1. 基礎理論

1.1 計算の基礎理論
1.2 プログラムの基礎理論
1.3 数理応用
1.4 プログラム言語
1.5 アルゴリズム

1.1 計算の基礎理論

1.1.1 情報理論

1.1.2 論理と集合

1.1.3 グラフ理論

1.1.1 情報理論

● 情報量

情報の持つ不確実性の度合（ある出来事に対して、「それがどのくらい起こりにくいか」を表す量）

$$\mathbf{I(J) = -\log_2 P(J) \quad [bit]}$$

I(J)：出来事Jの情報量　　P(J)：出来事Jの発生確率

例1：サイコロの目　　$I = -\log_2 \frac{1}{6} \fallingdotseq 2.58[\text{bit}]$

例2：コインの裏表　　$I = -\log_2 \frac{1}{2} = 1[\text{bit}]$

● 平均情報量（エントロピー）

ある出来事を確定するのに必要な平均的な情報量

$$H = \sum_{k=1}^{n} \{P(J_k) \times I(J_k)\} \quad [\text{bit}]$$

出来事J_kの発生確率

出来事J_kの情報量

例：お天気

晴れ：50%　情報量$=-\log_2 \frac{1}{2} = 1[\text{bit}]$

曇り：25%　情報量$=-\log_2 \frac{1}{4} = 2[\text{bit}]$

雨：25%　情報量$=-\log_2 \frac{1}{4} = 2[\text{bit}]$

平均情報量$=(\frac{1}{2} \times 1)+ (\frac{1}{4} \times 2)+ (\frac{1}{4} \times 2)=1.5[\text{bit}]$

晴れ　曇り　雨

● 情報源のいろいろ

- マルコフ情報源

ある情報源で起こった出来事の発生確率が、この**情報源で過去に起こったn個の出来事の発生確率に依存（関係）する**とき、この情報源を**マルコフ情報源**と呼ぶ。

n=1の場合、単純マルコフ情報源。

- エルゴード情報源

ある情報源で起こった出来事の発生確率は、この**情報源を長期間観察した結果、ある値に収束する**。この情報源を**エルゴード情報源**と呼ぶ。

一般的に、世間で起こる出来事は、このエルゴード情報源。

1.1.2 論理と集合

●論理演算

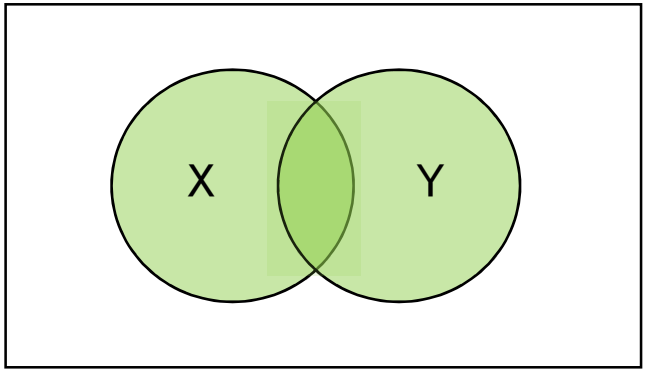
論理和(OR)

◆論理式

$$X + Y$$

◆真理値表

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1



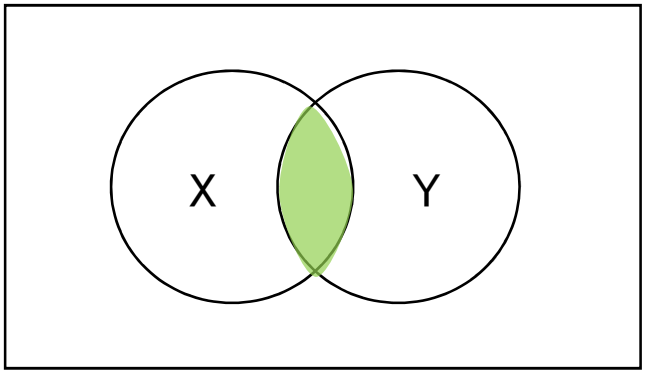
◆ベン図

XかYのどちらか一方が1ならば、1になる
また、両方1ならば、1になる

論理積(AND)

$$X \cdot Y$$

X	Y	X･Y
0	0	0
0	1	0
1	0	0
1	1	1

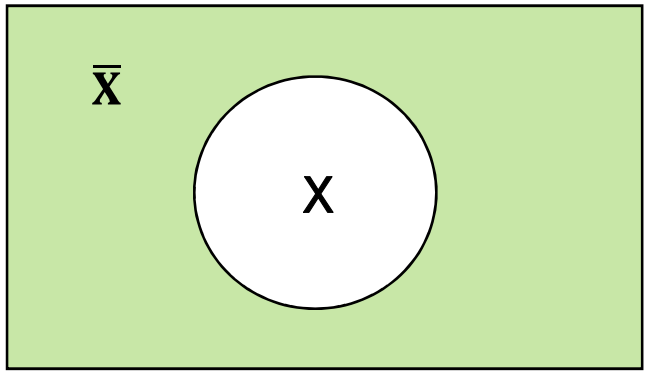


XとYがどちらも1ならば、1になる

否定(NOT)

$$\overline{X}$$

X	$\overline{X}$
0	1
1	0

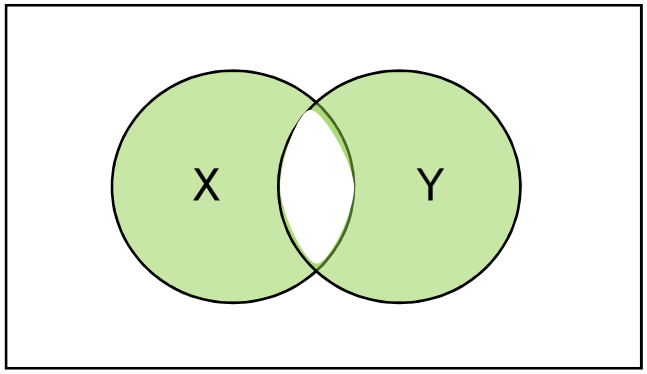


Xが0ならば、1になる

排他的論理和(XOR)

$$x \oplus y$$

X	Y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

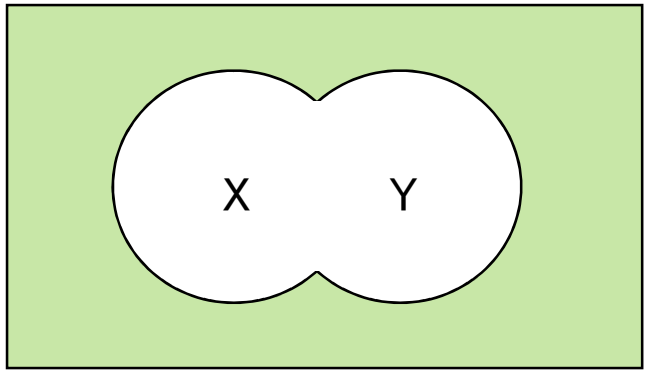


XかYのどちらか一方が1ならば、1になる

否定論理和(NOR:NOT,OR)

$$\overline{X + Y}$$

X	Y	$\overline{X + Y}$
0	0	1
0	1	0
1	0	0
1	1	0

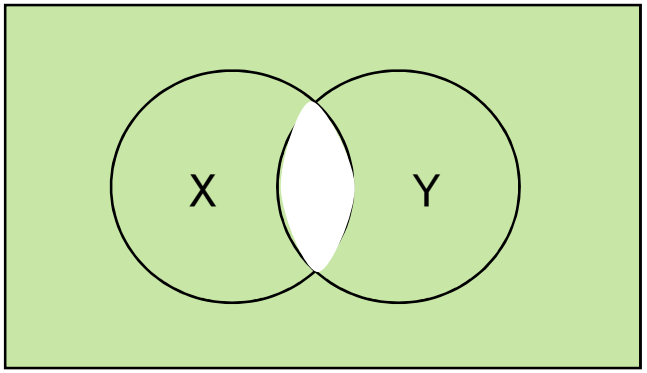


XとYの両方が0ならば、1になる

否定論理積(NAND:NOT,AND)

$$\overline{X \cdot Y}$$

X	Y	$\overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0



XとYがどちらも1ならば、0になる

論理演算　基本公式

0と1の演算

$$x \cdot 0 = 0 \qquad x \cdot 1 = x$$
$$x + 0 = x \qquad x + 1 = 1$$

分配法則

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$
$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

同一の演算

$$x \cdot x = x \qquad x + x = x$$

交換法則

$$x \cdot y = y \cdot x \qquad x + y = y + x$$

結合法則

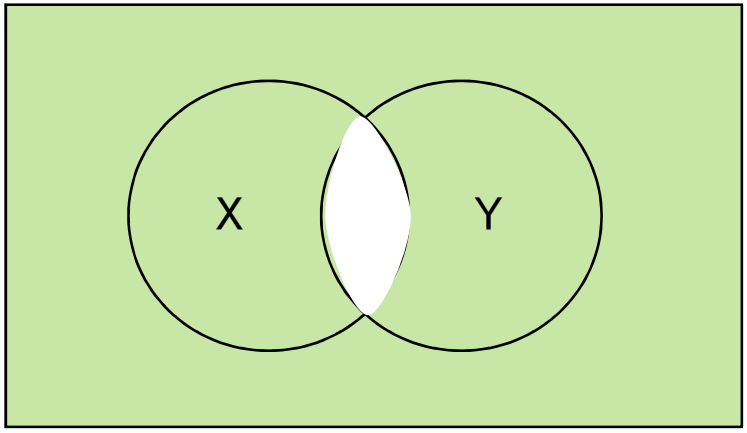
$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$
$$x + (y + z) = (x + y) + z$$

吸収法則

$$x \cdot (x + y) = x \qquad x + (x \cdot y) = x$$
$$x \cdot (\bar{x} + y) = x \cdot y \qquad x + (\bar{x} \cdot y) = x + y$$

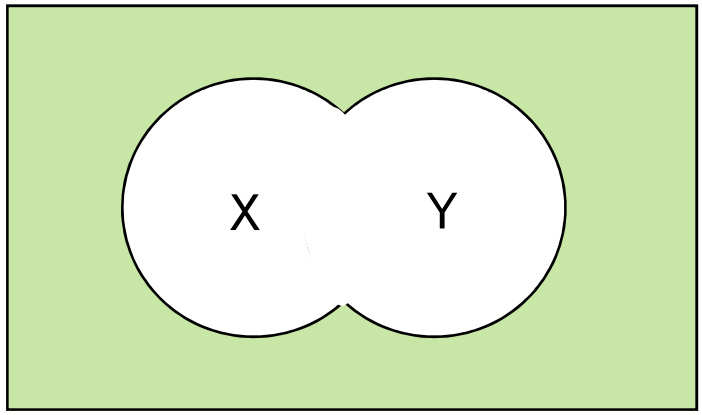
ドモルガンの法則1

$$\overline{X \cdot Y} = \overline{X} + \overline{Y}$$

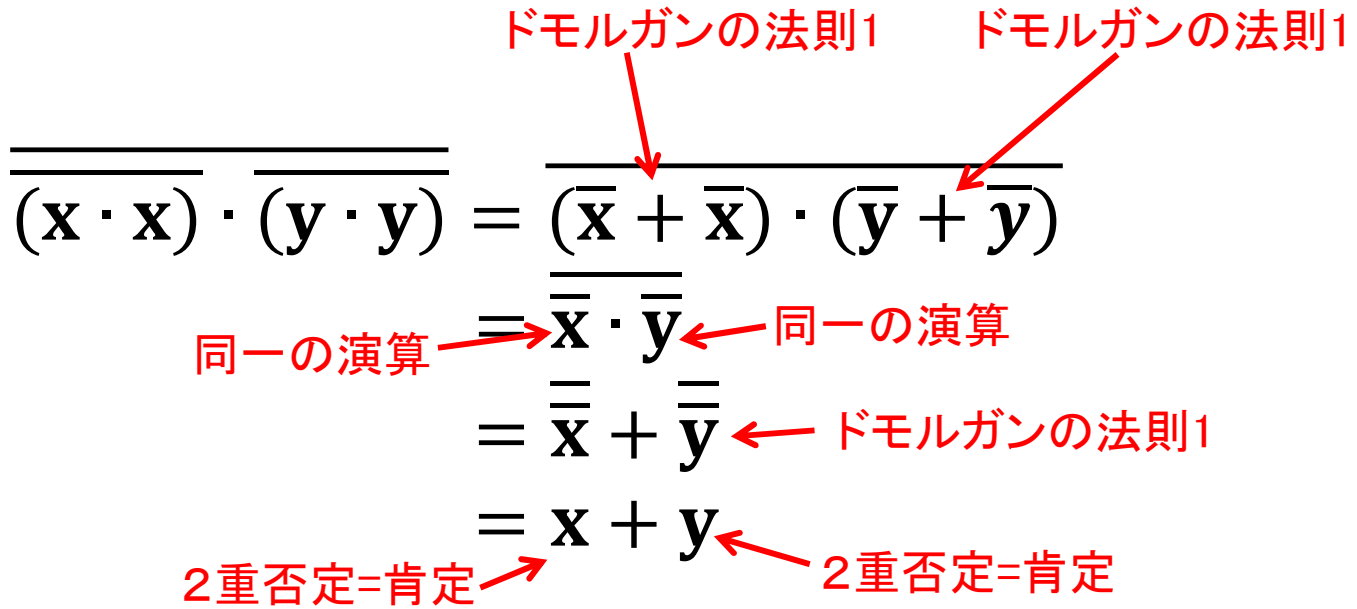


ドモルガンの法則2

$$\overline{X + Y} = \overline{X} \cdot \overline{Y}$$



複雑な論理式を、公式を使って簡略化する



●カルノー図

複雑な論理式を簡易化（簡単に）するために用い、論理式の各項が取りうるすべての値を図化したもの

複雑な論理式 → 公式 or カルノー図 → 簡単な論理式

例：論理式の簡単化

複雑な論理式

$$\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}+\overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}+\overline{A}\cdot B\cdot\overline{C}\cdot D+\overline{A}\cdot B\cdot C\cdot D$$
$$+A\cdot B\cdot\overline{C}\cdot D+A\cdot B\cdot C\cdot D$$

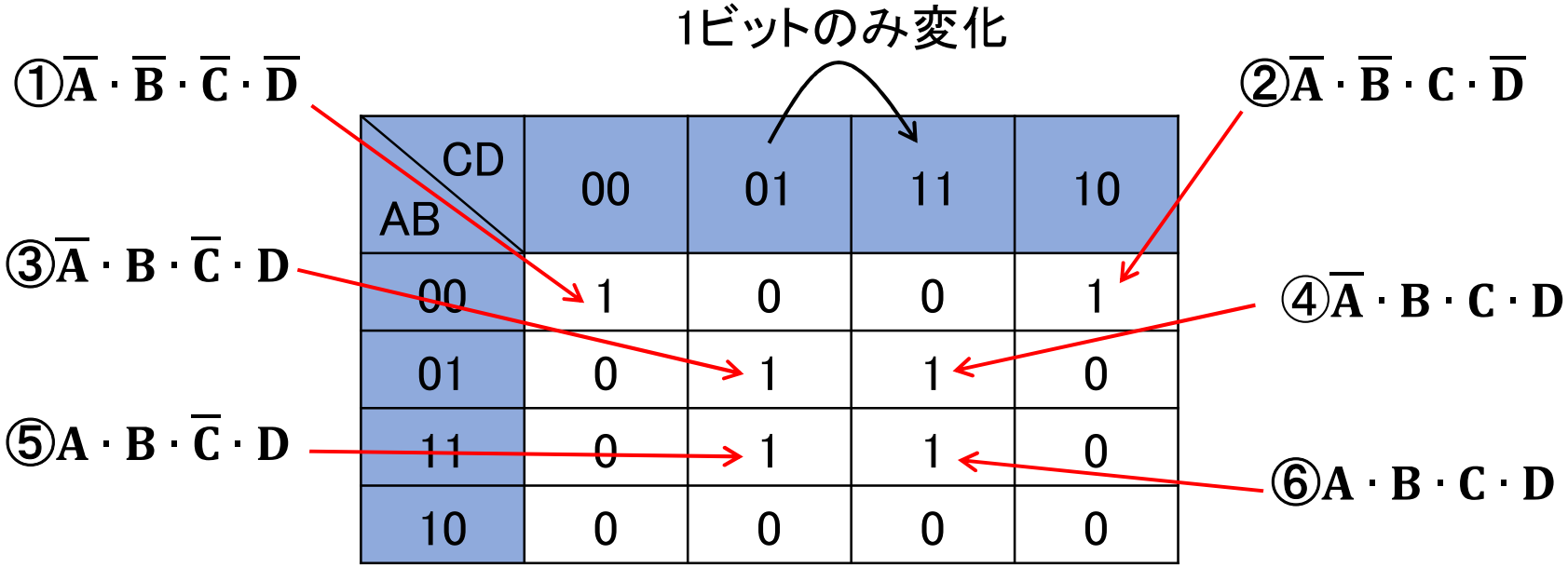
↓ 公式

簡単な論理式

$$B\cdot D+\overline{A}\cdot\overline{B}\cdot\overline{D}$$

複雑な論理式をカルノー図にする

$$\underset{①}{\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\overline{D}}+\underset{②}{\overline{A}\cdot\overline{B}\cdot C\cdot\overline{D}}+\underset{③}{\overline{A}\cdot B\cdot\overline{C}\cdot D}+\underset{④}{\overline{A}\cdot B\cdot C\cdot D}+\underset{⑤}{A\cdot B\cdot\overline{C}\cdot D}+\underset{⑥}{A\cdot B\cdot C\cdot D}$$



カルノー図から簡単な論理式を導く

カルノー図中のすべての「1」のセルを、次の規則に従って囲む

- セルの数は、できるだけ多くなるようにする
- 囲んだセルの数が、2^n個（1,2,4,8,・・・個）になるようにする
- 同じセルを、2つ以上の枠で囲んでもよい
- カルノー図の上の端、左右の端は連続していると考える

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

囲んだセルを(簡単な)論理式にする

- 緑の枠の論理式

$\overline{A} \cdot \overline{B} \cdot \overline{D}$

- 赤の枠の論理式

$B \cdot D$

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

●論理演算の応用

複雑な論理式を、公式を使って簡易化（簡単に）する

分配法則

$$X = \overline{A} \cdot \overline{B} + \overline{A} \cdot B + A \cdot \overline{B}$$

$$= \overline{A} \cdot \left(\overline{B} + B\right) + A \cdot \overline{B}$$

$\overline{B} + B = 1$

$$= \overline{A} + A \cdot \overline{B}$$

$\overline{A} \cdot 1 = \overline{A}$（0と1の演算）

$$= \overline{A} + \overline{B}$$

吸収法則

$$= \overline{A \cdot B}$$

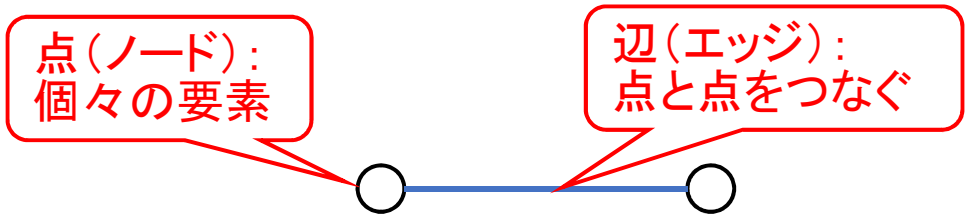
ドモルガンの法則

1.1.3 グラフ理論

- **グラフとは**

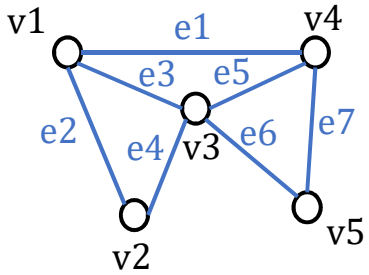
　点（ノード）の集合と辺（エッジ）の集合からなる図形で、複雑に絡み合ったデータを表すのに適している

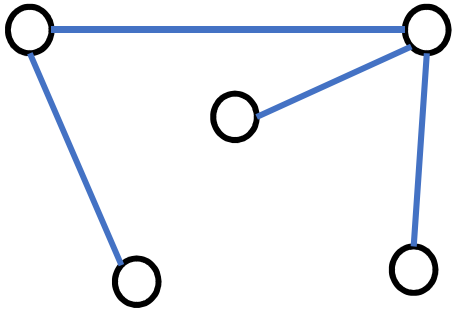
例：オフィスのLAN配線図、インターネットのサイト構成図



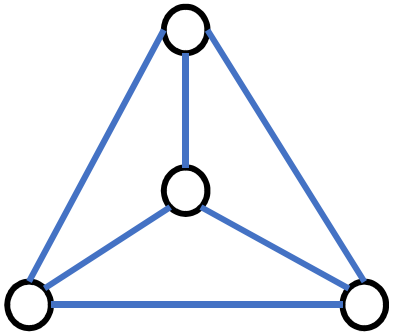
- **グラフの定義と種類**

グラフG=(V, E) → 点（ノード）の集合V=(v1,v2,v3,v4,v5)
辺（エッジ）の集合E=(e1,e2,e3,e4,e5,e6,e7)





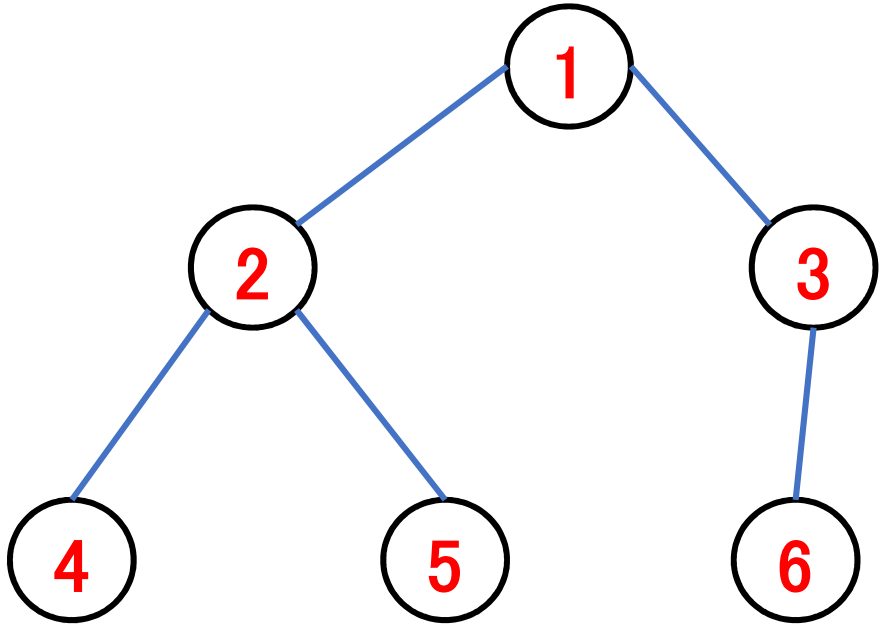
- 木：閉じていないグラフ



- 完全グラフ：2点間がすべて隣接している

- **木の巡回法**
  木において、すべての点を巡回する方法

  - 幅優先順
    同一レベルの点を順番に巡回する

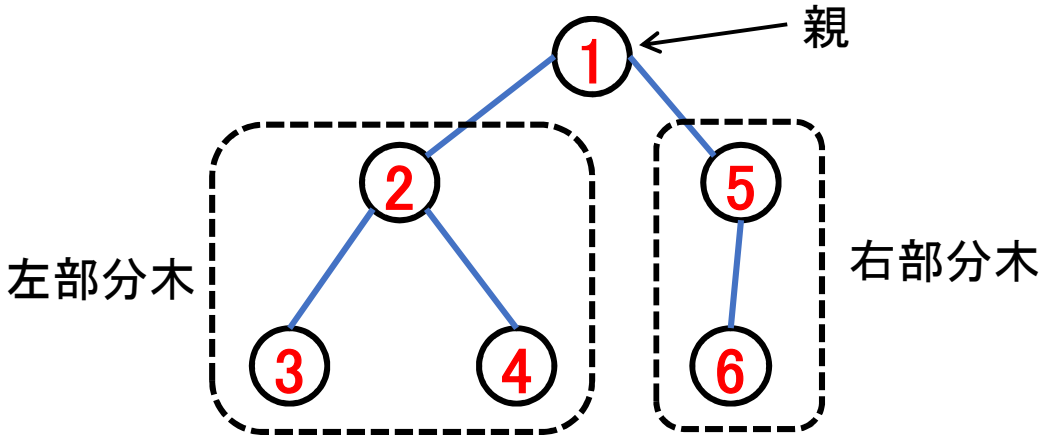


※番号順に巡回する

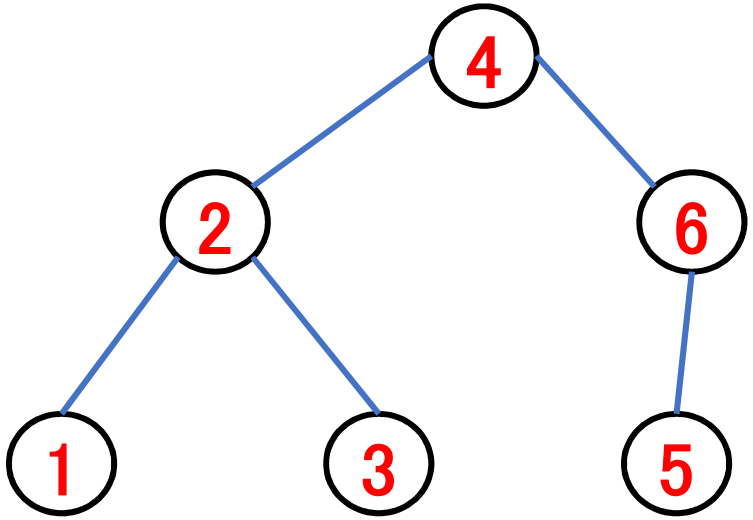
- 深さ優先順

※番号順に巡回する

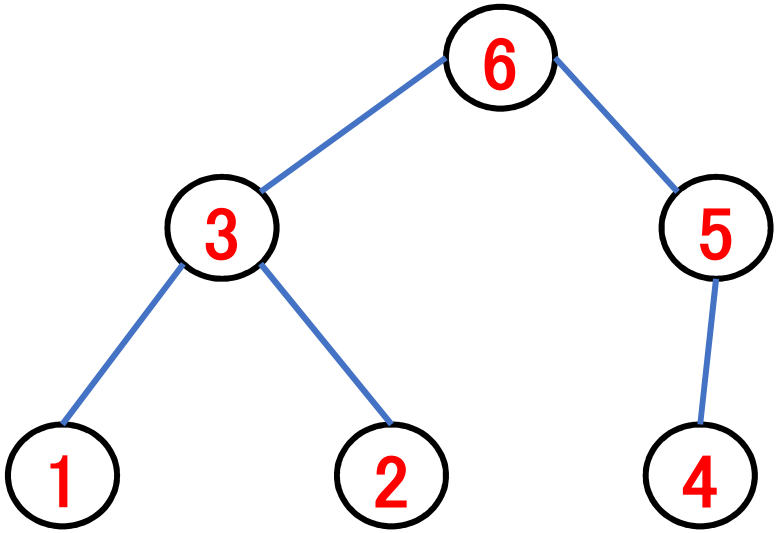
〈先行順〉 親→左部分木→右部分木



〈中間順〉 左部分木→親→右部分木



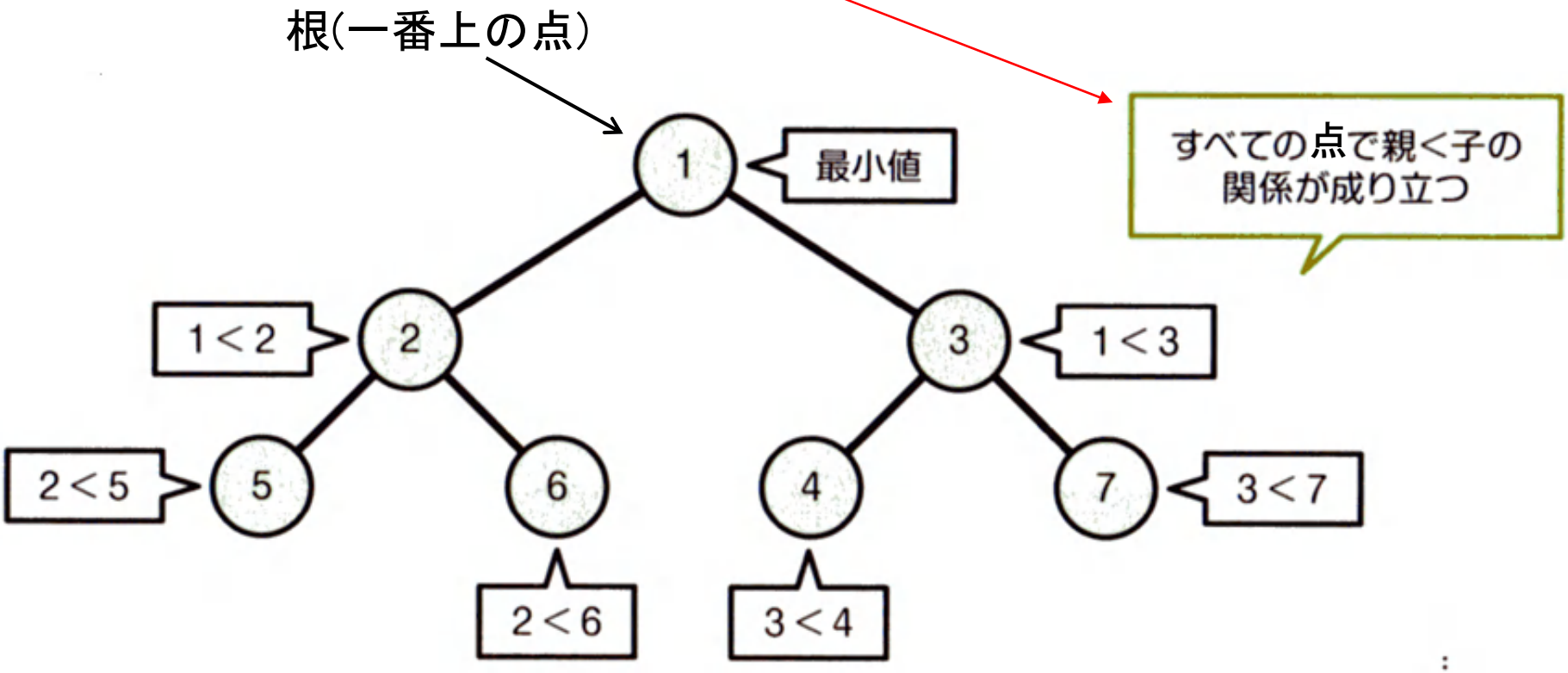
〈後行順〉 左部分木→右部分木→親



※番号順に巡回する

- **ヒープ**

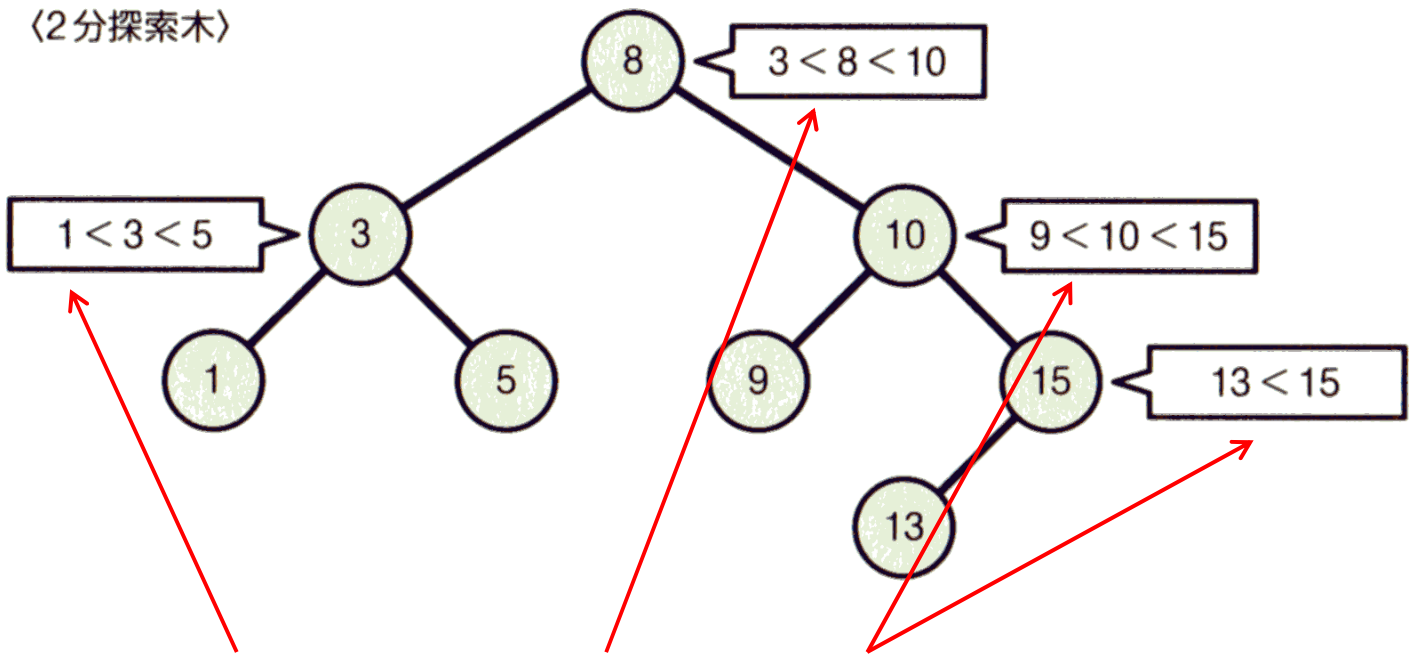
  すべての点で「親＜子」または「親＞子」の関係が成り立つ木構造で、根が最小値になる



※点(ノード)内は数値

- **2分探索木**

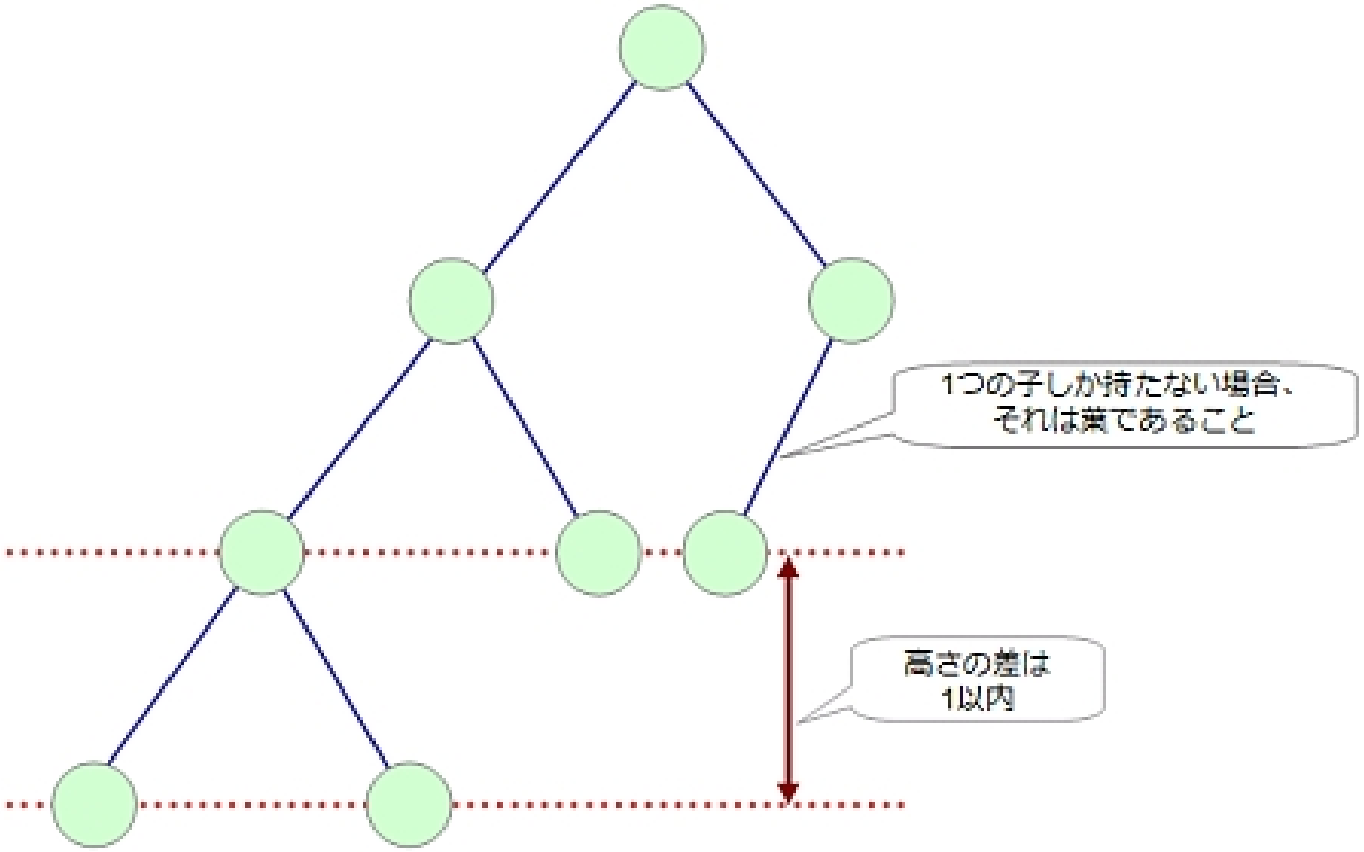
　親のデータが、左部分木のデータより大きく、右部分木のデータよりも小さいとき、2分探索木と呼び、必要なデータを検索するときには、少ない回数で検索できる



全ての点で（ 左の子の値 ＜ 親の値 ＜ 右の子の値 ）の条件を満たしている

- **AVL木（Adelson-Velskii and Landis‘ tree）**
  部分木の深さ（差）が、最大1である2分探索木

<特徴>
- 2つの子を持つ各節点では、左部分木と右部分木の高さが1以内であること
- 1つの子しか持たない各節点では、その子は葉であること



- **B木**

根からすべての葉までの深さが等しく、かつ、以下の条件を満たす多分木

〈特徴〉

- 根以外の各節では、k個以上の要素がある
- 各節では、最大2k個の要素がある
- 各節では、要素数に1を足した数の子がある

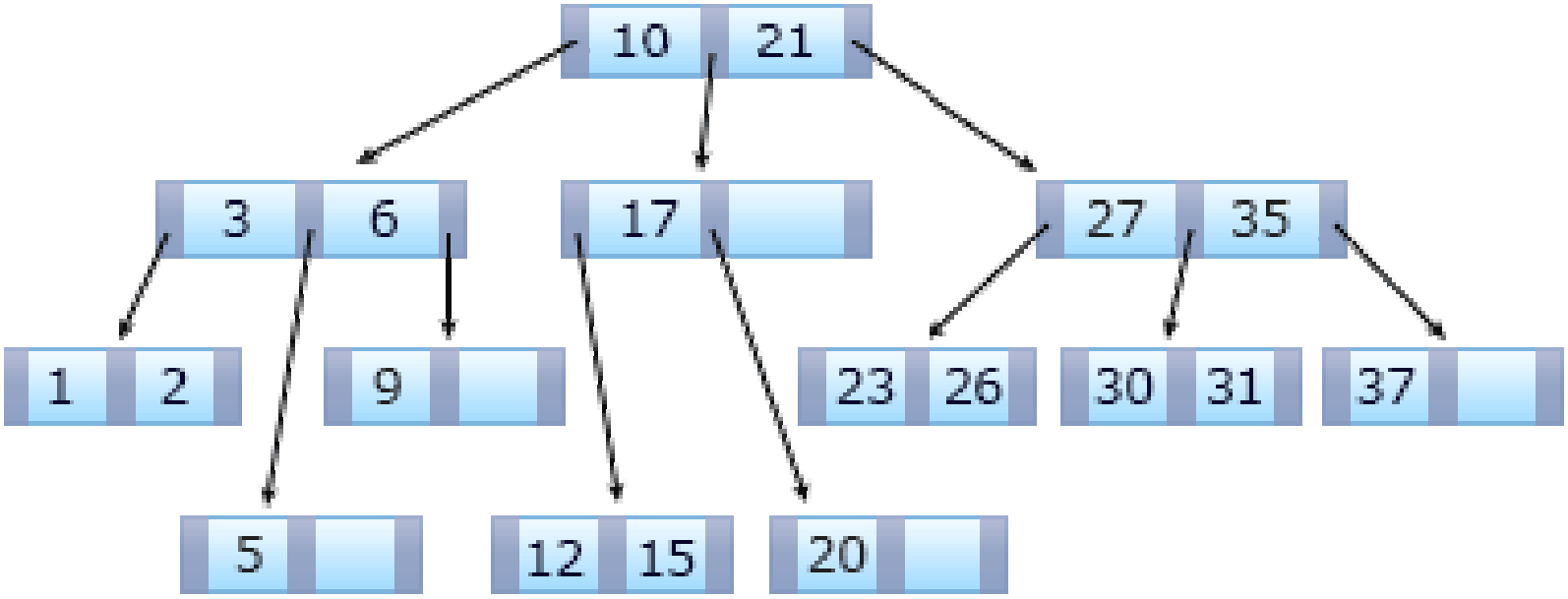
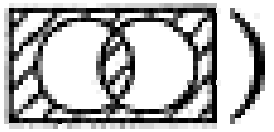


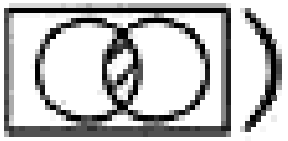
図　B木

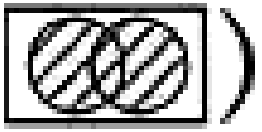
■ 過去問題1

任意のオペランドに対するブール演算Aの結果とブール演算Bの結果が互いに否定の関係にあるとき，AはBの(又は，BはAの)相補演算であるという。排他的論理和の相補演算はどれか。

ア：等価演算 

イ：否定論理和 

ウ：論理積 

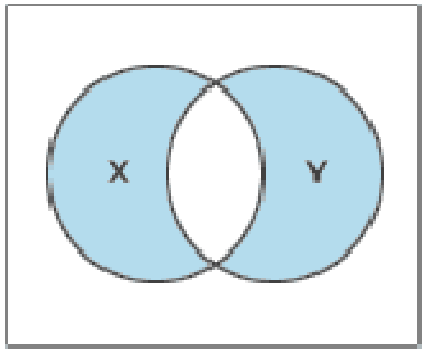
エ：論理和 

相補演算とは、集合演算によって得られる結果が互いにもう一方の演算の補集合となっている関係、すなわちAとA、X AND YとNOT (X AND Y)のような関係になっているものを言う。

排他的論理和(XOR)は、2つの入力値が異なれば真、同じであれば偽を返す論理演算で、演算結果は次のような真理値表となる。

排他的論理和の真理値表とベン図

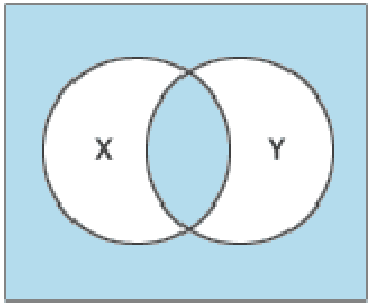
X	Y	結果
0	0	0
0	1	1
1	0	1
1	1	0



排他的論理和の相補演算になるのは、XORの補集合(XORのベン図の白い部分)が結果として得られる演算なので、答えとして適切なのは「等価演算」ということになる。

等価演算の真理値表とベン図

X	Y	結果
0	0	1
0	1	0
1	0	0
1	1	1



1.2 プログラムの基礎理論

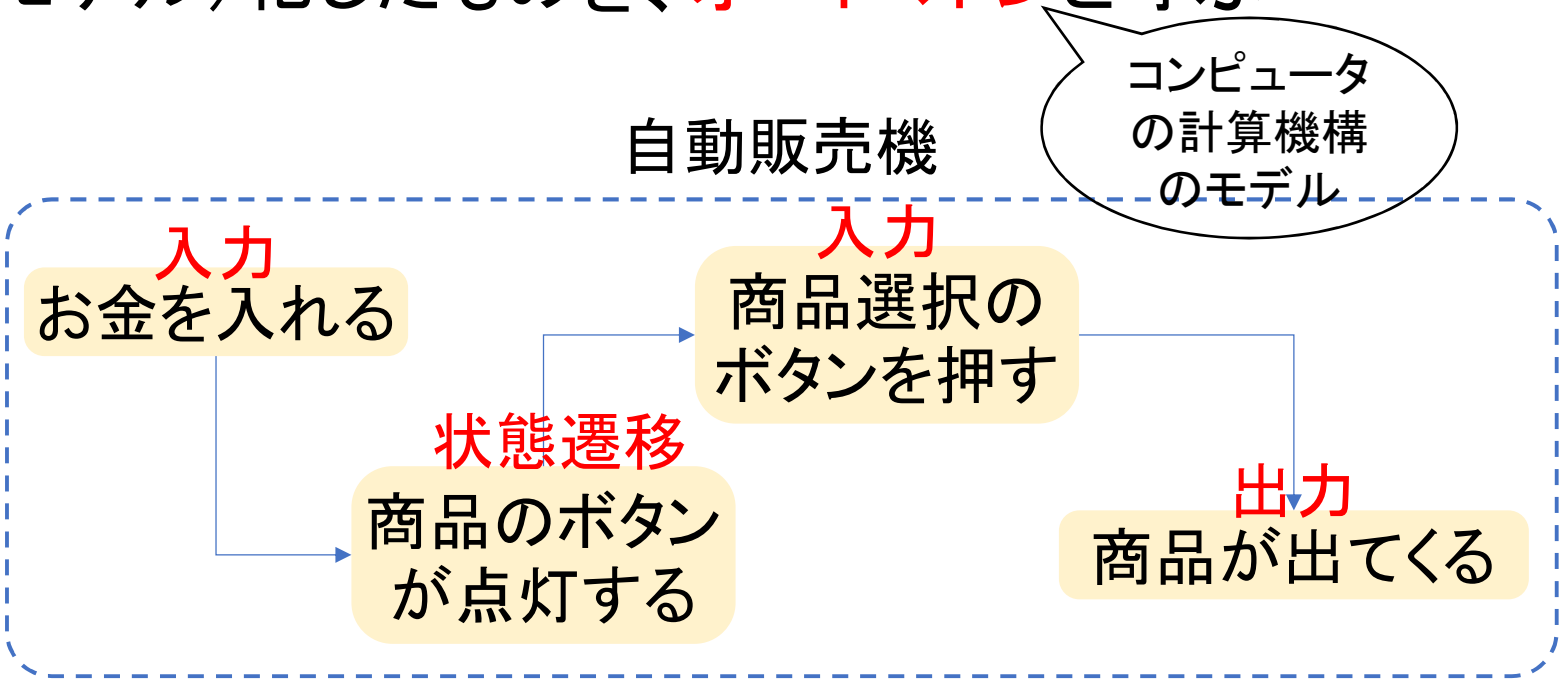
1.2.1 オートマトン

1.2.2 計算量と正当性

1.2.3 演算と精度

1.2.1 オートマトン

「内部の状態が、入力を記憶することで変わっていく（遷移する）」このように、状態遷移を伴う動作を抽象（モデル）化したものを、オートマトンと呼ぶ



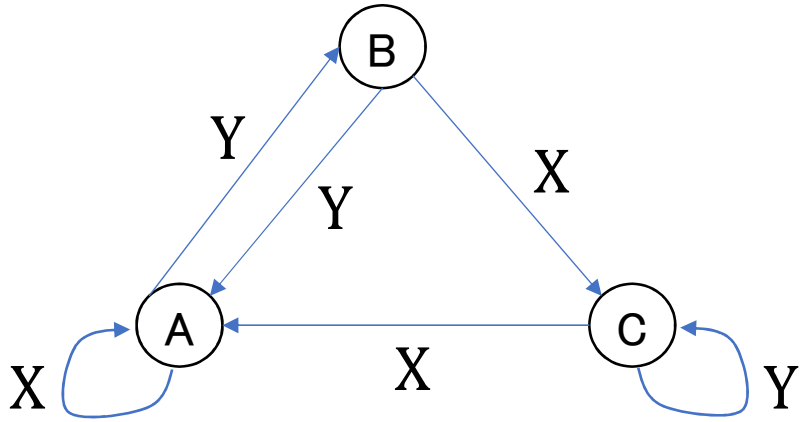
状態遷移表 と 状態遷移図

オートマトンは、状態遷移表や状態遷移図で表現することができる

状態遷移表

入力 / 現在の状態	入力 X	入力 Y
状態 A	状態 A	状態 B
状態 B	状態 C	状態 A
状態 C	状態 A	状態 C

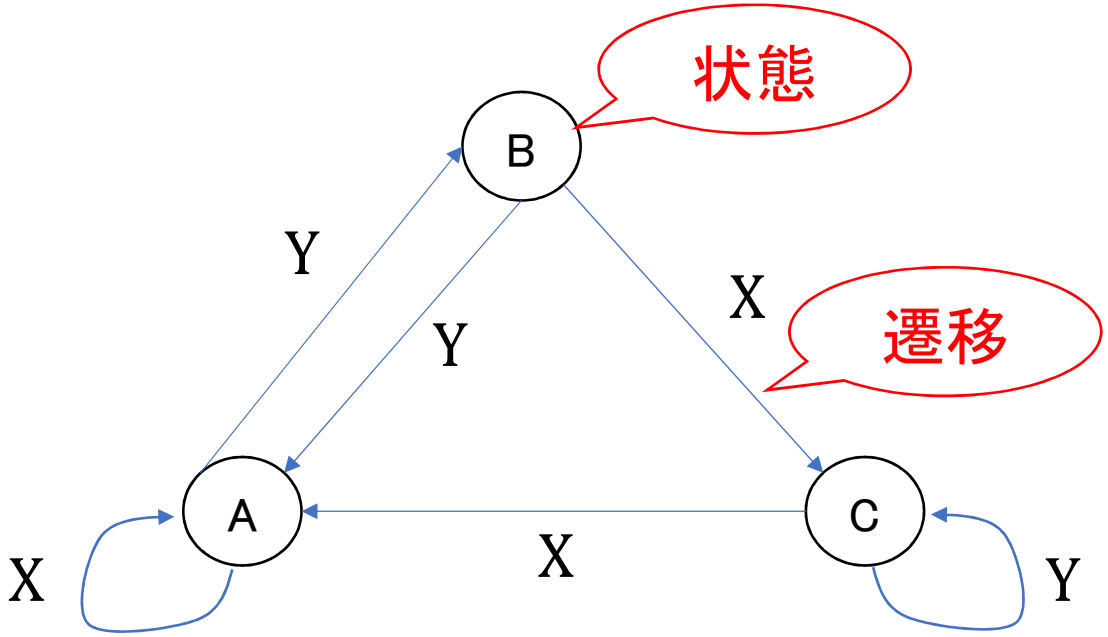
入力によって
状態が変わる



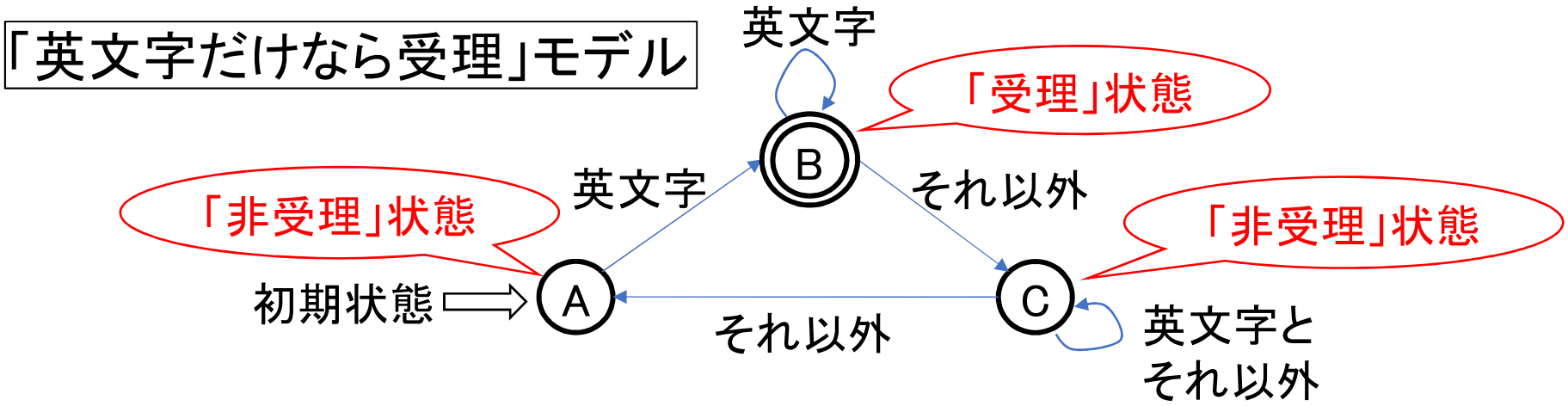
状態遷移図

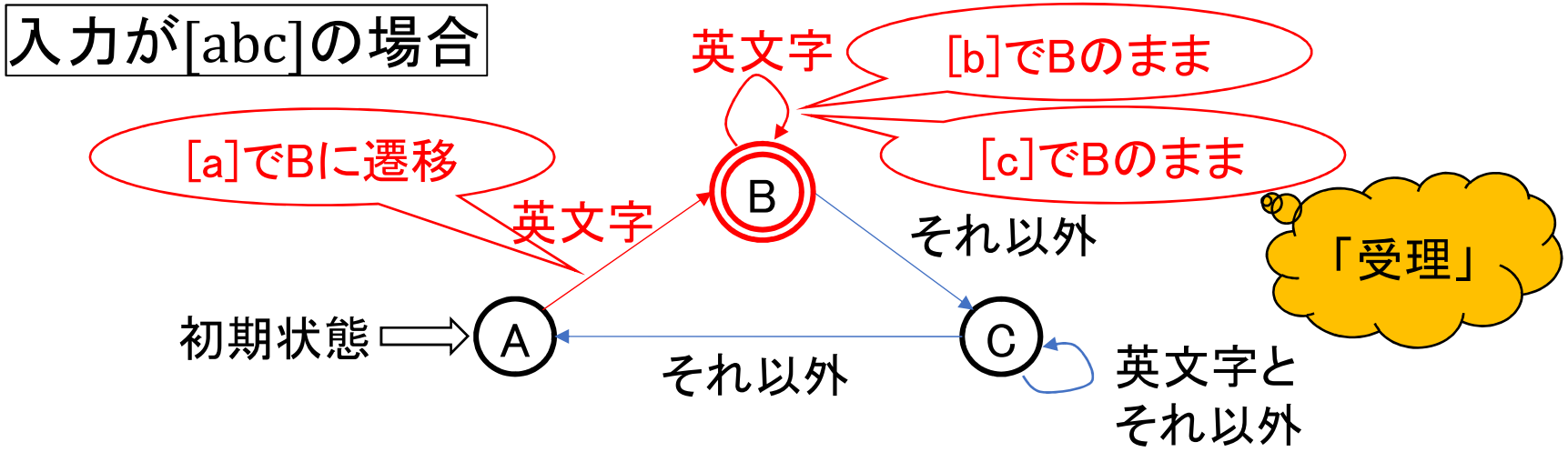
有限オートマトン

状態や遷移の数が、有限の場合のモデルを、有限オートマトンと呼び、オートマトンの代表的なモデル

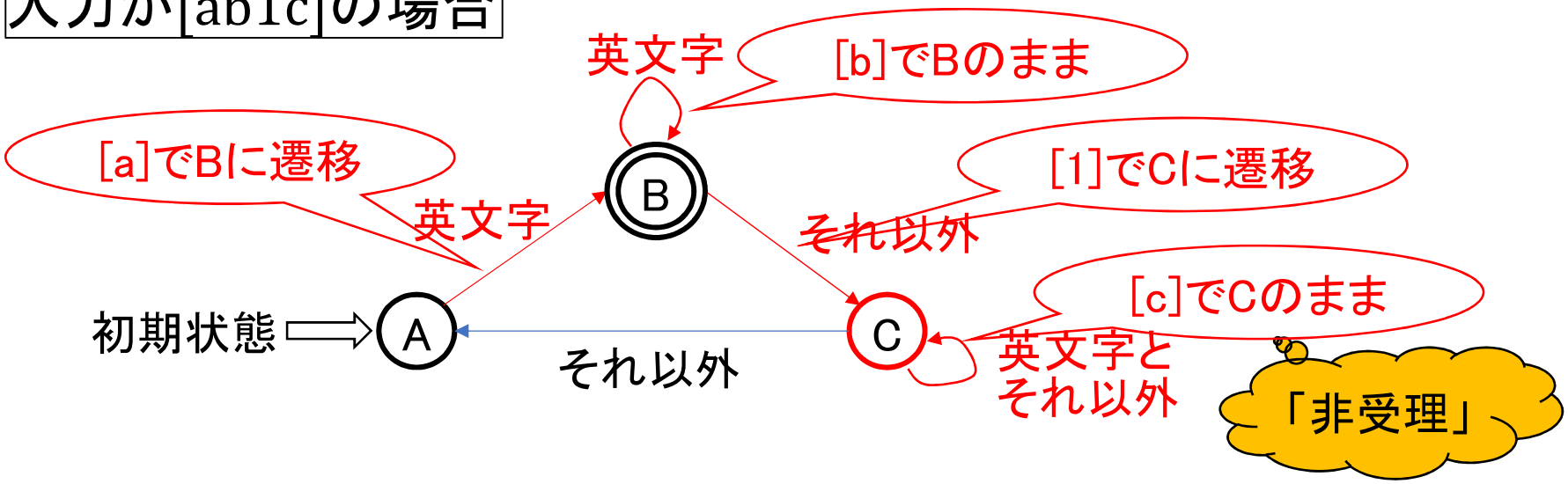


有限オートマトンでは、状態に「受理」状態と「非受理」状態があり、入力が終了した時点で、「非受理」状態となったときに、入力した値は不適切となる





入力が[ab1c]の場合


英文字
[b]でBのまま
[a]でBに遷移
英文字
B
[1]でCに遷移
それ以外
初期状態
A
C
[c]でCのまま
それ以外
英文字と
それ以外
「非受理」

1.2.2 計算量と正当性

- 計算量
  「計算の複雑さ」とも呼ばれ、アルゴリズムの評価の1つの尺度

- 領域計算量
  計算の実行開始から終了までに使用する記憶領域
- 時間計算量（通常は計算量と呼ぶ）
  計算の実行開始から終了までの所要時間

計算量の特徴

- 計算量の加算は、大きな計算量の方に引きずられる

$$O(f(n)) + O(g(n)) = O(\max(O(f(n)), O(g(n))))$$

計算f(n)の計算量　　計算g(n)の計算量

- 計算量の乗算は、２重ループのように考えればよい

$$O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$$

● 正当性

「計算が正しいことを証明すること」で、証明するレベルによって部分正当性と全正当性がある

- 部分正当性

入力条件を満たした計算は実行すると、停止した時点で、常に出力条件を満たしていること

- 全正当性

入力条件を満たした計算は実行すると、必ず停止し、このとき必ず出力条件を満たしていること

1.2.3 演算と精度

- 固定小数点表示と浮動小数点表示

固定小数点表示　0.00000015

浮動小数点表示　0.15×10^{-6}

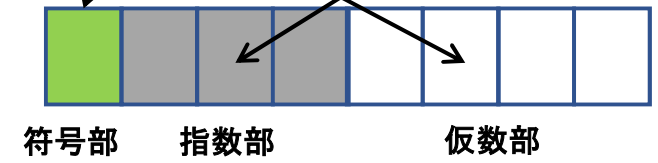
仮数　基数（コンピュータでは、通常2）　指数

- 固定小数点は、ゼロ”0”の数が増え（減）れば桁も増え（減）る
- 浮動小数点は、指数によって小数点の位置が動く
  数値を広範囲に少ない桁数で表現できる（コンピュータ向き）

浮動小数点の表記方法　：　± 仮数 × 基数^指数

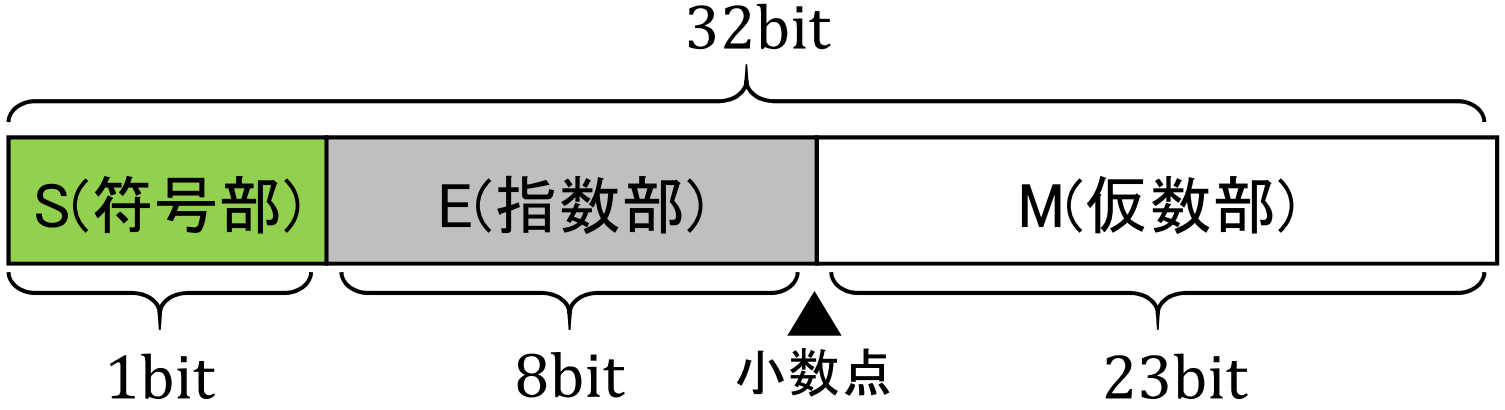
あらかじめ決められている（2または16）

ビット割り当て
（8ビットの場合）



● IEEE754

32ビットの浮動小数点数を、[符号部][指数部][仮数部]の形式で表現する



S：仮数部の符号(0:+,1:−)

E：指数部(2を基数として+127 → バイアス127)

M:仮数部(仮数−1の2進小数で表す → 1.????の小数部)

IEEE754では、

数値を$(-1)^S(2^{E-127})(1.M)$で表す

例1: $(1.5)_{10} = +1.5 \times 10^0$

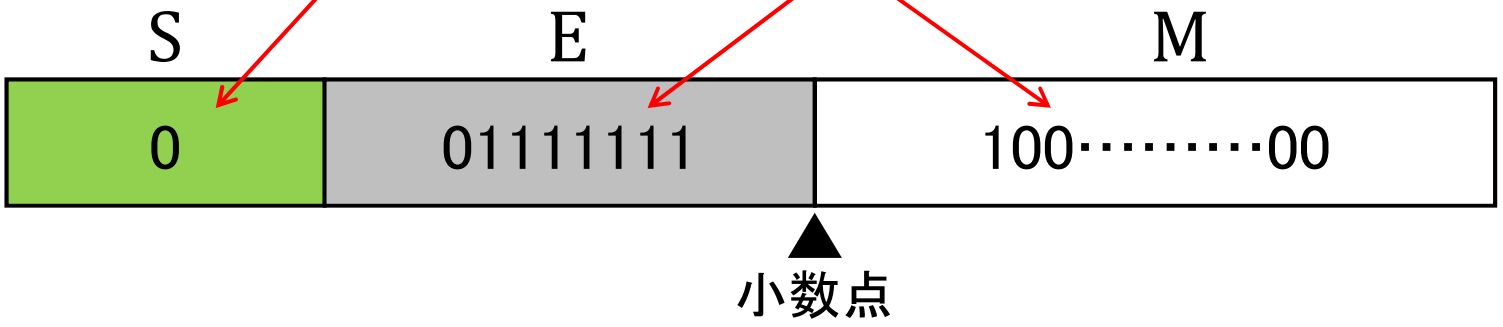
S=0

E=0+127=127

$(1.5)_{10} = (1.1)_2$

2進数に変換して正規化

M=1

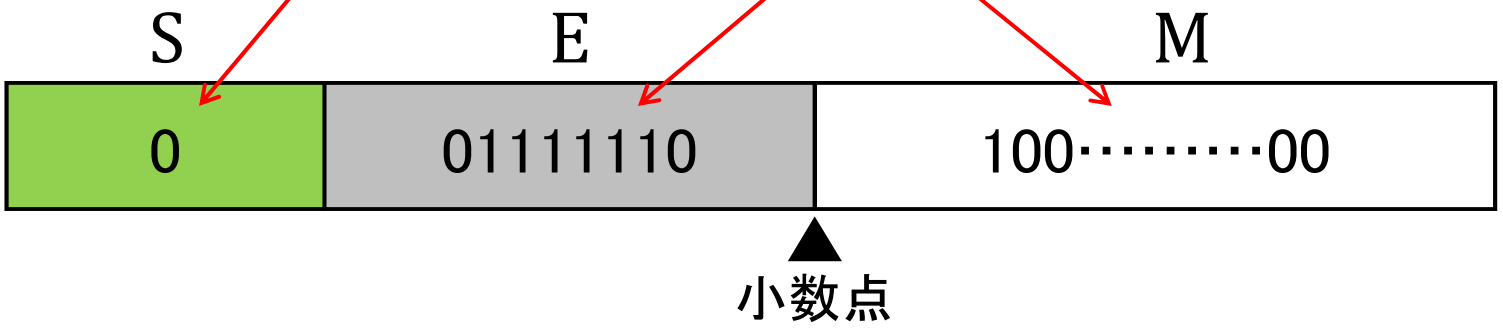


例2: $(0.75)_{10} = +7.5 \times 10^{-1}$

S=0　　E=−1+127=126

$(0.75)_{10} = (0.1)_2$　2進数に変換して正規化

M=1



● 誤差

実際の数値とコンピュータ内部で表現される数値との間に生じたズレを、誤差と呼ぶ

- 絶対誤差=｜測定値ー真値｜
- 相対誤差=$\frac{\text{絶対誤差}}{\text{真値}} \fallingdotseq \frac{\text{絶対誤差}}{\text{測定値}}$
- 系統誤差

何らかの原因で、測定値、観測値、計測値などが真値と比べて、同じ方向(増加方向など)に偏る誤差

● 誤差の発生する原因

● 丸め誤差

限られた桁数の範囲で数値を表現するときに、四捨五入、切り上げ、切り捨てなどの操作で起こる誤差

10進数：0.1 → 2進数：0.000110011 1・・・

入らない

小数点以下8桁

● 打ち切り誤差

計算を途中で打ち切るために起こる誤差

円周率3.1415・・・などのような数値に対して、あらかじめ決めた数値(3.14)で打ち切る

● 情報落ち

絶対値が大きな値と絶対値が小さな値の足し算や引き算したとき、小さな値の情報が無視されて、結果に反映されないために発生する誤差

仮数部が4桁の計算では、

$0.1234 \times 10^{4} + 0.4321 \times 10^{-4} = 0.1234 \times 10^{4} + 0.000000004321 \times 10^{4}$

$= 0.123400004321 \times 10^{4}$

指数を同じにして計算する

4桁の仮数部に入らない

- **桁落ち**

絶対数が等しい2つの数値で引き算をした時、有効桁数が減少するために起こる誤差

$1.234 \times 10^{-1} - 1.233 \times 10^{-1} = 0.100 \times 10^{-3}$

有効桁数が 4 桁

有効桁数が 1 桁

信頼性がない値

- **桁あふれ**

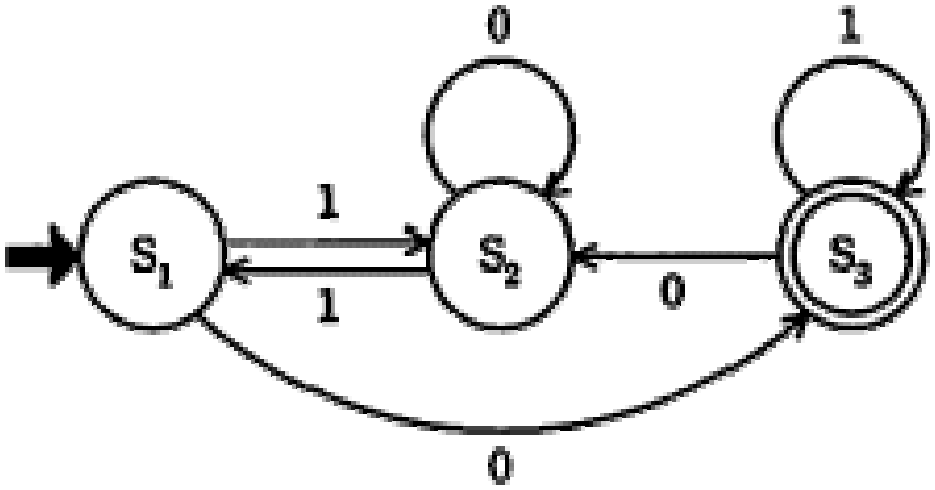
計算結果が、コンピュータで扱うことができるビット数を超えたことにより起こる誤差

ビット数の最大値を超えた場合　→　オーバーフロー

ビット数の最小値を下回った場合　→　アンダーフロー

■ 過去問題2

次に示す有限オートマトンが受理する入力例はどれか。ここで，S_1は初期状態を，S_3は受理状態を表している。

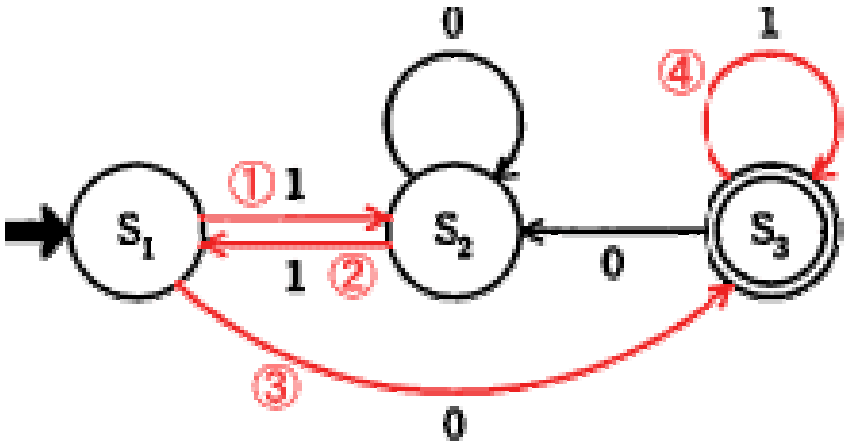


ア: 1011　　イ: 1100　　ウ: 1101　　エ: 1110

有限オートマトンは、入力値と入力されたときの状態で出力値が決まる順序機械に言語を識別するアルゴリズムを与えた数学的モデルである」

この有限オートマトンを文章で表すと、
1. S1状態から始まる
2. S1状態で0を出力した時はS3へ、1を出力した時はS2へ状態遷移する
3. S2状態で0を出力した時はS2を維持、1を出力した時はS1へ状態遷移する
4. S3状態で0を出力した時はS2へ状態遷移し、1を出力した時はS3を維持する
5. 入力列はS3で受理される

選択肢の中で唯一S3が受理可能な入力列は「1101」で出力順序は以下の通りです



1.3 数理応用

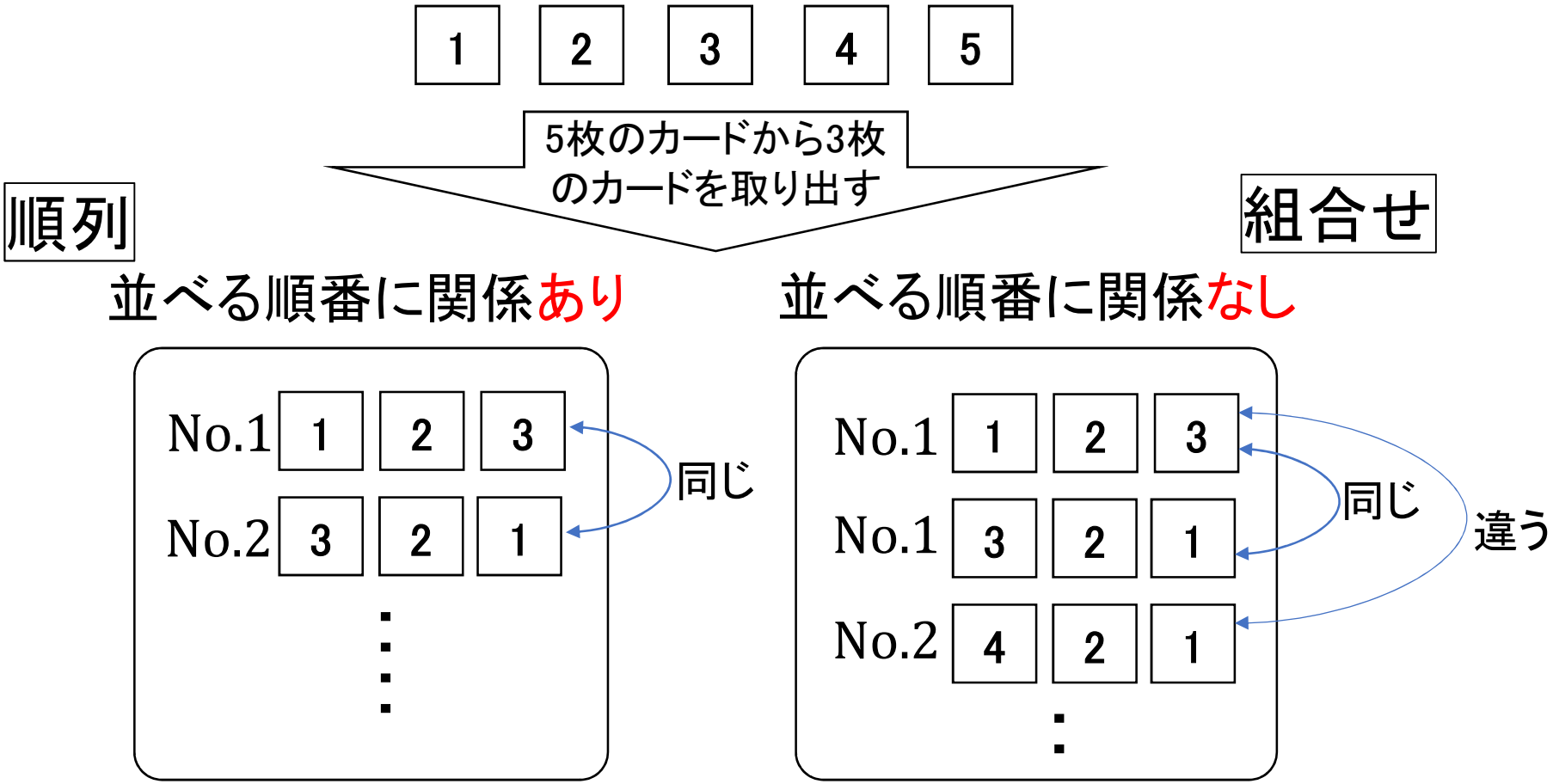
1.3.1 確率と確率分布

1.3.2 PERT

1.3.3 待ち行列

1.3.1 確率と確率分布

● 順列と組合せ



- 順列

  n個の中からr個($r \leqq n$)を選んで並べる並べ方${}_nP_r$

$$ {}_nP_r = \frac{n!}{(n-r)!} \text{（通り）} $$

1～5の数字を3つ選んで並べる並べ方は、何通り？

$$ {}_5P_3 = \frac{5!}{(5-3)!} = \frac{5\times4\times3\times2\times1}{2\times1} = \frac{120}{2} = 60\text{(通り)} $$

- 組合せ

  n個の中からr個($r \leqq n$)を選ぶ組合せ${}_nC_r$

$$ {}_nC_r = \frac{{}_nP_r}{r!} = \frac{n!}{r! \times (n-r)!} \text{（通り）} $$

1～5の数字を3つ選ぶ組合せは、何通り？

$$ {}_nC_r = \frac{{}_5P_3}{3!} = \frac{60}{3\times2\times1} = 10\text{(通り)} $$

●確率

1つの事象の起こりうる可能性を表した数
全事象の数をn、事象Aの数をn(A)とすると、このときの確率P(A)は、

$$P(A) = \frac{\text{事象}A\text{の数}}{\text{全事象の数}} = \frac{n(A)}{n}$$

「1の目が出る」を事象Aとすると

全事象  6通り

事象A  1通り

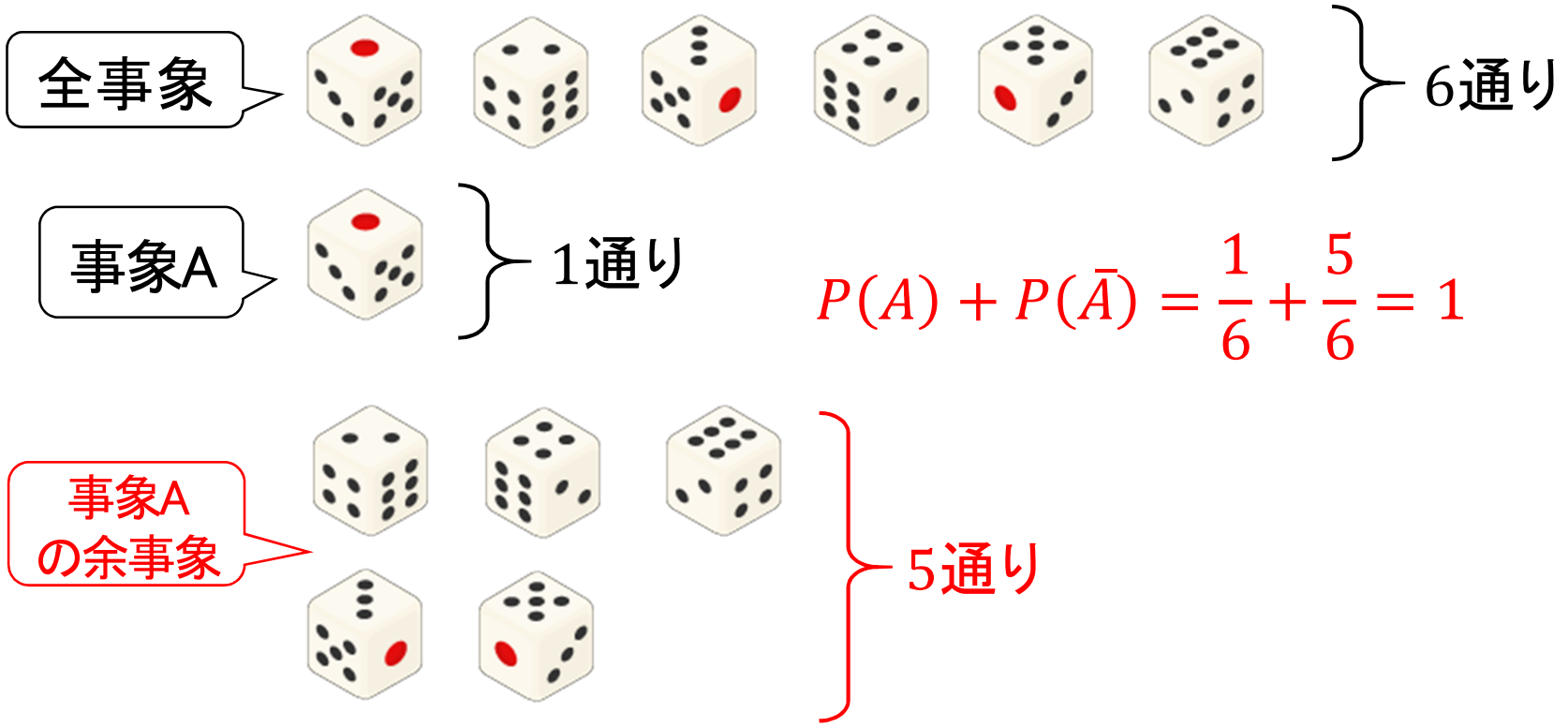
$$P(A) = \frac{1}{6}$$

- **余事象**

事象Aでない事象で、$\overline{A}$と表す

$$P(A) + P(\bar{A}) = 1$$

事象A「1の目が出る」の余事象



- **排反事象**

2つの事象のうち、一方が発生すれば、もう一方は発生しない事象があれば、これらを排反事象と呼ぶ

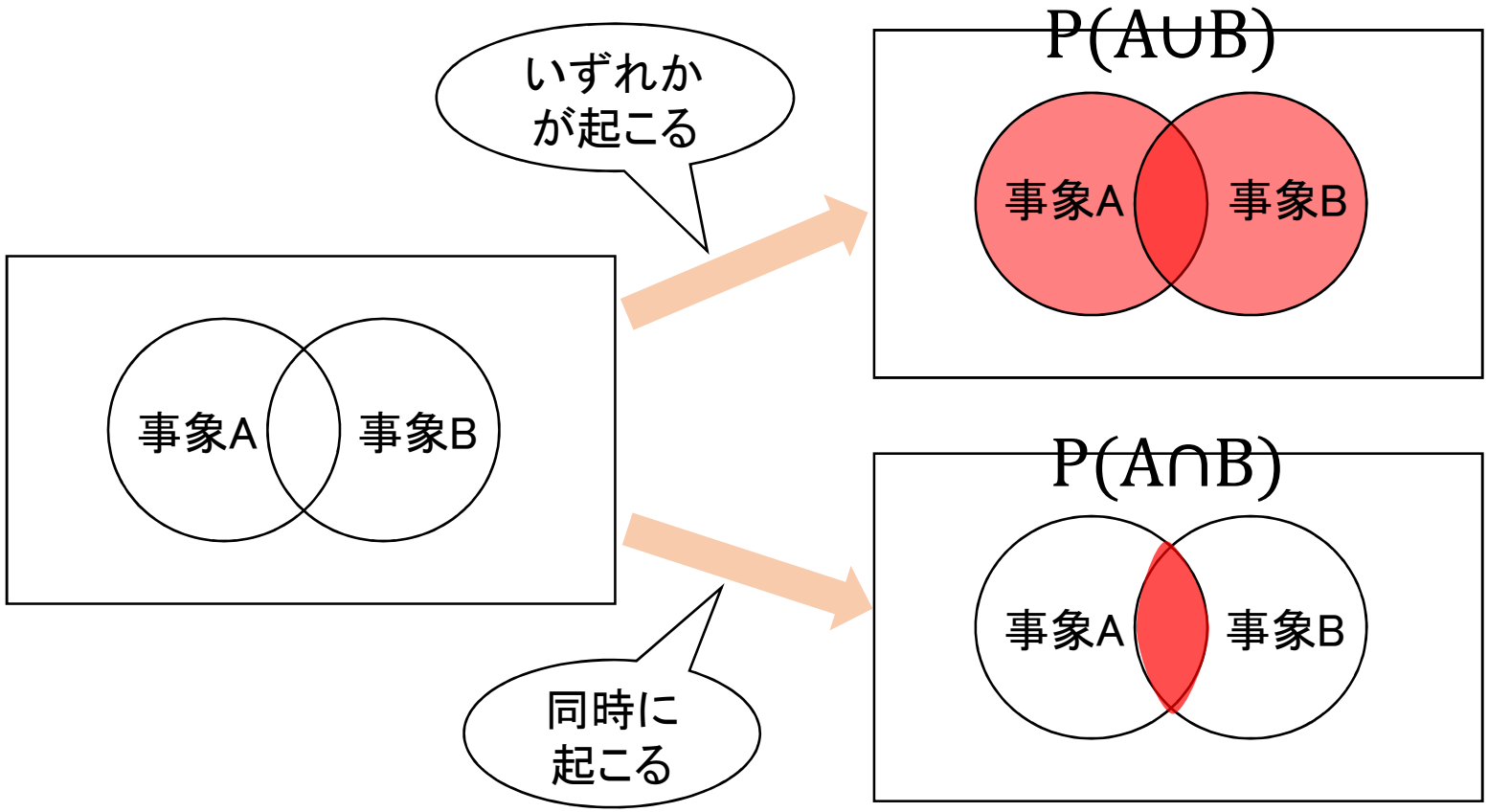
事象A「1・3・5(奇数)の目が出る」

事象B「2・4・6(偶数)の目が出る」

同時に起こることはない（排反事象）

- **確率の加法定理と乗法定理**

2つの事象のうち、そのいずれかが起こる確率を加法定理、どちらも同時に起こる確率を乗法定理と呼ぶ

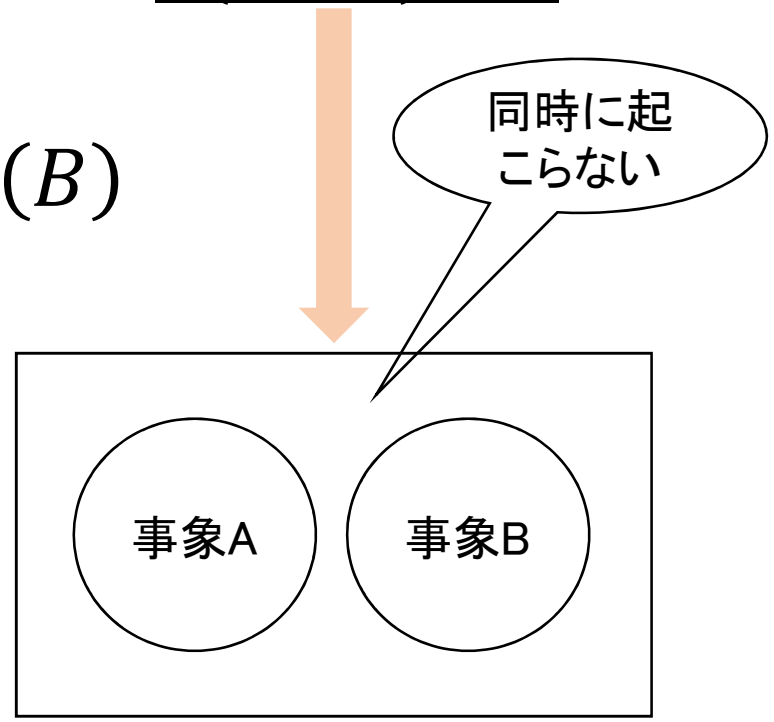


✓ 確率の加法定理

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

但し、事象Aと事象Bが排反事象の場合、$P(A \cap B) = 0$なので、

$$P(A \cup B) = P(A) + P(B)$$



✓ 確率の乗法定理

$$P(A \cap B) = P(A) \cdot P(B)$$

2つのサイコロを同時に振って、両方のサイコロに、1の目が出る確率を求める

$$P(A \cap B) = \frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$$

1.3.2 PERT
(Program Evaluation and Review Technique)

プロジェクトを管理する1方法で、プロジェクトの日程や要員、資源割り当てに使用

- 前進計算
  - プロジェクトの所要日数の決定に使用
  - 最早結合点時刻（プロジェクトを最も早く開始できる日程）を求める

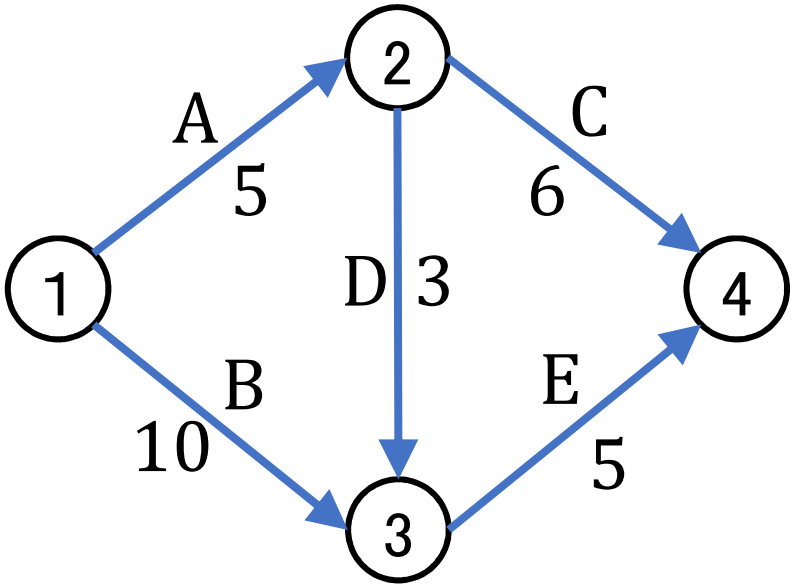
いつから次のプロジェクトに取りかかれるか？

- 後進計算
  - 最遅結合点時刻（プロジェクトを最も遅く開始できる日程）を求める

いつまでにプロジェクトを開始しないといけないか？

アローダイヤグラム

プロジェクトの流れと所要日程を、分かりやすく図に表したもの



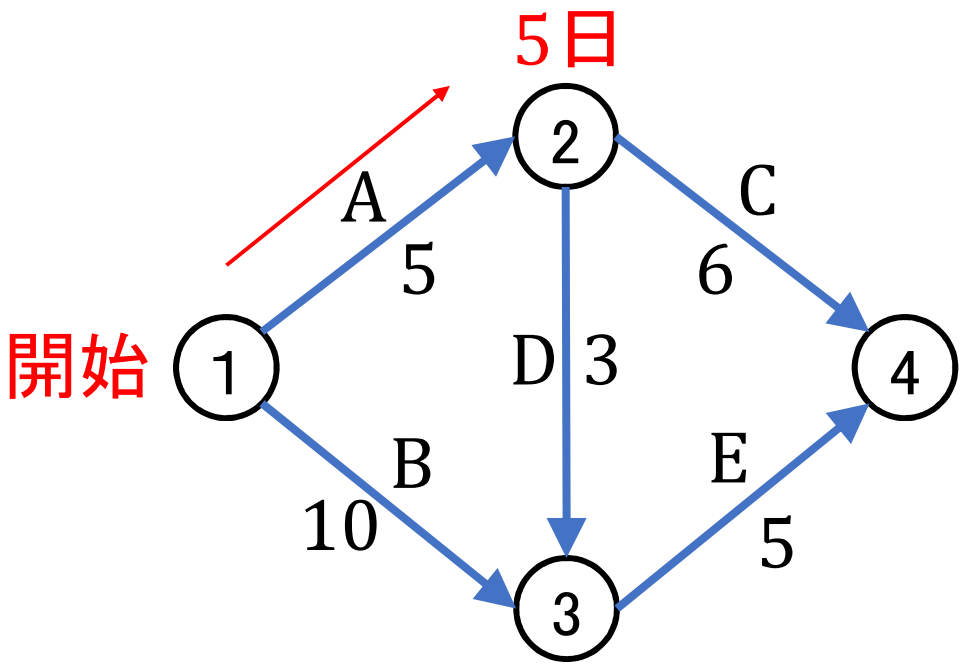
- プロジェクトの開始と終了（結合点）
- 数字は先頭からの順番

- 矢印はプロジェクトの流れ
- 記号はプロジェクト名、数字は日数

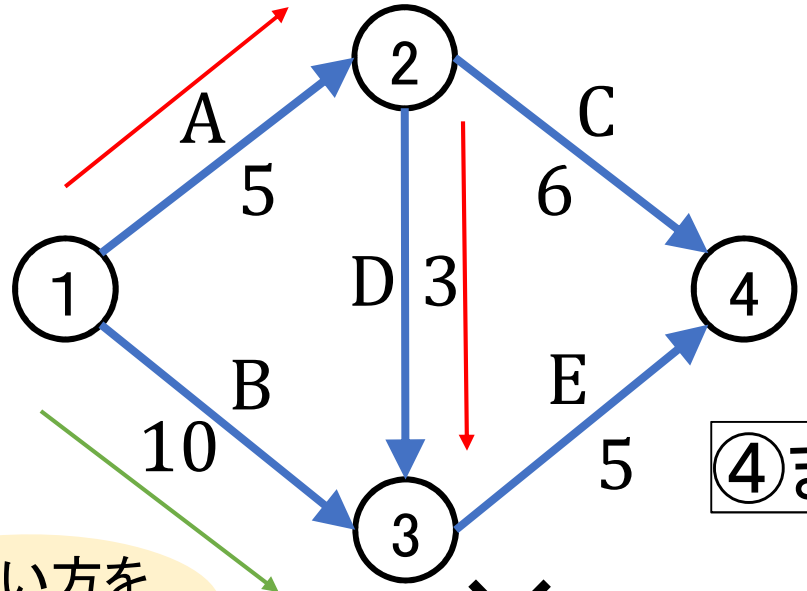
● 前進計算

プロジェクトを①から順次、②～④へと最早結合点時刻を求めて行く方法

②までの最早結合点時刻



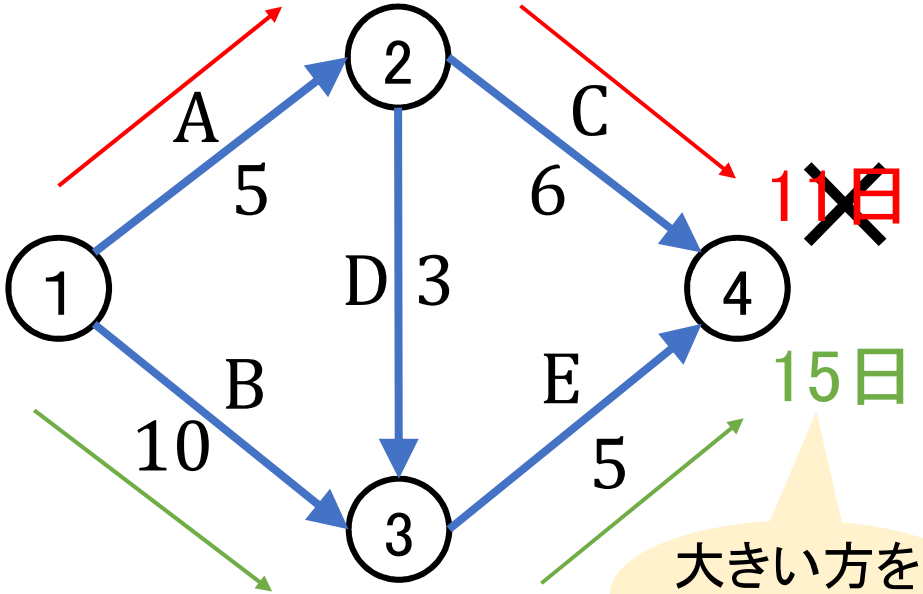
③までの最早結合点時刻



大きい方を
採用する

10日　~~8日~~

④までの最早結合点時刻

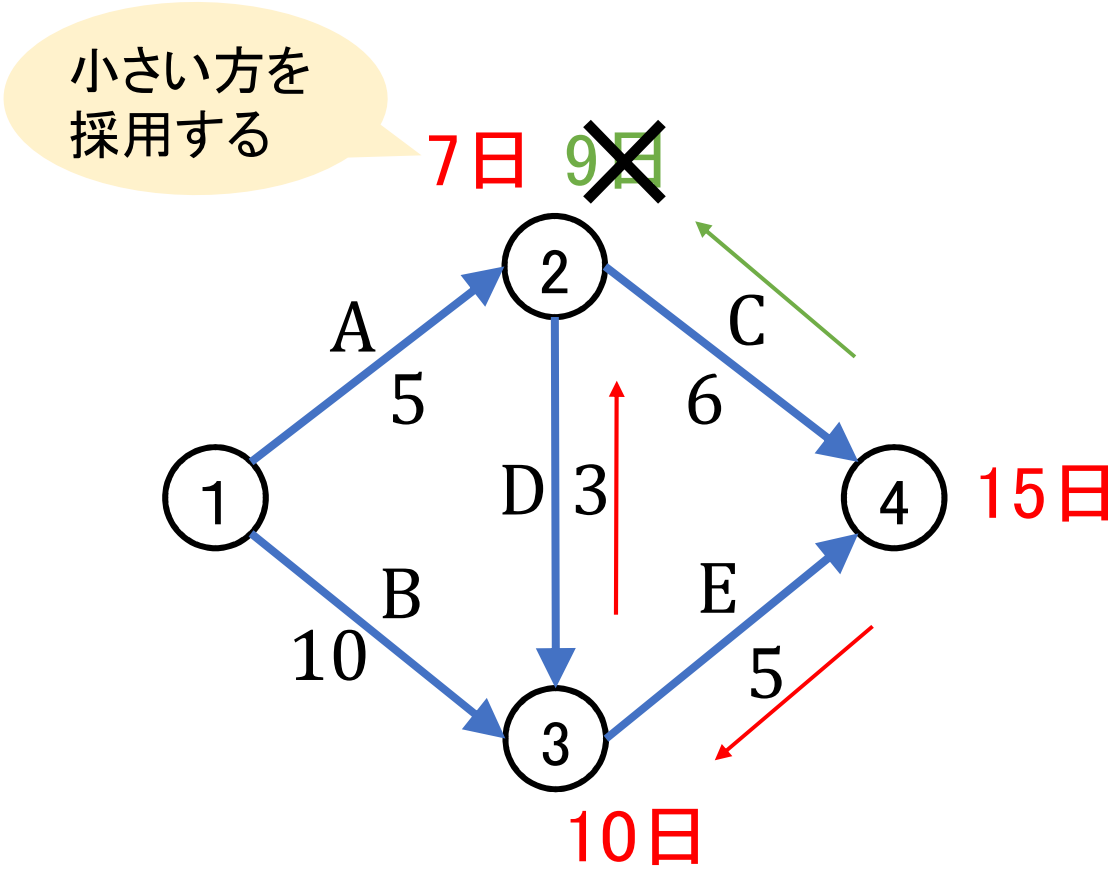


大きい方を
採用する

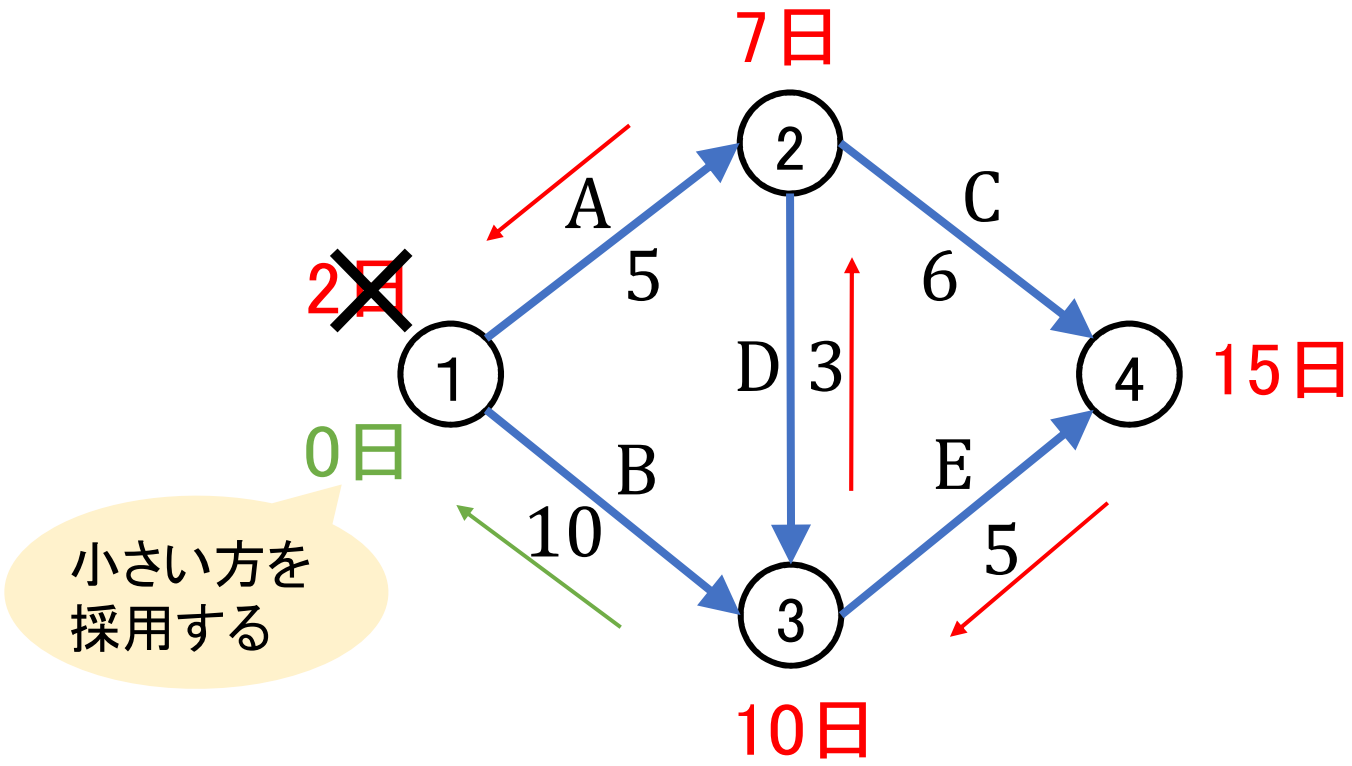
● 後進計算

プロジェクトを④から最遅結合点時刻を求めて行く方法

②までの最遅結合点時刻

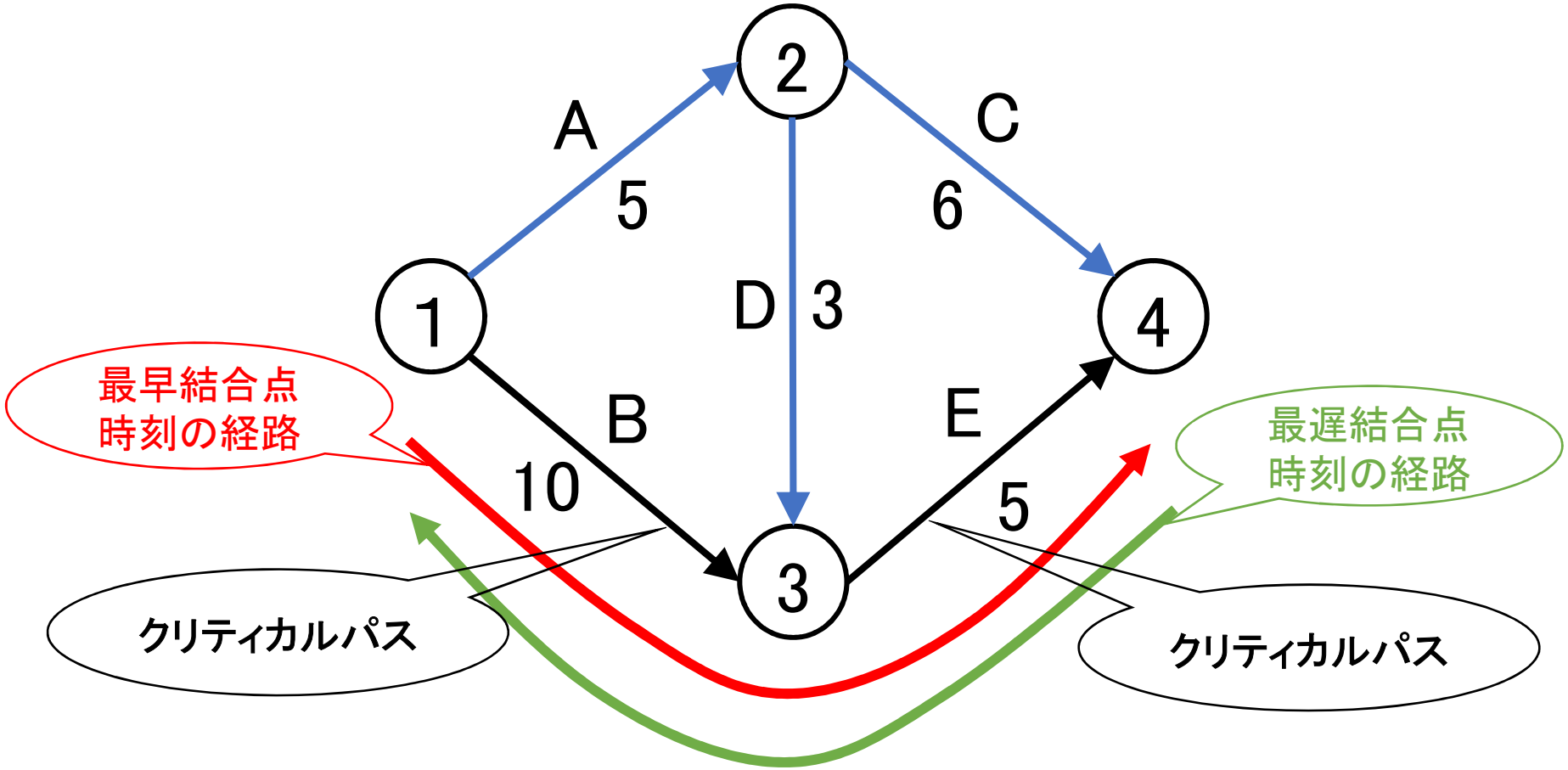


①までの最遅結合点時刻



クリティカルパス

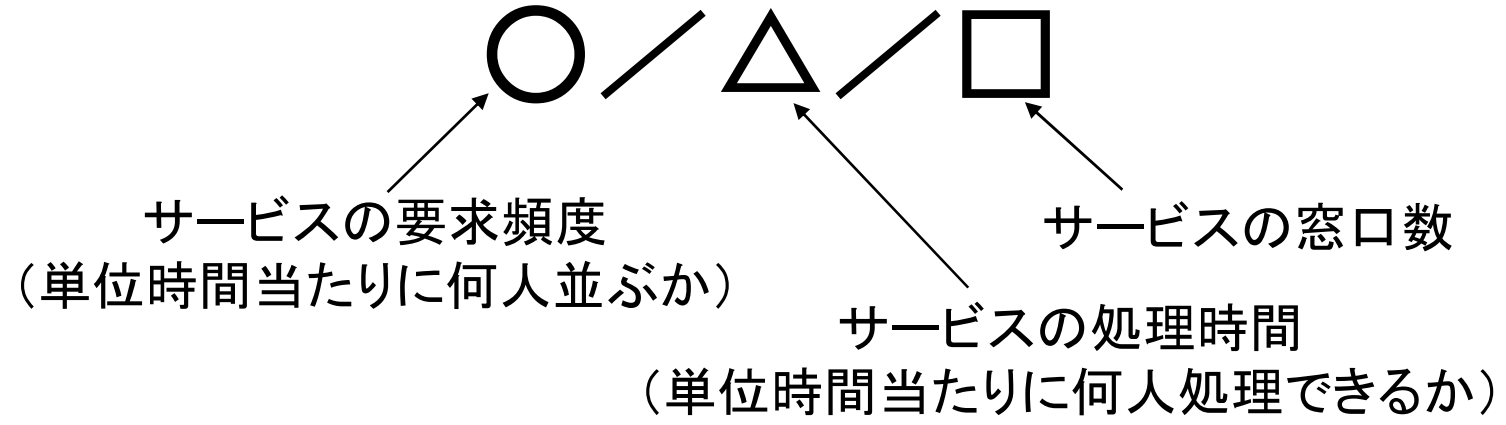
最早結合点時刻と最遅結合点時刻の等しい経路



1.3.3 待ち行列

- ケンドールの記法

  平均待ち時間を求めるための待ち行列の状態を表す



システム設計でよく使うのは、M／M／1

サービスの要求頻度=ランダム

サービスの処理時間=指数分布

サービスの窓口数=1

- 待ち行列の公式

[M/M/1]モデルで必要な公式

$$\rho = \lambda / \mu$$

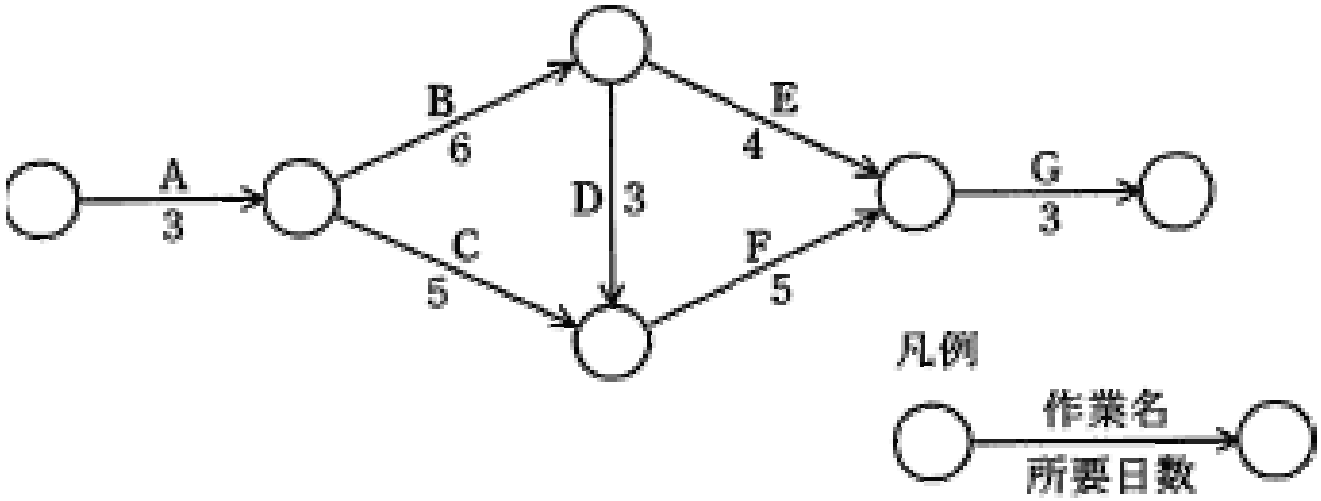
利用率（サービスの窓口が処理中である確率）

平均到着率（単位時間当たりの平均到着数）

平均処理率（単位時間当たりの平均処理数）

■ 過去問題3

下図のプロジェクトを最短の日数で完了したいとき、作業Eの最遅開始日は何日目か？



ア: 9　　イ: 12　　ウ 13　　エ:17

アローダイアグラムの問題ですが、クリティカルパスから一歩進んで**最遅開始日**を求める問題である

しかし、まずはクリティカルパスを求める必要がある。ルートは、

- A→B→E→G（3+6+4+3=16日）
- A→B→D→F→G（3+6+3+5+3=20日）
- A→C→F→G（3+5+5+3=16日）

の3通りしかないので、**クリティカルパスはA→B→D→F→G（20日）**だと、すぐにわかる。

作業Eと作業Fの結合点に着目すると、クリティカルパス上では作業開始から17日目にあたることが分かる。つまり作業Eは、最低でもその4日前の13日目までに開始すれば、プロジェクトの最短完了日数を守ることができる。つまり作業Eの最遅開始日は、13日目となる。

1.4 プログラム言語

1.4.1 プログラム構造と基本制御構造
1.4.2 コンパイル技法
1.4.3 プログラム言語と種類・特徴

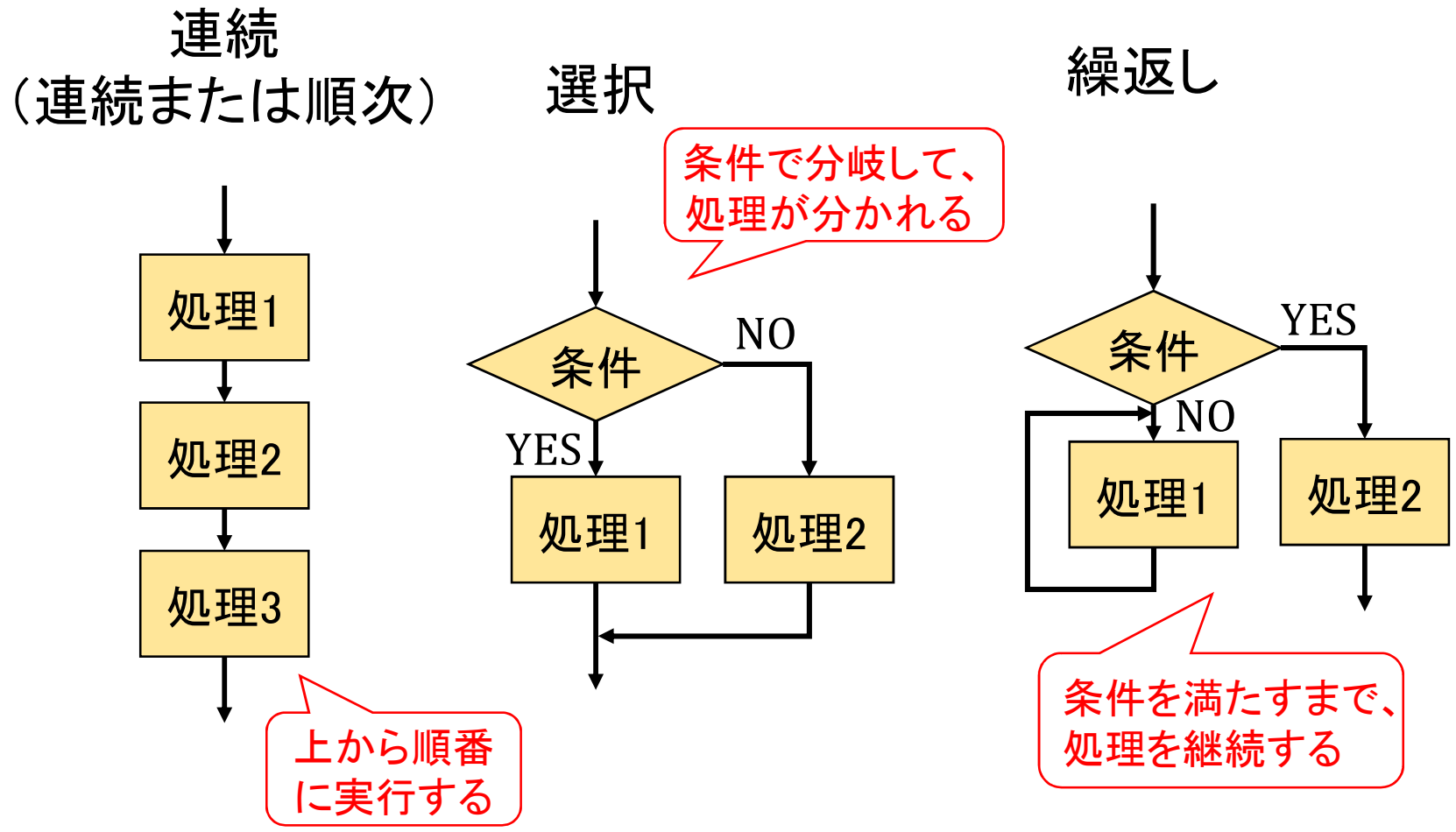
1.4.1 プログラム構造と基本制御構造

● プログラム構造

- 再帰（リカーシブ）
  プログラム実行中に、実行中の処理内容を呼び出すことができる
- 再使用可能（リユーザブル）
  主記憶に再度読み出さなくとも、複数のタスクから使用できる
  - ✓ 再入可能（リエントラント）：同時に複数のプログラムで使用できる
  - ✓ 逐次再使用可能：同時に使用できない
- 再配置可能（リロケータブル）
  プログラム実行中に、主記憶の保存位置を変更できる

● 基本制御構造

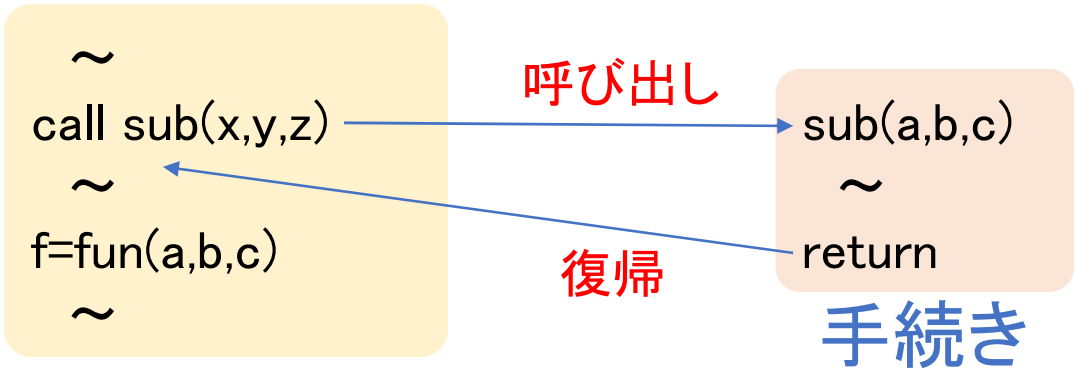
プログラムの流れには、以下の3つの構造がある



● 手続きと関数

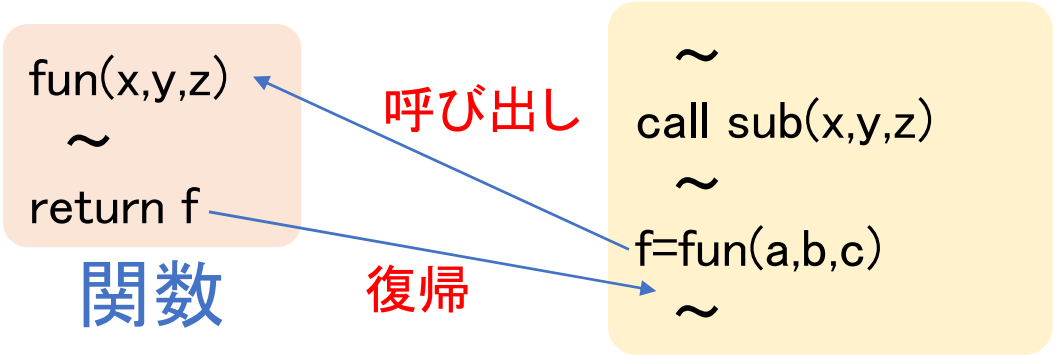
- 手続き

プログラムの中で、呼び出しと復帰の機能がある部分



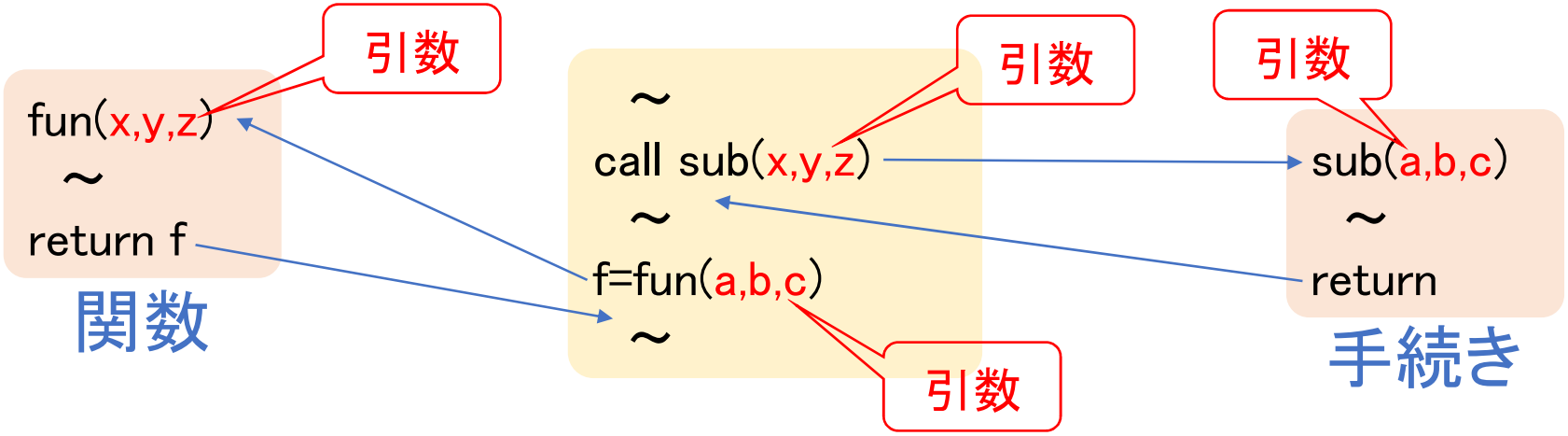
- 関数

手続きでの値を受けて、結果を戻す機能のある部分



- 引数

手続きや関数とデータをやり取りするために使用する変数



- 引数の使い方には、2種類がある

値呼び出し	サブプログラム内や関数内で、引数の値を変更しても、呼び出し元での値は変わらない
参照呼出し	サブプログラム内や関数内で、引数の値を変更すると、呼び出し元での値が変わる

1.4.2 コンパイル技法

```
/**
 * Simple HelloButton() method.
 * @version 1.0
 * @author john doe <doe.j@example.com>
 */
HelloButton()
{
  JButton hello = new JButton( "Hello, wor
  hello.addActionListener( new HelloBtnList

  // use the JFrame type until support for t
  // new component is finished
  JFrame frame = new JFrame( "Hello Button"
  Container pane = frame.getContentPane();
  pane.add( hello );
  frame.pack();
  frame.show();              // display the fra
}
```

ソースコード
（C言語,Fortranなど）

コンパイラ

ソースコードを機械語に変換する

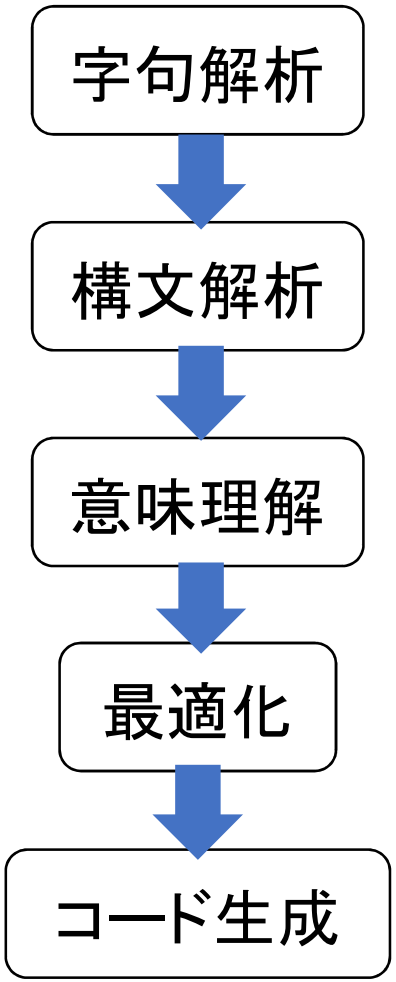
```
A 002000  C2 30       REP #$30
A 002002  18          CLC
A 002003  F8          SED
A 002004  A9 34 12    LDA #$1234
A 002007  69 21 43    ADC #$4321
A 00200A  8F 03 7F 01 STA $017F03
A 00200E  D8          CLD
A 00200F  E2 30       SEP #$30
A 002011  00          BRK
A 2012

r
  PB  PC   NVmxDIZC  .A   .X   .Y   SP   DP  DB
; 00 E012  00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

  PB  PC   NVmxDIZC  .A   .X   .Y   SP   DP  DB
; 00 2013  00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00:UU..............
```

機械語

人間が理解できる

CPUが理解できる

● コンパイラの処理手順



読み込んだプログラムを字句（単語）ごとに切り出し、それらがどんな意味（変数名か演算子かなど）をもつのかを決定する

意味を決定した字句を、構文を木構造で表現した、構文木を元に構文に当てはめていく。演算は、項で演算子を挟むという性質上、二分木の構造で表現できる。

構文解析で判別した、各単語や演算子の意味を正確に理解する。（例えば、整数型の演算子）

命令ごとに変わる待ち時間を、順番を変えるなどして効率のよいプログラムに変更する

CPUが理解できる機械語に変換する

● 最適化

- 畳み込み
  定数を予め計算しておく

$$x = 60 * 60 \quad \Rightarrow \quad x = 3600$$

- 式の簡略化
  同じ式は、置換えて簡略化する

$$\begin{cases} x = p + q \\ y = (p + q) * 3 \end{cases} \quad \Rightarrow \quad \begin{cases} x = p + q \\ y = x * 3 \end{cases}$$

- ループの再編
  - ✓ ループ内で値の変わらない定数を、ループ外へ出す
  - ✓ 多重ループを一重ループにする

● 逆ポーランド記法

計算式を機械語に変換する（コンパイルする）際に、変換しやすいような表現形式に変更する

基本変換方法：演算対象の2つの変数の後ろに演算子を置く

あとは以下の計算技法を使って、計算式をまとめる

① 途中計算式の最小単位を別の文字に置換える
② 計算式全体が最小単位になったら、置換えた文字をすべて戻す

問：計算式X=(A+B)×Cを逆ポーランド記法で表せ

X=(A+B)×C → X=AB+×C

AB+をPに置き換える X=P×C

X=P×C → X=PC×

PC×をQに置き換える X=Q

X=Q → XQ=

QをPC×に戻す

PをAB+に戻す

PとQを元の文字に戻す

XQ= → XPC×= → XAB+C×=

逆ポーランド記法

問：逆ポーランド記法のXAB+C×=を通常の計算式に直せ

① 式の左側から右側に向かって見て、最初の演算子を探す

$\overrightarrow{XAB+}C\times=$

② 探した演算子+の直前の2つの文字（変数）を演算子+で結ぶ

$XAB+C\times= \rightarrow XA+BC\times=$

③ A+BをPに置き換える　$XPC\times=$

④ 式の左側から右側に向かって見て、最初の演算子を探す

$\overrightarrow{XPC}\times=$

⑤ 探した演算子×の直前の2つの文字（変数）を演算子×で結ぶ

XPC×= → XP×C=

⑥ P×CをQに置き換える XQ=

⑦ 式の左側から右側に向かって見て、最初の演算子を探す

XQ=

⑧ 探した演算子×の直前の2つの文字（変数）を演算子×で結ぶ

X=Q

⑧ PとQを元の文字に戻す

QをP×Cに戻す　PをA+Bに戻す　通常の計算式

X=Q → X=P×C → X=A+B×C

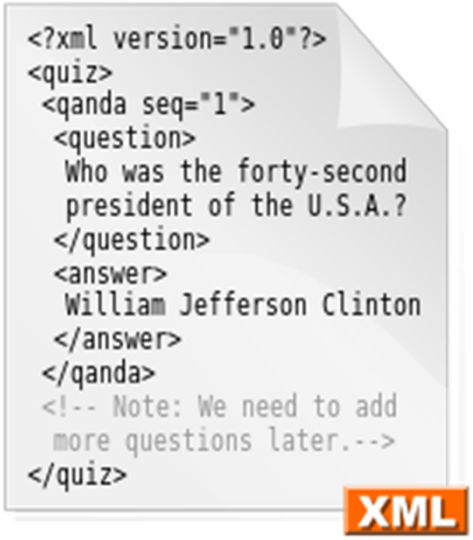
1.4.3 プログラム言語の種類・特徴

- プログラム言語の分類と特徴
  - 手続き型
    問題の解決手順をアルゴリズムとして記述する
    （C言語,COBOL,Fortran,BASICなど）
  - 関数型
    関数定義と関数呼び出しで構成し、データをリストにして処理する（LISP,APLなど）
  - 論理型
    データの関係を論理式で定義し、関係を証明しながら推論を繰り返して処理する（Prologなど）
  - オブジェクト指向型
    データとアルゴリズムをカプセル化したオブジェクトによって処理する(Java,C++,Smalltalkなど）

- マークアップ言語

テキスト(文字)データ中に、特定の記号で囲まれたタグ(tag)と呼ばれる表記を用いて、構造や見栄えなどを記述するもの。また、バイナリデータ中に埋め込むものなど、様々な種類がある。

マークアップ言語を用いることで、文字列の一部を見出しに指定したり、段落の開始や終了を指示したり、文字色などの見栄えを設定したり、といった情報をデータ中に埋め込んで、文書データとすることができる。



マークアップ言語の特徴

言 語 名	特 徴
SGML (Standard Generalized Markup Language)	文書の構造やデータの意味などを、タグを使って記述する。大量の文書データの記述に適している。
HTML (HyperText Markup Language)	文章の構造や修飾、表示の仕方についての情報を文書に埋め込んで記述する。Webページの記述に適している。
XML (eXtensible Markup Language)	SGMLをインターネット用に最適化したもので、タグをユーザ自身で定義できる

● スクリプト言語

OSやアプリケーションソフトの動作や機能などをプログラムの形で記述できる。また、この言語は、実行可能形式への変換作業などを省略・自動化したり、少ない記述量でも実行できるなど、仕様や開発手順が簡略化されている。

スクリプト言語の特徴(1/2)

言 語 名	特 徴
Visual Basic	「フォーム」と呼ばれるウィンドウにアプリケーションソフトの構成要素となる部品(ActiveXコントロール)を張り付け、部品の設定や部品間の関係を指定することでアプリケーションソフトを開発することができる。アプリケーションソフトが容易に開発できるように工夫された独特の**開発環境**と共に提供されている。
Java Script	Webページに組み込まれたプログラムを、Webブラウザ上で実行することができる。実行環境をWebブラウザに組み込んで利用することが多い。

スクリプト言語の特徴(2/2)

言 語 名	特 徴
Perl (Practical Extraction and Report Language)	テキスト処理やファイル処理に適した言語。ブラウザに内蔵したインタプリタで実行できる。
PHP (Hypertext Preprocessor)	Webサーバの機能を拡張し、動的にWebページを生成するために用いられる。実行環境をWebサーバに組み込んで、利用されることが多い。

● 代表的なプログラミング言語の特徴

- COBOL(Common Business Oriented Language)

会計処理や事務処理に適したプログラミング言語。1960年代から使われている言語で、現在でも、長年使われている企業の会計システムなどで広く利用されている。

- C(言語)

汎用的な言語で様々な分野で広く利用されているが、特にハードウェアを直接制御するプログラムの開発で利用される機会が多い

- Pascal(Philips' Automatic Sequence Calculator)

ALGOLを基本とし、構造化プログラミングに適した制御構造を導入、さらにプログラミングに必要なさまざまなデータ構造を定義できるように豊富なデータ構造を提供している

- Java

オペレーティングシステムやマイクロプロセッサーの種類にかかわらず、Java仮想マシンというソフトウェア上であれば実行できる。インターネットの普及に伴い、ウェブブラウザーで動作する「Javaアプレット」というアプリケーションプログラムが広く利用されている。

- C++

C言語にオブジェクト指向関連の仕様などを追加した言語で、C言語の仕様を引き継いだ上位互換であり、ほとんどのCプログラムはそのままC++プログラムとしても有効である。高度で抽象的な機能と低水準(ハードウェア寄り)な機能を兼ね備えた汎用的なプログラミング言語として、様々な用途で広く普及している。

- PostScript

Adobe Systems社が開発したページ記述言語。高解像度でも美しく印刷できるデータを少ないデータ量で表現でき、DTP業務に用いるプリンタやソフトウェアなどを中心に普及した。

■ 過去問題4

式A＋B×Cの逆ポーランド表記法による表現として、適切なものはどれか？

ア:＋×CBA　イ:×＋ABC　ウ:ABC×＋　エ:CBA＋×

通常の式を、**逆ポーランド表記法**(後置表記法)で表現するのは、A＋B　を　AB＋**が基本**。1回変換した部分は1つの項とみなすことに注意して、普通に計算式を解くのと同じ順番で変換ですることで、逆ポーランド表記法の式になる。優先順位は、通常の計算式と同様に括弧付き“()”→積商算“×÷”→和差算“”＋－“　の順番。

最初に、B×Cの部分を変換する

A＋B×C → A＋BC×

次に、BC×を1つの項とみなしてAとの＋演算部分を変換する

A＋BC× → ABC×＋

1.5 アルゴリズム

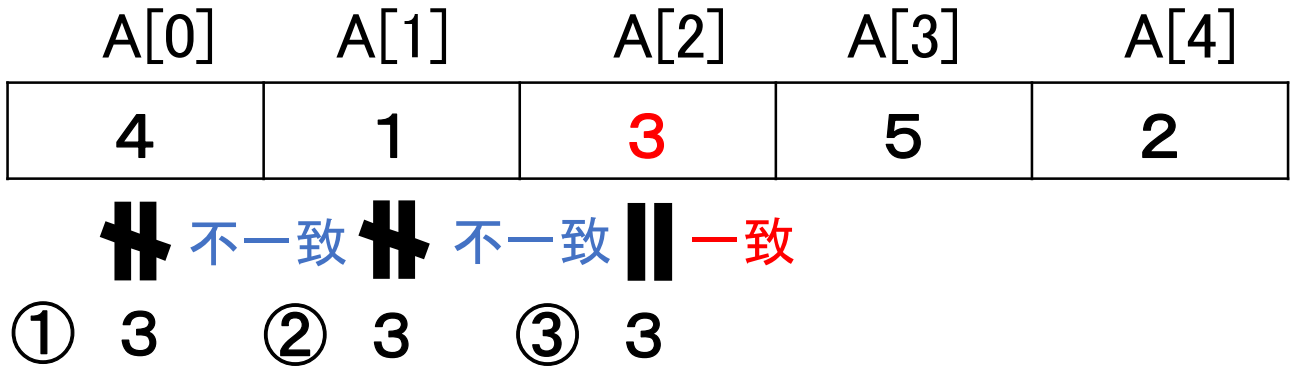
1.5.1 探索アルゴリズム
1.5.2 整列アルゴリズム

1.5.1 探索アルゴリズム
（目的のデータを見つける方法）

● **線形探索**

先頭の配列データから順番に目的のデータと比較しながら、合致しているかを調べる

※目的のデータが、3の場合



平均比較（探索）数 =（ｎ＋１）／２

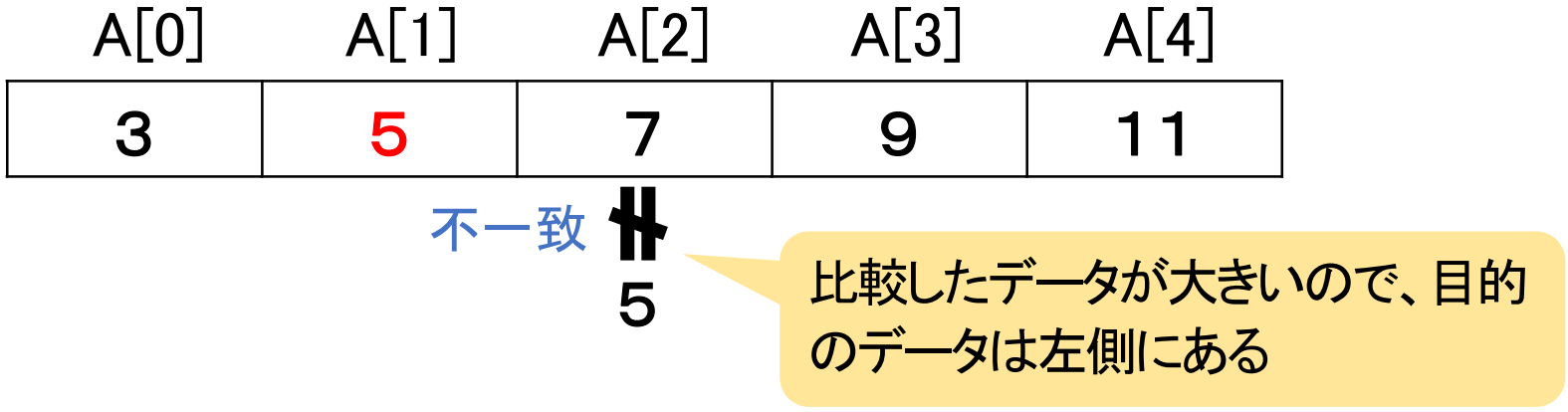
ｎ = 最大比較数（配列数）

● 2分探索

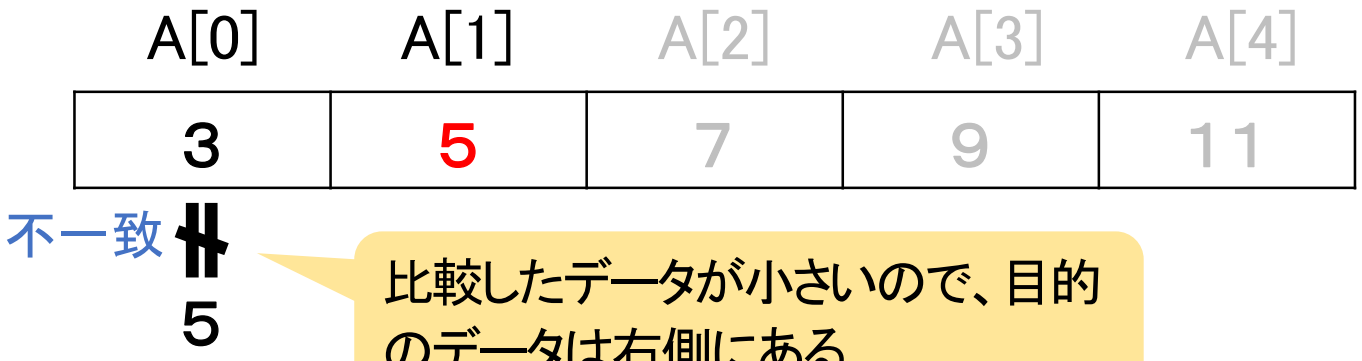
あらかじめ配列データが[昇順]や[降順]に並べられている場合、配列の中央に位置するデータと目的のデータを順次比較する。比較の結果、配列データの半分が切り捨てられ、残った半分で同様の比較を続ける。

※目的のデータが5で、小さい順に並べられている場合

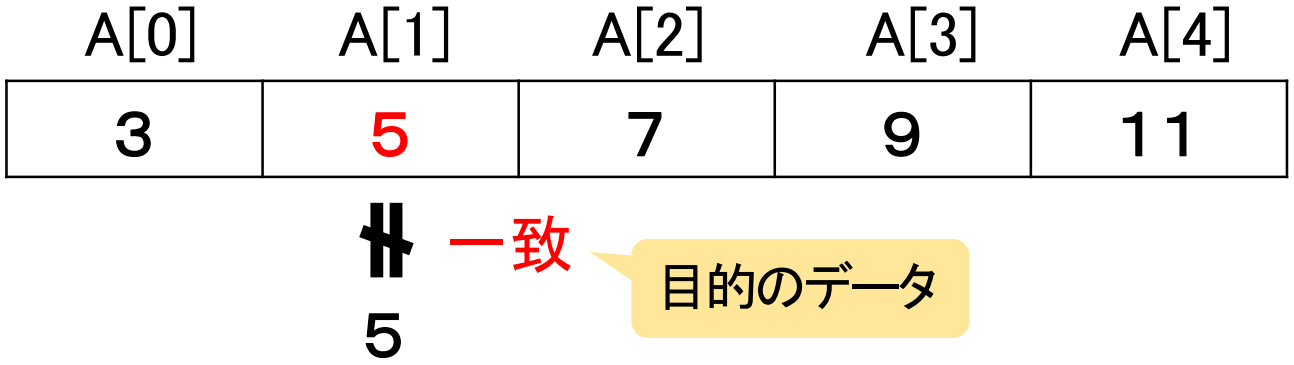
①検索データの真ん中に位置するデータと比較する



②検索位置を左半分に移して、再度真ん中の配列データと比較する



③検索位置を右半分に移して、再度真ん中に位置するデータと比較した結果、目的のデータを見つけた



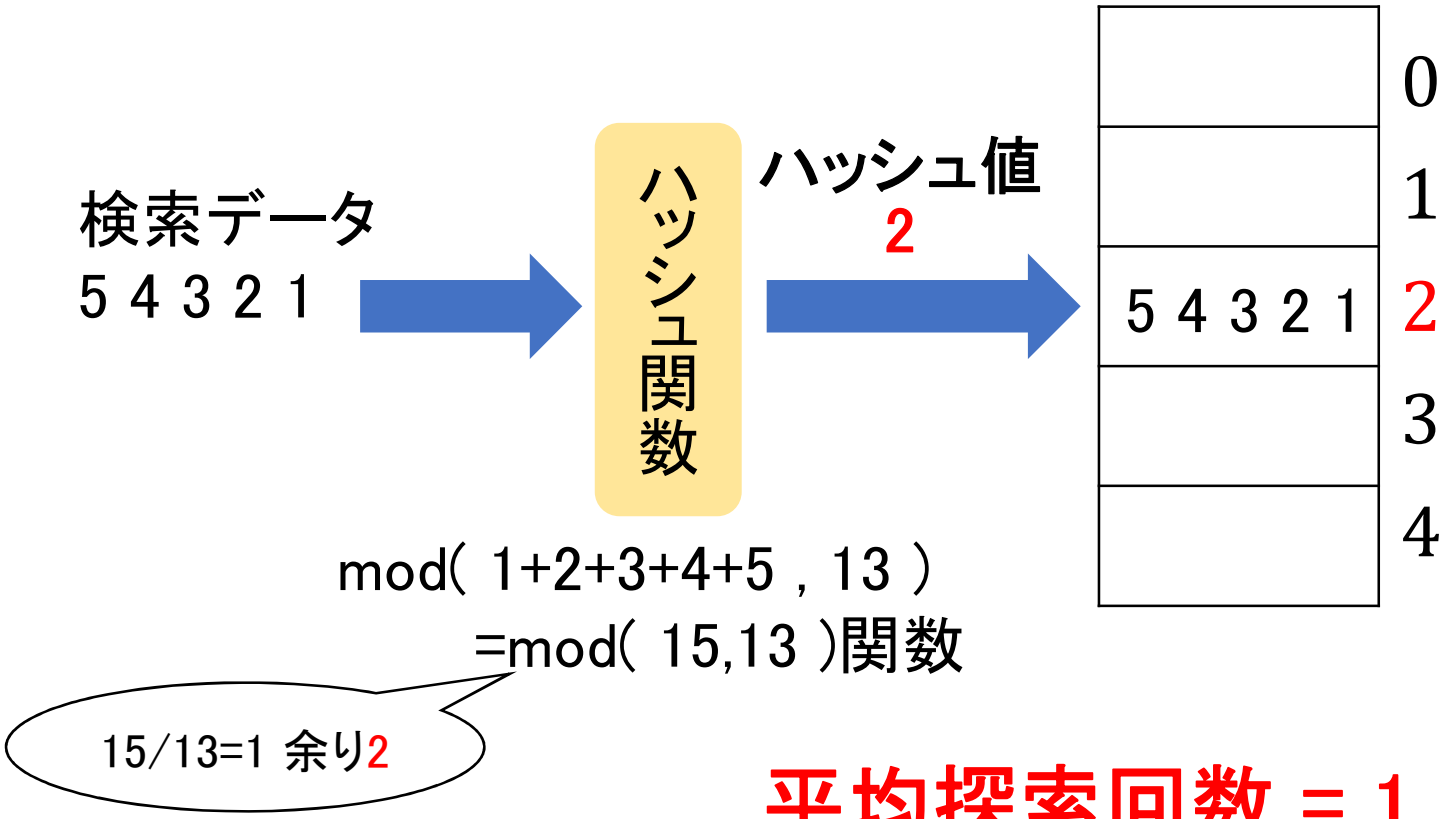
- 2分岐探索の比較回数

平均比較回数=$\log_2 n$

最大比較回数=$(\log_2 n)$+1

● ハッシュ探索

予め定めた関数（ハッシュ関数）で求めた格納場所（アドレス値）に、検索データを格納する。一方、データを検索するときも、格納したときと同じ関数で格納場所を求めて検索する。



平均探索回数 = 1

1.5.2 整列（ソート）アルゴリズム

- 大量のデータを使いやすいように並べ替えるためのアルゴリズム
- 一般的には、小さい順（昇順）または大きい順（降順）に並べる

● バブルソート

隣接するデータの大小を比較して、必要に応じて入れ替える

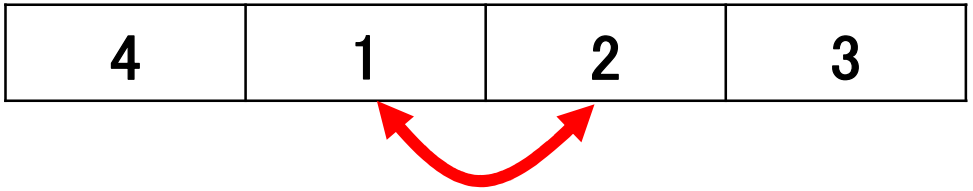
※昇順に並べる

4	1	3	2

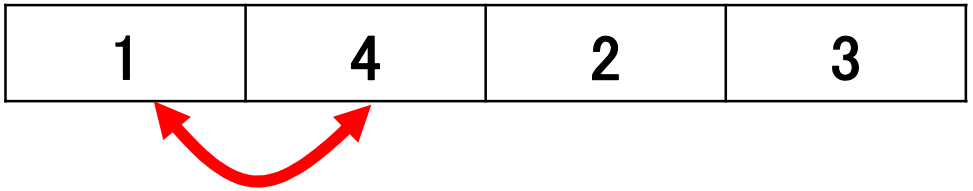
①右から順番に、2つの数値（3と2）を比較して、小さな数値（2）を左に置く（交換）

4	1	2	3

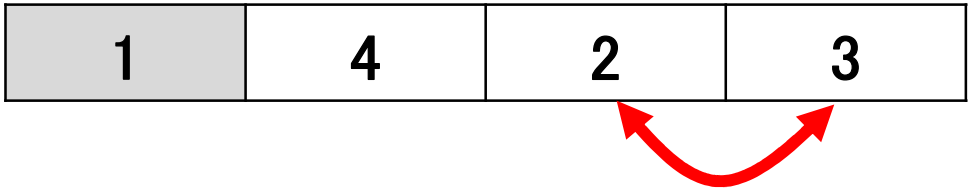
②2つの数値（1と2）を比較して、小さな数値（1）を左に置く（そのまま）



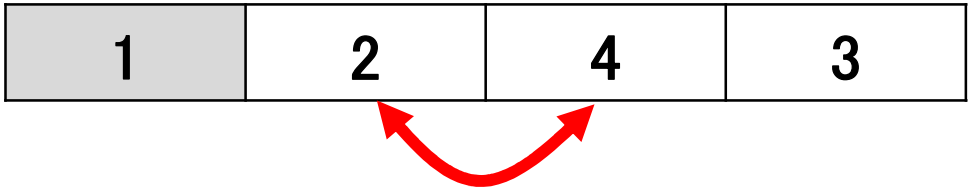
③2つの数値（4と1）を比較して、小さな数値（1）を左に置く（交換）



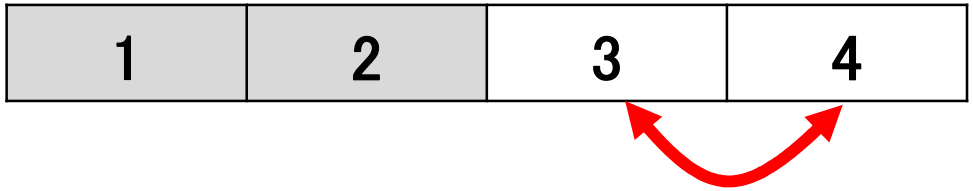
④一番小さい数値(1)が決定したので、次の2つの数値(2と3)を比較し、
小さい数値(2)を左に置く（そのまま）



⑤2つの数値(2と3)を比較し、小さい数値(2)を左に置く（交換）



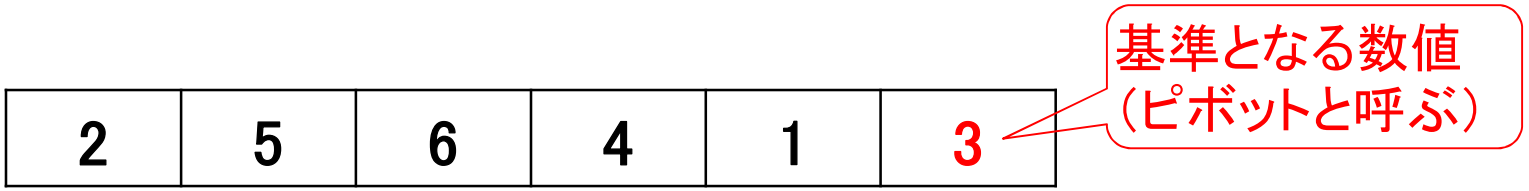
⑥2番目に小さい数値(2)が決定したので、次の2つの数値(4と3)を
比較し、小さい数値(3)を左に置く（交換）



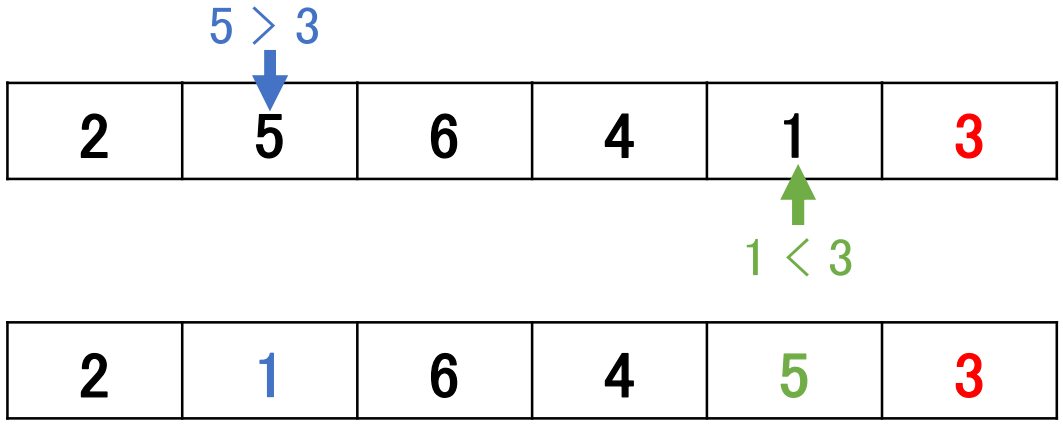
⑦これで、全ての数値が昇順に整列した

● クイックソート

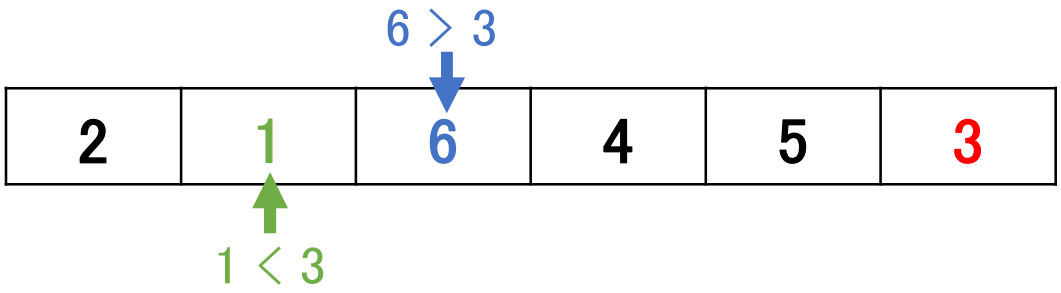
数列の中から適当な基準値を選び、それより[小さな数値のグループ]と[大きな数値のグループ]に分ける。分けたそれぞれのグループに対して、同じ操作を繰り返す。



① 数列の右側からピボットと比較して小さな数値を、左からピボットと比較して大きな数値を検索して、その数値を入れ替える



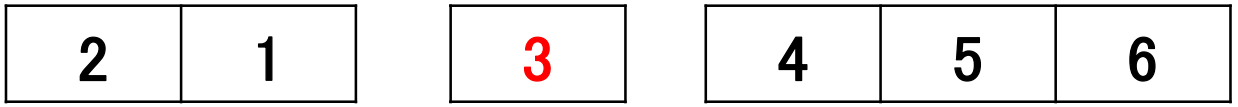
② 再び、数列の右側からピボットと比較して小さな数値を、左からピボットと比較して大きな数値を検索する



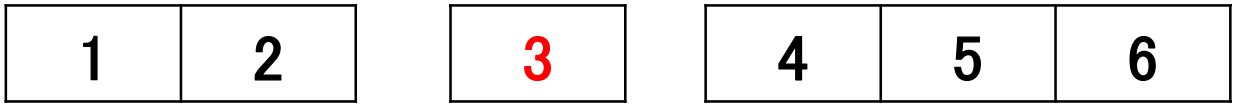
③ ①の操作の矢印(↑↓)の位置が、左右逆になったので、並べ替えは終了

④ 矢印 ↓ の数値6(最大値)とピボット3を入れ替える

⑤ ピボット3を中心に3つの部分数列に分ける



⑥ 各部分数列についてクイックソートする



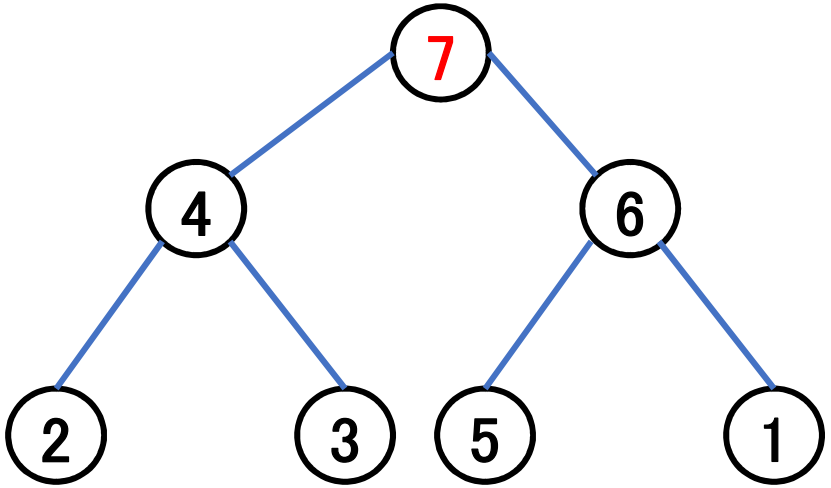
⑦ ソート後に、各部分数列を接続して、整列完了

● ヒープソート

未整列の数列を“順序木”と呼ばれる木構造に表現する。この中から、最大値と最小値を取り出して整列済み部分へと移動しながら、未整列部分を縮めて整列させる。

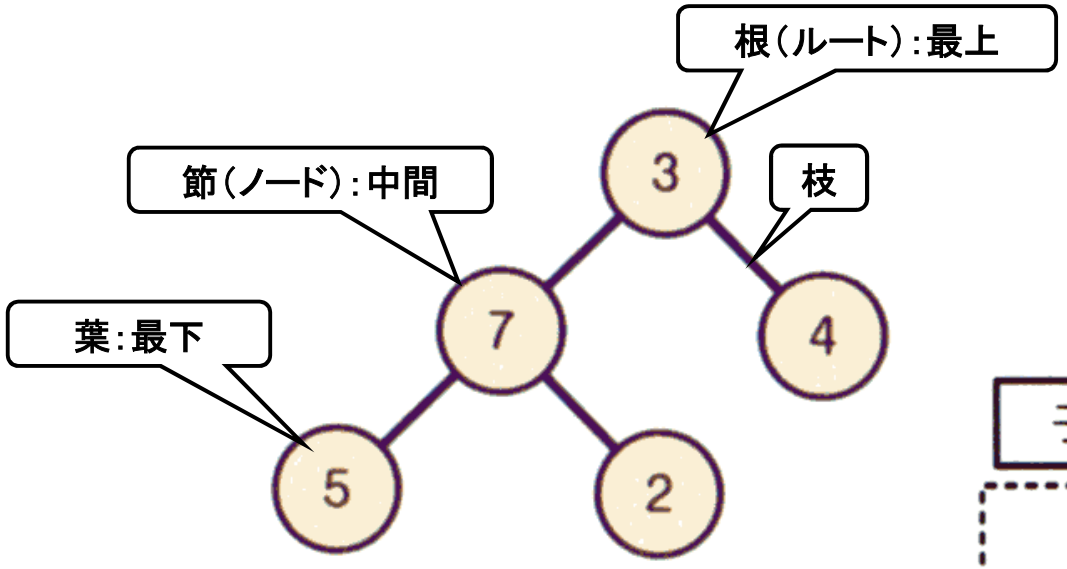
7	4	6	2	3	5	1

① 数列を完全2分木で表現する（完全2分木：どの部分木も親の数値が子の数値よりも大きい（または、小さい）構造 ）

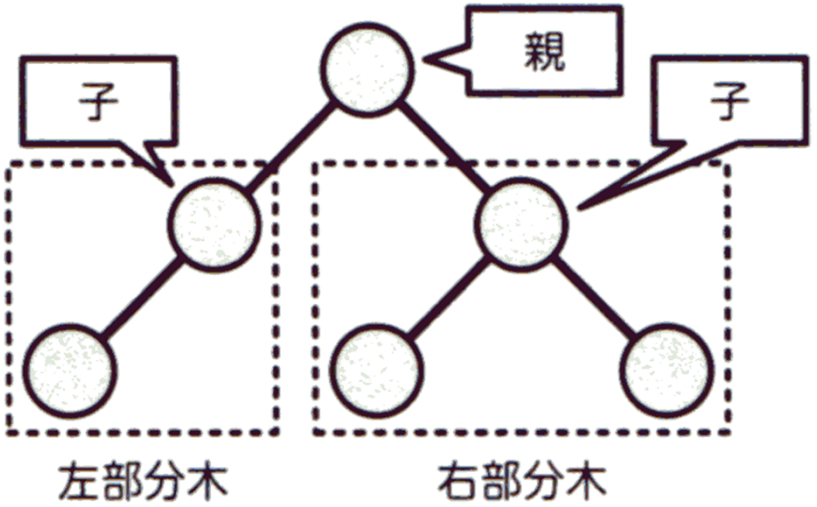


順序木（ツリー）構造

数値間に親子関係や主従関係がある

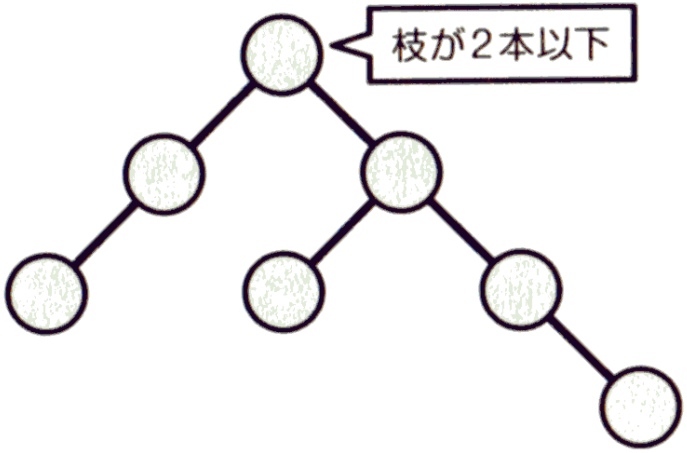


＜数値の呼び名＞

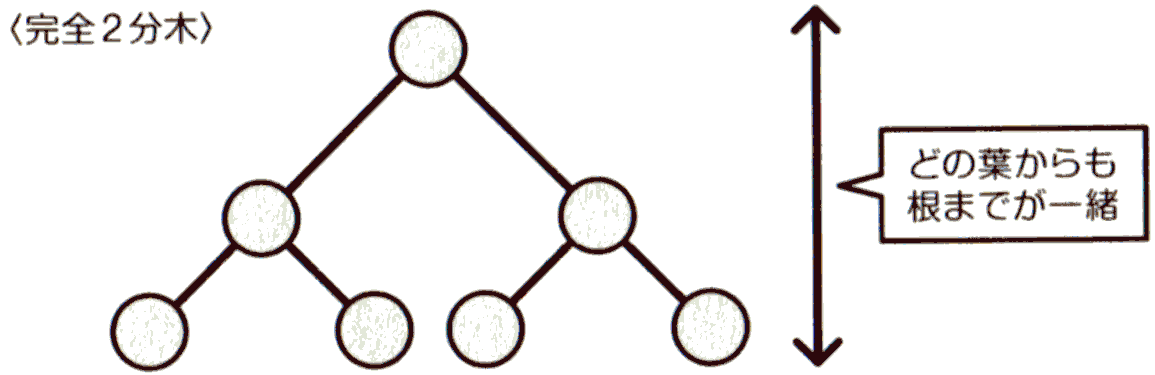


2分木

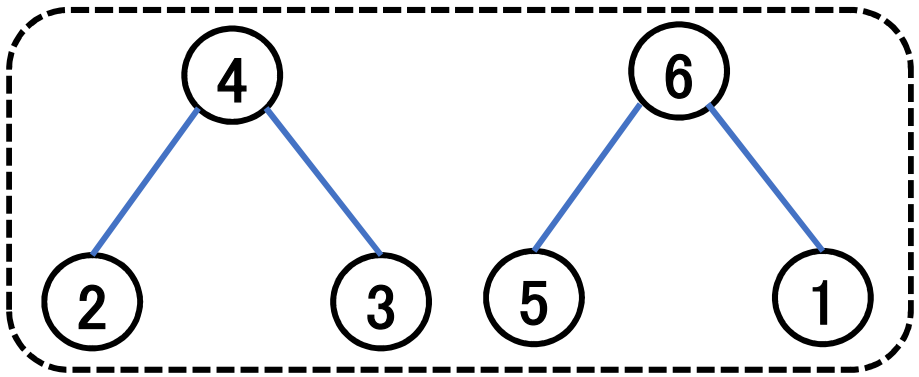
- 根や各節から出る枝が、すべて2本以下の木を2分木と呼ぶ



- 根から葉までの枝が、すべて一緒の2分木を完全2分木と呼ぶ



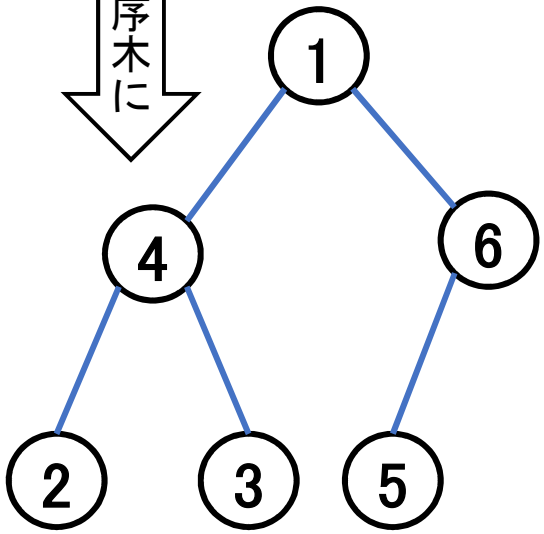
②順序木の根（最大値）7を取り出し、残った数列を順序木で表現する



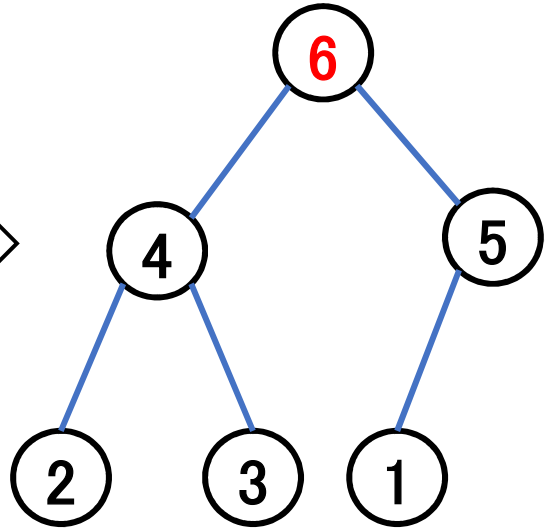
取り出した最大値7の数列

7

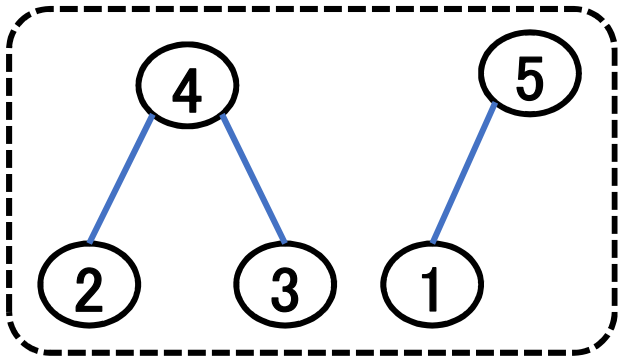
一つの順序木に



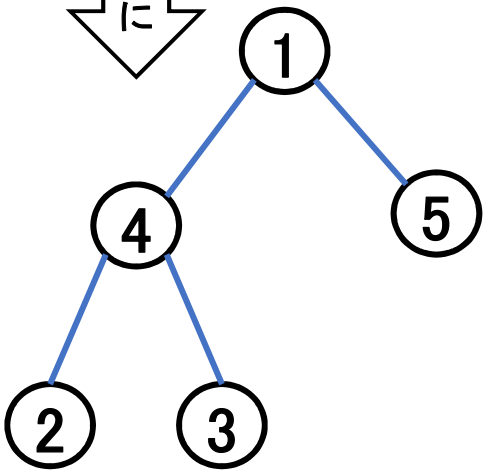
再構成

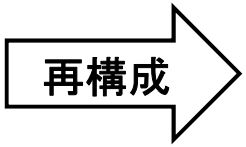


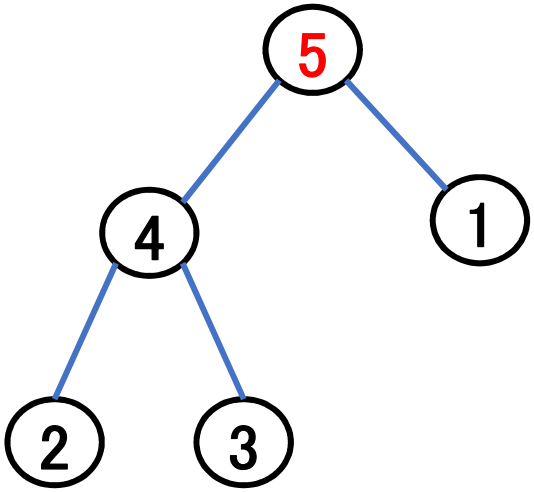
③再構成した順序木の根（最大値）6を取り出し、②で取り出した数列7の横に並べる



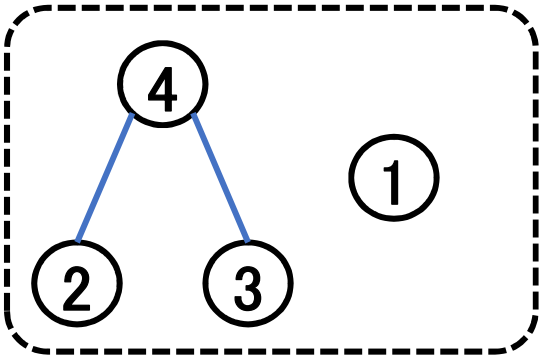
6	7





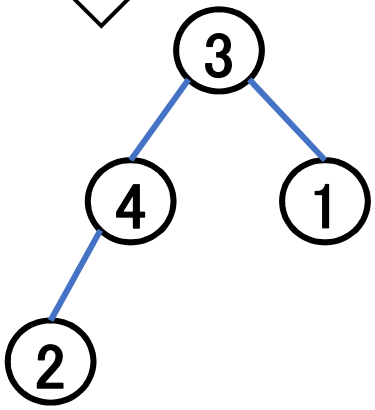


④再構成した順序木の根（最大値）5を取り出し、③で取り出した数列6の横に並べる

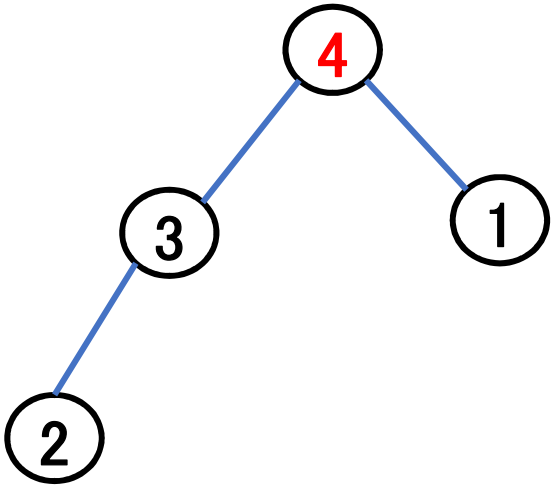


5	6	7

一つの順序木に

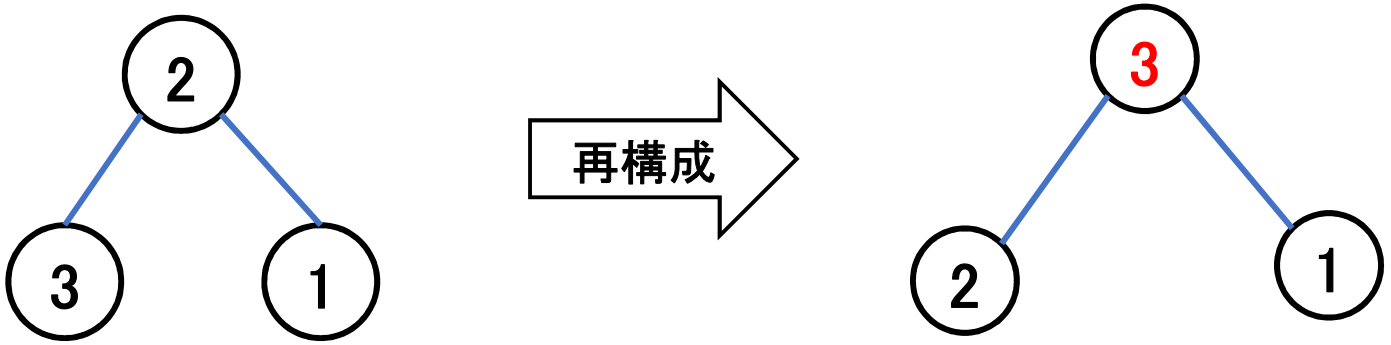


再構成



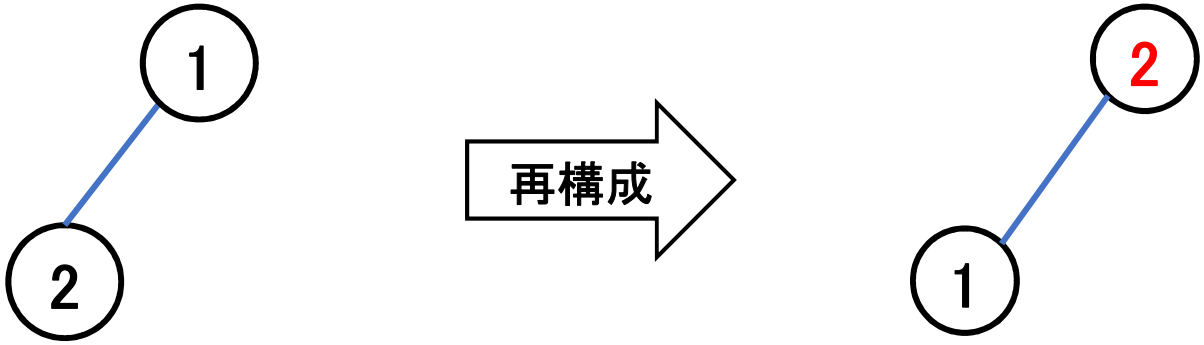
⑤再構成した順序木の根（最大値）4を取り出し、④で取り出した数列5の横に並べる

4	5	6	7



⑥再構成した順序木の根（最大値）3を取り出し、⑤で取り出した数列4の横に並べる

3	4	5	6	7



⑦再構成した順序木の根（最大値）2を取り出し、⑥で取り出した数列3の横に、残った数列1も続けて並べる

1	2	3	4	5	6	7

■ 過去問題5

ヒープソートの説明として、適切なものはどれか？

ア：ある間隔おきに取り出した要素から成る部分列をそれぞれ整列し、更に間隔を詰めて同様の操作を行い、間隔が1になるまでこれを繰り返す。

イ：中間的な基準値を決めて、それよりも大きな値を集めた区分と、小さな値を集めた区分に要素を振り分ける。次に、それぞれの区分の中で同様な処理を繰り返す。

ウ：隣り合う要素を比較して、大小の順が逆であれば、それらの要素を入れ替えるという操作を繰り返す。

エ：未整列の部分を順序木にし、そこから最小値を取り出して整列済の部分に移す。この操作を繰り返して、未整列の部分を縮めていく。

ヒープソートは、未整列データを「親の値≦子の値」(または「親の値≧子の値」)の関係をもつ順序木として表現し、整列後の根の値(最小値または最大値)を取り出すことを繰り返して整列を行う方法です

ア：シェルソート
イ：クイックソート
ウ：バブルソート
エ：ヒープソート