JOI 2013-2014 本選
切り取り線(Cutting) 解説

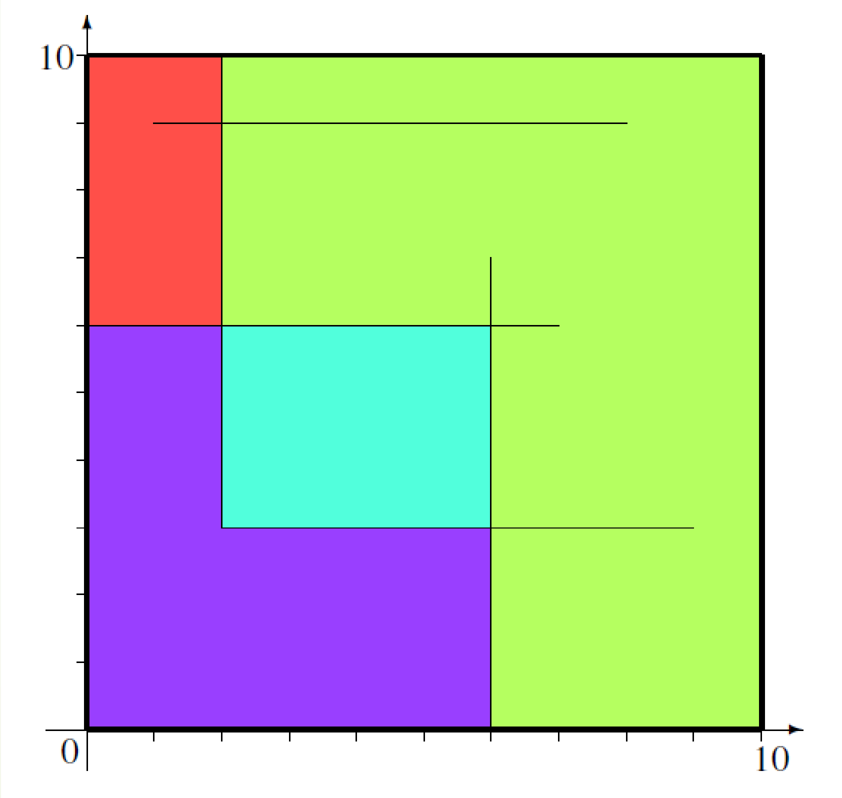
問題概要

- 長方形の紙があります
- たくさんの切り取り線に沿って切り取ります
- いくつの部分に分かれる？

入力例

- 入力例 1
- 切り分けられる部分ごとに色を塗ると右図のようになる

- 部分の中に一見余分に見えるような切り取り線があっても一向にかまわない



おことわり

- 解説の時間は限られているので，よくあるアルゴリズムについての詳細は省略します

- Segment Tree, 座標圧縮，Union Find など
- 「プログラミングコンテストチャレンジブック」
- ほか，インターネット上の資料などを参照してください

探索による解法

- 小課題 1
- 紙がとても小さく，切り取り線も少ない
- 1 × 1 のマス目が W × H 個並んでいると思ってもまだ大丈夫
- 隣り合うマス目の間に切り取り線があるか記録する
- 紙がいくつの部分に分かれるか探索
  - 幅優先探索 (BFS) / 深さ優先探索 (DFS)
- O(WH + N(W+H))
- がんばって探索を書くと小課題 1 が解ける (5 点)

座標圧縮

- 小課題 2
- 切り取り線は 1000 個以下だが，座標がいたずらに大きい
- 座標圧縮しましょう

- 覚えるべき座標は，紙の端と，切り取り線の座標で 1 回以上現れるものだけ
- X, Y 座標ごとに高々 2N + 2 個

- O(N^2)
- 座標圧縮して探索すれば小課題 1, 2 が解ける (10 点)

探索の限界

- 小課題 3, 4, 5
- もはや切り取り線は 100000 本も存在する
- 普通に探索しようとしたらそもそもメモリが足りない

「長方形の紙」の除去

- 紙の外に切り取り線はない
- 紙の辺を特別扱いするのは面倒
- 無限に広い平面を，紙の辺と切り取り線たちで分割する問題に置き換える
  - すると，分割数 – 1 (長方形の外を除外) が答えになる

領域の分割 -> 領域の併合

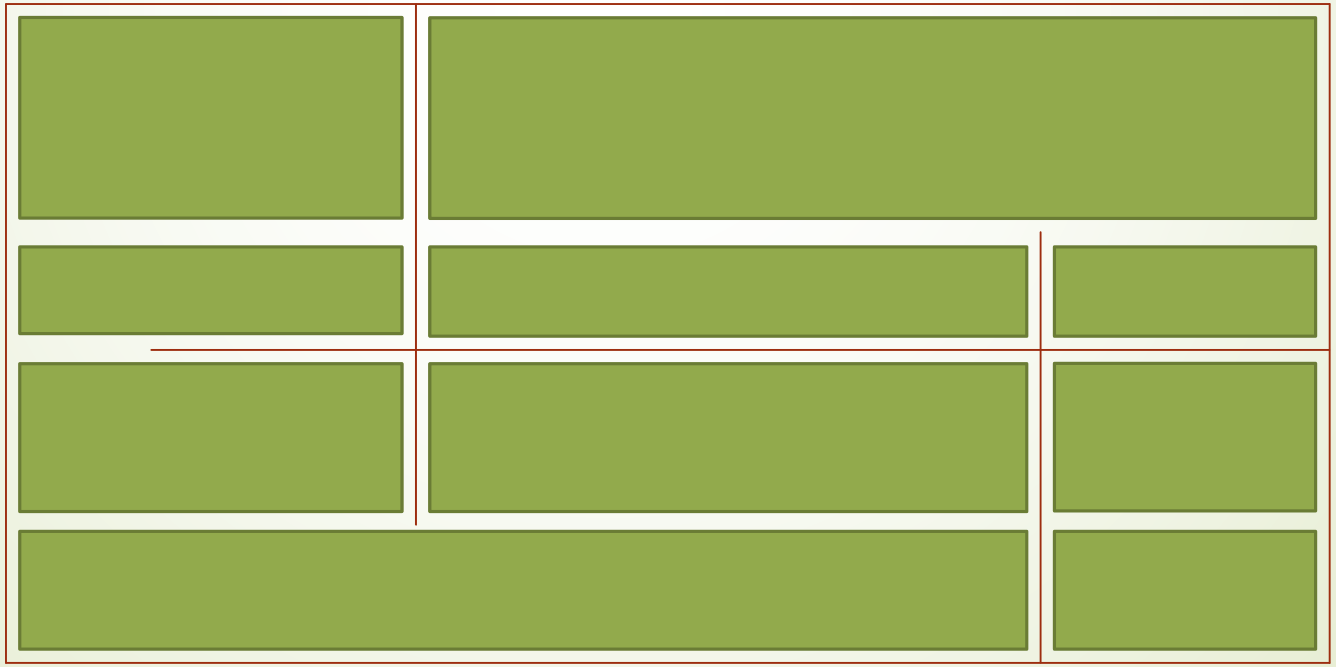
- 間違っても，分割された領域の形なんて考えたくない！
  - サンプルの「J」「O」「I」どころでない大変な形が出てくる
- 領域の分割は大変
- 逆に，領域をくっつけるのはそれ自体は簡単
  - 今は，領域の数にしか興味がない
  - Union Find が使える
- まず，領域を激しくぶった切って，あとでくっつけよう

領域をバラバラにする

- 長方形なら扱いやすい
- 領域を，長方形に分割して考えよう
- でも，さすがに全部 1 × 1 とかはあまりうれしくない

領域の分け方

- たとえば…
  - Y 座標については，全部すべての場所で区切る
  - Y 座標ごとには，X 座標は必要な部分だけで区切る



領域併合による自明な解法

- さっきみたいに予め領域をたくさんの部分にまず分割する
- 領域ごとに Union Find のノードを持たせる
- 隣り合っている場所をすべて調べて，くっつける
  - 幸い，Y 方向についてのみ見ればよい
  - X 方向は分割の段階で全部くっつけておいた

- O(N^2) から何もよくなっていない！

観察

- Y 座標ごとに，X 方向でどう分割されてるか眺める
- Y 座標が 1 動いただけでは分割の様子はほとんど変わらない
  - 分割の様子が変わるのは，Y 方向の線分が出たり消えたりするとき
- しかも Y 座標が 1 動いたくらいで領域を別扱いする必要もあまりない
  - X 方向の線分が邪魔しなければ同じ領域

平面走査へ帰着

- X 方向の分割の状態を持って，Y 座標を -∞ → ∞ と動かして処理する
  - 分割の状態は，適切なデータ構造を使う
- 動かしてる途中に，線分が出てきたらうまく対処する
- たとえば，「線が出てくる」「線が消える」「X 方向の線」などのイベントを並べておいて，ソートして最初から見る
- X 座標の状態をデータ構造に持って，Y 座標を動かすっていうのは頻出手法です
  - 2013 春合宿 Construction
  - 2012 春合宿 Fortune Telling など...

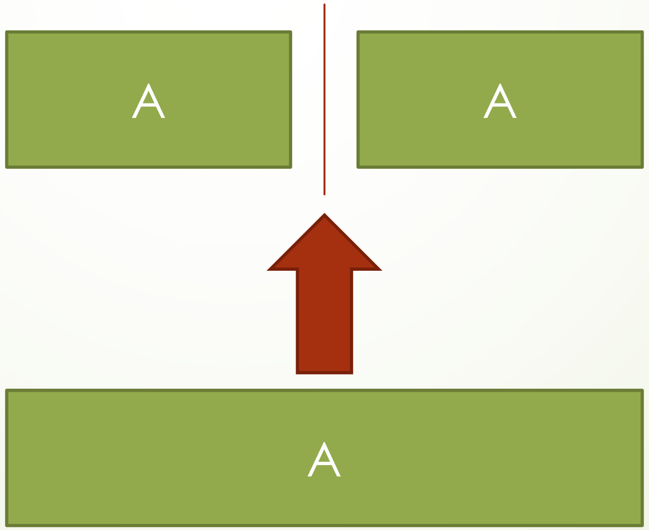
データ構造

- 各領域の左端，右端くらいは覚えててほしい
- Union Find のノードも覚えててほしい

- たとえば，set に (left, right, node) たちを放り込む
  - right は次(右)の領域の left なので特に覚えなくてもよい
  - set なので検索が O(log N) でできる

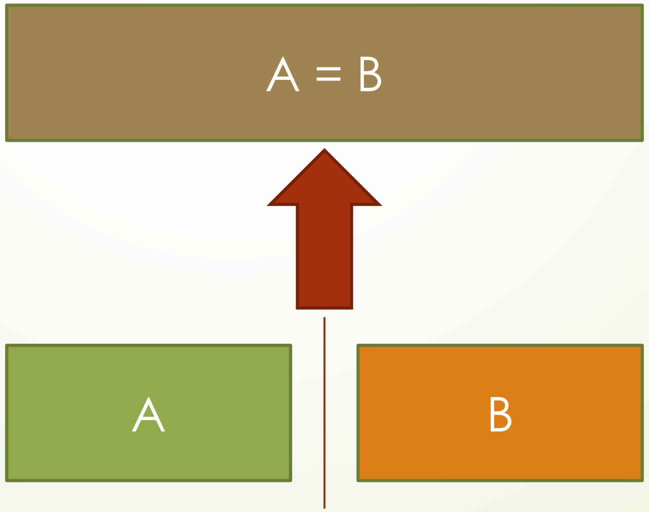
線分イベント(1): Y 方向線分の出現

- 領域は分割されます
- 分割後の 2 つの領域の Union Find のノードは両方とも分割前と同じ



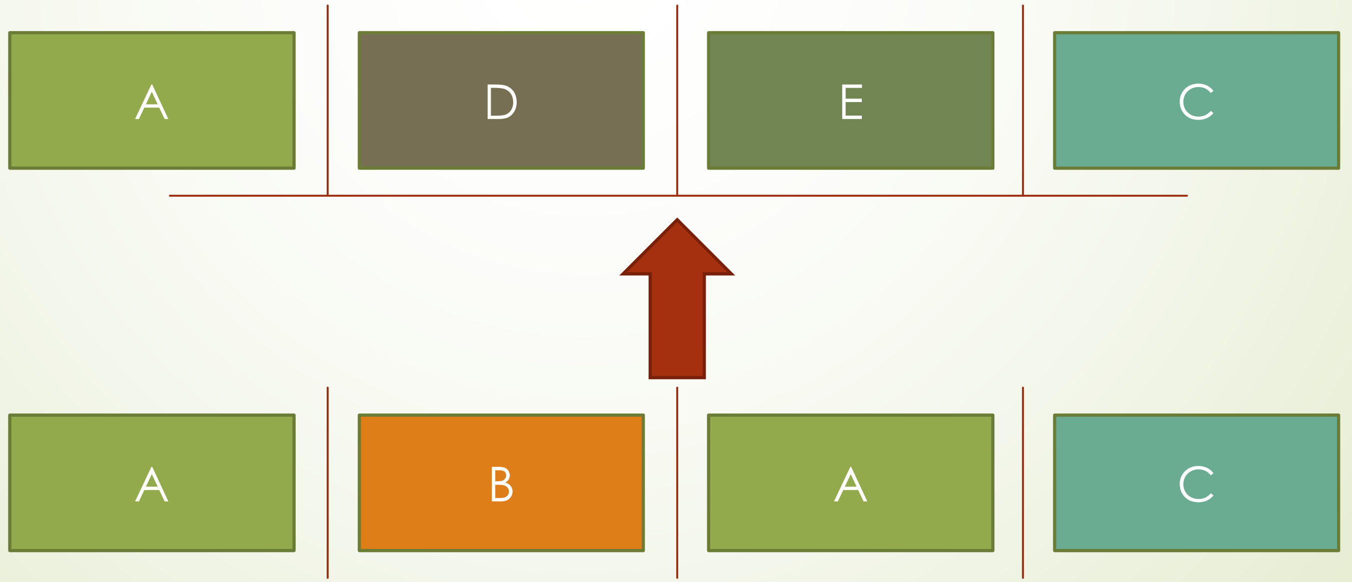
線分イベント(2): Y 方向線分の消滅

- 領域はくっつきます
- 元々の 2 つの領域の Union Find ノードは merge される
- 新たな領域に対応するノードは，今 merge したノード



線分イベント(3): X 方向線分

- 領域の状態に変化はありません
- が，Union Find ノードが新しくなります
- その線分によって完全に覆われる領域たちすべてのノードを新品にする
  - 古いものは，その領域に関わっていたことは忘れてしまう



データ構造に対する効率よい処理

- set を使ったとします
  - 自分で Segment Tree を書く場合は適切に機能を実装
- Y 方向線分が出てきたとき，それがどの領域に関わるかは O(log N) でわかる
- 領域の追加/削除も O(log N)
- Union Find の時間は定数みたいなもの

- X 方向線分については，「どこからどこまで更新する必要があるか」までは O(log N)
- その更新が大変
  - だいたい，線分たちの交点の数に比例する時間
- それでも，ここまでで小課題 3 は解けて，全部で 30 点

ボトルネック

- 「その線分によって完全に覆われる領域たちすべてのノードを新品にする」
- これがとんでもないネック　O(N^2 log N) の元凶
- 困ったことに，関係するノードは全部新品にしないといけない
  - 後でどこが使われるかわからない！

遅延処理

- 逆転の発想！！！
- 「ここらへんは新品」という情報だけ覚えておく
  - 新品は，使う前に新しいノードに更新する

- 範囲に対して操作をする Segment Tree でたまに使う手法
- 今回は，Segment Tree を媒体にノードの生成を遅延させて制御する

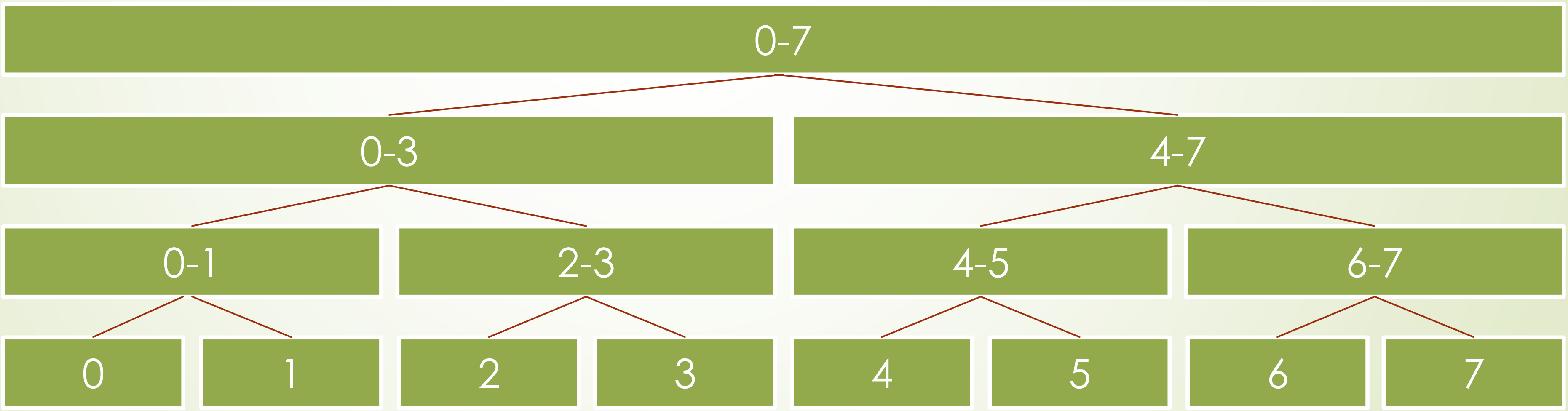
必要な処理

- 1. ある範囲のノードにすべて新品というフラグを立てる
- 2. ある位置のノードを新品でなくする

- ノードを読み出すときは，フラグを確認して新品だったら使う前に新しいものに変えておく
- 書き込むときは，ノードを新品でなくしてから書き込む

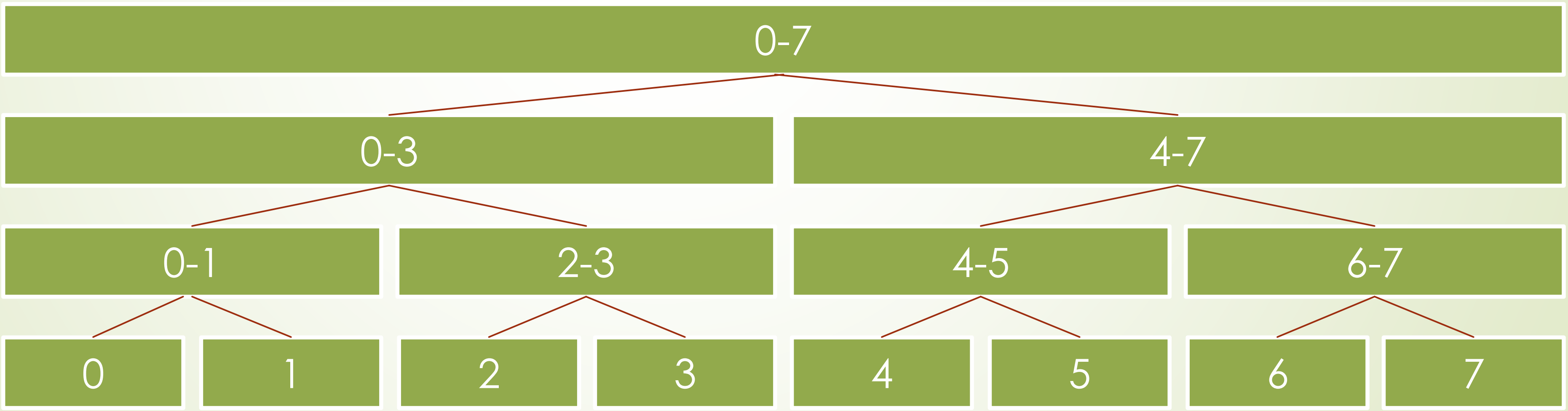
Segment Tree の作り方

- 構造は普通のものと同じで，ツリーの節に範囲の情報を持たせる
- 節にフラグが立っていたら，その範囲全部新品



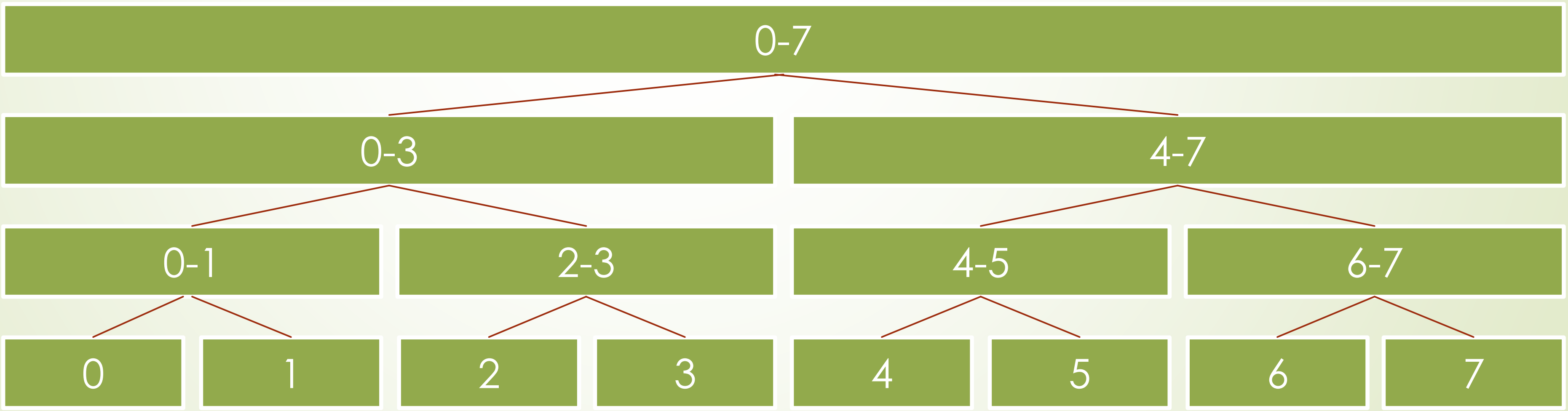
Segment Tree の更新 (1)

- ある範囲にフラグを立てるとき
- 一番上から節を見ていって，今の節の範囲が操作範囲に全部含まれていたらフラグを立て，全くかぶってなかったらやめ，微妙にかぶってる場合は下の両方の節に対して操作する



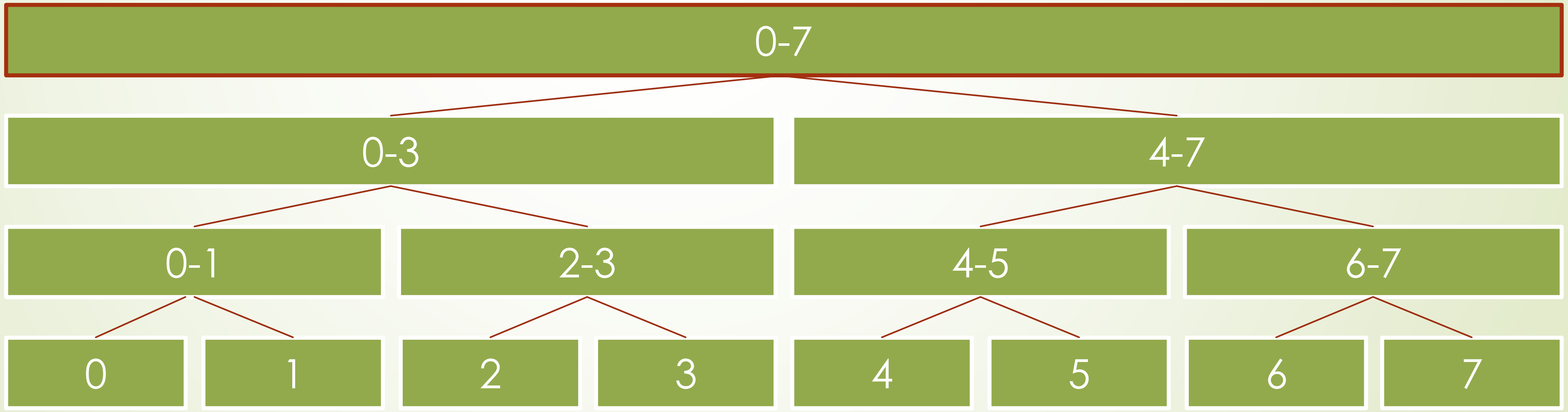
Segment Tree の更新 (2)

- 2, 3, 4, 5, 6, 7 に対してフラグを立てる場合の例



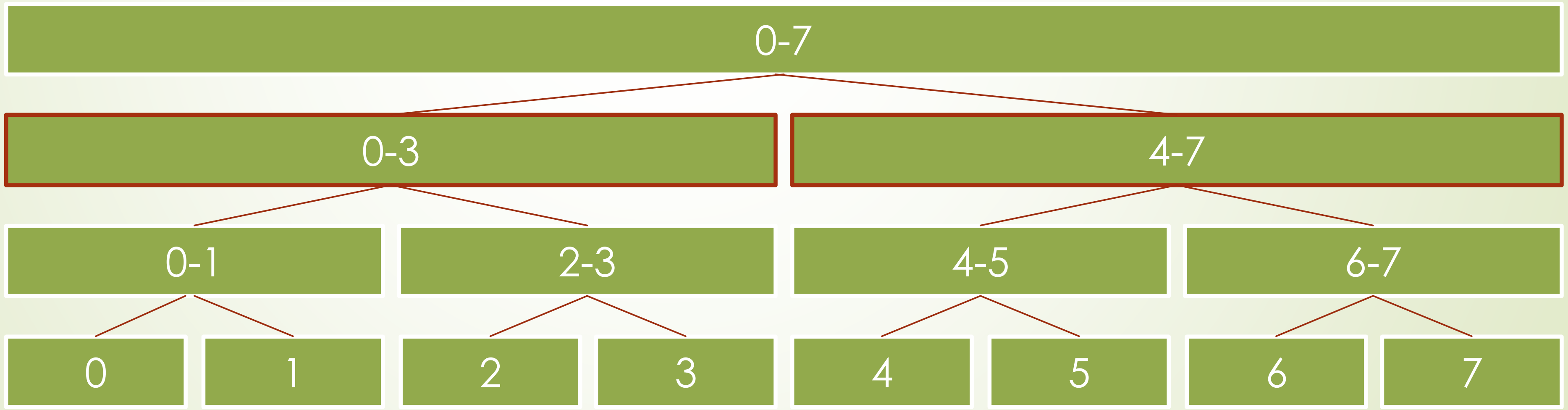
Segment Tree の更新 (2)

- 2, 3, 4, 5, 6, 7 に対してフラグを立てる場合の例



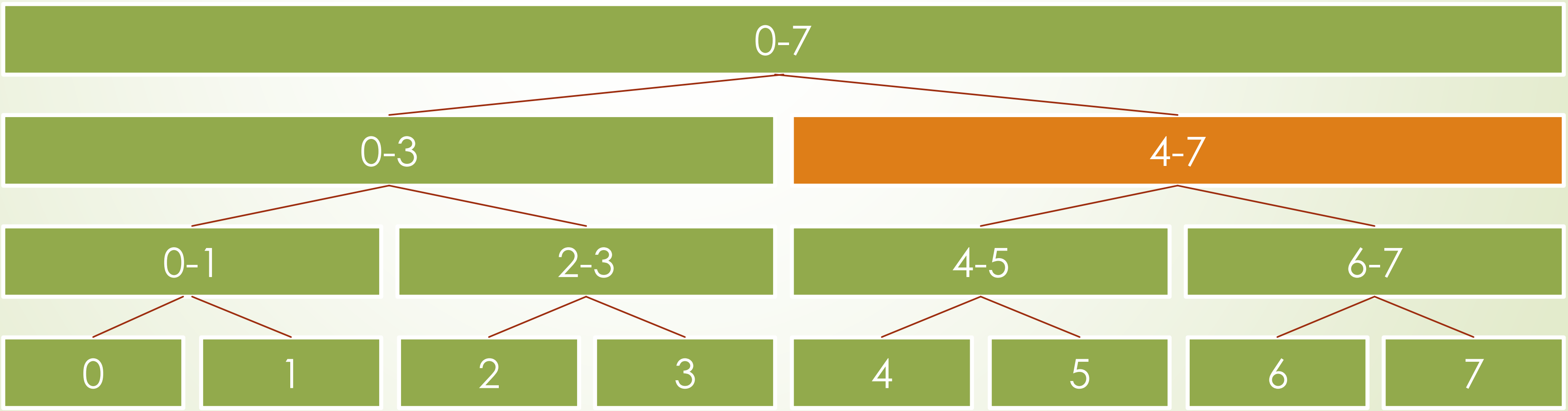
Segment Tree の更新 (2)

- 2, 3, 4, 5, 6, 7 に対してフラグを立てる場合の例



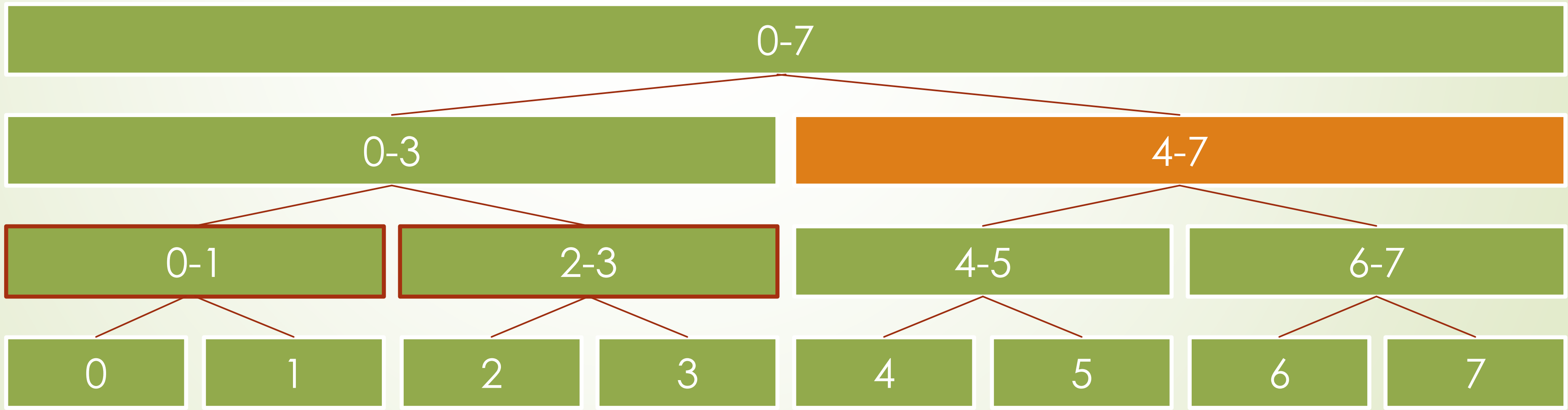
Segment Tree の更新 (2)

- 2, 3, 4, 5, 6, 7 に対してフラグを立てる場合の例



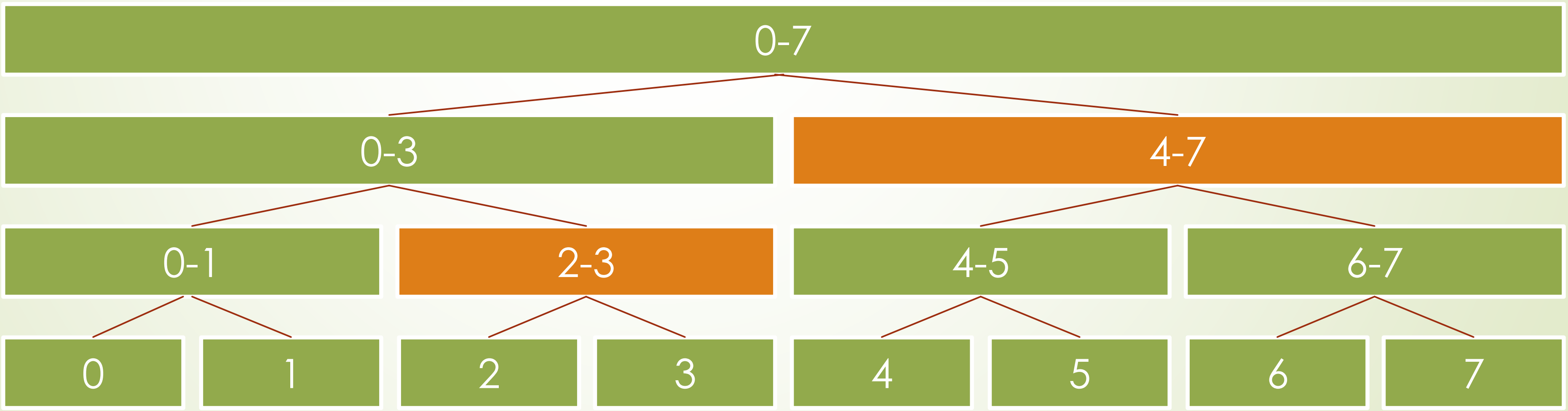
Segment Tree の更新 (2)

- 2, 3, 4, 5, 6, 7 に対してフラグを立てる場合の例



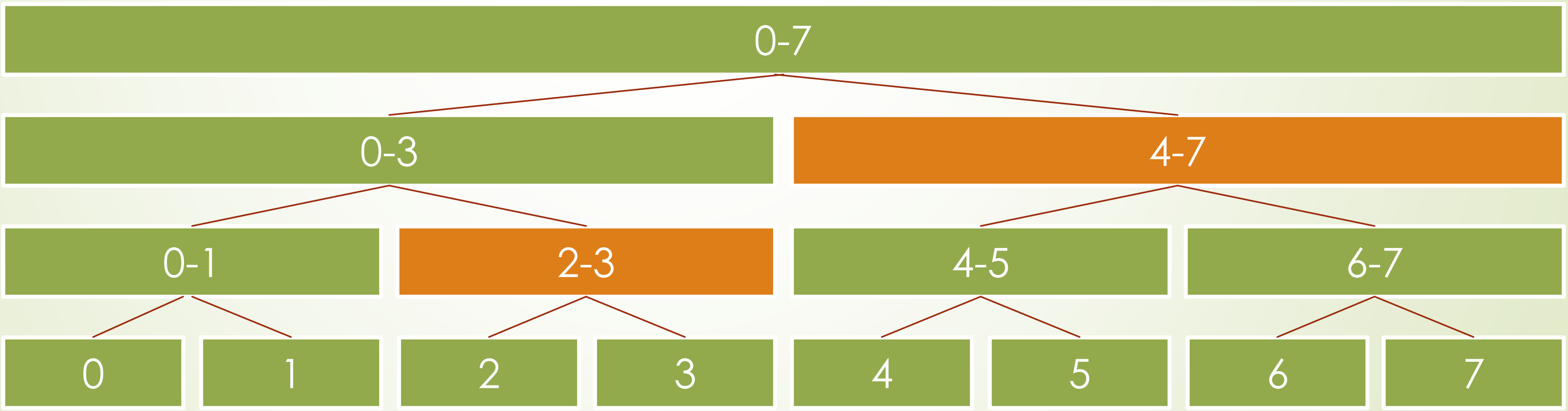
Segment Tree の更新 (2)

- 2, 3, 4, 5, 6, 7 に対してフラグを立てる場合の例



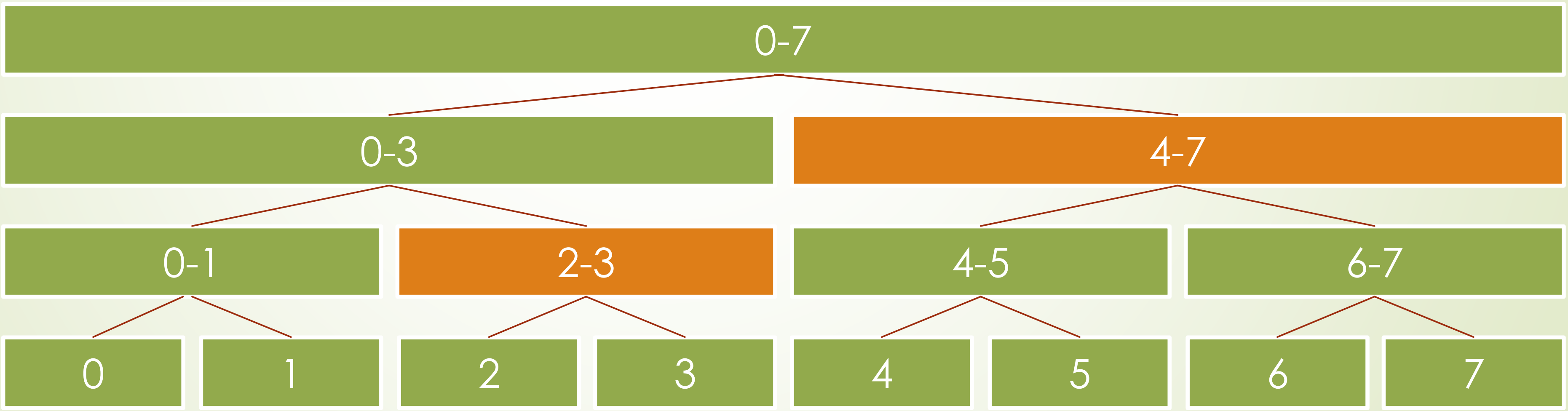
Segment Tree の更新 (3)

- ある位置のフラグを取り除くとき
- その位置を含む節たちを上から見ていって，フラグが立っている節があったら消し，左右の子にフラグを立てて，次の節を見る



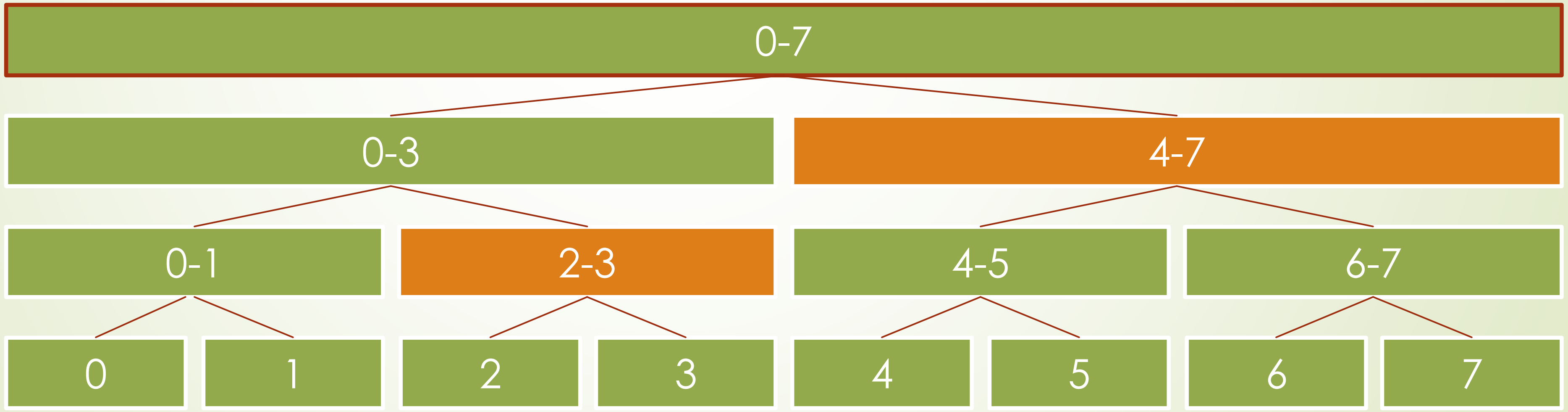
Segment Tree の更新 (4)

- 6 のフラグを取り除く場合の例



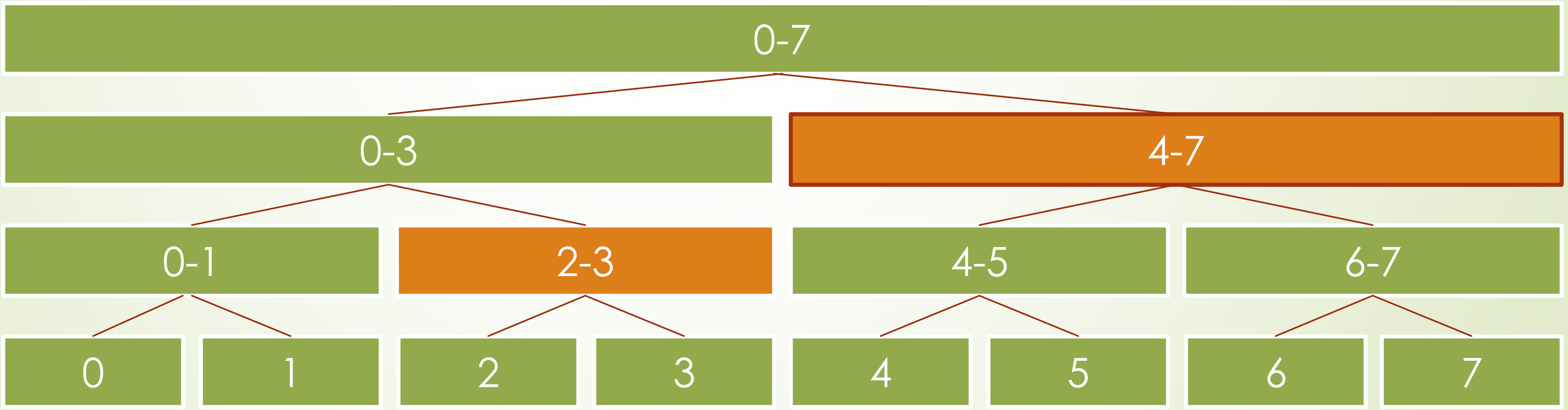
Segment Tree の更新 (4)

- 6 のフラグを取り除く場合の例



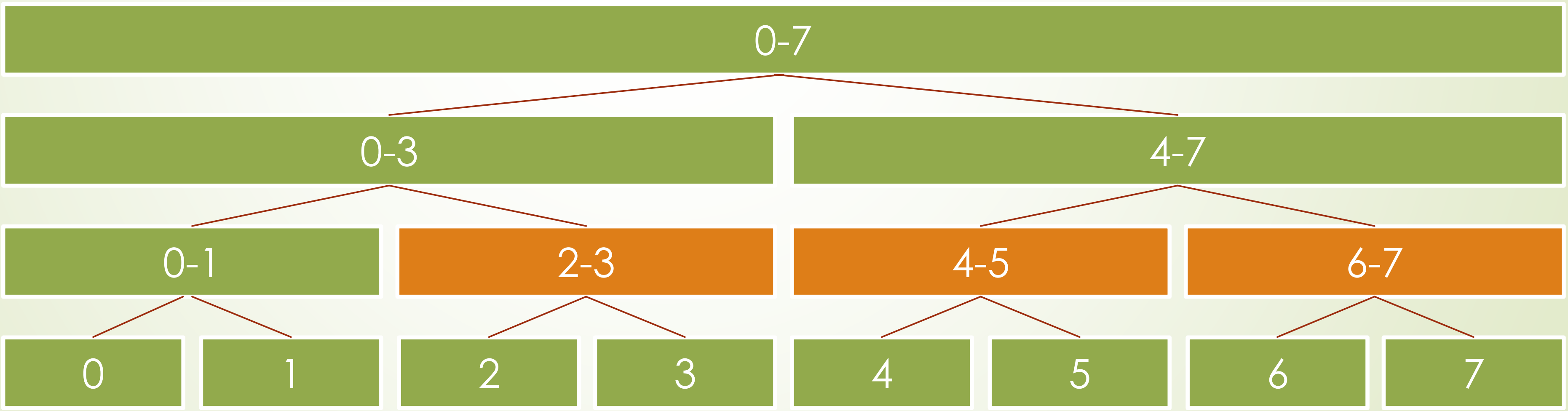
Segment Tree の更新 (4)

- 6 のフラグを取り除く場合の例



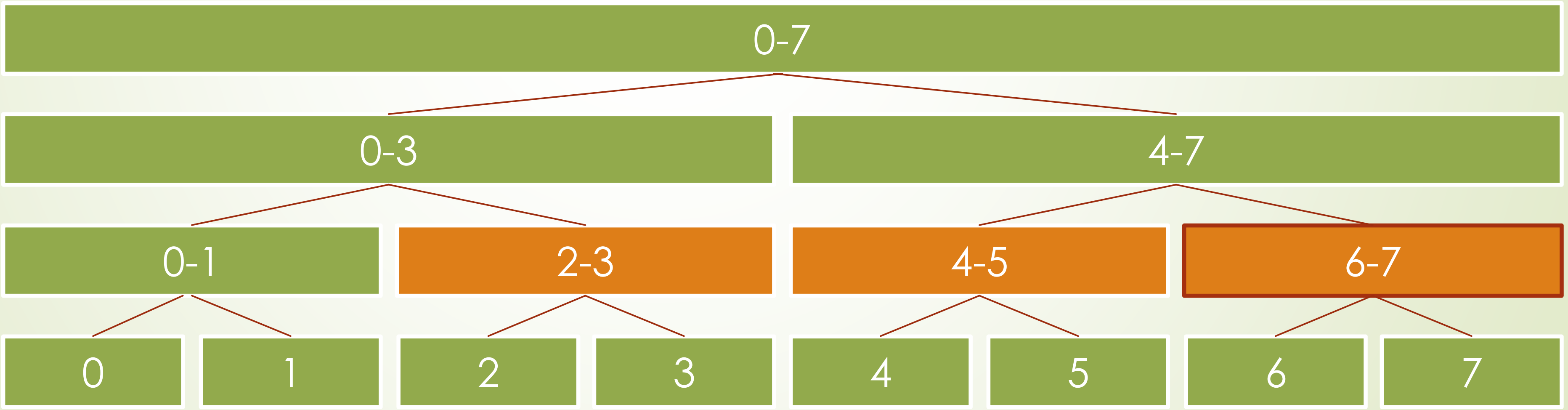
Segment Tree の更新 (4)

- 6 のフラグを取り除く場合の例



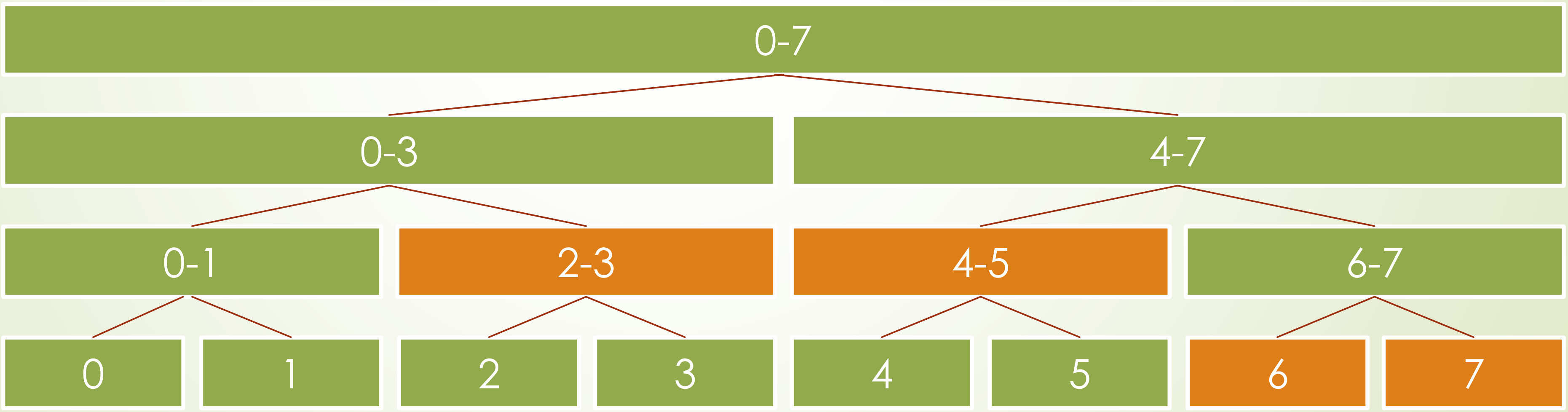
Segment Tree の更新 (4)

- 6 のフラグを取り除く場合の例



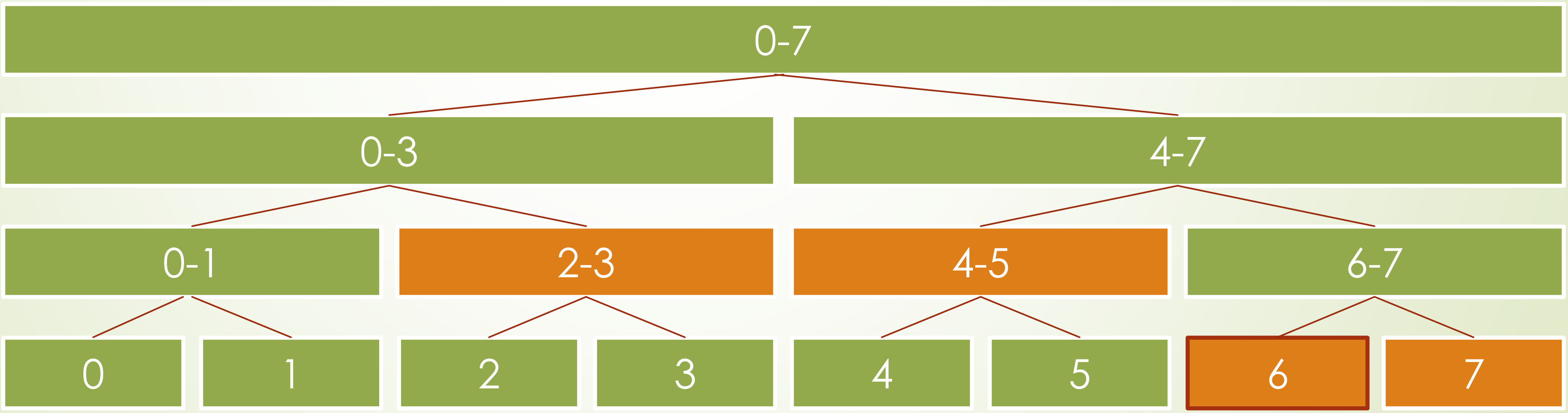
Segment Tree の更新 (4)

- 6 のフラグを取り除く場合の例



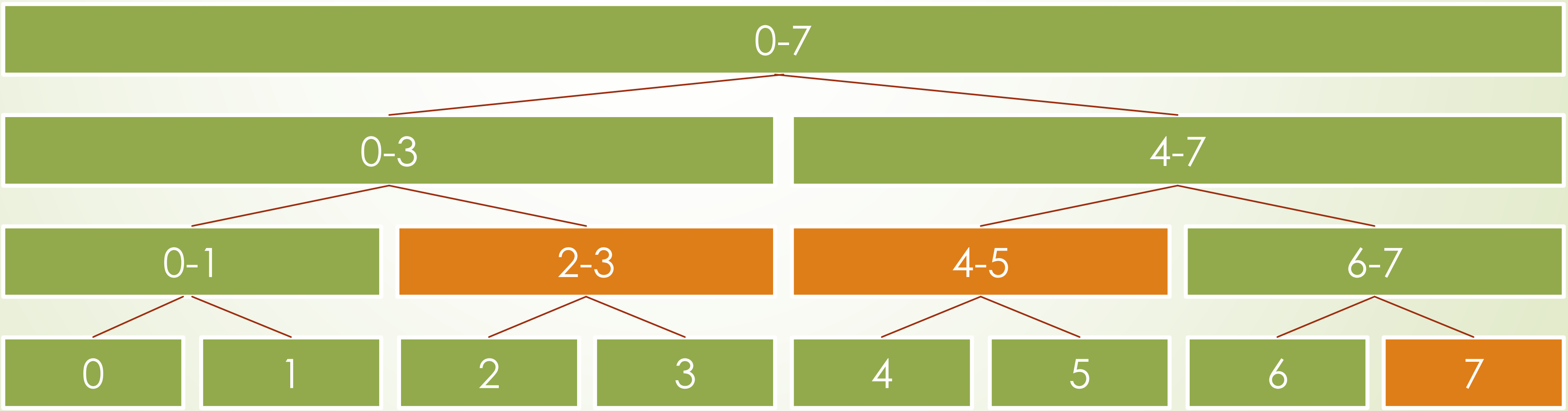
Segment Tree の更新 (4)

- 6 のフラグを取り除く場合の例



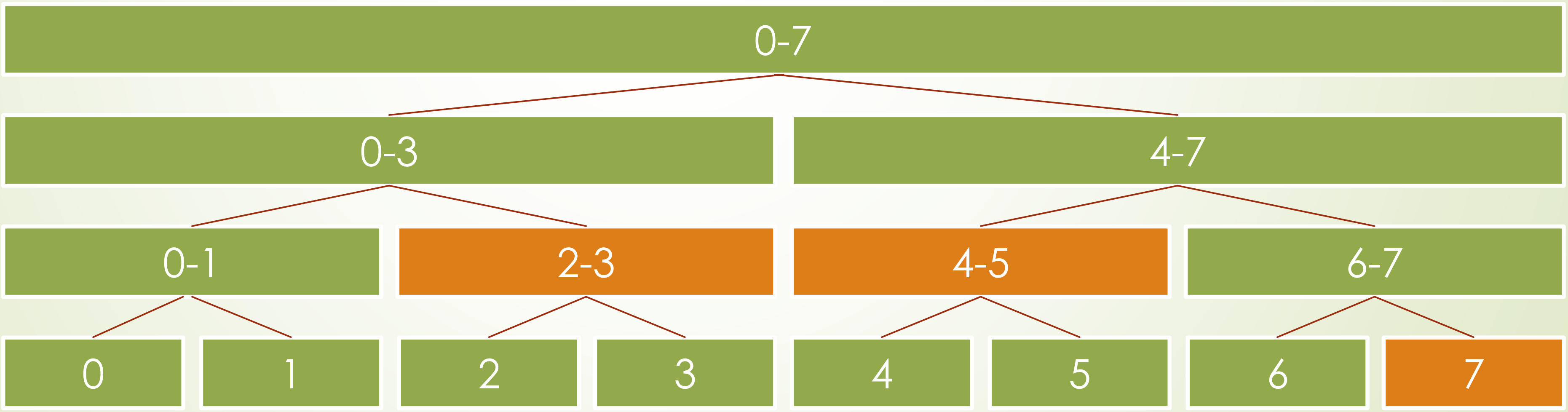
Segment Tree の更新 (4)

- 6 のフラグを取り除く場合の例



Segment Tree の参照

- ある位置にフラグが立っているか知りたいとき
- その位置を含む節たちをすべて調べて，１つでもフラグが立っていればフラグが立っている

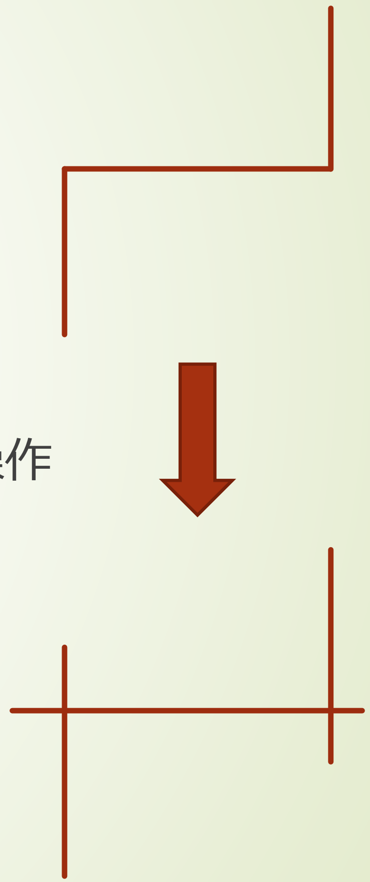


諸注意

- 線分が端点で交わってる場合
- 少しだけ伸ばしましょう

- あるいは，同じ y 座標に出てきたとき適切な順番で操作
- Y 方向線分出現→X 方向線分→Y 方向線分消滅，の順で操作

- 同じ種類のものが同じ位置にある場合はどの順番で操作してもよいです



まとめ

- この Segment Tree を使って，Union Find のノードの生成を遅らせる
- 操作の回数は O(N)
- それぞれの操作は O(log N)
- 全部で O(N log N)

- これで 100 点

別解

- 「長方形の紙を除去」のところから分岐
- 分割数を別な方法で求められないか？

観察 (1)

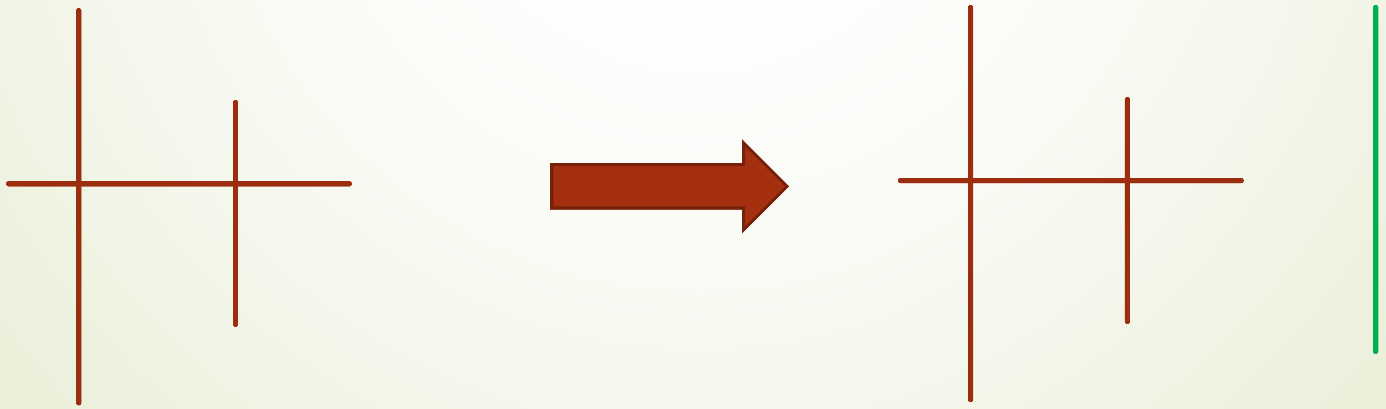
- 平面にでたらめに線分（切り取り線）をたくさん描いてみて観察
- 領域の分割数，線分の数，線分の交点数，線分の連結成分数に注目

- 何もないときは領域は 1 個に分かれる

観察 (2)

- 線分を描いたが，その線分が何とも交わっていないとき

- 領域の数は変わらない
- 辺の数は 1 増える，線分の連結成分は 1 増える，交点は増えない



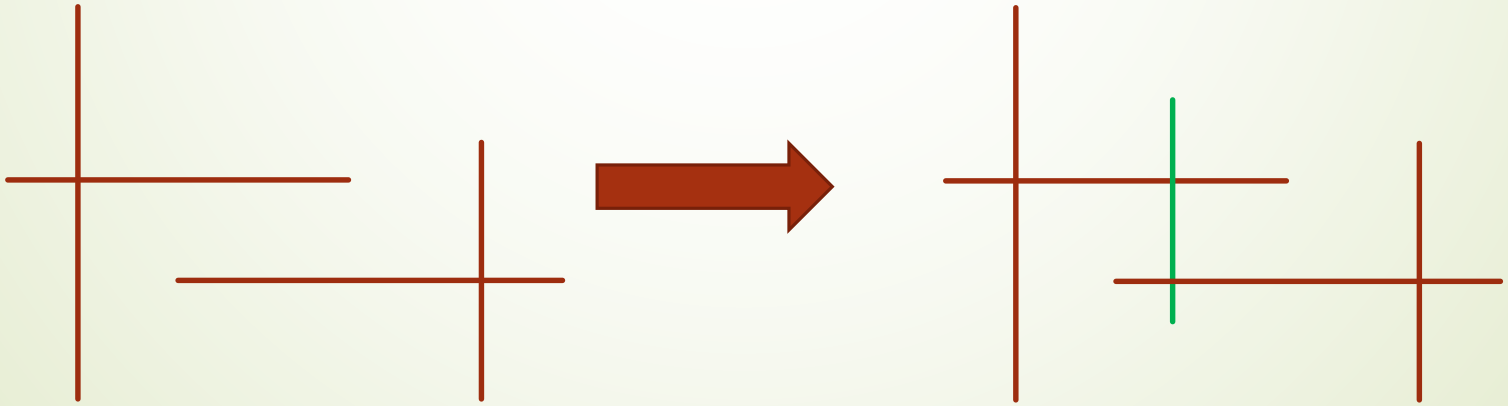
観察 (3)

- 他の線分との交点が 1 個しかない線分を描いたとき

- 領域の数は変わらない
- 辺の数は 1 増える，線分の連結成分は増えない，交点は 1 増える

観察 (4)

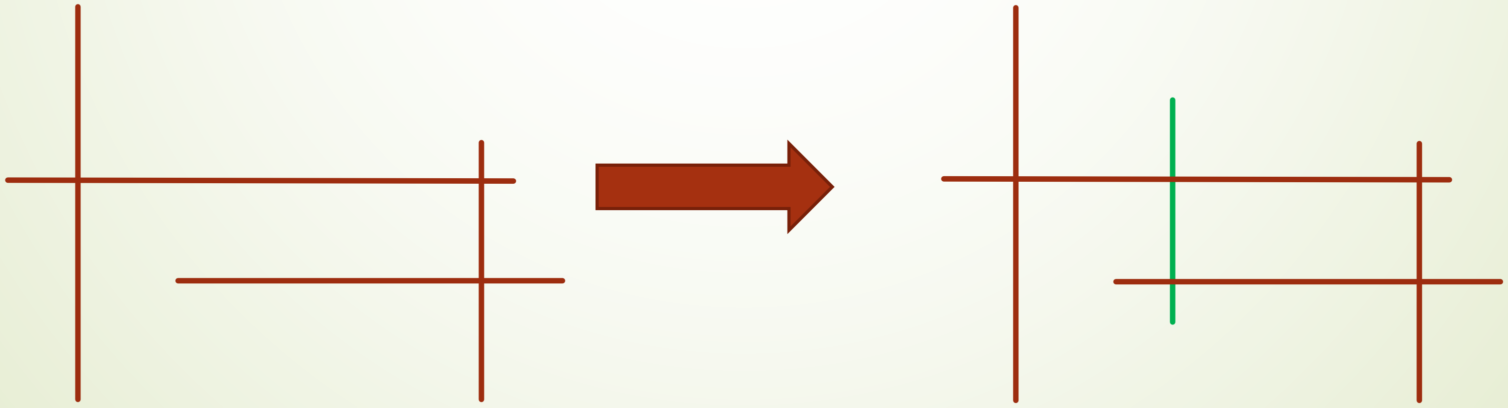
- 2 つの異なる連結成分を結ぶように線分を描いたとき
- 領域の数は変わらない
- 辺の数は 1 増える，線分の連結成分は 1 減る，交点は 2 増える



観察 (5)

- 同じ連結成分内を結ぶように線分を描いたとき

- 領域の数は 1 増える
- 辺の数は 1 増える，線分の連結成分は変わらない，交点は 2 増える



観察 (6)

- 表にしてみます

領域の数	交点の数	線分の数	連結成分数
±0	±0	+1	+1
±0	+1	+1	±0
±0	+2	+1	-1
+1	+2	+1	±0

- (交点の数) + (連結成分数) – (領域の数) - (線分の数) = 一定，が成り立ってるように見える
- その定数は -1

Magical Formula

- 結局，領域の分割数は,

  (分割数) = (交点の数) - (切り取り線の数) + (切り取り線の連結成分数) + 1

  で求められる
- 「Euler の定理」と呼ばれています
- 交点の数は，Segment Tree を使って平面走査を行えば求められる
  - 詳細は省略します（最初の解の方法とだいたい同じ）
- 切り取り線の連結成分の数が問題

小課題 3, 4

- 小課題 4 はもう解けてます
  - 条件に，連結成分の数が 1 と書いてある
  - これで 20 点
- 小課題 3 もほとんど解けたも同然
  - 平面走査を行うときに，交差しているものを Union Find でまとめる
  - 別な 20 点
- 小課題 3 で解けなさそうなら小課題 4 で解く，とかすると両方の点が得られる

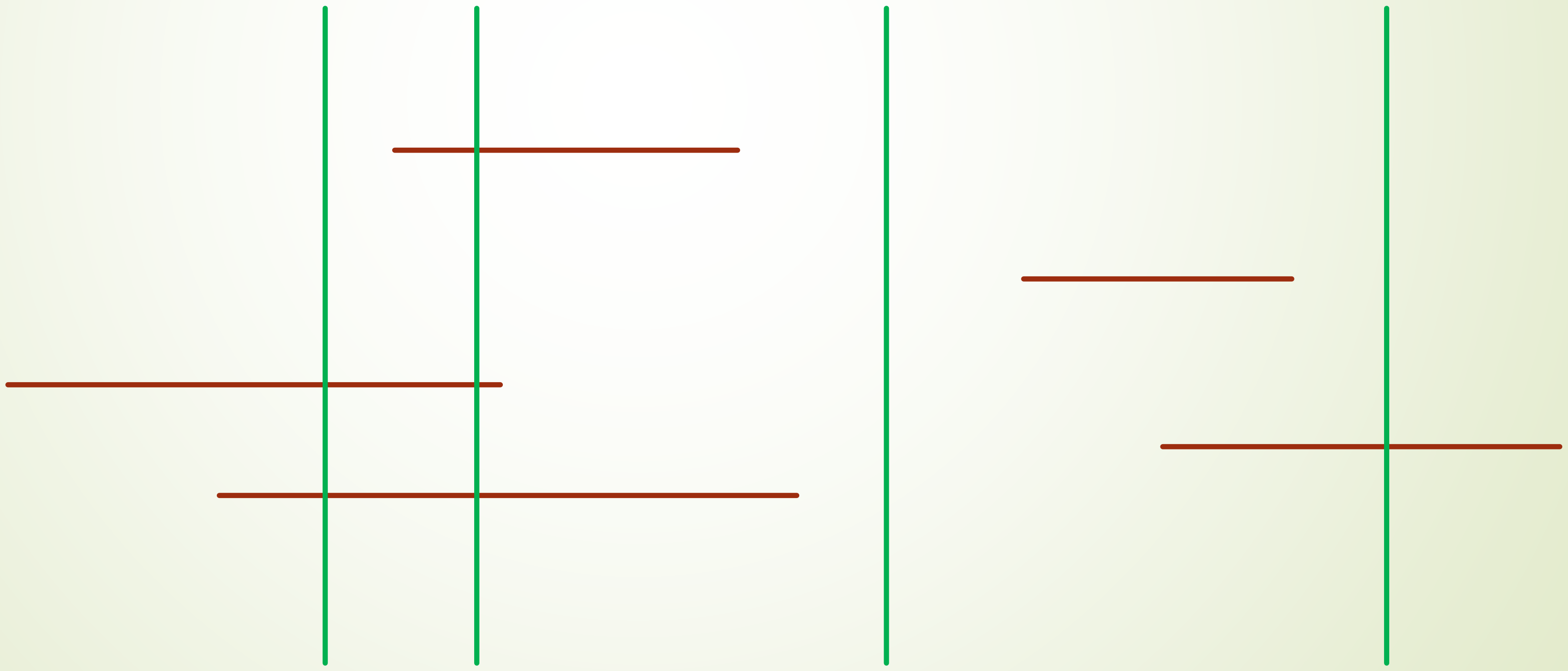
- ここまでで，50 点

満点に向けて

- 切り取り線の連結成分の数を高速に求めないといけない
- 交点の数は最大 O(N^2) になりうるので普通に求めるわけにはいかない

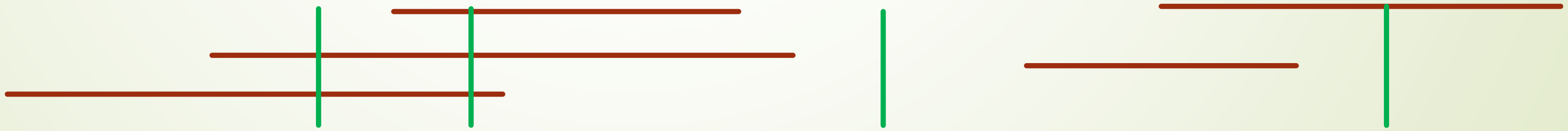
極端な場合の考察 (1)

- Y 方向の線分がすべて (-∞, ∞) を占める（無限に長い）場合



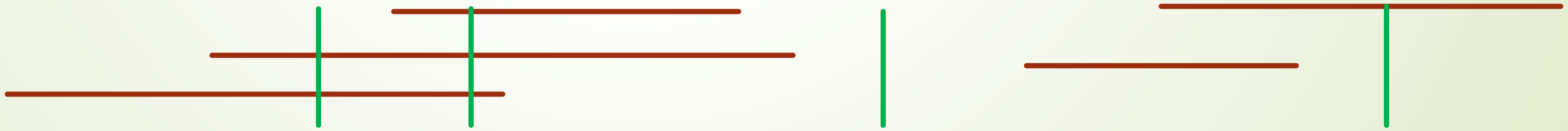
極端な場合の考察 (2)

- Y 座標はもう忘れてよい
- Y 方向の線分(直線?)は，そこにある X 方向線分たち同士(およびその線分)をつなげる働きをする
- この上で，連結成分の数を求めたい



極端な場合の考察 (3)

- X 座標が小さいほうから見ていく
  - X 方向線分が現れたり消えたりしたらその都度操作
  - Y 方向線分が現れたら，その線分および今ある X 方向線分をすべて同じ連結成分にする
- これも最悪 O(N^2) 回の併合操作が必要

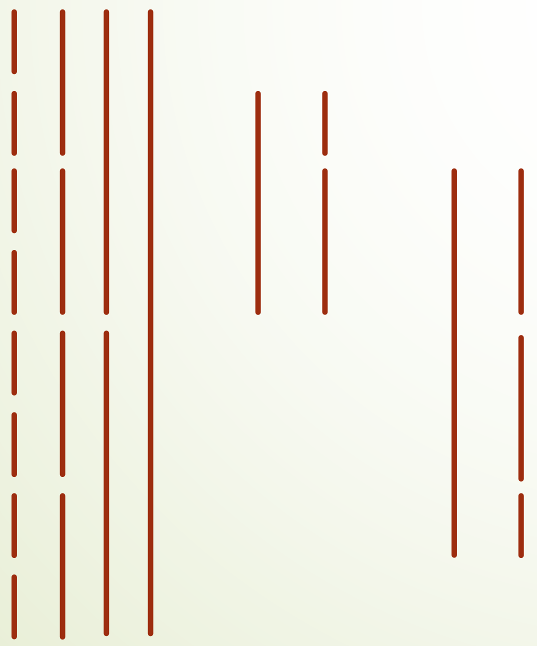


極端な場合の考察 (4)

- 意味のない併合操作を何回も行うことになる
- 少し考えると，併合操作を行った後も，今ある X 方向線分を全部覚える必要はない！
  - 一番最後まで残るものだけ残して捨ててしまってよい
- これを行うと，併合操作の回数は O(N) 回になる
- あらかじめ線分の出現位置でソートを行うので，O(N log N)

線分の分解

- Segment Tree の要領で Y 方向の線分たちをぶった切る
- 切断された断片が取りうる範囲は O(N) 種類ある
- 各線分は O(log N) 個に分断される



X 方向線分の取り扱い

- X 方向線分は，その Y 座標を含んでいる範囲すべてに割り当てる
- その上で，各範囲に割り当てられている線分たちを併合処理する
- 同じ「範囲」内では，X 座標さえ重なっていれば併合が行える
  - さっきの「極端な場合」の解法が使える
- これで，交わる線分たちはすべて併合される

計算量解析

- 線分はそれぞれ O(log N) 個に増える → 全部で O(N log N) 個
- 各 Y 座標範囲について (K 個のものが入っている範囲の場合)
  - ソートに O(K log K) : 範囲全部では最悪 O(N log^2 N)
  - 順番にたどるのは O(K × (Union Find)) : 範囲全部では O(N log N)
- 全部で O(N log^2 N)，これでも通る

- 100 点

得点分布