AtCoder Regular Contest 038
解説

AtCoder株式会社

はじめに

今回の問題は４問とも（２人）ゲーム系の問題でした

ゲーム系の問題は大体、以下いずれかで解くことが出来ます

・後ろから探索する

・Grundy数を求める

・adhoc（≒独特な）な必勝法を見つける

ARC 038 A問題 解説
「カードと兄妹」

snuke

問題概要

カードが N 枚あり、カード i には整数 A[i] が書かれている

２人で交互に１枚ずつ取っていく

取ったカードに書かれた数の和がスコアとなる

２人とも自分のスコアを最大化しようとするとき、

先手のスコアはいくらになるか？

～制約～

1 ≦ N ≦ 1000

考察

各ターンではプレイヤーはどのカードを取ればいいか？

→ 残っているカードのうち書かれている数が最大のもの！

解法

A[1~N] を降順にソートしておく

A[1]+A[3]+A[5]+…を求める

タイプ

A 問題は、分類するほど複雑なゲームではありませんが、

あえて言うならば３つ目のタイプでしょうか

- 後ろから探索する
- Grundy数を求める
- adhocな必勝法を見つける

ARC 038 B問題 解説
「マス目と駒」

snuke

問題概要

H×Wのマス目があり、いくつかのマスには障害物がある

最初、駒を左上のマスにおき、２人で交互に動かす

動かす先は、１つ下・１つ右下・１つ右のいずれかのマス

障害物のあるマスや、盤外に駒を動かすことはできない

駒を動かせなくなった方の負け

先手と後手のどちらが勝つか？

～制約～

2 ≦ H,W ≦ 100

部分点（30点）- 制約

$2 \leqq H,W \leqq 4$

盤がとても小さい

部分点（30点）- 解法

全探索をする

```
judge(i, j): # (i,j)に駒がある状態から開始したときの勝敗
  if (i,j) が盤外 or 障害物: return 勝ち
  if judge(i+1,j) == 負け: return 勝ち
  if judge(i+1,j+1) == 負け: return 勝ち
  if judge(i,j+1) == 負け: return 勝ち
  return 負け
```

judge(1,1)が答えとなる

（盤外と障害物を2行目のように処理すると実装が楽になります）

部分点（30点） - 計算量

O(分岐数^再帰の深さ) で抑えられる

再帰の深さは高々 6 （盤面が4x4以下だから）で、

各ステップでは高々 3 通りにしか分岐しないため、

この部分点では間に合う

満点 - 制約

$2 \leqq H,W \leqq 100$

盤が大きい

満点 - 解法

メモ化再帰にする

```
mem[1000][1000] を「未定」で初期化
judge(i, j): # (i,j)に駒がある状態から開始したときの勝敗
  if (i,j) が盤外 or 障害物: return 勝ち
  if mem[i][j] != 未定: return mem[i][j]
  result = 負け
  if judge(i+1,j) == 負け: result = 勝ち
  if judge(i+1,j+1) == 負け: result = 勝ち
  if judge(i,j+1) == 負け: result = 勝ち
  return mem[i][j] = result
```

満点 - 計算量

再帰呼び出しの回数を O(HW) で抑えることができる

計算量も O(HW) となり、間に合う

タイプ

B 問題は、１つ目のタイプでしょうか

・後ろから探索する

・Grundy数を求める

・adhocな必勝法を見つける

ソースコードの見た目は前から探索しているように見えますが、実際に勝敗が確定していくのは後ろの状態からなので、「後ろから探索する」ということにしています

ARC 038 C問題 解説

「茶碗と豆」

snuke

問題概要

N 個の茶碗があり、茶碗 i には整数C[i]が書かれている

茶碗 i には A[i] 個の豆が入っている

２人で交互に、茶碗 0 以外から豆を 1 つ選んで移動させていく

茶碗 i から選んだ場合は 茶碗 i-C[i] ～茶碗 i-1 のどれかに移す

豆を選べなくなった方の負け

先手と後手のどちらが勝つか？

～制約～

2 ≦ N ≦ 10^5

0 ≦ A[i] ≦ 10^9

部分点１（60点）- 制約

$2 \leqq N \leqq 100$

$0 \leqq A[i] \leqq 10$

茶碗も豆も少ない

部分点１（60点）- 解法

grundy数を求める

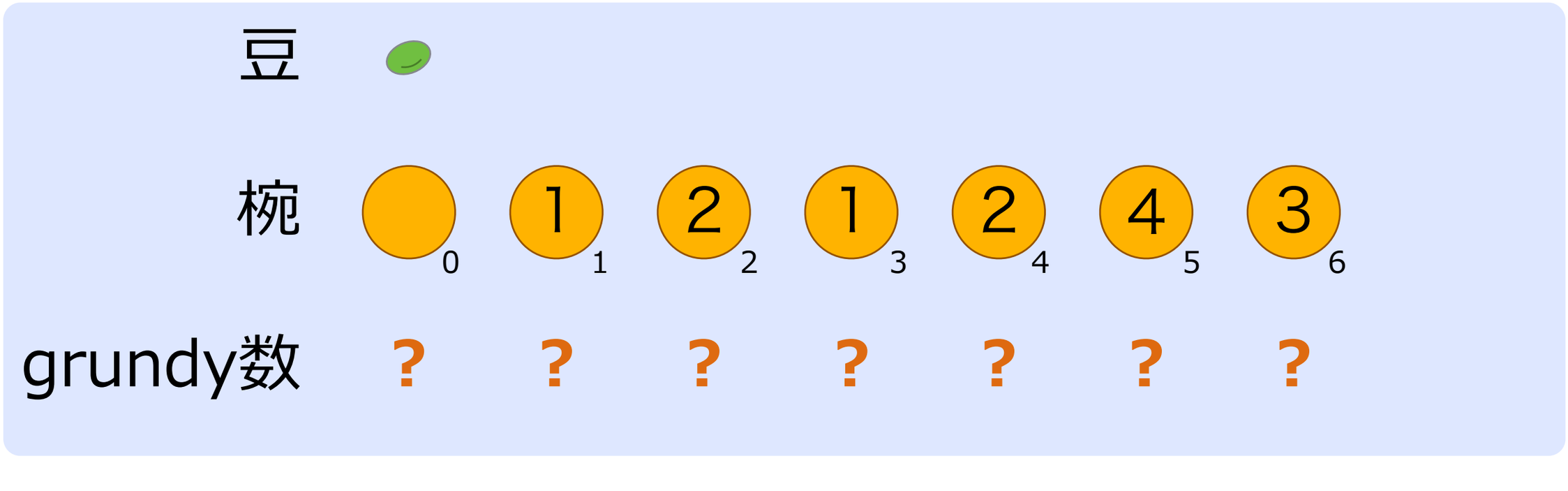
Grundy数

grundy数とは

> ある状態から 1 回の遷移で行ける状態の
> grundy数の集合に含まれない最小の非負整数

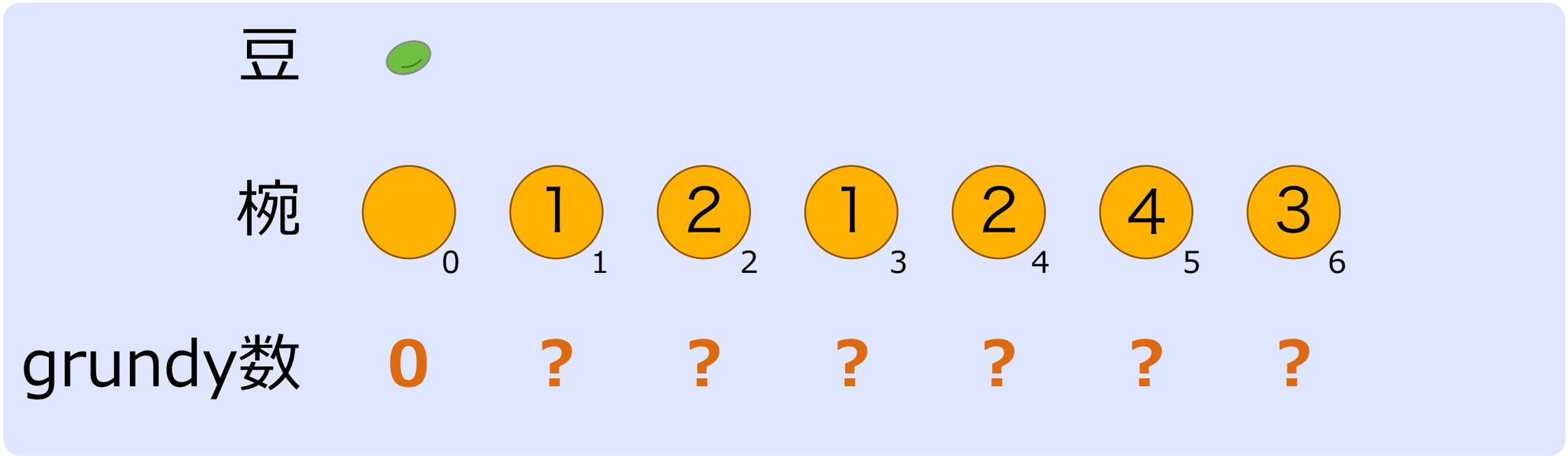
なにやら複雑そうなので、例を出して計算してみます

Grundy数



豆が１つだけのときを考えます

Grundy数



椀０に１つ豆がある状態のgrundy数を求めます

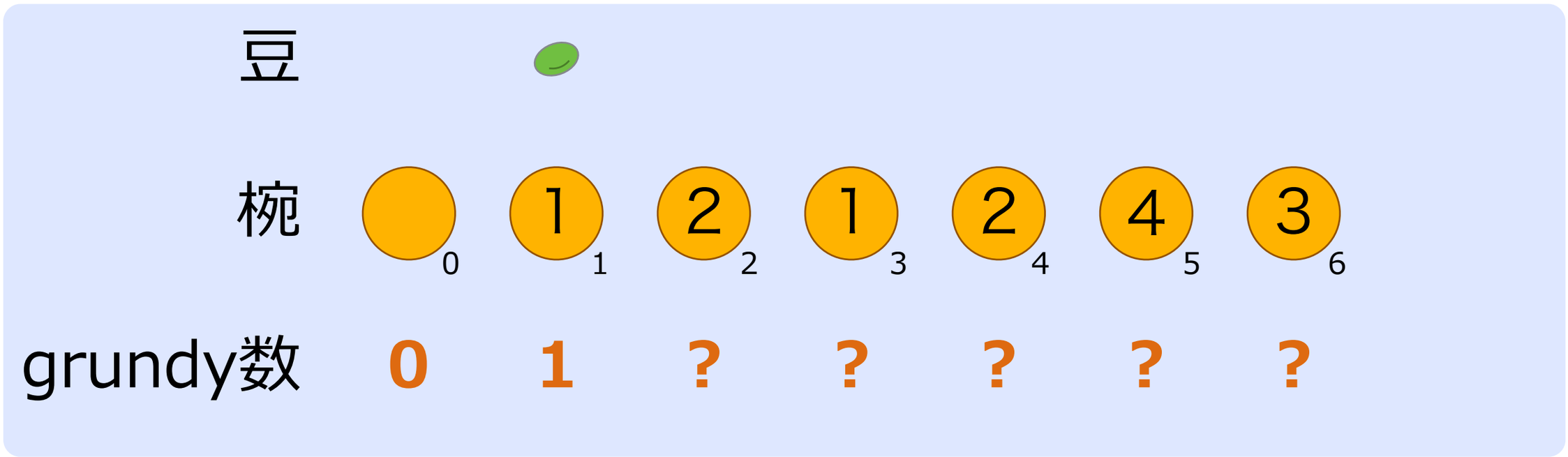
豆を移す操作がこれ以上できないため、

遷移先のgrundy数の集合は {}（空集合）となります

集合に含まれない最小の０以上の整数は０なので、

この状態のgrundy数は**0**となります

Grundy数



椀１に１つ豆がある状態のgrundy数を求めます

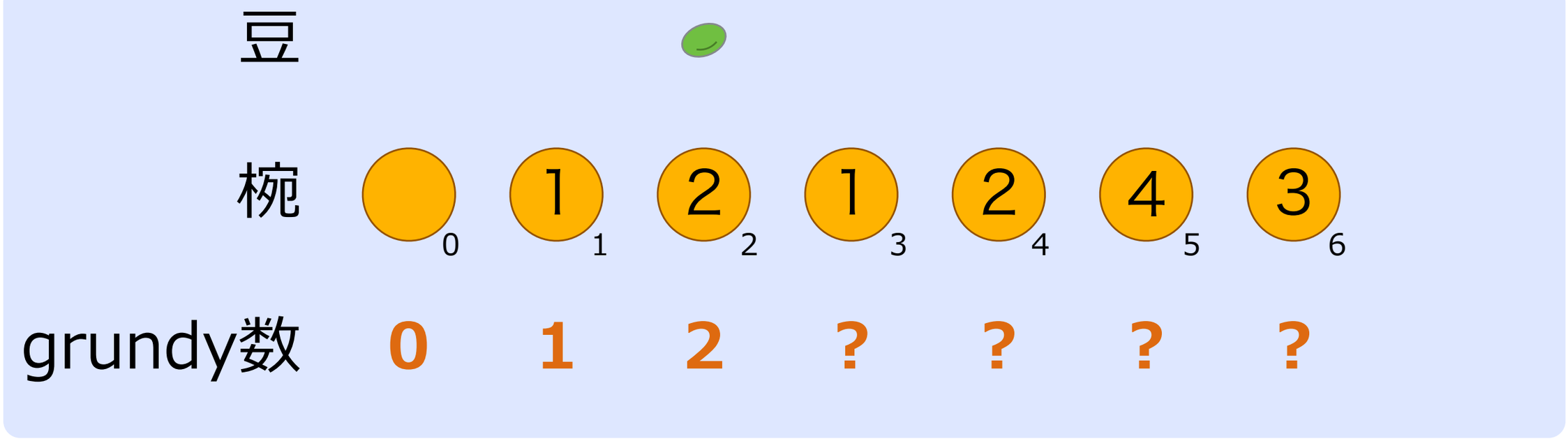
椀１からは椀０に豆を移せるため、

遷移先のgrundy数の集合は { **0** } となります

集合に含まれない最小の０以上の整数は１なので、

この状態のgrundy数は **1** となります

Grundy数



椀２に１つ豆がある状態のgrundy数を求めます

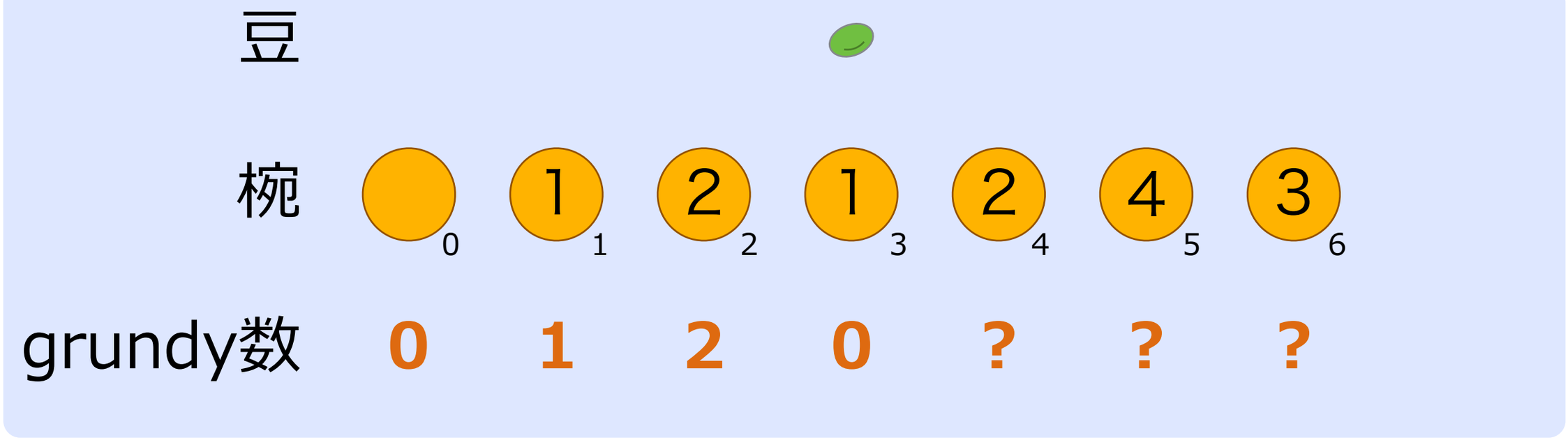
椀２からは椀０と椀１に豆を移せるため、

遷移先のgrundy数の集合は {**0**, **1**} となります

集合に含まれない最小の０以上の整数は２なので、

この状態のgrundy数は**2**となります

Grundy数



椀３に１つ豆がある状態のgrundy数を求めます

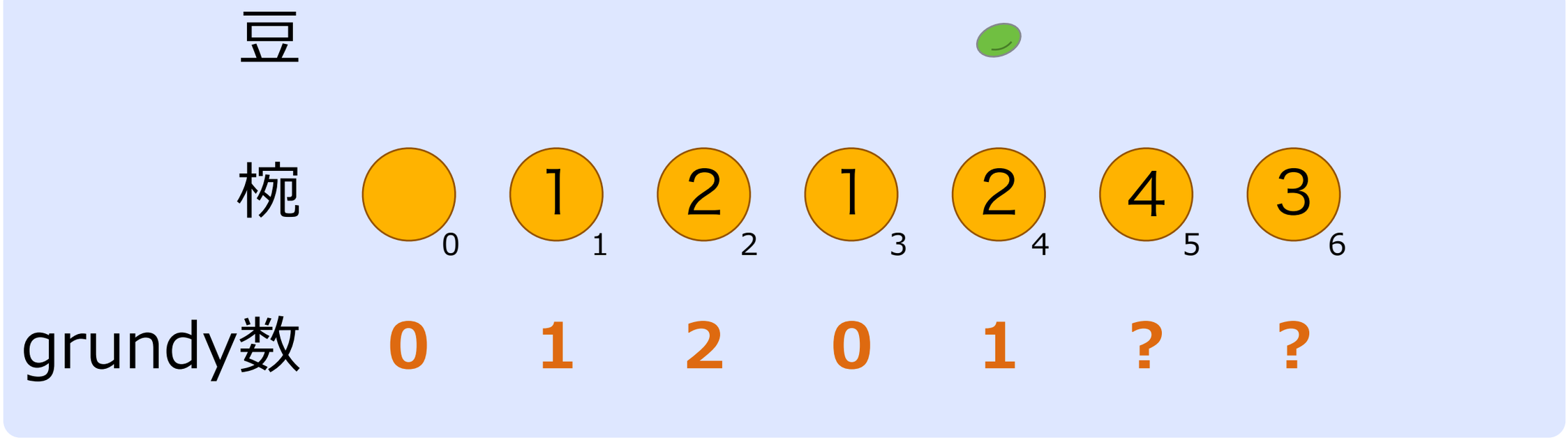
椀３からは椀２に豆を移せるため、

遷移先のgrundy数の集合は {**2**} となります

集合に含まれない最小の０以上の整数は０なので、

この状態のgrundy数は**0**となります

Grundy数



椀４に１つ豆がある状態のgrundy数を求めます

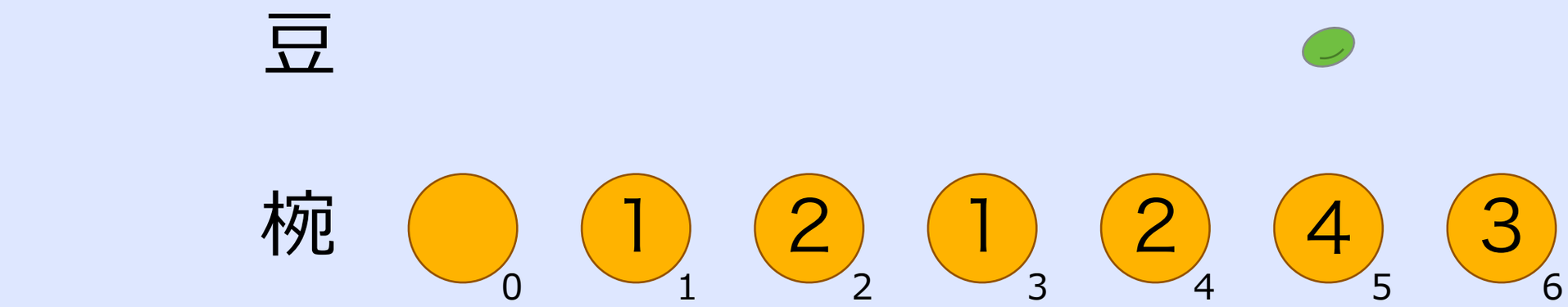
椀４からは椀２と椀３に豆を移せるため、

遷移先のgrundy数の集合は {**2**,**0**} となります

集合に含まれない最小の０以上の整数は１なので、

この状態のgrundy数は**1**となります

Grundy数



椀５に１つ豆がある状態のgrundy数を求めます

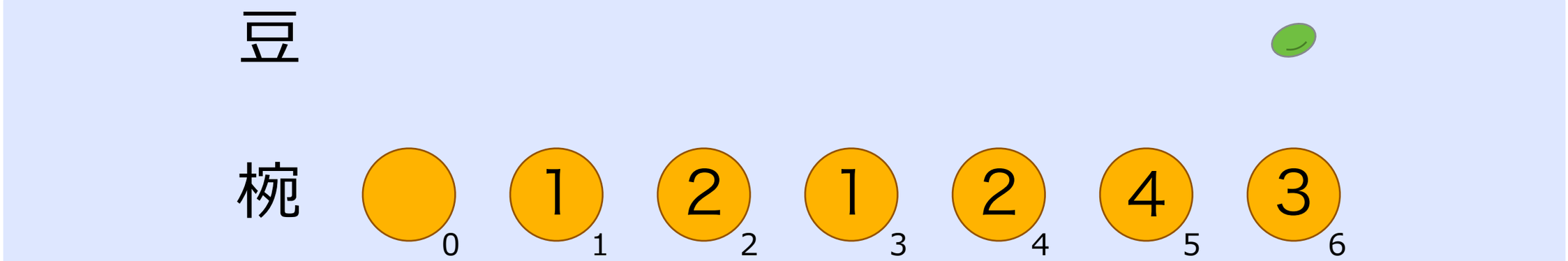
椀５からは椀１と椀２と椀３と椀４に豆を移せるため、

遷移先のgrundy数の集合は {**0**, **1**, **2**} となります

集合に含まれない最小の０以上の整数は３なので、

この状態のgrundy数は**3**となります

Grundy数



椀６に１つ豆がある状態のgrundy数を求めます

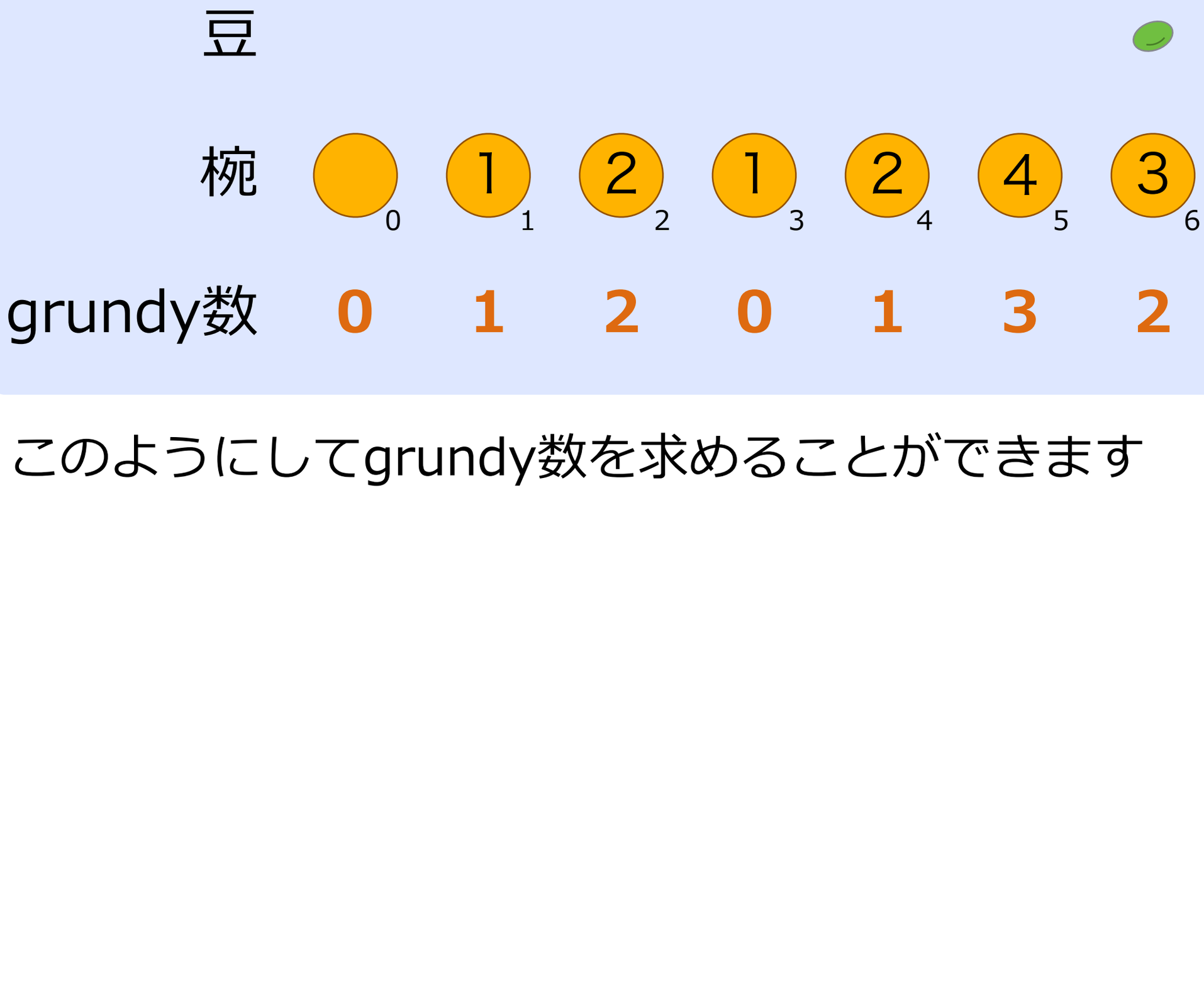
椀６からは椀３と椀４と椀５に豆を移せるため、

遷移先のgrundy数の集合は {**0**, **1**, **3**} となります

集合に含まれない最小の０以上の整数は２なので、

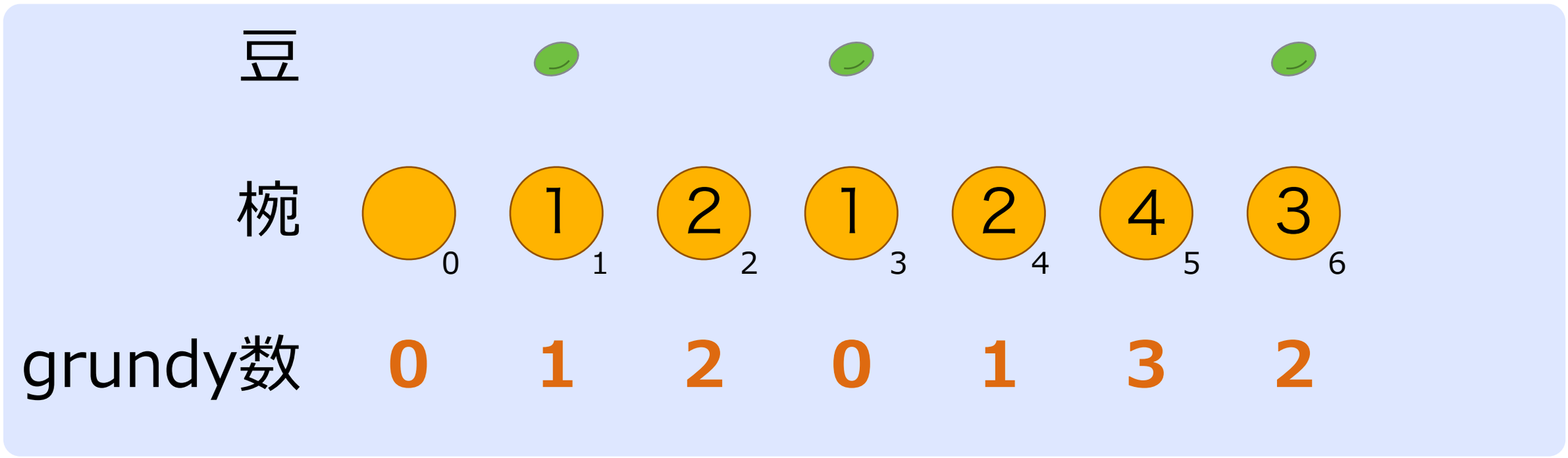
この状態のgrundy数は**2**となります

Grundy数



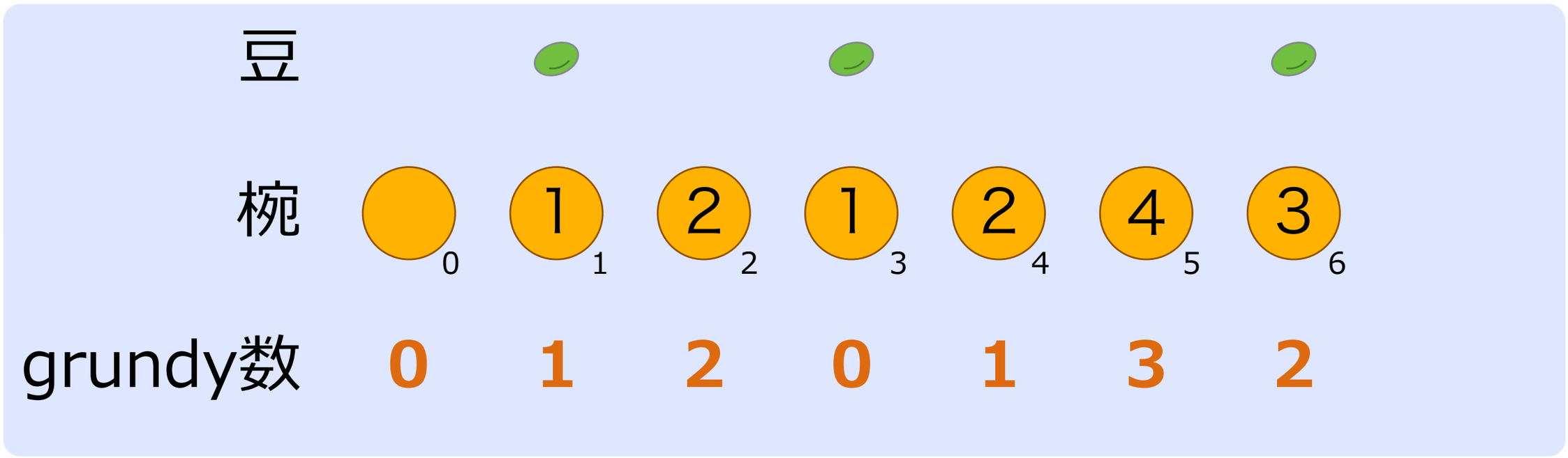
このようにしてgrundy数を求めることができます

Grundy数



豆が２つ以上ある場合はどうすればいいでしょうか

Grundy数

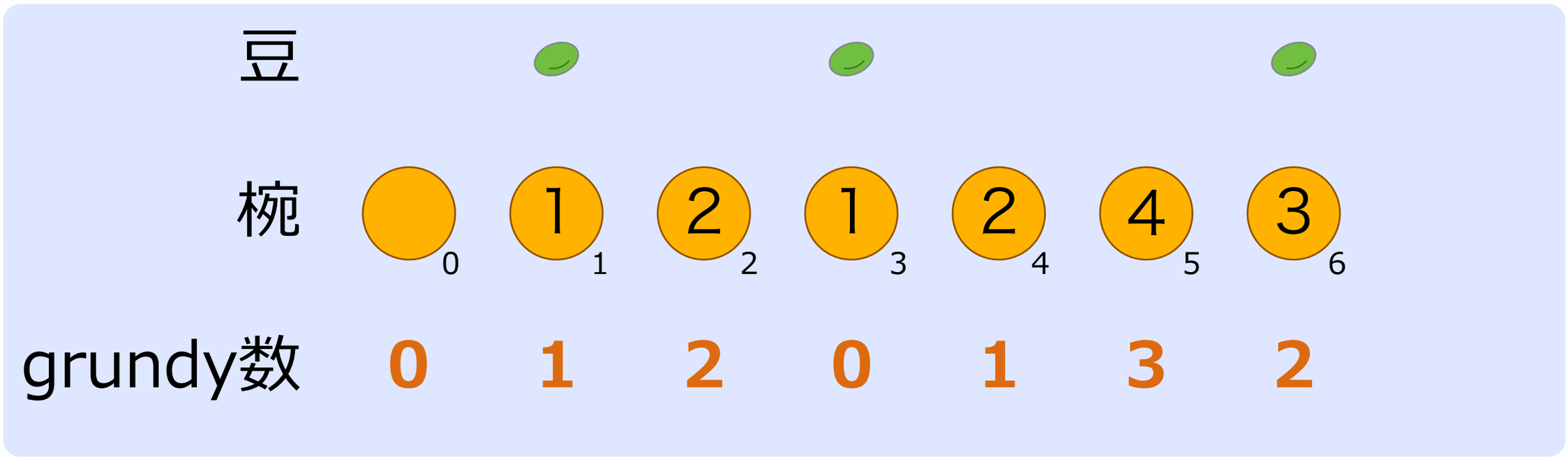


定義どおりに状態の遷移を考えていけば、

grundy数を求めることができます

しかし、実はそのようなことをしなくても求められます

Grundy数



・椀１に１つ豆がある状態のgrundy数： **1**

・椀３に１つ豆がある状態のgrundy数： **0**

・椀６に１つ豆がある状態のgrundy数： **2**

これらの **xor** の値がこの状態のgrundy数となっています

つまり、 **1** xor **0** xor **2** ＝ **3** がこの状態のgrundy数です

Grundy数

grundy数を求めると何が嬉しいのでしょうか？

・grundy数が 0 でない状態からは、
必ずgrundy数が 0 の状態に遷移できる

・grundy数が 0 の状態からは、
grundy数が 0 の状態に遷移できない

という性質から、あるゲームの状態のgrundy数が

・ 0 でないならば先手の勝ち

・ 0 ならば後手の勝ち

ということが言えるのです

部分点１（60点）- 解法

「椀 i に１つ豆がある状態のgrundy数」を

　i ＝ 0 から順に i ＝ N-1 まで求めて、

各豆についてのgrundy数のxorを求める

この値が１ならば先手の勝ちで、 0ならば後手の勝ち

部分点１（60点）- 解法

「椀 i に１つ豆がある状態のgrundy数」 = g[i] とする

g[i] を求めるときには、

・g[i-C[i]]～g[i-1] に含まれない最小の０以上の整数

を求める必要があります

→

「０が含まれるか」「１が含まれるか」・・・

を順番に調べていけば O(N^2) で求めることができます

「x が含まれるか」を調べるときに適切なデータ構造（例えばsetなど）を用いることで、O(N log N)やO(N)にできます

部分点１（60点）- 計算量

g[i] を求めるのに O(N^2) かけたとすると、

すべての g[i] を求める計算量は O(N^3)

また、豆の個数は最大で max(A[i])*(N-1) 個なので、

grundy数のxorをとる計算量は O(max(A[i])*N)

となり、この部分点では間に合います

部分点２（40点）- 制約

$2 \leqq N \leqq 100$

$0 \leqq A[i] \leqq 10^9$

茶碗は少ないが、豆は多い

部分点２（40点）- 解法

部分点解法１の解法だと、

すべての g[i] を求める計算量は O(N^3)

なので問題ないですが、

grundy数のxorをとる計算量は O(max(A[i])*N)

なので間に合いません

部分点２（40点）- 解法

xorの性質として、

・同じ数の xor は 0

というものがあります

例えば、3 xor 3 = 0 で、 99 xor 99 = 0 です

部分点２（40点）- 解法

grundy数のxorをとるとき、

同じ数のxorを大量にとっています

例えば A[i] = 5 のとき、

・g[i] xor g[i] xor g[i] xor g[i] xor g[i]

という計算をしていますが、これは、

・g[i] xor g[i] xor g[i] や、

・g[i]

と同じです

このように同じ数を２つずつ相殺させることを考えます

部分点２（40点）- 解法

つまり、A[i] がどのような数であろうと、

勝敗には A[i] の偶奇しか関係ないのです

部分点２（40点）- 計算量

grundy数のxorをとる計算量は、

A[i] が奇数であるような所の g[i] のxorをとるだけなので

O(N) で良いことになり、

この部分点は間に合います

満点（おまけ）- 制約

2 ≦ N ≦ 10^5

0 ≦ A[i] ≦ 10^9

茶碗も多い

上級者向けのおまけの制約です

満点（おまけ）- 解法

g[i] をもっと高速に求めなければなりません

g[i] は、

・g[i-C[i]]～g[i-1] に含まれない最小の 0 以上の整数

でした

満点（おまけ）- 解法

「g[k]~g[i-1] に l~r がすべて含まれるような最小のk」

という情報を持ったsegtreeを更新しつつg[i]を求めていく

（各グランディー数について、一番最後に現れた位置を持つイメージ）

このsegtreeで二分探索を行うことにより、

O(log N) で g[i] を求めることができます

計算量は O(N log N) となり、間に合います

タイプ

C 問題は、２つ目のタイプでしょうか

・後ろから探索する

・Grundy数を求める

・adhocな必勝法を見つける

ARC 038 D問題 解説
「有向グラフと数」

snuke

問題概要

N 頂点 M 辺の有向グラフがある

頂点 i には整数 X[i] が書かれている

最初、頂点１に駒を置き、２人で交互に辺に沿って動かす

駒を動かす代わりに「終了宣言」をすることもできる

終了宣言がされるか、後手が10^9回駒を動かした時点でゲームが終了し、そのときに駒のある頂点に書かれている整数がゲームのスコアとなる

先手がスコアを最大化しようとし、後手がスコアを最小化しようとするとき、ゲームのスコアはいくつになるか？

部分点（30点） - 制約

$2 \leqq N \leqq 1000$

$1 \leqq M \leqq 2000$

グラフが小さい

部分点（30点）- 解法

ターン数が 10^9 でも 10^9 - 1 でも結果は変わらない

ターン数がいくら大きくても、結局堂々巡りになり、
先手がどこかで妥協しなければならない

もっというと、ターン数は 2N ターンまででも変わらない

ゲームの状態は [駒がある頂点] と [手番] で決まり、

同じ状態に 2 回訪れるということは、堂々巡りになるということなので意味がない

部分点（30点）- 解法

[ターン][駒がある頂点][手番] という状態を考えて、

それぞれの状態からの結果を求める

手番が先手の時は、遷移先の状態の結果のうち最大を選び、

手番が後手の時は、遷移先の状態の結果のうち最小を選ぶ

というメモ化再帰探索をする

（min-max法 と呼ばれる手法です）

部分点（30点） - 計算量

状態数が O(ターン数 * 頂点数)

遷移数が O(ターン数 * 辺数)

となっているため、計算量は O (N*(N+M)) となり、

この部分点では間に合う

満点 - 制約

$2 \leqq N \leqq 100,000$

$1 \leqq M \leqq 200,000$

グラフが大きい

満点 - 解法

・スコアを x 以上にすれば先手の勝ち
・スコアを x 未満にすれば後手の勝ち

というゲームの結果を計算できたとする

x が小さければ小さいほど先手が勝ちやすく、

x が大きければ大きいほど先手が勝ちにくいため、

x の値で二分探索ができる

満点 - 解法

・スコアを x 以上にすれば先手の勝ち
・スコアを x 未満にすれば後手の勝ち

というゲームは、

書かれている数が x 以上の頂点を青く塗り、

書かれている数が x 未満の頂点を白く塗ったと思えば、

・青い頂点で終了すれば先手の勝ち
・白い頂点で終了すれば後手の勝ち

というゲームになる

満点 - 解法

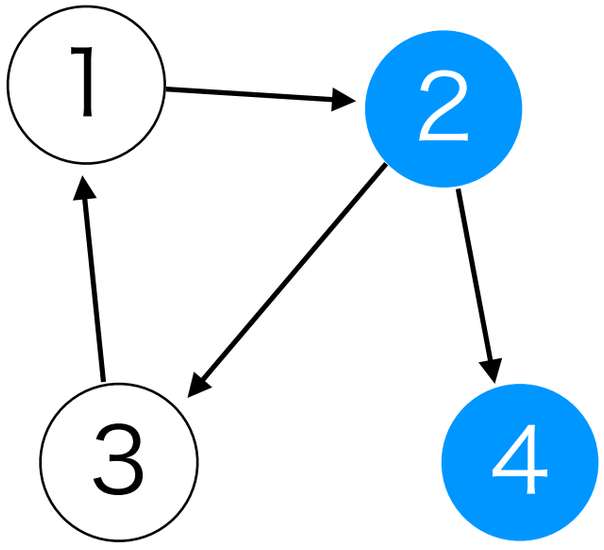
・黒い頂点で終了すれば先手の勝ち
・白い頂点で終了すれば後手の勝ち

というゲームの結果を計算できれば良くなった

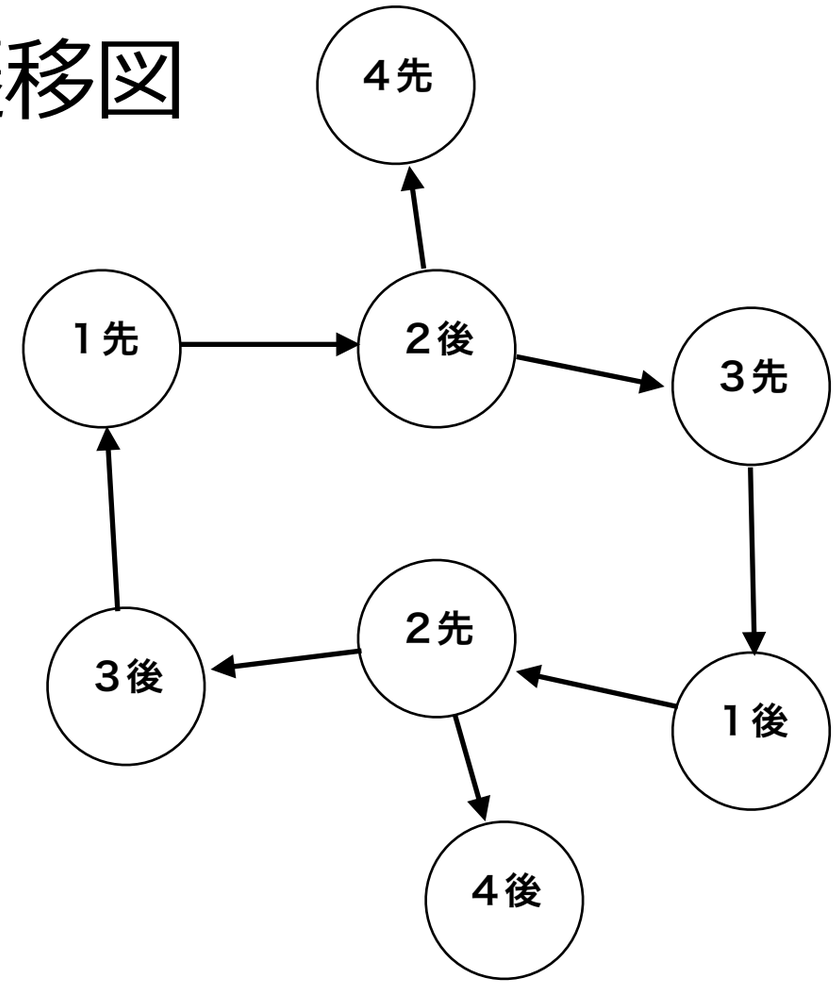
満点 - 解法

状態が [駒のある頂点][手番] の状態遷移図を考える

元のグラフ



状態遷移図



満点 - 解法

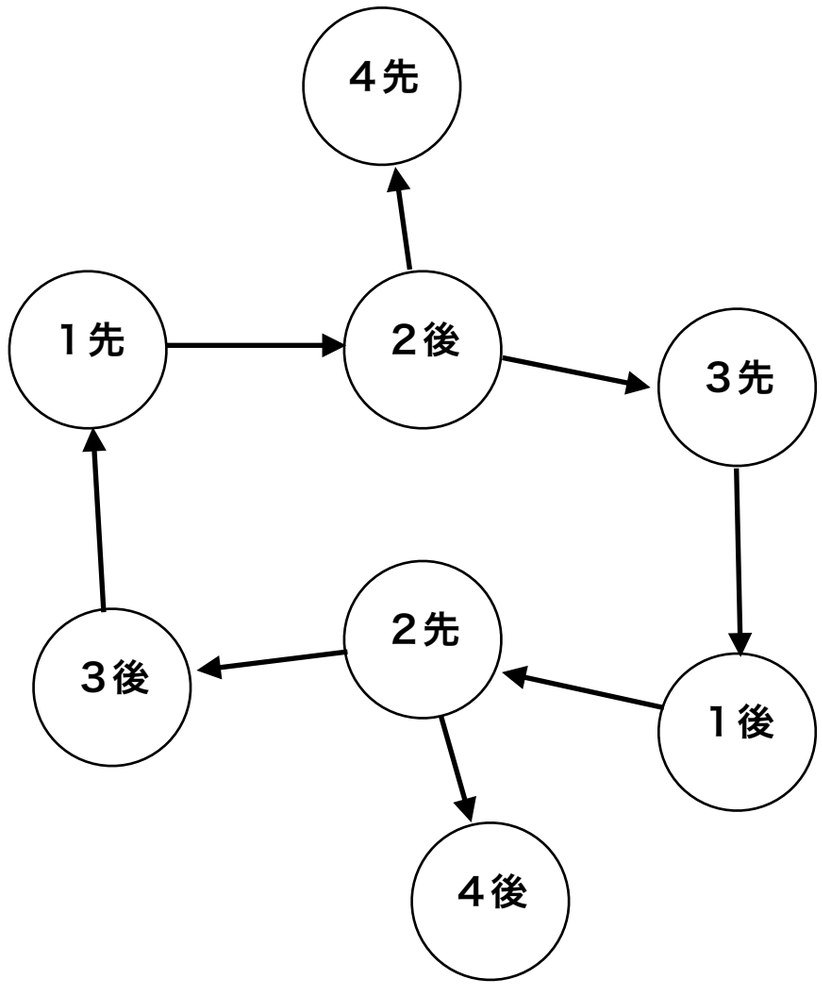
状態遷移図の頂点の色を

黒：手番プレイヤーの勝ち

灰：未定

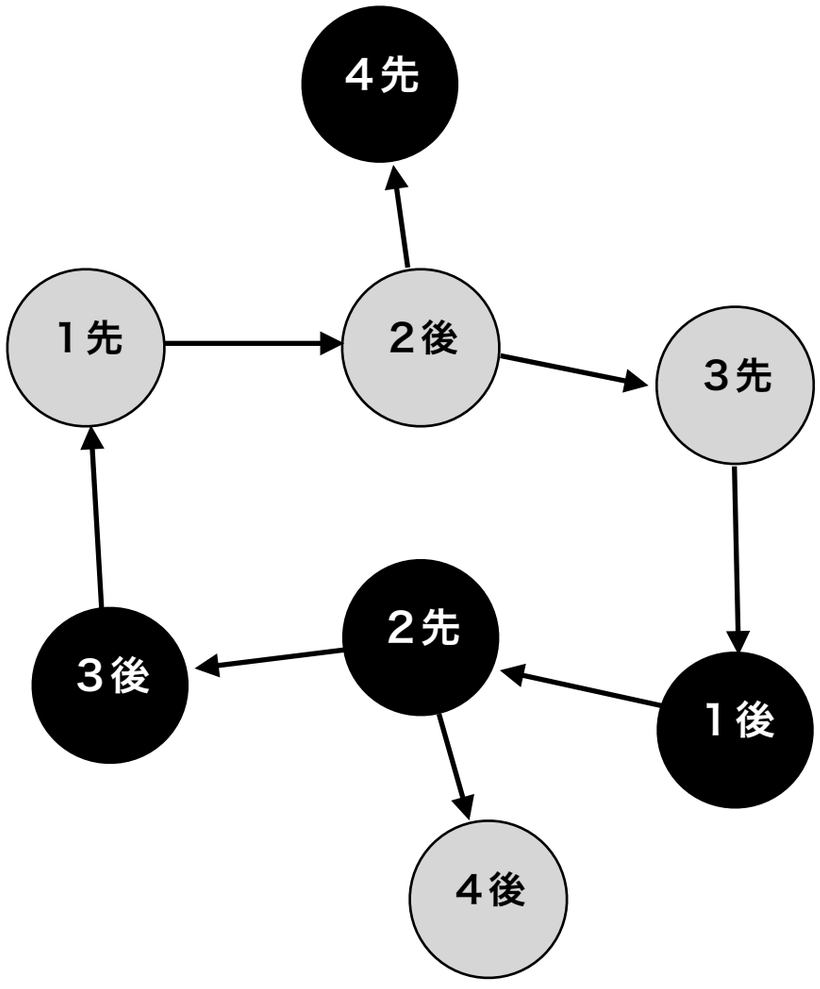
白：手番プレイヤーの負け

とします



満点 - 解法

最初はこうなっています



満点 - 解法

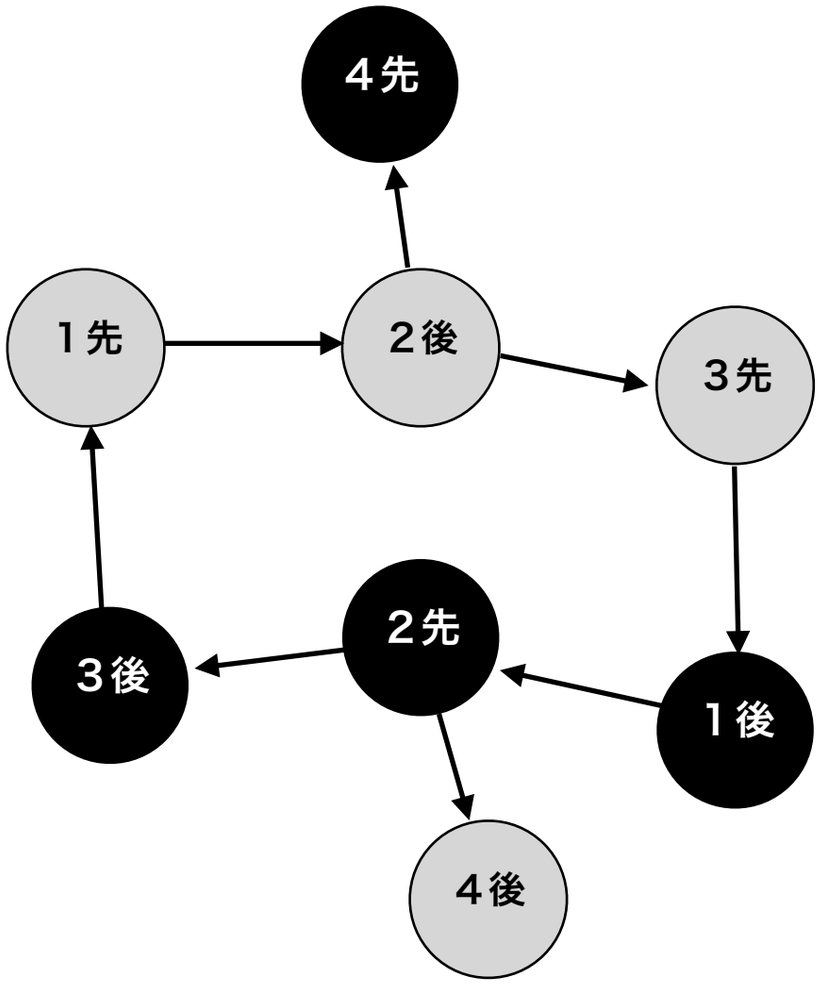
これを、

・遷移先に白があれば黒にする

・遷移先が全て黒なら白にする

というルールにそって、

色付けしていきます



満点 - 解法

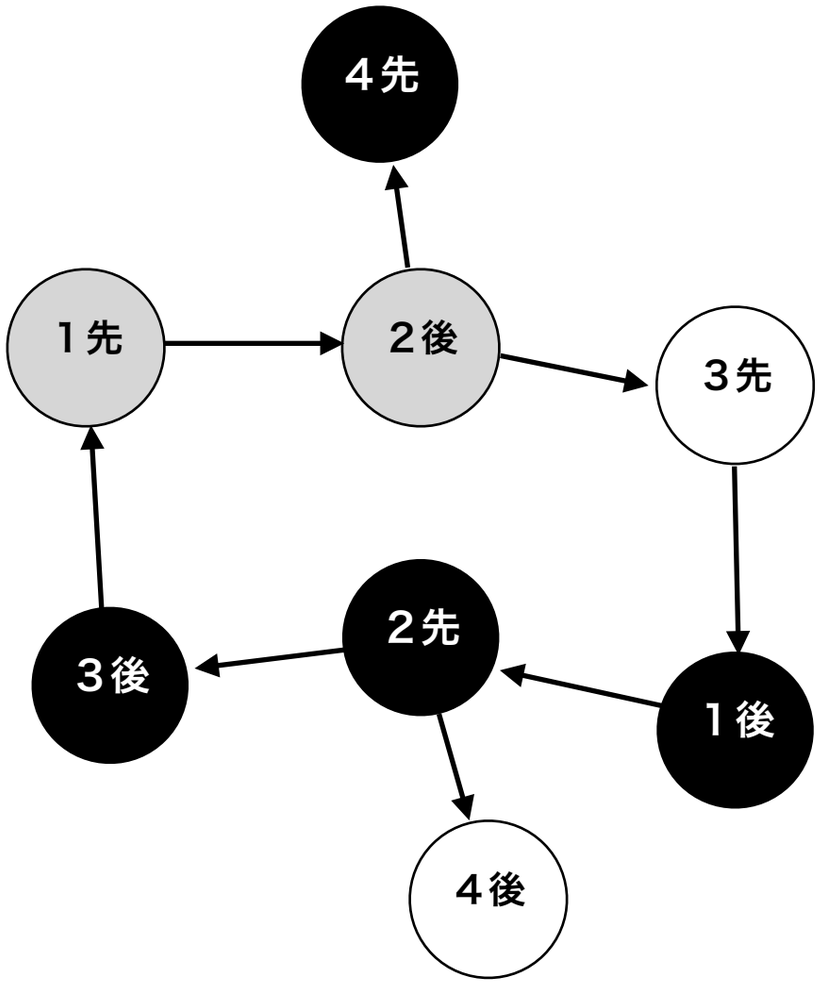
これを、

・遷移先に白があれば黒にする

・遷移先が全て黒なら白にする

というルールにそって、

色付けしていきます



満点 - 解法

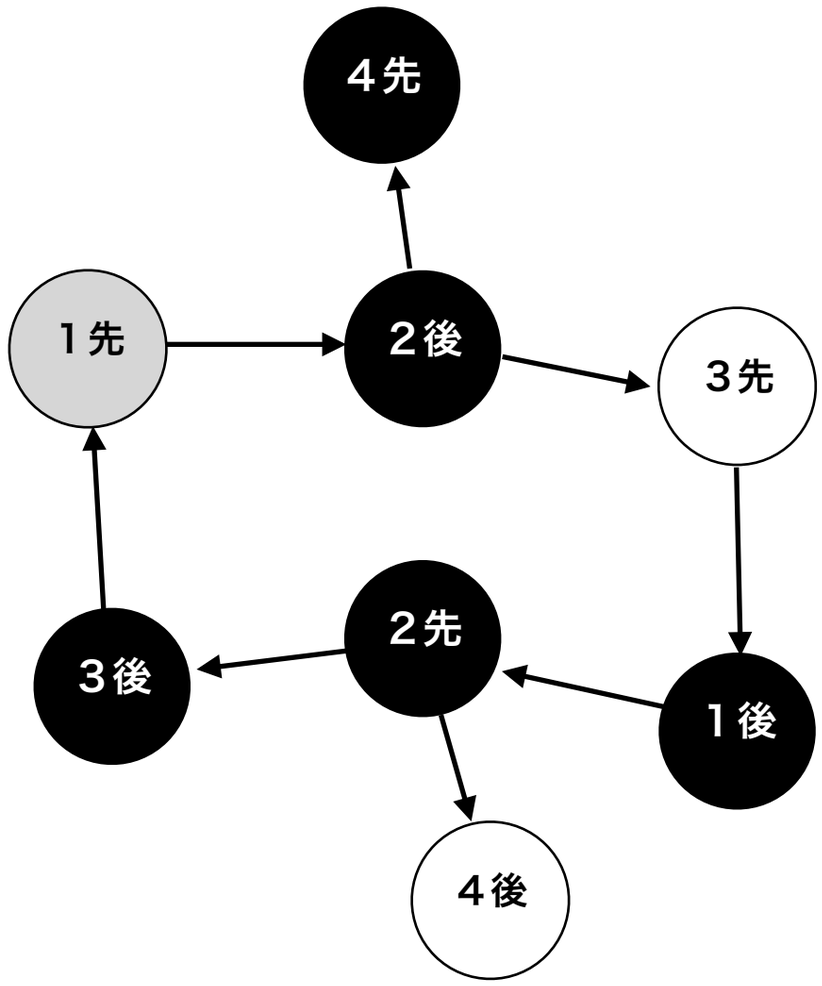
これを、

・遷移先に白があれば黒にする

・遷移先が全て黒なら白にする

というルールにそって、

色付けしていきます



満点 - 解法

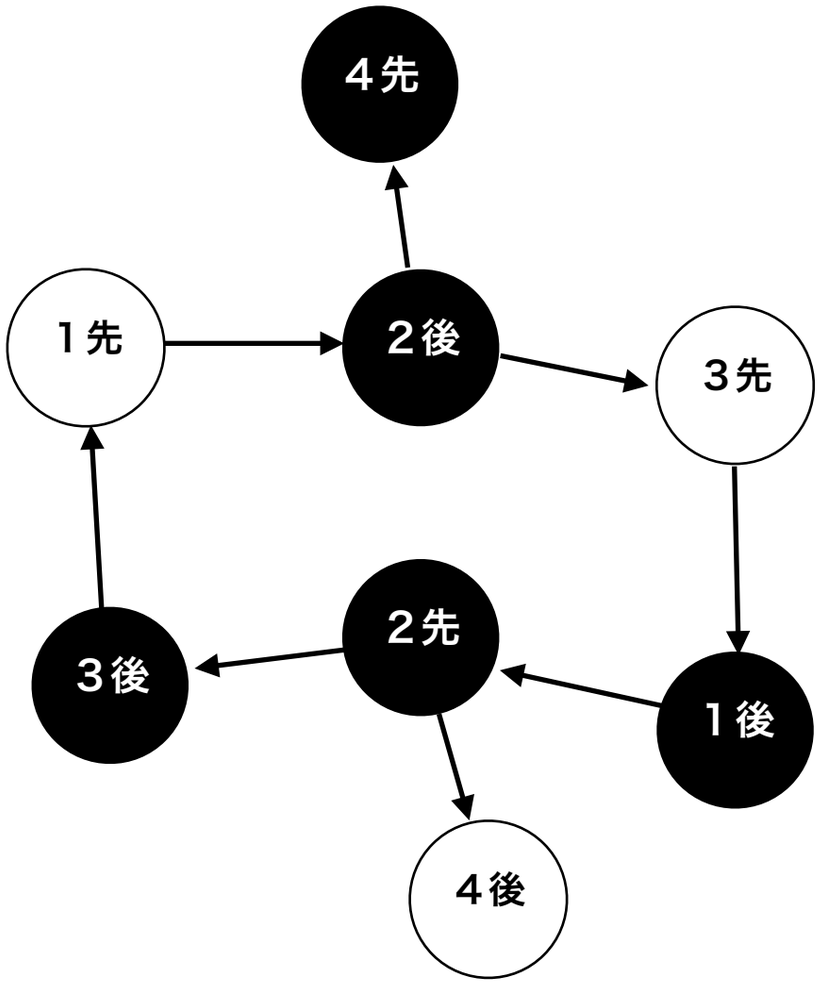
これを、

・遷移先に白があれば黒にする

・遷移先が全て黒なら白にする

というルールにそって、

色付けしていきます



満点 - 解法

このルールで塗って行ったときに、最終的に

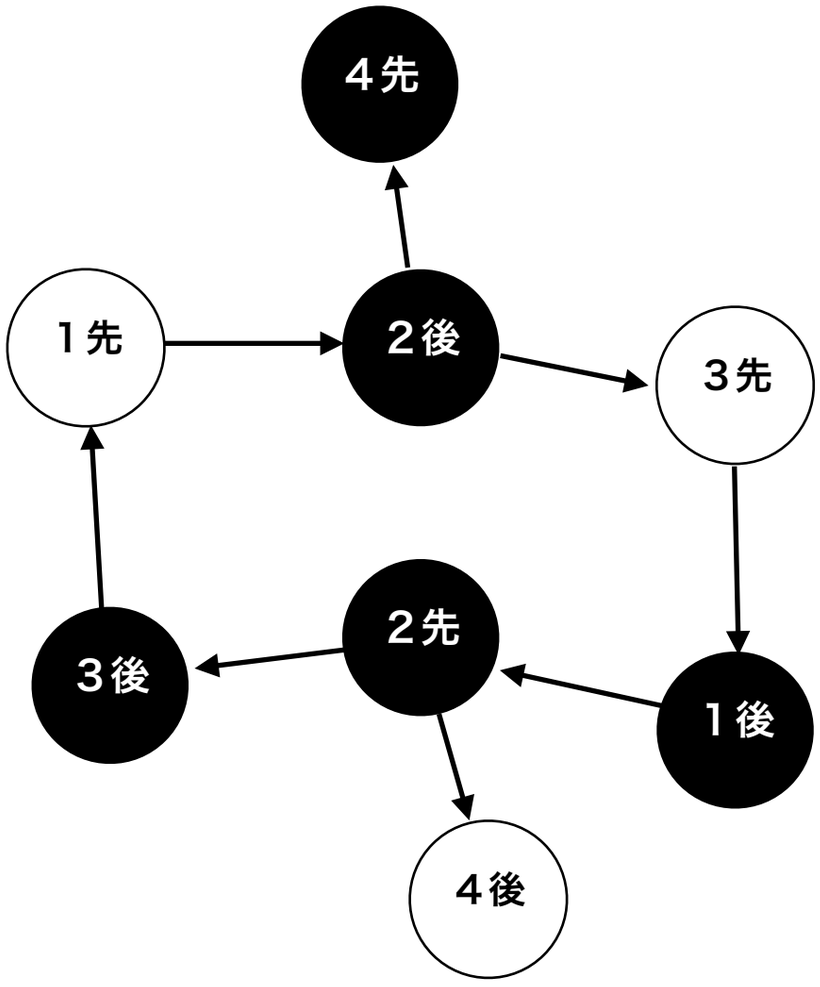
黒の頂点：勝ち

白の頂点：負け

灰の頂点：引き分け

となります

（灰の頂点の場合は堂々巡りになります）

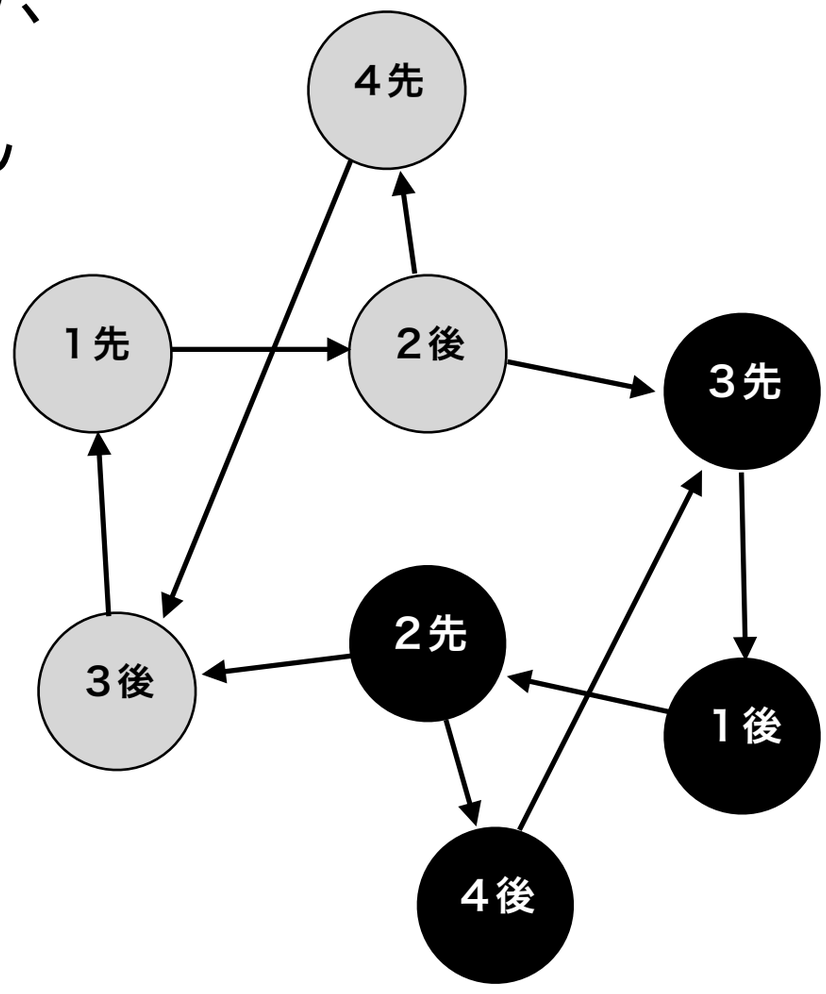


満点 - 解法

灰の頂点は例えば以下のような状況で出てきます

灰の頂点のサイクルができており、

これ以上ルールを適用できません



満点 - 解法

このような色塗りの手順は、BFSの要領で行えばよいです

「遷移先が全て黒」というのは、

「遷移先の黒の個数が出次数と等しい」と考えて実装します

このような手法で勝敗を判定するアルゴリズムを

後退解析

と呼ぶそうです

どうぶつしょうぎの解析などにも使われている手法です

満点 - 計算量

計算量は、二分探索＋後退解析で

O((N+M) log max(X))

となり間に合います

タイプ

D 問題は、１つ目のタイプでしょうか

・後ろから探索する

・Grundy数を求める

・adhocな必勝法を見つける