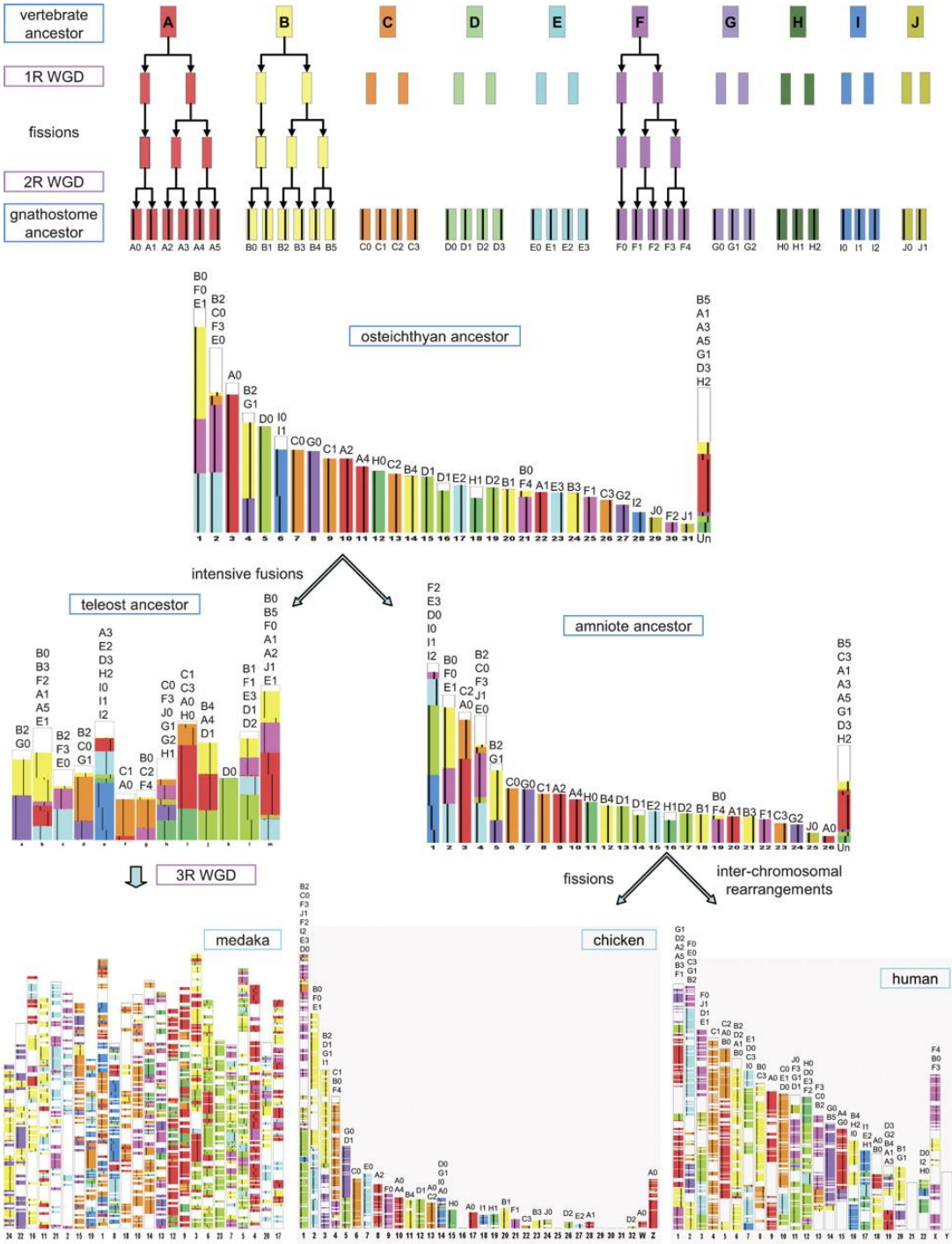
10: Chaining
Seeded Alignment の比較ゲノム学への応用

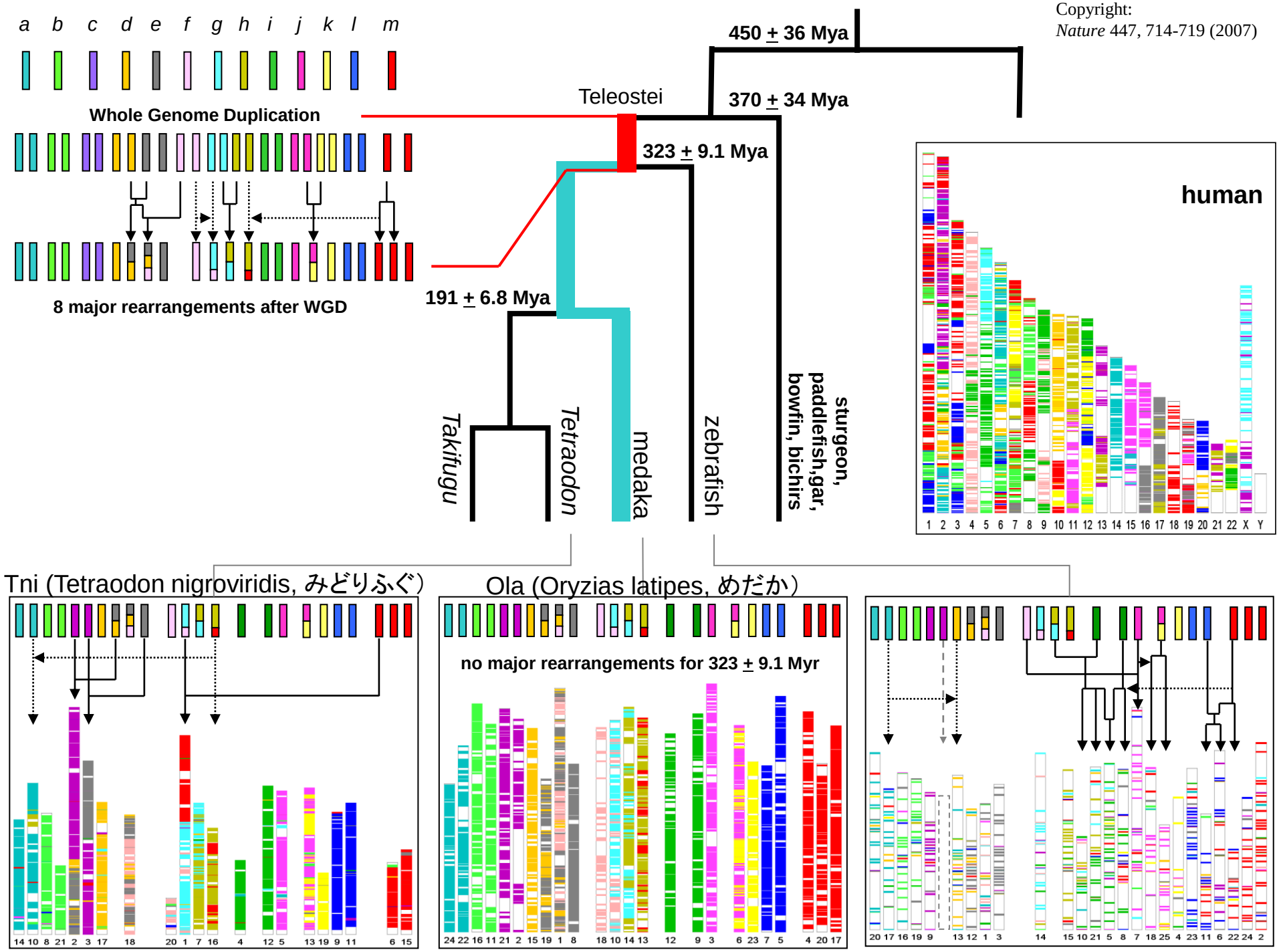
- 複数の生物種ゲノムの間で、類似領域を高速に探す
- ゲノムが進化する過程を描出できる
  （哺乳類、脊椎動物、昆虫、植物の理解が進む）
  ⇒　進化的に保存された遺伝子群のプロファイル化
- ヒトとチンパンジーの比較ゲノム
  保存されている領域のなかで、他と比べてヒトゲノムだけで進化スピードが速い領域を探す
  ⇒　ヒトの知能を育んだ遺伝子を探すヒント
  　　HAR1, *FOXP2* 等（2009年8月号日経サイエンス）

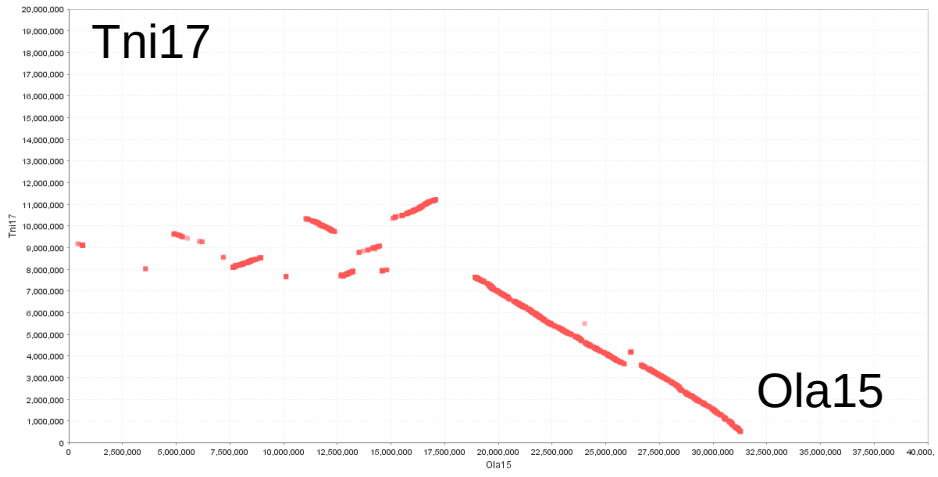
Reconstructed ancestral chromosomes.

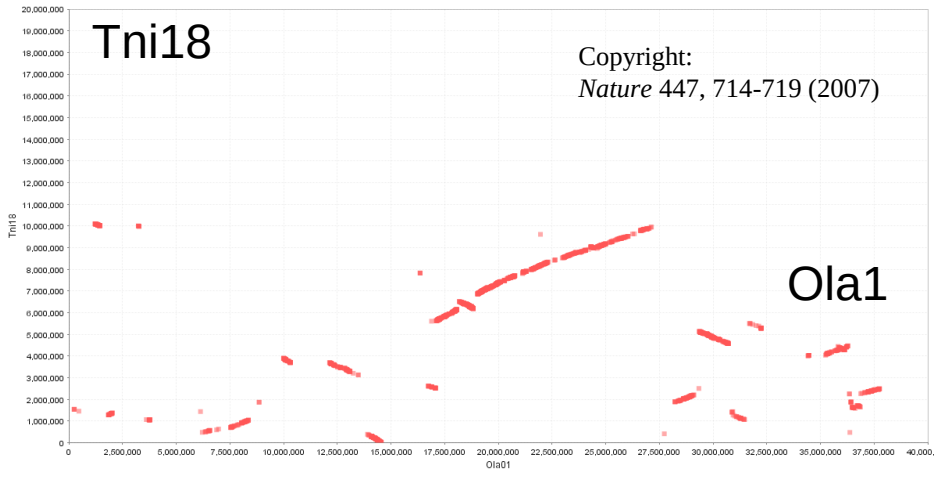


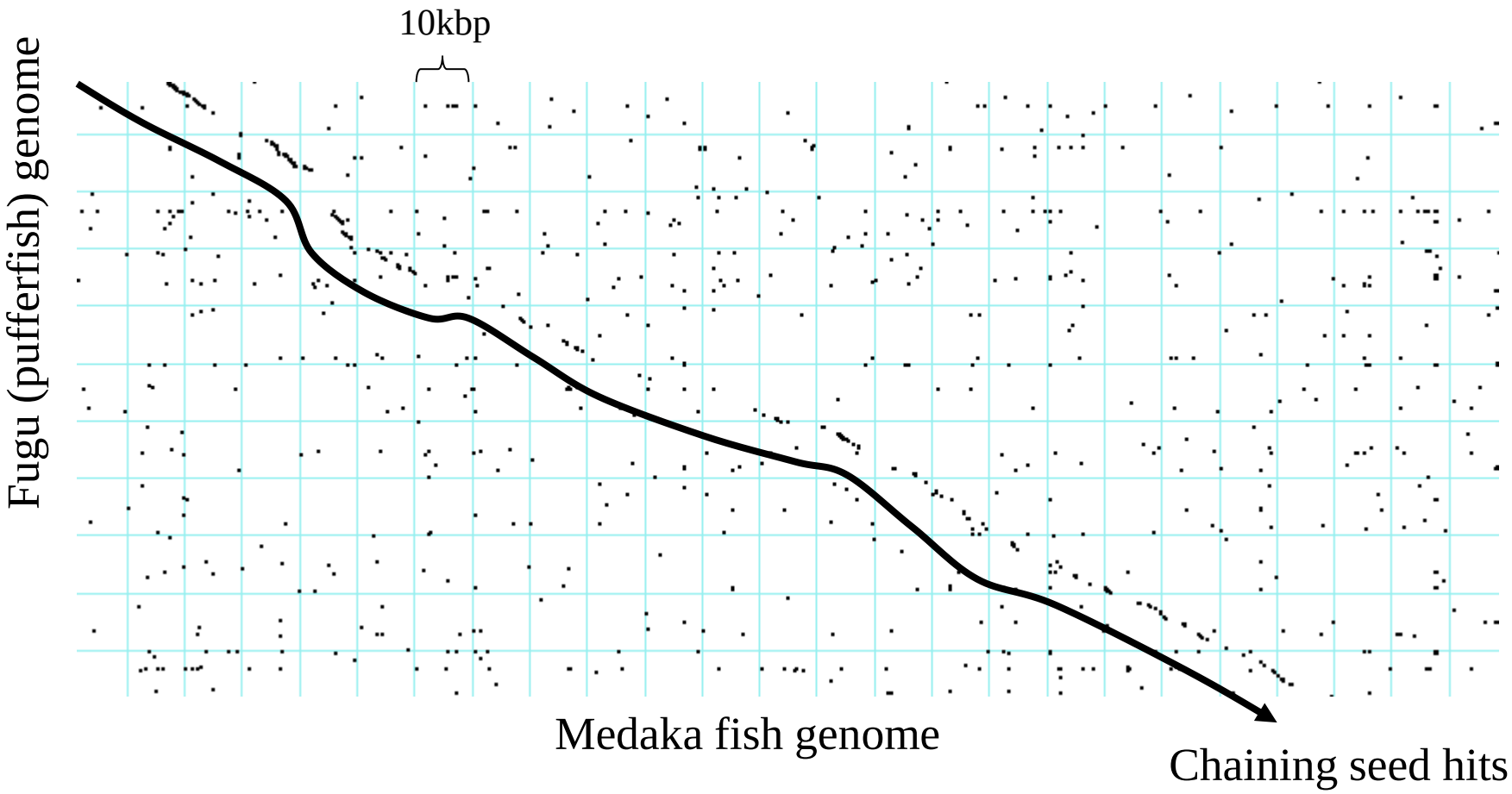
Reconstructed ancestral chromosomes. Ancestral vertebrate chromosomes A, B, and F had two alternative scenarios, fusions or fissions, between the 2R WGD events, as shown in Fig. 3. Thus, the number of proto-chromosomes ranges from 10 to 13 depending on the choice of two alternatives. The figure illustrates the scenario in which only fissions took place. Ten reconstructed proto-chromosomes in the vertebrate ancestor shown at the top are assigned distinct colors, and their daughter chromosomes in the gnathostome ancestor are distinguished by their respective vertical bars. In the genomes of the osteichthyan, teleost, and amniote ancestors, and human, chicken, and medaka genomes, genomic regions are assigned colors and vertical bars that represent correspondences of individual regions to the proto-chromosomes in the gnathostome ancestor from which respective regions originated. Unassigned blocks are shown in the rightmost chromosome (Un) in the osteichthyan and amniote ancestors.

Genome Res. 2007;17:1254-1265

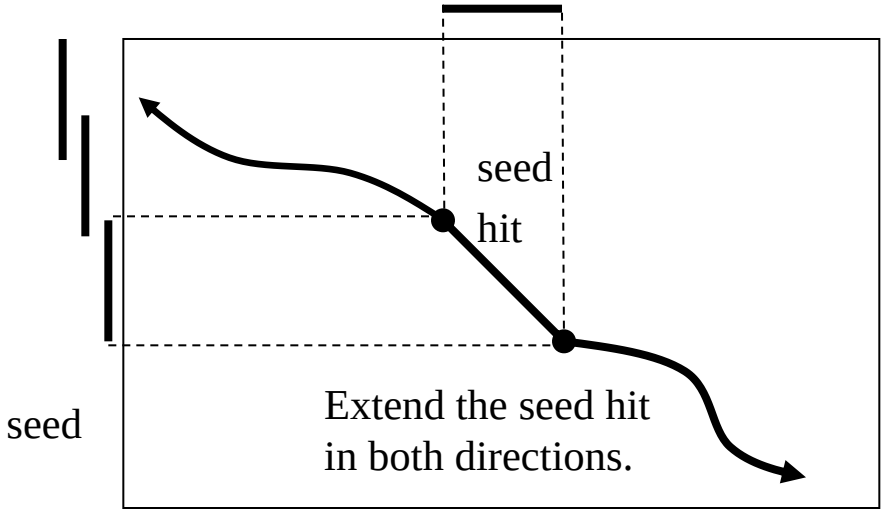


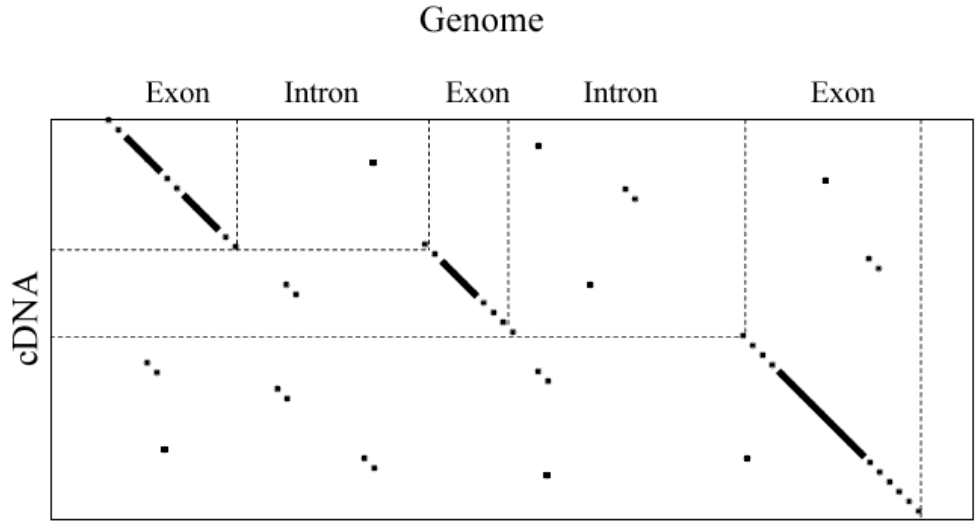




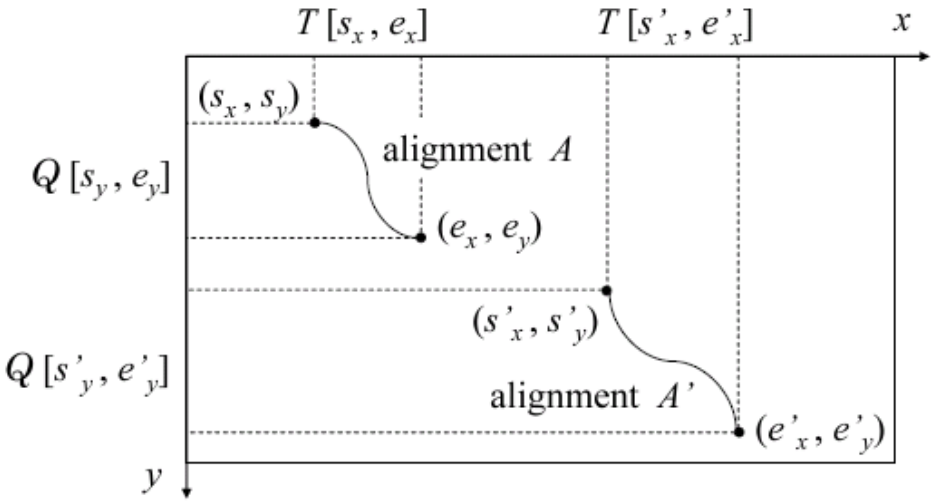


- ゲノムを比較する際には、相同性の高い長い領域を抽出したい
  ギャップの数は少なくしたい
- Gotoh のアルゴリズム　ギャップ数を最小化した解を出力
  しかし最悪時間計算量 $O(mn)$　m と n は比較する配列の長さ
  現実的な時間では、大規模ゲノムを分析できない
- 近似解を出力する高速な方法が必要





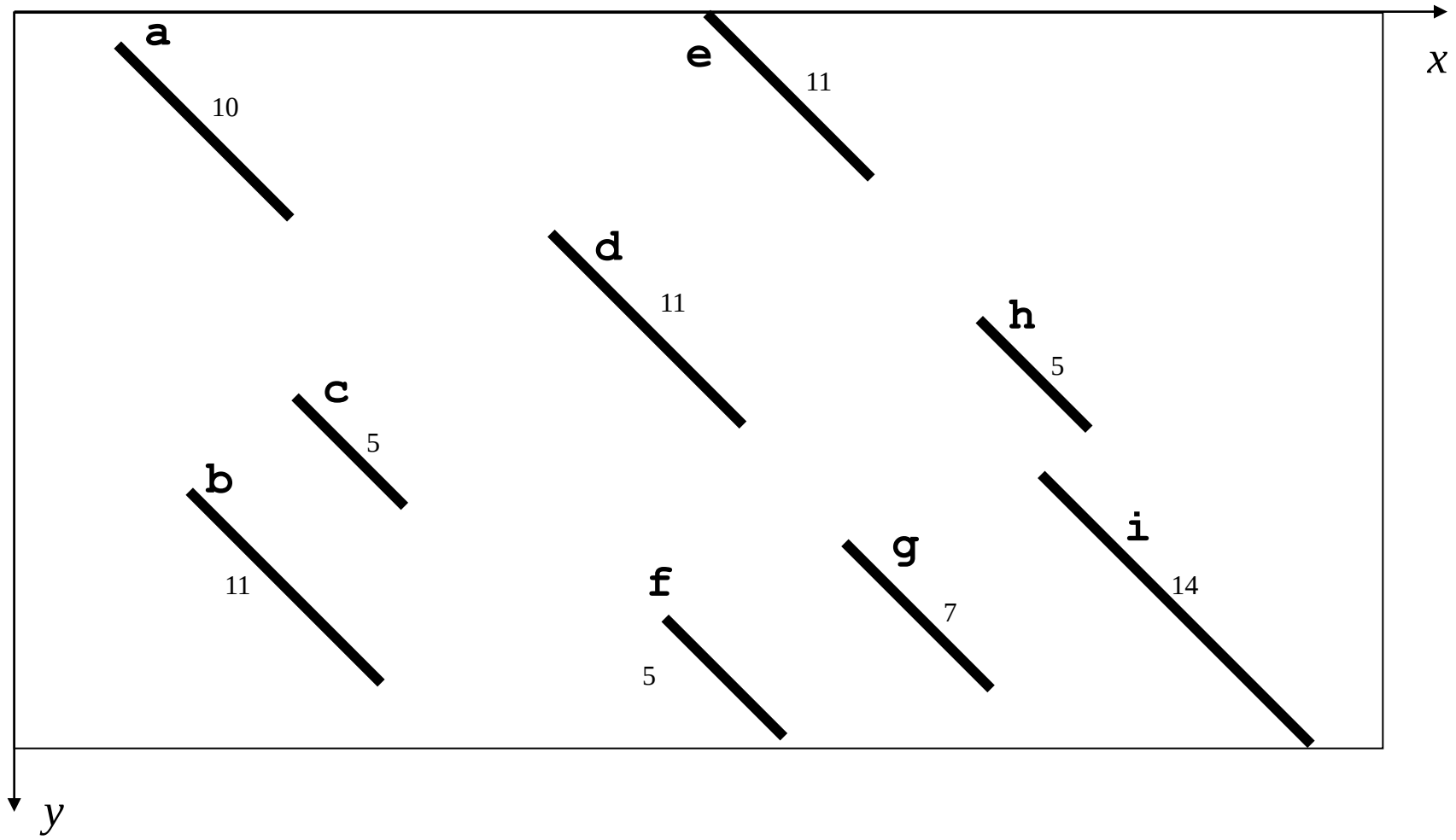
Definition 7.3 Let T and Q be two input strings. A local alignment of the substrings $T[s_x, e_x]$ and $Q[s_y, e_y]$ is represented as a path from (s_x, s_y) to (e_x, e_y) in the edit graph, and let $(s_x, s_y) \rightarrow (e_x, e_y)$ denote the local alignment. (s_x, s_y) and (e_x, e_y) are called the *startpoint* and *endpoint* of A. Local alignment A, $(s_x, s_y) \rightarrow (e_x, e_y)$, is a *predecessor* of local alignment A', $(s'_x, s'_y) \rightarrow (e'_x, e'_y)$, if both $e_x \leq s'_x$ and $e_y \leq s'_y$ hold. This predecessor relationship is denoted by $A \prec A'$, and A' is called a *successor* of A. A *chain* is a sequence of alignments $A_1 \prec A_2 \prec \ldots \prec A_k$.



Use seed matches for alignments.

Each alignment is associated with a real value score. The score of a chain is defined as the sum of scores of all alignments in the chain. Our objective is to compute the optimal chain ending at each alignment that maximizes the score.

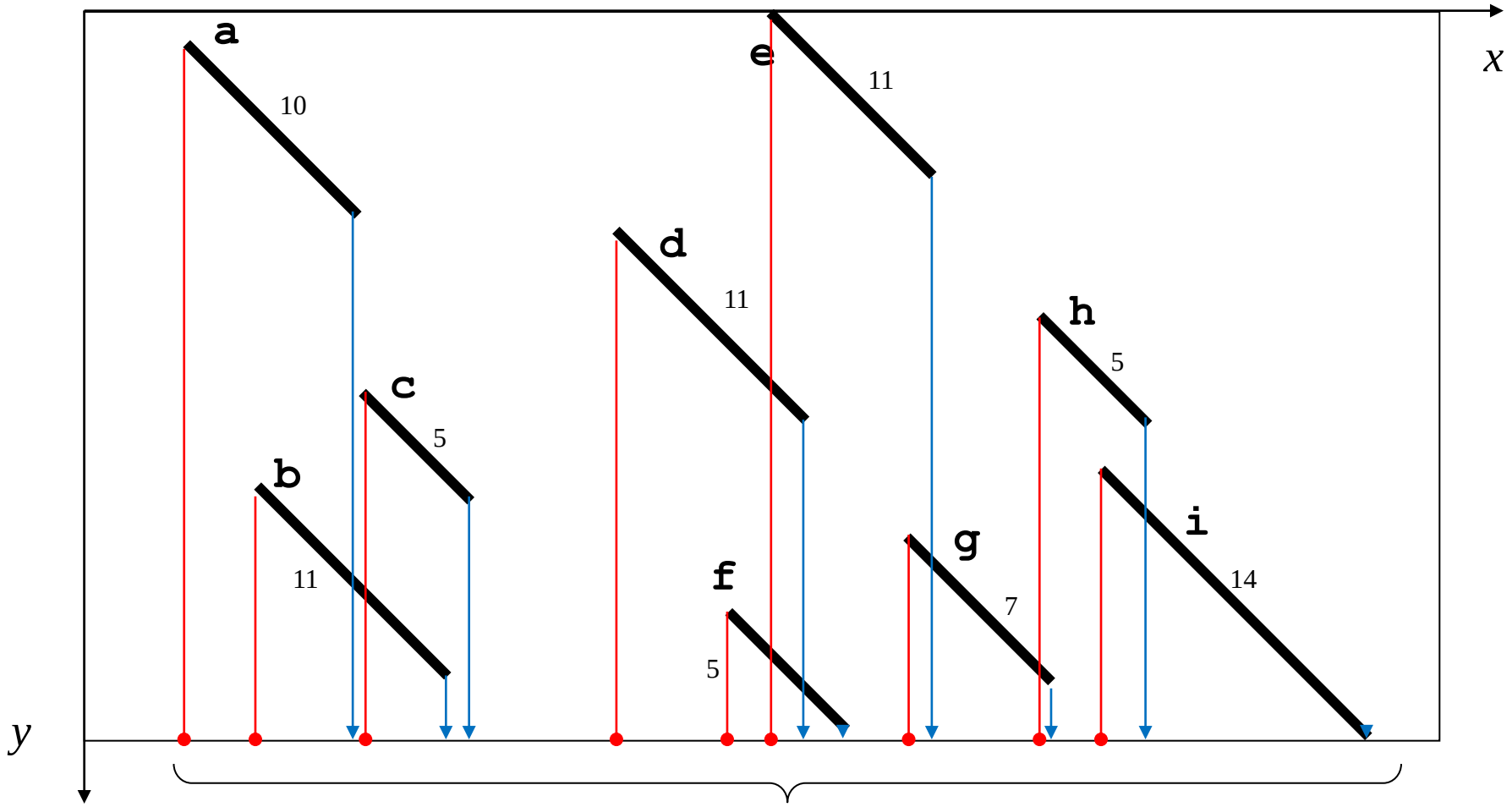
chain	score	
a < d < i	35	optimal chain
a < c < i	29	
e < i	25	
d < i	25	
a < i	24	
c < i	19	



各アラインメントで終了する最大スコアの chain を計算するアルゴリズム

初期化ステップ

- 各アラインメント（太い斜線）の開始点と終了点の x 座標 s_x と e_x をソートしてリスト X に入れる.
- アラインメントのリスト Y（終了点 y 座標で昇順ソート、空集合で初期化）を用意.



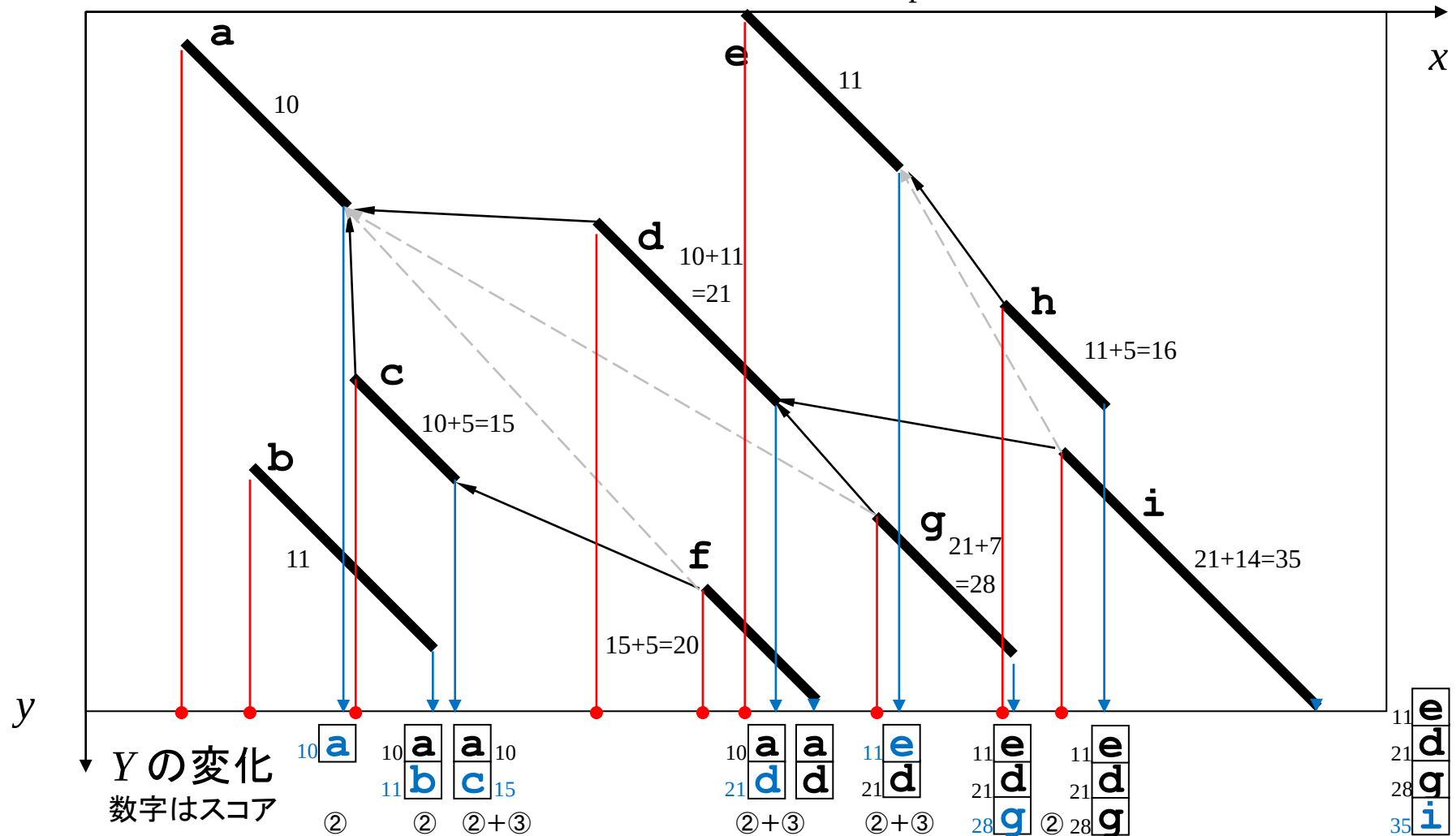
リスト X に登録された x 座標　　赤は開始点　青は終了点

繰返しステップ： X 中の x 座標（アラインメントを A）を順番に処理． A の開始点か終了点かで分類．

① 開始点の場合：A の開始点 y 座標より、終了点 y 座標が小さい（図では上）アラインメントが Y 中に存在するならば、スコア最大（直上に存在）を predecessor として選択し、chain のスコアを計算．

② 終了点の場合：A の終了点 y 座標より、終了点 y 座標が小さく、かつスコアが A 以上のアラインメントが Y に存在するならば、スコアを最大化する predecessor として A は使えないので Y に追加しない．存在しないならば、A を Y へ追加（ただし終了点 y 座標で昇順に並ぶように）．

③ 終了点の場合：A の終了点 y 座標より終了点 y 座標が大きく（図では下）、スコアが A 以下のアラインメント B が Y に存在するならば、B はスコアを最大化する predecessor として使えないので削除．



定理　アラインメントの数を n とすれば、アルゴリズムは最悪計算量 $O(n \log n)$ で各アラインメントで終了する最大スコアの chain を計算する.

証明
値の検索、挿入、削除を各々 $O(\log n)$ で計算できる平衡木（2－3木，赤黒木等）をつかって. アラインメントのリスト Y を実装する.

リスト X 中の i 番目(i=0,1,…) のアラインメント A を処理する際に以下の性質が成り立つことを帰納法で証明.

I. リスト Y のアラインメントは、終了点 y 座標とスコアどちらの値でも昇順にソートされている.
II. 処理済みアラインメントで終了する chain には、最大スコアが付与されている.

繰返しステップの各ケースで、性質 I II を確認

① リスト Y は変更されないので、性質 I は成立. 条件を満たすアラインメントが存在すれば、そのなかでアラインメント A の開始点 y 座標に最も近い終了点をもつアラインメントを $O(\log n)$ で検索でき、性質 I からそのスコアは最大となる. これに A 自身のスコアを加算すれば、性質 II を満たすスコアが得られる.

② スコアが変更されないので、性質 II は成立. 条件を満たすアラインメントが存在しない場合、A の終了点 y 座標より小さく（図で上）、スコアが A 以上の元は存在しない. つまり、終了点 y 座標とスコア双方で昇順になるように（性質 I）、A を $O(\log n)$ で Y へ挿入できる.

③ スコアが変更されないので、性質 II は成立. 条件を満たす元が存在する場合、性質 I が一時的に満たされないが、削除して再び性質 I を保証できる. このような元が複数存在することもあるが、性質 I より連続したブロックとして Y 上に存在する. するとブロックの先頭を検索し、各元を削除する処理は、各々 $O(\log n)$ で実行可能.

Y へ1つアラインメントを挿入／削除する手間は $O(\log n)$ なので、全体で $O(n \log n)$ の計算量.

平衡木(2－3木, 赤黒木 等)について

- ソート列を木構造で表現し、検索／挿入／削除の各操作を $O(\log n)$時間で実行する(n は操作数)、とても実用的なデータ構造
- 2－3木は2年生4学期に五十嵐先生が 講義
- 他にも赤黒木(red-black tree) があり、以下の教科書を薦める
(3階のプログラミング実習室の本棚に全部あります)
  - Robert E. Tarjan「データ構造とネットワークアルゴリズム」1983.
第4章 75-94ページで平衡木を解説. アルゴリズムが好きな人に愛読されていて、他の章も読みやすい.
  - Cormen, Leiserson, Rivest, Stein "Introduction to Algorithms" MIT Press 2001.
第13章　273-301ページで赤黒木を解説. 厚い本だが解説は非常に丁寧.
計算機科学のアルゴリズム関係の教科書はこれと、以下の本で「ほとんど」事足りてしまう.
    - Garey and Johnson "Computers and Intractability: A Guide to the Theory of NP-Completeness"
    - Knuth "The art of programming"　Vol. I, II, III
    - Motwani and Raghavan "Randomized Algorithms"
- 自己調節2分木(self-adjusting binary tree, splay tree)
  - 検索／挿入／削除の各操作は $O(n)$ かかることもあるが、全体で $O(n \log n)$ しかかからないように工夫(chaining にはこれで十分)
  - 2－3木や赤黒木に比べて単純な操作で済む(Tarjan の本で解説)