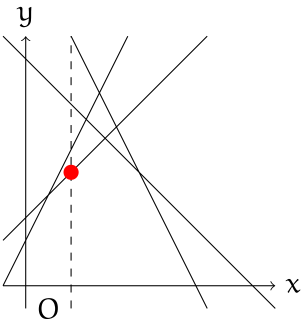
Convex hull trick（以下 CHT）を以下の流れでお勉強します.

- なにができるの？
- どう実装するの？
- どんな問題に応用できるの？

できる操作

xy-平面上の直線の集合 L に対して，次の操作を高速に行う．

- 集合に新たな直線を追加する．
- L の直線のうち，与えられた座標 x での y の最小値を答える．

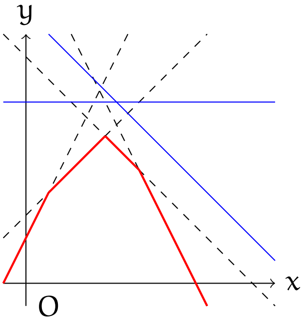


実装のやり方

大きく分けて以下の二つがありそう.

- 連想配列で処理するもの
  - ✓ オンライン処理できる
  - × 実装を雑にやると log がニつつく
- セグメント木で処理するもの
  - Li Chao (segment) tree とか呼ばれる
  - ✓ 線分の追加もできるかも？
  - × オフライン処理しなきゃかも（クエリの先読み）
    - 最小値クエリの x 座標（の候補）の数を N として全体で $O(N)$ 空間，クエリあたり $O(\log N)$ 時間かかる.
    - × $0 \leqslant x \leqslant 10^9$ とかだとつらい.
    - ✓ $0 \leqslant x \leqslant 10^5$ とかならオンライン処理もできそう.

重要な性質



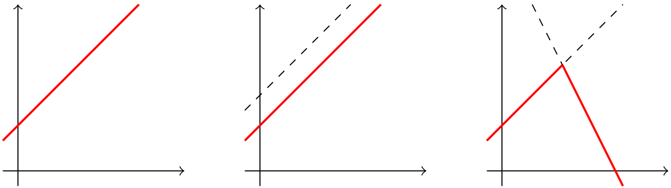
- 最小値をとる部分について傾きが単調減少となる．
- どの点においても最小値をとらない直線は無視してよい．

→ 無視できるかどうかの判定はどうしよう？

追加クエリ I

こういうのは帰納的に考えると楽で，base case は次の通り．

- 一本目の直線 → 常に必要．
- 二本目の直線
  - 傾きが同じなら切片が大きい方は必要ない．
  - そうでなければ両方必要．



追加クエリ II

次のような状況下で直線を追加することを考えるとよい.

- 不要な直線はすべて捨てられている.
- 必要な直線が二本以上存在する.

自明なケースの判定：

- 傾きが最小・最大の直線は常に必要.
  - x の絶対値が十分大きい点を考えよ.
- 傾きが同一の直線は切片が最小のもののみ必要.

追加クエリ III

追加する直線 ℓ' に対して，直線二本 ℓ_1, ℓ_2 を以下のように選ぶ：

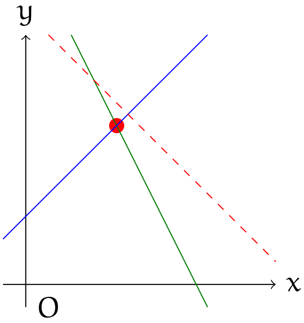
- 傾き順で並ぶ直線群に ℓ' を入れるときに ℓ' をはさむ二直線.
  !! 最小値をとる部分の傾きは単調減少
- $\ell_?, \ldots, \ell_1, \ell', \ell_2, \ldots, \ell_?$.

e.g. 傾き $\{5, 4, 2, 1\}$ の直線群に対して傾き 3 の直線 ℓ' を追加する

→ ℓ_1 は傾き 4，ℓ_2 は傾き 2 の直線.

追加クエリ IV

ℓ' と ℓ_1, ℓ_2 の位置関係を調べる.

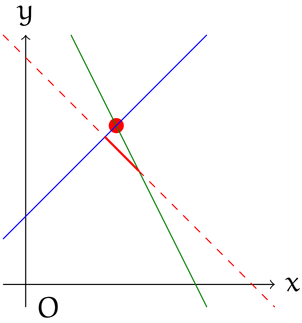


ℓ_1 と ℓ_2 の交点より上を ℓ' が通る場合, ℓ' は不要.

交点を通る場合も不要としてよい.

追加クエリ IV

ℓ' と ℓ_1, ℓ_2 の位置関係を調べる.



ℓ_1 と ℓ_2 の交点より下を ℓ' が通る場合, ℓ' は必要.

交点との ℓ' との位置関係を調べる方法を考えればよい.

追加クエリ V

$$\begin{aligned} \ell_1 &: \quad y = a_1 x + b_1 \\ \ell_2 &: \quad y = a_2 x + b_2 \\ \ell' &: \quad y = a' x + b' \end{aligned}$$

とすると，ℓ_1, ℓ_2 の交点 (x_c, y_c) は計算できる．
で，$a'x_c + b'$ と y_c の大小関係を考えるとよい．

結局，次の条件が成り立つとき ℓ' は不要だとわかるはず．
不等号とか逆だったら許して．

$$(a' - a_2) \cdot (b' - b_1) \geqslant (a_1 - a') \cdot (b_2 - b').$$

簡単な式で判定できることだけわかってもらえたらいいです．

追加クエリ VI

結局，追加クエリは次のように処理すればよい．

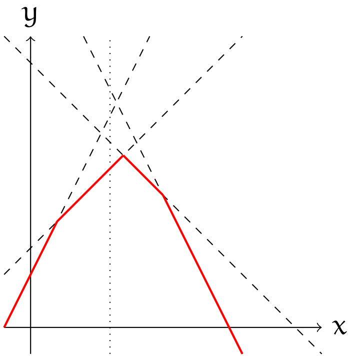
1. ℓ' に対して前述の判定を行う．
   → 不要なら終了．必要なら ℓ' を L に追加．
2. ℓ_2 が不要になっていないか同様に判定し，不要なら消す．
   2+. 必要な直線が見つかるまで ℓ_2 を選び直し，消し続ける．
3. ℓ_1 についても同様．

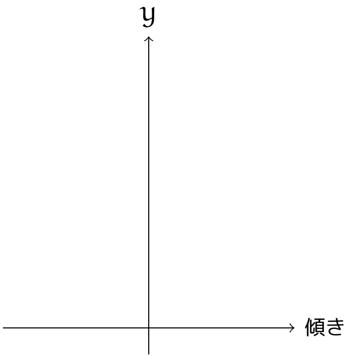
これをがんばって実装するとよいです．できますね．

ここからは最小値クエリを考えます．

重要な性質

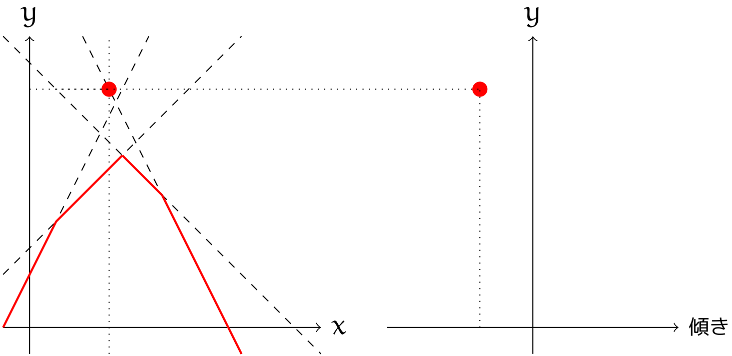
ある x 座標に注目して，各直線の傾きと y 座標の関係を見てみる．





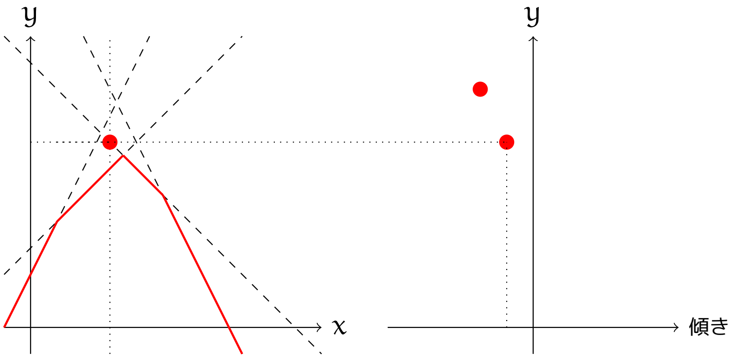
重要な性質

ある x 座標に注目して，各直線の傾きと y 座標の関係を見てみる．



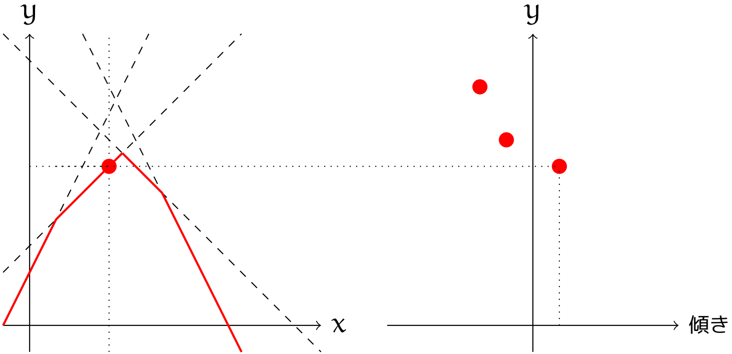
重要な性質

ある x 座標に注目して，各直線の傾きと y 座標の関係を見てみる．



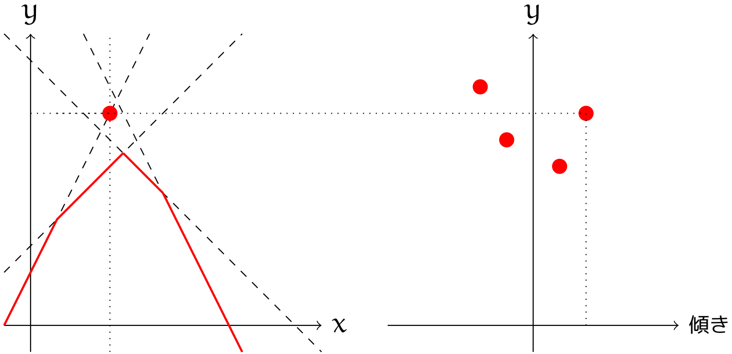
重要な性質

ある x 座標に注目して，各直線の傾きと y 座標の関係を見てみる．



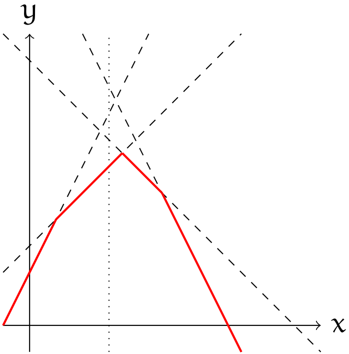
重要な性質

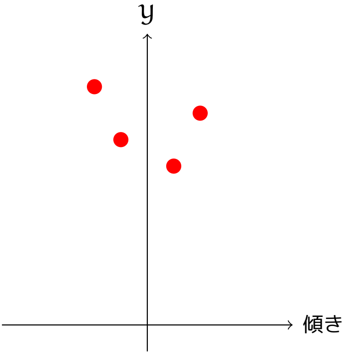
ある x 座標に注目して，各直線の傾きと y 座標の関係を見てみる．



重要な性質

ある x 座標に注目して，各直線の傾きと y 座標の関係を見てみる．





凸になっている！（この最小値がクエリの答え）

最小値クエリ

離散な凸関数の最小値は二分探索で求められる.

連想配列で直線を管理すると，ある点での値を計算しやすそう.

$$\mathrm{map}[\text{傾き}] \to \text{切片}$$

傾きの降順でソートして持っておくと追加が楽.
→ 平衡二分探索木ベースの連想配列（`std::map`）がよさそう.

各ステップで毎回連想配列から探すと $O((\log|L|)^2)$ 時間.
→ 工夫してなんとかするとよさそう.

限定された状況下での最適化

- 追加する直線の傾きが単調減少な場合
  → 挿入位置が常に末尾なので，`std::vector` とかでよい．
- さらに最小値クエリが単調増加の場合
  → 古い直線から不要になっていく．`std::deque` が適役．
  - ならし $O(1)$ 時間で各クエリを処理できる．わーい．

補足（追記）

ある直線と，それが最小値をとる区間との対応付けを覚えておいて処理する方法のが自然かも？

なんだけど，整数だと区間の端点を切り下げたり切り上げる必要があって，これはC++の除算の仕様と気持ちが合わなくて無限にバグらせがち．やめたい．やめてしまった．

ひとやすみ

ここまでが連想配列ベースの実装.

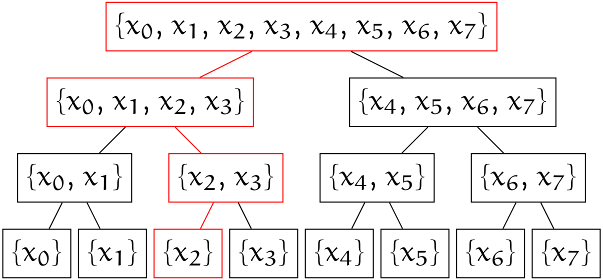
ここからはセグメント木ベースの実装.

実装方針

最小値クエリで聞かれる点 $\{x_0, x_1, \ldots, x_{N-1}\}$ が既知とする．

各ノードはいくつかの点をカバーしていて，一つの直線を持つ．
直線群のうち x_i での値が最小となる直線は，x_i をカバーするノードのいずれかが持っているように保つ．

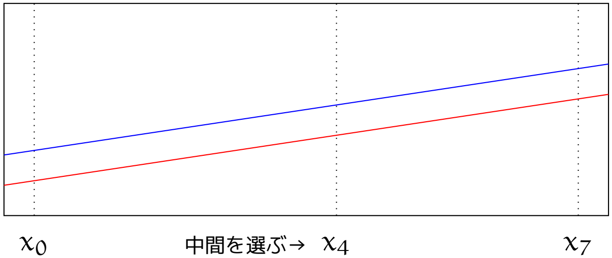
$y = 0x + \infty$ みたいな直線で初期化しておく．



追加クエリ

根ノードからトップダウンに更新していく．ノードの持つ直線 ℓ と追加する直線 ℓ' について，3 つの x 座標での値を比較する．

$\{x_0, x_1, \ldots, x_7\}$ を例に考える．

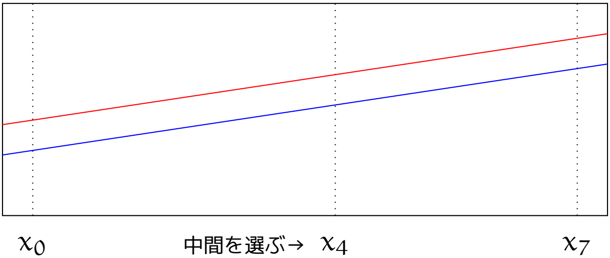


両端点で下回っているなら ℓ を ℓ' で置き換えて終了．

追加クエリ

根ノードからトップダウンに更新していく．ノードの持つ直線 ℓ と追加する直線 ℓ' について，3 つの x 座標での値を比較する．

$\{x_0, x_1, \ldots, x_7\}$ を例に考える．

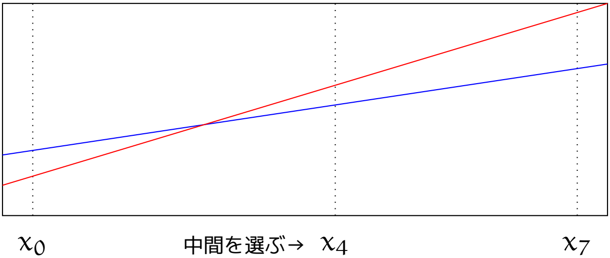


両端点で上回っているなら何もせず終了．

追加クエリ

根ノードからトップダウンに更新していく．ノードの持つ直線 ℓ と追加する直線 ℓ' について，3 つの x 座標での値を比較する．

$\{x_0, x_1, \ldots, x_7\}$ を例に考える．

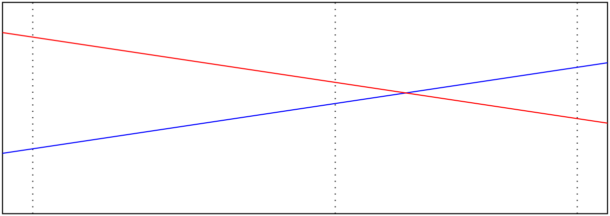


1 点で下回っているなら，下回っている方の子ノードを更新．

追加クエリ

根ノードからトップダウンに更新していく．ノードの持つ直線 ℓ と追加する直線 ℓ' について，3 つの x 座標での値を比較する．

$\{x_0, x_1, \ldots, x_7\}$ を例に考える．

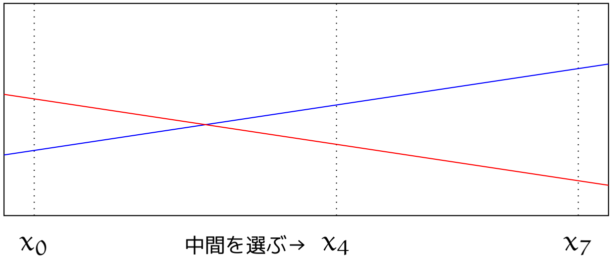


1 点で下回っているなら，下回っている方の子ノードを更新．

追加クエリ

根ノードからトップダウンに更新していく．ノードの持つ直線 ℓ と追加する直線 ℓ' について，3 つの x 座標での値を比較する．

$\{x_0, x_1, \ldots, x_7\}$ を例に考える．

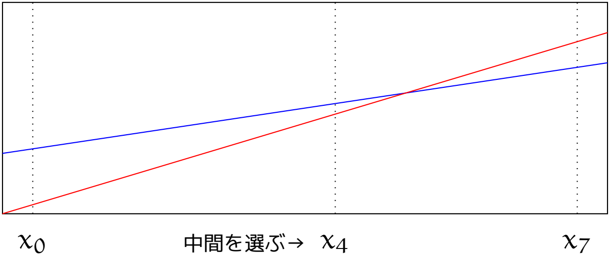


それ以外のとき，ℓ と ℓ' を交換すると 1 点のパターンに帰着可能．

追加クエリ

根ノードからトップダウンに更新していく．ノードの持つ直線 ℓ と追加する直線 ℓ' について，3 つの x 座標での値を比較する．

$\{x_0, x_1, \ldots, x_7\}$ を例に考える．



それ以外のとき，ℓ と ℓ' を交換すると 1 点のパターンに帰着可能．

最小値クエリ

作り方から明らか.

- x_i をカバーするノードを全部見る.
- それらのノードの持つ直線の x_i での値の最小値が答え.

ひとやすみ

ここまでは実装の話.

ところで CHT はなんの名前なの？ 概念？ 実装？

- ??? : {CHT, Li Chao tree, . . . }
- CHT : {Li Chao tree, . . . }

ハッシュは連想配列を指す用語ではない！（素振り）

で，ここからは応用の話.

応用 – 例題

EDPC Z – Frog 3

N 個の足場があって，それぞれ高さは $h_1, h_2, \ldots, h_N$ です．
足場 j から i へ跳ぶと，コスト $(h_i - h_j)^2 + C$ かかります．
足場 1 から N まで行くときの最小コストを求めてね．
$1 \leqslant N \leqslant 2 \times 10^5$，$1 \leqslant h_1 < \cdots < h_N \leqslant 10^6$，$1 \leqslant C \leqslant 10^{12}$．

$$dp[i] = \min_{1 \leqslant j < i} \{dp[j] + (h_i - h_j)^2 + C\}???$$

$O(N^2)$ かかって破滅では…？

解説

展開してみる.

$$\mathrm{dp}[i] = \min_{1 \leqslant j < i} \{\mathrm{dp}[j] + h_i^2 - 2h_i h_j + h_j^2 + C\}$$

解説

展開してみる.

$$\mathrm{dp}[i] = \min_{1 \leqslant j < i} \{\mathrm{dp}[j] + h_i^2 - 2h_i h_j + h_j^2 + C\}$$

j から見た定数たちは外に出せる.

$$\mathrm{dp}[i] = \min_{1 \leqslant j < i} \{\mathrm{dp}[j] - 2h_i h_j + h_j^2\} + h_i^2 + C$$

解説

展開してみる.

$$dp[i] = \min_{1 \leqslant j < i} \{dp[j] + h_i^2 - 2h_i h_j + h_j^2 + C\}$$

j から見た定数たちは外に出せる.

$$dp[i] = \min_{1 \leqslant j < i} \{dp[j] - 2h_i h_j + h_j^2\} + h_i^2 + C$$

i に関してまとめてみる.

$$dp[i] = \min_{1 \leqslant j < i} \{(-2h_j)h_i + (dp[j] + h_j^2)\} + h_i^2 + C$$

ん?

遷移を眺める

$$\mathrm{dp}[2] = \min\{(-2h_1)h_2 + (\mathrm{dp}[1] + h_1^2)\} + h_2^2 + C;$$

遷移を眺める

$$\begin{aligned} dp[2] &= \min\{(-2h_1)h_2 + (dp[1] + h_1^2)\} + h_2^2 + C; \\ dp[3] &= \min\{(-2h_1)h_3 + (dp[1] + h_1^2), \\ &\qquad (-2h_2)h_3 + (dp[2] + h_2^2)\} + h_3^2 + C; \end{aligned}$$

遷移を眺める

$$
\begin{aligned}
dp[2] &= \min\{(-2h_1)h_2 + (dp[1] + h_1^2)\} + h_2^2 + C; \\
dp[3] &= \min\{(-2h_1)h_3 + (dp[1] + h_1^2), \\
&\qquad (-2h_2)h_3 + (dp[2] + h_2^2)\} + h_3^2 + C; \\
dp[4] &= \min\{(-2h_1)h_4 + (dp[1] + h_1^2), \\
&\qquad (-2h_2)h_4 + (dp[2] + h_2^2), \\
&\qquad (-2h_3)h_4 + (dp[3] + h_3^2)\} + h_4^2 + C;
\end{aligned}
$$

遷移を眺める

$$
\begin{aligned}
dp[2] &= \min\{(-2h_1)h_2 + (dp[1] + h_1^2)\} + h_2^2 + C; \\
dp[3] &= \min\{(-2h_1)h_3 + (dp[1] + h_1^2), \\
&\qquad (-2h_2)h_3 + (dp[2] + h_2^2)\} + h_3^2 + C; \\
dp[4] &= \min\{(-2h_1)h_4 + (dp[1] + h_1^2), \\
&\qquad (-2h_2)h_4 + (dp[2] + h_2^2), \\
&\qquad (-2h_3)h_4 + (dp[3] + h_3^2)\} + h_4^2 + C;
\end{aligned}
$$

↑ 傾き　　　↑ 切片

遷移を眺める

$$
\begin{aligned}
dp[2] &= \min\{(-2h_1)h_2 + (dp[1] + h_1^2)\} + h_2^2 + C; \\
dp[3] &= \min\{(-2h_1)h_3 + (dp[1] + h_1^2), \\
&\qquad (-2h_2)h_3 + (dp[2] + h_2^2)\} + h_3^2 + C; \\
dp[4] &= \min\{(-2h_1)h_4 + (dp[1] + h_1^2), \\
&\qquad (-2h_2)h_4 + (dp[2] + h_2^2), \\
&\qquad (-2h_3)h_4 + (dp[3] + h_3^2)\} + h_4^2 + C;
\end{aligned}
$$

↑ 傾き　　　↑ 切片

$dp[i]$ では，$x = h_i$ における直線群の最小値を求めている！
→ CHT で求められる！

まとめ

一般に，以下のような遷移をする DP に対して強気になれる．

$$\mathrm{dp}[i] := \min_{1 \leqslant j < i} \{p(j)q(i) + r(j)\} + s(i)$$

$i = 2, \ldots, N$ に対して次の処理をすればよい．

1. 直線 $y = p(i-1) \cdot x + r(i-1)$ を追加する．
2. 直線群の $x = q(i)$ での最小値を元に $\mathrm{dp}[i]$ を計算する．

問題たち

- https://atcoder.jp/contests/dp/tasks/dp_z
- https://atcoder.jp/contests/jag2015summer-day4/tasks/icpc2015summer_day4_i
- https://atcoder.jp/contests/colopl2018-final-open/tasks/colopl2018_final_c
- https://yukicoder.me/problems/no/409
- http://codeforces.com/contest/631/problem/E

バグとのたたかい

私は Li Chao tree を実装するのに 9 回 WA を出しました．

100	4290 Byte	AC	217 ms	32000 KB
0	4286 Byte	WA	195 ms	30968 KB
0	4263 Byte	WA	201 ms	33124 KB
0	4175 Byte	WA	205 ms	30960 KB
0	4113 Byte	WA	199 ms	26984 KB
0	4051 Byte	WA	194 ms	27108 KB
0	4045 Byte	TLE		
0	4099 Byte	WA	204 ms	27108 KB
0	3940 Byte	WA	194 ms	27108 KB
0	3939 Byte	WA	193 ms	27000 KB
0	3264 Byte	WA	192 ms	26992 KB
0	3310 Byte	CE		

自分とのたたかい

久々に fastest を得る遊びをしました (8 ms).


問題: Z - Frog 3 言語: -

結果: AC ユーザ: 検索 リセット

提出日時	問題	ユーザ	言語	得点	コード長	結果	実行時間	メモリ	
2019-03-31 22:36:50	Z - Frog 3	rsk0315	C++14 (GCC 5.4.1)	100	4579 Byte	AC	8 ms	5632 KB	詳細
2019-01-19 22:43:52	Z - Frog 3	denverjin	C++14 (GCC 5.4.1)	100	2086 Byte	AC	9 ms	3328 KB	詳細

Thank you!

終

制作・著作