データ構造とアルゴリズム
第6週

掛下　哲郎
kake@is.saga-u.ac.jp

前回のまとめ

- ヒープ (heap)
  - 最大値を取り出す
  - 挿入・取り出しともに O(log n). 「木」を利用
- 二分探索木（Binary Search Tree）
  - 木が平衡していれば探索，挿入，削除ともにO(log n）
  - 最悪の場合はO(n)
  - どのようにして平衡条件を維持するか？

講義スケジュール

週	講義計画
1－2	導入
3	探索問題
4－5	基本的なデータ構造
6	動的探索問題とデータ構造
7	アルゴリズム演習（第1回）
8－9	データの整列
10－11	グラフアルゴリズム
12	文字列照合のアルゴリズム
13	アルゴリズム演習（第2回）
14	アルゴリズムの設計手法
15	計算困難な問題への対応

これまで

- さまざまなデータ構造
  - 配列
  - 連結リスト
  - ハッシュ表
  - 二分探索木
  - キュー，スタック，ヒープ
- それぞれについて、探索、挿入、削除の手間を考察

今日の内容

動的探索問題とデータ構造

- 探索対象のデータが、動的に変化
- 挿入や削除が頻繁に行われる状況

動的データ構造

- 2分探索木
- 動的ハッシュ表
- 平衡2分探索木(2色木)

場面設定

- DBプログラムを設計・開発する
  - １億エントリのデータを想定
    n = 100,000,000
  - データの追加，削除が
    頻繁に行われる
- 正しく動くのはもちろん、高速性が要求される

まず，データ構造を検討しよう

配列を使えばどうか？

- 配列（順序なし）
  - 探索：O(n)
  - 挿入：O(1)
  - 削除：O(n)
- 配列（順序あり）
  - 探索：O(log n)
  - 挿入：O(n)
  - 削除：O(n)

10億命令/秒 のコンピュータ
O(n) → 0.1秒
O(log n) → 0.000000027秒

O(n)なら，10ジョブ/秒で
システムがパンク

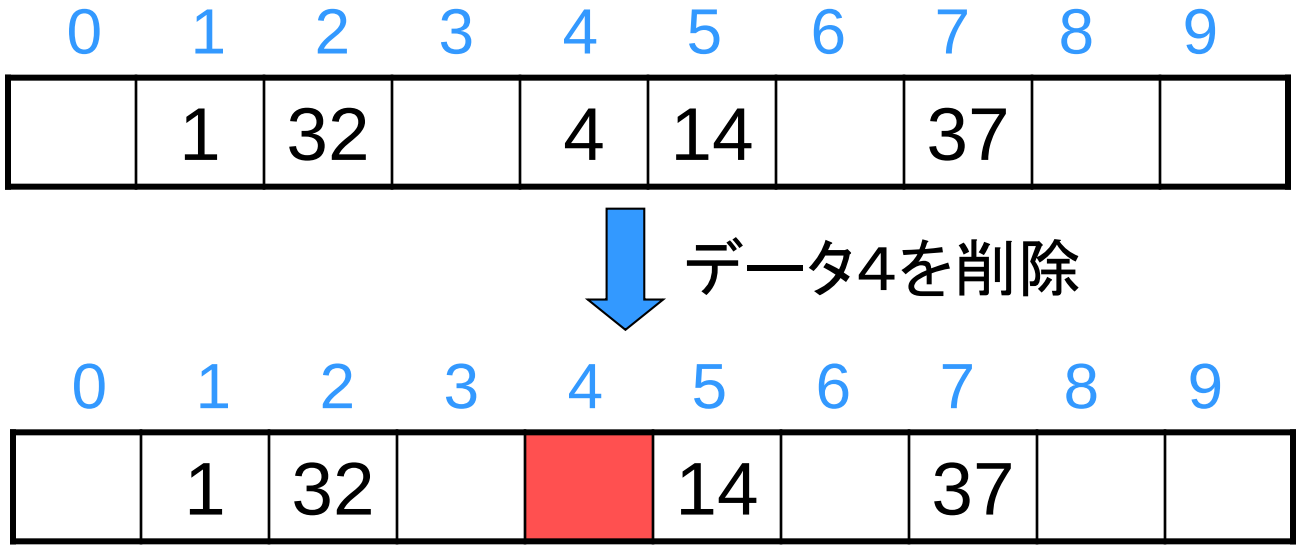
動的ハッシュ法

データの追加・削除が行われる場合のハッシュ法の拡張

- データ x の格納位置を、関数hash(x)で求める
  - m：ハッシュ表Sのサイズ（データ数nの1.5～2倍）
  - x：格納したいデータ
  - i = hash(x)：0～m-1の整数を返す関数
- データ挿入：S[i] にデータxを格納
  - 異なるxで同じiになったとき ⇒ 次の位置から空きを探して格納
- データ探索：hash[x]の位置から順次探索
- データ削除：データを探索して削除
  - 問題：データの削除を行うとき、ハッシュ表から該当データを削除するだけで問題ないか？

「削除跡」の問題

- データ: { 1, 4, 14, 32, 37}
- hash(x) = x % 10

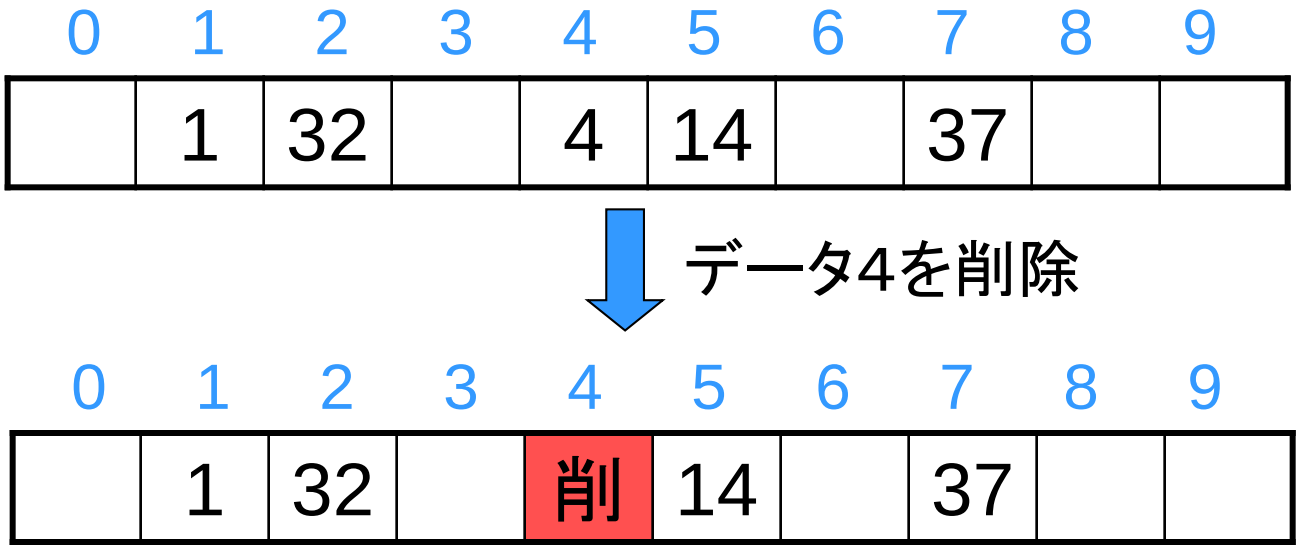


削除後のハッシュ表では、14の探索に失敗！

要書き込み

「削除跡」の問題 ― 対策

- 最初からの「空き」と、削除後の「空き」を区別



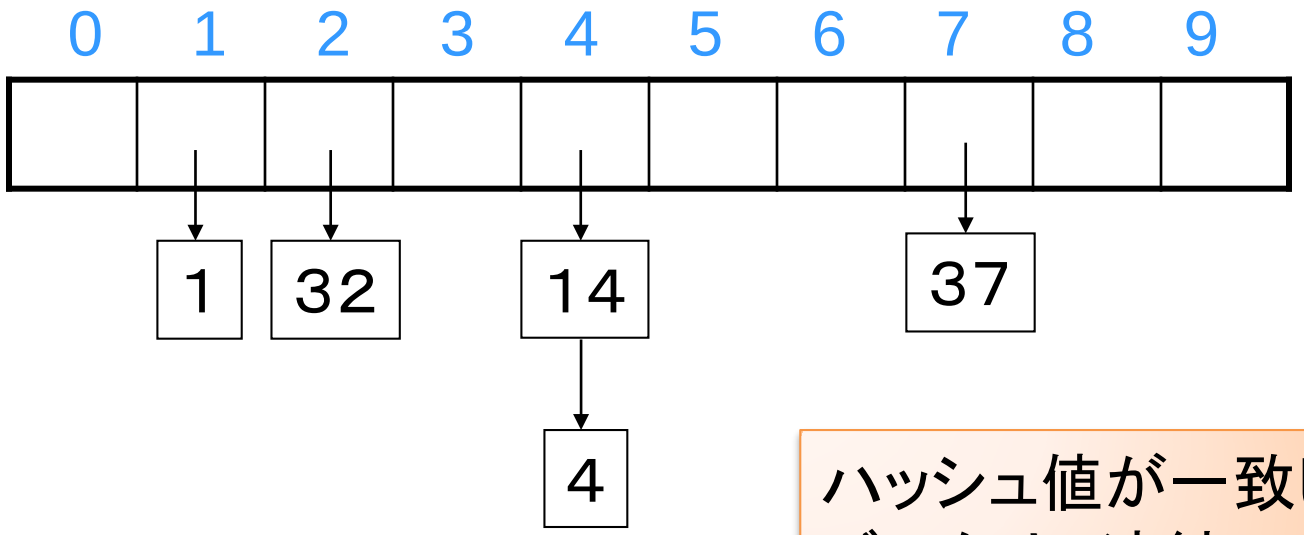
探索時：「削」は「空き」とみなさない
挿入時：「削」は「空き」とみなす

アニメーション教材
講義HP・第6週 ⇒ 削除痕を
考慮した動的ハッシュ法

※「削」が増えると効率が低下
↓ハッシュ表を定期的に再構成

参考:chaining 方式(分離連鎖法)

- データ: { 1, 4, 14, 32, 37}
- hash(x) = x % 10



ハッシュ値が一致したデータは，連結リストを利用して格納

アニメーション教材
講義HP・第6週 ⇒ 分離連鎖法を用いた動的ハッシュ法

要書き込み

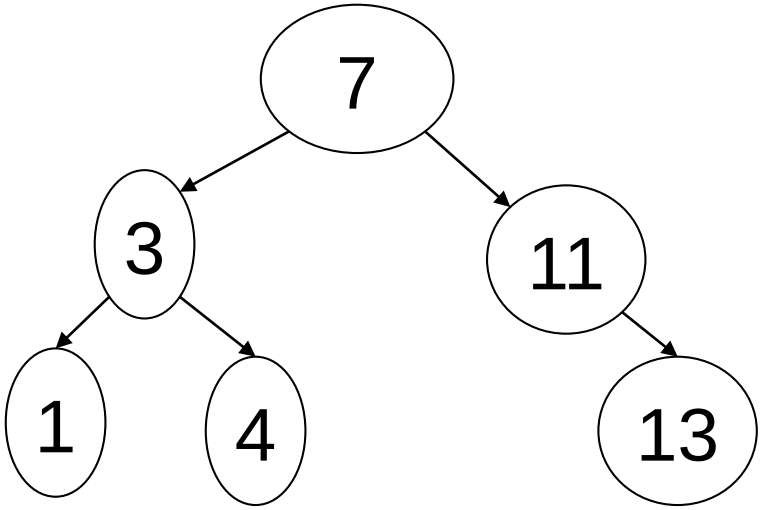
探索、挿入、削除はバランスよく

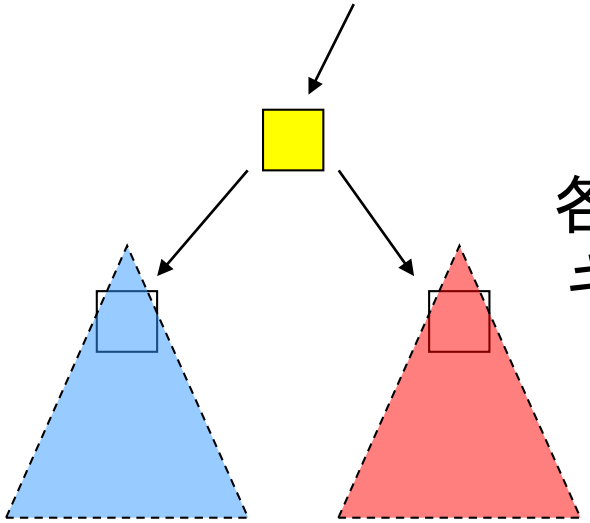
- 3つの操作が、バランスよく効率良くならないか？
- 配列
  - 高速化には「順序」を維持 → ずらす → O(n)の壁
  - データ数が固定
- 一次元リスト
  - ポインタを先頭から辿る必要がある → O(n)の壁
- 二分探索木
  - 平均的にO(log n)で探索，挿入，削除ができる
  - 木構造のバランスが崩れると、最悪O(n)

木構造のバランスが崩れないように工夫

2分探索木

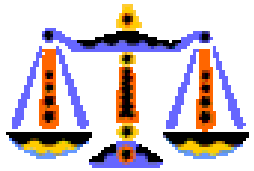
- 2分探索木





各節点における，
キーの大小関係

平衡2分探索木



- バランスを維持する仕組みを備えた二分探索木
  - バランスが崩れそうになると，補正をかける
    - AVL木
    - 2色木
- 基本的には
  - まず，2分探索木と同様に挿入や削除を行う
  - 「バランスの崩れ具合」をチェックし、必要なら簡単な変形操作を行ってバランスを微調整する
    - 1重回転，2重回転

AVL木

以下の平衡条件を満たす2分探索木

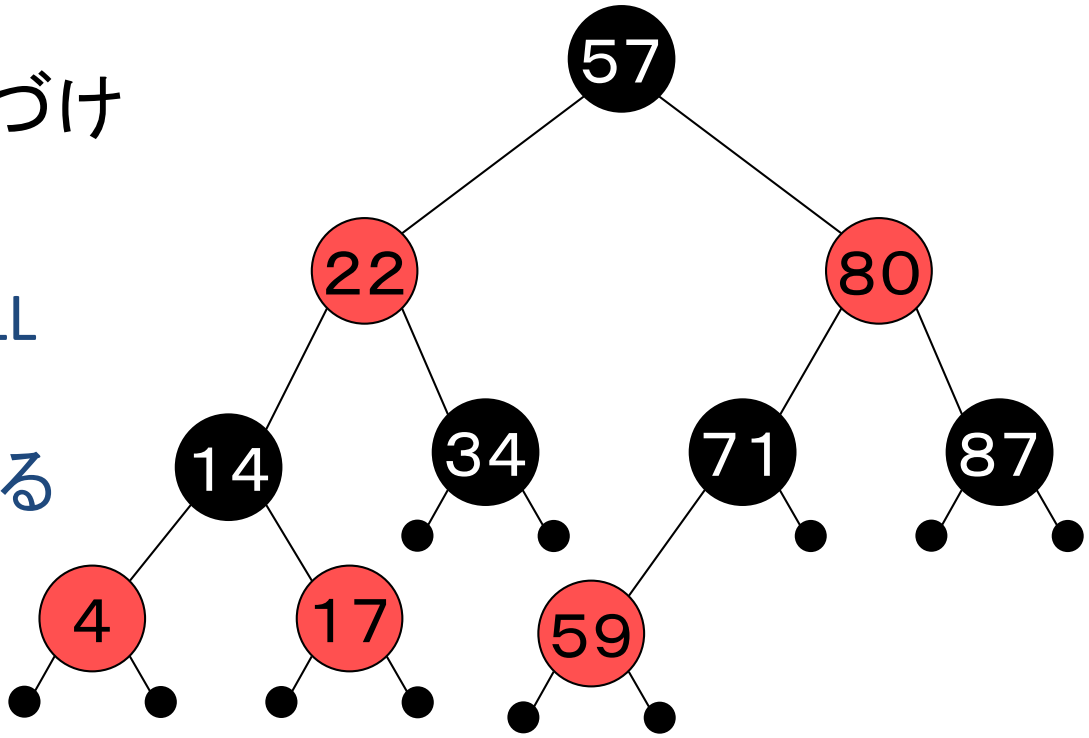
AVL木の平衡条件

1. 2つの子をもつ各節点において、左部分木の高さと右部分木の高さが高々1しか異ならない。
2. 1つの子しか持たない各節点において、その子は葉である。

AVL ・・・発明者のイニシャル
Adel’son-Vel’skii と Landis

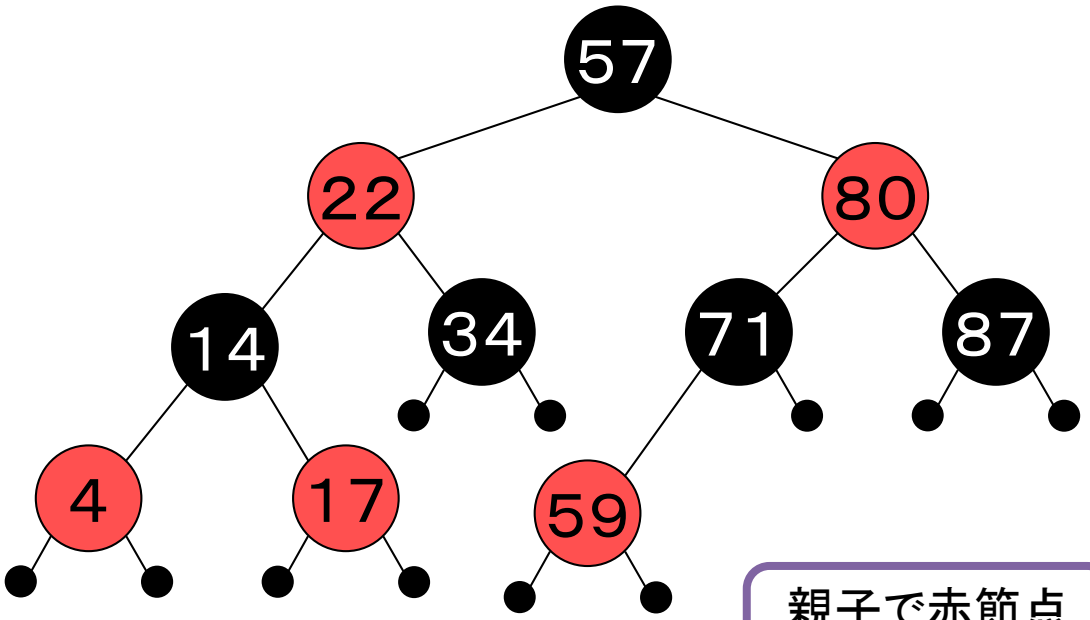
2色木（Red-Black Tree，赤黒木）

- 2色木平衡条件を満たす2分探索木
  - 探索アルゴリズムは，二分探索木と同じ
- 赤・黒の2色で色づけ
- 説明の都合上
  - 子ポインタがNULLのとき，そこには空の葉が存在すると考える。



要書き込み

2色木平衡条件



親子で赤節点が続かない

黒節点だけ見れば平衡木

(RB0) どの内部節点も2つの子を持つ

(RB1) 各節点は、赤または黒のいずれかの色

(RB2) (空の)葉はすべて黒

(RB3) 赤節点の子は、両方とも黒

(RB4) 根から葉までのどの経路も、同じ数の黒の節点を含む

2色木の高さ

- n個のデータを含む2色木の高さはO(log n)
  - 根から葉までの経路に含まれる黒節点の節点数をhとおく。
    - どの経路でも同じ (RB4)
  - 根から葉までの経路に含まれる節点数はh以上2h以下。
    - 赤節点の子は黒 (RB3) → |赤節点| ≦ |黒節点|
  - $2^{h-1} - 1 \leqq n \leqq 2^{2h-1} - 1$

    → 大ざっぱに計算

    $(\log n) / 2 \leqq h \leqq \log n$

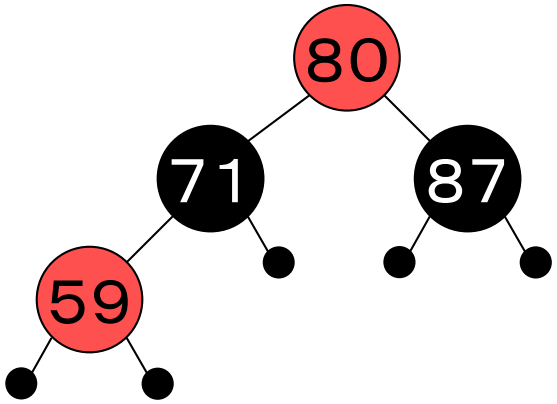
    $(\log n) \leqq 2h \leqq 2 \log n$

2色木へのデータ挿入（基本手順）

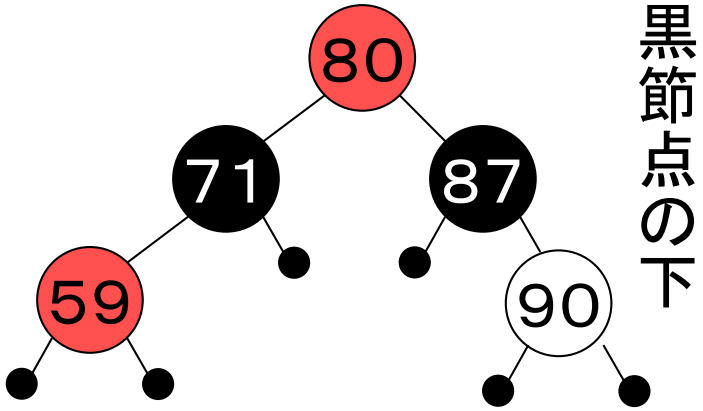
1. とりあえず挿入
   - 通常の2分探索木に対するデータ挿入と同じアルゴリズムを用いてデータを挿入
2. データ挿入後の二色木がRB0～RB4を満たすならば，挿入操作終了
3. 平衡条件を満たさないならば調整
   - 2色木平衡条件を満たすように、木を変形する

ステップ1：とりあえず挿入

通常の2分探索木への挿入方法で、とりあえずデータを入れる。

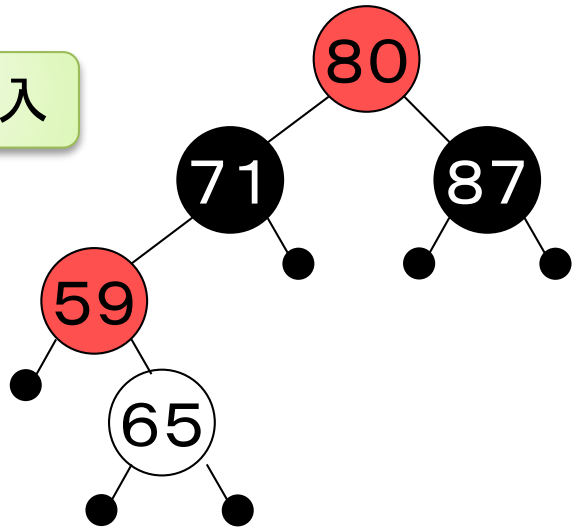


90を挿入



黒節点の下

65を挿入

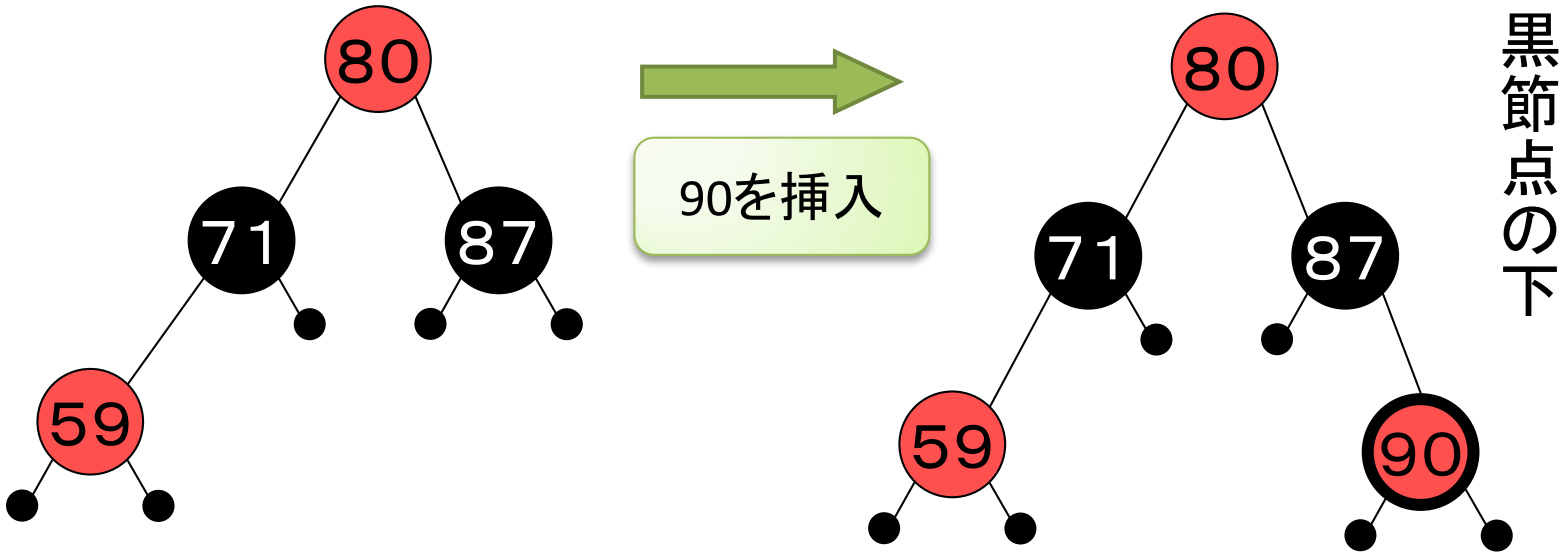


赤節点の下

場合1：黒節点の下に挿入
場合2：赤節点の下に挿入

ステップ2：黒節点の下へ挿入した場合

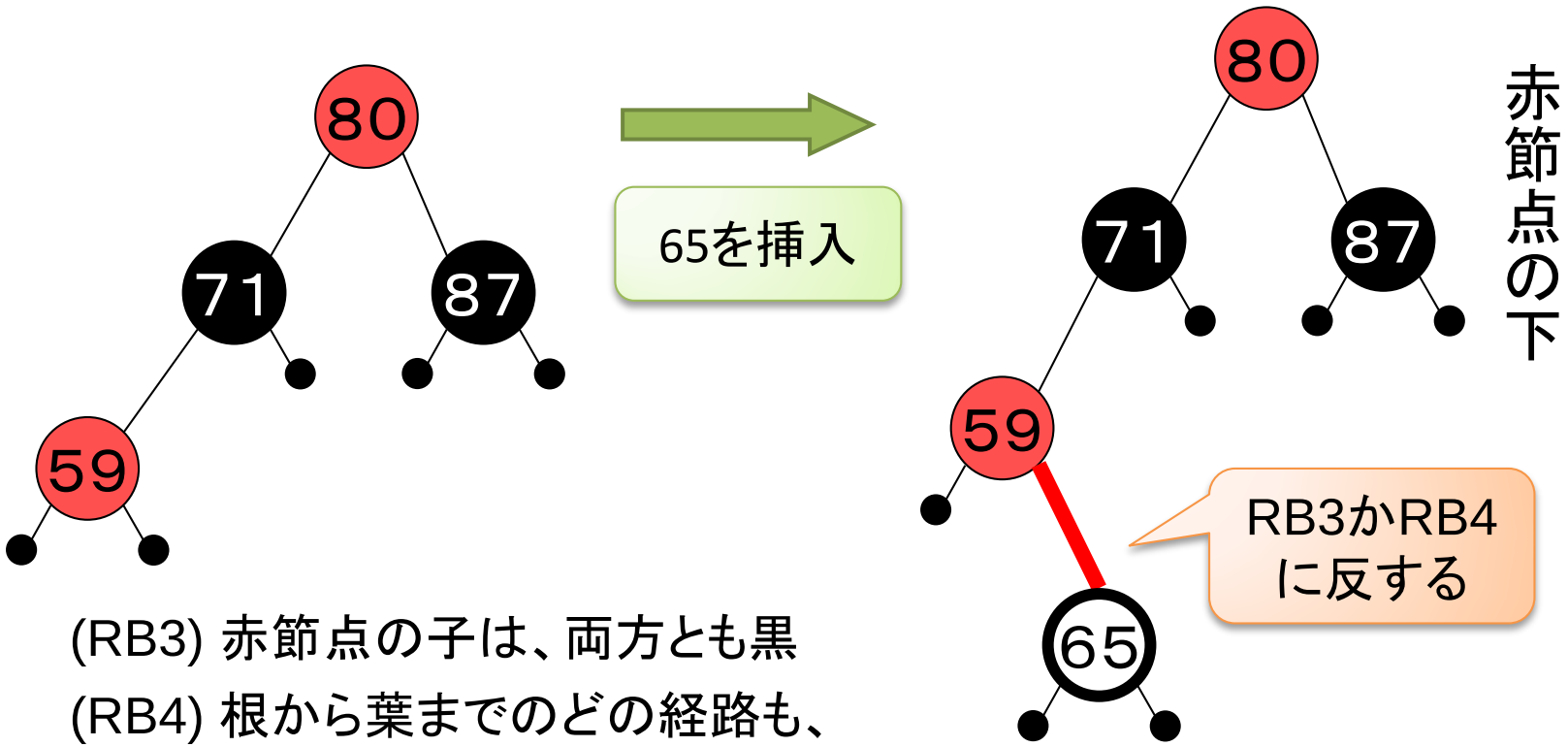
- 黒節点の下へ挿入した場合は、補正の必要なし（平衡条件を満たす）。



挿入した節点は赤とする

ステップ2：赤節点の下へ挿入した場合

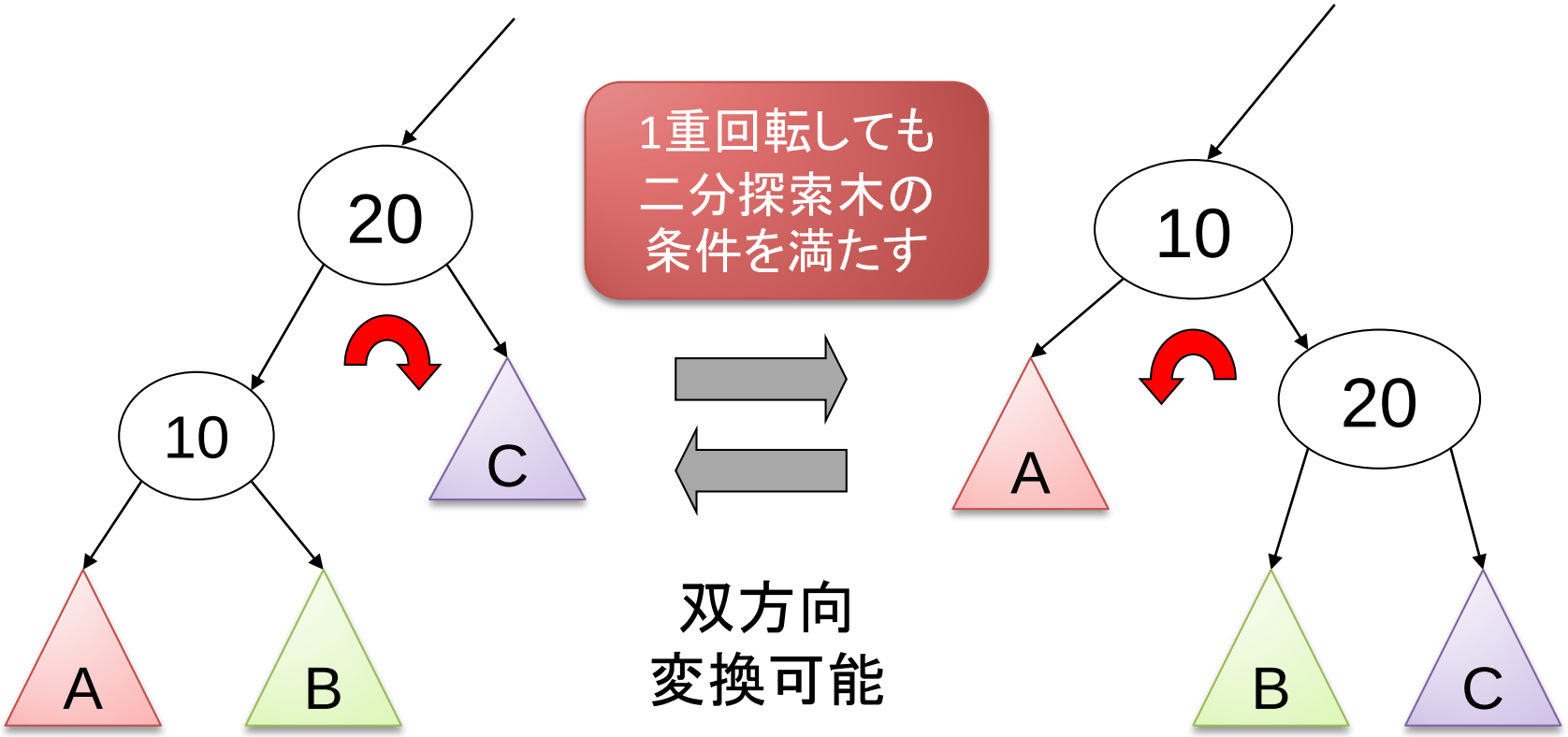
- 赤節点の下へ挿入した場合は、補正の必要あり（平衡条件を満たさない）



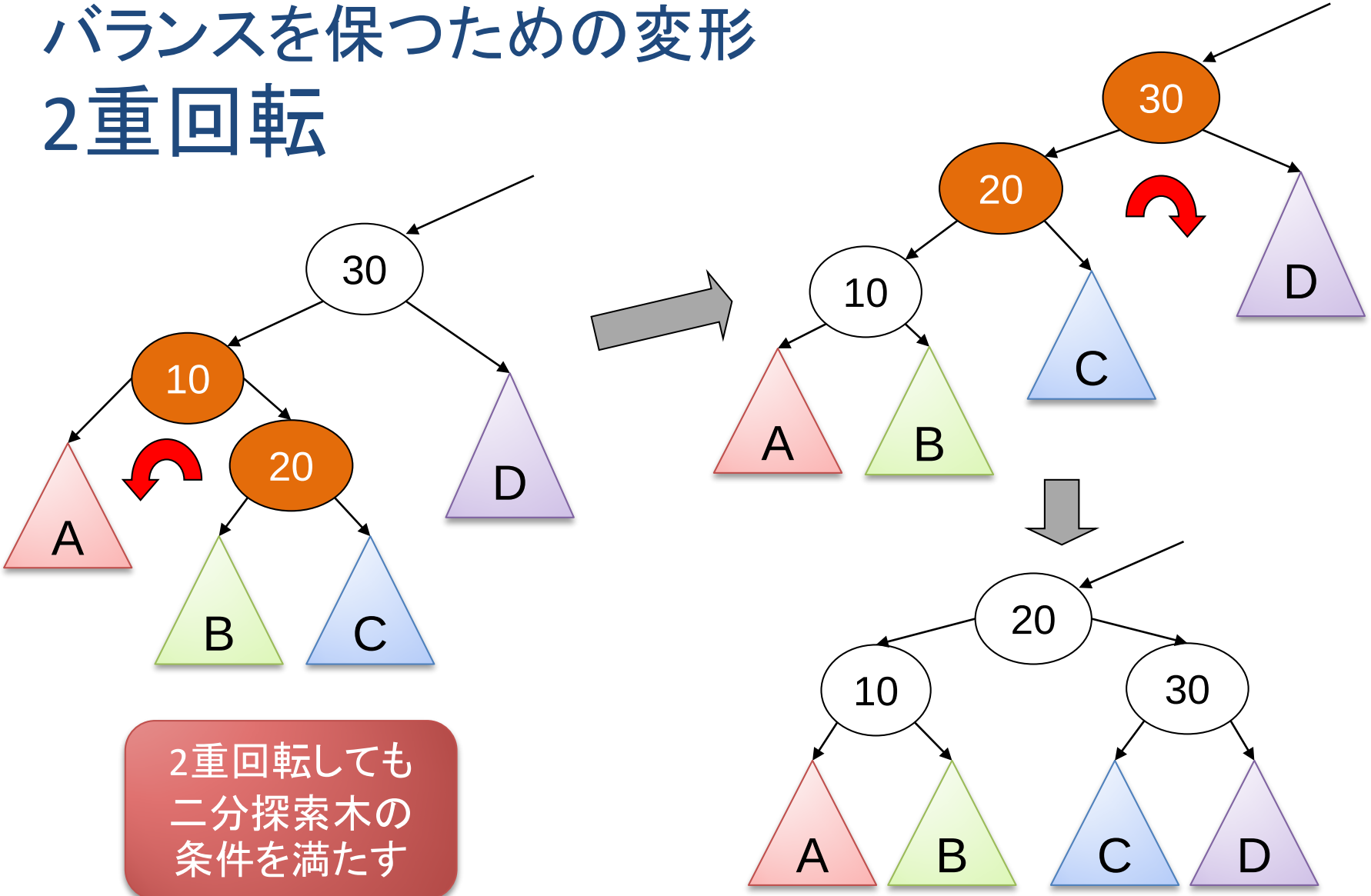
(RB3) 赤節点の子は、両方とも黒

(RB4) 根から葉までのどの経路も、同じ数の黒の節点を含む

バランスを保つための変形
1重回転



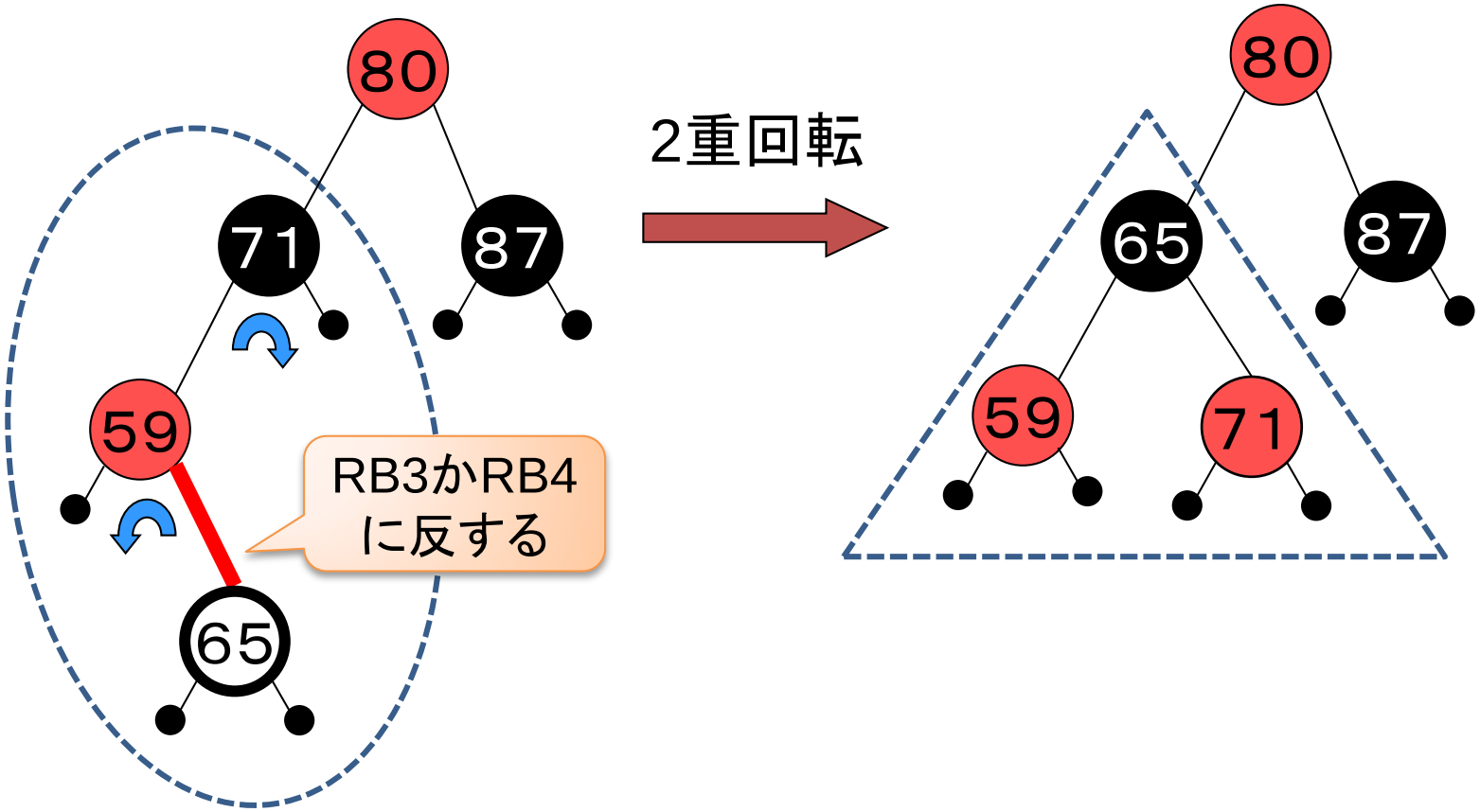
バランスを保つための変形
2重回転



2重回転しても
二分探索木の
条件を満たす

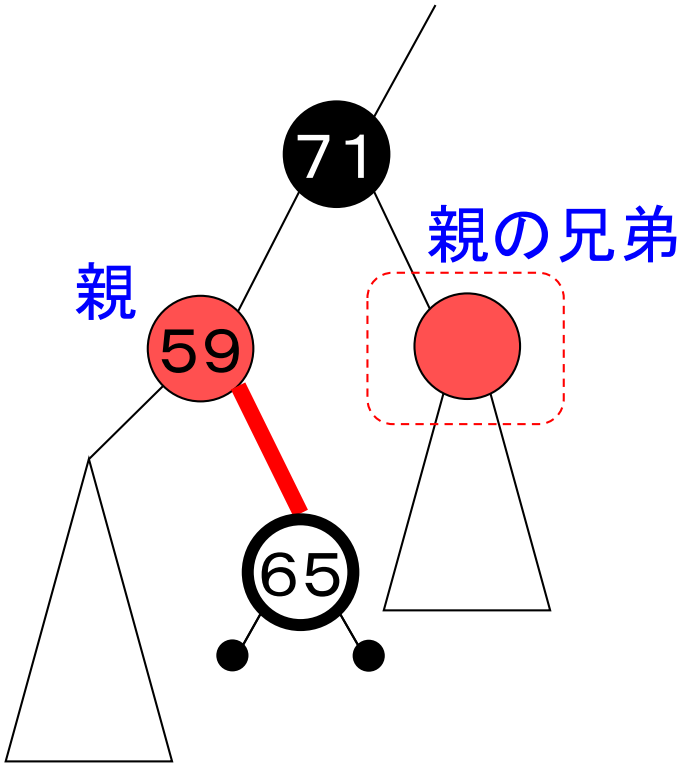
ステップ2：必要なら調整（パターン2）

- 例の場合・・・

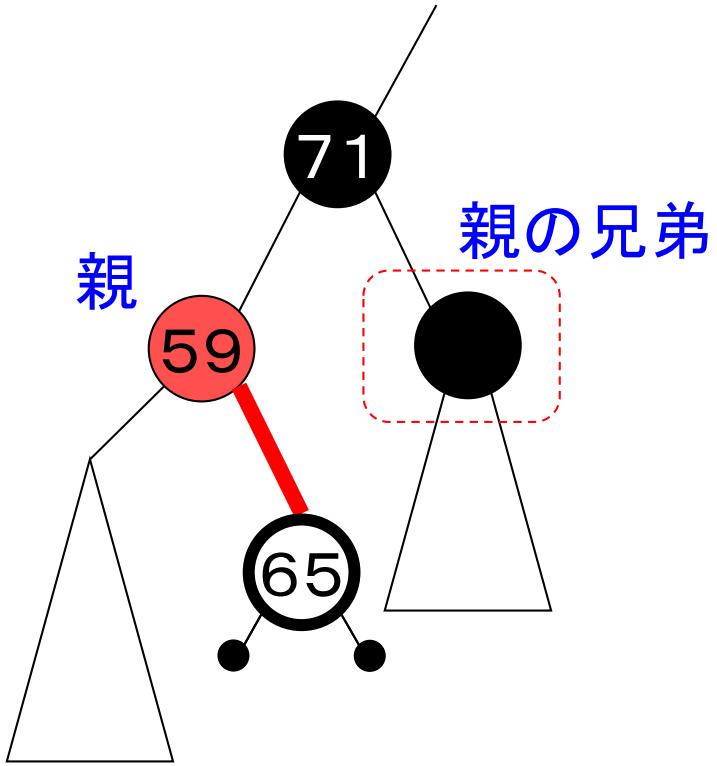


ステップ2：必要なら調整（パターン2）

- 一般の場合（赤節点の下への挿入）

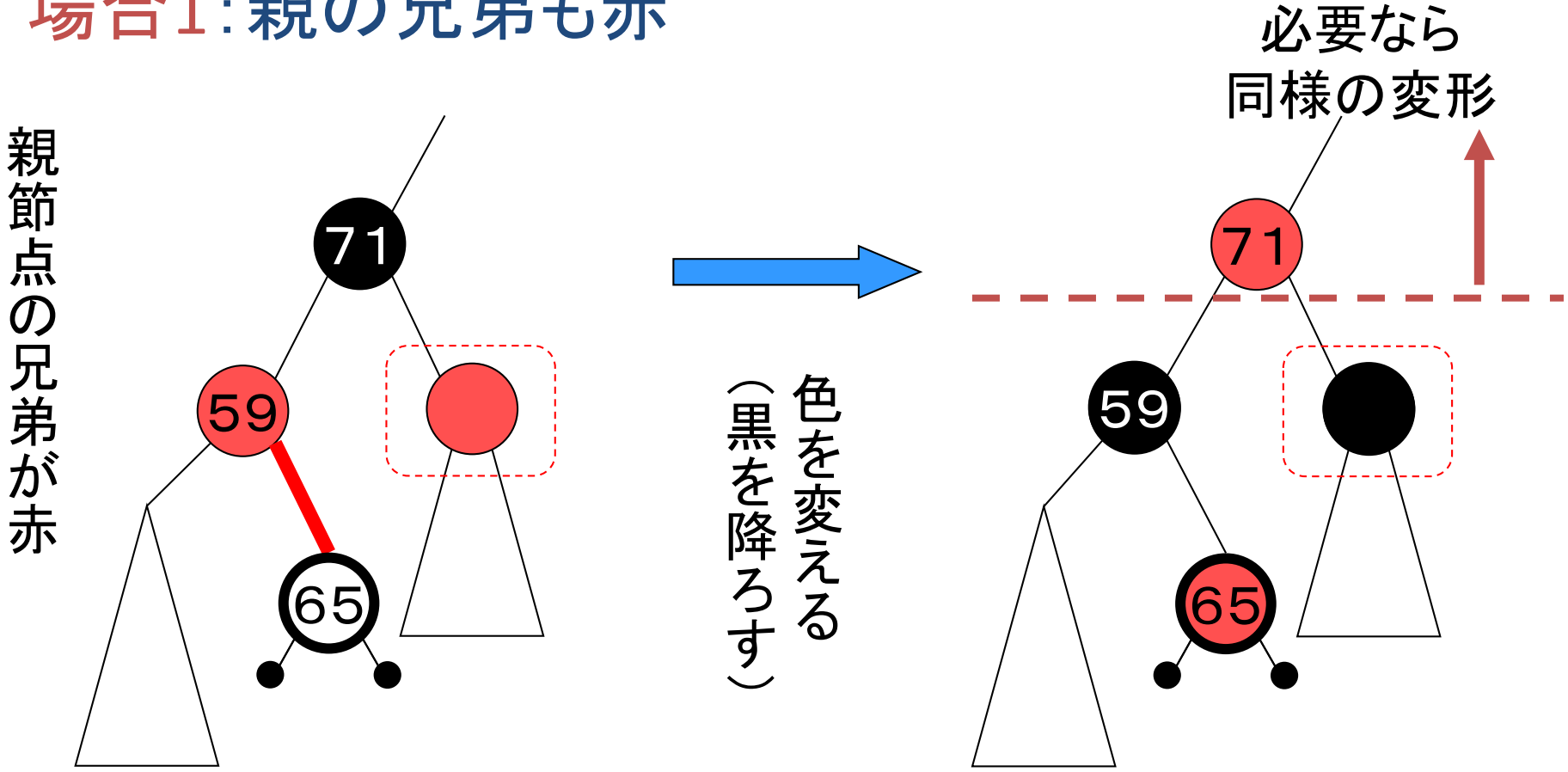


親節点の兄弟が黒か赤かで場合分け



赤節点の下への挿入

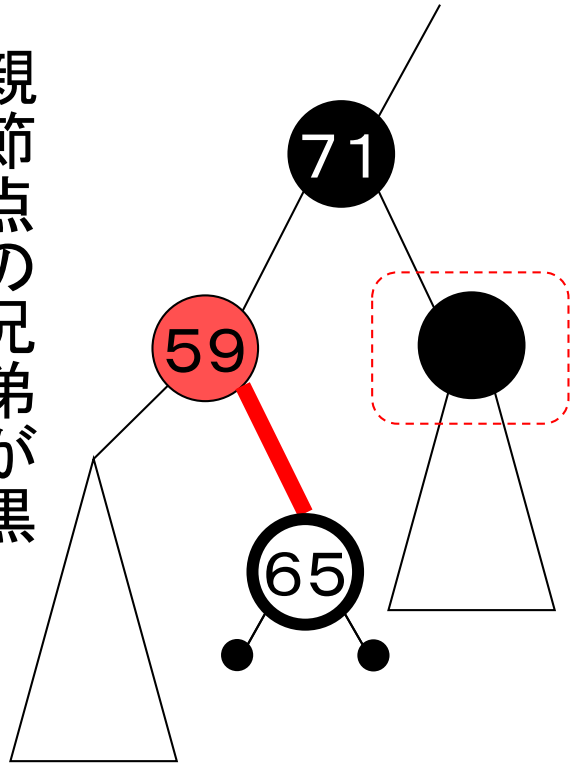
場合1：親の兄弟も赤



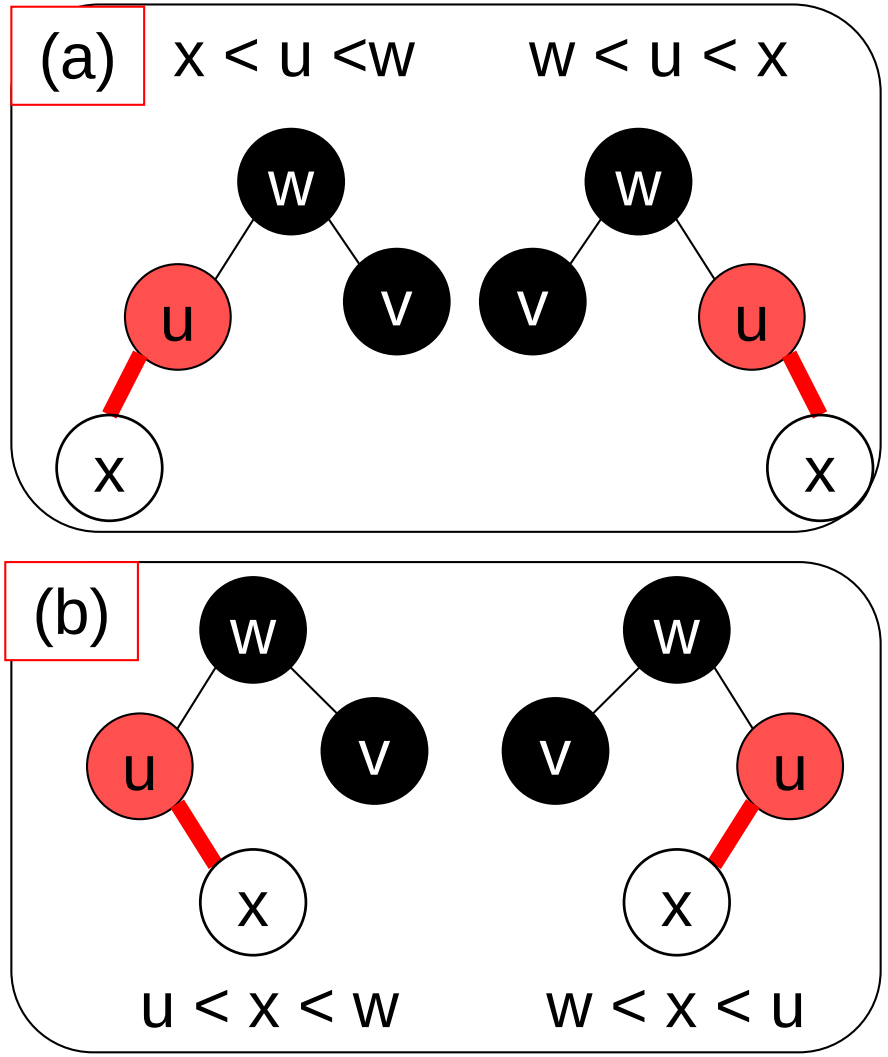
赤節点の下への挿入

場合2：親の兄弟は黒

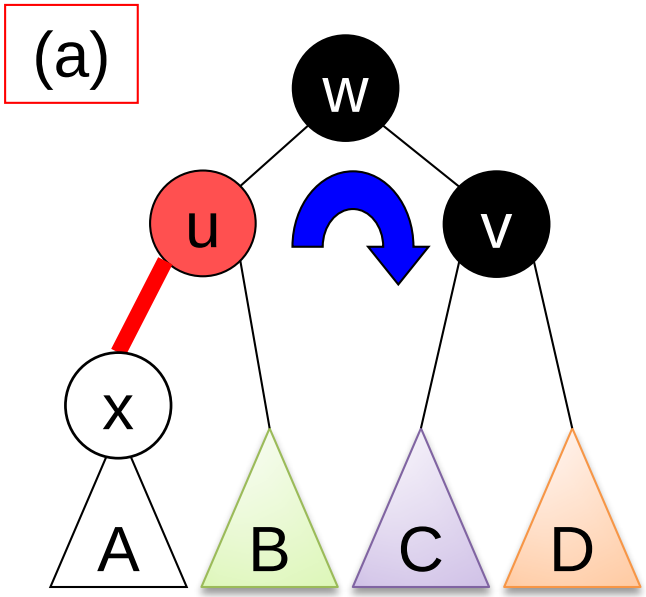
親節点の兄弟が黒



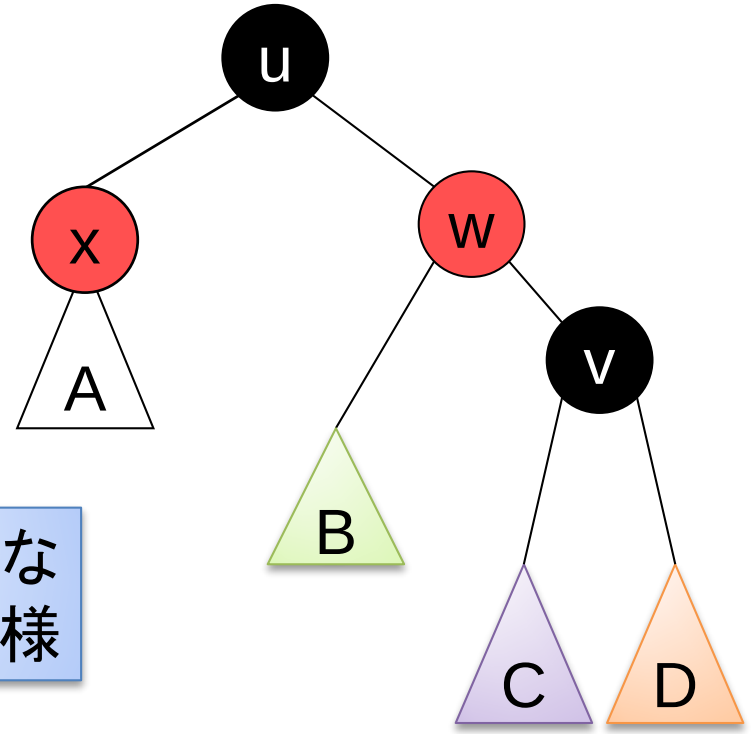
4パターンを2つに場合分け



ステップ2：必要なら調整（パターン2）



1重回転



※左右対称なものも同様

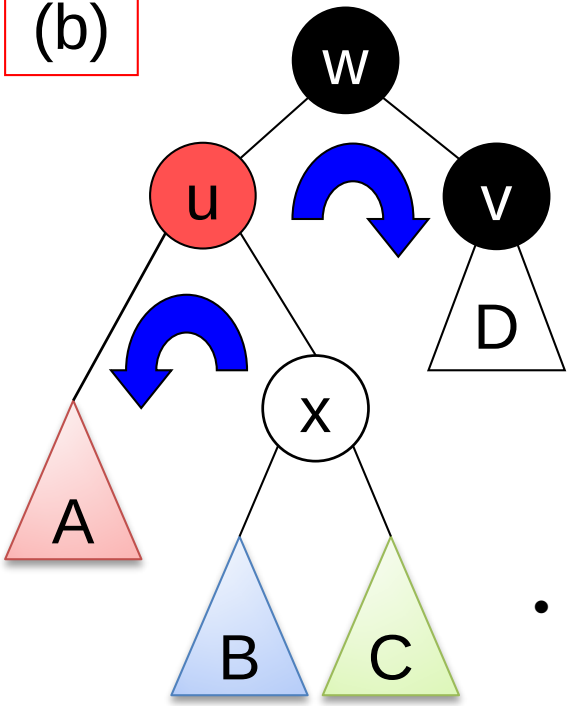
x < u <w

- xを黒にすると部分木Aの高さが1つ増える．⇒ RB4に違反
- xを赤にすると赤節点同士が親子になる．⇒ RB3に違反

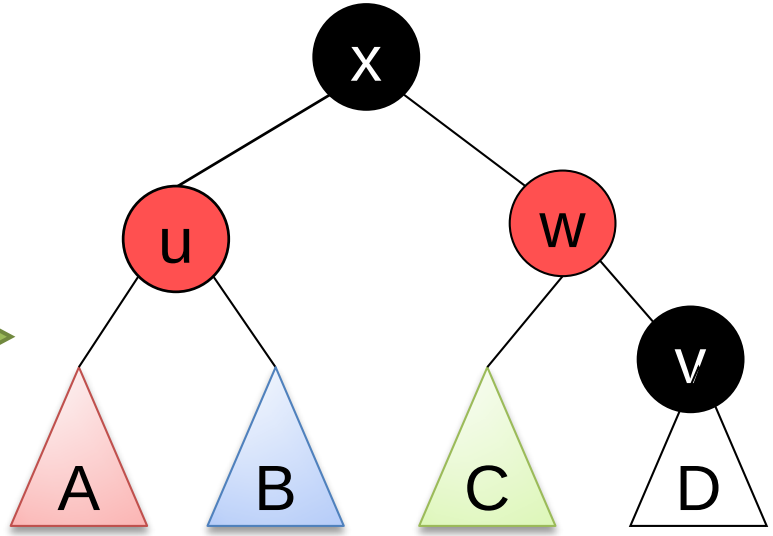
RB3，RB4ともにOK

ステップ2：必要なら調整（パターン2）

(b)



2重回転



u < x < w

- xを黒にすると部分木B, Cの高さが1つ増える．⇒ RB4に違反
- xを赤にすると赤節点同士が親子になる．⇒ RB3に違反

※左右対称なものも同様

要書き込み

2色木への挿入アルゴリズムの場合分け

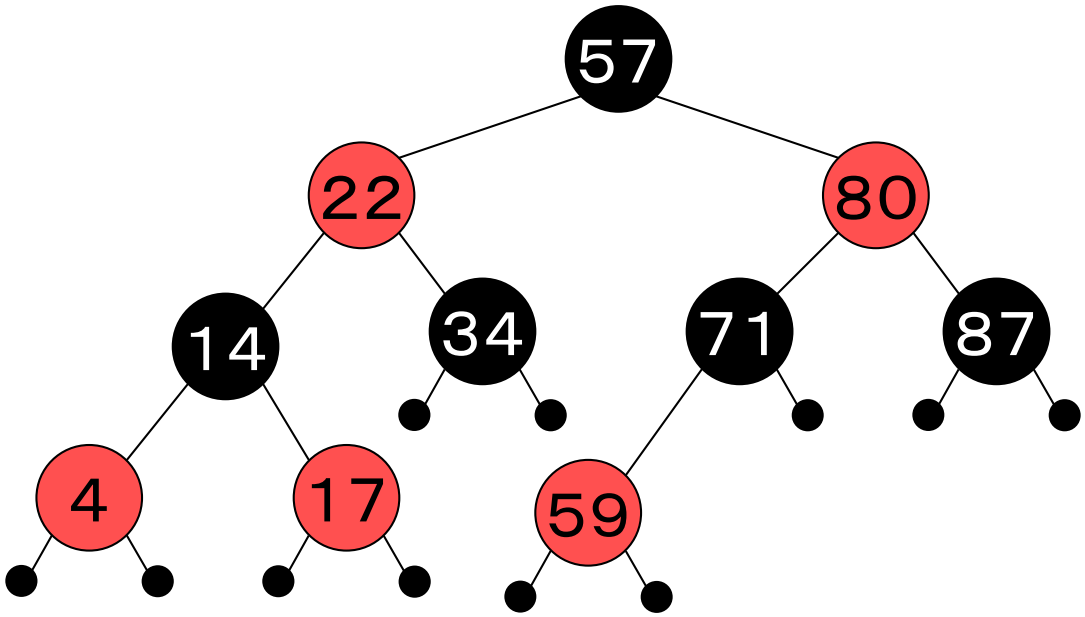
- 黒節点の下に挿入
  - 補正の必要無し. *挿入完了.*
- 赤節点の下に挿入
  - 補正の必要あり.
    - 親節点の兄弟が赤
      - 色の変更. 必要ならさらに遡る
    - 親節点の兄弟が黒
      - (a)パターン → 1重回転
      - (b)パターン → 2重回転

2色木からのデータ削除：基本方針

1. とりあえず削除
   - 通常の2分探索木に対するデータ削除と同じアルゴリズムを用いてデータを削除
2. データ削除後の二色木がRB0～RB4を満たすならば，削除操作終了
3. 平衡条件を満たさないならば調整
   - 2色木平衡条件を満たすように、木を変形する

2色木からのデータ削除
3つの場合

演習問題
場合1〜3の例を埋めよ



場合1：節点xの2つの子が共に空の葉

例： 4, 17, 59, 34, 87

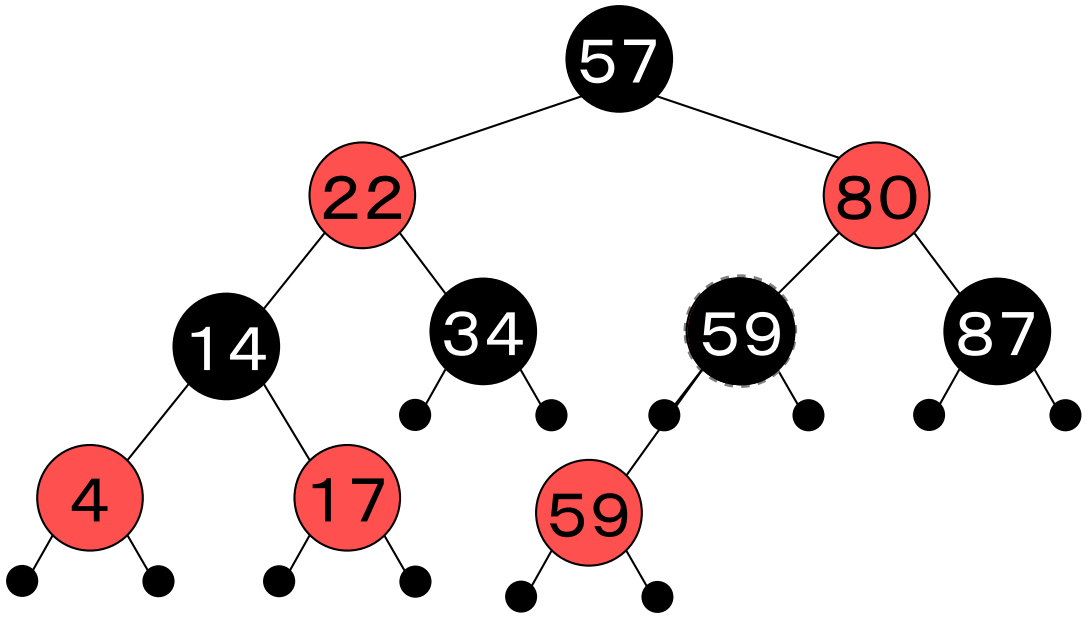
場合2：節点xの一方の子のみが空の葉

例： 71

場合3：節点xの2つの子が共に空の葉でない

例： 57, 22, 80, 14

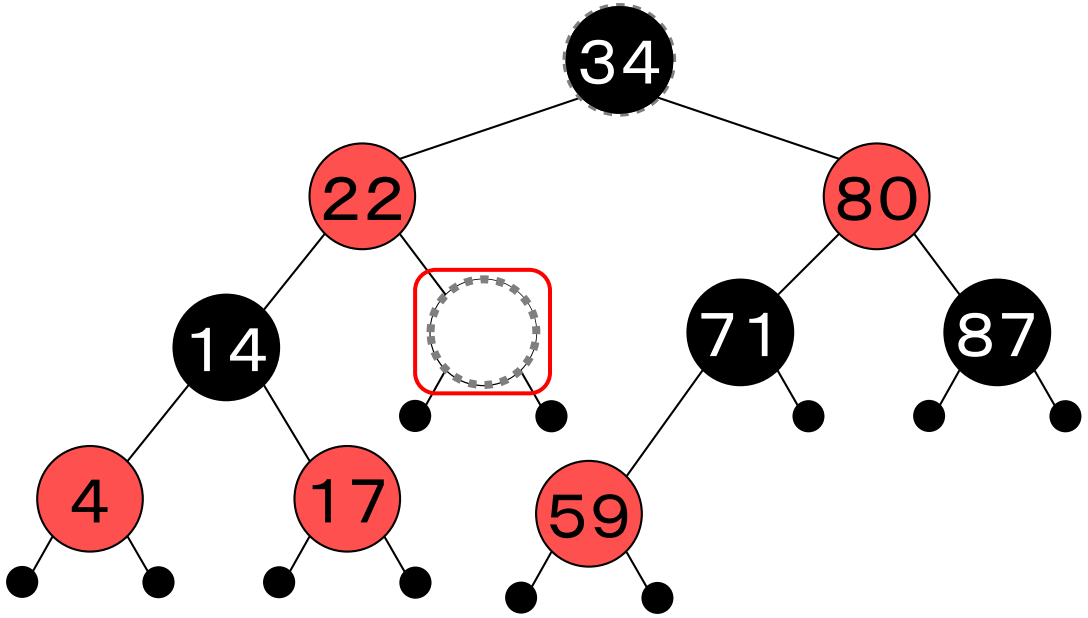
2色木からのデータ削除：場合2
節点xの一方の子のみが空の葉



例：71を削除.

1. 71を削除する.
2. 子で71を置き換える.
   子節点の色は赤
3. 節点の色を黒に変更.

2色木からのデータ削除：場合3
節点xの2つの子が共に空の葉でない



例：57を削除.

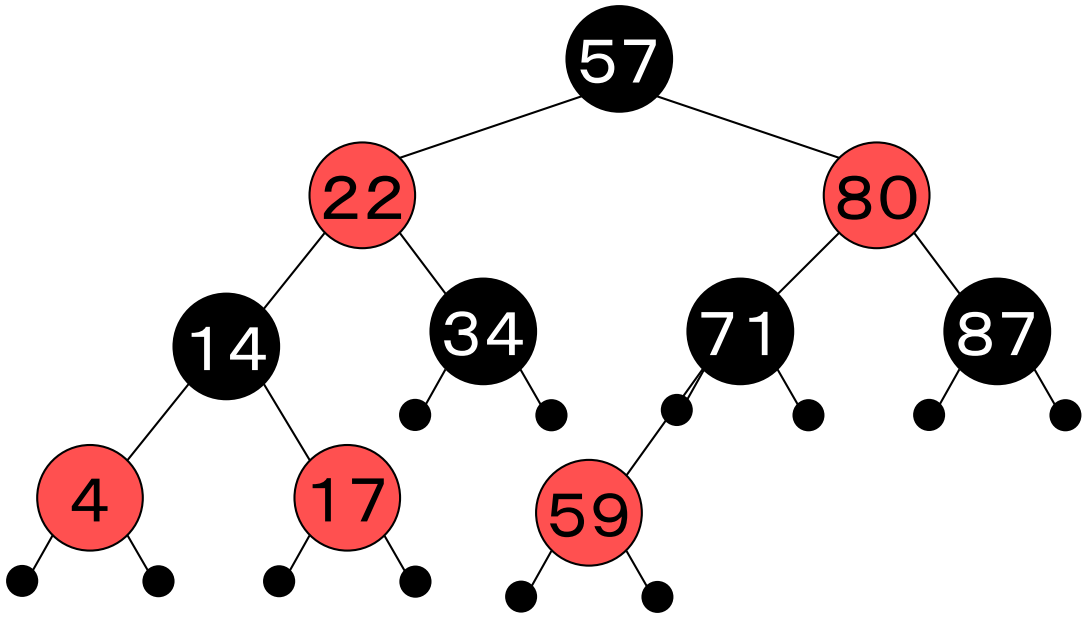
1. 57を削除する.

2. 57の直前キーを求める.

3. 直前キーを削除したい節点に移動.

4. 直前キーを削除
   直前キーの右の子は空
   ⇒ 場合1か場合2を適用

2色木からのデータ削除：場合1
節点xの2つの子が共に空の葉



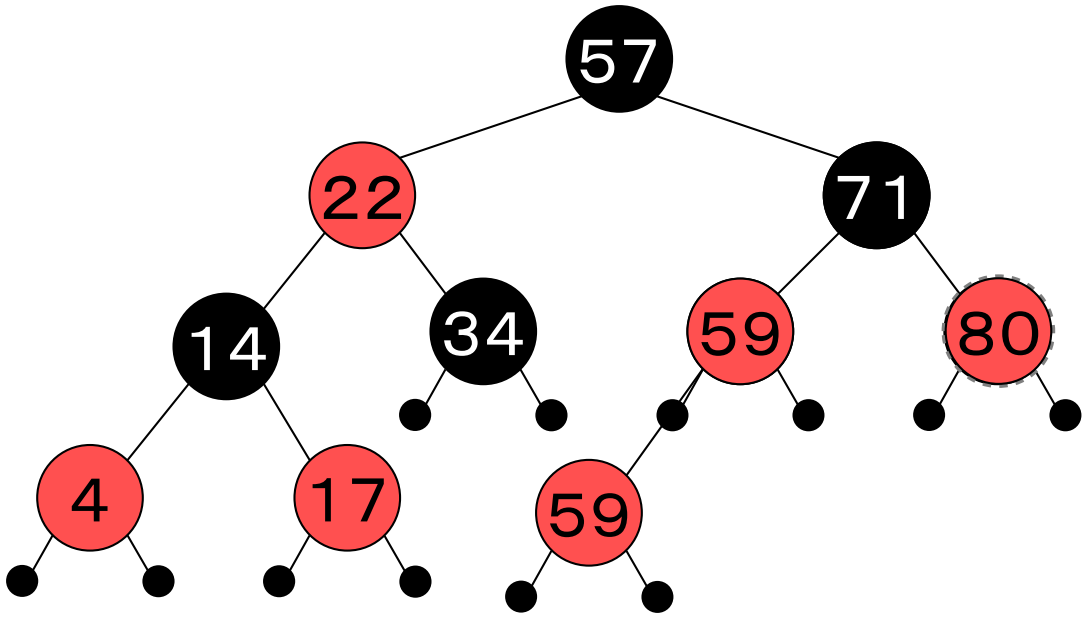
場合1-1
- 節点xが赤節点

場合1-2
- 節点xが黒節点

例：59を削除.

場合1-1は簡単
⇒ 調整は不要

2色木からのデータ削除：場合1
節点xの2つの子が共に空の葉



場合1-2

- 節点xが黒節点

例：87を削除.

1. 87を削除

2. 1重回転する.

要書き込み

本日のまとめ

- 動的探索問題とデータ構造
  - 探索対象のデータが、動的に変化
    - 挿入や削除が頻繁に行われる状況
- 動的ハッシュ
- 2分探索木
  - 探索，挿入，削除　⇒ 平均O(log n)，最悪O(n)
  - データ挿入，削除の順序によっては効率低下
- 平衡2分探索木（2色木）
  - とりあえず挿入・削除
  - 必要ならば平衡条件を保持するように調整
  - 探索，挿入，削除が平均・最悪ともO(log n)

ただし，アルゴリズムは複雑

確認テスト（第６回）

1. 削除痕を考慮した動的ハッシュ方式のアルゴリズム（データの探索，登録，削除）
2. 分離連鎖法を用いた動的ハッシュ方式のアルゴリズム（データの探索，登録，削除）
3. 二色木に対するデータ挿入とデータ削除

次回はPC演習

ノートPCを忘れないように