データ構造・アルゴリズム論 解説資料集 1

山田 俊行

https://www.cs.info.mie-u.ac.jp/~toshi/lectures/algorithm/

2023 年 10 月

アルゴリズムで扱う問題の例

アルゴリズムとは，問題を解くための機械的に実行できる手順のことである．アルゴリズムで扱う問題を記述するには，入力と出力がそれぞれ何かを，取り得る値の範囲や制約条件を明確にして，できるだけ正確に述べるとよい．

●最大公約数

入力　正整数 m, n
出力　m と n の最大公約数

●列の最大値

入力　正整数 n，整数列 $a_1, \ldots, a_n$
出力　列要素 a_1 から a_n の中の最大値

● 1 変数整数多項式の値

入力　非負整数 n，整数列 $a_0, \ldots, a_n$，整数 x
出力　多項式 $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ の値

●整列

入力　正整数 n，整数列 $a_1, \ldots, a_n$
出力　列 $a_1, \ldots, a_n$ の要素を昇順に並べ替えた結果の列

●部分和

入力　非負整数 n，非負整数列 $a_1 \ldots, a_n$，非負整数 m
出力　部分集合 $I \subseteq \{1, \ldots, n\}$ のうち，$\Sigma_{i \in I} a_i = m$ を満たすもの (もしあれば)

●一筆書き (オイラー路)

入力　非負整数 n，整数の対の列 $(a_1, b_1), \ldots, (a_n, b_n)$
出力　整数列 $x_0, \ldots, x_n$ のうち，全単射 $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ が存在して，
各 $i \in \{1, \ldots, n\}$ について，$(x_{i-1}, x_i) = (a_{\pi(i)}, b_{\pi(i)})$ か $(x_{i-1}, x_i) = (b_{\pi(i)}, a_{\pi(i)})$
を満たすもの (もしあれば)

●最短巡回路

入力　正整数 n，実数平面上の座標 $(x_1, y_1), \ldots, (x_n, y_n)$
出力　n 点を 1 度ずつ通って回る最短の経路

アルゴリズムの表現 (教科書 1.1, 1.3 節)

アルゴリズムは，現実のプログラミング言語を使って表せる．この資料では，アルゴリズムの例として，最大公約数の計算と多項式の値の計算の二つを，C プログラムや文章として記述する．

●最大公約数を求めるアルゴリズム

仕様

入力　正整数 m, n
出力　m と n の最大公約数

定義に沿ったプログラム

```
1  int gcd(int m, int n) {
2    int max, i;
3    max = 1;
4    for (i = 2; i <= n; i++) {
5      if (m % i == 0 && n % i == 0) {
6        max = i;
7      }
8    }
9    return max;
10 }
```

反復を使った互除法のプログラム

```
1  int gcd(int m, int n) {
2      while (n > 0) {
3          int rem = m % n;
4          m = n;
5          n = rem;
6      }
7      return m;
8  }
```

一方が他方で割り切れるなら，その除数が最大公約数である．
そうでない間は，一方を他方で割った余りで置き換えることを繰り返す．

再帰を使った互除法のプログラム (教科書 p. 10 コード 1.4)

```
1  int gcd(int m, int n) {
2      if (n == 0) return m;
3      else        return gcd(n, m % n);
4  }
```

●多項式の値を求めるアルゴリズム

仕様

入力　非負整数 n, 整数配列 $a[0..n]$，整数 x
出力　多項式 $a_n x^n + \cdots + a_1 x^1 + a_0 x^0$ の値

定義に沿ったプログラム

```
1  int eval(int n, int a[], int x) {
2      int val, i;
3      val = 0;
4      for (i = n; i >= 0; i--) {
5          int pow, j;
6          pow = 1;
7          for (j = 1; j <= i; j++) {
8              pow *= x;
9          }
10         val += a[i] * pow;      /* pow == x^i */
11     }
12     return val;
13 }
```

ホーナー法によるプログラム

```
1  int eval(int n, int a[], int x) {
2      int val, i;
3      val = 0;
4      for (                    ) {
5          val =               ;
6      }
7      return val;
8  }
```

添え字 i を n から 0 まで変化させながら，
0 を初期値として「x を掛けて a_i を足す」という計算を繰り返す.

O 記法 (教科書 1.2 節)

O 記法を使えば，計算量などの関数値の増え方の度合いを簡潔に表せる．

●定義に沿った証明法

定義 (教科書 p. 4)

$f(n)$ は $\mathrm{O}(g(n))$
$:\Leftrightarrow$ 正実数 c と正整数 m が存在し，
任意の正整数 n について，$n \geq m$ ならば $f(n) \leq c\,g(n)$
$\Leftrightarrow$ $\exists c \in \mathbb{R}_+\ \exists m \in \mathbb{N}_+\ \forall n \in \mathbb{N}_+\ (n \geq m \Rightarrow f(n) \leq c\,g(n))$

※ $\mathbb{R}_+$ と $\mathbb{N}_+$ は，正実数全体と正整数全体の集合を表す．

例 1

$f(n) = (n^2 + 5n + 4)/2$ のとき，$f(n)$ は $\mathrm{O}(n^2)$.

【方針】ある正整数以上の n で常に $f(n) \leq c\,n^2$，となる正実数 c を見つける．c としては，係数の和 $1/2 + 5/2 + 2$ をとる．

【証明】$n \geq 1$ のとき

$$\begin{aligned} f(n) &= (n^2 + 5n + 4)/2 \\ &= (1/2)n^2 + (5/2)n + 2 \\ &\leq (1/2 + 5/2 + 2)n^2 \qquad (n \geq 1 \text{ のとき常に } 1 \leq n \leq n^2) \\ &= 5n^2. \end{aligned}$$

よって，$c = 5$ とおけば，$n \geq 1$ で常に $f(n) \leq c\,n^2$.

例 2

$f(n) = (n^2 + 5n + 4)/2$ のとき，$f(n)$ は $\mathrm{O}(n)$ ではない．

【方針】背理法による．

【証明】$f(n)$ が $\mathrm{O}(n)$ であると仮定して矛盾を導く．この仮定と O 記法の定義より，正実数 c と正整数 m が存在し，任意の正整数 n について，$n \geq m$ ならば $f(n) \leq c\,n$．ここで，

$$\begin{aligned} & f(n) \leq c\,n \\ \Leftrightarrow\ & (n^2 + 5n + 4)/2 \leq c\,n \\ \Leftrightarrow\ & n^2 + 5n + 4 \leq 2\,c\,n \\ \Leftrightarrow\ & n(n - (2c - 5)) \leq -4. \end{aligned}$$

$2c - 5$ は正整数とは限らないので，$n = \max(m, \lceil 2c - 5 \rceil)$ とおくと，n は正整数で $n \geq m$ を満たすから，上記の不等式が成り立つ．一方，n の定め方より $n > 0$ かつ $n - (2c - 5) \geq 0$ だから，$n(n - (2c - 5)) \geq 0$ であり，上記の不等式に矛盾する．

※ $\lceil x \rceil$ は x 以上の最小整数を表す．

●別の証明法

例 3

$f(n) = (n^2 + 5n + 4)/2$ のとき，$f(n)$ は $\mathrm{O}(n^2)$.

【方針】ある正整数以上の n で常に $f(n) \leq c\,n^2$，となる正実数 c を見つける．
c としては，最大次数の項の係数 $1/2$ を超える最小の整数である 1 をとる．

【証明】$c = 1$ とおく．このとき，

$$\begin{aligned} & f(n) \leq c\,n^2 \\ \Leftrightarrow\quad & n^2 + 5n + 4 \leq 2n^2 \\ \Leftrightarrow\quad & n^2 - 5n - 4 \geq 0 \end{aligned}$$

だから，2 次方程式の解の公式を使えば，$n \geq (5 + \sqrt{41})/2$ で常に $f(n) \leq c\,n^2$ であると分かる．よって，$n \geq 6$ で常に $f(n) \leq c\,n^2$.

性質

$f(n)$ は $\mathrm{O}(g(n))$ ではない
$\Leftrightarrow$ $\neg$ $f(n)$ は $\mathrm{O}(g(n))$
$\Leftrightarrow$ $\neg\, \exists c \in \mathbb{R}_+\, \exists m \in \mathbb{N}_+\, \forall n \in \mathbb{N}_+\, (n \geq m \Rightarrow f(n) \leq c\,g(n))$
$\Leftrightarrow$ $\forall c \in \mathbb{R}_+\, \forall m \in \mathbb{N}_+\, \exists n \in \mathbb{N}_+\, (n \geq m \ \wedge\ f(n) > c\,g(n))$
$\Leftrightarrow$ 任意の正実数 c と正整数 m について，
正整数 n が存在し，$n \geq m$ かつ $f(n) > cg(n)$.

例 4

$f(n) = (n^2 + 5n + 4)/2$ のとき，$f(n)$ は $\mathrm{O}(n)$ ではない.

【方針】任意の正実数 c と正整数 m について，$n \geq m$ かつ $f(n) > c\,n$ を満たす正整数 n を見つける．

【証明】c を任意の正実数，m を任意の正整数とする．

$$\begin{aligned} & f(n) > c\,n \\ \Leftrightarrow\quad & (n^2 + 5n + 4)/2 > c\,n \\ \Leftrightarrow\quad & n^2 + (5 - 2c)n + 4 > 0 \end{aligned}$$

であり，ここで実数関数 f' を $f'(x) = x^2 + (5 - 2c)x + 4$ と定める．

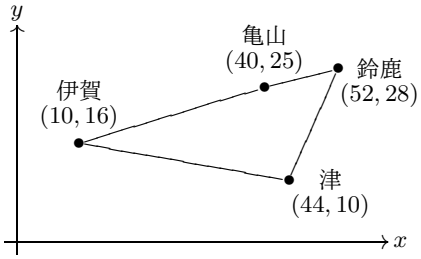
- 2 次方程式 $f'(x) = 0$ に実数解がないとき，任意の正整数 n について $n^2 + (5-2c)n + 4 > 0$ だから，$n = m$ とおけば，$n \geq m$ かつ $f(n) > c\,n$.
- 2 次方程式 $f'(x) = 0$ に実数解 x, x' (ただし $x \leq x'$) があるとき
$n > x'$ を満たす任意の正整数 n について $n^2 + (5 - 2c)n + 4 > 0$ だから，x' を超える最小の正整数を m' と表して $n = \max(m, m')$ とおけば，$n \geq m$ かつ $f(n) > c\,n$.

データ構造 (教科書 2 章 p. 15)

データ構造とは，データの集合に構造を与えて，基本操作を効率よく実行できるようにしたものである．

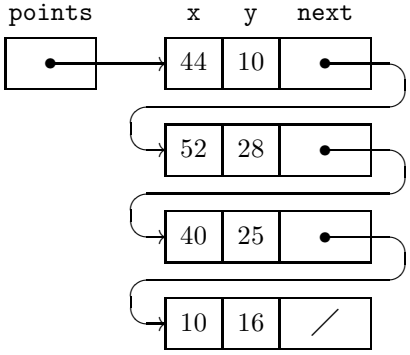
●データ構造の使用例

多角形 (頂点の座標の列)



↓

構造体のリスト



頂点間の距離

	津	鈴鹿	亀山	伊賀
津	0.0	19.7	15.5	34.5
鈴鹿	19.7	0.0	12.4	43.7
亀山	15.6	12.4	0.0	31.3
伊賀	34.5	43.7	31.3	0.0

↓

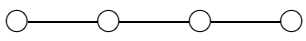
2 次元配列

	0	1	2	3
0	0.0	19.7	15.5	34.5
1	19.7	0.0	12.4	43.7
2	15.6	12.4	0.0	31.3
3	34.5	43.7	31.3	0.0

●講義で扱う主なデータ構造

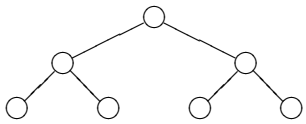
列と集合

ものの集まりを一列に並べて管理
例：配列，リスト，スタック，キュー，ハッシュ表



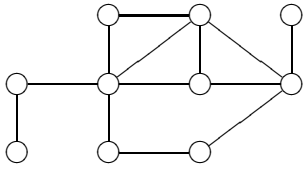
2 分木

ものの集まりを階層構造で管理
例：ヒープ，2 分探索木，2 色木



グラフ

ものの間の関係を連結構造で管理
例：隣接行列，隣接リスト



列の操作 (教科書 2.1 節)

列 (リスト) のデータ構造を使えば，ものの並びを操作できる．

●列の操作

基本操作 (教科書 p. 17)

生成	create()	
挿入	insert(l, k, a)	l … 列 (リスト)　k … 位置　a … 要素
削除	delete(l, k)	
参照	access(l, k)	
空判定	empty(l)	
部分列	sub(l, k)	

他の操作

要素数，要素の探索，要素の更新，全体の複製，など

例

$\xrightarrow{l \leftarrow \mathsf{create}()}$		$l = [\,]$			
insert$(l, 1, 10)$					
insert$(l, 2, 20)$					
$\xrightarrow{\mathsf{insert}(l, 3, 30)}$		$l = [10, 20, 30]$			位置は 1 から
$\xrightarrow{\mathsf{insert}(l, 2, 500)}$		$l = [10, 500, 20, 30]$			2 番目が 500
$\xrightarrow{\mathsf{delete}(l, 2)}$		$l = [10, 20, 30]$			
$\xrightarrow{\mathsf{access}(l, 3)}$	30	$l =$ 〃			
$\xrightarrow{\mathsf{empty}(l)}$	偽	$l =$ 〃			
$\xrightarrow{l' \leftarrow \mathsf{sub}(l, 2)}$		$l =$ 〃	$l' = [20, 30]$		l は変化なし l' は第 2 要素以降の列
$\xrightarrow{l' \leftarrow \mathsf{sub}(l, 4)}$		$l =$ 〃	$l' = [\,]$		
$\xrightarrow{\mathsf{empty}(l')}$	真	$l =$ 〃	$l' =$ 〃		

列の実現 (教科書 2.1 節)

ものの並びを扱うための列の実現方法と，列の基本操作の計算量を学ぶ.

●列

列

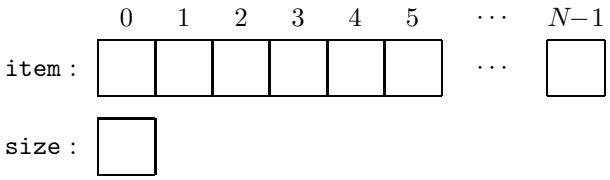
0 個以上のものの並び

(基本操作は資料集 1 p. 8)

[ ]

[10, 20, 30]
 ↑ ↑ ↑
 位置

配列による実現



＋

ー

計算量

生成

挿入

削除

参照

空判定

··· 現在の要素数　　O(1) 時間 ···

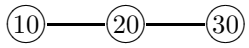
··· 最大　〃　　O(n)　〃　···

ー

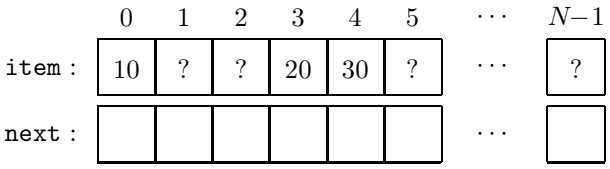
●連結リスト

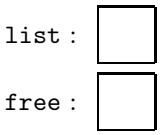
連結リスト

要素を一つずつたどる形に表す列

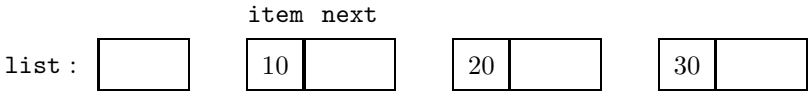


配列による実現





構造体とポインタによる実現



＋

＋

－

計算量

生成

挿入

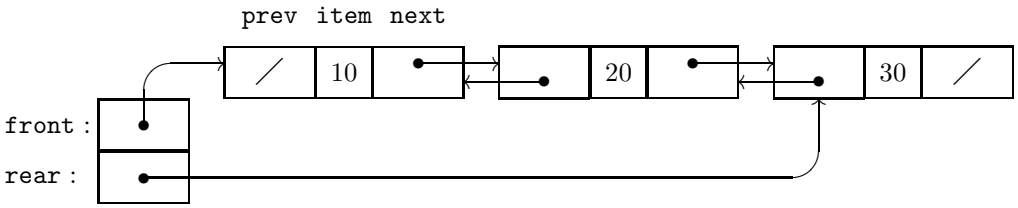
削除

参照

空判定

双方向リスト

先頭と末尾の両方から，双方向に要素をたどれるリスト



スタック と キュー の操作 (教科書 2.2, 2.3 節)

スタックとキューを使うと，列に対する操作位置を限定して効率化を図れる．

●スタックの操作

基本操作 (教科書 p. 19)

	生成	create()	
(最上段への)	挿入	push(s, a)	s … スタック　a … 要素
(最上段からの)	削除	pop(s)	
(最上段の)	参照	top(s)	
	空判定	empty(s)	

例 (スタックの最上段を列の末尾として表示)

		$s = [10, 20, 30]$	
$\xrightarrow{\text{push}(s, 40)}$		$s = [10, 20, 30, 40]$	
$\xrightarrow{\text{pop}(s)}$		$s = [10, 20, 30]$	
$\xrightarrow{\text{top}(s)}$	30	$s =$ 〃	s は変化なし
$\xrightarrow{\text{empty}(s)}$	偽	$s =$ 〃	

●キューの操作

基本操作 (教科書 p. 21)

	生成	create()	
(末尾への)	追加	insert(q, a)	q … キュー　a … 要素
(先頭からの)	削除	delete(q)	
(先頭の)	参照	top(q)	
	空判定	empty(q)	

例

		$q = [10, 20, 30]$	
$\xrightarrow{\text{insert}(q, 40)}$		$q = [10, 20, 30, 40]$	
$\xrightarrow{\text{delete}(q)}$		$q = [20, 30, 40]$	
$\xrightarrow{\text{top}(q)}$	20	$q =$ 〃	q は変化なし
$\xrightarrow{\text{empty}(q)}$	偽	$q =$ 〃	

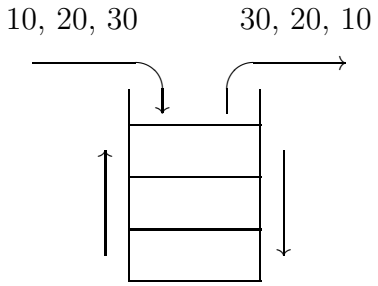
スタックとキューの 実現 (教科書 2.2, 2.3 節)

スタックとキューについて，実現方法と基本操作の計算量を学ぶ.

●スタック

スタック

追加や削除の位置を一方の端だけに限る列
(基本操作は資料集 1 p. 11)



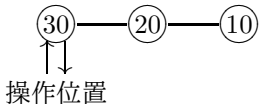
後入れ先出し (last-in first-out, LIFO)

入れた逆順に出す方式

列表現

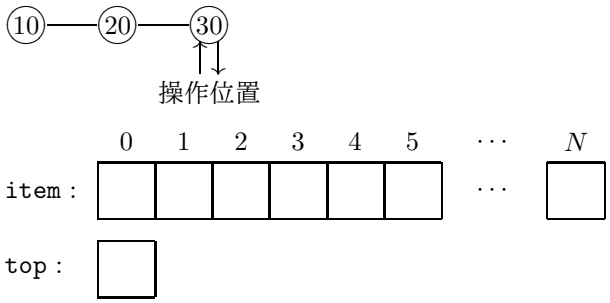
$[30, 20, 10]$ $[10, 20, 30]$
↑ ↑

列の流用による実現



	列操作の流用	時間量 配列 連結リスト
生成		
追加		
削除		
参照		
空判定		

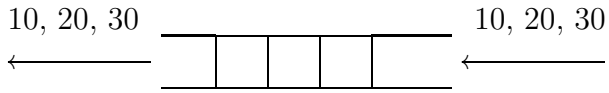
配列による実現



●キュー

キュー (待ち行列)

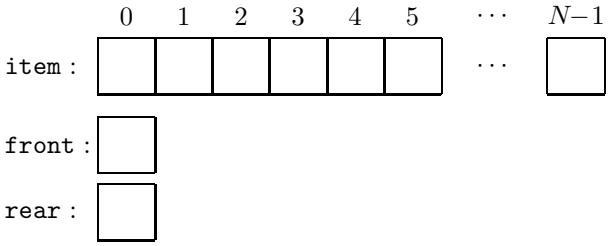
追加の位置を一方の端に，削除の位置を他方の端に限る列
(基本操作は資料集 1 p. 11)



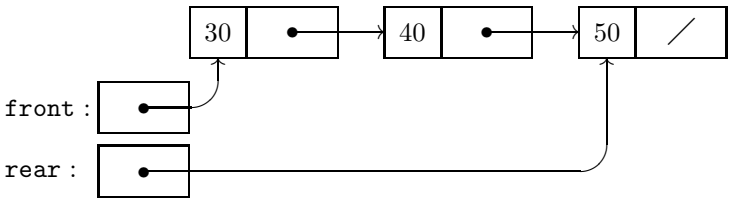
先入れ先出し (first-in first-out, FIFO)

入れた順に出す方式

配列による実現



連結リストによる実現



デク (double-ended queue, deque)

両端で 追加・削除 できる列

集合 と 写像 の実現

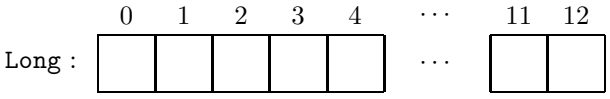
ものの集まりに対して，重複を防いだ処理や，順序を気にしない処理をするとき，集合のデータ構造が適する．また，ある集合の要素を別の集合の要素に対応付けるとき，写像のデータ構造が使える．

●集合

配列による集合の実現

$\mathrm{Long} = \{1, 3, 5, 7, 8, 10, 12\}$

↓ 自然数の集合を の配列で表す.



＋

－

計算量

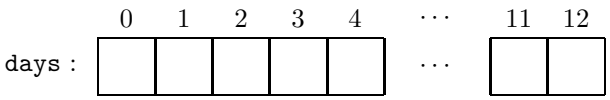
生成

挿入，削除，所属判定，空判定

●写像

配列による写像の実現

$\mathrm{days} : \{1, \ldots, 12\} \to \mathbb{Z}$
$\mathrm{days}(12) = 31$ など



＋

－

対の集合による 写像の表現

Days =

実装例 :

辞書 と 順位付きキュー の操作 (教科書 4.2, 2.4 節)

辞書は，データの検索 (データを格納しているかの判定) ができるデータ構造であり，順位付きキューは，優先度の高いデータから順に取り出せるデータ構造である．

●辞書の操作

基本操作 (教科書 p. 49)

生成	create()	
挿入	insert(d, a)	d … 辞書　a … 要素
削除	delete(d, a)	
所属判定	member(d, a)	
空判定	empty(d)	

例 (辞書を集合として表示)

		$d = \{ 10, 20, 40 \}$
$\xrightarrow{\text{insert}(d, 30)}$		$d = \{ 10, 20, 30, 40 \}$
$\xrightarrow{\text{delete}(d, 20)}$		$d = \{ 10, 30, 40 \}$
$\xrightarrow{\text{member}(d, 30)}$	真	$d =$ 〃
$\xrightarrow{\text{member}(d, 20)}$	偽	$d =$ 〃

●順位付きキューの操作

基本操作 (教科書 p. 23)

	生成	create()	
	挿入	insert(p, a)	p … 順位付きキュー　a … 要素
(最小値の)	削除	deletemin(p)	
(最小値の)	参照	findmin(p)	
	空判定	empty(p)	

例 (順位付きキューを集合として表示)

		$p = \{ 10, 20 \}$
$\xrightarrow{\text{insert}(p, 40)}$		$p = \{ 10, 20, 40 \}$
$\xrightarrow{\text{insert}(p, 30)}$		$p = \{ 10, 20, 30, 40 \}$
$\xrightarrow{\text{deletemin}(p)}$		$p = \{ 20, 30, 40 \}$
$\xrightarrow{\text{findmin}(p)}$	20	$p =$ 〃

辞書の実現 (教科書 4.2, 4.5 節)

所属判定もできるデータ集合である辞書の実現方法と，辞書の基本操作の時間計算量を学ぶ.

●辞書

辞書

挿入・削除・所属判定 を扱うデータ構造
(基本操作は資料集 1 p. 15)

キー

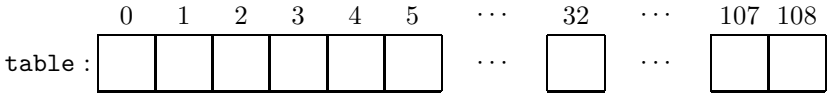
比較のために直接参照する項目

例: 番号をキーとして名前との対応を表す辞書

ハッシュ表による実現

格納位置を分散させて，高速に格納・検索できるようにした，データの集まり

例: 集合 $S = \{3, 107, 110, 250, 325, 1200\}$ を表すハッシュ表



x	3	107	110	250	325	1200
$x \mod 109$						

かちあいへの対処

-
-

ハッシュ関数

要素を格納位置に対応させる関数

ハッシュ表の操作手順

(1)

(2)

時間量

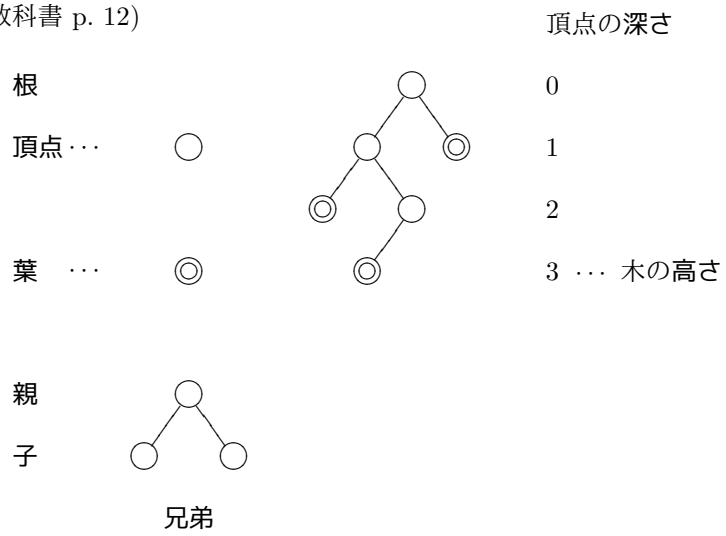
ハッシュ値を O(1) で求め，各位置のデータ集合を連結リストで表す場合

	平均	最悪	
生成			··· 格納位置の個数
挿入			
所属判定			··· 現在の要素数
削除			

木の用語 (教科書 1.4, 4.2, 2.4 節)

木のデータ構造を使うと，ものの集まりを階層的に編成して扱える．木の基本用語をまとめる．

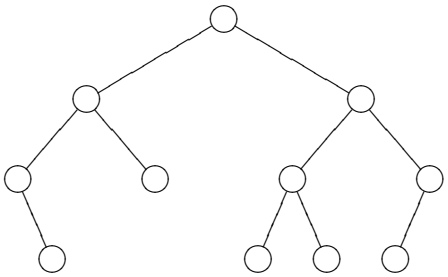
●木の用語 (教科書 p. 12)



● **2** 分木の用語 (教科書 pp. 50, 22)

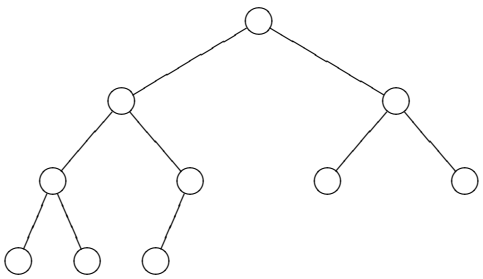
2 分木

各頂点が 2 個以下の子をもつ木



完全 **2** 分木

最下段でない頂点の親がどれも子を 2 個もち，最下段を左詰めにする 2 分木



順位付きキューの実現 (教科書 2.4 節)

線形順序の定義されたデータ集合を表せる順位付きキューの実現方法と，基本操作の時間計算量を学ぶ.

●順位付きキュー

順位付きキュー

任意の要素の挿入と，最小値の 参照・削除 ができるデータ構造
(基本操作は資料集 1 p. 15)

列による実現

- 追加順に要素を保持

$$[6,8,2] \xrightarrow{+4} \qquad \xrightarrow{-\min}$$

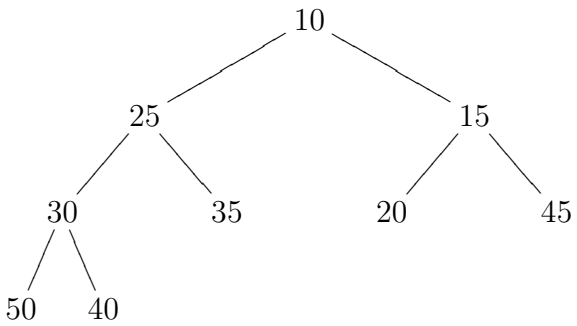
- 大小順 〃

$$[8,6,2] \xrightarrow{+4} \qquad \xrightarrow{-\min}$$

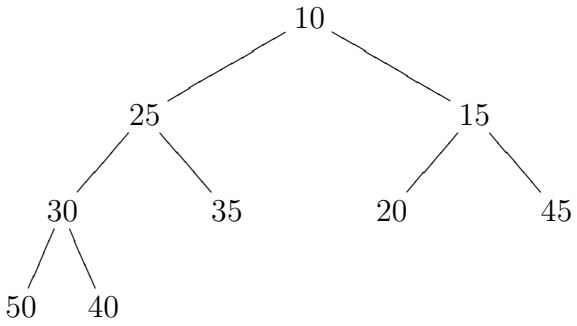
ヒープ

ヒープ条件 (親 $\leq$ 子) を満たす完全 2 分木

例: 集合 $\{10, 15, 20, \ldots, 50\}$ を表すヒープ

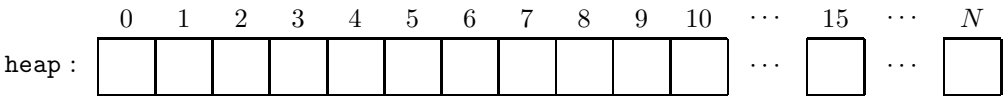


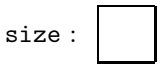
ヒープの例



配列によるヒープの実現

根から葉へ向けて 1 段ずつ，左から右の順で格納





- 根は
- heap[i] の左の子は
- 〃　右　〃

計算量 (最悪時間)

	追加順の列	ヒープ	大小順の列	··· 現在の要素数
生成				··· ヒープの高さ
挿入				最悪：
削除				
参照				

関数値の増え方の比較 (教科書 1.2 節)

計算量の見積もりに使う様々な関数を，値の増え方の観点から比較する．

●関数値の増え方の大小関係

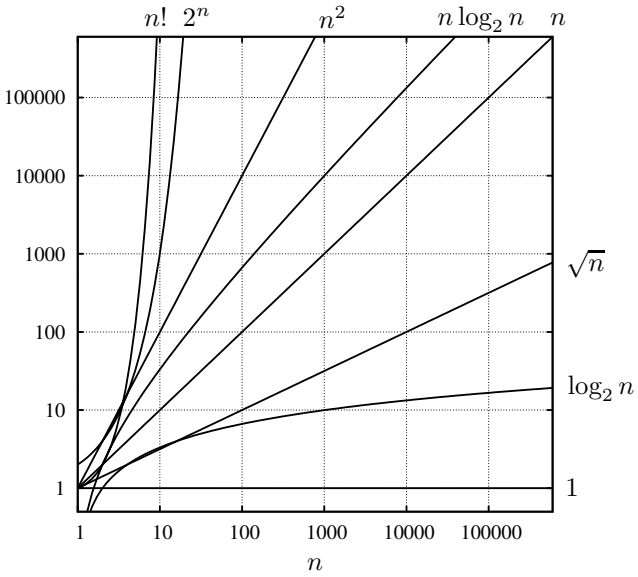
関数の値の増え方の大小関係 $\ll$ を

$$f(n) \ll g(n) \quad :\Longleftrightarrow \quad f(n) \text{ が } \mathrm{O}(g(n)) \text{ であり，かつ，} g(n) \text{ が } \mathrm{O}(f(n)) \text{ ではない}$$

により定めると，以下の大小関係が成り立つ．

$$1 \ll \log_2 n \ll \sqrt{n} \ll n \ll n \log_2 n \ll n^2 \ll 2^n \ll n!$$

この関係は，次の両対数グラフでも確認できる (参考：教科書 p. 6)．



平衡探索木 (教科書 4.3 節)

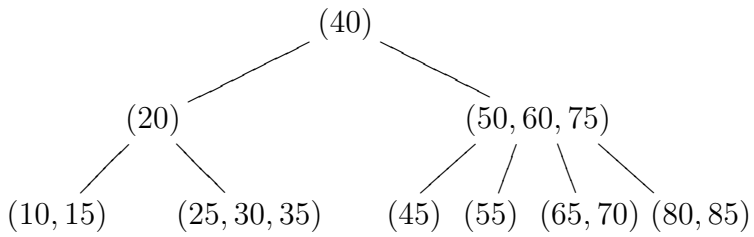
代表的な平衡探索木である 2 色木を，その基礎である 2-3-4 木と共に理解する．

● 2-3-4 木

定義

以下の条件を満たす木
- 葉の深さが全て同一
- 頂点に 1 ～ 3 個の要素を保持
- 子をもつならその数は (頂点の要素数) + 1 (つまり 2 ～ 4 個)
- 各頂点について (左の子孫の要素) < (頂点の要素) < (右の子孫の要素)

例



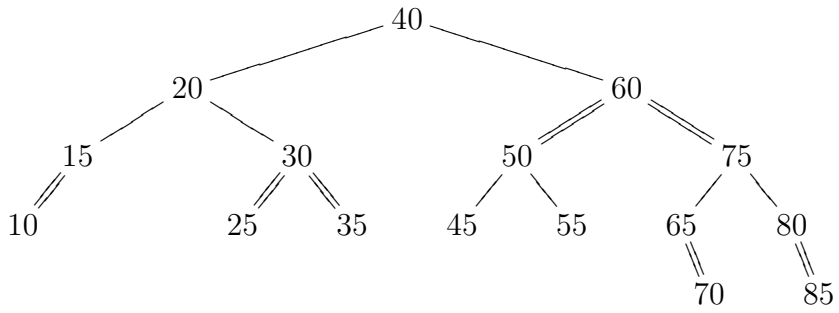
「2-3-4 木」は略称で,「平衡 2-3-4 探索木」が正式名称.

● 2 色木

定義

以下の条件を満たす，辺に赤か黒の色の付いた 2 分木
- 葉から根に向けて赤辺が続かない
- 子が一つ以下のどの頂点から出発しても, 根までに通る黒辺の数が同一
- 各頂点について (左の子孫の要素) < (頂点の要素) < (右の子孫の要素)

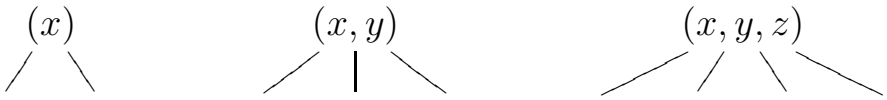
例



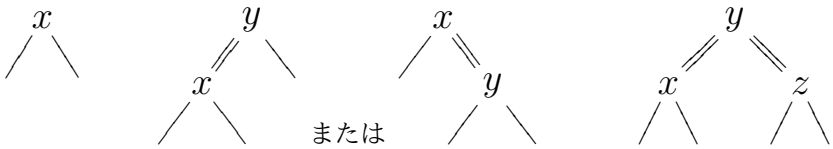
赤辺を二本線 ═══ で，黒辺を一本線 ——— で表す.

● **2-3-4** 木と **2** 色木の対応

2-3-4 木



2 色木



●参考文献

R. セジウィック,『アルゴリズム C・新版—基礎・データ構造・整列・探索』, 近代科学社, 2018. 13.3 節 (トップダウン 2-3-4 木) と 13.4 節 (赤黒木) を参照.

2-3-4 木の基本操作

平衡木のうち理解しやすい 2-3-4 木の，基本操作のアルゴリズムについて理解する．

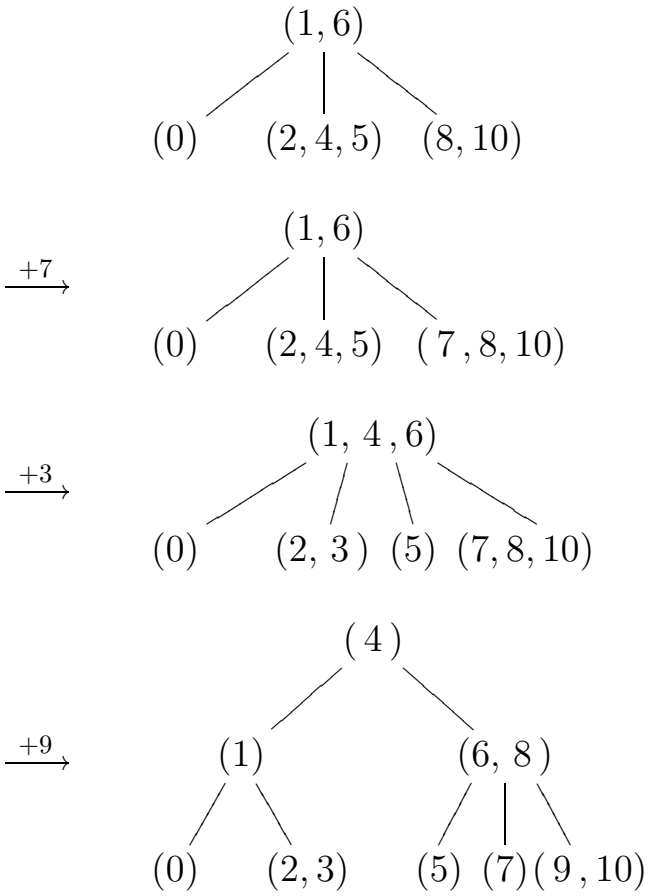
● 2-3-4 木の基本操作

所属判定 (要素の探索)

2 分探索と同様 (複数要素の頂点では逐次探索)

挿入

探索で通る各 3 要素頂点を，分割して中央値を親へ移し，
探索の終点 (2 要素以下の葉) に加える



削除

(授業では扱わない)

2 色木の基本操作の手順 (教科書 4.3 節)

代表的な平衡 2 分木である 2 色木の，基本操作のアルゴリズムについて理解する．

● 2 色木の基本操作

所属判定 (要素の探索)

2 分探索木と同じ (色情報を無視)

挿入

2 分探索木や 2-3-4 木と同様
根から葉へ探索しながら色替え (2-3-4 木での分割に相当) をし，赤辺と葉を追加
色替えや追加で赤辺が上下に続くなら，平坦化で修正

追加

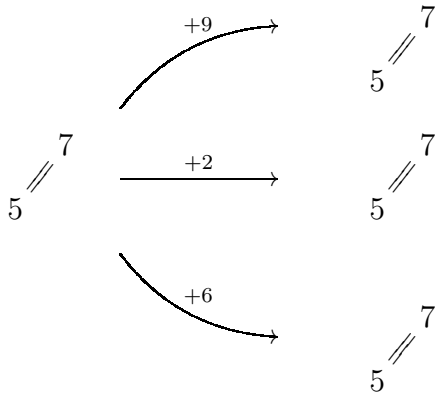
2-3-4 木 $\left(\quad (3) \quad \xrightarrow{+6} \quad (3 \quad) \quad \right)$

2 色木 $\quad 3 \quad \xrightarrow{+6} \quad 3$

追加後の平坦化

2-3-4 木
$$\left(\quad (5,\ 7) \quad \begin{array}{l} \xrightarrow{+9} \ (5,\ 7 \quad) \\ \xrightarrow{+2} \ (\quad 5,\ 7) \\ \xrightarrow{+6} \ (5, \quad 7) \end{array} \quad \right)$$

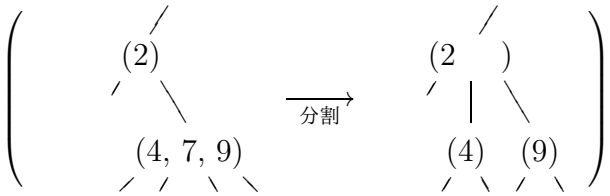
2 色木



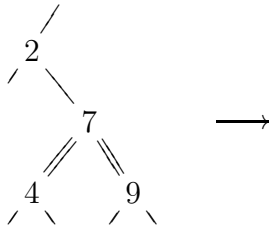
挿入 (続き)

色替え (2-3-4 木の分割に相当, 教科書 p.58 図 4.8)

2-3-4 木

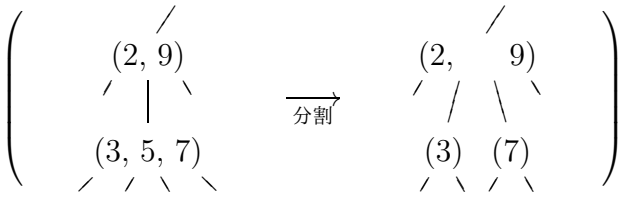


2 色木

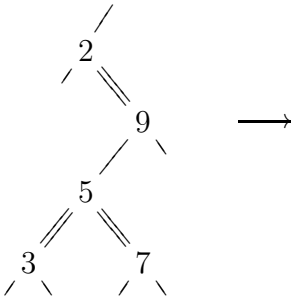


色替え後の平坦化 (教科書 p.59 図 4.9)

2-3-4 木



2 色木



削除

(授業では扱わない)

●参考文献

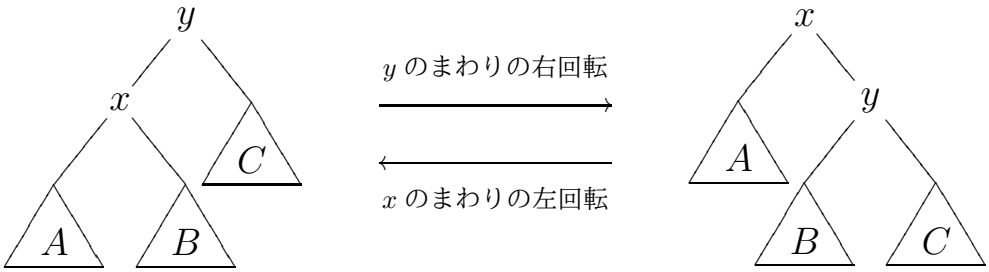
R. セジウィック,『アルゴリズム C・新版—基礎・データ構造・整列・探索』, 近代科学社, 2018.
13.3 節 (トップダウン 2-3-4 木) と 13.4 節 (赤黒木) を参照.

2 分木の回転 (教科書 4.3 節)

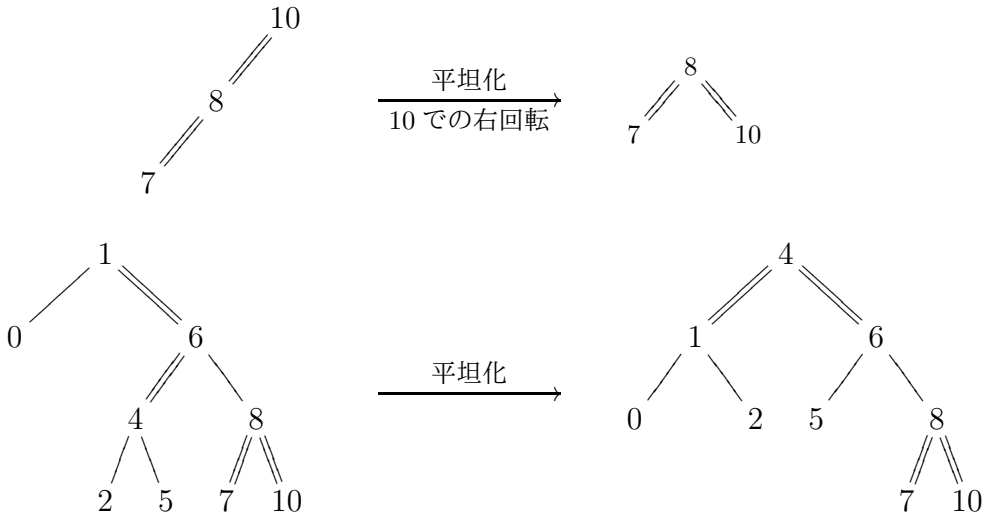
2 分木を変形するための基本操作である回転について理解する.

● 2 分木の回転

回転 (rotation)



応用例 (2 色木の平坦化：演習問題 問 9)



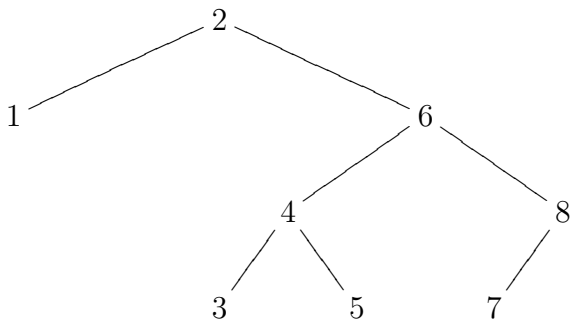
2 分木の走査 (教科書 4 章 演習問題 pp. 75–77)

2 分木の頂点や辺を順に処理するための，走査のアルゴリズムを理解する．

● 2 分木の走査

走査 (traverse)

各要素を処理しながら，データ構造全体を辿ること



先行順(行きがけ順) (preorder)

根 $\longrightarrow$ 左部分木 $\longrightarrow$ 右部分木

2 1 6 4 3 5 8 7

中間順(通りがけ順) (inorder)

左部分木 $\longrightarrow$ 根 $\longrightarrow$ 右部分木

1 2 3 4 5 6 7 8

後行順(帰りがけ順) (postorder)

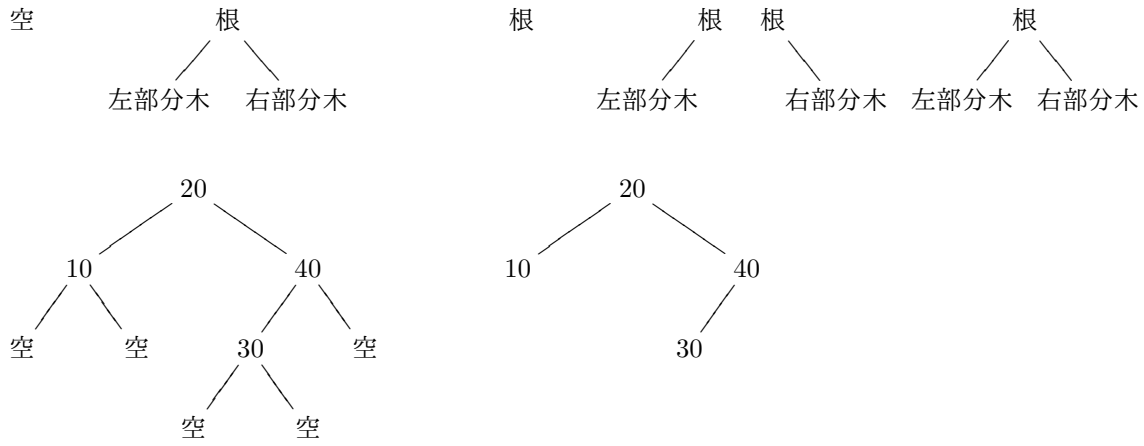
左部分木 $\longrightarrow$ 右部分木 $\longrightarrow$ 根

1 3 5 4 7 8 6 2

2 分木の 再帰構造と走査 (教科書 4 章 演習問題 pp. 75–77)

2 分木の再帰構造に基づく走査アルゴリズムを理解する.

● 2 分木の再帰構造



● 2 分木の走査アルゴリズム

```
traverse(T) {
    if (T が空) return
    visit(T の根)        // 先行順
    traverse(T の左部分木)
    visit(T の根)        // 中間順
    traverse(T の右部分木)
    visit(T の根)        // 後行順
}
```

```
traverse(T) {
    visit(T の根)        // 先行順
    if (T が左の子をもつ) traverse(T の左部分木)
    visit(T の根)        // 中間順
    if (T が右の子をもつ) traverse(T の右部分木)
    visit(T の根)        // 後行順
}
```