データ構造とアルゴリズム 第4,5回

1. アルゴリズムの重要性
2. 探索問題
3. 基本的なデータ構造
4. 動的探索問題とデータ構造
5. データの整列
6. グラフのアルゴリズム
7. 文字列のアルゴリズム
8. アルゴリズム設計手法

第４章　動的探索問題とデータ構造

4.1 線形リスト上での探索

4.2 ２分探索木

4.3 平衡２分探索木

4.4 動的ハッシュ法

「第２章 探索問題」との違い

新たなデータの挿入，
既にあるデータの削除
を考える

この章の学習目標

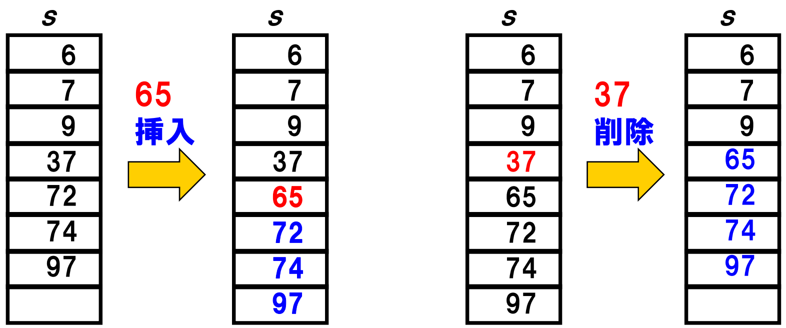
- 探索問題での挿入・削除とは何か，応用例を用いて説明できる
- 2分探索木，平衡2分探索木とは何か説明できる
- 挿入・削除のアルゴリズムを説明できる
  - 逐次探索，２分探索，2分探索木，2色木，ハッシュ法
- 挿入・削除アルゴリズムの計算時間を説明できる
  - 最悪時だけでなく，平均計算時間も
- 挿入・削除を含めた探索アルゴリズムのプログラムを書ける

4.1 線形リスト上での探索

- **２分探索**（配列で実現している場合）
  - データを小さい順に格納
    - 探索に必要な時間 $O(\log n)$
    - 挿入，削除に必要な時間 $O(n)$
- **逐次探索**
  - データの格納順は任意
    - 探索に必要な時間 $O(n)$
    - 挿入，削除に必要な時間 $O(1)$
      - 挿入：最後のデータとして挿入
      - 削除：最後のデータを削除位置に移動

２分探索：挿入，削除

- **２分探索**（配列で実現している場合）
  - データを小さい順に格納
    - **挿入，削除**に必要な時間 $O(n)$

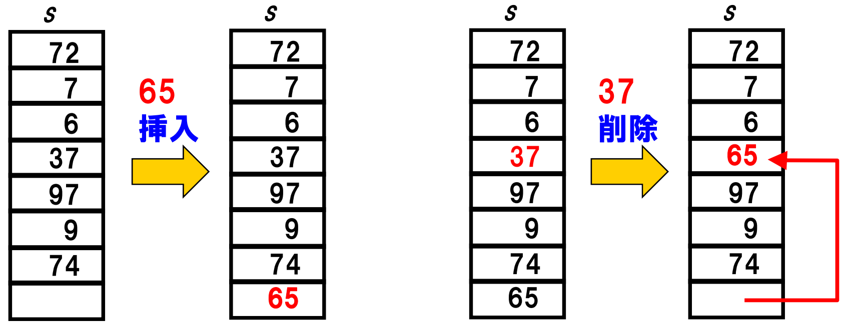


72, 74, 97 を移動

65, 72, 74, 97 を移動

逐次探索：挿入，削除

- **逐次探索**
  - データの格納順は任意
    - **挿入，削除**に必要な時間 $O(1)$



最後に挿入

削除して
最後の要素を移動

第４章　動的探索問題とデータ構造

4.1 線形リスト上での探索

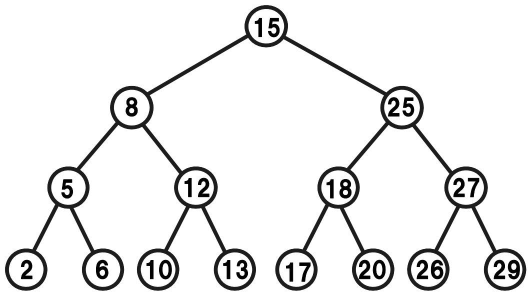
4.2 ２分探索木

4.3 平衡２分探索木

4.4 動的ハッシュ法

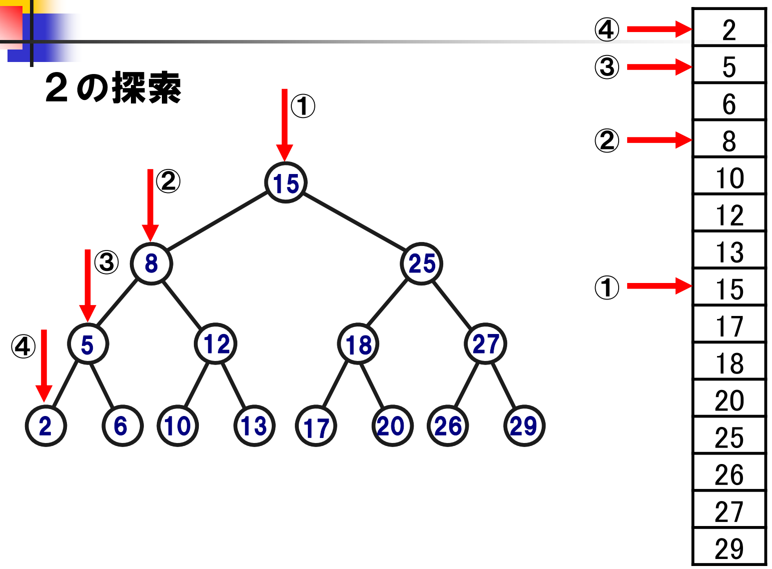
4.2 ２分探索木

- 3.3節 **２分探索法**に対応するデータ構造
  - 図3.5（p.42）の木型データ構造（**２分探索木**）
  - データの挿入，削除は考えていない
    - **完全２分木状**



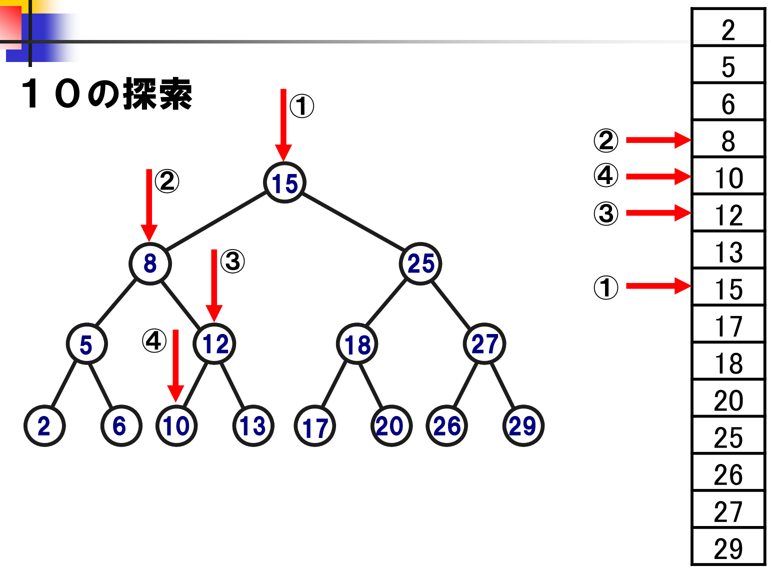
２分探索木と２分探索法の対応

２の探索



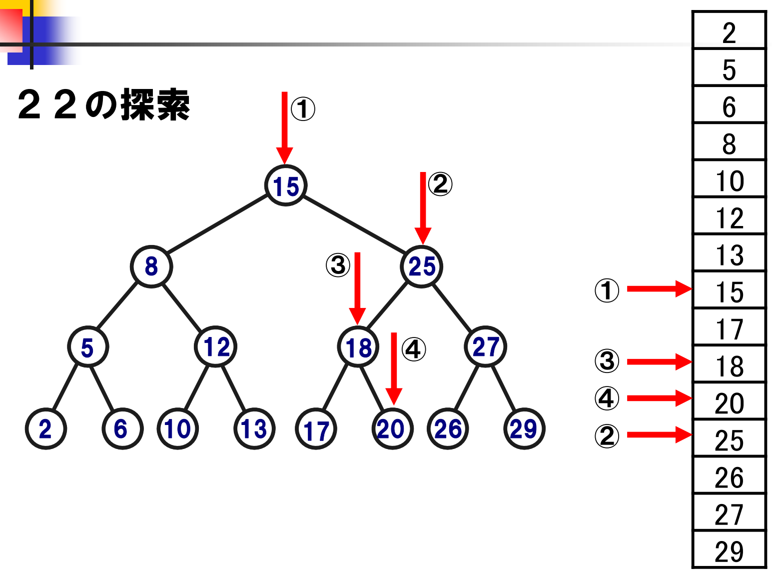
２分探索木と２分探索法の対応

１０の探索



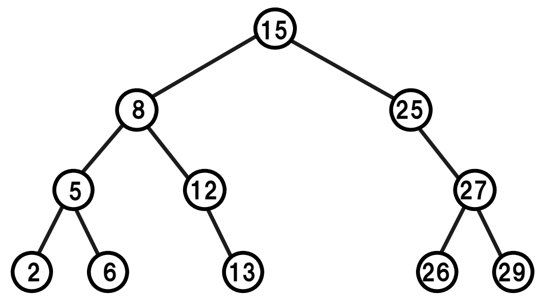
２分探索木と２分探索法の対応

２２の探索



２分探索木条件

- **２分探索木条件**（完全２分木とは限らない）
  - ２分探索木の各節点 v に対し，
    - v の**左部分木**のデータ（キー）は v **以下**
    - v の**右部分木**のデータ（キー）は v **以上**



２分探索木の探索

- **２分探索木条件**
  - ２分探索木の各節点 v に対し，
    - v の左部分木のデータ（キー）は v 以下
    - v の右部分木のデータ（キー）は v 以上
- 探索手続き　**探索時間**：根からデータまでの距離に比例

```
x を入力する;
v = root（根）とする．
while（v が NULL でない）{
    if（x == 節点vのキー値(v->data)）v を出力して終了;
    if（x < 節点vのキー値）v = v の左の子とする;
    else v = v の右の子とする;
}
見つからなかったと報告して終了;
```

２分探索木へのデータの挿入

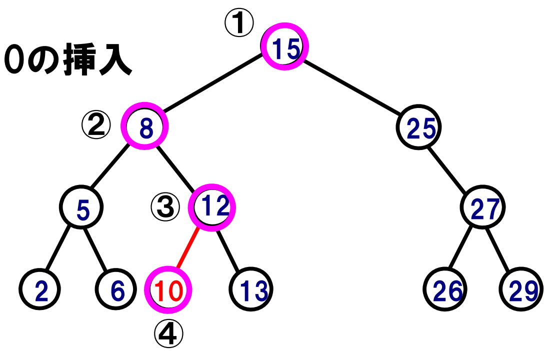
- **挿入手続き**（アルゴリズム4.1）
  - キー x のレコードの挿入
  - **挿入時間**：根から挿入位置までの距離に比例

```
v = root;
while（v がNULLでない）{
    if（x < vのキー値）v = vの左の子;
    else v = vの右の子;
}
新しいレコードを作り，そこへのポインタをvとする;
vのキー値 = x とし，vの左右の子をNULLにする．
```


２分探索木へのデータの挿入

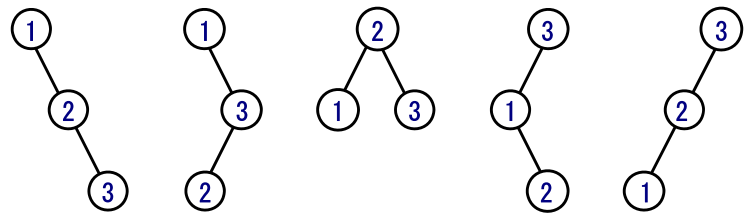
- **挿入手続き**（アルゴリズム4.1）
  - キー x のレコードの挿入
  - **挿入時間**：根から挿入位置までの距離に比例

10の挿入



4.2.2 ２分探索木のバランス

- レコードの挿入順により，異なる２分探索木を構成



(a) 1,2,3　(b) 1,3,2　(c) 2,1,3 2,3,1　(d) 3,1,2　(e) 3,2,1

挿入順序

第４章　動的探索問題とデータ構造

4.1 線形リスト上での探索

4.2 ２分探索木

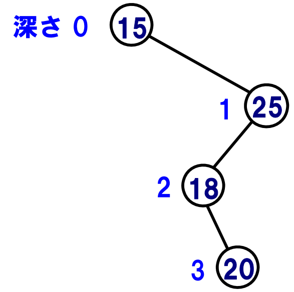
1 ２分探索木条件と新たな要素の追加
2 ２分探索木のバランス
3 最悪の場合の探索時間
4 平均的な探索時間
5 ２分探索木からの要素の削除
6 直前キーの探索
7 最大値と最小値の探索

4.3 平衡２分探索木

4.4 動的ハッシュ法

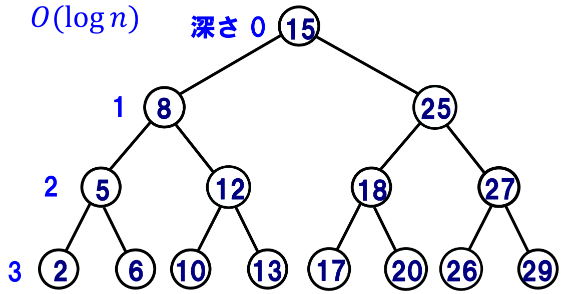
4.2.3 最悪の場合の探索時間

- **探索時間**（n：２分探索木中のデータ数）
  - 比較回数：**探索データの深さ（根からの距離）＋1**
    - **最悪**の２分探索木：
      - 鎖状の２分探索木
      - **最悪時**　$O(n)$



4.2.3 最悪の場合の探索時間

- **探索時間**（n：２分探索木中のデータ数）
  - 比較回数：**探索データの深さ（根からの距離）＋1**
    - **理想的**な２分探索木：
      - データをなるべく根の近くに配置
      - **最悪時**　$O(\log n)$



4.2.4 平均的な探索時間

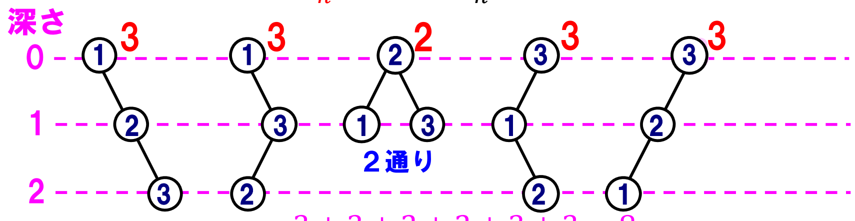
- **探索時間**（n：２分探索木中のデータ数）
  - 比較回数：**探索データの深さ（根からの距離）＋1**
  - **平均**探索時間
    - **各データの探索確率を $1/n$ と仮定**
      - 探索の失敗は考えない
    - **鎖状**の２分探索木　$O(n)$
    - **完全**２分探索木　$O(\log n)$
    - **平均的な**２分探索木では？

平均的な２分探索木での平均探索時間

- **平均的な２分探索木**
  - データの挿入順序が決まれば2分探索木は決まる
  - $n!$ 通りの挿入順序は等確率と仮定
- **平均探索時間評価のための仮定**
  - (1) 全てのキーの探索は等確率
  - (2) 探索の失敗は考慮しない
- **平均探索時間：$O(\log n)$**　完全２分探索木と同じオーダ

平均的な木での平均探索時間

- 探索時間
  - 比較回数：探索データの深さ（根からの距離）＋１
- 平均探索時間：$O(\log n)$
- 平均的な木：$n!$ 通りの挿入順序が等確率
- D_n：n 頂点の２分探索木の全節点の深さの総和の平均
  - 平均探索時間：$\frac{D_n}{n} + 1 = O(\frac{D_n}{n})$

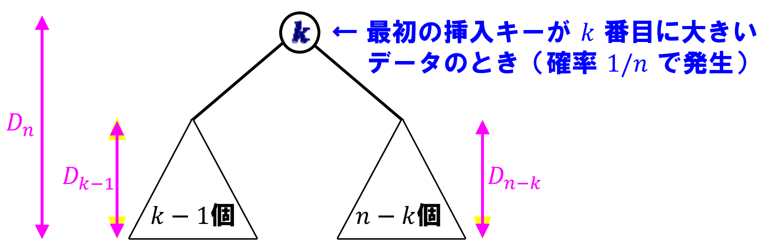


$$D_n = \frac{3+3+2+2+3+3}{3!} = \frac{8}{3}$$

平均的な木での平均探索時間

- 平均探索時間：$\frac{D_n}{n} + 1 = O(\frac{D_n}{n})$
  - D_n：n 頂点の２分探索木の全節点の深さの総和の平均
    - $D_0 = D_1 = 0$
    - $D_n = (\sum_{k=1}^{n}(n-1+D_{k-1}+D_{n-k}))/n$

左右部分木の各節点の深さが１増加　左部分木　右部分木



平均的な木での平均探索時間

- 平均探索時間：$\frac{D_n}{n} + 1 = O(\frac{D_n}{n})$
  - D_n：n 頂点の２分探索木の全節点の深さの総和の平均
    - $D_0 = D_1 = 0$
    - $D_n = (\sum_{k=1}^{n}(n-1+D_{k-1}+D_{n-k}))/n$
  - この漸化式を解くと，$D_n = O(n \log n)$
- **平均探索時間：$O(\log n)$**

漸化式を解いてみよう (1)

- $D_1 = 0$
- $D_n = \left(\sum_{k=1}^{n}(n-1+D_{k-1}+D_{n-k})\right)/n$
- この漸化式を解くと，$D_n = O(n\ \log n)$

$$
\begin{aligned}
D_n &= \left(\sum_{k=1}^{n}(n-1+D_{k-1}+D_{n-k})\right)/n \\
&= \left(\sum_{k=1}^{n}(n-1+2D_{k-1})\right)/n \\
&= \mathrm{n}-1+\frac{2}{n}\cdot\sum_{k=1}^{n}D_{k-1} \\
&= \mathrm{n}-1+\frac{2}{n}\cdot\sum_{k=0}^{n-1}D_k
\end{aligned}
$$

$A_n = D_n - 2$ とおく

$$A_n + 2 = n-1+\frac{2}{n}\cdot\sum_{k=0}^{n-1}(A_k+2)$$

$$A_n = n-1+\frac{2}{n}\cdot\sum_{k=0}^{n-1}A_k+4-2 = n+1+\frac{2}{n}\cdot\sum_{k=0}^{n-1}A_k$$

漸化式を解いてみよう (2)

- $A_n = n+1+\frac{2}{n}\cdot\sum_{k=0}^{n-1}A_k$

$$n\cdot A_n = n(n+1)+2\cdot\sum_{k=0}^{n-1}A_k \qquad (1)$$

$$(n-1)A_{n-1} = (n-1)n+2\cdot\sum_{k=0}^{n-2}A_k \qquad (2)$$

$(1)-(2)$

$$n\cdot A_n-(n-1)A_{n-1} = 2n+2A_{n-1}$$

$$n\cdot A_n = (n+1)A_{n-1}+2n$$

両辺を $n(n+1)$ で割る

$$\frac{A_n}{n+1} = \frac{A_{n-1}}{n}+\frac{2}{n+1} = \frac{A_{n-2}}{n-1}+\frac{2}{n}+\frac{2}{n+1} = \frac{A_1}{2}+\frac{2}{3}+\frac{2}{4}+\cdots+\frac{2}{n}+\frac{2}{n+1}$$

漸化式を解いてみよう (3)

- $\frac{A_n}{n+1} = \frac{A_1}{2}+\frac{2}{3}+\frac{2}{4}+\cdots+\frac{2}{n}+\frac{2}{n+1}$

$A_1 = D_1 - 2$ より，$A_1 = -2$

$$
\begin{aligned}
\frac{A_n}{n+1} &= -1+\frac{2}{1}+\frac{2}{2}+\frac{2}{3}+\frac{2}{4}+\cdots+\frac{2}{n}+\frac{2}{n+1}-\left(\frac{2}{1}+\frac{2}{2}\right) \\
&= 2H_{n+1}-4 \qquad (H_n = O(\log n) \quad \text{調和級数）}) \\
&= O(\log n)
\end{aligned}
$$

$A_n = O(n\log n)$

$A_n = D_n - 2$ より，$D_n = O(n\log n)$

第４章　動的探索問題とデータ構造

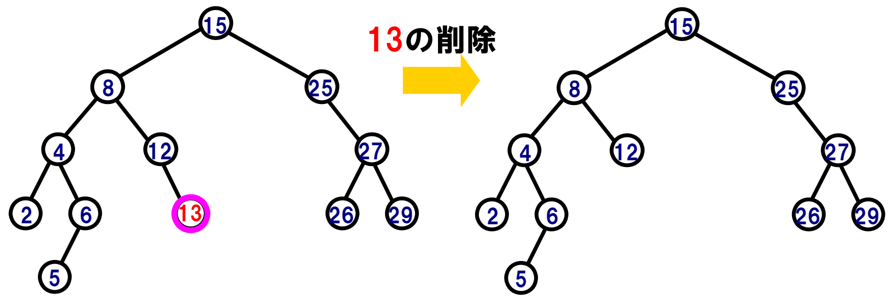
２分探索木からの削除

- **削除の方法**
  1. 削除するキーを探索
  2. そのキーを持つ節点を削除
  3. ２分探索木に変形
     - a. 削除する節点が葉の場合
     - b. 削除する節点が子を１個もつ場合
     - c. 削除する節点が子を２個もつ場合

２分探索木からの削除

a. 削除する節点が葉の場合

単純に節点を削除

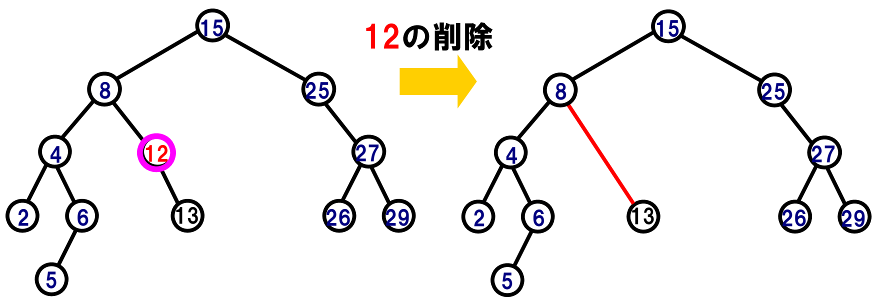


２分探索木からの削除

b. 削除する節点が子を１個もつ場合

節点を削除し，親と子を接続する

２分探索木条件が成立することに注意

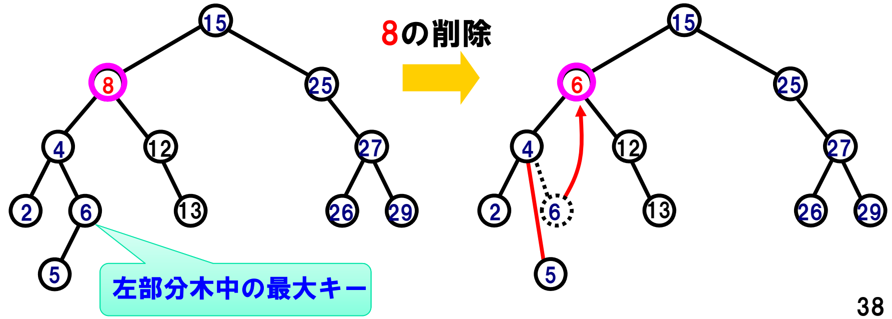


２分探索木からの削除

c. 削除する節点が子を２個もつ場合

節点を削除し，その左部分木中の最大キーを移動

２分探索木条件が成立することに注意



２分探索木からの削除

c. 削除する節点が**子を２個**もつ場合

節点を削除し，その**左部分木中の最大キー**を移動

２分探索木条件が成立することに注意

左部分木最大のキー：

探索が容易

削除が容易：右の子をもたず，子は高々１個

ケース a または b の処理

左部分木の根から探索開始

1.右の子があれば，右部分木で探索

2.右の子がなければ，そのノードが最大値

２分探索木からの削除

- **削除**にかかる時間：最悪 $O(n)$　**平均** $O(\log n)$
  1. 削除するキーを**探索**
     - 最悪 $O(n)$　平均 $O(\log n)$
  2. そのキーを持つ節点を**削除**
     - $O(1)$
  3. ２分探索木に**変形**
     - a, b　$O(1)$
     - c　最悪 $O(n)$　平均 $O(\log n)$

       左部分木での最大値探索に要する時間

直前/直後/最小/最大キーの探索

- **最小キー，最大キーの探索**
  - ２分探索木中の最小キー，最大キーを探索する
    - 最小キー：左の子をたどれるだけたどる
    - 最大キー：右の子をたどれるだけたどる
  - $O(h)$時間　（h：２分探索木の高さ）
- **直前キー，直後キーの探索**
  - ２分探索木中のキーを昇順にならべたときに，
    x の直前のキー，x の直後のキーを探索する
    - 直前キー：x に左の子があれば，左部分木の最大値
      x に左の子がなければ，根から x への経路で
      最後に右の子を選んだ節点のキー
  - $O(h)$時間　（h：２分探索木の高さ）

データ構造とアルゴリズム 第４,５回

1. アルゴリズムの重要性
2. 探索問題
3. 基本的なデータ構造
4. 動的探索問題とデータ構造
5. データの整列
6. グラフのアルゴリズム
7. 文字列のアルゴリズム
8. アルゴリズム設計手法

第４章　動的探索問題とデータ構造

4.3 平衡２分探索木

- **２分探索木**
  - 探索，挿入，削除の**平均時間**の優れたデータ構造
    - **平均** $O(\log n)$
  - ただし**最悪時**は $O(n)$
    - 木のバランスが崩れたとき
    - 木の形は挿入や削除の順で決まる
      例：データを昇順に挿入すると鎖状の木

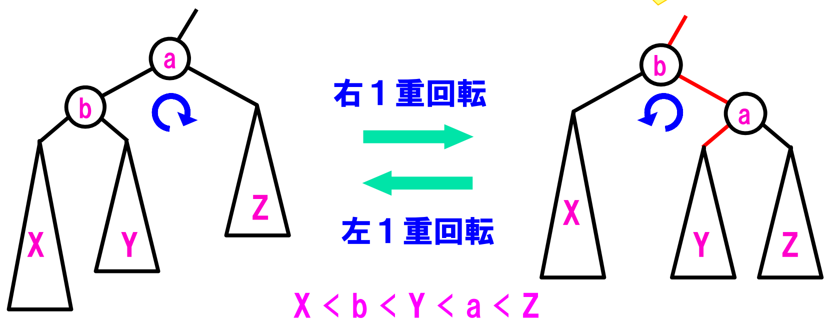
4.3 平衡２分探索木

- **平衡２分探索木**
  - 挿入，削除時の変形時に，
    木のバランスを**適度に**保つ
  - 探索，挿入，削除の**最悪時間** $O(\log n)$
    - 挿入，削除時のバランス操作の時間を含む
  - **ＡＶＬ木，２色木**など

バランスを保つための変形操作

- 回転操作
  - ２分探索木条件を保持
  - 効率よく実行可能　$O(1)$時間
  - １重回転，２重回転

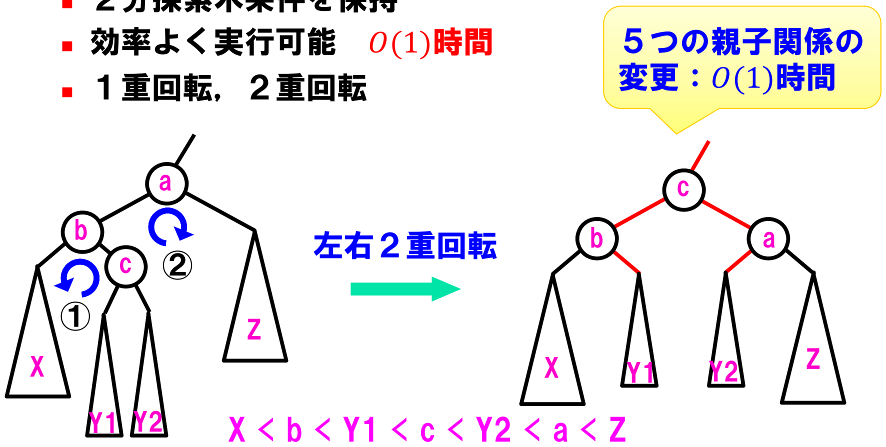
３つの親子関係の変更：$O(1)$時間



$X < b < Y < a < Z$

バランスを保つための変形操作

- 回転操作
  - ２分探索木条件を保持
  - 効率よく実行可能　$O(1)$時間
  - １重回転，２重回転

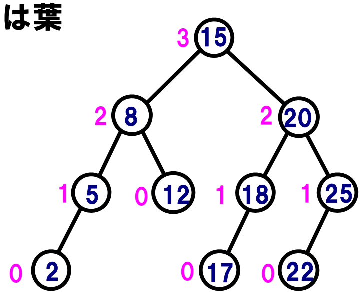


4.3.2 平衡条件

- 回転操作を用いた平衡２分探索木
  - ＡＶＬ木
  - ２色木

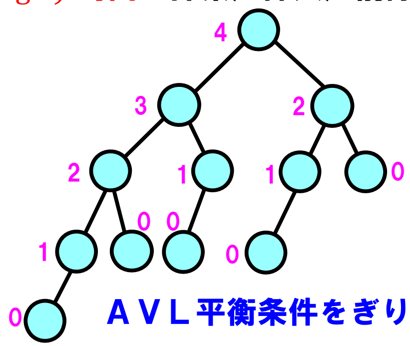
ＡＶＬ木

- ＡＶＬ木平衡条件（以下が葉以外の全頂点で成立）
  - ２つの子をもつ節点
    - 左右の部分木の高さの差≦１
      - 高さ：子孫の葉からの最大距離
  - １つの子しか持たない節点
    - その子は葉



ＡＶＬ木

- ＡＶＬ木
  - 木の高さは $O(\log n)$
    - ＡＶＬ木平衡条件から保証される
    - $O(\log n)$ 時間の探索，挿入，削除を可能にする



ＡＶＬ平衡条件をぎりぎり満たす例

２色木

- **２色木平衡条件**

  (RB0) どの内部頂点も２つの子をもつ

  (RB1) 各節点は，赤または黒のいずれかの色を持つ

  (RB2) 葉はすべて黒である

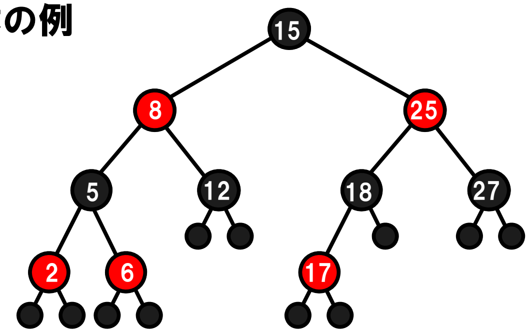
  　　　葉はデータ（キー）を保持しない

  (RB3) 赤節点の子は両方とも黒である

  (RB4) 根から葉までの全経路は同数の黒節点を含む

２色木

- ２色木の例



２色木平衡条件
(RB0) どの内部頂点も２つの子をもつ
(RB1) 各節点は，赤または黒のいずれかの色を持つ
(RB2) 葉はすべて黒である（葉はデータを保持しない）
(RB3) 赤節点の子は両方とも黒である
(RB4) 根から葉までの全経路は同数の黒節点を含む

4.3.3 ２色木の高さ

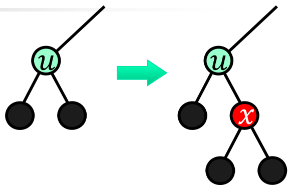
- **n 個のキーを含む２色木の高さ：$O(\log n)$**
  - h：根から葉までの各経路の黒節点数
    （根から葉までの各経路は同数の黒節点を含む）
  - 根から葉までの各経路の節点数
    h 以上　$2h$ 以下（∵赤節点の子は黒，葉は黒）
  - ２色木に含まれるキー数
    $2^{h-1}-1$ 以上　$2^{2h-1}-1$ 以下
  - $2^{h-1}-1 \le n \le 2^{2h-1}-1$ より，
    $h = O(\log n)$

4.3.4 ２色木の探索時間

- ２色木の探索
  - ２色木は２分探索木の一種
  - 探索は２分探索木と同じ方法
  - **最大探索時間**：木の高さに比例　$O(\log n)$

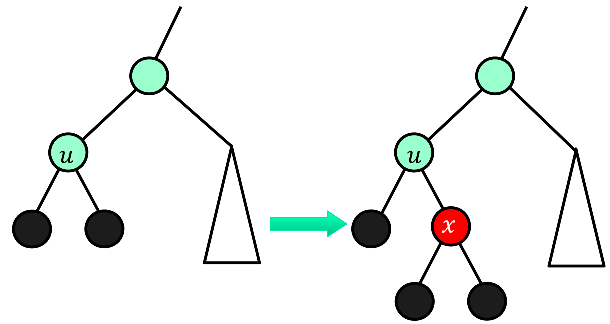
4.3.5 ２色木への挿入

- ２色木への挿入
  1. **２分探索木と同様の方法で挿入**
     - 空の葉（黒）にキー x を挿入
     - x の子として２つの空の葉（黒）を作成
     - x の色を赤に変色
       （根から葉への経路の黒節点数を保つため）
  2. **２色木条件を満たすための変形・変色**
     - x の親 u が赤のとき
       （「赤節点の子は黒」という条件を満たすため）



4.3.5 ２色木への挿入

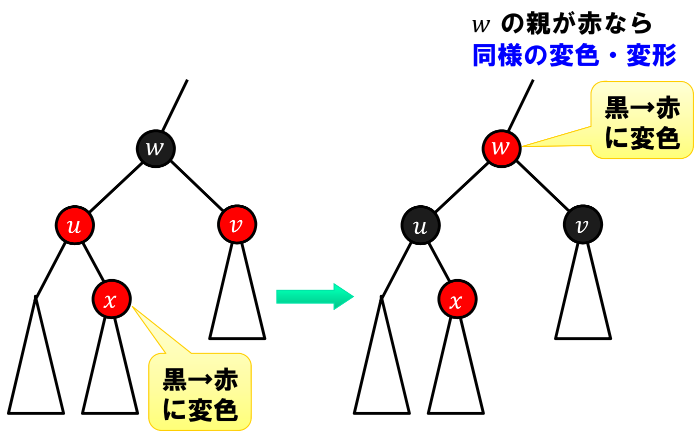
1. **２分探索木と同様の方法で挿入**
   - 空の葉（黒）にキー x を挿入
   - x の子として２つの空の葉（黒）を作成
   - x の色を赤に変色
     （根から葉への経路の黒節点数を保つため）



挿入：２色木の変形 (1)

x の親 u が赤のとき，２色木の変形（変色）が必要

i. v が赤節点の場合（v：u の兄弟）

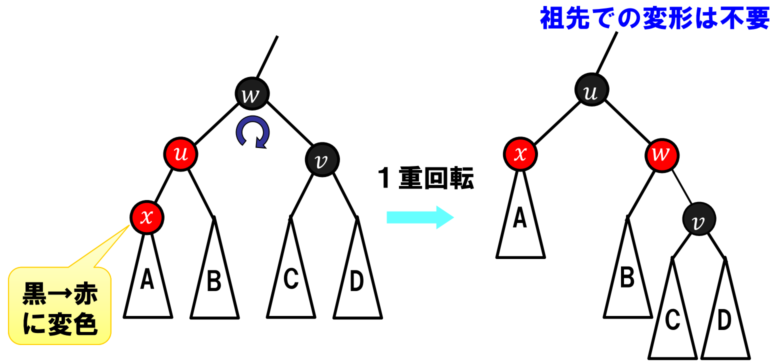


挿入：２色木の変形 (2)

x の親 u が赤のとき，２色木の変形（変色）が必要

ii. v が黒節点の場合（v：u の兄弟）

a. x が u の左の子で u が w の左の子，または，
x が u の右の子で u が w の右の子 の場合

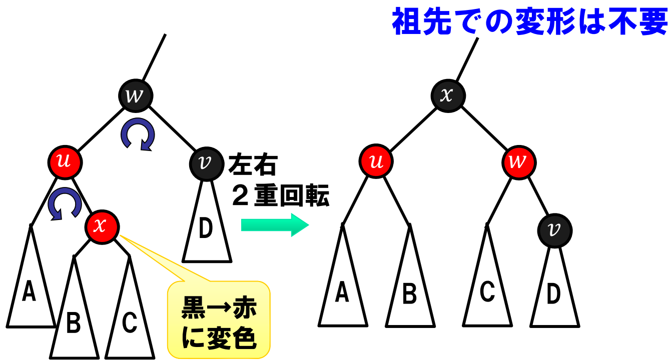


挿入：２色木の変形 (3)

x の親 u が赤のとき，２色木の変形（変色）が必要

ii. v が黒節点の場合（v：u の兄弟）

b. x が u の右の子で u が w の左の子，または，
x が u の左の子で u が w の右の子 の場合

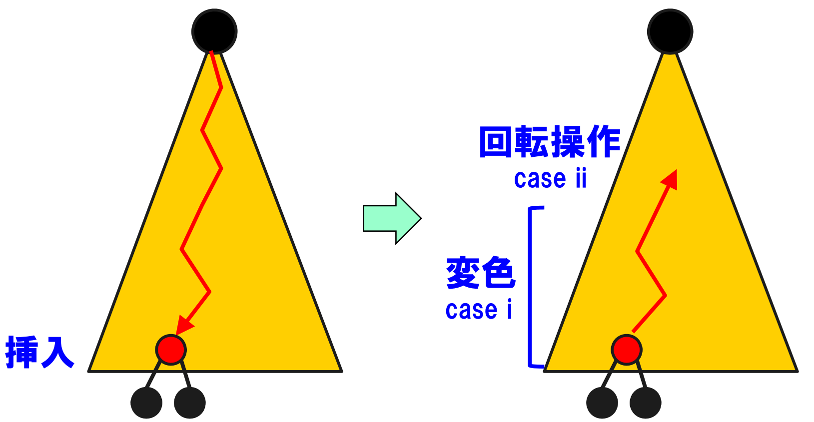


挿入の時間計算量

- 挿入の時間計算量：$O(\log n)$
  - 挿入場所の決定・挿入 $O(\log n)$
  - 変形・変色
    - 挿入した節点から根に向かって
    - 各変形・変色は $O(1)$ 時間
      - 回転操作は高々１回

挿入操作の概観

1. ２分探索木と同様に，空の葉に挿入
2. バランス条件回復のために，変色／変形

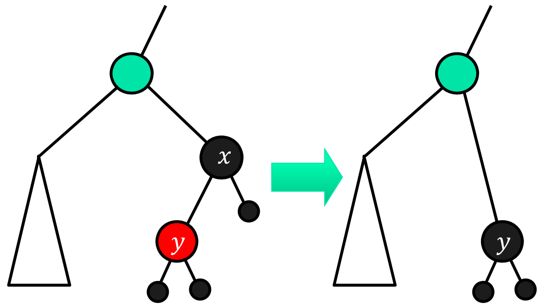


4.3.6 ２色木からの削除

- ２色木からのキー x の削除
  1. ２分探索木と同様の方法で削除
     i. x の２つの子がともに空の葉のとき
        x を空の葉に置換
     ii. x の１つの子のみが空の葉のとき
        x を抜き取り，x の親と葉でない子 yを直結
     iii. x の２つの子がともに空の葉でないとき
        左部分木の最大キーを節点 x に移動
        最大キーを保持していた節点 z を削除
        z の削除は i，ii いずれかの方法で
  2. ２色木条件を満たすための変形・変色

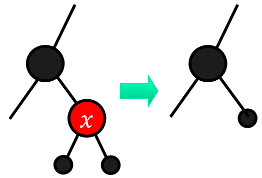
２色木からの削除 (1)

- **２色木からのキー x の削除**
  - ii. x の１つの子のみが空の葉のとき
    - x を抜き取り，x の親と葉でない子 yを直結
    - y は赤節点で２つの空の葉（黒）を持つ
    - x は黒節点

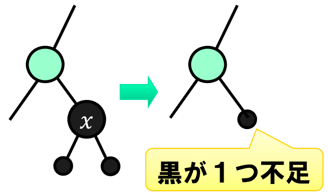


２色木からの削除 (2)

- ２色木からのキー x の削除
  - i. x の２つの子がともに空の葉のとき
    - x を空の葉に置換
    - x が赤節点の場合
      - 問題なし

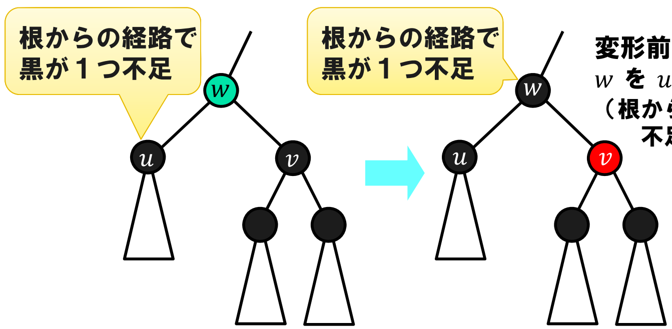


    - x が黒節点の場合
      - 根から葉までの黒節点数を揃えるために木の変形・変色が必要



２色木からの削除 (3)

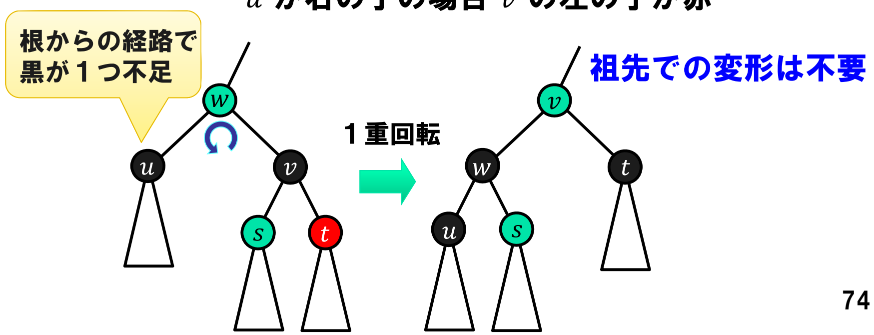
- **２色木からのキー x の削除**
  - i. x の２つの子がともに空の葉のとき
    - x が黒節点の場合
      - u：黒が１つ不足（x に置換された空の葉）
      - a. uの兄弟 v が黒，v の２つの子がともに黒



変形前の w が黒なら
w を u とみなして変形
（根からの経路で黒が１つ
不足するため）

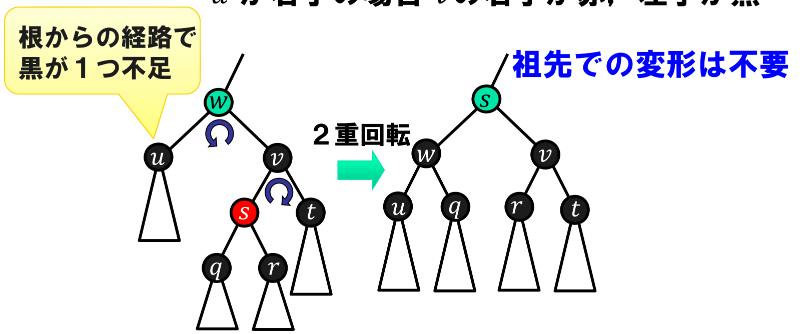
２色木からの削除 (4)

- **２色木からのキー x の削除**
  - i. x の２つの子がともに空の葉のとき
    - x が黒節点の場合
      - b. u の兄弟 v が黒，
        u が左の子の場合 v の右の子が赤，または
        u が右の子の場合 v の左の子が赤



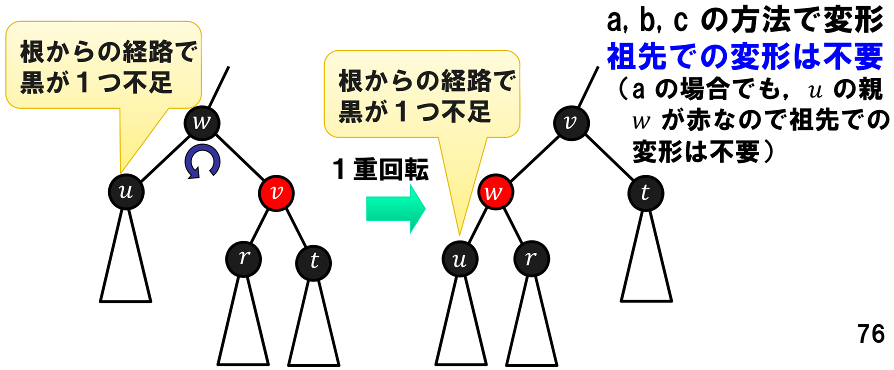
２色木からの削除 (5)

- ２色木からのキー x の削除
  - i. xの２つの子がともに空の葉のとき
    - x が黒節点の場合
      - c. u の兄弟 v が黒，
        u が左子の場合 vの左子が赤，右子が黒，または
        u が右子の場合 vの右子が赤，左子が黒



２色木からの削除 (6)

- ２色木からのキー x の削除
  - i. x の２つの子がともに空の葉のとき
    - x が黒節点の場合
      u : x に置換された空の葉
      - d. u の兄弟 v が赤

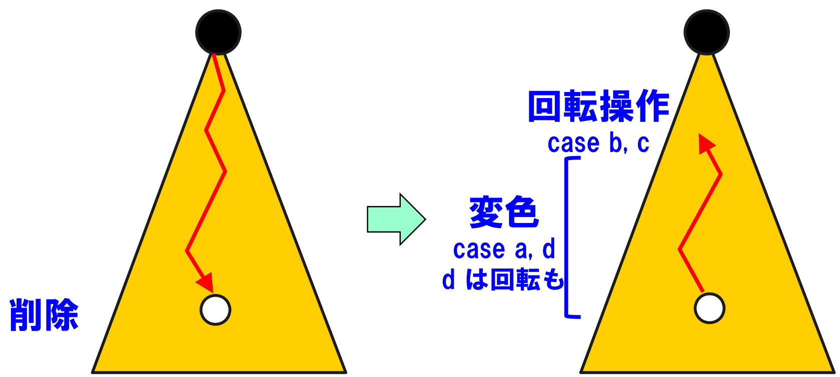


削除の時間計算量

- 削除の時間計算量：$O(\log n)$
  - 削除キーの探索・削除 $O(\log n)$
    - 左部分木中の最大キーの探索を含む
  - 変形・変色
    - 削除した節点から根に向かって
    - 各変形・変色は $O(1)$ 時間
      - 回転操作
        - １重回転：高々２回（case d の次に b）
        - ２重回転：高々１回（case c）

削除操作の概観

1. ２分探索木と同様に削除
2. バランス条件回復のために，変色／変形



第４章　動的探索問題とデータ構造

4.4 動的ハッシュ法

- ハッシュ法を挿入，削除ができるように拡張
- ハッシュ表への挿入：キー x の挿入
  - データを配列に蓄える手続きと同様

```
j = hash(x); //ハッシュ値を計算
while (htb[j] ≠ 0) //ハッシュ表で空いている場所を探す
    j = (j+1) % m; //次の場所へ移動
htb[j] = x; //最初の空き場所に x を格納
```


ハッシュ表への挿入と探索：例

0	
1	
2	
3	3
4	１５
5	５
6	２５
7	７
8	
9	２０
10	

25 の挿入

挿入：５　７　１５
２０　３　２５

ハッシュ関数
h(x) =x mod 11

② ２５の探索②

③ ６の探索　③

① ２０の探索①

0	
1	
2	
3	3
4	１５
5	５
6	２５
7	７
8	
9	２０
10	

動的ハッシュ法：削除

- ハッシュ表からのキー x の削除
  1. キー x の探索
  2. キー x の削除
     - キーが格納されていたことを示す印が必要

（参考）キー x を探索する手続き

```
探索すべきデータ x を入力する;
j = hash(x);
while (htbl[j] ≠ 0 かつ htbl[j] ≠ x)
    j = (j+1) % m; //次の場所へ移動
if htbl[j] = x then j を返して終了;
else -1 を返して終了;
```

ハッシュ表への削除と探索：例

0	
1	
2	
3	３
4	１５
5	*
6	２５
7	７
8	
9	２０
10	

５の削除

ハッシュ関数
h(x) = x mod 11

２５の探索

0	
1	
2	
3	３
4	１５
5	*
6	２５
7	７
8	
9	２０
10	

まとめ　第４章 動的探索問題とデータ構造

4.1 線形リスト上での探索
4.2 ２分探索木
4.3 平衡２分探索木
4.4 動的ハッシュ法

この章の学習目標

- 探索問題での挿入・削除とは何か，応用例を用いて説明できる
- 2分探索木，平衡2分探索木とは何か説明できる
- 挿入・削除のアルゴリズムを説明できる
  - 逐次探索，２分探索，2分探索木，2色木，ハッシュ法
- 挿入・削除アルゴリズムの計算時間を説明できる
  - 最悪時だけでなく，平均計算時間も
- 挿入・削除を含めた探索アルゴリズムのプログラムを書ける