アルゴリズムとデータ構造 並びに同演習

～第9回 再帰的アルゴリズム～

総合情報学専攻
メディア情報学専攻
橋本直己
naoki@cs.uec.ac.jp

再帰とは？

- ある事象は，それが自分自身を含んでいたり，それを用いて定義されているときに，再帰的（recursive）であるという．

- 例）自然数の再帰的定義
  （recursive definition）
  - （a）1は自然数である
  - （b）ある自然数の直後の整数も自然数である

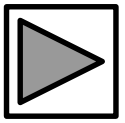
※ 簡潔かつ効率的な表現が可能（プログラムでも）

階乗値

- 整数nの階乗は，以下のように再帰的に定義される
  - 階乗n!の定義（nは非負整数）
    - (a)0 != 1
    - (b)n > 0 ならば n! = n × (n − 1)!
  - 例）10の階乗 10! = 10 × 9! = 10 × 9 × 8!

演習9-1

- prog9-1.cを実行し，再帰的なプログラムの動作を確認せよ.



再帰関数呼出し(1)

- factorial(3) = 3 × factorial(2)
  = 3 × 2 × factorial(1)
  = 3 × 2 × 1 × factorial(0)
  = 3 × 2 × 1 × 1

- 関数factorialは，行うべき計算を実現する為に，関数factorialを呼び出す

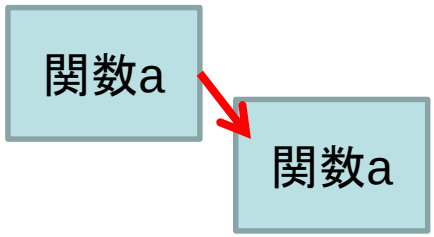
→ 再帰関数呼出し(recursive function call)

再帰関数呼出し(2)

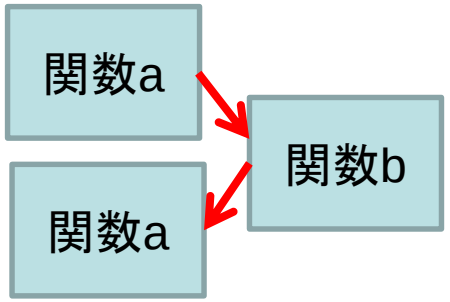
- 再帰関数呼出しは「“自分自身の関数”を呼び出す」と考えるよりも，「“自分自身と同じ関数”を別途呼出す」と考えた方が自然
  - 自分自身を呼び出したら，延々と自分を呼び出し続けて無限ループしてしまう

直接的な再帰と間接的な再帰

- 階乗を求める関数factorialは，直接factorialを呼び出す＝直接的な（direct）再帰と呼ぶ
- 関数aが関数bを呼び出し，その関数bが関数aを呼び出す＝間接的な（indirect）再帰と呼ぶ．



直接的な再帰



間接的な再帰

再帰的アルゴリズムの適用

- 再帰的アルゴリズムが適しているのは，解くべき問題や計算すべき関数，あるいは処理すべきデータ構造が再帰的に定義されている場合である．

- 再帰的手続きによって階乗数を求めるのは，あくまでも解説のためであり，現実的には適切ではない．

演習9−2

- prog9−1.cで使用したfactorial関数を，再帰関数呼び出しを用いないように修正せよ
  - ヒント：再帰的呼び出しの代わりに、while文を使用せよ

ユークリッドの互除法

- 二つの整数値の最大公約数(greatest common divisor)を求める.

【利用する特性】

- 整数 a, b とすると「a = bk + r」と表せる
- aとbの最大公約数をG，bとrの最大公約数をG’とすると，上式より，Gはrの約数でなければならない（G≦G’）
- 逆に、bとrの最大公約数G’は、上式よりaの約数、つまりG≧G’
- よって，『aとbの最大公約数G』=『bとrの最大公約数G’』

アルゴリズムで表現すると

整数a, bの場合：

まずaをbで割ってみる

1. 割り切れた場合：bが最大公約数
2. 割り切れなかった場合：
   - r = a % b （aをbで割った余り）
   - bとrの最大公約数を求める(再帰)

※ aとbの大小関係は無視してもOK

a<bの場合：a % b = a, よってbとaの最大公約数を求めることになる（自動的にaとbが逆になる）

演習9−3

ユークリッドの互除法を用いて、二つの整数の最大公約数を求めるプログラムを、再帰関数呼び出しを用いて作成せよ。

prog9-3の動作

- 例：整数 14と6の最大公約数(gcd)を求める

1. 14 / 6 = 2 余り 2, よって6はgcdではない
2. 再帰呼出：6と2のgcdを求める
3. 6 / 2 = 3 余り 0, よって2がgcdである.

故に, 14と6のgcdは2である

再帰アルゴリズムの解析（演習9−4）

- 再帰プログラムの動作を解析し，再帰に関する理解を深めましょう.

- まずprog9−4.c 内の 関数 recurを読解せよ.

補足：関数recur内では，再帰呼出しを2回行っている.
2回以上再帰呼出を行う関数は，真に(genuinely) 再帰的であると呼ばれる.

prog9-4.cの実行結果

- 真に再帰的な関数は，短くても複雑な挙動を示します．
- 右例では，入力4の場合の出力を示しています．
- 入力が3や5の場合の結果を想像することは困難．

```
> ./prog9-4
整数を入力せよ:4
1
2
3
1
4
1
2
>
```

【実行結果】

以降では，この関数の挙動を解析します．

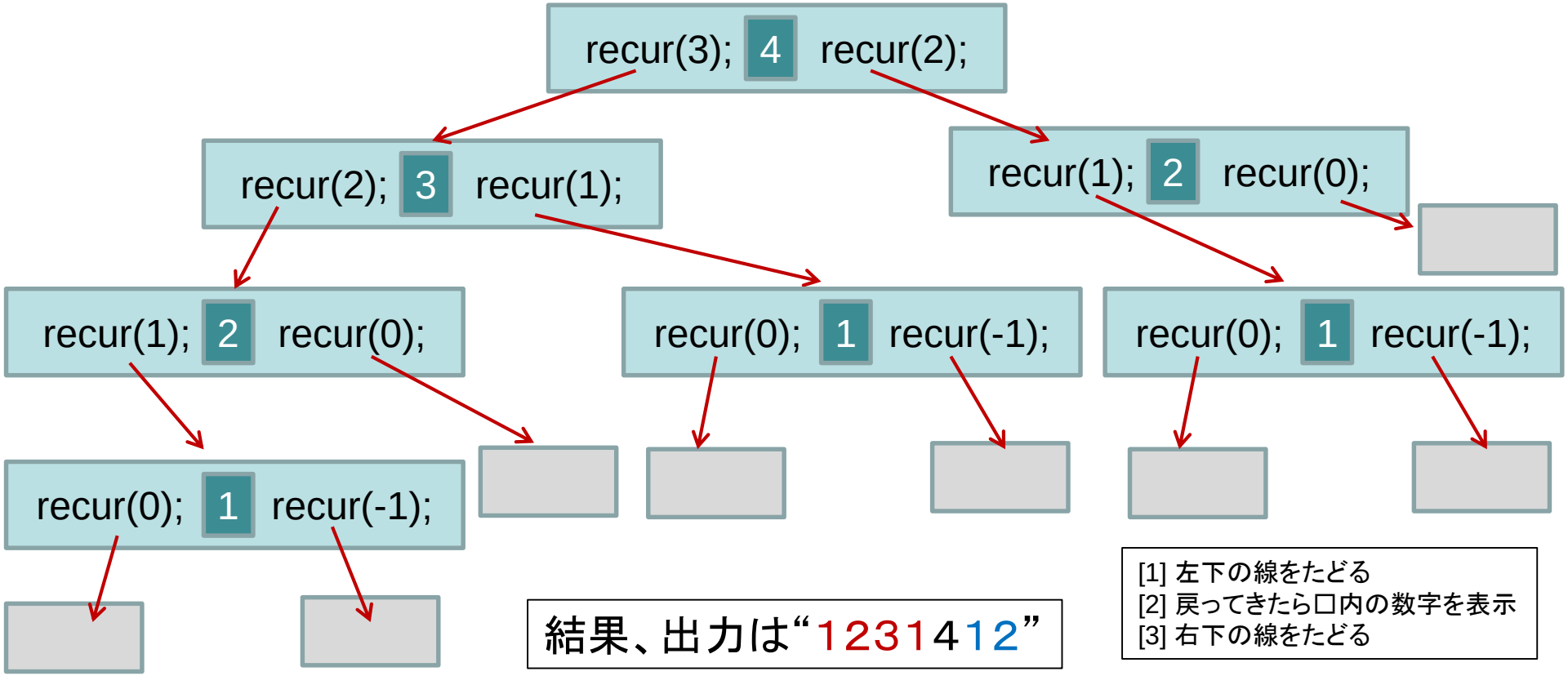
トップダウン解析

- 仮引数nに4を受け取った関数recurの動作

（a）recur (3)
（b）4を出力
（c）recur (2)

- (a), (b), (c)の順番に実行されるが，“(b)の4”の前に何が出力されるかは，(a)を実行してみなければ分からない
- なので，プログラムの実行順に処理を追いかけてみる（＝トップダウン解析）

トップダウン法（top-down method）



- この図をたどると，プログラム全体の出力が分かる
- しかし無駄も多い（recur(1), recur(0)の重複）

ボトムアップ解析(1)

- 上から解析することが効率的とは限らない
- 次は，下から積み上げる方法で解析

- まず，nが正であるときのみ具体的な動作をするので，recur(1)について考える

（a）recur (0)
（b）1を出力
（c）recur (-1)

  - recur(0), recur(−1)は何もしないので，結果recur(1)は1を出力するだけと分かる.

ボトムアップ解析(2)

- 次に，recur(2)について考える．

  （a）recur (1)
  （b）2を出力
  （c）recur (0)

  – recur(1)は1を出力するので，結果“12”を出力

- これをrecur(4)まで積み上げていくことで，recur(4)の出力を知ることができる．

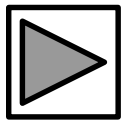
ボトムアップ法

- recur(0): 何もしない
- recur(1): recur(0), [1], recur(−1) → 1
- recur(2): recur(1), [2], recur(0) → 12
- recur(3): recur(2), [3], recur(1) → 1231
- recur(4): recur(3), [4], recur(2) → 1231412

ボトムアップ解析結果

演習9-5

- 以下に示す関数recur2のトップダウン解析およびボトムアップ解析を行え
  - 仮引数 n = 4 とせよ
  - 紙に書く or Text Editor



```
void recur2(int n)
{
    if ( n > 0 ) {
        recur2( n - 2 );
        printf("%d¥n", n);
        recur2( n - 1 );
    }
}
```

ここまでのまとめ

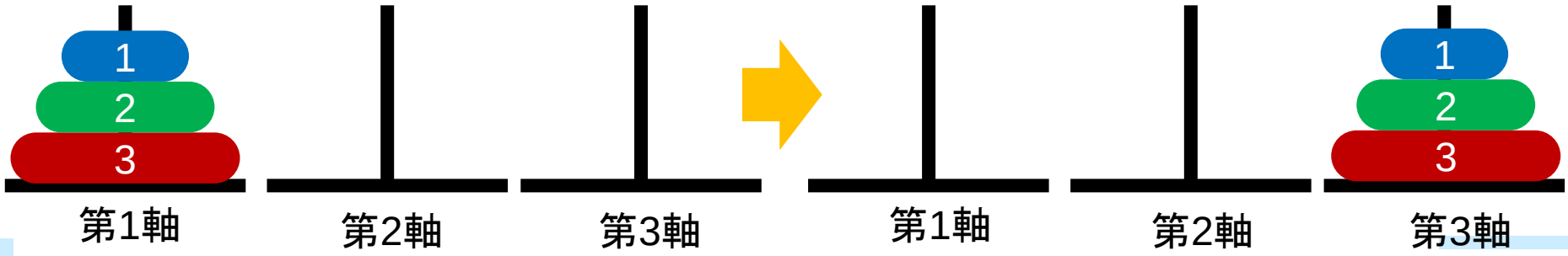
- 以下は非常に重要！！
  - 再帰関数呼び出し
    - ユークリッドの互除法
  - 再帰アルゴリズムの解析
    - トップダウン解析
    - ボトムアップ解析

本日の演習課題

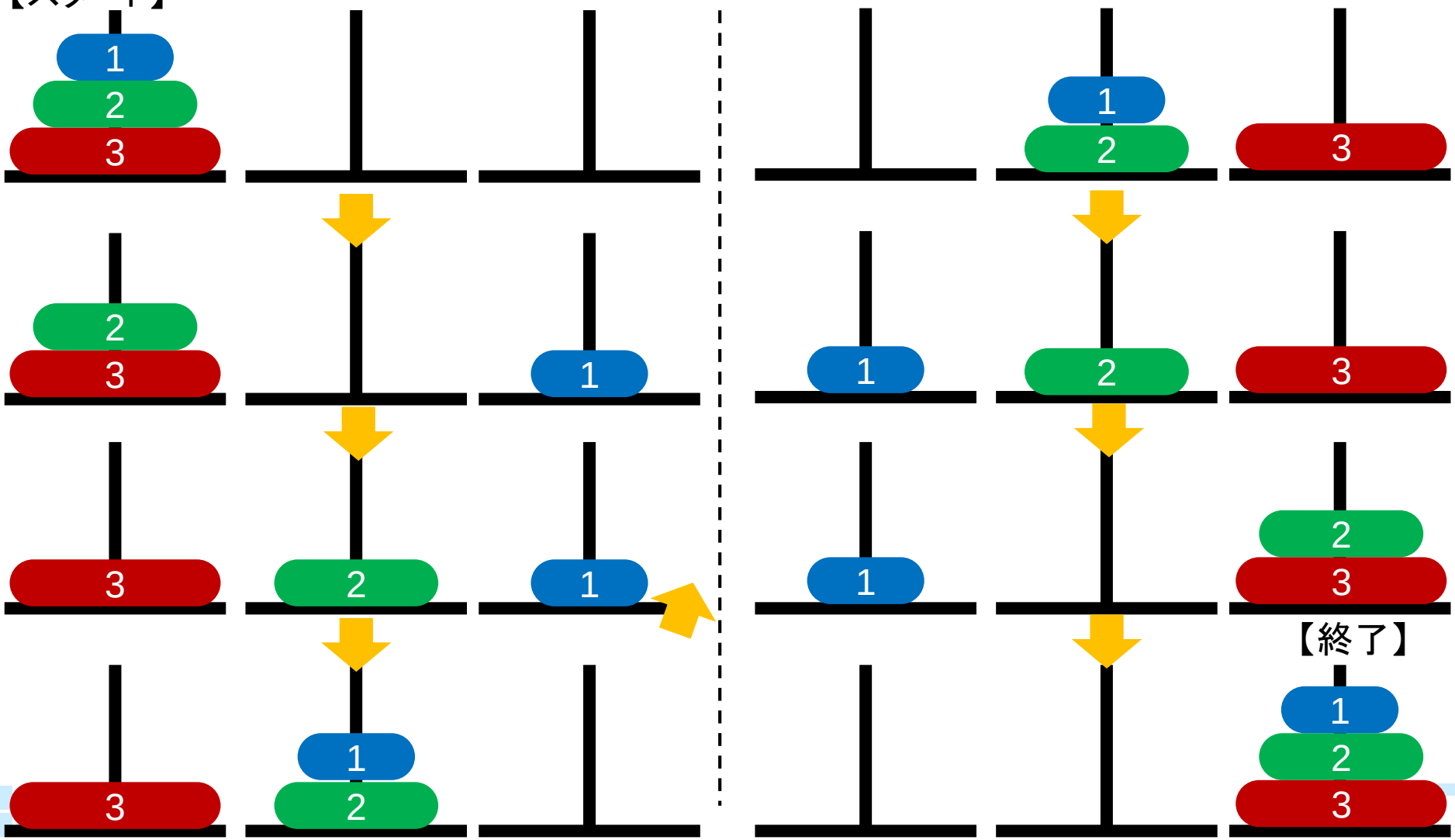
- 有名な「ハノイの塔」と「8王妃問題」を取り上げます。
- 講義中の説明に基づいて、演習時間に実装してもらいます。

ハノイの塔

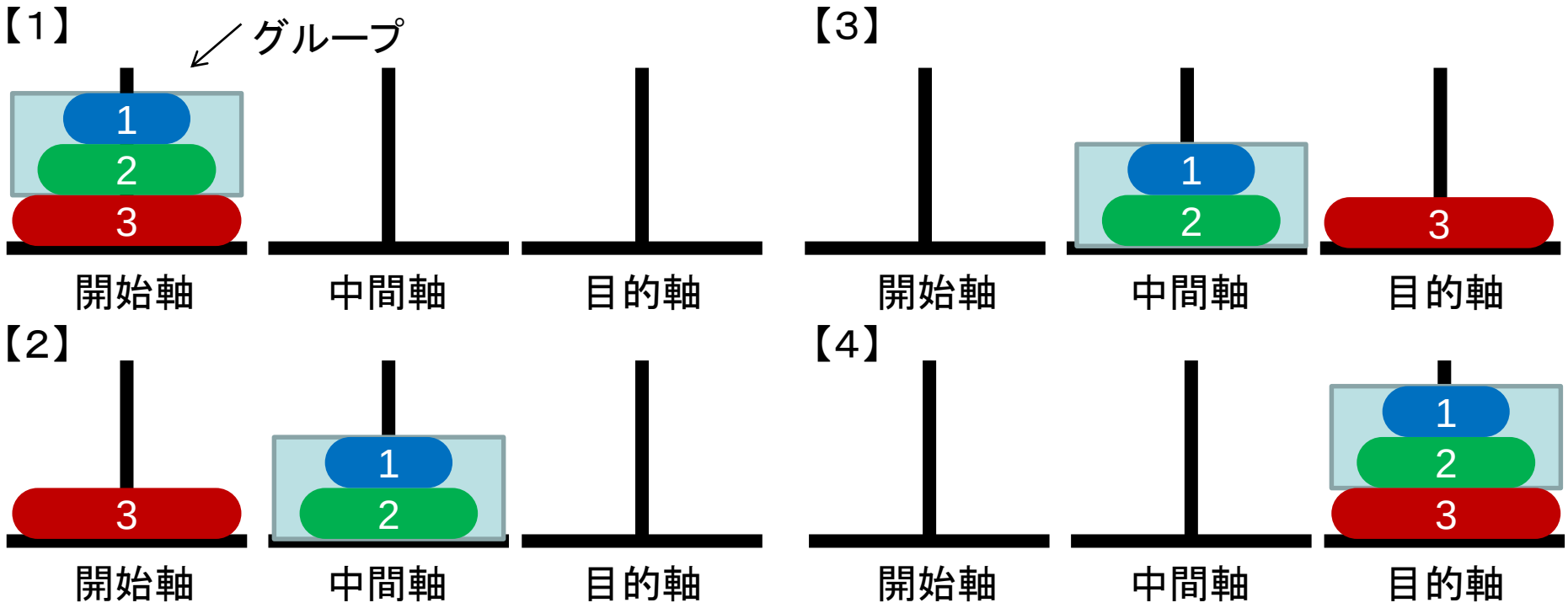
- ハノイの塔(towers of Hanoi)は，重なった円盤を3本の柱の間で移動する問題.
  - 全ての円盤の大きさは異なる
  - 初期は第1軸上．これを第3軸上に移動する
  - 移動は一枚ずつ
  - より大きな円盤を上に重ねることはできない



円盤が3枚のときの解法



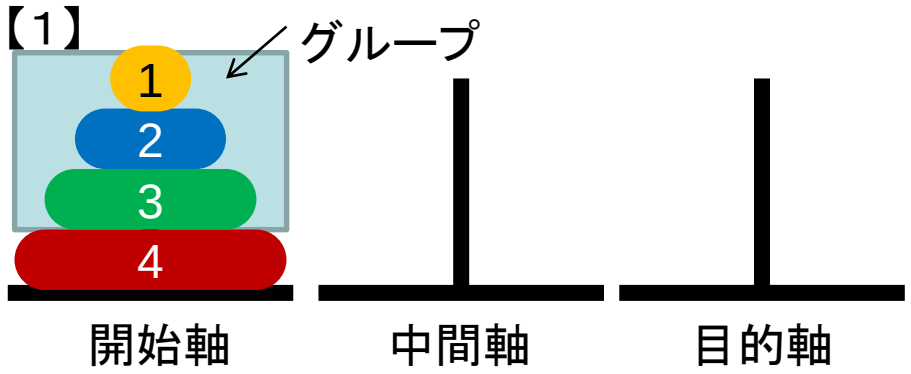
ハノイの塔の考え方(円盤が3枚)

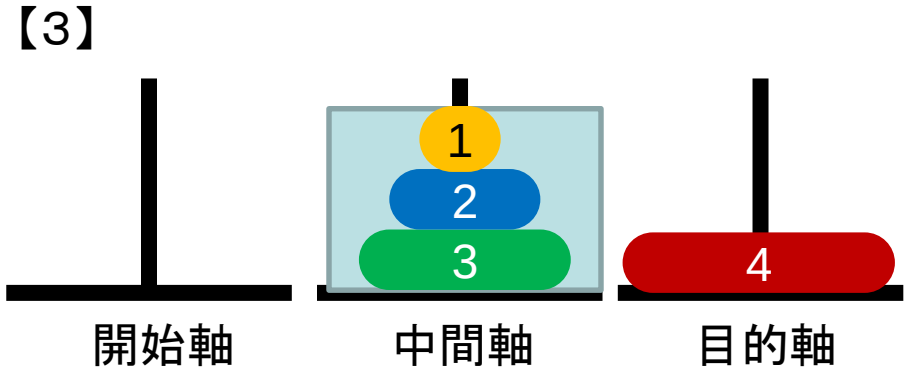


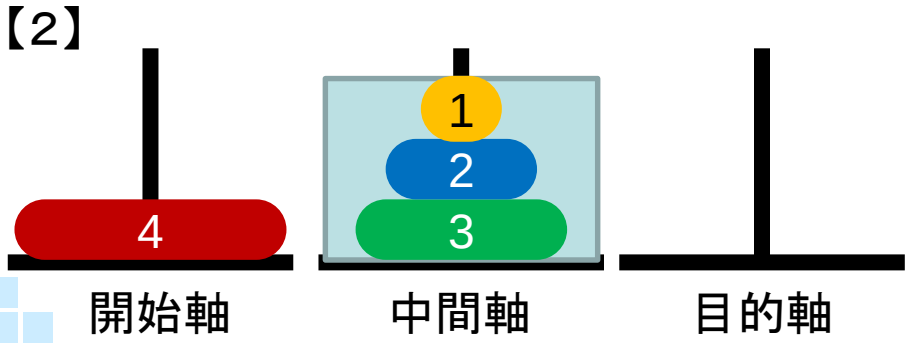
- 最大の円盤を最短のステップで目的軸へ移動する為には，上に乗っている円盤群（グループ）を、いったん中間軸に移せばよい.

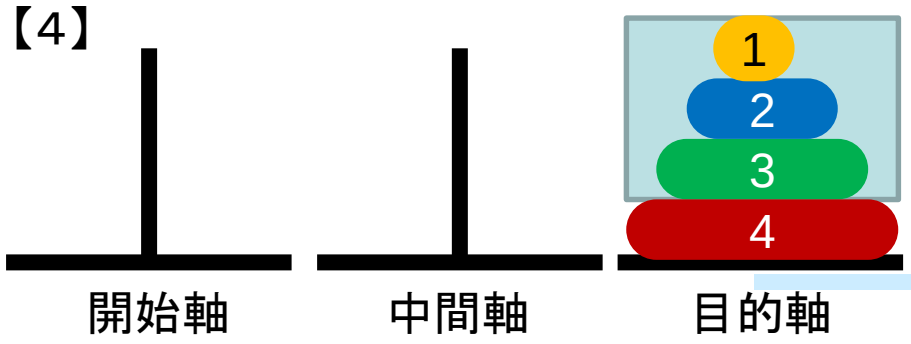
円盤が4枚になると

- 円盤１～３を重ねたものをグループとする.
- グループが大きくなるが，動かし方は前スライドで紹介済み



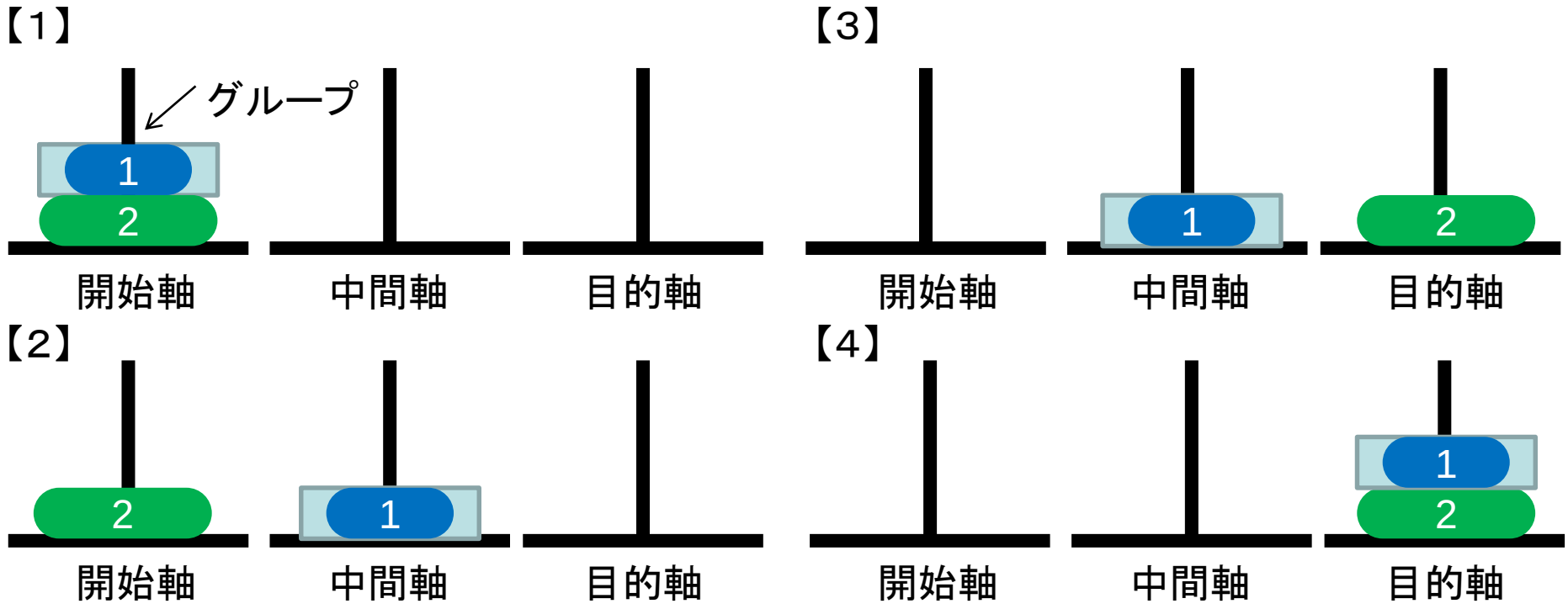






円盤が2枚になると

- 円盤1だけをグループとして考えると，円盤が3枚のときと同じ考え方で実現出来る．



つまり

- そこに置かれている最も大きい円盤以外の円盤を“グループ”とみなせば，円盤の枚数とは無関係に，全く同じ手続きで実現可能.

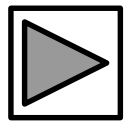
円盤がn枚の時にアルゴリズム

1. 底の円盤を除いたn−1枚の円盤を，開始軸から中間軸へ移動
2. 底の円盤を開始軸から目的軸へ移動
3. 底の円盤を除いたn−1枚の円盤を，中間軸から目的軸へ移動

演習では、これをプログラムで実現してみよう！

演習9-6

- 以上の考えに基づいてハノイの塔を実現するプログラム prog9-6.c を完成させよ.



【説明】

- 関数moveは再帰的に定義せよ(前ページ参照)
- 関数moveの引数noは移動すべき円盤の枚数
- xは開始軸、yは目的軸
- 軸は整数値1, 2, 3で表す.
  - このとき, 開始軸・目的軸がどの軸であっても, 中間軸は6 − x − y として求められる

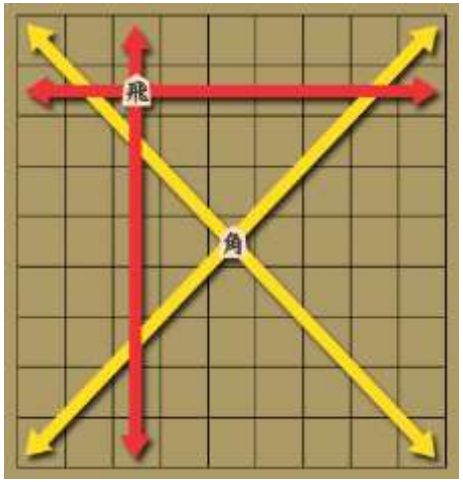
8王妃問題(8-Queen Problem)

8×8のチェス盤において，8個の王妃を互いに取り合うことのないように配置せよ.

- チェスの王妃は，将棋での飛車と角の動きを併せ持っている.
- つまり，縦・横・斜めのライン上のコマをとることができる.

チェス

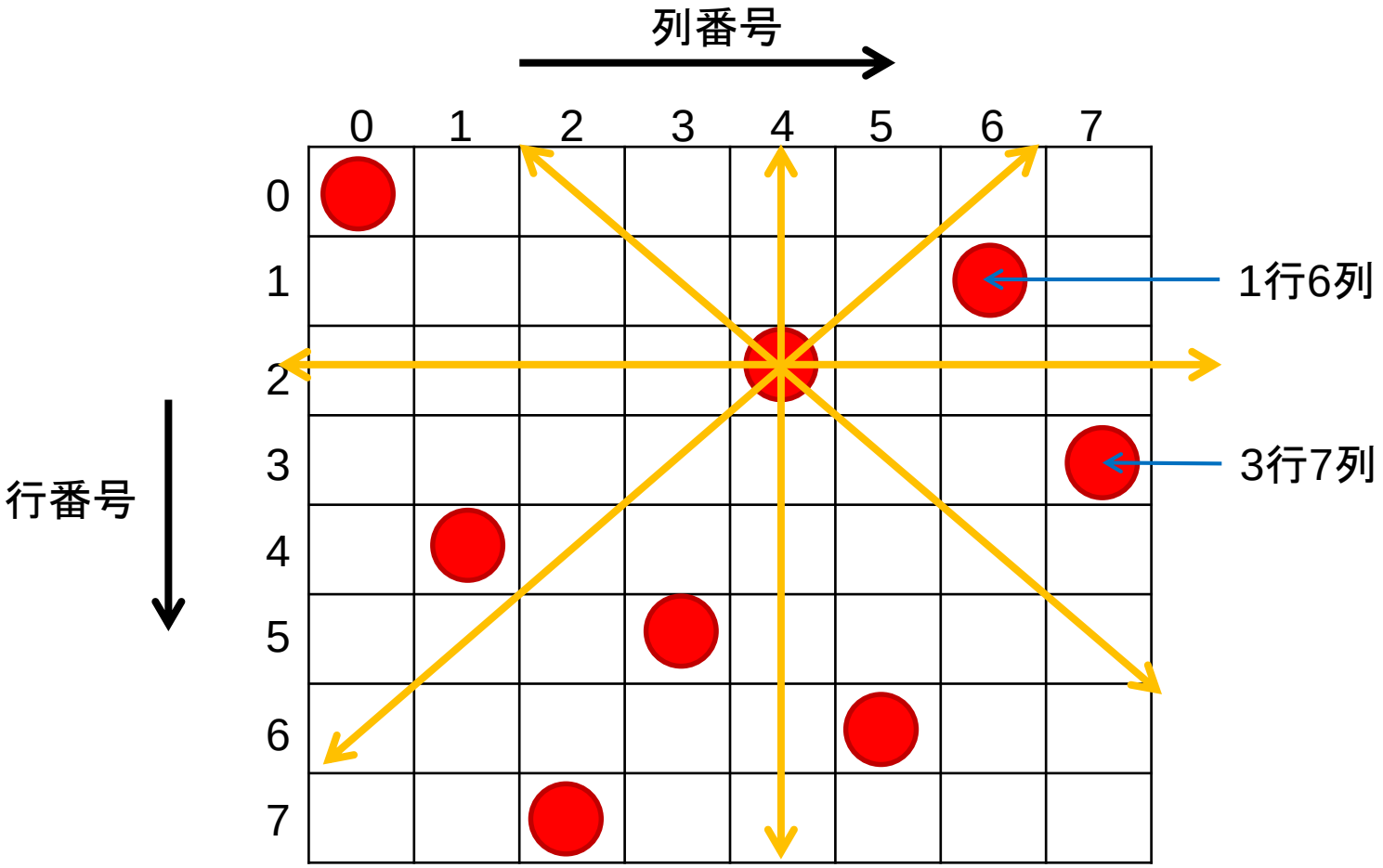
将棋



飛車と角の動き

解答の一例

- 問題の解答は複数ある．以下はその一例．



王妃の配置(1)

- チェス盤は8×8すなわち64個のマス
- 8個の王妃を置く組合せは

  64 × 63 × ･･･ × 57 = 178,462,987,637,760通り

  ※ 全探索は非現実的

王妃は列（縦）方向のコマを取れるため

【方針1】

各列には王妃を1個だけ配置する.

- 組合せは 8×8×･･･ × 8 = 16,777,216通り

※ 激減するが，それでもまだ多い

王妃の配置(2)

王妃は行（横）方向のコマを取れるため

【方針2】

各行に王妃は1個だけ配置する

$8 \times 7 \times \cdots \times 1 = 40{,}320$通り

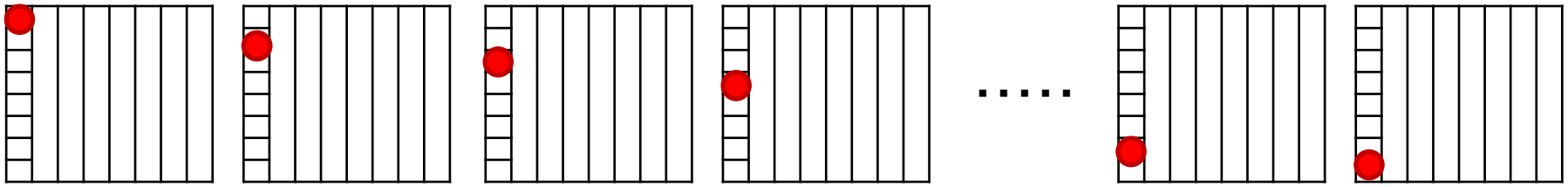
- それでも，解答を探すのはとても大変
- そこで方針1に戻って，可能性のある組合せを列挙するアルゴリズムを考える

（計算機の力を活用）

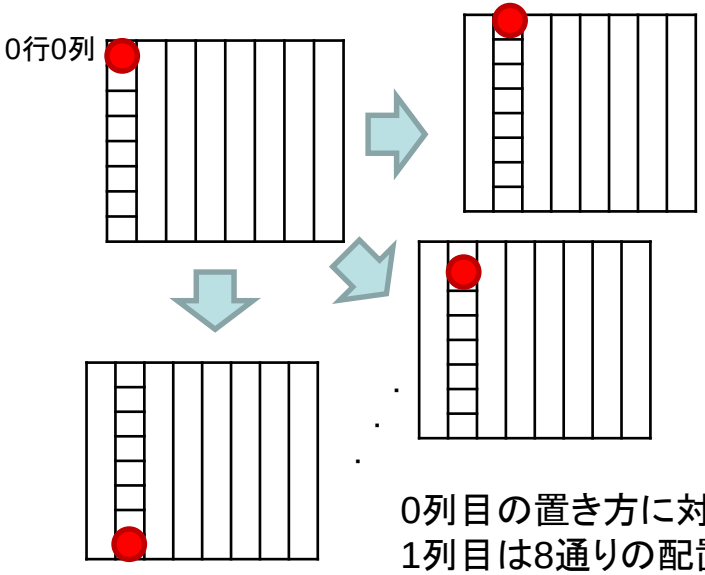
組合せの列挙(方針1)

各列に1つだけ王妃を置ける

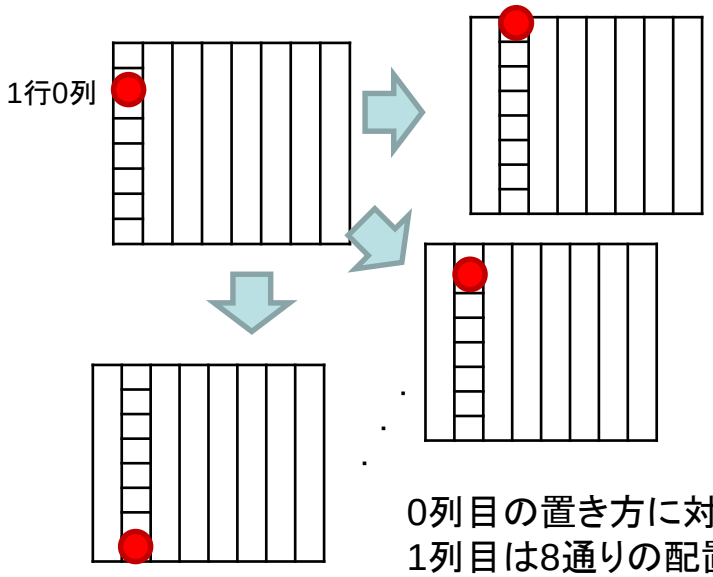
- 0列目の配置を決める(8通り)



- 1列目の配置を決める(8通り)



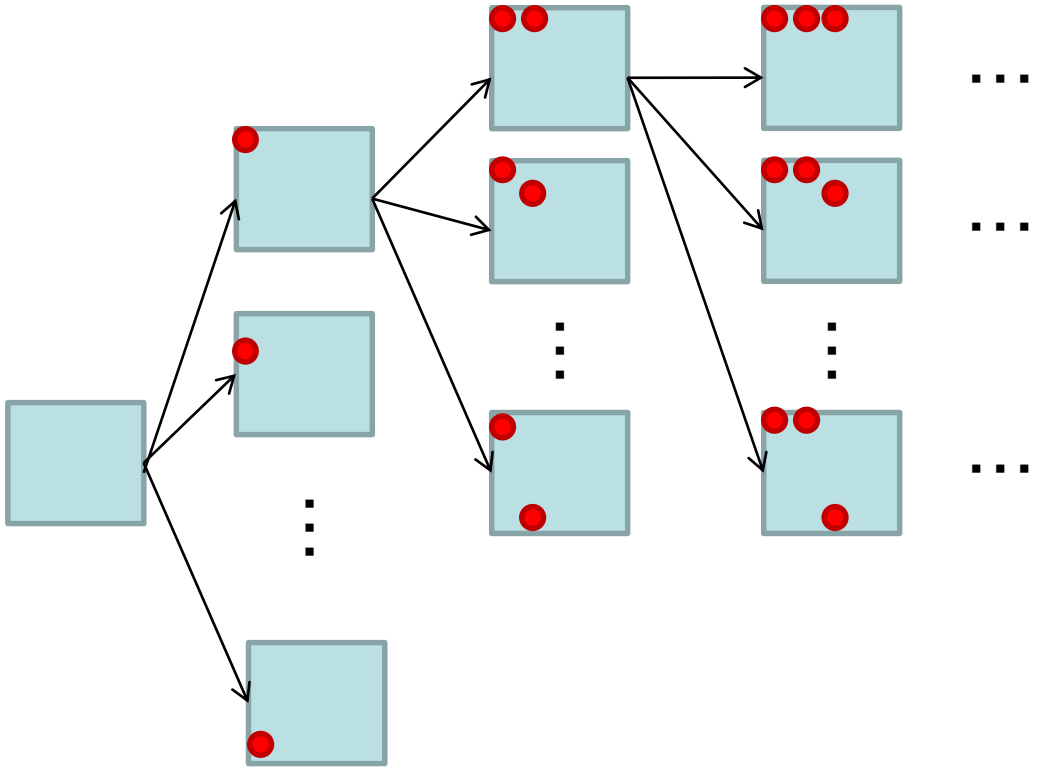
0列目の置き方に対して
1列目は8通りの配置



0列目の置き方に対して
1列目は8通りの配置

分岐操作

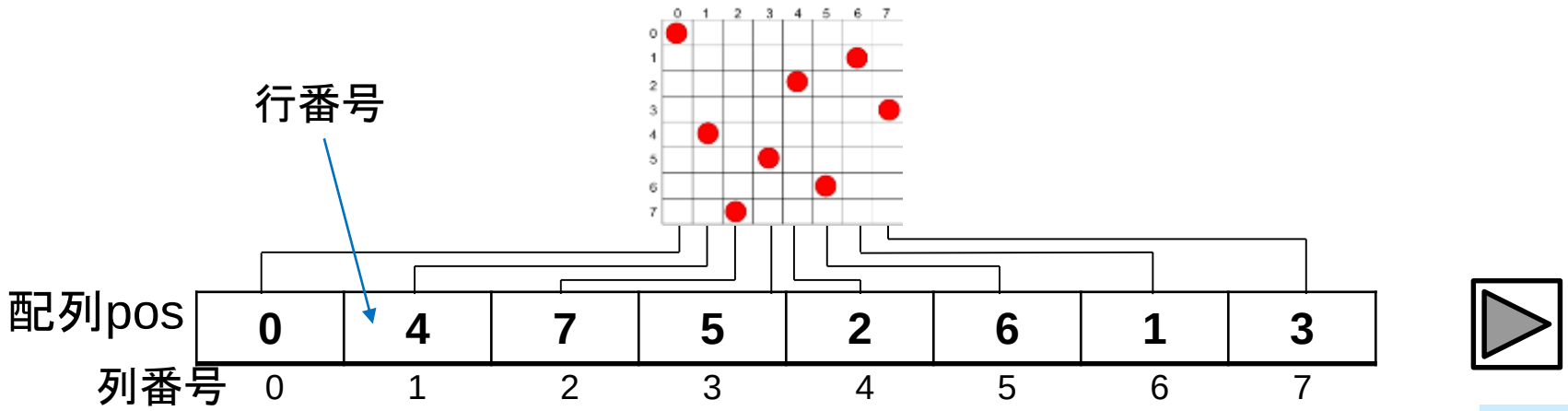
- 以上のように，どんどん枝分かれを行っていく操作を分岐(branching)と呼ぶ．



【方針1】を満たした
組み合わせを全て列挙

演習9−7

- 分岐操作によって組合せを列挙するプログラム prog9−7.c を実行し，その内容を読解せよ．
  - 配列posは，各列において，何行目に王妃が配置されたのかを格納する



実行結果

```
> ./prog9-7
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 2
 0 0 0 0 0 0 0 3
 0 0 0 0 0 0 0 4
 0 0 0 0 0 0 0 5
 0 0 0 0 0 0 0 6
 0 0 0 0 0 0 0 7
 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 1
 0 0 0 0 0 0 1 2
 0 0 0 0 0 0 1 3
 0 0 0 0 0 0 1 4
（以下、省略）
```

補足：分割統治法

- ハノイの塔や，8王妃問題のように，問題を小問題（部分問題）に分割し，小問題の解を統合して全体の解を得ようとする方法を分割統治法（*divide and conquer*）と呼ぶ.

限定操作

- 分岐操作を行っても，組合せを列挙するだけであり，8王妃問題を解くことはできない（当然）

- 次に，【方針2】である，

「各行には王妃を1個だけ配置する」

という考えを組み入れる．

演習9-8

- prog9-7.cに，方針2を加えて，組合せを列挙するプログラムを作成せよ

- 条件：
  - j行に王妃が配置されていれば，flag[j] = 1とすることで，その行に重複して王妃を配置できないようにせよ.
  - int flag[8];

解説

```
void set(int i)
{
 int  j;

 for(j = 0; j < 8; j++) {

    pos[i] = j;
    if (i == 7 )    /* 全列に配置終了 */
     print();
    else

     set(i + 1);




 }
}
```

【方針1】

```
void set(int i)
{
 int  j;

 for(j = 0; j < 8; j++) {
  if (!flag[j]) {   /* j行には王妃は未配置 */
    pos[i] = j;
    if (i == 7 )    /* 全列に配置終了 */
     print();
    else {
     flag[j] = 1;
     set( i+1 );
     flag[j] = 0;
    }

  }
 }
}
```

重要

【方針1＋方針2】

実行結果

```
＞./prog9-8
 0 1 2 3 4 5 6 7
 0 1 2 3 4 5 7 6
 0 1 2 3 4 6 5 7
 0 1 2 3 4 6 7 5
 0 1 2 3 4 7 5 6
 0 1 2 3 4 7 6 5
 0 1 2 3 5 4 6 7
 0 1 2 3 5 4 7 6
 0 1 2 3 5 6 4 7
 0 1 2 3 5 6 7 4
 0 1 2 3 5 7 4 6
 0 1 2 3 5 7 6 4
 0 1 2 3 6 4 5 7
 0 1 2 3 6 4 7 5
 0 1 2 3 6 5 4 7
（以下、省略）
```

分岐限定法

- 必要のない分岐操作を省略するための手法を限定操作(*bounding*)と呼び，分岐操作と合わせて分岐限定法(*branching and bounding method*)と呼ぶ.

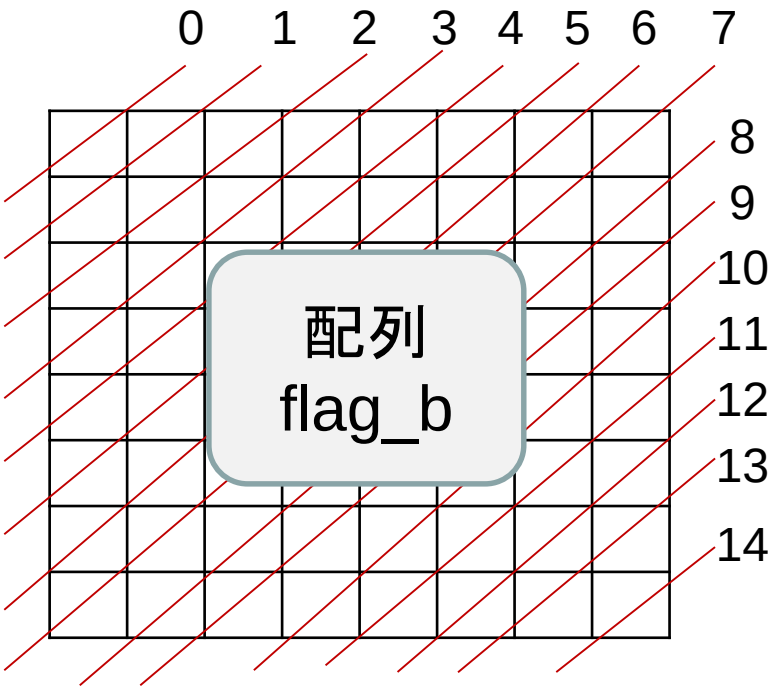
８王妃問題のための分岐限定操作

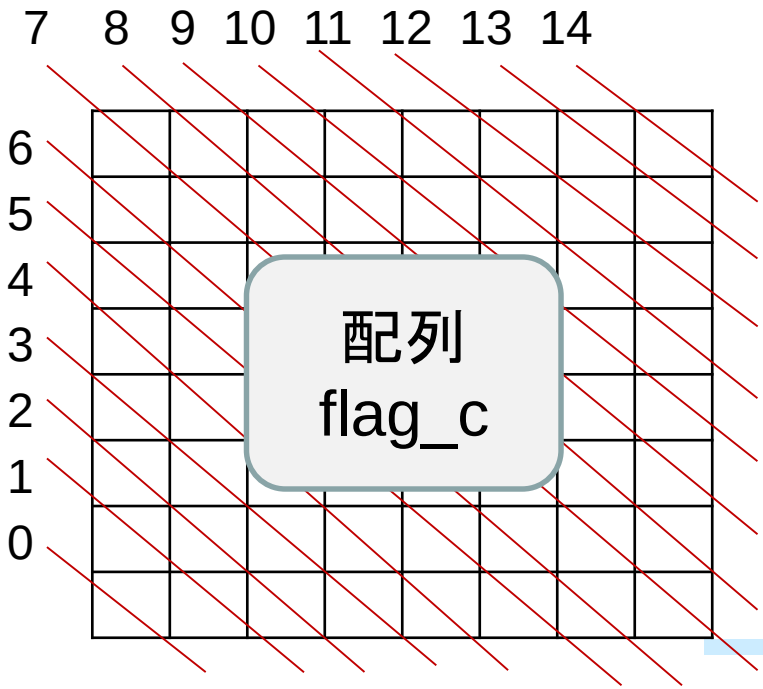
- 王妃は斜め方向のコマをとることができる

- prog9-8.cに，どの斜めライン上にも王妃を1個だけしか配置できないことを限定操作として追加することで，8王妃問題が解ける．

新たなflagの追加

- 配列flag_bとflag_cは，／方向および＼方向の対角線上に王妃が配置されているかどうかを示す





演習9-9

- prog9-8.cにflag_bとflag_cを導入して，8王妃問題の解を列挙するプログラムを作成せよ.

  （prog9-8におけるflagはflag_aに改名せよ）

ヒント

i列 j行目の場所に着目しているとき:

- 限定：各行で王妃は1つ　　【If文内での条件式】

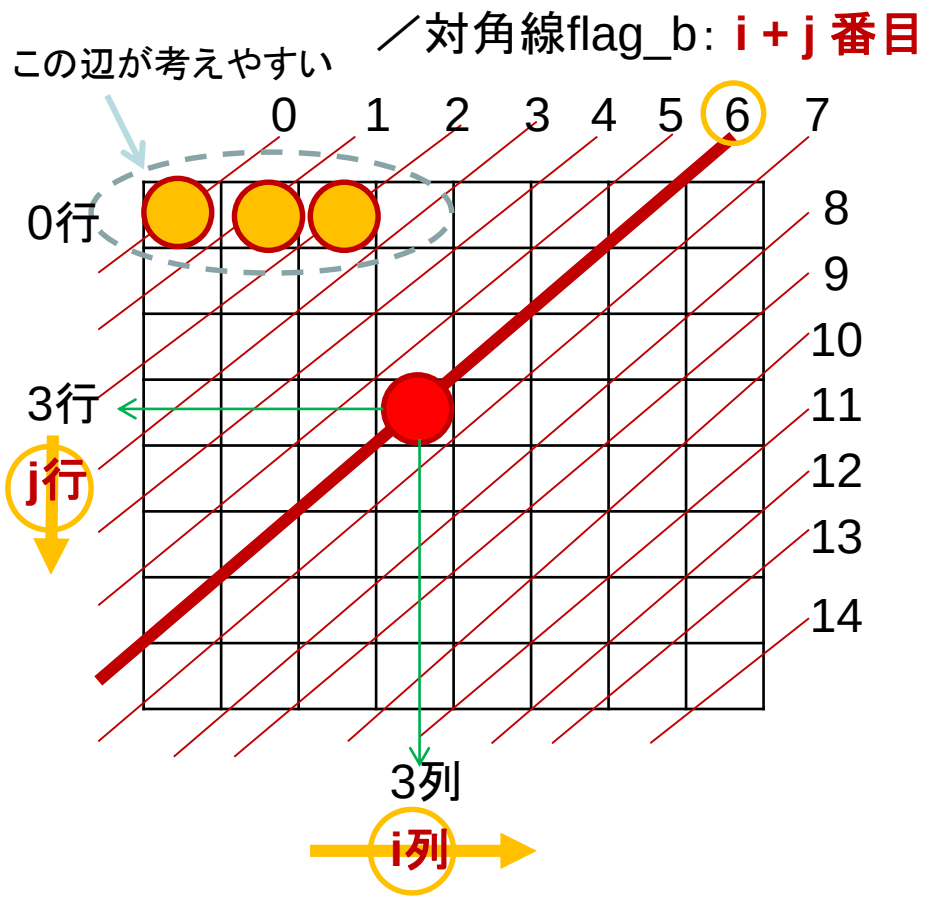
  flag_a[j] != 1　　if(! flag_a[j])
- 限定：／対角線に王妃は1つ

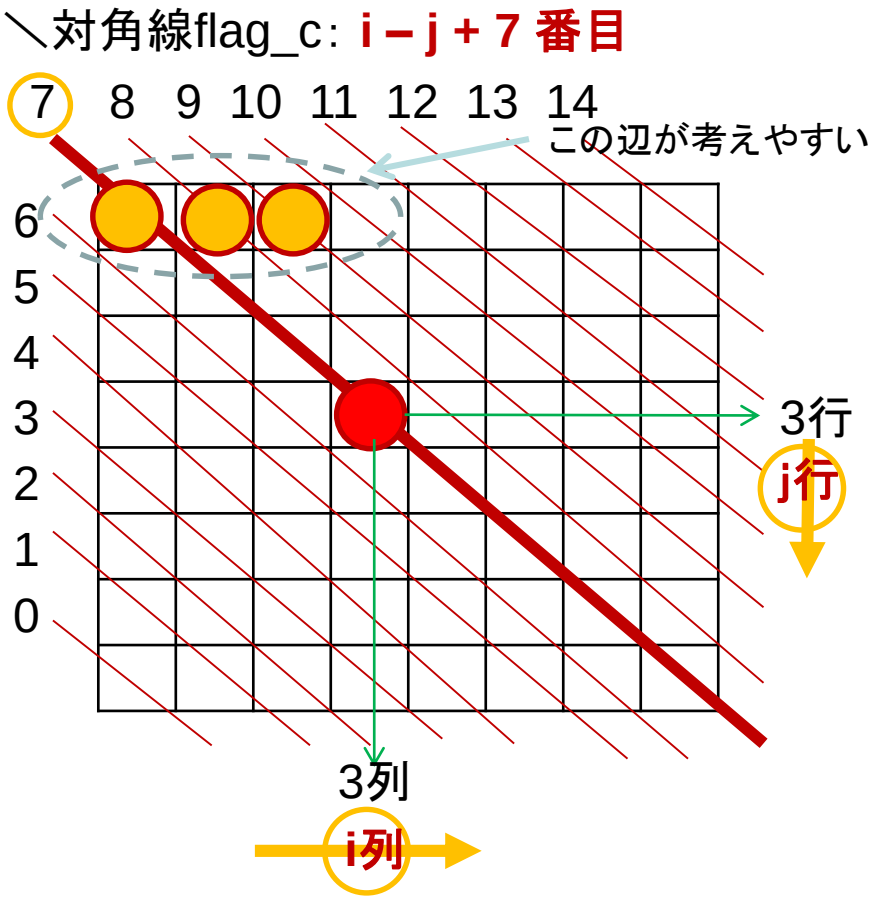
  flag_b[i + j] != 1　　if(! flag_b[i + j])
- 限定：＼対角線に王妃は1つ

  flag_c[i − j + 7] != 1　　if(! flag_c[i − j + 7])

※！は論理演算子（否定）、真を偽、偽を真に変換する

ヒント(続き)


この辺が考えやすい
／対角線flag_b: i＋j 番目
0 1 2 3 4 5 6 7
8
9
10
11
12
13
14
0行
3行
j行
3列
i列


＼対角線flag_c: i－j＋7 番目
7 8 9 10 11 12 13 14
この辺が考えやすい
6
5
4
3
2
1
0
3行
j行
3列
i列

実行結果

```
> ./prog9-9
0 4 7 5 2 6 1 3
0 5 7 2 6 3 1 4
0 6 3 5 7 1 4 2
0 6 4 7 1 3 5 2
1 3 5 7 2 0 6 4
1 4 6 0 2 7 5 3
1 4 6 3 0 7 5 2
1 5 0 6 3 7 2 4
1 5 7 2 0 3 6 4
1 6 2 5 7 4 0 3
1 6 4 7 0 3 5 2
1 7 5 0 2 4 6 3
2 0 6 4 7 1 3 5
2 4 1 7 0 6 3 5
2 4 1 7 5 3 6 0
2 4 6 0 3 1 7 5
2 4 7 3 0 6 1 5
2 5 1 4 7 0 6 3
2 5 1 6 0 3 7 4
2 5 1 6 4 0 7 3
2 5 3 0 7 4 6 1
2 5 3 1 7 4 6 0
2 5 7 0 3 6 4 1
2 5 7 0 4 6 1 3
2 5 7 1 3 0 6 4
2 6 1 7 4 0 3 5
2 6 1 7 5 3 0 4
2 7 3 6 0 5 1 4
3 0 4 7 1 6 2 5
3 0 4 7 5 2 6 1
3 1 4 7 5 0 2 6
3 1 6 2 5 7 0 4
3 1 6 2 5 7 4 0
3 1 6 4 0 7 5 2
3 1 7 4 6 0 2 5
3 1 7 5 0 2 4 6
3 5 0 4 1 7 2 6
3 5 7 1 6 0 2 4
3 5 7 2 0 6 4 1
3 6 0 7 4 1 5 2
3 6 2 7 1 4 0 5
3 6 4 1 5 0 2 7
3 6 4 2 0 5 7 1
3 7 0 2 5 1 6 4
3 7 0 4 6 1 5 2
3 7 4 2 0 6 1 5
4 0 3 5 7 1 6 2
4 0 7 3 1 6 2 5
4 0 7 5 2 6 1 3
4 1 3 5 7 2 0 6
4 1 3 6 2 7 5 0
4 1 5 0 6 3 7 2
4 1 7 0 3 6 2 5
4 2 0 5 7 1 3 6
4 2 0 6 1 7 5 3
4 2 7 3 6 0 5 1
4 6 0 2 7 5 3 1
4 6 0 3 1 7 5 2
4 6 1 3 7 0 2 5
4 6 1 5 2 0 3 7
4 6 1 5 2 0 7 3
4 6 3 0 2 7 5 1
4 7 3 0 2 5 1 6
4 7 3 0 6 1 5 2
5 0 4 1 7 2 6 3
5 1 6 0 2 4 7 3
5 1 6 0 3 7 4 2
5 2 0 6 4 7 1 3
5 2 0 7 3 1 6 4
5 2 0 7 4 1 3 6
5 2 4 6 0 3 1 7
5 2 4 7 0 3 1 6
5 2 6 1 3 7 0 4
5 2 6 1 7 4 0 3
5 2 6 3 0 7 1 4
5 3 0 4 7 1 6 2
5 3 1 7 4 6 0 2
5 3 6 0 2 4 1 7
5 3 6 0 7 1 4 2
5 7 1 3 0 6 4 2
6 0 2 7 5 3 1 4
6 1 3 0 7 4 2 5
6 1 5 2 0 3 7 4
6 2 0 5 7 4 1 3
6 2 7 1 4 0 5 3
6 3 1 4 7 0 2 5
6 3 1 7 5 0 2 4
6 4 2 0 5 7 1 3
7 1 3 0 6 4 2 5
7 1 4 2 0 6 3 5
7 2 0 5 1 4 6 3
7 3 0 2 5 1 6 4
>
```

全92通り

今回の演習内容

- 講義の復習：
  - 演習9-1～9-5（計5問）
- 演習独自の課題：
  - 演習9-6～9-9（計4問）
- 提出課題：※ 今回は2題とも必須課題
  - 課題9-1～9-2（計2問）