JOI 春合宿 2012 Day2
回転 (Rotate)

秋葉 拓哉

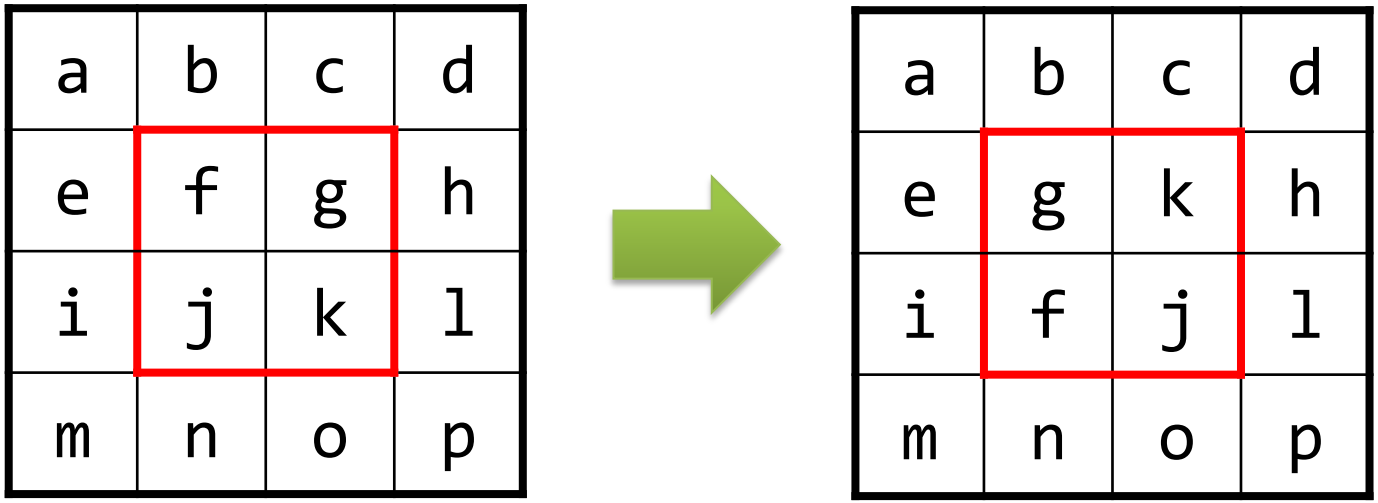
問題概要

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

$N \times N$ の盤面 $(N \leq 1000)$

各マスにはアルファベット

問題概要



Q 回，正方形領域を回転 $(Q \leq 2000)$

終了後の盤面を出力

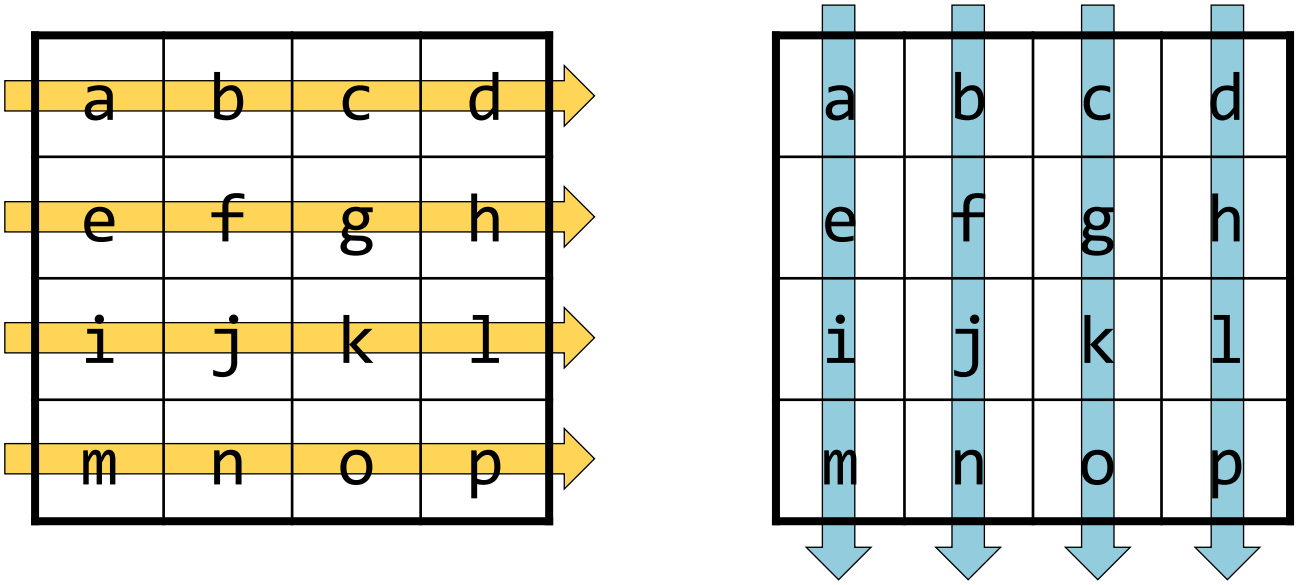
愚直な解法 (10 点)

1. 二次元配列に盤面を確保
2. 毎回，本当に回転する

$O(QN^2)$ 時間，$N, Q \leq 100$ のセットは OK

（デバッグ用にも便利なのでどちらにせよ実装するべし）

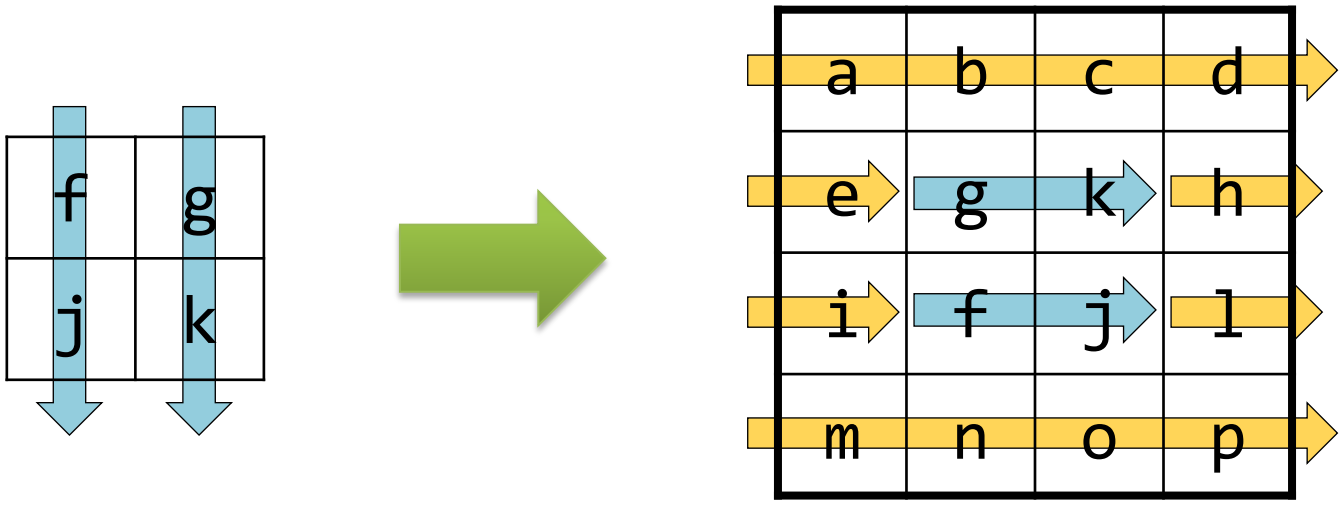
高速な解法へのアイディア



盤面を横向きと縦向きで列で管理

列に**適切なデータ構造**を用いる

高速な解法へのアイディア

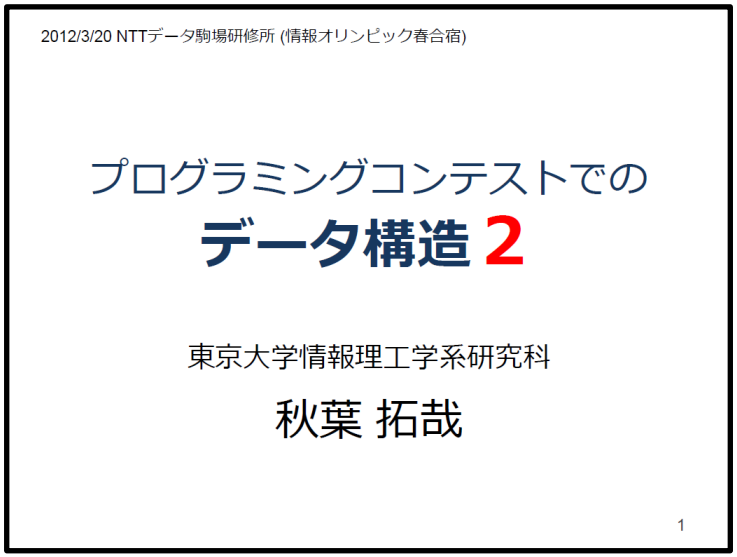


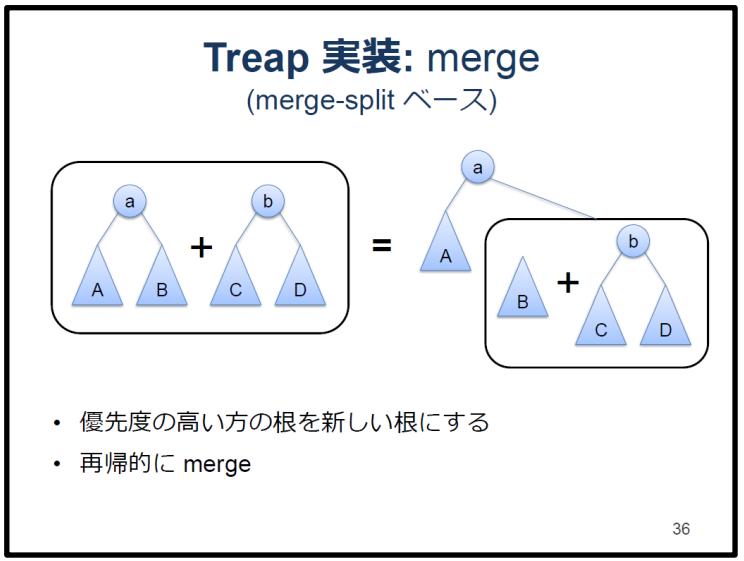
回転は付け替えるだけ

(4 方向分管理しておくか，2 方向で反転をサポート)

適切なデータ構造？

列の split, merge と言えば…
この問題講義でやったやつだ!!

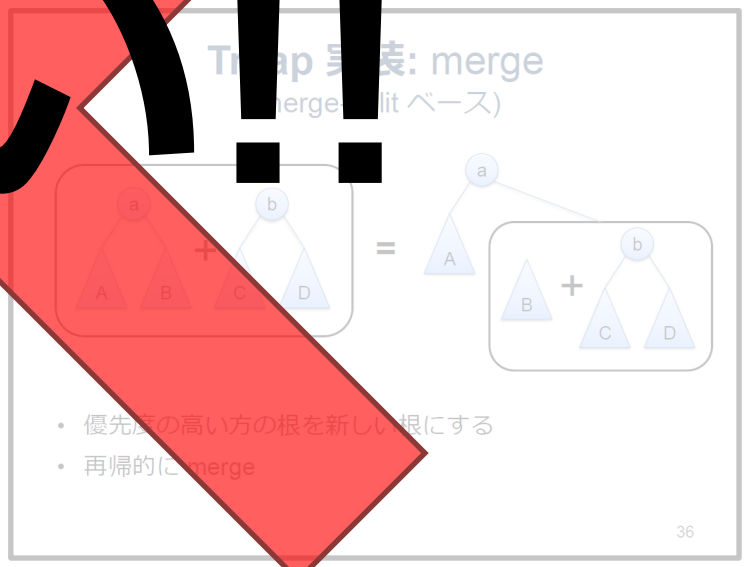




適切なデータ構造？

列の split, merge と言えば…
この問題講義でやったやつだ!!

遅い!!



平衡二分探索木を使うと…

- 計算量 $O(NQ \log N)$
- $N = 1000, Q = 2000 \rightarrow NQ = 2\,000\,000$

- 数百万のオーダーに log がつくのはそれだけで緊張
- 数方向について処理をするし，
- 平衡二分探索木の定数も大きい

…それで 2 秒はちょっとヤバイと見積もれる

…その前に！

平衡二分探索木は本当に必要？

- いつものじゃダメ？
  - 配列，線形リスト
  - Std::set, std::map
  - Binary Indexed Tree，バケット法，セグメント木
- 実装が面倒なので楽に避けられたら避けたい
- 実際のとこ，本当に必要になる問題はレア

平衡二分探索木まとめ

- まずはもっと容易な道具を検討！
  - 配列, リスト, std::map，BIT，セグメント木，バケット法
  - 必要な場所だけ作るセグメント木
- 実装が楽な平衡二分探索木を選ぼう
  - **今回**: Treap / Randomized Binary Search Tree
  - **他**: スプレー木, Scapegoat 木, Block Linked List, Skip List
- 実装しよう
  - insert / erase ベース　vs.　merge / split ベース
  - 更新遅延

…でもやっぱその前に！

- 動的木は本当に必要？いつものじゃダメ？
- やはり，実装が面倒，避けられたら避けたい
- 実際のとこ，本当に必要になる問題は**皆無**

（特別賞を狙いたいのであればこの限りではない）

Link-Cut 木まとめ

- まずはもっと容易な道具を検討！
  - クエリの平方分割
- 実装しよう （下に行くほど大変）
  - expose, link, cut, root, 頂点の情報に関する質問
  - evert, 頂点の情報の更新
  - 辺の情報に関する質問・更新

…その前に！

平衡二分探索木は本当に必要？

- いつものじゃダメ？
  - 配列，線形リスト
  - Std::set, std::map
  - Binary Indexed Tree，バケット法，セグメント木
- 実装が面倒なので楽に避けられたら避けたい
- 実際のとこ，本当に必要になる問題はレア

平衡二分探索木まとめ

- まずはもっと容易な道具を検討！
  - 配列, リスト, std::map，BIT，セグメント木，バケット法
  - 必要な場所だけ作るセグメント木
- 実装が楽な平衡二分探索木を選ぼう
  - **今回**: Treap / Randomized Binary Search Tree
  - **他**: スプレー木, Scapegoat 木, Block Linked List, Skip List
- 実装しよう
  - insert / erase ベース　vs.　merge / split ベース
  - 更新遅延

…でもやっぱその前に！

- 動的木は本当に必要？いつものじゃダメ？
- やはり，実装が面倒，避けられたら避けたい
- 実際のとこ，本当に必要になる問題は**皆無**

（特別賞を狙いたいのであればこの限りではない）

Link-Cut 木まとめ

- まずはもっと容易な道具を検討！
  - クエリの平方分割
- 実装しよう （下に行くほど大変）
  - expose, link, cut, root, 頂点の情報に関する質問
  - evert, 頂点の情報の更新
  - 辺の情報に関する質問・更新

…その前に！

平衡二分探索木は本当に必要？

- いつものじゃダメ？
  - 配列，線形リスト
  - Std::set, std::map
  - Binary Indexed Tree，バケット法，セグメント木
- 実装が面倒なので楽に避けられたら避けたい
- 実際のとこ，本当に必要になる問題はレア

平衡二分探索木まとめ

- まずはもっと容易な道具を検討！
  - 配列，リスト，std::map，BIT，セグメント木，バケット法
  - 必要な場所だけ作るセグメント木
- 実装が楽な平衡二分探索木を選ぼう
  - **今回**: Treap / Randomized Binary Search Tree
  - **他**: スプレー木, Scapegoat 木, Block Linked List, Skip List
- 実装しよう
  - insert / erase ベース　vs.　merge / split ベース
  - 更新遅延

…でもやっぱその前に！

- 動的木は本当に必要？いつものじゃダメ？
- やはり，実装が面倒，避けられたら避けたい
- 実際のとこ，本当に必要になる問題は**皆無**

（特別賞を狙いたいのであればこの限りではない）

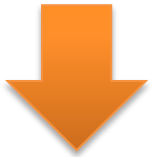
Link-Cut 木まとめ

- まずはもっと容易な道具を検討！
  - クエリの平方分割
- 実装しよう （下に行くほど大変）
  - expose, link, cut, root, 頂点の情報に関する質問
  - evert, 頂点の情報の更新
  - 辺の情報に関する質問・更新

高速な解法 (100 点)

ポイント

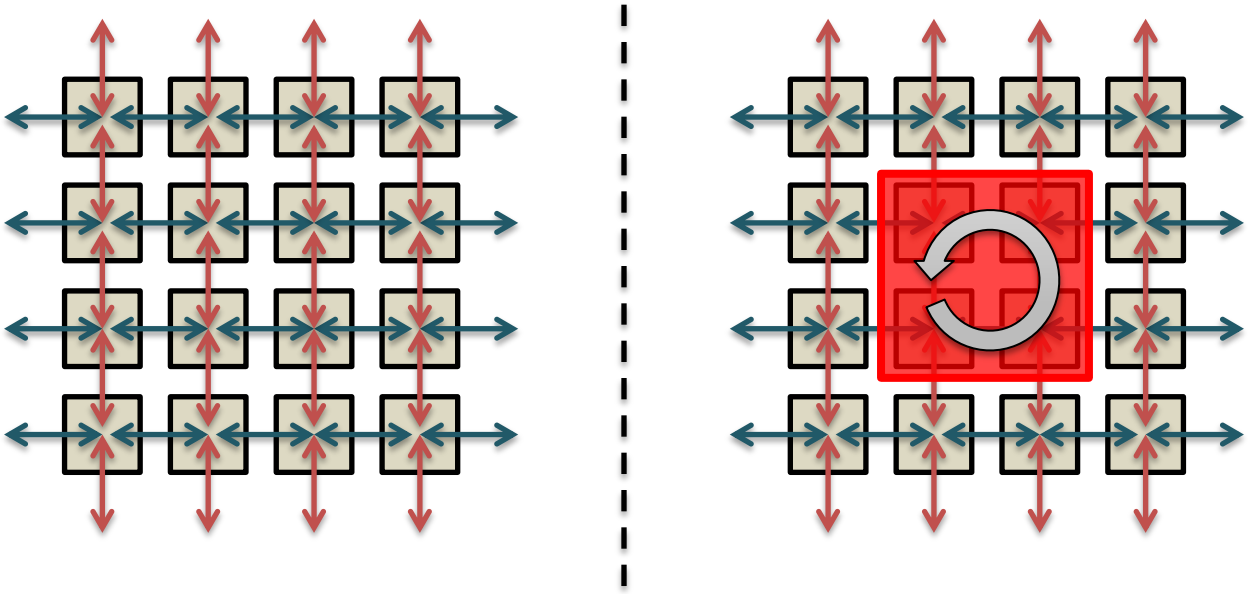
一度に $O(N)$ はかけてよい



アイディア

実はリストでできる!!

高速な解法 (100 点)



右にも下にも行けるリストでやる

数方向分持っておいて取り替えたり,

入ってきたら次に出ていく方向を管理したりすれば良い

高速な解法 (100 点)

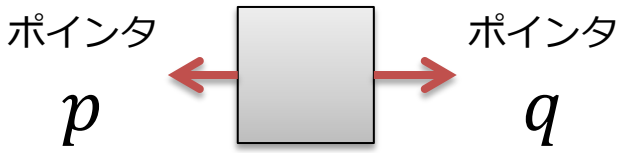
何故リストでも良い？

- むしろ，何故，平衡二分探索木を使いたかったか？
  → 切りたい場所をすぐ見つけられ，すぐ切れるから

- でも，一回に $O(N)$ はかかっても良い
  - よって，切る場所を見つけるのに全体で $O(N)$ かかって良い
  - 切る場所まで端からたどることが許される

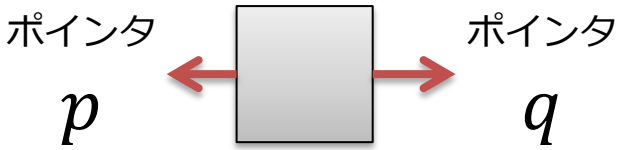
おまけ: Xor-Linked List

普通の Double-Linked List



p と q の両方を覚えてる

Xor-Linked List



$p \otimes q$ を覚えておく

- $p \otimes q$ を覚えておけば，実は十分な事が多い
  - p から来たら，$(p \otimes q) \otimes p = q$ より，q がわかる
  - q から来たら，$(p \otimes q) \otimes q = p$ より，p がわかる
- 領域が減るだけでなく，左右反転が容易にでき便利
  - 張り替えるだけで良い

得点分布