アルゴリズムとデータ構造

http://cobayasi.com/koza/kihon/6_algorithm&data.pdf

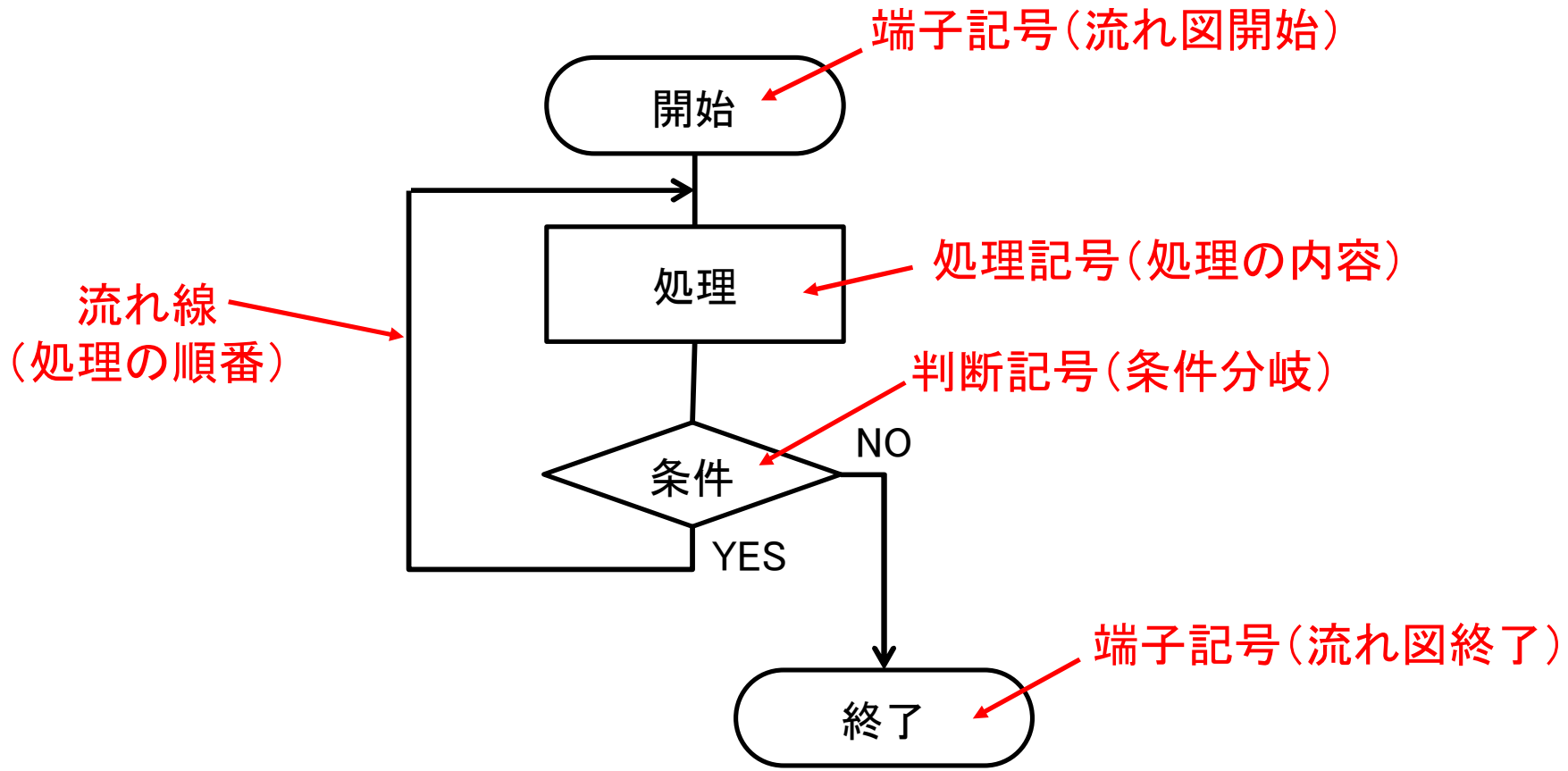
1. アルゴリズム★
2. 配列 ★
3. キューとスタック ★★★
4. リスト構造 ★★
5. 木構造 ★★
6. 探索アルゴリズム ★★★
7. 整列アルゴリズム★
8. 再帰アルゴリズム★★★

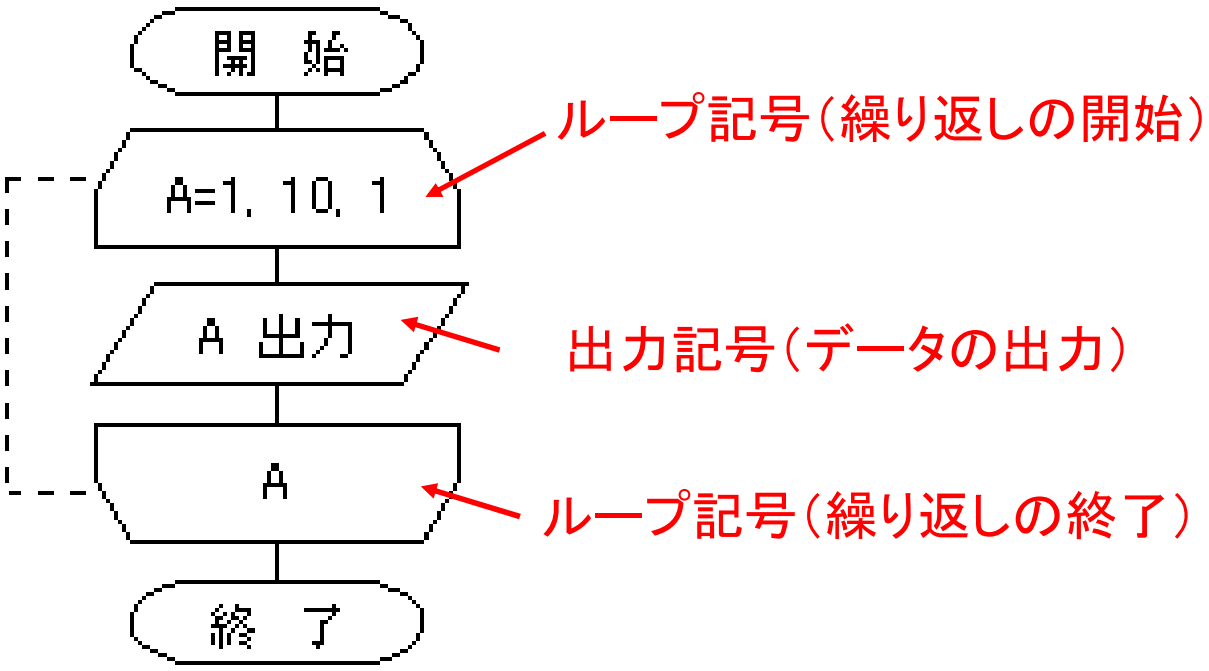
★1. アルゴリズム

プログラムの処理手順

●流れ図(フローチャート)

プログラムの処理手順(アルゴリズム)を図式化したもの



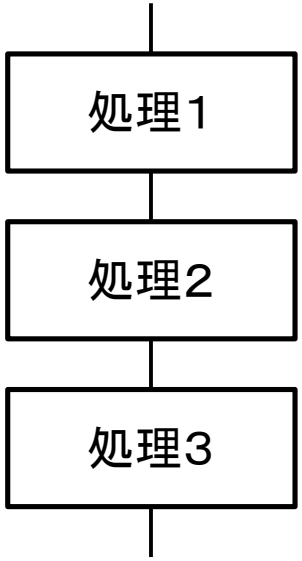


変数Aに初期値=1を与え、1ずつ増加しながら、10回処理（Aを出力）繰り返す

• 流れ図の処理パターン

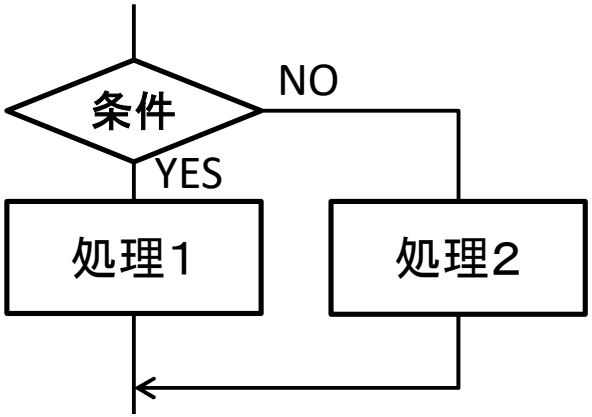
■ 順次

上から順番に処理を実行



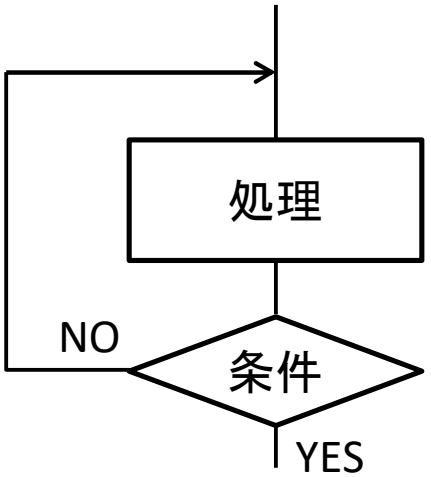
■ 選択

条件によって異なる処理を実行



■ 繰り返し

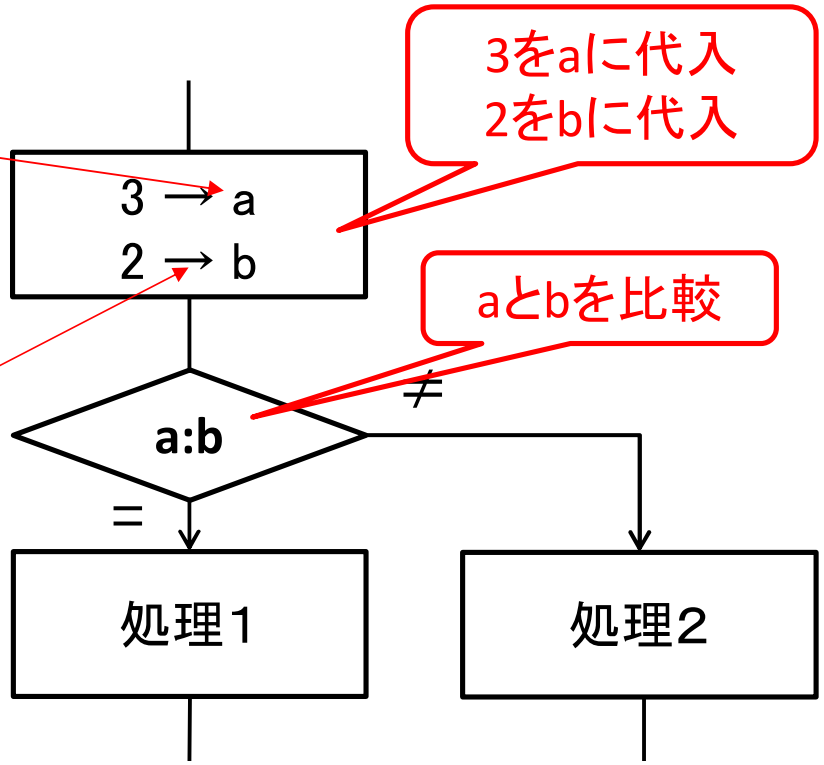
条件が満足するまで処理を実行



● 変数と代入

- ■ 変数とは、プログラムで扱う文字や数値（いわゆるデータ）を記憶する入れもの
- ■ 変数に名前を付けると、プログラムの中で、複数のデータを扱いやすくなる
- ■ 変数にデータを入れることを、代入と呼ぶ

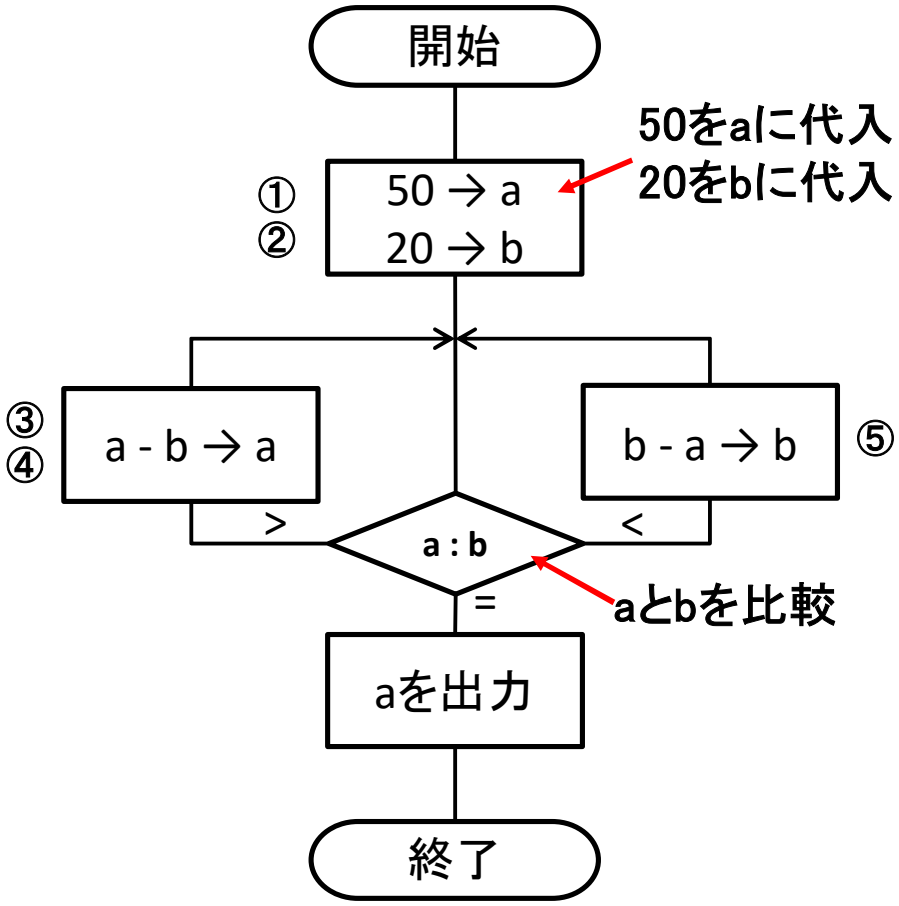
＜変数を使った流れ図＞



●トレース

流れ図に書かれている処理内容が正しいかを確認すること。確認結果をまとめた表 ⇒ **トレース表**

＜流れ図＞



＜トレース表＞

手順	a	b
①	50	
②	50	20
③	30	20
④	10	20
⑤	10	10

変数a　変数b

50 − 20をaに代入

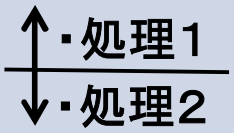
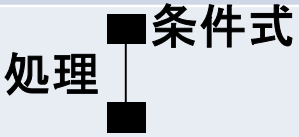
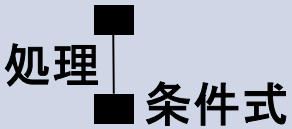
30 − 20をaに代入

20 − 10をbに代入

● 擬似言語

実際のプログラミング
には、使用しない

流れ図と同じ役割をする、アルゴリズムを文書や記号などを使って表す言語（流れ図の別記法）

記述型式	説　明
○	手続き、変数、型などを宣言する
・変数←式	変数に式の値を代入する
	処理の選択。条件式を満足するときには処理1を、満足しないときには処理2を実行
	前判定繰り返し処理。条件を満足する間は処理を続ける。条件は処理よりも先に来る
	後判定繰り返し処理。条件を満足する間は処理を続ける。条件は処理の後に来る

擬似言語の使用例

nの階乗を求める

〇整数型：Factorial（整数型:n）
〇整数型：f , i

・f ← 1
・i ← 2

■ i <= n
　・f ← f X i
　・i ← i + 1
■

・return f

コメント

変数型宣言（関数Factorial、整数型引数n）
（変数fとi）

変数に値を代入（変数fに値1を代入）
（変数fに値2を代入）

条件式（条件i<=nが満足している間は処理を繰り返す）

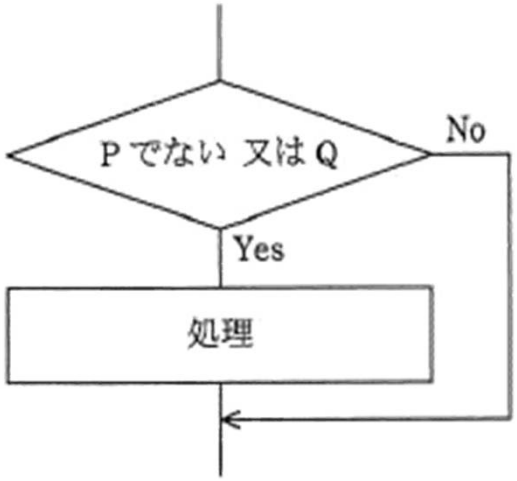
処理内容（乗算）

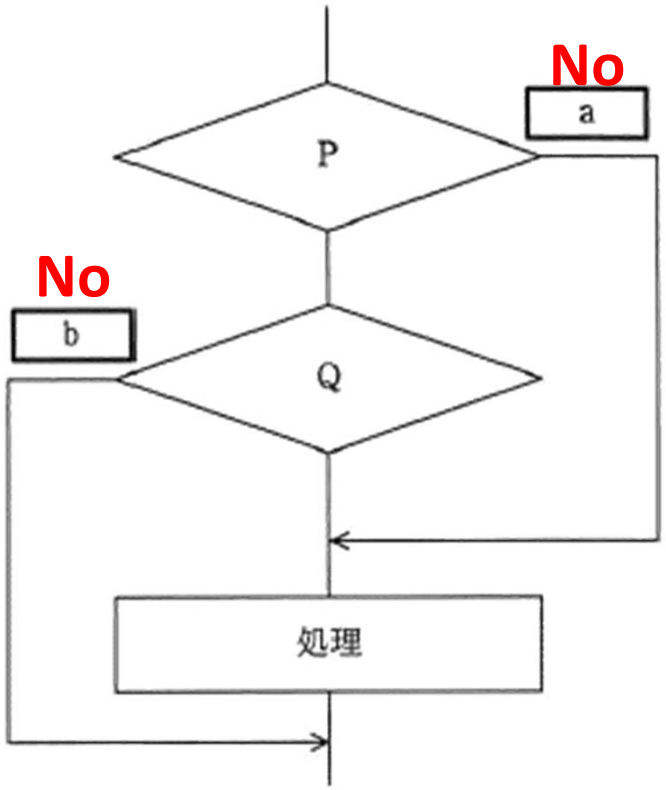
処理内容（カウンタiをインクリメント）

戻り値（変数fの値を戻す）

【過去問題】

「右の流れ図」が「左の流れ図」と同じ動作をするために、aとbに入るYesとNoの組合せはどれか。



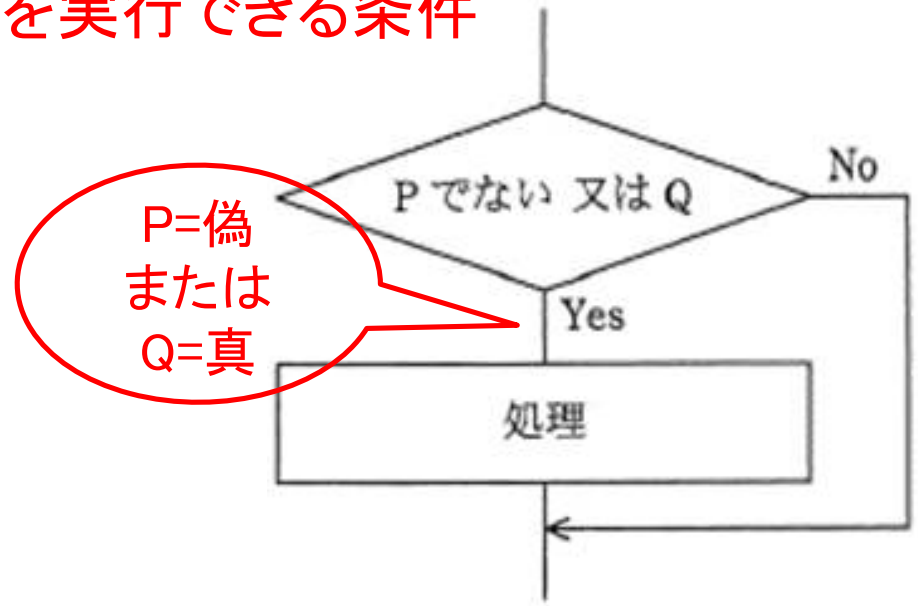


	a	b
ア	No	No
イ	No	Yes
ウ	Yes	No
エ	Yes	Yes

【解説】

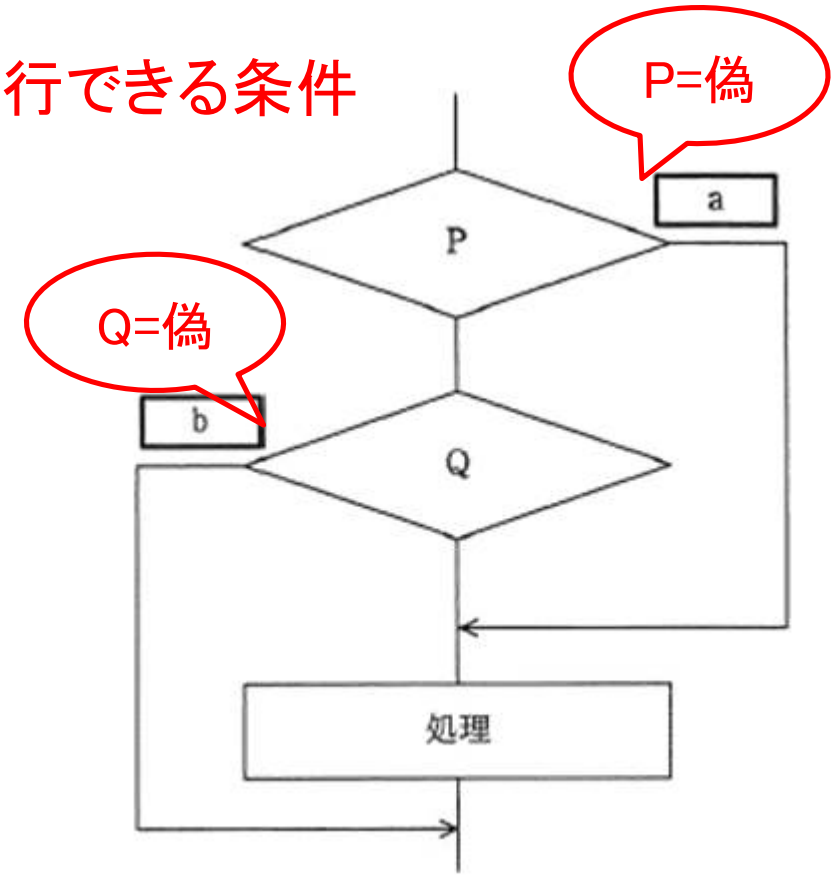
「左の流れ図」と「右の流れ図」で、処理が実行できる条件を考える

- 「左の流れ図」で処理を実行できる条件



分岐条件「Pでない又はQ」が、真となる場合に処理を実行する。“又は”というのは“OR演算”と同じ意味なので、処理を実行するためには、「Pが偽」と「Qが真」のどちらかを満たせばよい。

• 「右の流れ図」で処理を実行できる条件



aは処理が実行できる方向、bは処理が実行しない方向に分岐している。Pに関して処理が実行される条件は、「P=偽」なのでaには”No”、Qに関して処理が実行されない条件は、「Q=偽」なのでbにも”No”が入る。

★2. 配列

●データ構造

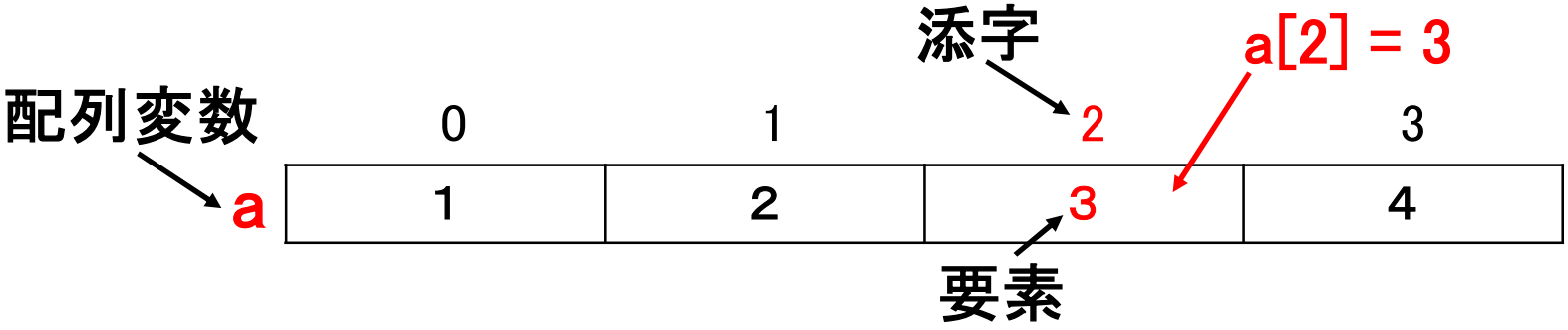
- データがメモリ上にどのような構造で保存されているか？
- データ構造の違いによって、プログラム（アルゴリズム）の効率が大きく異なる

■ データ構造の種類

- 配列
- キュー
- スタック
- リスト構造
- 木構造

● 配　列

■１次元配列（データが横一列に配置されている）



■２次元配列（データが縦横に配置されている）

a[1,2] = 7

a	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

【過去問題】

次の規則に従って配列の要素A[0],A[1], … ,A[9]に正の整数kを格納する。kとして16,43,73,24,85を順に格納したとき、85が格納される場所はどこか。

ここで、x mod yは、xをyで割った剰余を返す。また、配列の要素は全て0に初期化されている。

〔規則〕

① A[k mod 10]＝0ならば、k→A[k mod 10] とする。

② ①で格納できないとき、A[(k＋1) mod 10]＝0ならば、k→ A[(k＋1) mod 10] とする。

③ ②で格納できないとき、A[(k＋4) mod 10]＝0ならば、k→A[(k＋4) mod 10] とする。

ア A[3]　イ A[5]　ウ A[6]　エ A[9]

【ヒント】kの値が格納される配列を、1つずつ考える

＜16の格納場所＞
16 mod 10＝6,A[6]＝0（初期化されている）なので、16をA[6]に格納する
A[16 mod 10]=A[6]➡A[6]=16

＜43の格納場所＞
43 mod 10＝3,A[3]＝0なので、43をA[3]に格納する
A[43 mod 10]=A[3]➡A[3]=43

＜73の格納場所＞
73 mod 10＝3,A[3]には、すでに43が入っているので格納できない
(73＋1) mod 10＝4,A[4]＝0なので、73をA[4]に格納する
A[73+1 mod 10]=A[4]➡A[4]=73

＜24の格納場所＞
24 mod 10＝4,A[4]には、すでに73が入っているので格納できない
(24＋1) mod 10＝5,A[5]＝0なので、24をA[5]に格納する
A[24+1 mod 10]=A[5]➡A[5]=24

＜85の格納場所＞
85 mod 10＝5,A[5]には、すでに24が入っているので格納できない
(85＋1) mod 10＝6,A[6]には、すでに16が入っているので格納できない
(85＋4) mod 10＝9,A[9]＝0なので、85をA[9]に格納する
A[85+4 mod 10]=A[9]➡A[9]=85

格納する場所を、ハッシュ関数で求める
問題は、たいへんよく出題される

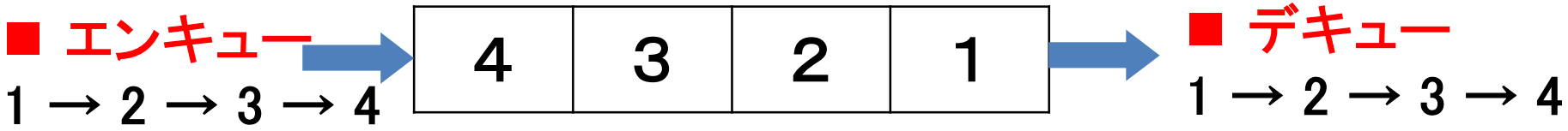
★★★3. キューとスタック

データを保存したり取り出す方法

ディスクの入出力
やOSのタスクの実行で使用

- キュー

  入れた順番で出力する（FIFO:First-In First-Out）

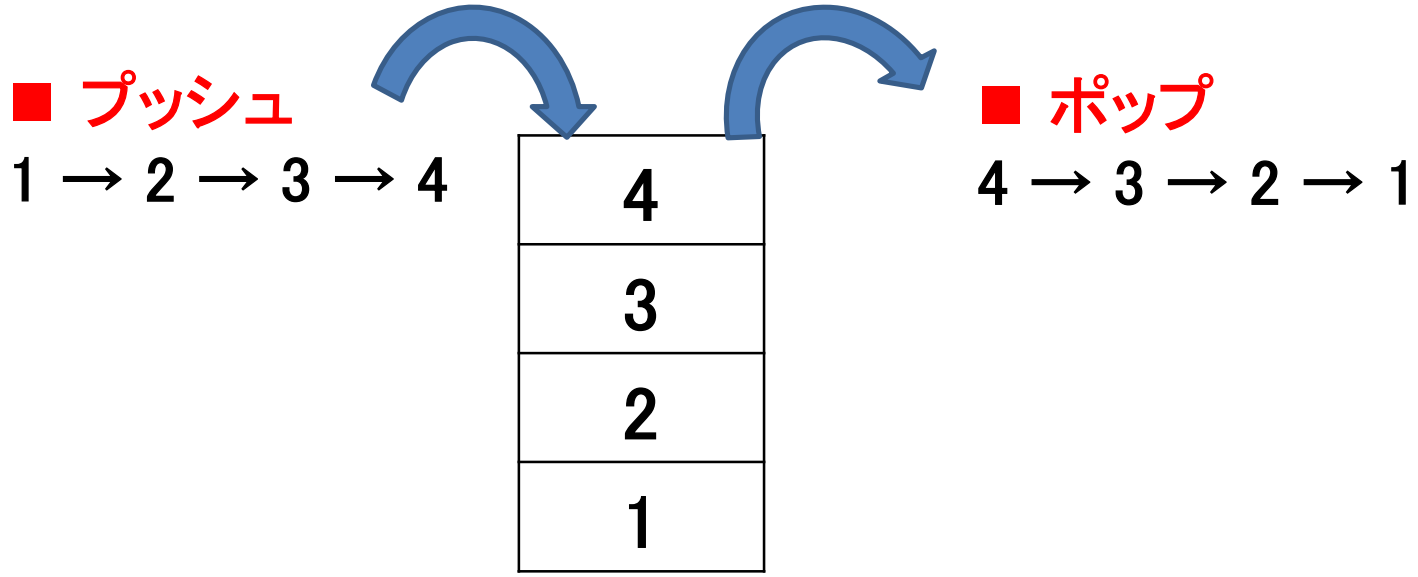


プログラムの実行中に他の
関数を呼び出すときに使用

- スタック

  入れた順番とは逆の順番で出力する

  （LIFO:Last-In First-Out）



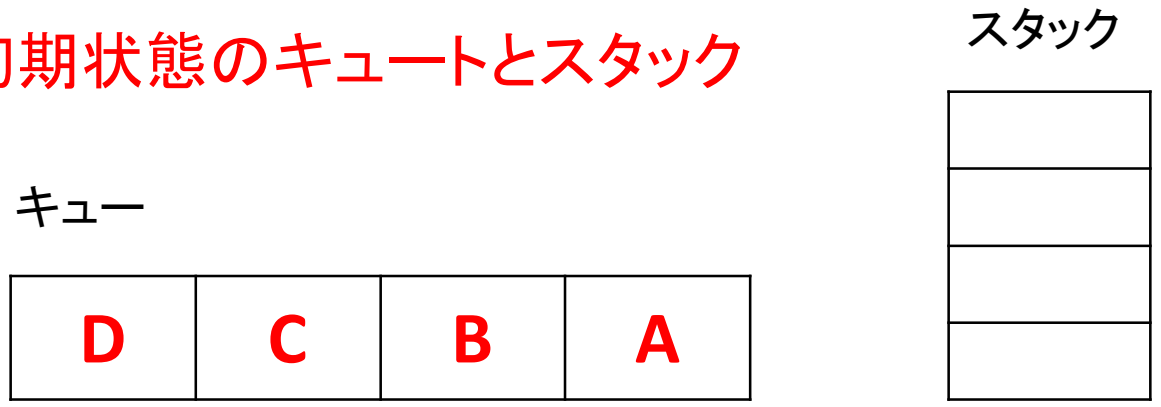
【過去問題】

四つのデータA,B,C,Dが、この順に入っているキューと空のスタックがある。手続pop_enqとdeq_pushを使って、キューの中のデータをD,C,B,Aの順に並べ替えるとき、deq_pushの実行回数は最小で何回か。

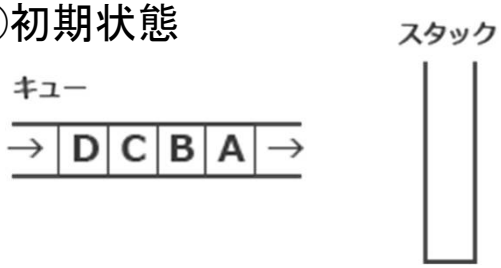
ここで、pop_enqはスタックから取り出したデータをキューに入れる操作であり、deq_pushはキューから取り出したデータをスタックに入れる操作である。

ア 2　イ 3　ウ 4　エ 5

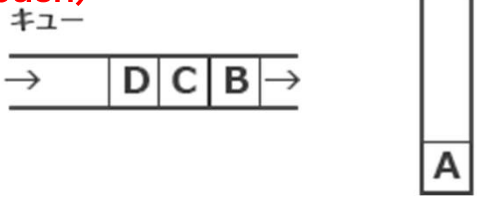
【ヒント】初期状態のキューとスタック



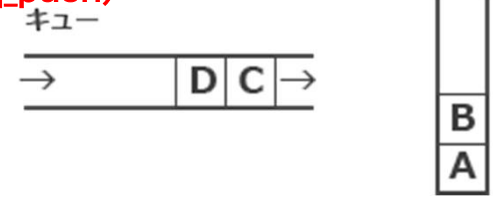
①初期状態



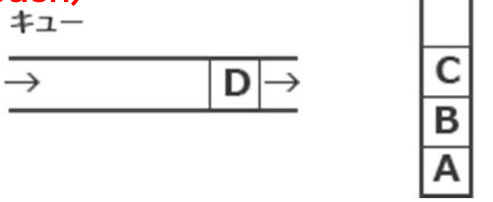
②キューからAを取り出す
(deq_push)



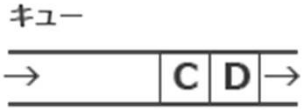
③キューからBを取り出す
(deq_push)



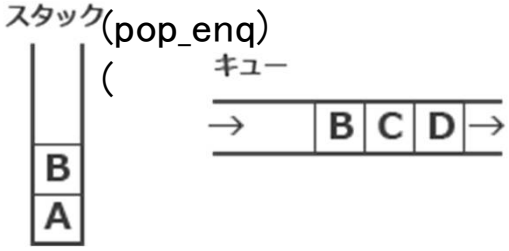
④キューからCを取り出す
(deq_push)



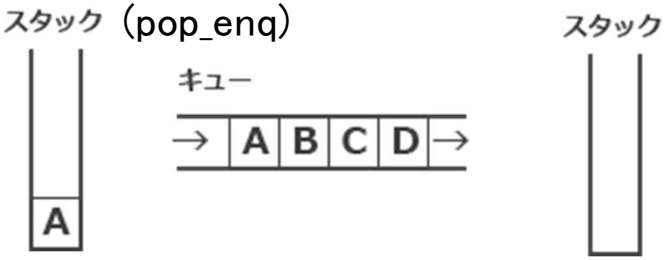
⑤スタックからCを取り出す
(pop_enq)



⑥スタックからBを取り出す
(pop_enq)
(



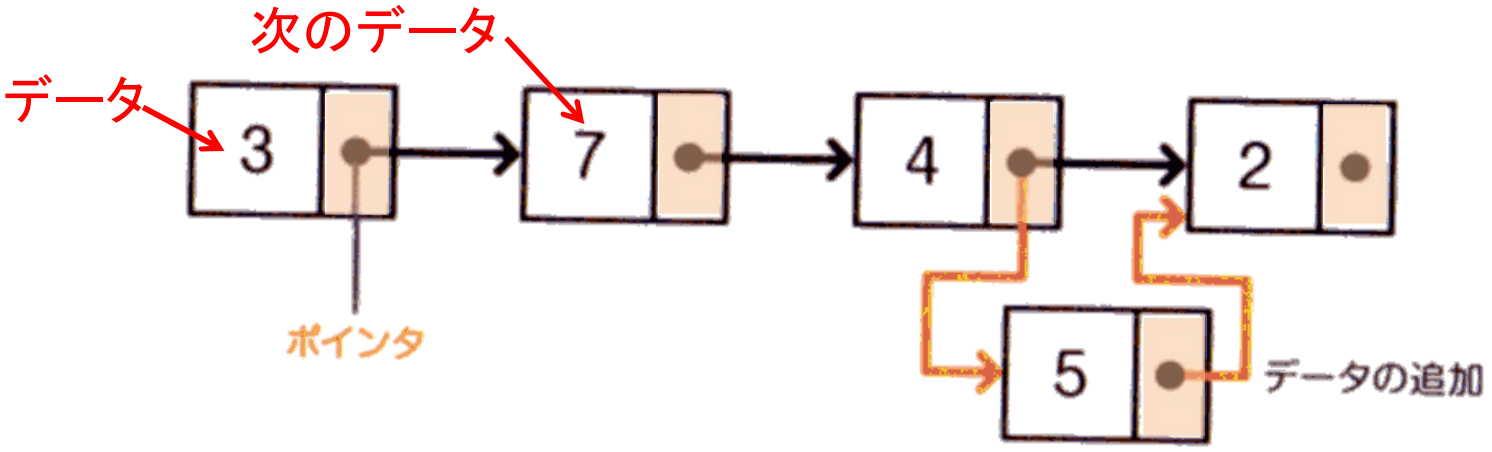
⑦スタックからAを取り出す
(pop_enq)



★★4. リスト構造

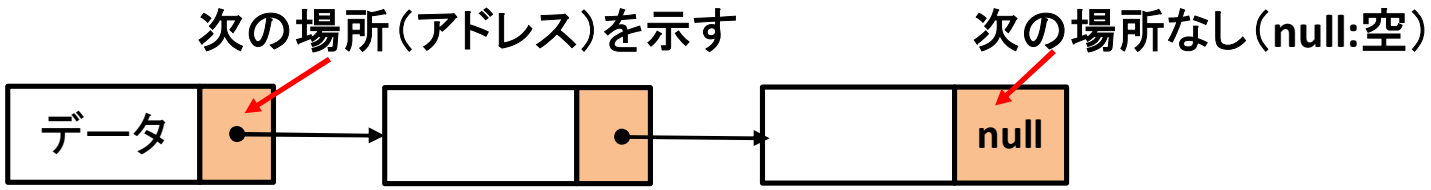
●リスト構造とは

- ■ データ同士を直線的に繋いだ構造
- ■ つなげる順番は、決まっていない。ポインタが指したデータを次に繋ぎ、途中に別のデータをつなげることもできる。また、削除もできる
- ■ データを取り出すときには、ポインタを使って、データの位置をずらしながら取り出す



●リスト構造の種類

- 単方向連結リスト
  データの後ろに1つのポインタを付けて連結する

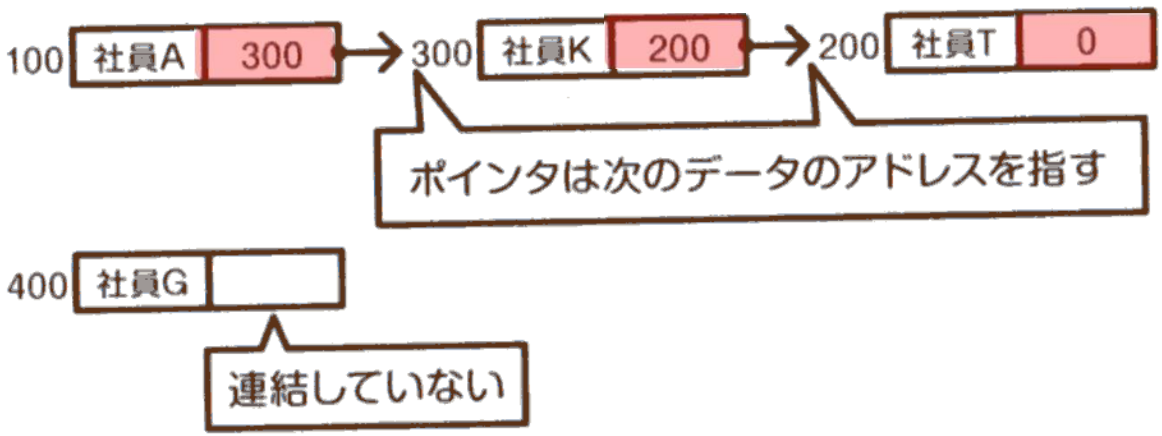


- 双方向連結リスト
  データの前後にポインタを付けて連結する

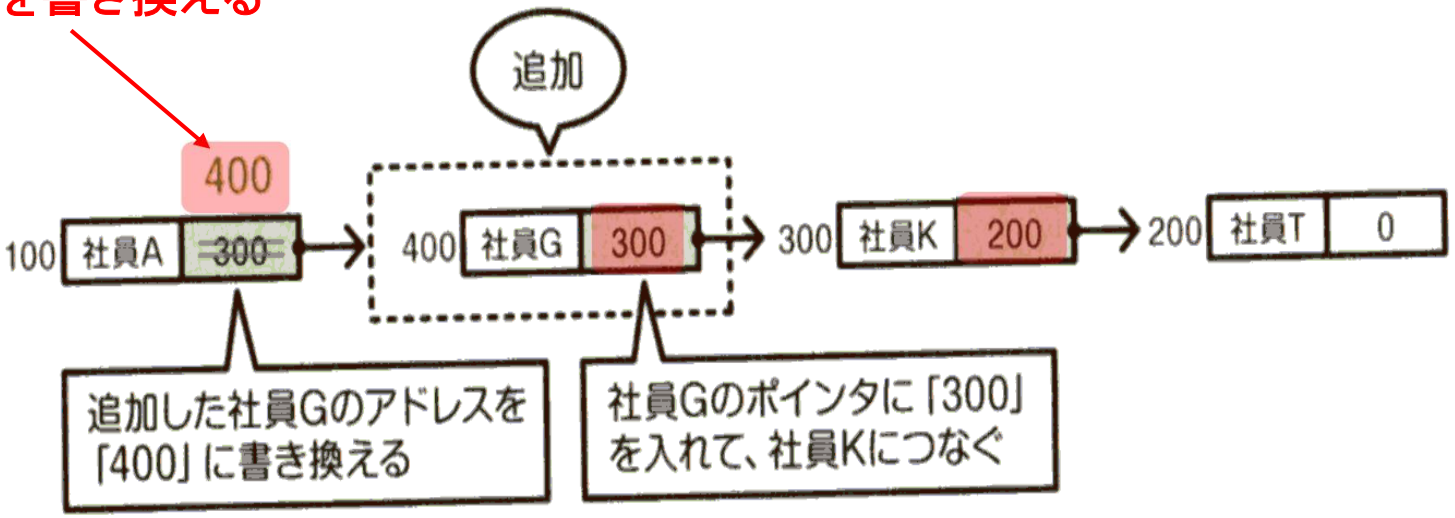
●データの追加と削除（単方向連結リスト）

■ 社員Gを、社員Aと社員Kの間に追加する場合

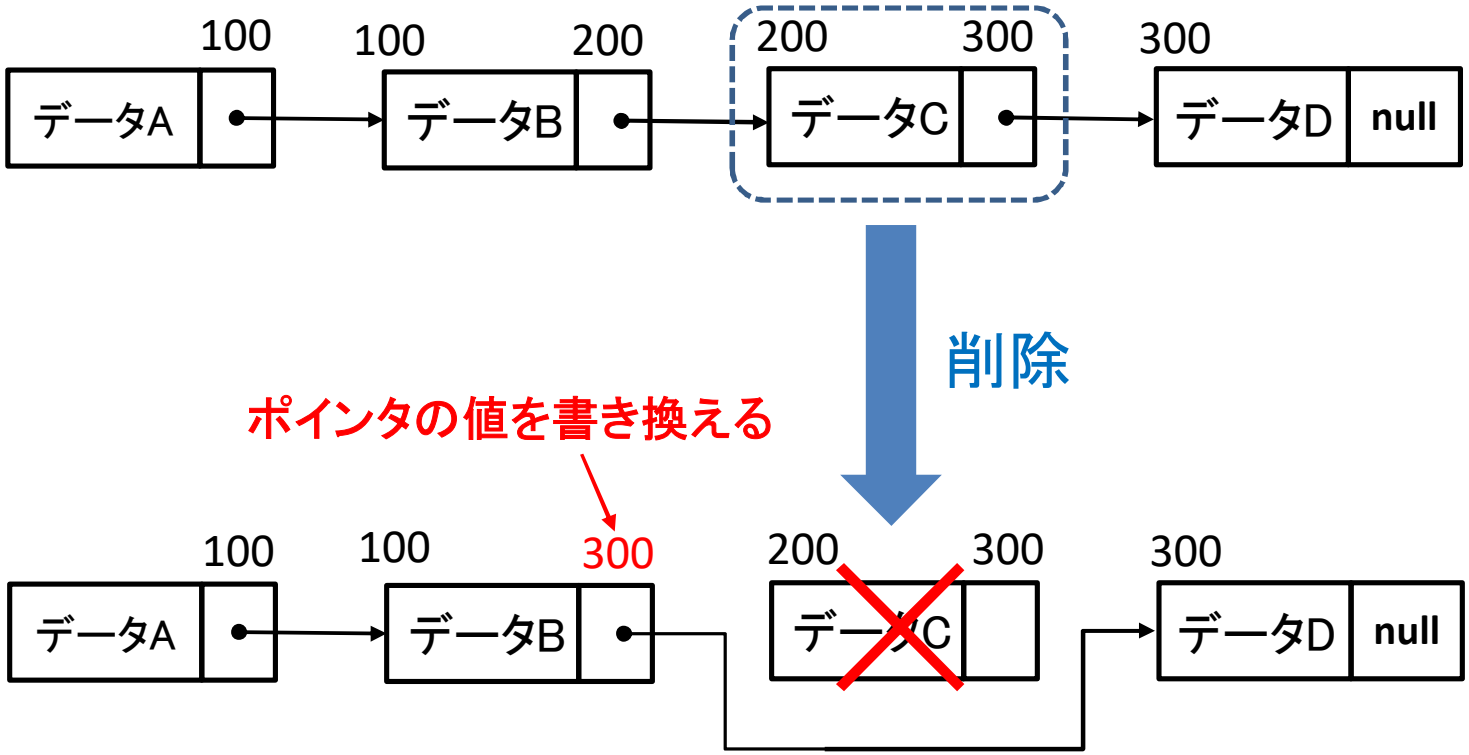
アドレス	データ	ポインタ
100	社員A	300
200	社員T	0
300	社員K	200
400	社員G	



ポインタの値を書き換える



■ データCを削除する場合



【過去問題】

下の左表は、配列を用いた連結セルによるリストの内部表現であり、リスト[東京，品川，名古屋，新大阪]を表している。このリストを[東京，新横浜，名古屋，新大阪]に変化させる操作はどれか。下の右表から選べ。

ここで、A(i, j)は表の第i行第j列の要素を表す。例えば、A(3,1)＝“名古屋”であり、A(3,2)＝4である。また、→は代入を表す。

列

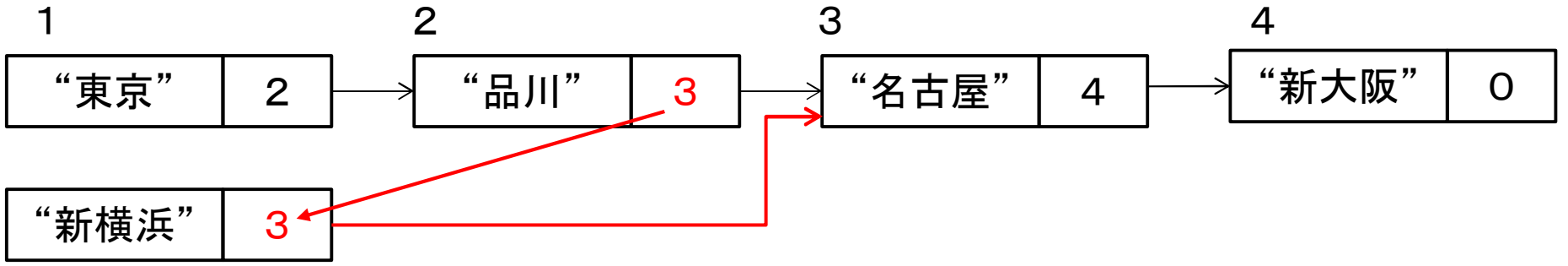
A	1	2
1	“東京”	2
2	“品川”	3
3	“名古屋”	4
4	“新大阪”	0
5	“新横浜”	

行

	第1の操作	第2の操作
ア	5→A(1,2)	A(A(1,2),2)→A(5,2)
イ	5→A(1,2)	A(A(2,2),2)→A(5,2)
ウ	A(A(1,2),2)→A(5,2)	5→A(1,2)
エ	A(A(2,2),2)→A(5,2)	5→A(1,2)

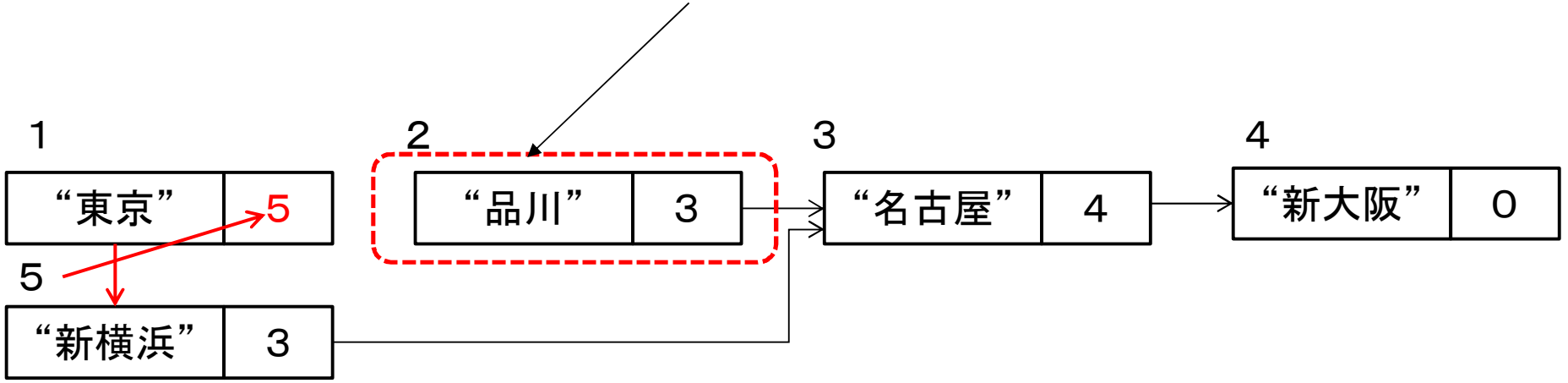
①“東京”の後続ノードの後続ポインタを“新横浜”の後続ポインタにコピーする

“東京”の後続ノードが格納されている位置はA(1, 2)によって得られる（値は2です）。そのノードの後続ポインタA(A(1，2)，2)すなわちA(2, 2)を、“新横浜”の後続ポインタA(5，2)にコピーする。A(5, 2)に3が代入されるので、“新横浜”は“名古屋”を指すことになる。



②“東京”の指す先を“新横浜”とする

“東京”の後続ノードを“新横浜”とするために、“東京”の後続ポインタA(1, 2)に5を代入する。その結果、どこからも指されない“品川”がリストから切り離されることになる。

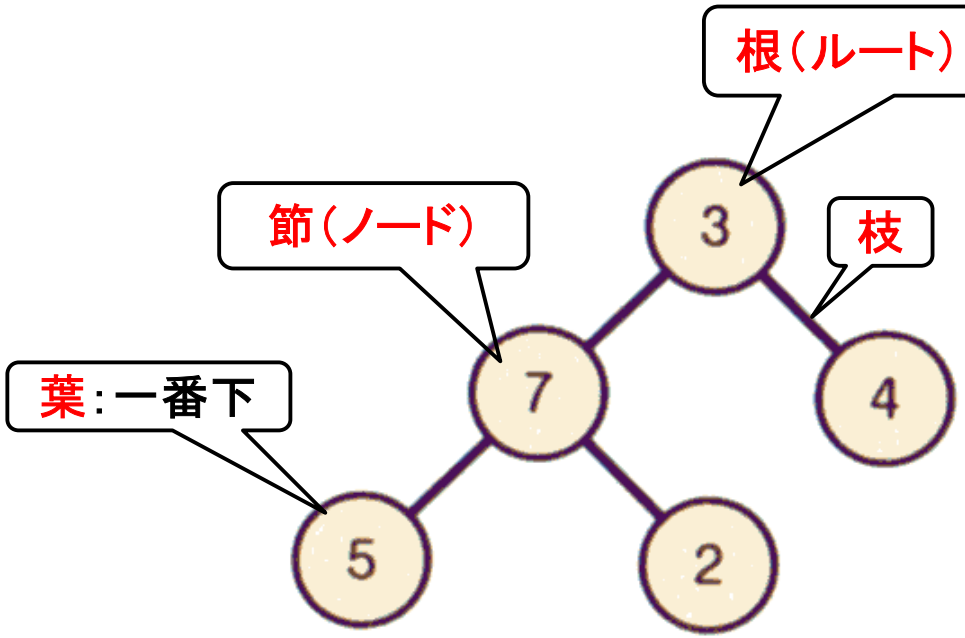


★★5. 木(ツリー)構造

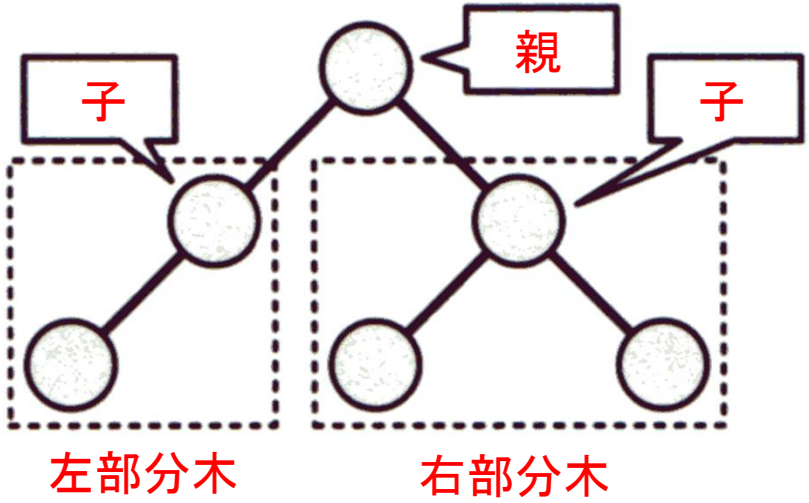
● 木構造とは

■ データ間に親子関係や主従関係がある構造

■ データを取り出すときには、各階層をたどって取り出す



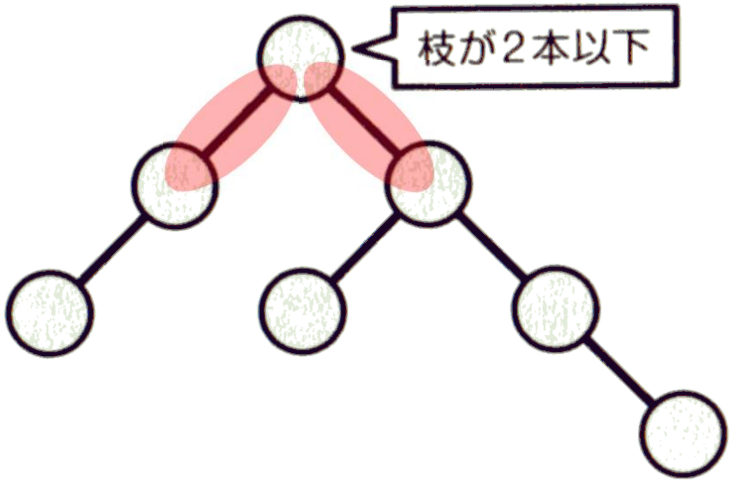
木構造:各データの呼び名



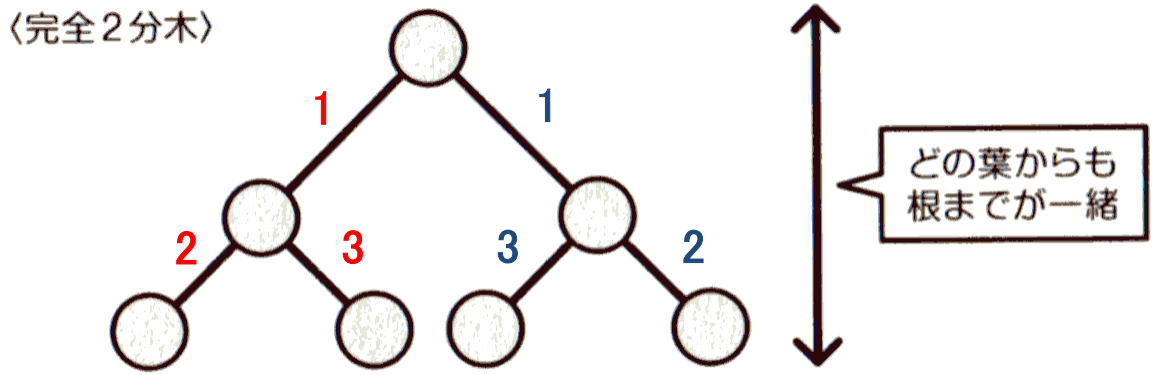
木構造:親子関係

● 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

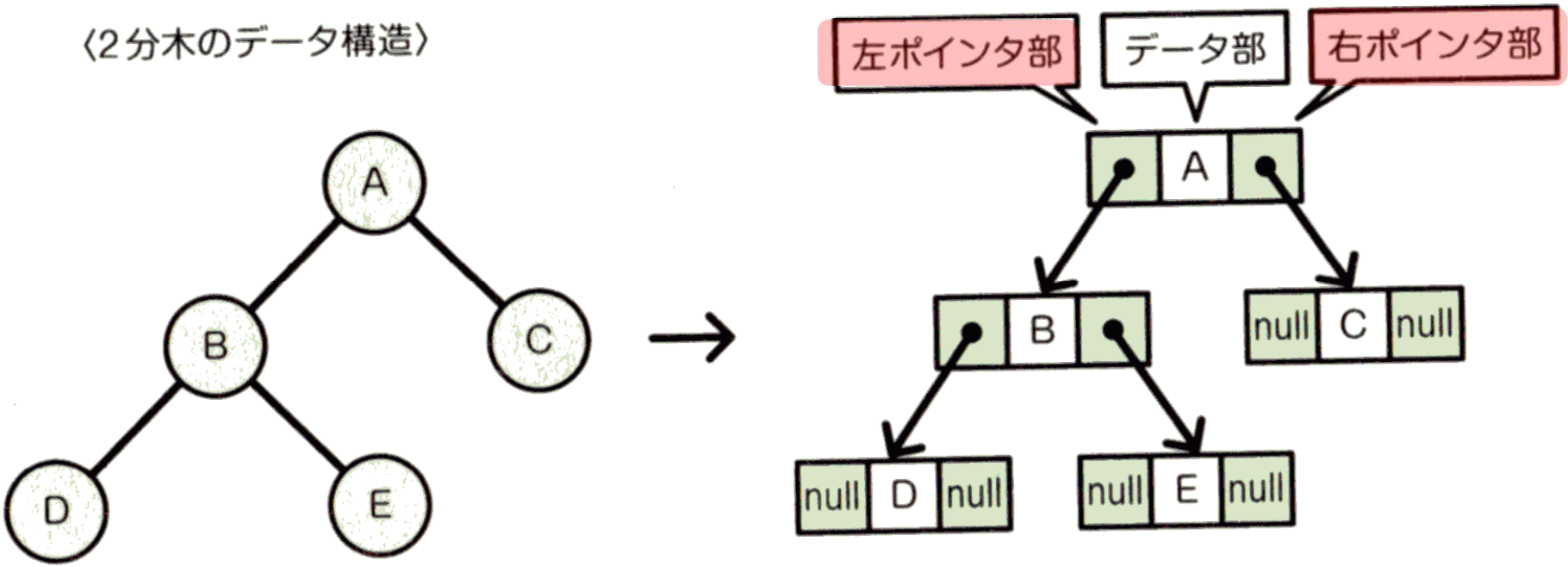


■ 根から葉までの枝の数が同数の2分木を完全2分木と呼ぶ



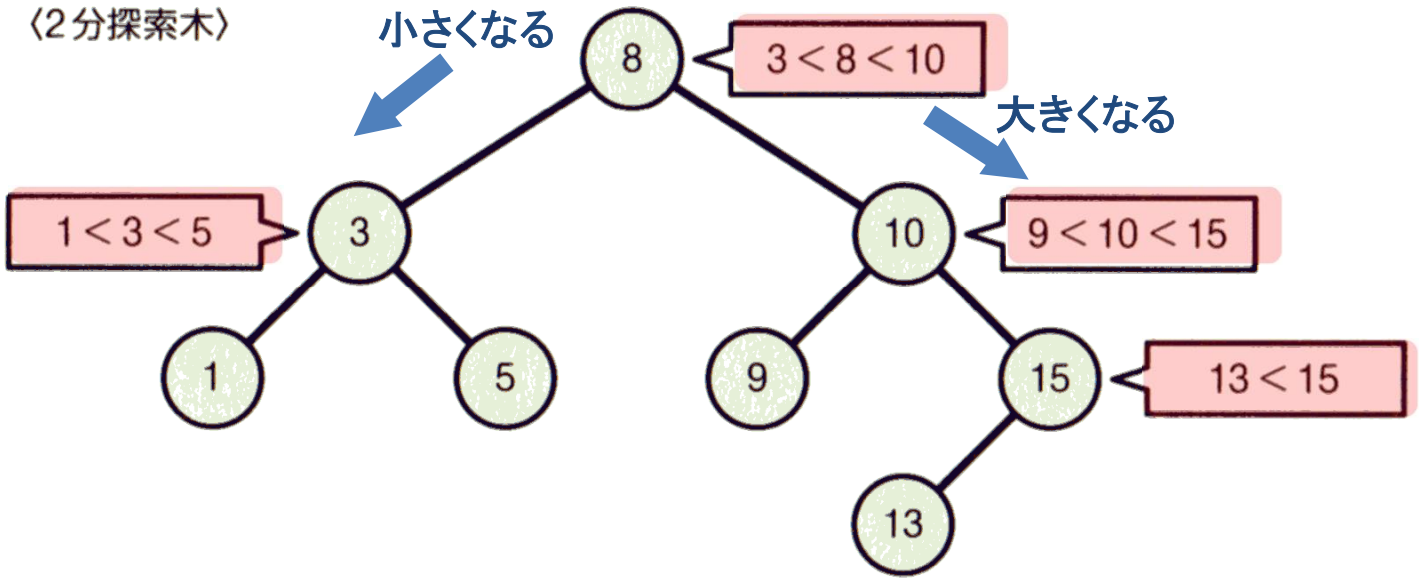
■ 2分木のデータ構造

親となっている節のポインタ部には、子の節のデータの場所が格納されている



● 2分探索木

　親のデータが、左部分木のデータより大きく、右部分岐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



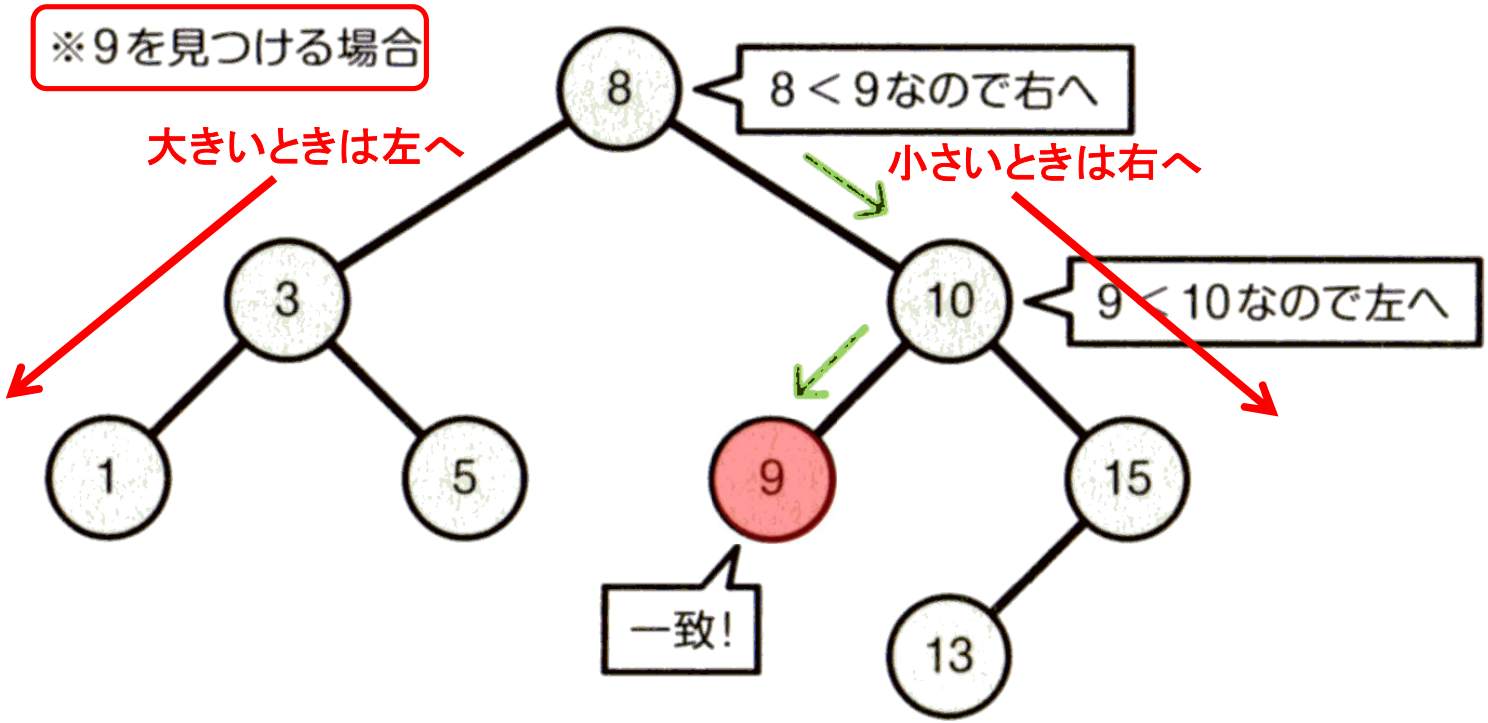
全ての節でも（ 左の子の値 ＜ 親の値 ＜ 右の子の値 ）の条件を満たしている

・2分検索木のデータ検索

2分探索木の中にあるデータを最短の手順で探し出す

＜手順1＞根の値と検索目的の値を比較する（8と9を比較）
＜手順2＞検索目的の値が大きければ右の節へ、小さければ左の節へ移動する（右へ）
＜手順3＞移動先の節の値と検索目的の値を比較する（10と9を比較）
＜手順4＞同様な操作を検索目的の値が見つかるまで繰り返す（左へ）

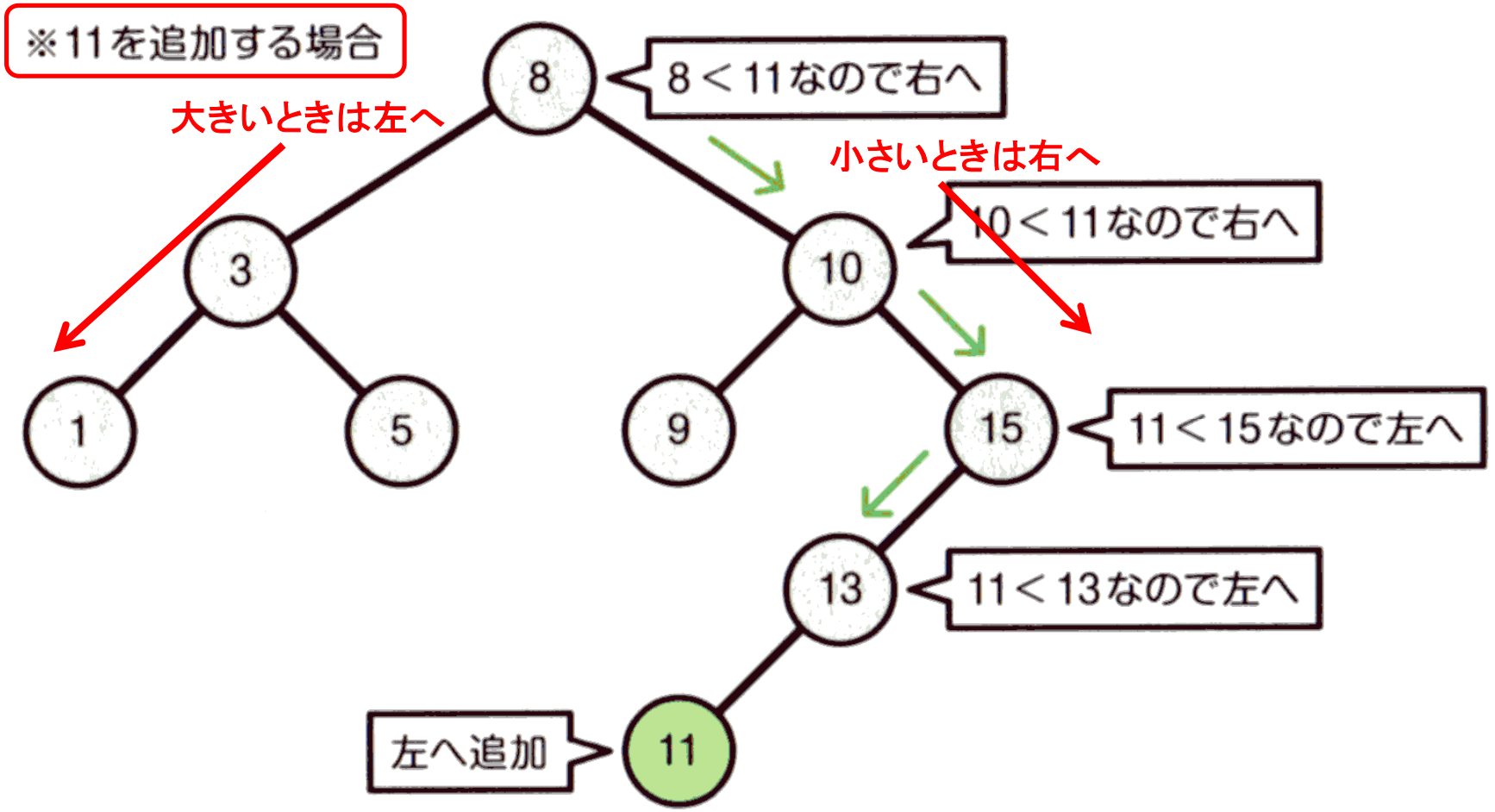
※赤字は図の場合



• 節の追加や削除による2分検索木の再編成

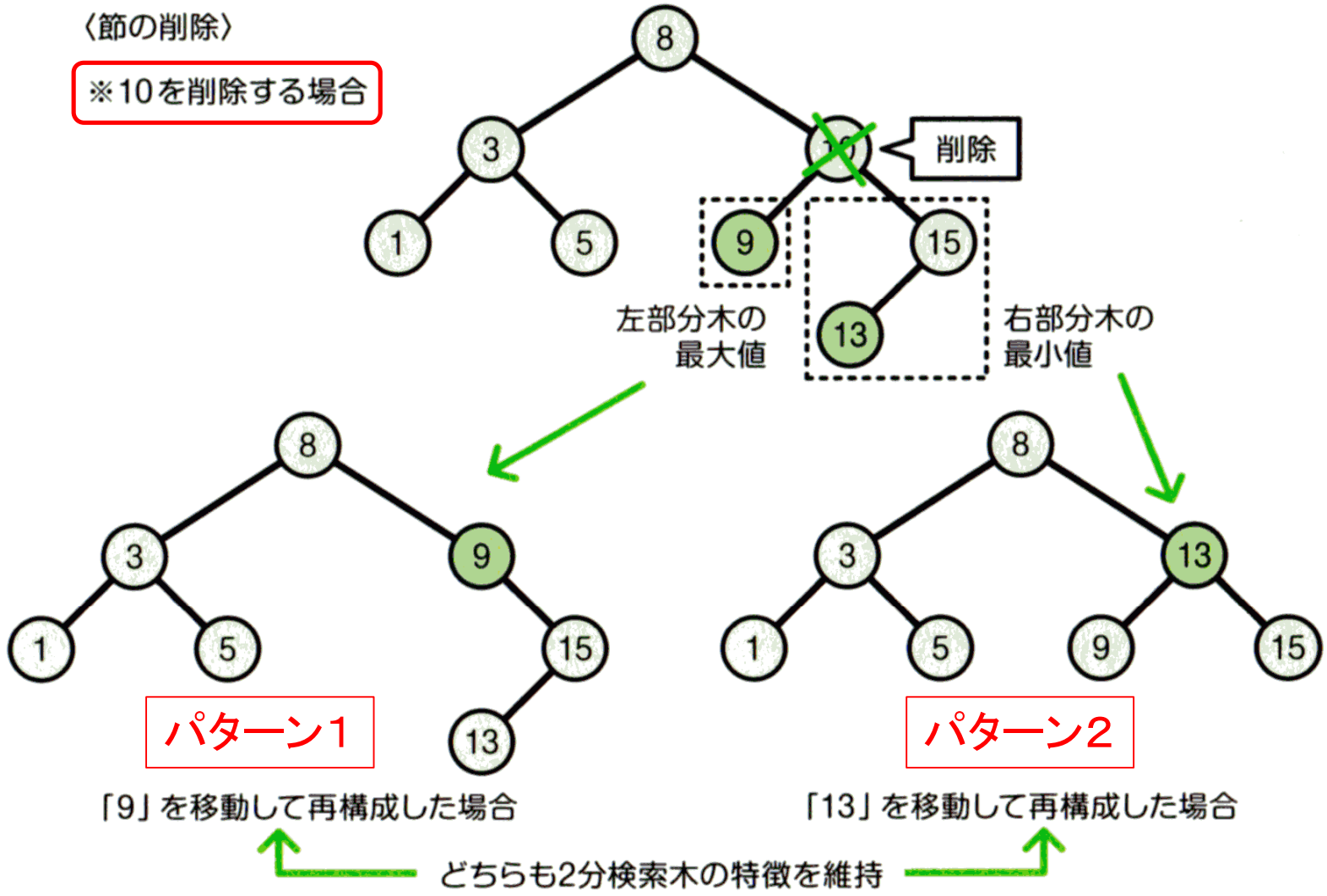
■ 追加する場合

データ検索と同様の操作で、正しい位置に追加できる



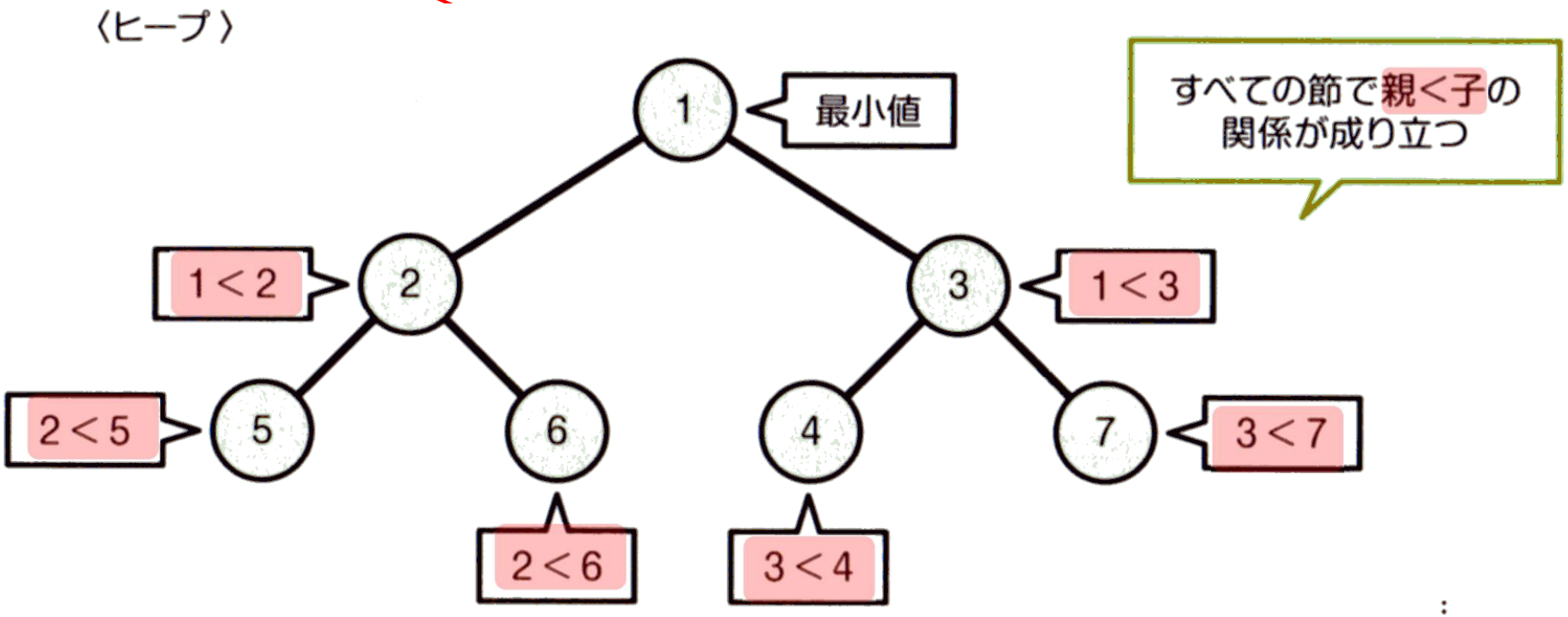
■ 削除する場合

2分検索木の性質(左の子の値 < 親の値 < 右の子の値)を損なわないように、木構造を再編成する



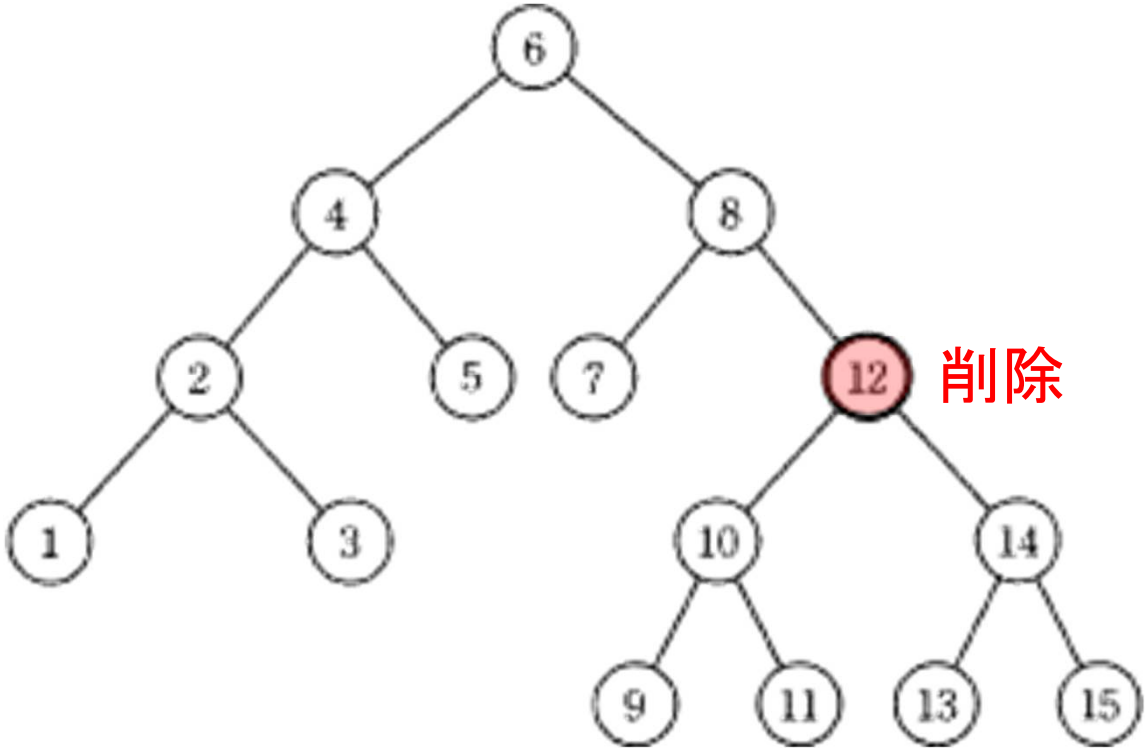
● ヒープ

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



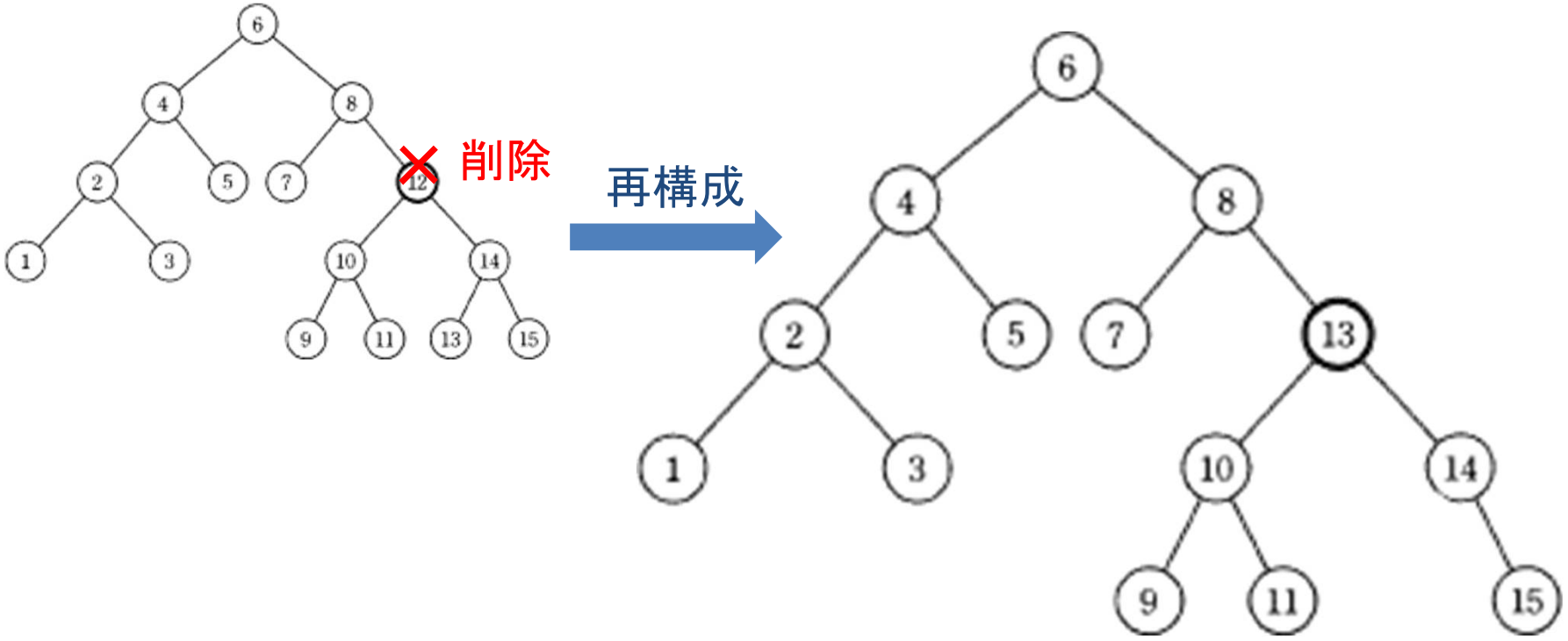
【過去問題】

次の2分探索木から要素12を削除したとき、その位置に別の要素を移動するだけで2分探索木を再構成するには、削除された要素の位置にどの要素を移動すればよいか。



ア. 9　　イ. 10　　ウ. 13　　エ. 14

2分探索木から要素12を削除すると、2分探索木の性質(要素同士の大小関係)を保つために、左部分木の最大値「要素11」、右部分木の最小値「要素13」を新たな節点として再構成する



【回答】
ウの節点となる要素13の左部分木に要素9、要素10、要素11、右部分木に要素14、要素15が配置されることになるので適切

★★★6.検索アルゴリズム

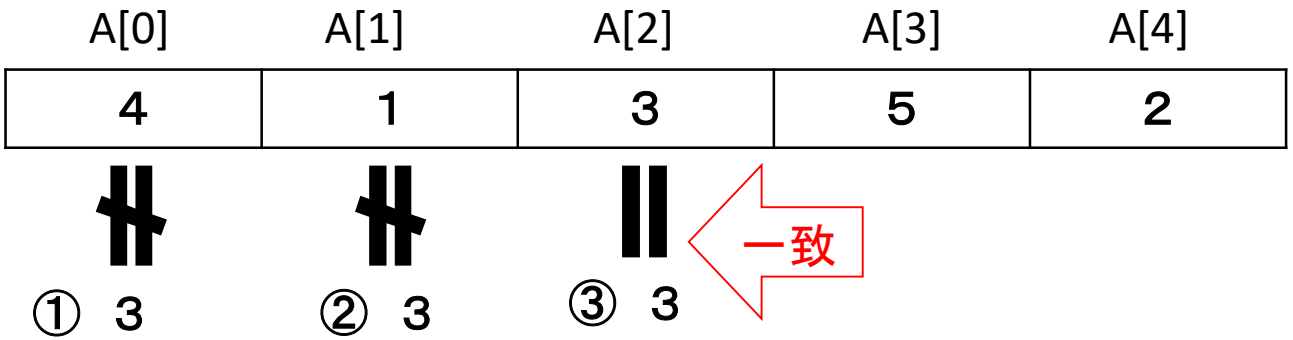
●検索とは

たくさんのデータの中から、目的のデータを探し出す

- 線形検索法（逐次探索法）

  先頭（左端）から順番に目的のデータと比較しながら、一致したデータを探す

目的のデータが、3の場合

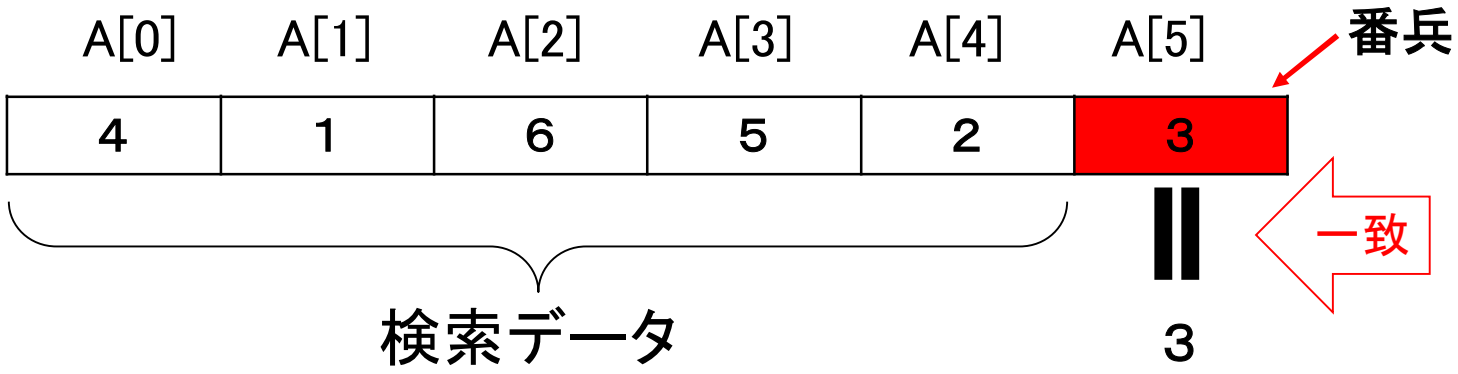


平均探索回数 ：（ｎ＋１）／２

ｎ＝データ個数

■線形検索法・番兵法（目的のデータがないとき）

検索データの最後に、あらかじめ（わざと）目的のデータを入れる



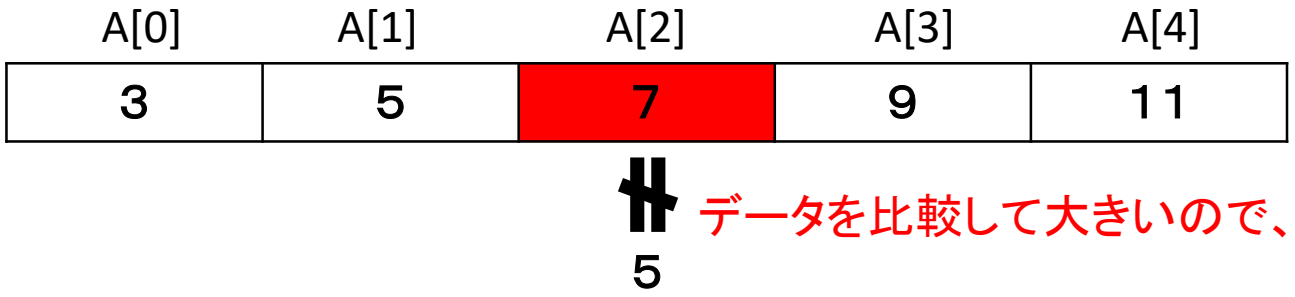
番兵として入れたデータが検索された場合は、
「検索できなかった」と判断する

- **2分検索法**

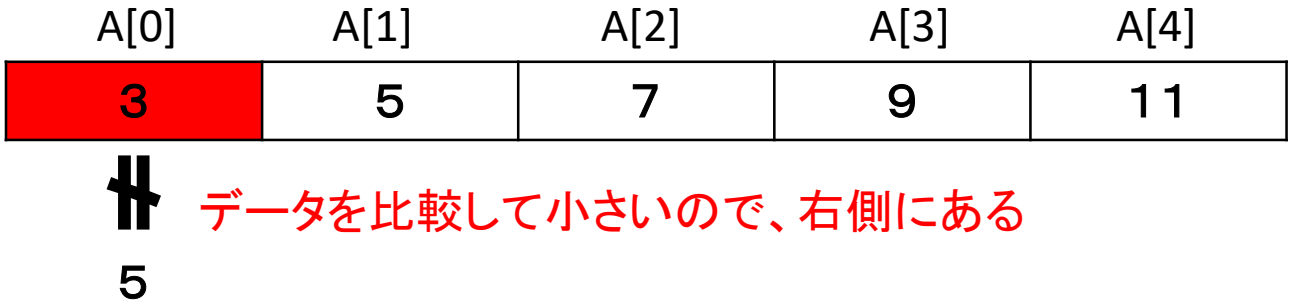
検索データがあらかじめ小さい順（昇順）や多い順（降順）に並べられている場合に使用する検索法

目的のデータが5で、小さい順に並べられている場合

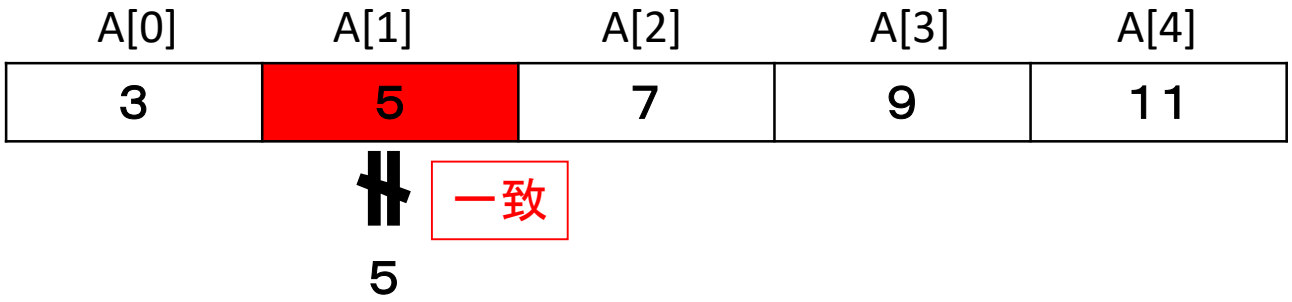
①検索データの真ん中に位置するデータと比較する



②検索位置を左半分に移して、再度真ん中に位置するデータと比較する



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する



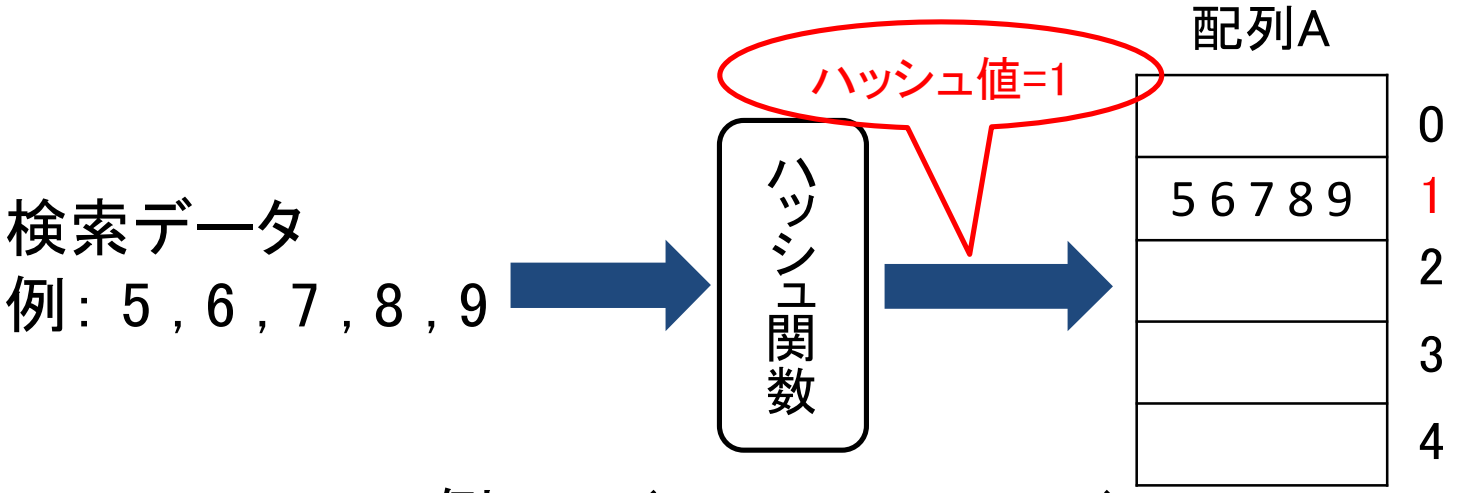
④目的のデータが見つからなければ、①～③の操作を繰り返す

平均探索回数 ： $log_2 n$

n = データ個数

● ハッシュ検索法

- データを格納する場所（アドレス値）を、予め決めた関数（ハッシュ関数）で求める
- 目的のデータを検索するときも、格納したときと同じハッシュ関数で格納場所（アドレス値=ハッシュ値）を求める



例：mod(5+6+7+8+9 , 17)
=mod(35,17)関数

平均探索回数 ： 1

【過去問題】

昇順に整列されたn個のデータが配列に格納されている。探索したい値を2分探索法で探索するときのおよその比較回数を求める式はどれか。

ア. Log_2n　　イ. $(log_2n+1)/2$

ウ. n　　エ. n^2

2分探索法は、探索データが昇順または降順に整列されているときに用いる方法。データ中央値と探索対象のデータを比較し、その大小により探索範囲を1／2ずつ狭めていくことで、線形探索と比べて効率よく探索することが可能。

平均比較回数:log_2n　　最大比較回数:log_2n+1

※nはデータ個数

★7. 整列（ソート）アルゴリズム

● 整列とは

■ 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

＊交換法（バブルソート）： 最大比較回数＝ $\frac{n(n-1)}{2}$

以下の数値を昇順に並べる

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

②2つの数値（3と1）を比較して、小さな数値（1）を左に置く（交換する）

③2つの数値（3と2）を比較して、小さな数値（2）を左に置く（交換する） ソート完了

＊選択法（選択ソート）　：　最大比較回数＝ $\frac{n(n-1)}{2}$

以下の数値を昇順に並べる

①左から順番に、2つの数値（3と2）を比較する　→　2の方が小さい

②2つの数値（2と1）を比較する　→　1の方が小さい

③最小値が1であることがわかったので、1を一番左に移動して、3を1の場所に移動する

ソート完了

＊挿入法（挿入ソート）

整列していないデータ列の中から、1つずつデータを取り出して、整列しているデータ列の適切な位置に、取り出したデータを挿入することを繰り返して、データを並べ替える

昇順に整列させる場合

①先頭（左端）のデータ（5）を整列済とする

5	3	1	4	2

②未整列データ（3）を整列済データ（5）と比較して、大きいデータが右に、小さいデータが左になるように、未整列データ（3）をデータ列に挿入する

3	5	1	4	2

③次の未整列データ（1）を整列済データ（5）と比較して、未整列データ（1）を左に挿入する

3	1	5	4	2

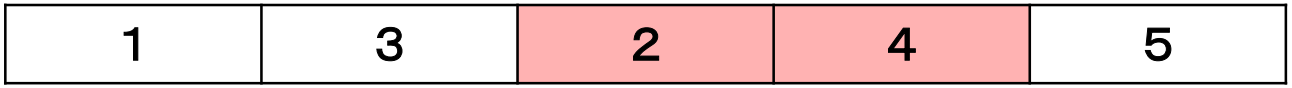
④整列済データ（3）と未整列データ（1）を比較して、未整列データ（1）を左へ挿入する。データ（1）の位置は先頭なので、このデータ（1）の位置は確定。

⑤次の未整列データ（4）を整列済データ（5）と比較して、未整列データ（4）を左に挿入する。

⑥整列済データ（3）と未整列データ（4）を比較する。未整列データ（4）はすでに整列済データ（3）の右なので、ここでは何もしないでこのまま。

⑦次の未整列データ（2）を整列済データ（5）と比較して、未整列データ（2）を左に挿入する。

⑧未整列データ（2）を整列済データ（4）と比較して、未整列データ（2）を左に挿入する。



⑨未整列データ(2)を整列済データ(3)と比較して、未整列データ(2)を左に挿入する。

1	2	3	4	5

⑩未整列データ(2)を整列済データ(1)と比較する。未整列データ(2)はすでに整列済データ(1)の右なので、ここでは何もしないでこのまま。

1	2	3	4	5

ソート完了

【過去問題】

配列A[i](i＝1,2,･･･,n)を、次のアルゴリズムによって整列する。行2～3の処理が初めて（1回目）終了したとき、必ず実現されている配列の状態はどれか。

[アルゴリズム]

行番号.

1. 行番号iを1からn－1まで1ずつ増やしながら行2～3を繰り返す
2. jをnからi＋1まで減らしながら行3を繰り返す
3. もしA[j]＜A[j－1]ならば, A[j]とA[j－1]を交換する

ア. A[1]が最小値になる　　イ. A[1]が最大値になる

ウ. A[n]が最小値になる　　エ. A[n]が最大値になる

例として“7315”の文字列(n＝4)を用いて整列の流れを考える

	1	2	3	4
A	7	3	1	5

＊行1:1から(4−1=)3まで増やしながら繰り返す

＊行2:i＝1，4から(1+1=)2まで減らしながら繰り返す

1. 行3:j＝4，A[4]とA[3]を比較。5＜1は成立しないので、交換はしない。
   [7315]

2. 行3:j＝3，A[3]とA[2]を比較。1＜3が成立するので、A[3]とA[2]交換する。
   [7135]

3. 行3:j＝2，A[2]とA[1]を比較。1＜7が成立するので、A[2]とA[1]交換する。
   [1735]

ここまでで行2～3の処理の1回目の処理が終了した。処理の結果、文字列の中で最小値である“1”が配列の先頭(A[1])に格納されていることが確認できる

★★★ 8. 再帰アルゴリズム

●同じパターンの繰り返し（再帰）アルゴリズム

自然数nの階乗n!を求める場合

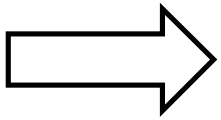
$$n! = n \times (n-1)!$$

同じパターン

5!を求める場合

計算式

```
f(5) = 5 x f(4)
     = 5 x 4 x f(3)
     = 5 x 4 x 3 x f(2)
     = 5 x 4 x 3 x 2 x f(1)
     = 5 x 4 x 3 x 2 x 1
     = 120
```



プログラム

```
f(n):
    if n = 1  then
        return 1
    else
        return n * f(n-1)
```

再帰関数

令和元年度 秋期
基本情報処理技術者試験問題・解答(ｱﾙｺﾞﾘｽﾞﾑとﾃﾞｰﾀ構造)

【問11 】

自然数nに対して，次のように再帰的に定義される関数f(n)を考える。
f(5)の値はどれか。

f(n):if n≦1 then return 1 else return n＋f(n－1)

ア 6　　イ 9　　ウ 15　　エ 25

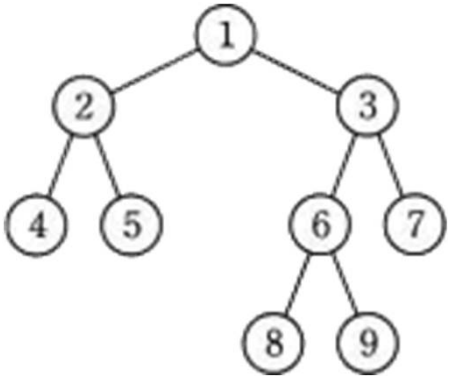
引数nが1以下のとき
　　1を返す
それ以外の場合
　　n＋f(n－1)を返す

f(5)
＝5＋f(4)　//f(5)=5＋f(4)
＝5＋4＋f(3)　//f(4)=4＋f(3)
＝5＋4＋3＋f(2)　//f(3)=3＋f(2)
＝5＋4＋3＋2＋f(1)　//f(2)=2＋f(1)
＝5＋4＋3＋2＋1　//f(1)=1
＝15

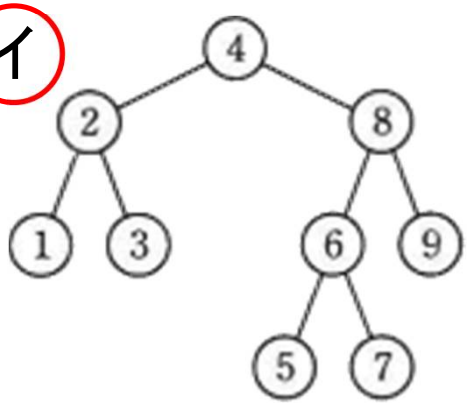
平成31年度 春期
基本情報処理技術者試験問題・解答(ｱﾙｺﾞﾘｽﾞﾑとﾃﾞｰﾀ構造)

【問5】
2分探索木として適切なものはどれか。ここで、1～9の数字は、各ノード(節)の値を表す。

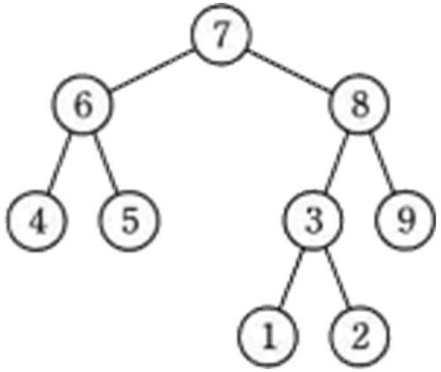
ア



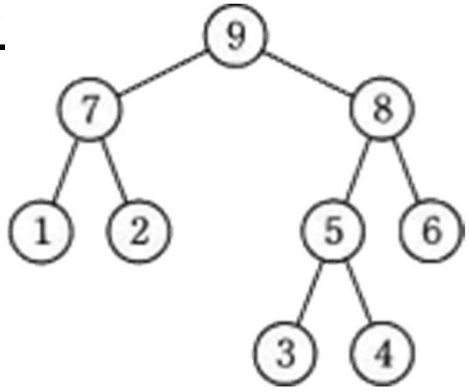
イ



ウ



エ



【解答】

2分探索木は、2分木の各節にデータを持つことで探索を行えるようにした木構造である。各節のデータは「その節から出る左部分木にあるどのデータよりも大きく、右部分木のどのデータよりも小さい」という条件があり、これを利用して効率的にデータを探索することができるようになる。

「左部分木の値 ＜ ノードの値 ＜ 右部分木の値」という2分探索木の条件に照らして選択肢の木構造を検証すると、2分探索木として適切な木は「イ」と分かる。