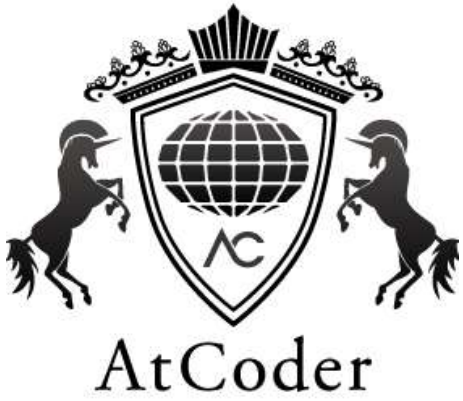
CODE FESTIVAL 2014 本選
解説



AtCoder株式会社 代表取締役
高橋 直大

A問題　50M走

A問題

AtCoder

- 問題概要
  - 50m走のタイムが与えられる。
  - 秒速平均何mで走ったかを出力せよ。

- 解説
  - 割り算をして出力
    - 与えられる秒数が整数であるため、整数同士の割り算をしないように注意！

B問題　暗算ゲーム

B問題

AtCoder

- 問題概要
  - 1桁の整数列が与えられる
  - 奇数番目の数を足し、偶数番目の数を引いた時、答えを求めよ。

- 解説
  - 指示通りに計算をする。

C問題　N進数

C問題　問題概要

AtCoder

- 整数Aが与えられる。
- NをN進数で表した数をf(N)と呼ぶ。
  - f(N) = AとなるNは存在するかを求めよ。
    - 存在する場合はその数字を出力
    - 存在しない場合は-1を出力

- 制約
  - 1 ≦ A ≦ 10^16

C問題　アルゴリズム

AtCoder

- f(A)は単調増加
  - 同じ桁数の時は、数字が大きい方がもちろん大きい
  - 桁数が違う時も、同様に大きい
    - 99進数で99と100進数で100では、100の方が大きい
      - 99進数で99<100であり、99進数の100より100進数の100の方が大きい
- つまり、Aを越えない最大のNを二分探索で求めれば良い
  - これが出来れば一応解ける

C問題　アルゴリズム

AtCoder

- そもそも、Nの取り得る範囲はどこまでか？
  - 入力されるAは10^16まで
    - f(10000)が10^16であることは、入出力例3より解る。
  - つまり、Nは、10から10000までの数字しか取り得ない

- よって、全探索してしまえば良い
  - 二分探索だとオーバーフローに気を付けなければならないが、全探索なら安心！

D問題　パスカルの三角形

1. 問題概要
2. アルゴリズム

D問題　問題概要

AtCoder

- 整数Nが与えられる。
- パスカルの三角形に整数Nが出現するのであれば、その段数および何番目かを出力せよ。
  - 複数ある場合はどれでも良い
  - 出現しない場合は、-1を出力

- 制約
  - 1≦N≦ 10^9
  - 出力の段数は2 * 10^9以下

D問題　アルゴリズム

AtCoder

- パスカルの三角形

1

1　1

1　2　1

1　3　3　1

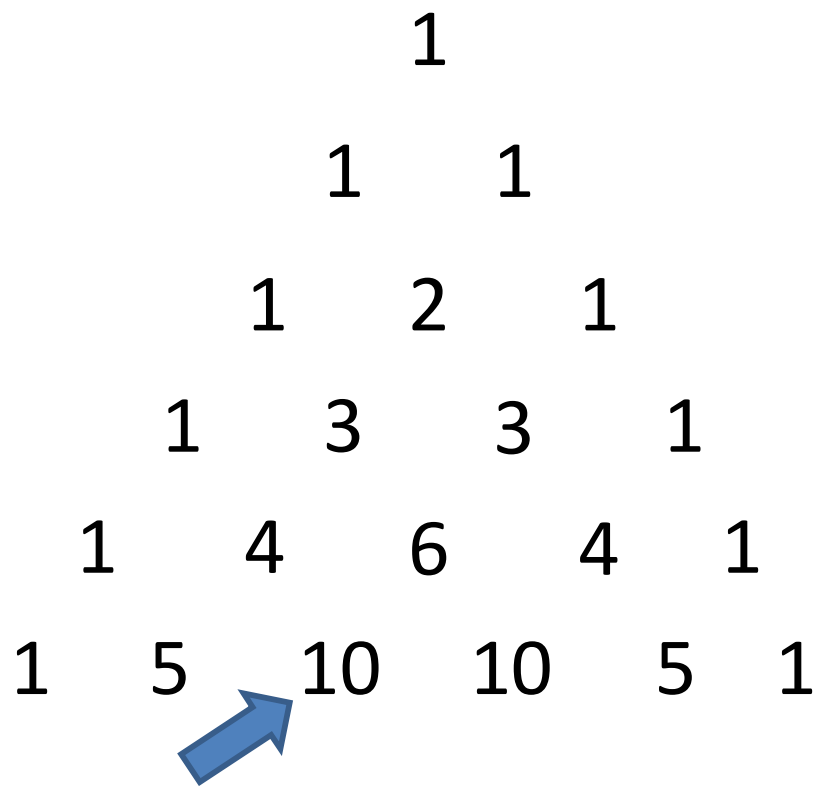
1　4　6　4　1

1　5　10　10　5　1

D問題　アルゴリズム

AtCoder

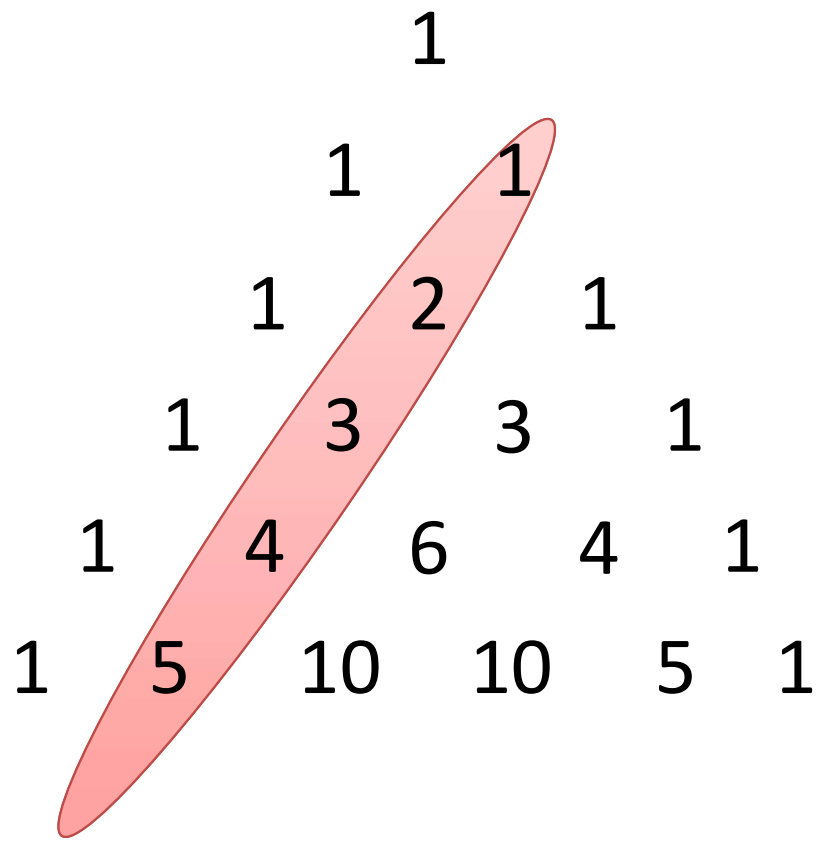
- パスカルの三角形
  - 例えば10を探そうとすると、こんなところにある
    - でも、大きな数字を探すのは大変そう！！！



D問題　アルゴリズム

AtCoder

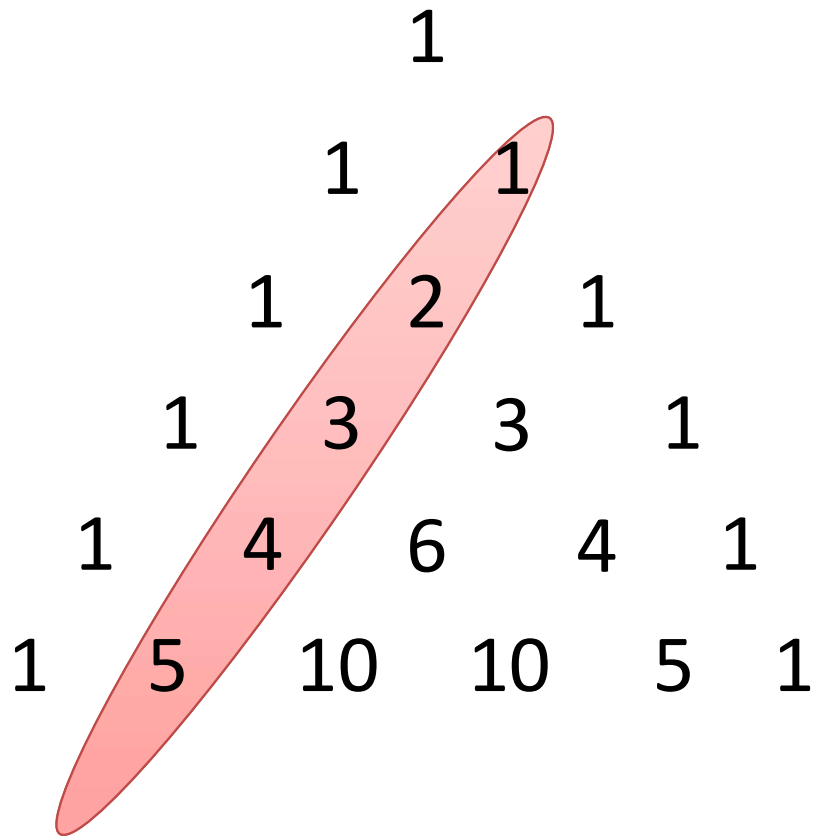
- パスカルの三角形
  - この列に注目！！！



D問題　アルゴリズム

AtCoder

- パスカルの三角形
  - この列に注目！！！
    - 1ずつ増えているので、この列の数字を出力すれば良い！



D問題　アルゴリズム

AtCoder

- パスカルの三角形の特性を生かす
  - 整数Nは、N+1段目の2番目に必ず現れる
  - これを出力すれば良い

AtCoder

E問題　常ならずグラフ

E問題 問題概要

AtCoder

- 長さNの数列$\{v_1,v_2,...v_N\}$がある($1 \leqq N \leqq 3{,}000$)
- ジグザグな部分列(長さ3以上)の中で最長のものの長さを求める.

ここで，ある部分列がジグザグであるとは，部分列を$\{X_1,X_2,...,X_N\}$とすると，

$$X_1 < X_2 > X_3 < X_4 < \ldots$$

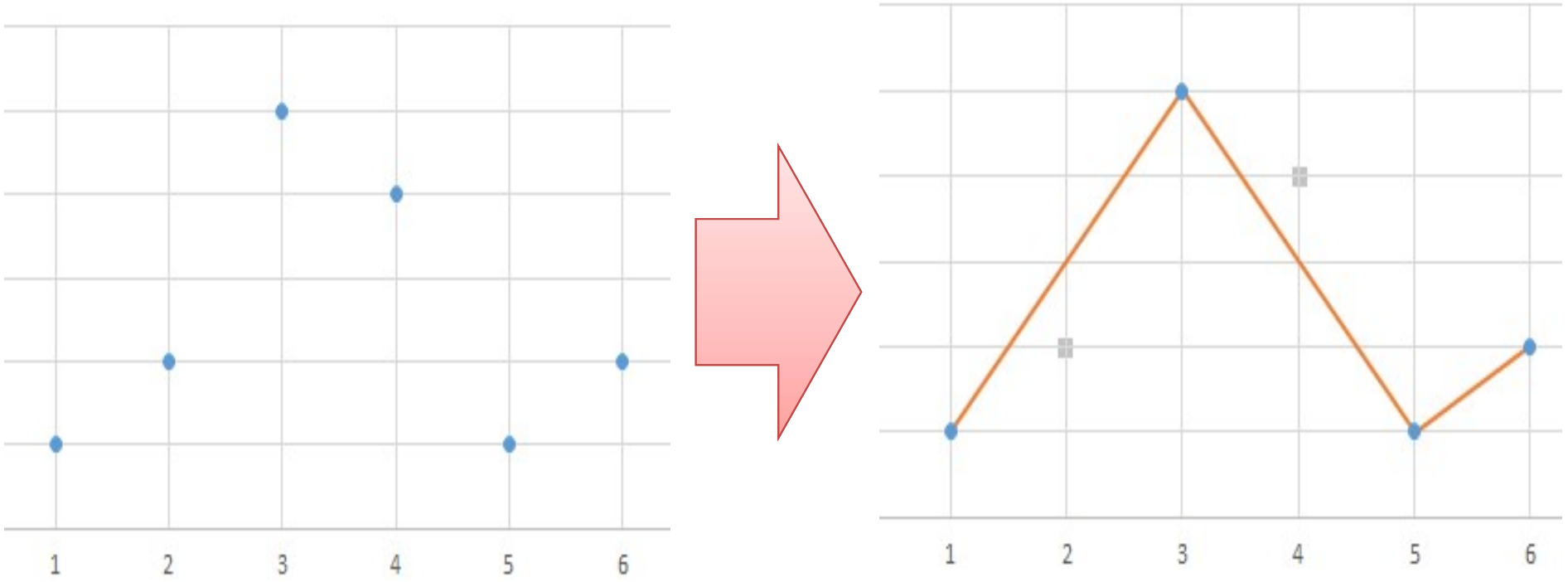
$$X_1 > X_2 < X_3 > X_4 < \ldots$$

のどちらか一方が成り立つことを言う.

E問題　問題概要(例)

AtCoder

- 以下の図の例(v={1,2,5,4,1,2}) では,
  最大4個の点が含まれる部分列{1,5,1,2}を作ることができる



E問題　解法(動的計画法) – 1/2

AtCoder

- 最長増加列を求める動的計画法と似ているが，以下を持つDP.
  - 現在DPで考えている部分列に次に追加する値は，部分列の末尾の値より
    大きくなるべきか小さくなるべきかを持つフラグ(2通り)
  - 現在DPで考えている部分列の末尾のインデックス(N通り)
- DP部分のコードは以下の通り

```
for(int i = 0 ; i < N ; i++){
        dp[i][0] = dp[i][1] = 1; // 初期化
        for(int j = 0 ; j < i ; j++){
                if( v[j] < v[i] ) dp[i][0] = max(dp[i][0], dp[j][1] + 1);
                if( v[j] > v[i] ) dp[i][1] = max(dp[i][0], dp[j][0] + 1);
        }
}
```

E問題　解法(動的計画法) – 2/2

AtCoder

- 状態がO(N)通り(2×N通り)，遷移はO(N)通り
  ∴時間計算量はO(N^2)
- 最大長が3未満の場合，0を出力しなければならないことに注意.

- 想定誤解答は，
  - TLE:「構築している部分列の末尾2つを保持してDP」→O(n^3)で遅い
  - WA:「ある場所から始めて，追加できるなら追加する」

でした.

E問題　余談

AtCoder

- セグメントツリー(BITも可)を用いると，
  １ ≦ N ≦ 100,000 でもぜんぜん解けます．
- 最長増加部分列問題に関して，プログラミングコンテストチャレンジブックに載っているようなセグメントツリーを用いないO(n log n)の DPが一般に知られていますが，そのような手法でこの問題を解こうとすると，割と大変です．
- この問題は，コードフェスティバルサブプロジェクトリーダー秋田毅さんのインターネット上のハンドルネームを眺めていたら思いつきました．

F問題　誤情報

F問題　問題概要

AtCoder

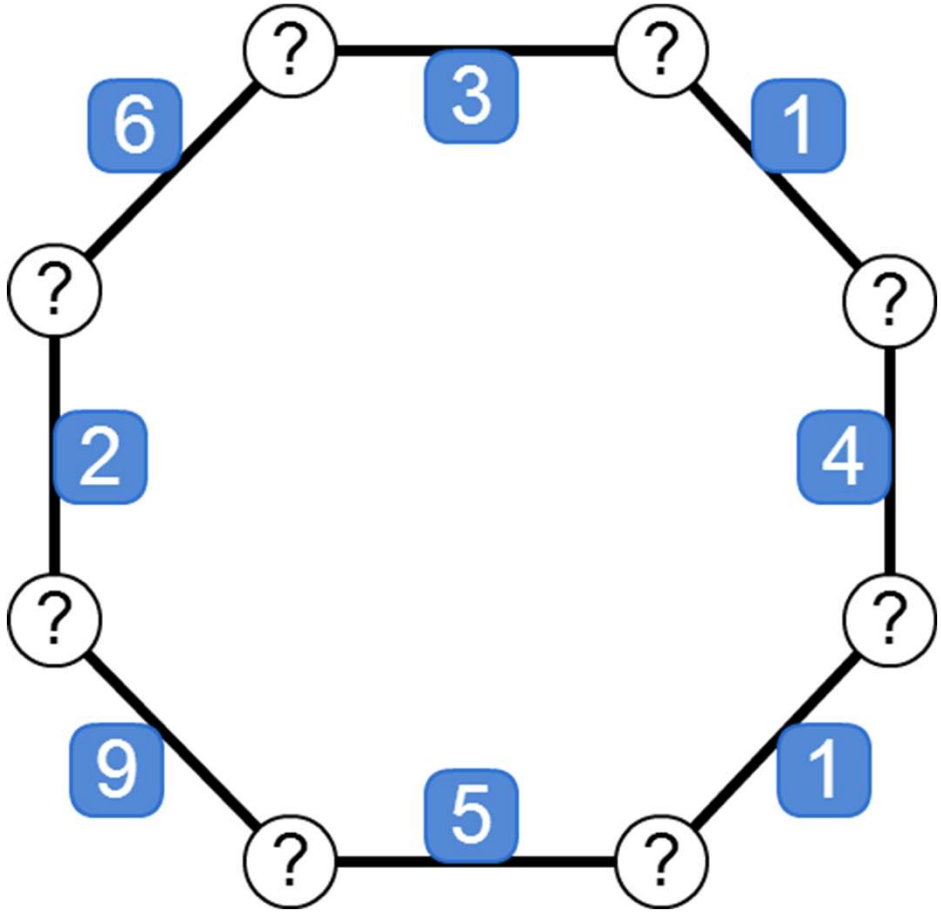
- サイズNの正整数列Aがある
- B[i] = gcd(A[i], A[i+1])
  - i = Nのときは gcd(A[N], A[1])
- B[i]にはいくつか誤りがある
- 考え得る誤りの個数のうち最小の物を求めよ。

- 1 ≦ N ≦ 10^5
- 1 ≦ B[i] ≦ 10^9

F問題　問題概要

AtCoder

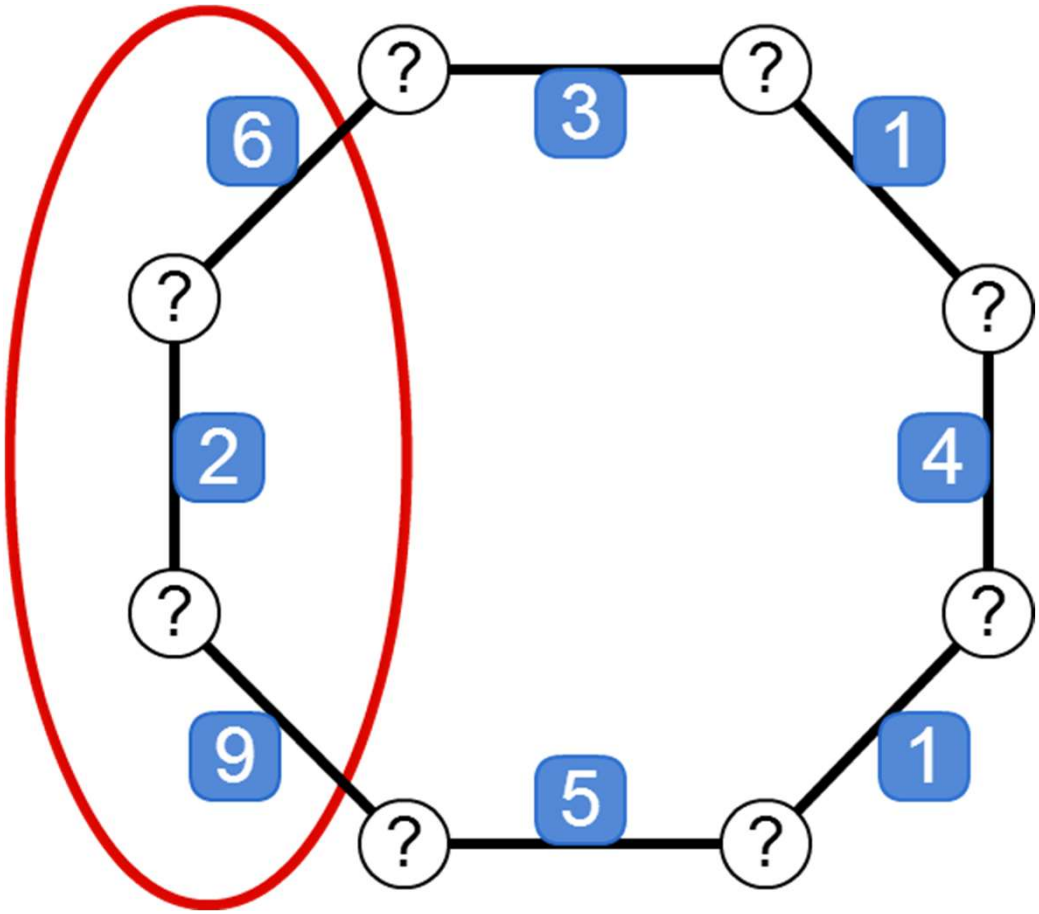
- 図示
  - 丸枠がA
  - 青い値がB
  - ぐるっと一周、となり合った値のgcd(最大公約数)を求めている



F問題　問題概要

AtCoder

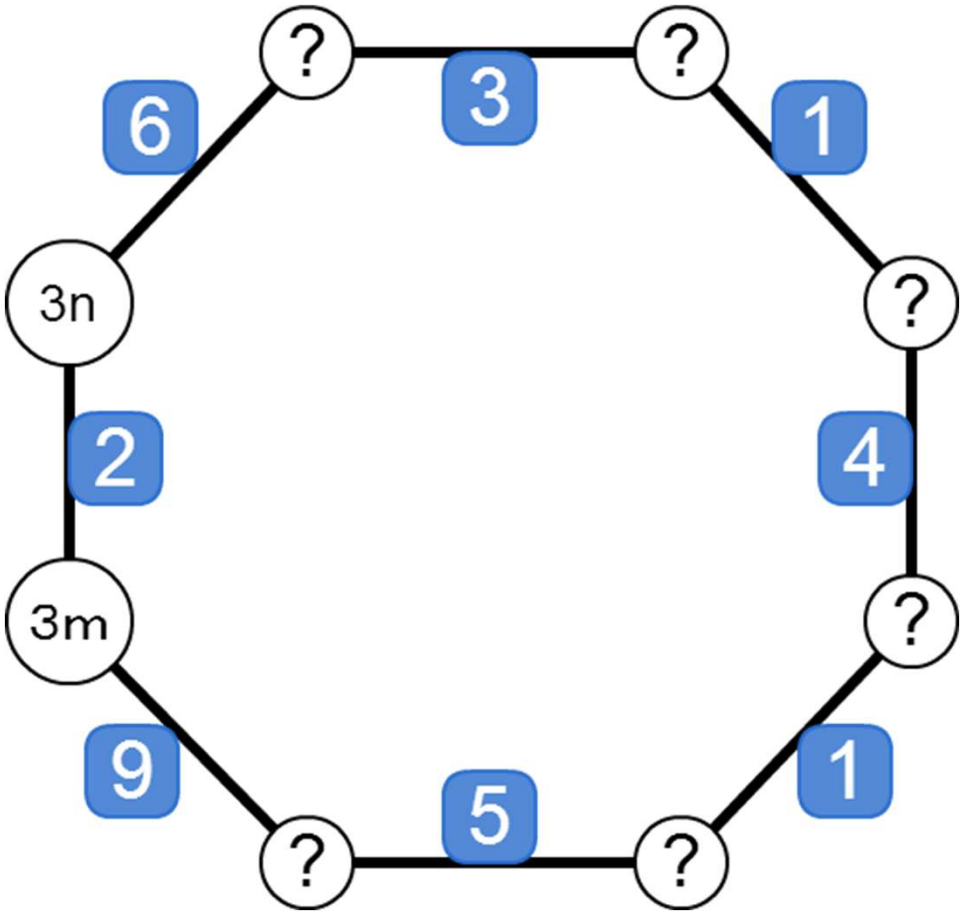
- 図示
  - 6,2,9の部分に注目してみる



F問題　問題概要

AtCoder

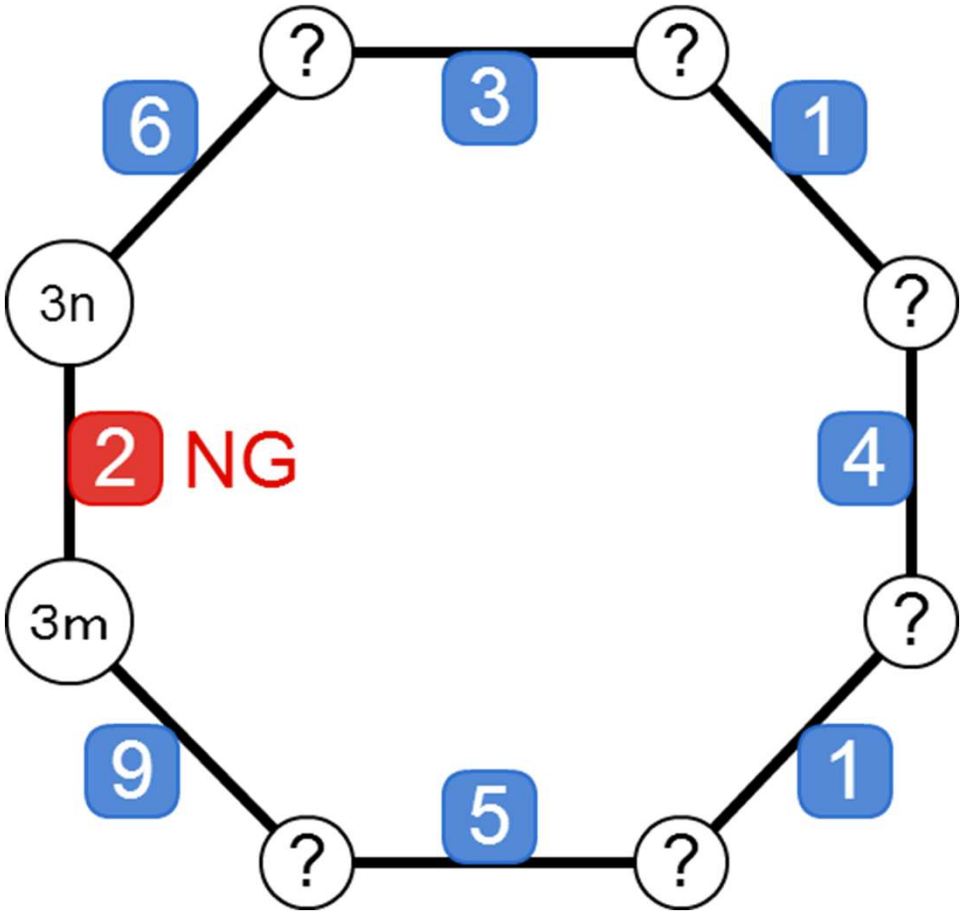
- 図示
  - 上の丸は6の倍数なので少なくとも3の倍数
  - 下の丸も同様に3の倍数



F問題　問題概要

AtCoder

- 図示
  - するとその最大公約数が2というのはおかしい
  - この6, 2, 9のうちいずれかは誤情報である



F問題　考察

AtCoder

- Bがどのようなときに矛盾が生じるのか？
- 先程の例: B =[9,2,6]
- A[1]もA[2]も3の倍数なのに最大公約数が2だというのはおかしい
- →B[i]はgcd(B[i-1], B[i+1]) の倍数でなければならない

考察

AtCoder

- B[i] が gcd(B[i-1], B[i+1])の倍数ならばどのような値でも大丈夫なのか？
  - 結論から言うと大丈夫

F問題　証明

AtCoder

- g = gcd(B[i-1], B[i+1])とする
  - B[i-1] = ag, B[i+1] = bg (a,bは互いに素)とする
  - B[i] = ngということになる
- Bに矛盾がないときA[i] = lcm(B[i-1], B[i])としても良い。(lcmは最小公倍数)
- B[i] = gcd(lcm(B[i-1], B[i]), lcm(B[i], B[i+1]),)

F問題　証明

AtCoder

- B[i] = gcd(lcm(B[i-1], B[i]), lcm(B[i], B[i+1]))

  = gcd(lcm(ag ,ng), lcm(ng, bg))

  = gcd(g * lcm(a, n), lcm(n, b))

  = g * gcd(lcm(a, n), lcm(n, b))

  = g * n * gcd(a / gcd(a,n), b / gcd(n,b))

  - ここでa, bは互いに素なので、約数同士も互いに素
  - よって gcd(a/gcd(a,n), b/gcd(n,b)) = 1

F問題　証明

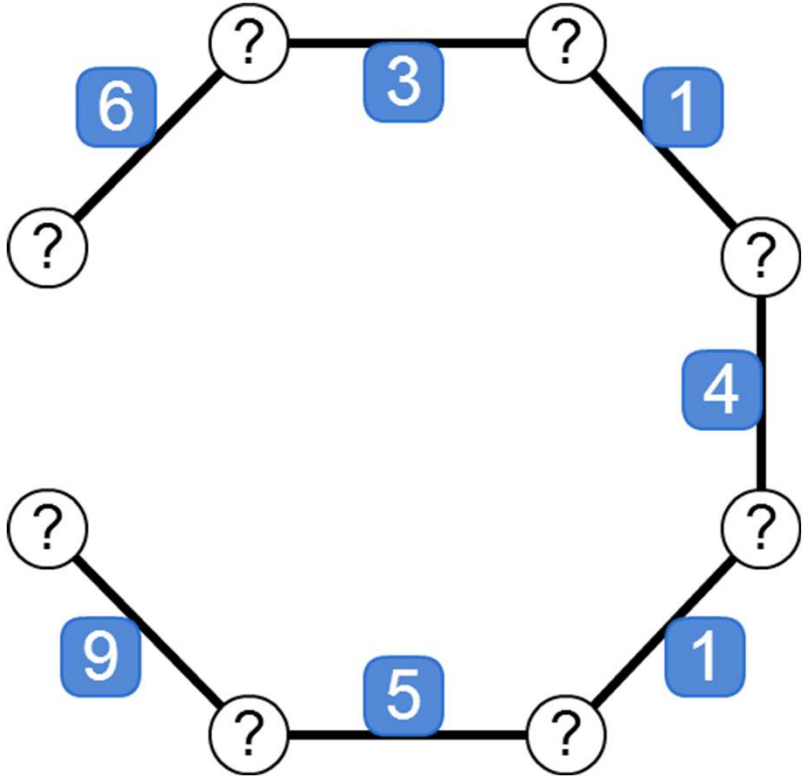
AtCoder

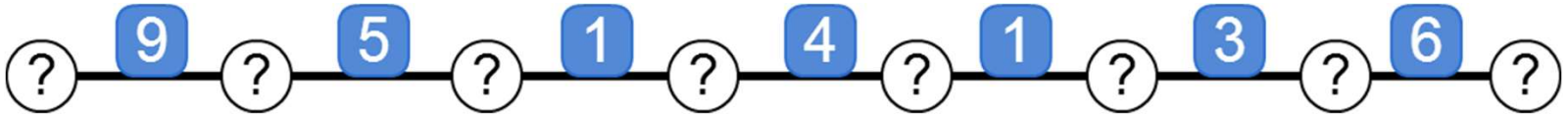
- B[i] = ngとなる
- よって「B[i-1], B[i], B[i+1]すべての情報が正しい」ことの必要十分条件は「B[i]がgcd(B[i-1], B[i+1])の倍数である」こと。

F問題　アルゴリズム

AtCoder

- まずB[N]が誤情報だと決めつけた時を考える
- B[N]がないものと考えても良い
- 輪っかになっていたものが一直線になって扱いやすくなる

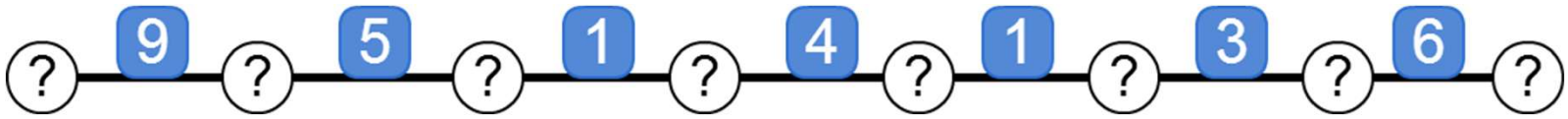




F問題　アルゴリズム

AtCoder

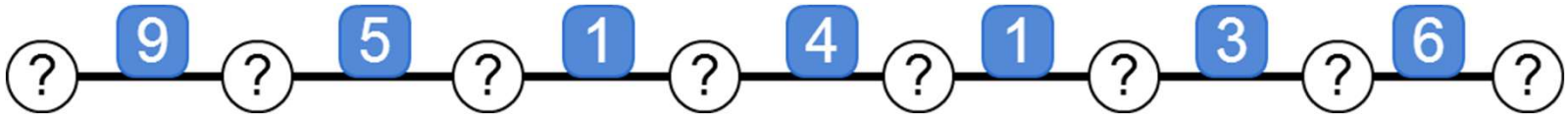
- 左から順番に使う(正しい情報とする)かどうかを決めていく
  - →2^Nの全探索
  - 明らかに間に合わない



F問題　アルゴリズム

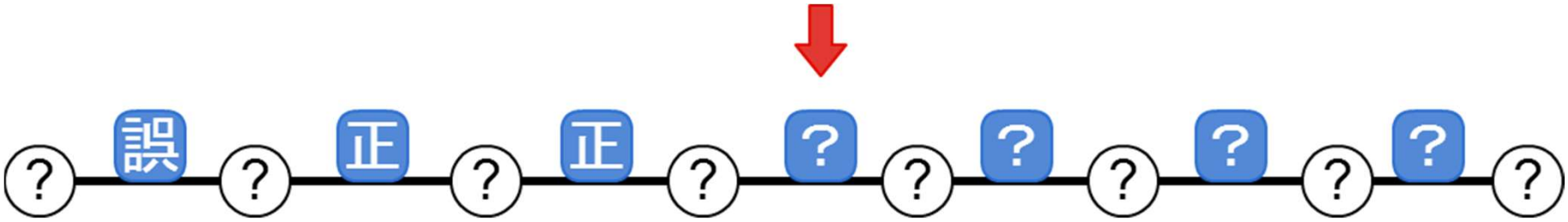
AtCoder

- 左から順番に使う(正しい情報とする)かどうかを決めていく
  - →2^Nの全探索
  - 明らかに間に合わない
- 実は矛盾がない限り貪欲に使ってよい



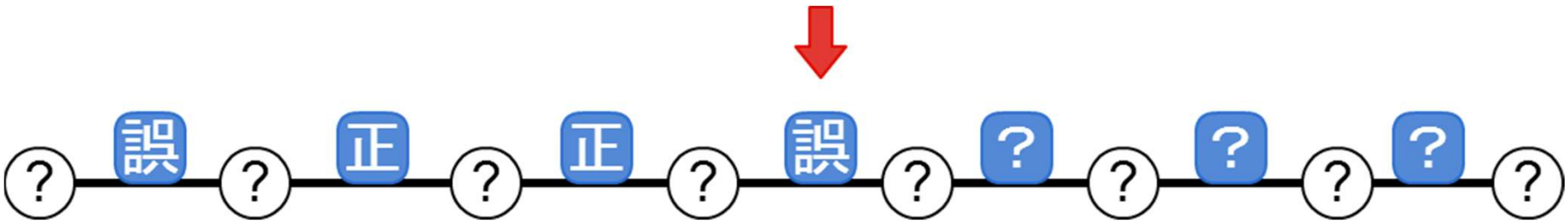
AtCoder

- 以下の赤矢印のところが正情報としても矛盾がないとする。
  - 正にも誤にもすることが出来る

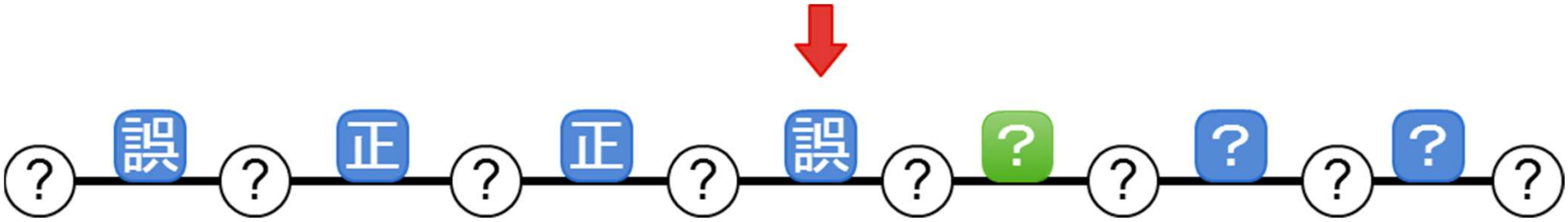


AtCoder

- 以下の赤矢印のところが正情報としても矛盾がないとする。
  - 正にも誤にもすることが出来る
- 誤にした方が最適であると仮定する

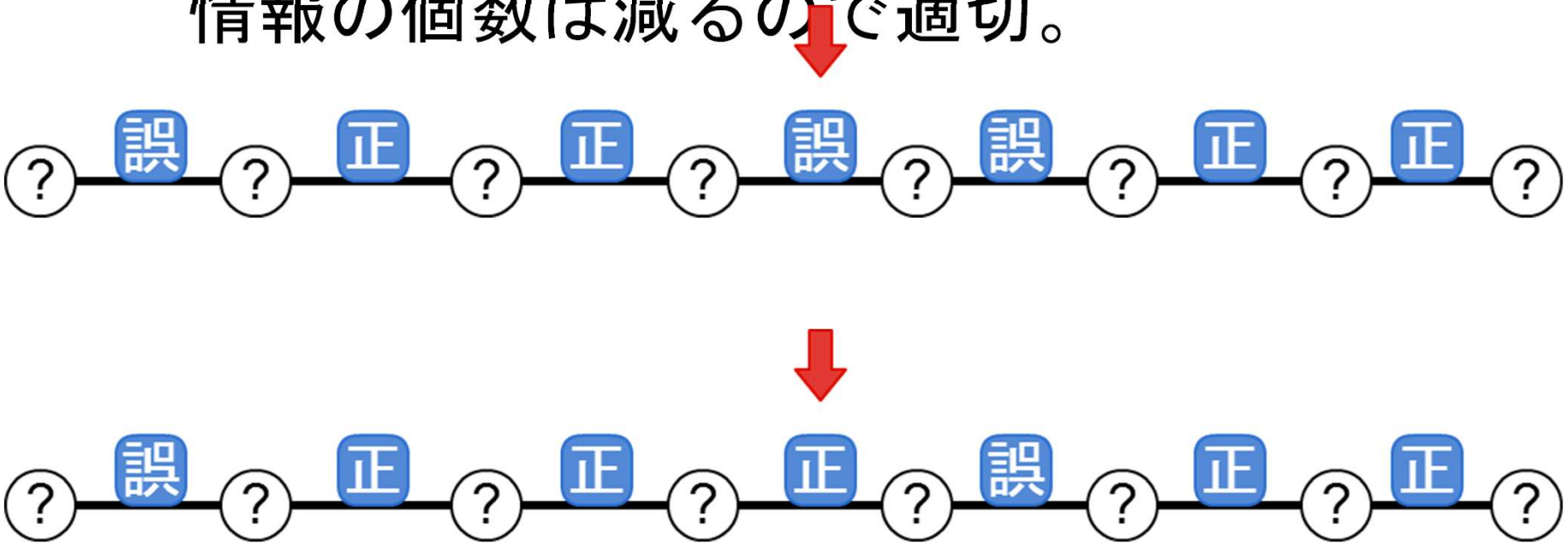


- 以下の赤矢印のところが正情報としても矛盾がないとする。
  - 正にも誤にもすることが出来る
- 誤にした方が最適であると仮定する
  - その最適解は赤矢印の次の情報(緑)の正誤によって2種類に分けることが出来る

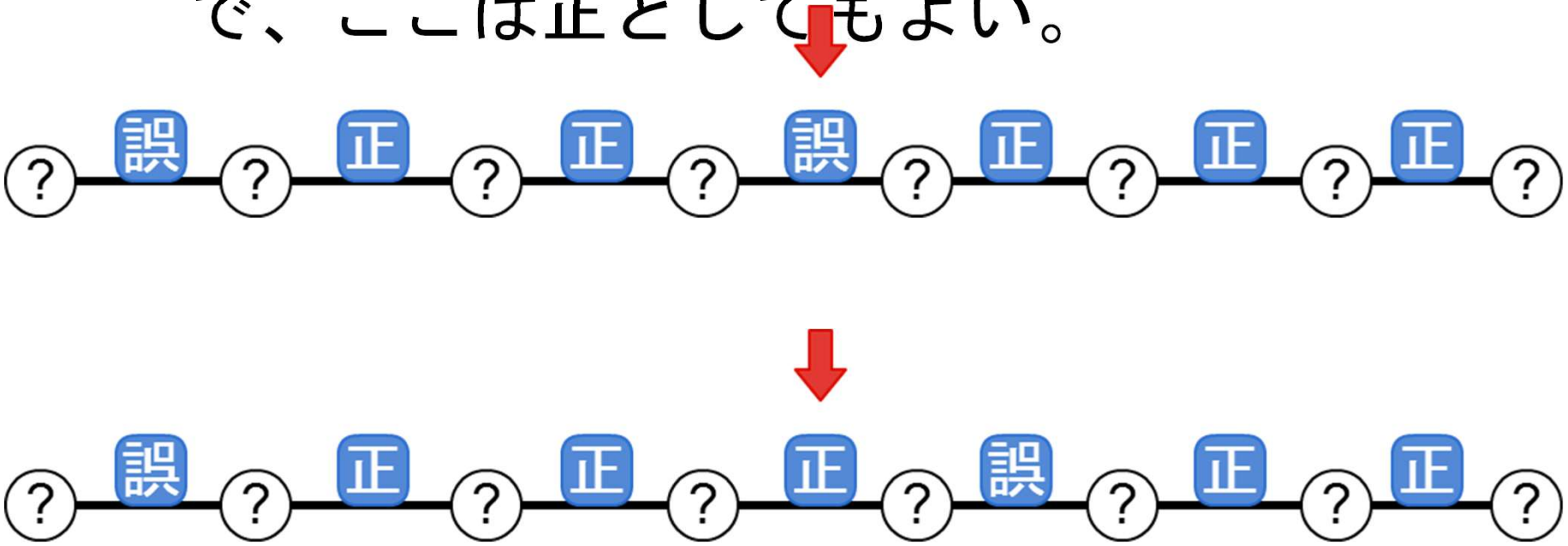


AtCoder

- 次の情報も誤の場合
  - 赤矢印の情報を正に変えても、ここ以降に影響が出ないので、ここは正としてもよい。誤情報の個数は減るので適切。



- 次の情報も正の場合
  - 赤矢印を正として、その次を誤に変換しても矛盾しない。また誤情報の和は変わらないので、ここは正としてもよい。



- よって、左から見ていき、貪欲に「矛盾がないならば正情報とする」というふうに正誤を決めれば誤情報が最小化される。

- 今回の問題はB[N]があるので少し工夫をしなければならない。
- まず、Bに誤情報が含まれているかどうか調べる。
  - 全ての隣りあう3要素について先ほどのチェックを行えば良い。
- 誤情報が含まれていなかったら0を出力
- 誤情報が含まれていたら、矛盾がおきた3要素のうち少なくとも1つは誤情報だとわかる

F問題　アルゴリズム

AtCoder

- B[k-1], B[k], B[k+1]で矛盾が起きたとする。
- このいずれかは誤情報なので
  - B[k-1]が誤情報だと決めつけたとき
  - B[k]が誤情報だと決めつけたとき
  - B[k+1]が誤情報だと決めつけたとき
- この3種類を全部試してその中で最も誤情報が少ないパターーンを出力すれば良い。
- O(N)

G問題　魔方陣

G問題　問題概要

AtCoder

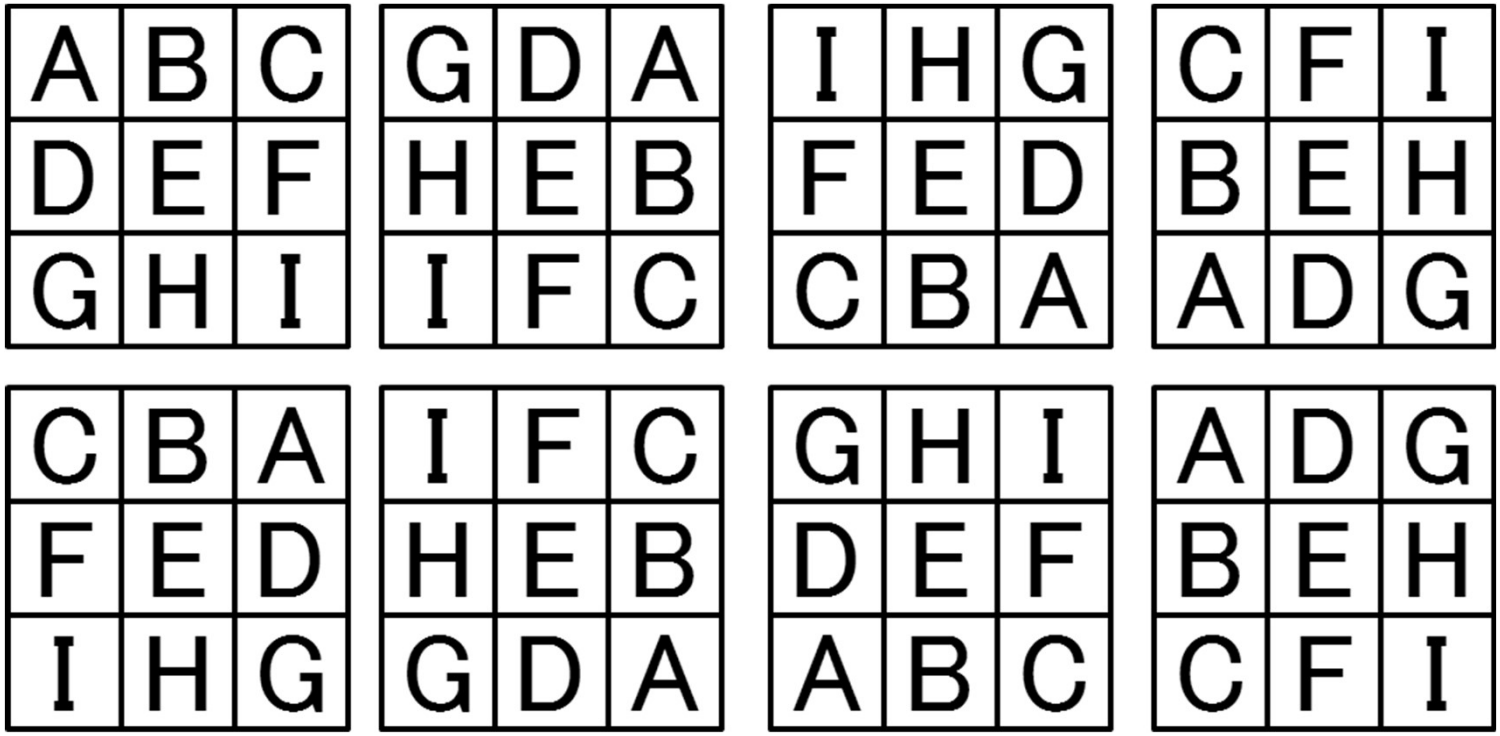
- 中央の値がNである積バージョンの魔方陣の個数を数え上げよ。ただし反転や回転で変換できるものは区別しないとする。

- 1 ≦ N ≦ 10^12

G問題　考察1

AtCoder

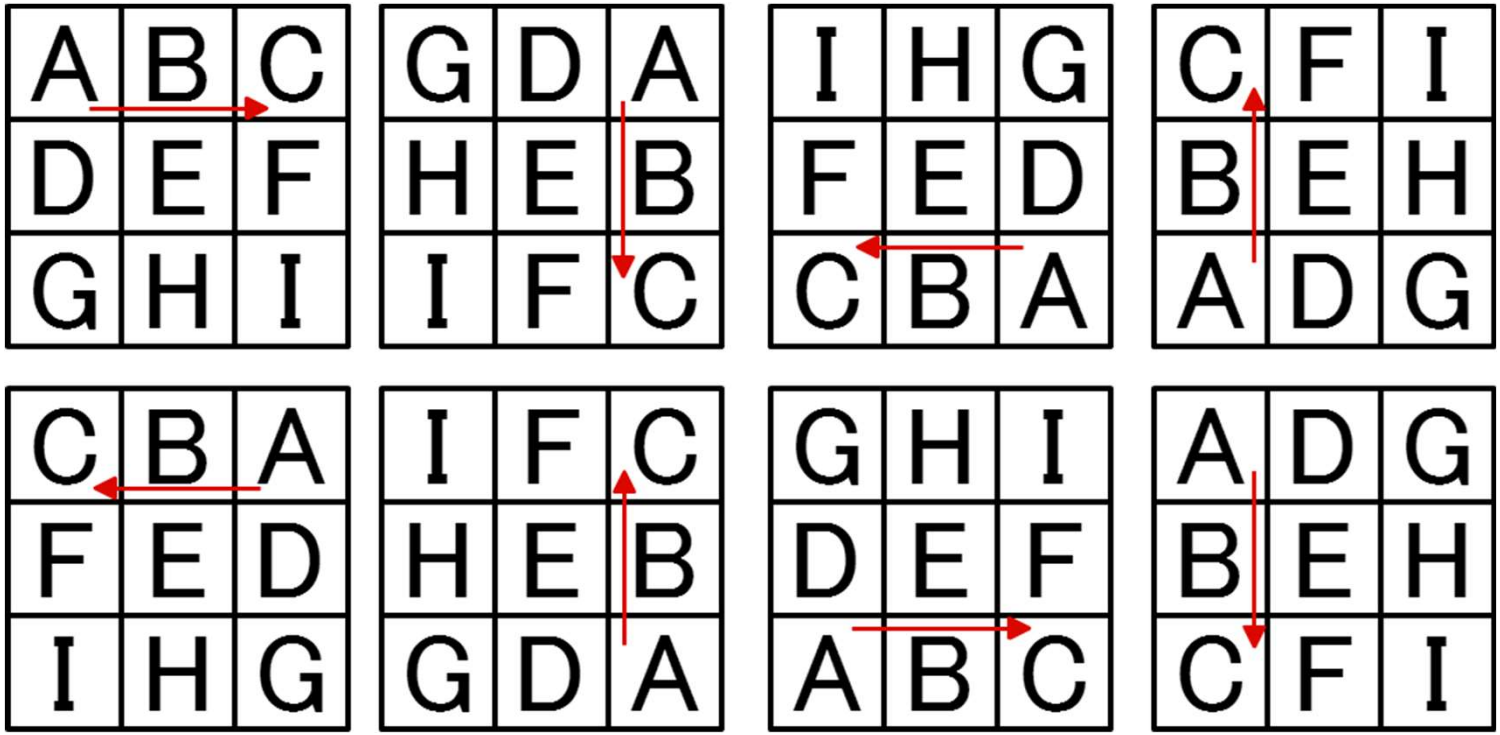
- もし反転や回転による変換を区別しなかった場合はどうなるのか？
- 魔方陣の各マスの値は全て違うのでちょうど8種類あることになる



G問題　考察1

AtCoder

- 回転や反転を区別せずに数え上げて、とから8で割れば答えが求まる



G問題　考察2

AtCoder

- 完成された魔方陣の各数字を素因数分解してみる。

5	100	2
4	10	25
50	1	20

5	2^2 5^2	2
2^2	2 5	5^2
2 5^2	1	2^2 5

G問題　考察2

AtCoder

- 条件よりどの列も積が等しい。素数ごとに考えても同様に積が等しい

1	2^2	2
2^2	2	1
2	1	2^2

5	5^2	1
1	5	5^2
5^2	1	5

G問題　和バージョンに還元

AtCoder

- 1種類の素数からなるので、積が等しいということは指数の和が等しいことと同値。

1	2^2	2
2^2	2	1
2	1	2^2

共に積が素数の3乗

5	5^2	1
1	5	5^2
5^2	1	5

0	2	1
2	1	0
1	0	2

共に和が3

1	2	0
0	1	2
2	0	1

G問題　和バージョンに還元

AtCoder

- 積の魔方陣はいくつかの和の魔方陣を合成したものだと考えることが出来る。

5	100	2
4	10	25
50	1	20

0	2	1
2	1	0
1	0	2

1	2	0
0	1	2
2	0	1

- 元　　素因数2　　素因数5

G問題　和バージョンに還元

AtCoder

- 和バージョンは同じ値が複数のマスに現れるということもある

5	100	2
4	10	25
50	1	20

0	2	1
2	1	0
1	0	2

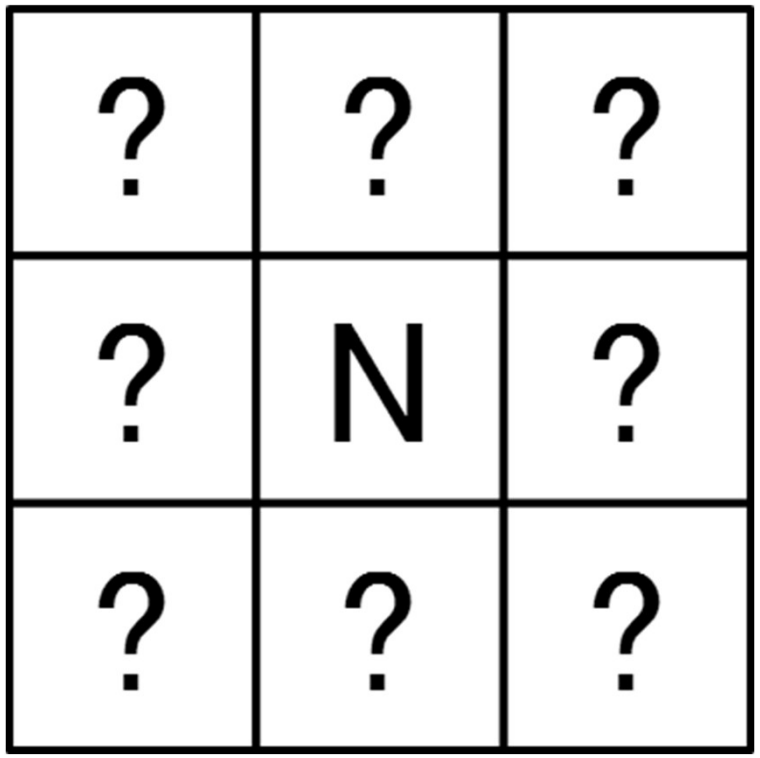
1	2	0
0	1	2
2	0	1

- 元　　素因数2　　素因数5

G問題　考察3

AtCoder

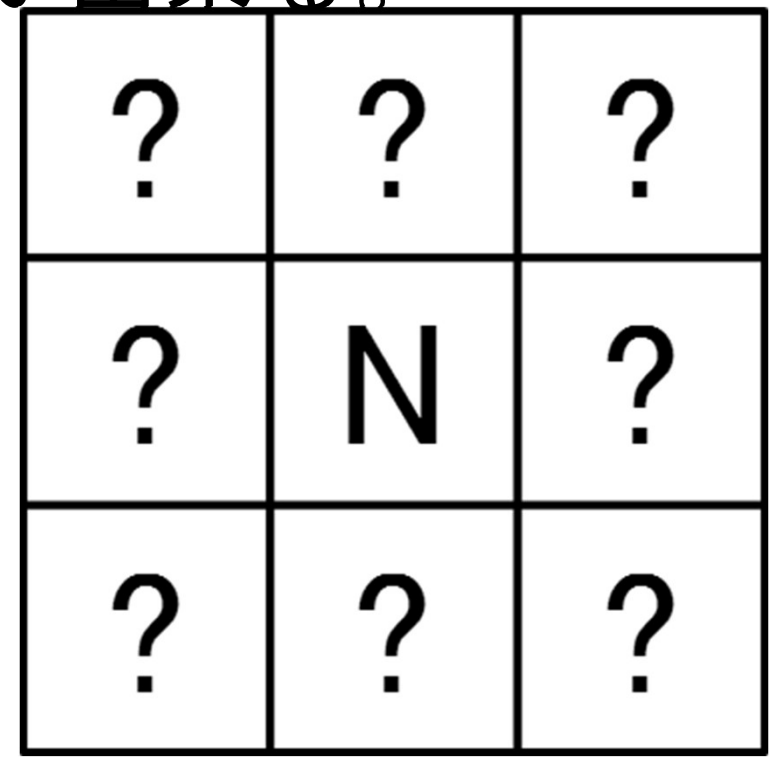
- 和バージョンはどのように数え上げればよいだろうか？



G問題　考察3

AtCoder

- 各列の和が等しい性質を利用して、いくつかのマスを決めるだけで、残りを計算することが出来る。



G問題　考察3

AtCoder

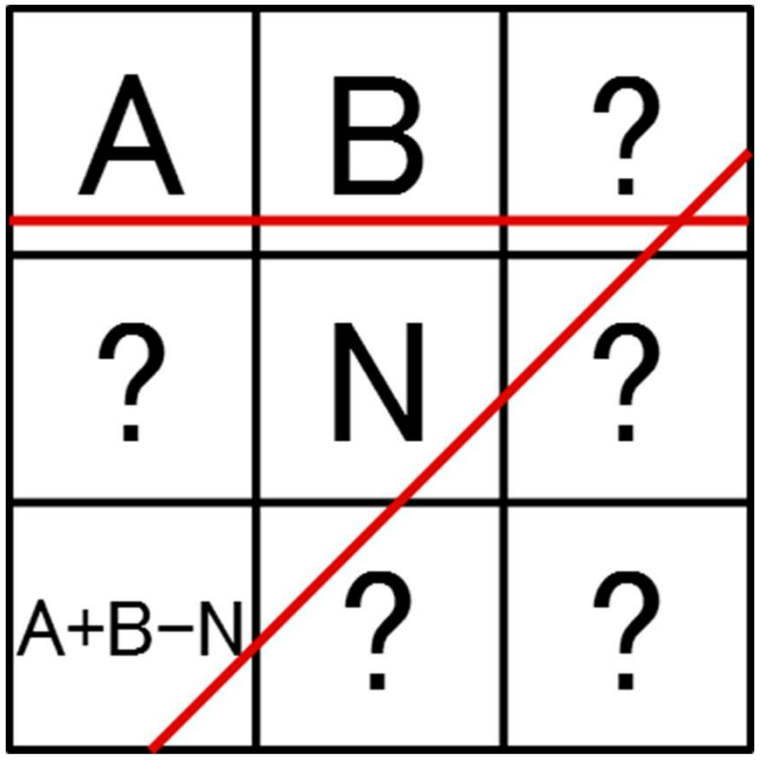
- 2つ数字を決め打ちしてみる。

A	B	?
?	N	?
?	?	?

G問題　考察3

AtCoder

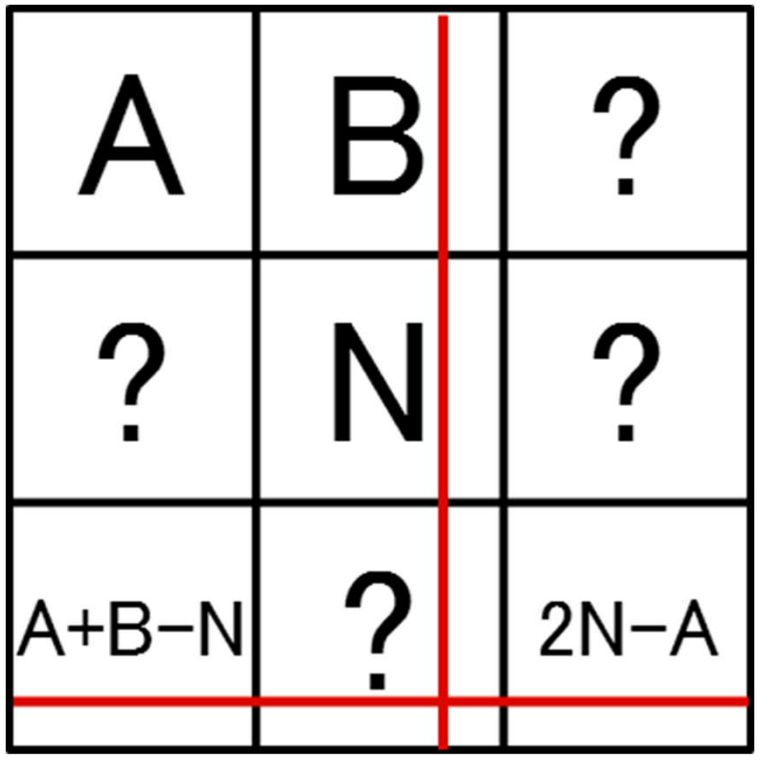
- この2列の和が等しいので、左下が決まる



G問題　考察3

AtCoder

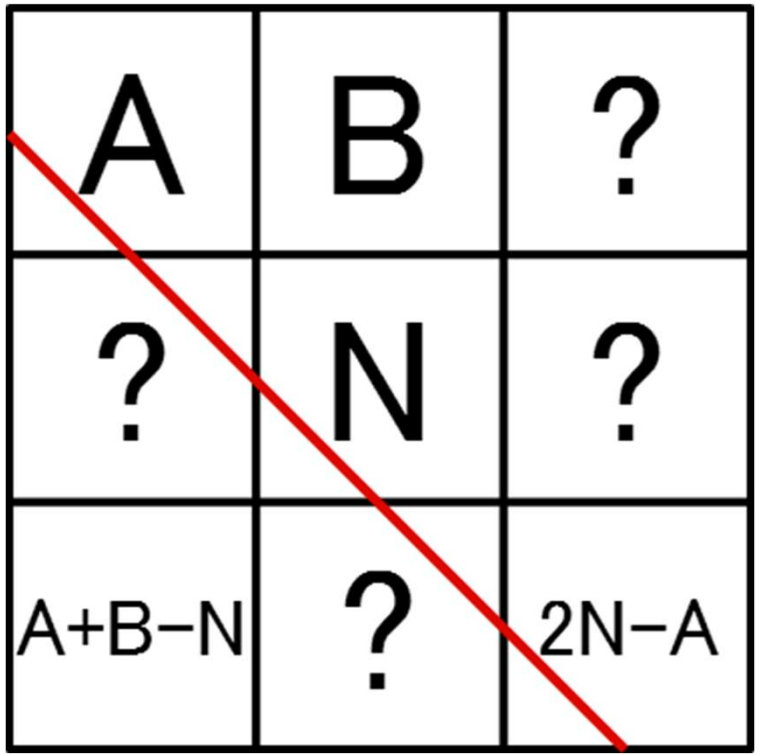
- 同様にこの2列で考えると、右下が決まる



G問題　考察3

AtCoder

- この列から、各列の和は3Nという事がわかる。
  - Nにしか依存しない！



G問題　考察3

AtCoder

- 列の和がわかったので残りは簡単に埋めることができる

A	B	3N−A−B
4N−2A−B	N	2A+B−2N
A+B−N	2N−B	2N−A

G問題　考察3

AtCoder

- 各マスの値は、元は素因数の指数なので0以上でなければならない。

A	B	3N−A−B
4N−2A−B	N	2A+B−2N
A+B−N	2N−B	2N−A

G問題　考察3

AtCoder

- 各マスの値は、元は素因数の指数なので0以上でなければならない。
- A,2N-A, B, 2N-Bが全て0以上なので、A,Bは０以上2N以下

A	B	3N−A−B
4N−2A−B	N	2A+B−2N
A+B−N	2N−B	2N−A

G問題　考察3

AtCoder

- A,Bにそれぞれ0以上2N以下の整数を当てはめて、各マスの値が0以上かどうか確認すれば魔方陣を重複無く全て列挙することが出来る
  - O(N^2)

A	B	3N−A−B
4N−2A−B	N	2A+B−2N
A+B−N	2N−B	2N−A

G問題　考察4

AtCoder

- 各素因数について和バージョンに還元して、それぞれの個数をただ掛けあわせて8で割った値が答えになるのだろうか？

G問題　考察4

AtCoder

- 各素因数について和バージョンに還元して、それぞれの個数をただ掛けあわせて8で割った値が答えになるのだろうか？
  - 答えはNO

G問題　考察4

AtCoder

- 場合によっては、和バージョンを組み合わせた時に積バージョンに同じ値が複数マスに現れることもある。

1	100	10
100	10	1
10	1	100

0	2	1
2	1	0
1	0	2

0	2	1
2	1	0
1	0	2

G問題　考察4

AtCoder

- 和バージョンの個数をただ掛けあわせた値から、下図のようなパターンの個数を引かなければならない。

1	100	10
100	10	1
10	1	100

0	2	1
2	1	0
1	0	2

0	2	1
2	1	0
1	0	2

G問題　考察4

AtCoder

- 下図のようなパターンはどのようなときに発生するのか？

1	100	10
100	10	1
10	1	100

0	2	1
2	1	0
1	0	2

0	2	1
2	1	0
1	0	2

G問題　考察4

AtCoder

- 同じ値が複数のマスに現れる魔方陣は対称性から以下の4種類に分けることが出来る

16	16	16
16	16	16
16	16	16

8	32	16
32	16	8
16	8	32

8	64	8
16	16	16
32	4	32

4	32	32
128	16	2
8	8	64

G問題　考察4

AtCoder

- 同じ値が複数のマスに現れる魔方陣は対称性から以下の4種類に分けることが出来る
  - 全探索して、値が9種類未満しか出てこないものを列挙するとわかる

A	A	A
A	A	A
A	A	A

A	C	B
C	B	A
B	A	C

B	D	B
A	A	A
C	E	C

C	A	A
D	E	F
B	B	G

G問題　考察4

AtCoder

- パターン1
  - 上下、左右、対角線すべてで線対称なパターン
  - 1種類の数字からなる魔方陣はこれのみ

A	A	A
A	A	A
A	A	A

AtCoder

- パターン2
  - 一つの対角線のみで線対称なパターン
  - 3種類の数字からなる魔方陣はこれのみ
  - 対称軸によって2種類に分けられる

A	C	B
C	B	A
B	A	C

B	C	A
A	B	C
C	A	B

G問題　考察4

AtCoder

- パターン3
  - 上下もしくは左右のみ線対称なパターン
  - 5種類の数字からなる魔方陣はこれのみ
  - 対称軸によって2種類に分けられる

B	D	C
A	A	A
B	E	C

B	A	B
D	A	E
C	A	C

G問題　考察4

AtCoder

- パターン4
  - 特に対称ではないが、同じ数が出てくるパターン
  - 7種類の数字からなる魔方陣はこれのみ
  - 回転や反転により4種類に分けられる

C	A	A
D	E	F
B	B	G

B	D	C
B	E	A
G	F	A

A	A	C
F	E	D
G	B	B

G	F	A
B	E	A
B	D	C

G問題　考察4

AtCoder

- 同じ対称軸を持つパターンのみを足し合わせると、出来上がった積バージョンの魔方陣も、同じ対称軸を持ってしまう。

A	A	A
A	A	A
A	A	A

A	C	B
C	B	A
B	A	C

B	D	B
A	A	A
C	E	C

C	A	A
D	E	F
B	B	G

G問題　考察4

AtCoder

- 下図は、右上から左下への対角線を対称軸とする物のみを足し合わせたため、できた積バージョンもパターン2になってしまっている

1	100	10
100	10	1
10	1	100

0	2	1
2	1	0
1	0	2

0	2	1
2	1	0
1	0	2

G問題　考察4

AtCoder

- 同じパターンとパターン1のみを足しあわせた時だけ、出来上がる積バージョンの魔方陣もそのパターンに当てはまり、同じ値を複数のマスに持つようになる
- パターン1が1種類、パターン2が2種類、パターン3が2種類、パターン3が4種類ある
  - パターン1のみが他のパターンにも重複して数え上げられるのであとから差し引かなければならない
- それぞれうまく計算して最後に引き算をすればついに答えが求まる

G問題　アルゴリズム

AtCoder

- Nを素因数分解して、各素因数が何回かけられているか調べる
- 各素因数について和バージョンの魔方陣に還元して全探索をする
  - O(中央の値^2) = O(log(N)^2)
- 1種類の値からなるもの、3種類からの値からなるもの、5種類からなるもの、7種類からなるものをそれぞれ別々に数え上げる
  - 同じ値が複数のマスに現れるパターンの個数を最後に引くときに使う

G問題　アルゴリズム

AtCoder

- i番目の素因数について1,3,5,7,9種類の値からなる魔方陣の個数をA[i],B[i],C[i],D[i],E[i]とする
- これらから構成される積バージョンの中で
  - 1種類の値からなる物の個数はA[i]の総積
  - 3種類の物の個数はB[i]/2+A[i]の総積の2倍-2
  - 5種類の物の個数はC[i]/2+A[i]の総積の2倍-2
  - 7種類の物の個数はD[i]/4+A[i]の総積の4倍-4
    - 最後の-2や-4は重複して数えられている1種類の値からなる物の個数である
  - 9種類の値からなる物の個数は(A[i]+B[i]+C[i]+D[i]+E[i])の総積

G問題　アルゴリズム

AtCoder

- 9種類の値からなるもの個数から9種類未満の値からなるものを引くと、回転や反転による変換も区別して数え上げた結果がでてくる
- 最後にこの値を8で割ると答えが求まる
- O(log(N)^2)

H問題　部屋割り

H問題　問題概要

AtCoder

- N人がK部屋に泊まる
- 先頭の人から順番に泊まる部屋を決めていく
  - 人数の多い部屋が好きな人は、現在人数が最も多い部屋に泊まる
  - 人数の少ない部屋が好きな人は、現在人数が最も少ない部屋に泊まる
  - どちらの人も、同じ条件の部屋が複数あった場合、等しい確率で部屋を選ぶ
- それぞれの人が、一緒に泊まることになる人数の期待値を出力しなさい。

- 制約
  - $1 \leqq N,K \leqq 200,000$

H問題　問題概要

AtCoder

- 例えば、N=7、K=3、S=100011の場合
  - 一人目は、最も人数の多い部屋に入る
    - 0人の部屋に入り、部屋の状況は{1, 0, 0}
  - 二人目は、最も人数の少ない部屋に入る
    - 0人の部屋に入り、部屋の状況は{1, 1, 0}
  - 三人目は、最も人数の少ない部屋に入る
    - 0人の部屋に入り、部屋の状況は{1, 1, 1}
  - 四人目は、最も人数の少ない部屋に入る
    - 1人の部屋に入り、部屋の状況は{2, 1, 1}
  - 五人目は、最も人数の少ない部屋に入る
    - 1人の部屋に入り、部屋の状況は{2, 2, 1}
  - 六人目は、最も人数の多い部屋に入る
    - 2人の部屋に入り、部屋の状況は{3, 2, 1}
  - 七人目は、最も人数の多い部屋に入る
    - 3人の部屋に入り、部屋の状況は{4, 2, 1}

H問題　問題概要

AtCoder

- 例えば、N=7、K=3、S=100011の場合
  - １、２、３人目は、一人の部屋、二人の部屋、四人の部屋に入る確率が等確率で存在する
    - よって、期待値は2.333333333333…
  - ４，５人目は、一人の部屋に入ることは絶対にないため、二人の部屋か四人の部屋のどちらか
    - 期待値は3
  - ６，７人目は、四人の部屋に確実に入る
    - 期待値は4

H問題　アルゴリズム

AtCoder

- 考察1
- ランダムに選択したとしても、人数の分布自体は変わらない
  - {1, 0, 0}でも{0, 1, 0}でも、「一人の部屋が1つ、0人の部屋が2つ」であることは変わらない。
- つまり、それぞれの人が、「何人の部屋に入るか」は、シミュレーションすることで簡単に求めることが出来る。

H問題　アルゴリズム

AtCoder

- 考察2
- 最後の人が選び終わった時、「N人部屋にいる人が一緒に泊まる人数はN人」であることはすぐに分かる。
- その直前では、「最後の人が入った人数の部屋」の期待値は変わるが、それ以外は変わっていない。
  - よって、「最後の人が入った人数の部屋」に対して、期待値を再計算してあげることにより、後ろからさかのぼって行くことが出来る。

H問題　アルゴリズム

AtCoder

- サンプル1での実験
  - 最後の人が選び終わった時、部屋の状況は{4, 2, 1}
    - この時、
      - 4人部屋の期待値は4
      - 2人部屋の期待値は2
      - 1人部屋の期待値は1
  - その直前は、{3, 2, 1}
    - この時、3人部屋の期待値は、一つ前の4人部屋の期待値と同じなので4
  - その直前は、{2, 2, 1}
    - この時、2人部屋の期待値は、新たに増えた2人部屋の期待値が4、元々あった2人部屋の期待値が2であるため、平均は3

H問題　アルゴリズム

AtCoder

- 動的計画法の計算式

num[i] : i人部屋の存在個数
sum[i] : num[i] * sum[i]
choose[i] : i番目の人が何人の部屋に入ったか
ret[i] : i番目の人の期待値

```
for(int i = N-1; i >= 0; i++){
        ret[i] = sum[choose[i] + 1] / num[choose[i] + 1];
        sum[choose[i] +1] -= ret[i];
        sum[choose[i]] += ret[i];
        num[choose[i]+1]--;
        num[choose[i]]++;
}
```

H問題　アルゴリズム

AtCoder

- 動的計画法のイメージ
  - 後ろから計算していく

部屋の人数	0	1	2	3	4
部屋の数	0	1	1	0	1
期待値	-	1	2	-	4
数×期待値	0	1	2	0	4

H問題　アルゴリズム

AtCoder

- 動的計画法のイメージ
  - 後ろから計算していく
    - 7番目の人は、3人部屋に入って4人部屋に
    - 4人部屋は期待値4なので、7番目の人の期待値は4

部屋の人数	0	1	2	3	4
部屋の数	0	1	1	0	1
期待値	-	1	2	-	4
数×期待値	0	1	2	0	4

H問題　アルゴリズム

AtCoder

- 動的計画法のイメージ
  - 後ろから計算していく
    - 7番目の人は、3人部屋に入って4人部屋に
    - 4人部屋は期待値4なので、7番目の人の期待値は4
    - 4人部屋の一人を3人部屋に移動させて、DPの表を更新する

部屋の人数	0	1	2	3	4
部屋の数	0	1	1	1	0
期待値	-	1	2	4	-
数×期待値	0	1	2	4	0

H問題　アルゴリズム

AtCoder

- 動的計画法のイメージ
  - 後ろから計算していく
    - 6番目の人は、2人部屋に入って3人部屋に
    - 3人部屋は期待値4なので、6番目の人の期待値は4
    - 3人部屋の一人を2人部屋に移動させて、DPの表を更新する

部屋の人数	0	1	2	3	4
部屋の数	0	1	1	1	0
期待値	-	1	2	4	-
数×期待値	0	1	2	4	0

H問題　アルゴリズム

AtCoder

- 動的計画法のイメージ
  - 後ろから計算していく
    - 6番目の人は、2人部屋に入って3人部屋に
    - 3人部屋は期待値4なので、6番目の人の期待値は4
    - 3人部屋の一人を2人部屋に移動させて、DPの表を更新する
      - 最終的に4人部屋になる部屋1つと、2人部屋になる部屋1つが混ざるので、期待値は3となる。

部屋の人数	0	1	2	3	4
部屋の数	0	1	2	0	0
期待値	-	1	3	-	-
数×期待値	0	1	6	0	0

H問題　アルゴリズム

AtCoder

- 動的計画法のイメージ
  - 後ろから計算していく
    - 5番目の人は、1人部屋に入って2人部屋に
    - 2人部屋は期待値3なので、6番目の人の期待値は3
    - 2人部屋の一人を1人部屋に移動させて、DPの表を更新する

部屋の人数	0	1	2	3	4
部屋の数	0	1	2	0	0
期待値	-	1	3	-	-
数×期待値	0	1	6	0	0

H問題　アルゴリズム

AtCoder

- 動的計画法のイメージ
  - 後ろから計算していく
    - 5番目の人は、1人部屋に入って2人部屋に
    - 2人部屋は期待値3なので、6番目の人の期待値は3
    - 2人部屋の一人を1人部屋に移動させて、DPの表を更新する

部屋の人数	0	1	2	3	4
部屋の数	0	2	1	0	0
期待値	-	2	3	-	-
数×期待値	0	4	3	0	0

H問題　アルゴリズム

AtCoder

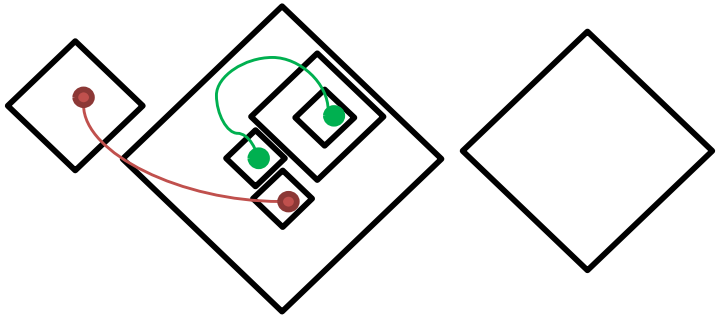
- 計算量
  - 最終状態までのシミュレーション O(N)
  - DPによる更新
    - 更新回数O(N)
    - 1回の更新にかかる計算量 O(1)
  - よって、全て合わせてO(N)で計算出来る。
- 別解として、分布の特徴を生かして場合分けするものもあり。
  - 相当ミスりやすいのでお勧めはしません

I問題　SHAPES

問題概要

AtCoder

- 右図のようなたくさんの45度回転した正方形が
  中心座標と，辺までのマンハッタン距離で与えられる.
- 互いの正方形は交差しない.

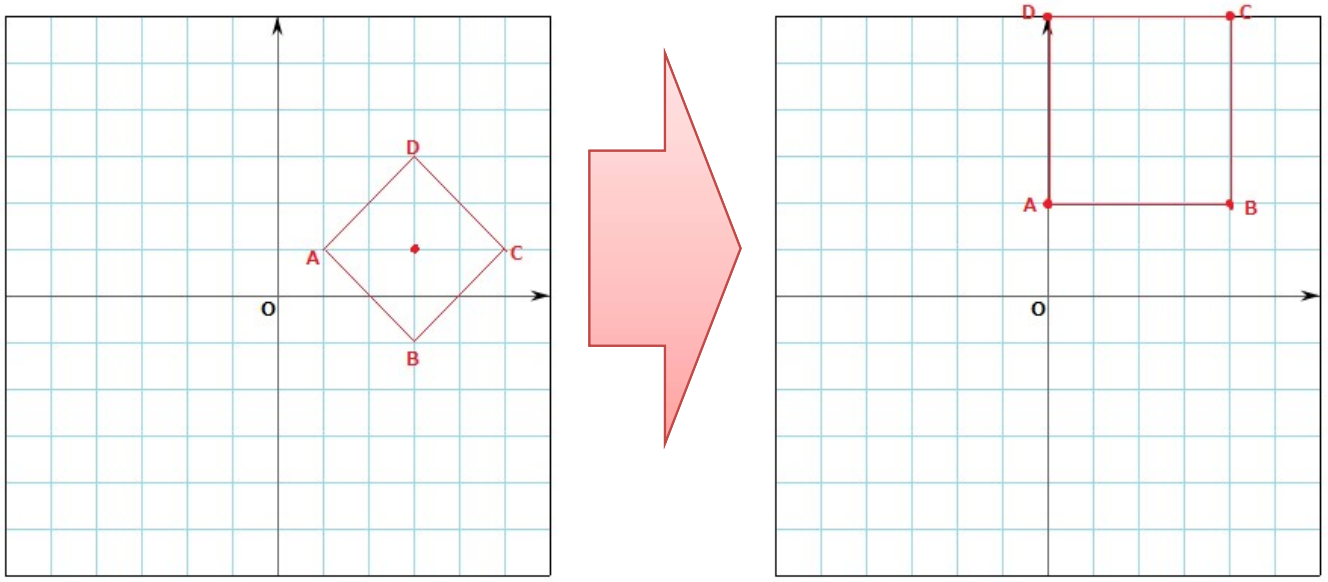


- 加えてある点からある点に移動したいというクエリ
  が複数与えられる.
- 各クエリで，できるだけ線を跨がずにその点に到達したい.
  その最小数はいくつか？

考察 1/2

AtCoder

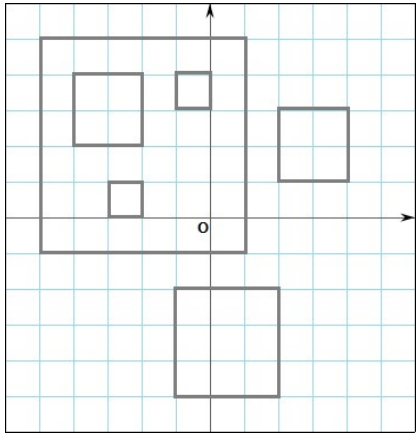
- 一般に，マンハッタン距離の境界は正方形を45度回転させた形になる.
- 原点に対して45度回転すると軸に平行な正方形集合になるので，
  全ての座標点を± 45度回転して√2倍しておけば整数点になって考えやすそう

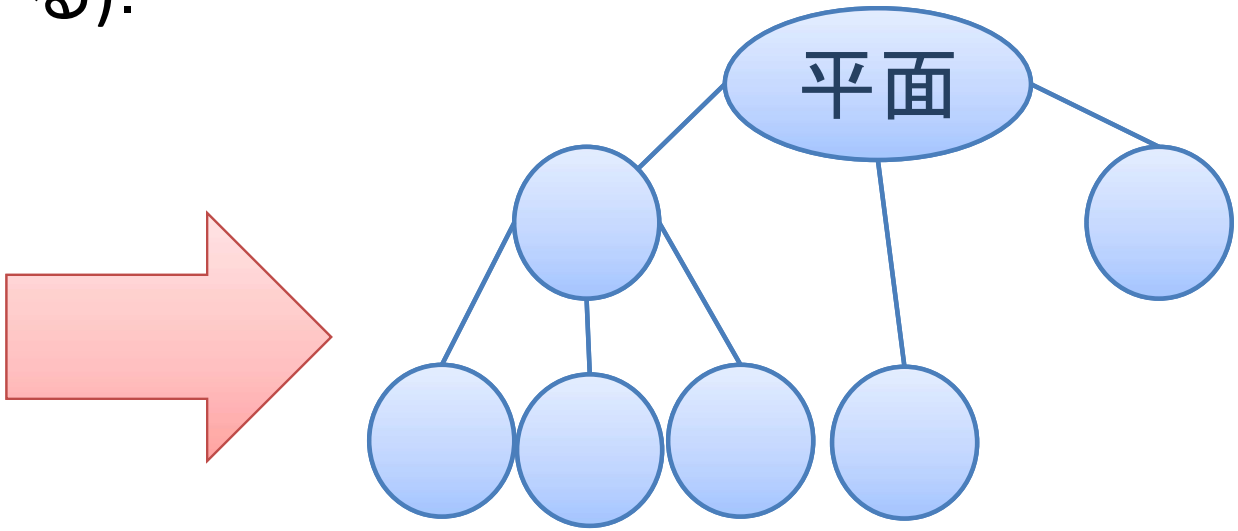


考察 2/2

AtCoder

- 軸に平行なので平面走査とかが使えないか？
- 正方形をグラフの頂点としたとき,
  ある正方形を包含する最小の大きさの正方形を親とすれば全体は木構造になっている(平面は全ての正方形を含むとする).



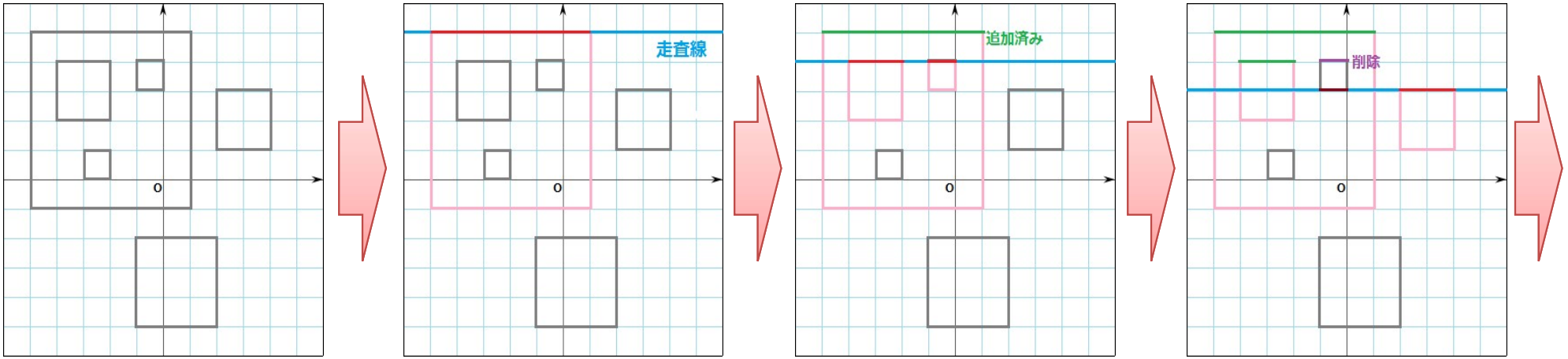


- クエリで与えられる2点がどこの正方形に含まれるか分かれば,
  木上の最短経路長がそのまま求めるものになるので,
  この木を構築すればよさそう

解法 1/3

AtCoder

- 与えられる全ての座標 (x,y) を (x-y,x+y) というふうに45度回転して√2倍しておく(回転行列を計算すると導ける．√2倍しとくと整数点になって都合が良い)
- 現れるx座標を座標圧縮しておく(クエリに現れる分も含め)
- 上端と下端をイベントとし，y座標順にイベント処理(詳しくは次ページ)

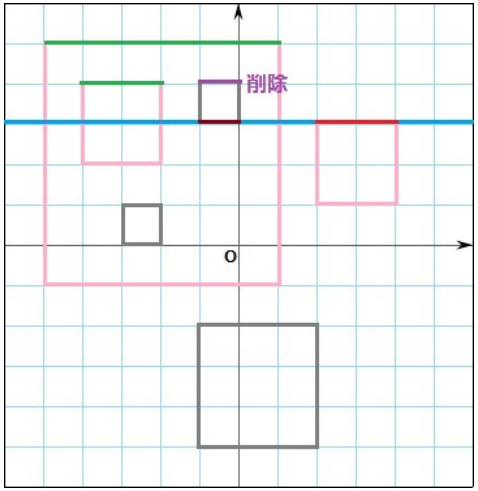


解法 2/3

AtCoder

- 親を求めるための「ある正方形を含む最小の正方形を求める操作」は，
  「イベント処理中の線分集合の中にある正方形の上端の辺を追加するとき，その上端の辺を含む線分の中で最小長のものを求める操作」に対応する
- ある上辺[l,r]を持つ正方形の親の線分は，線分集合に含まれる線分[a,b]で，a < l , r < bを満たすもののうち，最小のbを持つもの．
- これは，左端の値をキーとして，各要素が右端の値の集合を二分木(c++のset等)で持つBITを用いて実現できる．(どの正方形かが復元できるように番号もペアで持っておく)

参考→



解法 3/3

AtCoder

- 今までの説明では省いていたが，クエリの2点の座標もイベント処理の際一緒に処理して，どの正方形に含まれるか計算しておく(点をサイズ0の正方形として処理すると，記述量が減るかもしれない)
- あとは各クエリの木上の最短経路クエリに答えればよい
  →LCAを用いて O(Q log N)

計算量

AtCoder

- クエリの点位置特定も含め構築に，O((N+Q) log^2 (N+Q))
- クエリに対する処理 O(Q log N)

J問題　2つのカップ

問題概要

AtCoder

- 容量がA,Bリットルのカップが１つずつある
- 最初は両カップとも空の状態からはじめて以下のいずれかの操作
  - あるカップを水で満たす
  - あるカップを空にする
  - あるカップの水を、そのカップがからになるか、もう一方が満たされるまで、もう一方のカップに移す
- K回以下の操作でいずれかのカップにちょうどCリットルになるようにしたい
- 実現可能なCは何種類有るか求めよ

考察

AtCoder

- 一般性を失わないため、A ≧ Bと仮定してもよい
- AとBがともにgの倍数の時、両カップに入っている水の量も共にgの倍数。よってA,Bをgで割っても結果には影響しない。
- A,Bを最大公約数で割っても結果は変わらないので、A,Bは互いに素と仮定してもよい

考察

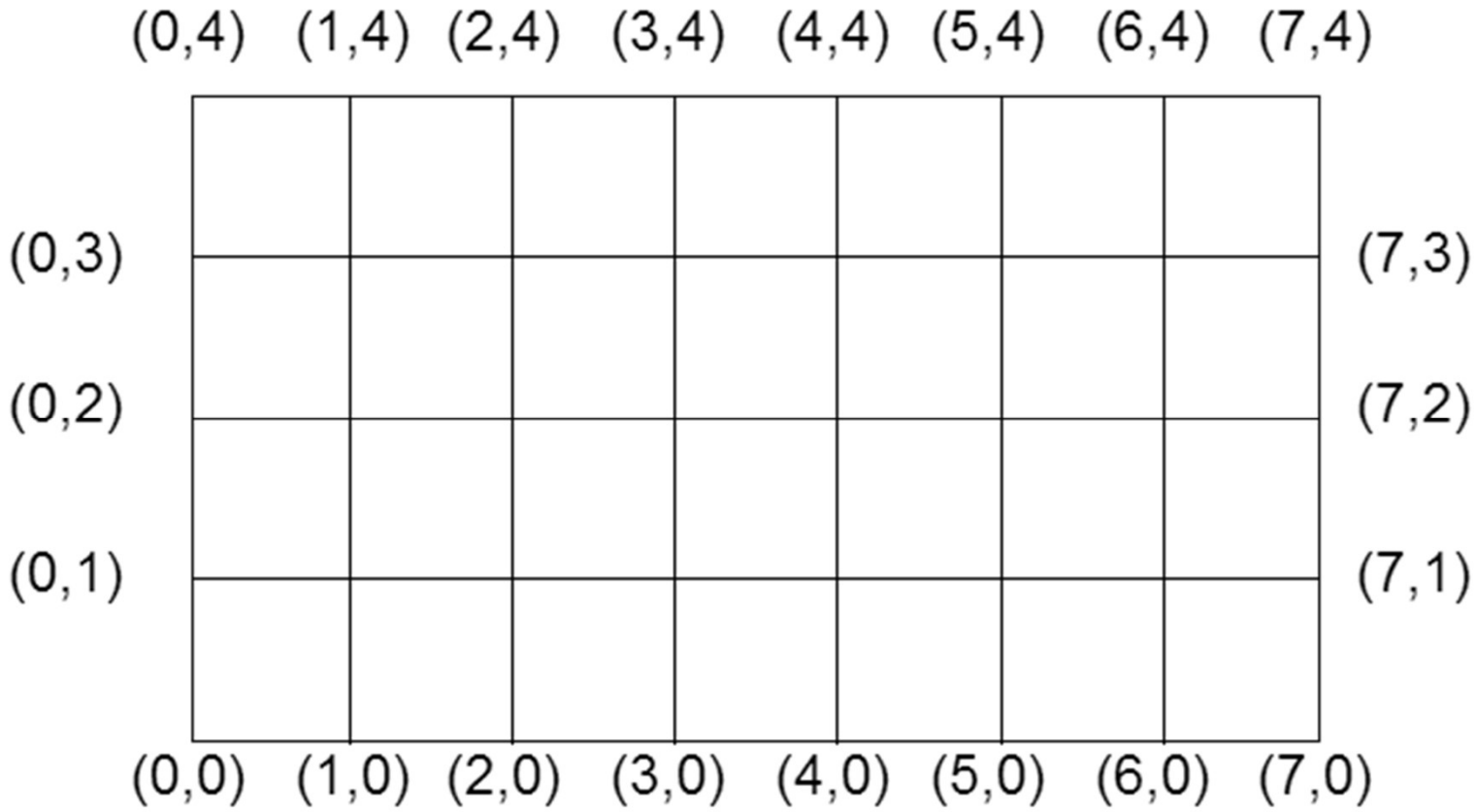
AtCoder

- 状態を（Aの水の量、Bの水の量）で表す
- 常に空のカップもしくは満たされたカップが少なくとも1つ存在する
- よって状態は(0, ?), (?, 0), (A, ?), (?, B)のいずれか
- これらの状態を座標平面上にプロットすると(0,0), (A,B)を対角とする軸平行な長方形の全ての格子点に対応する

プロット

AtCoder

- A=7,B=4のとき



考察

AtCoder

- 以下の操作はCの種類を増やさないので意味が無い
  - (A,*) → (A,B) or (A,0)
  - (*,B) → (A,B) or (0,B)
  - (0,*) → (0,0) or (0,B)
  - (*,0) → (0,0) or (A,0)
  - なぜならば(A,B)は2回, (A,0), (0,B)は1回,(0,0)は0回の操作で実現可能。3回目以降にわざわざこの状態を作る必要がない。

考察

AtCoder

- よってどの状態からも有効な遷移はたかだか2種類しかない
  - (A,*) → Aを捨てる or AからBに移す
  - (*,B) → Bを捨てる or BからAに移す
  - (0,*) → Aを満たす or BからAに移す
  - (*,0) → Bを満たす or AからBに移す
  - (A,0)や(0,B)の場合は遷移は1通りしかない

考察

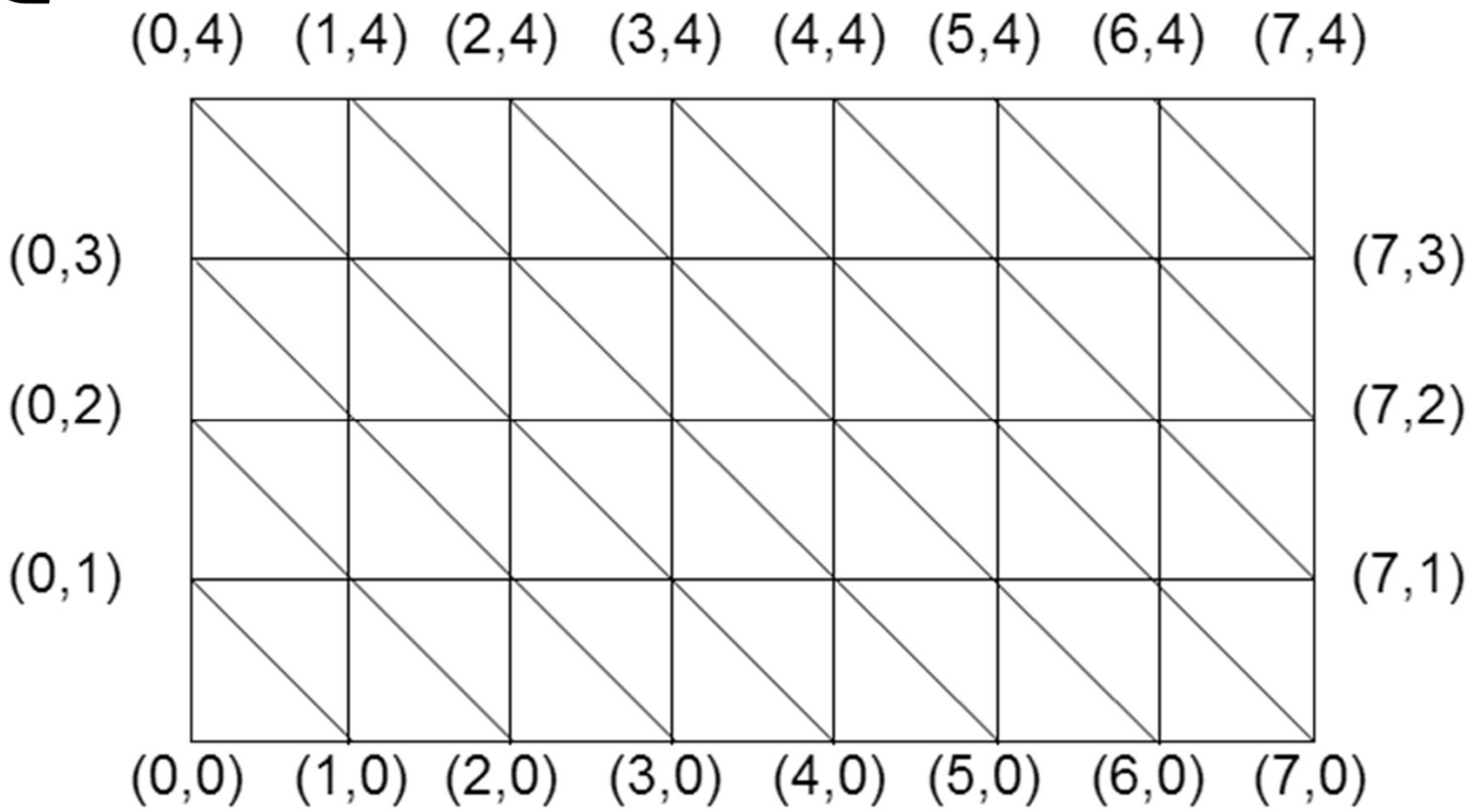
AtCoder

- よってどの状態からも有効な遷移はたかだか2種類しかない
  - (A,*) → Aを捨てる or AからBに移す
  - (*,B) → Bを捨てる or BからAに移す
  - (0,*) → Aを満たす or BからAに移す
  - (*,0) → Bを満たす or AからBに移す
  - (A,0)や(0,B)の場合は遷移は1通りしかない
- これを図示すると

プロット

AtCoder

- 水を移すとき、両カップの水の総和は一定



考察

AtCoder

- 有効な遷移はたかだか2つしか存在しない
- 状態Xから状態Yの遷移が有効ならば、状態Yから状態Xへの遷移も有効である。
- X→Y→Xという遷移は意味が無いので、同じ状態には戻らないという制約を点けることが出来る。するとある遷移を与えるとそれに続く遷移は1種類に定まる
- (A,0)と(0,B)は有効な遷移が1種類しかないのでこれらから始まる遷移が一意にきまる

考察

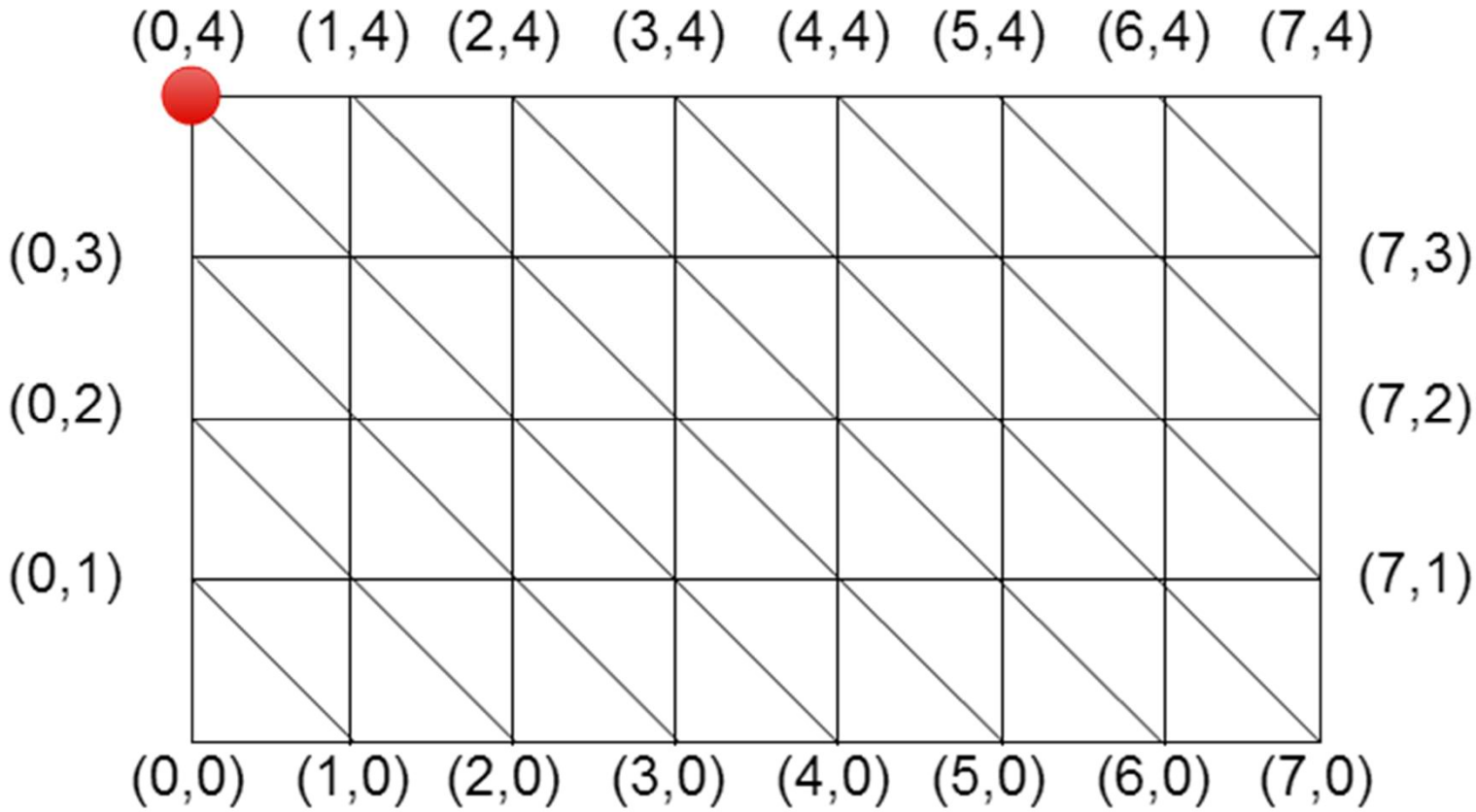
AtCoder

- (0,B)から始まる遷移：
- (0,B)→(B,0)→(B,B)→(2B,B)→(2B,B)→.....
- →((n-1)B,B)→(B,nB%A)→(0,nB%A)
- →(nB%A,0)→(nB%A,B)→((n+1)B%A,0)→...
- →((A-1)B%A,0)→((A-1)B%A,B)→(A,0)
- ※(X%Y = XをYで割った余り)
- (A,0)から始まる遷移はこの逆である
- これらは全ての状態を通過するので、有効な遷移の列はこの2種類しかない

プロット

AtCoder

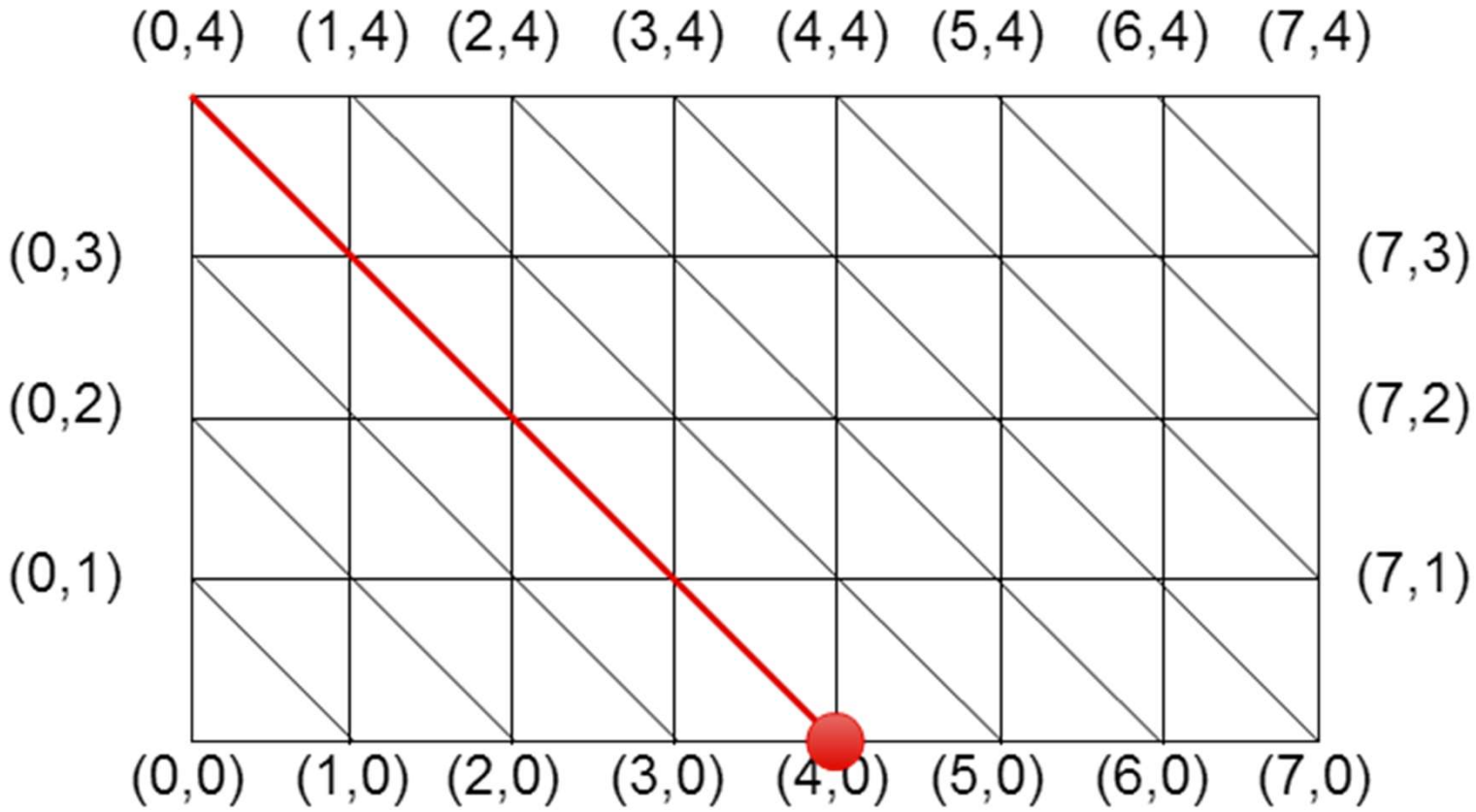
- (0,B)から始まる遷移



プロット

AtCoder

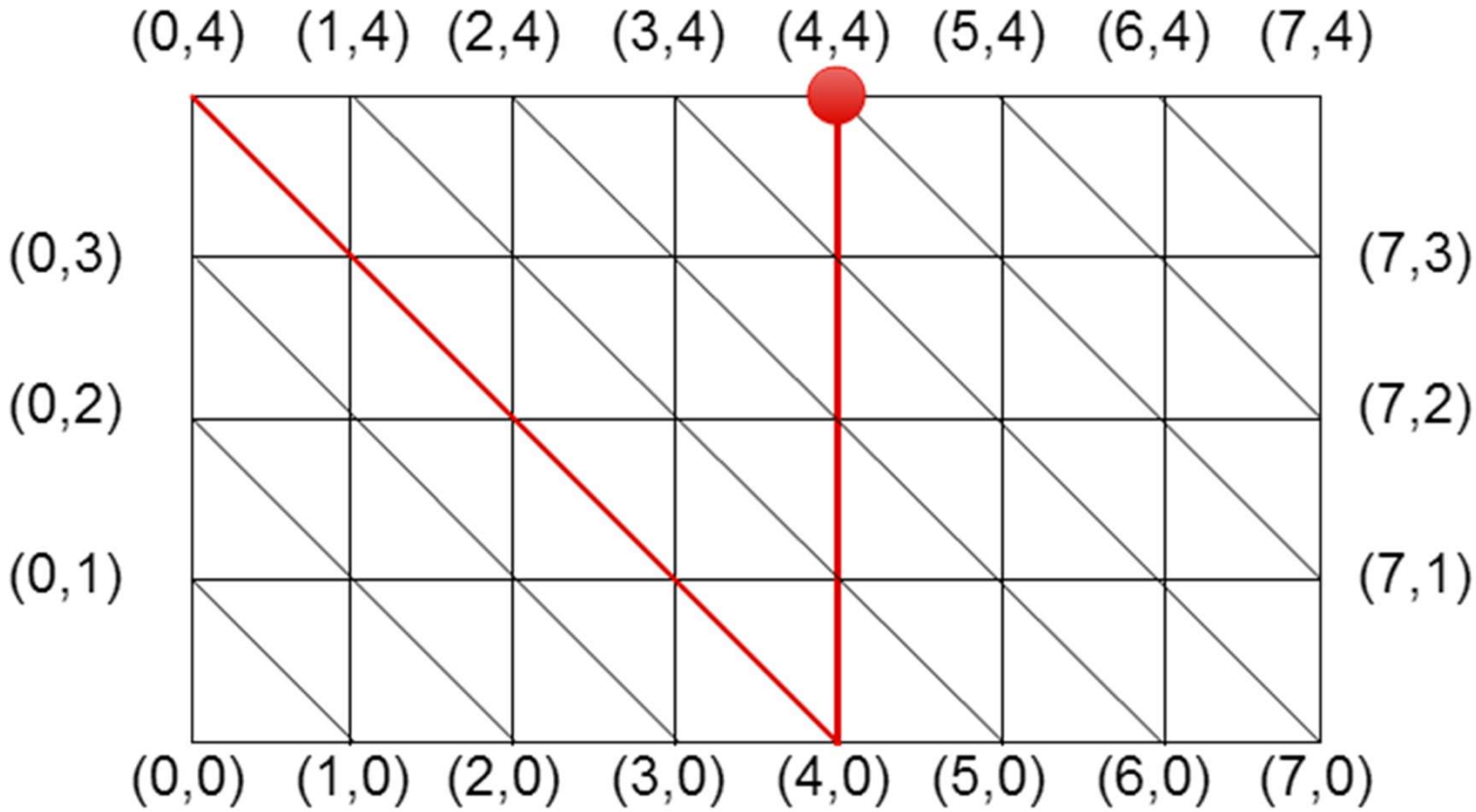
- (4,0)



プロット

AtCoder

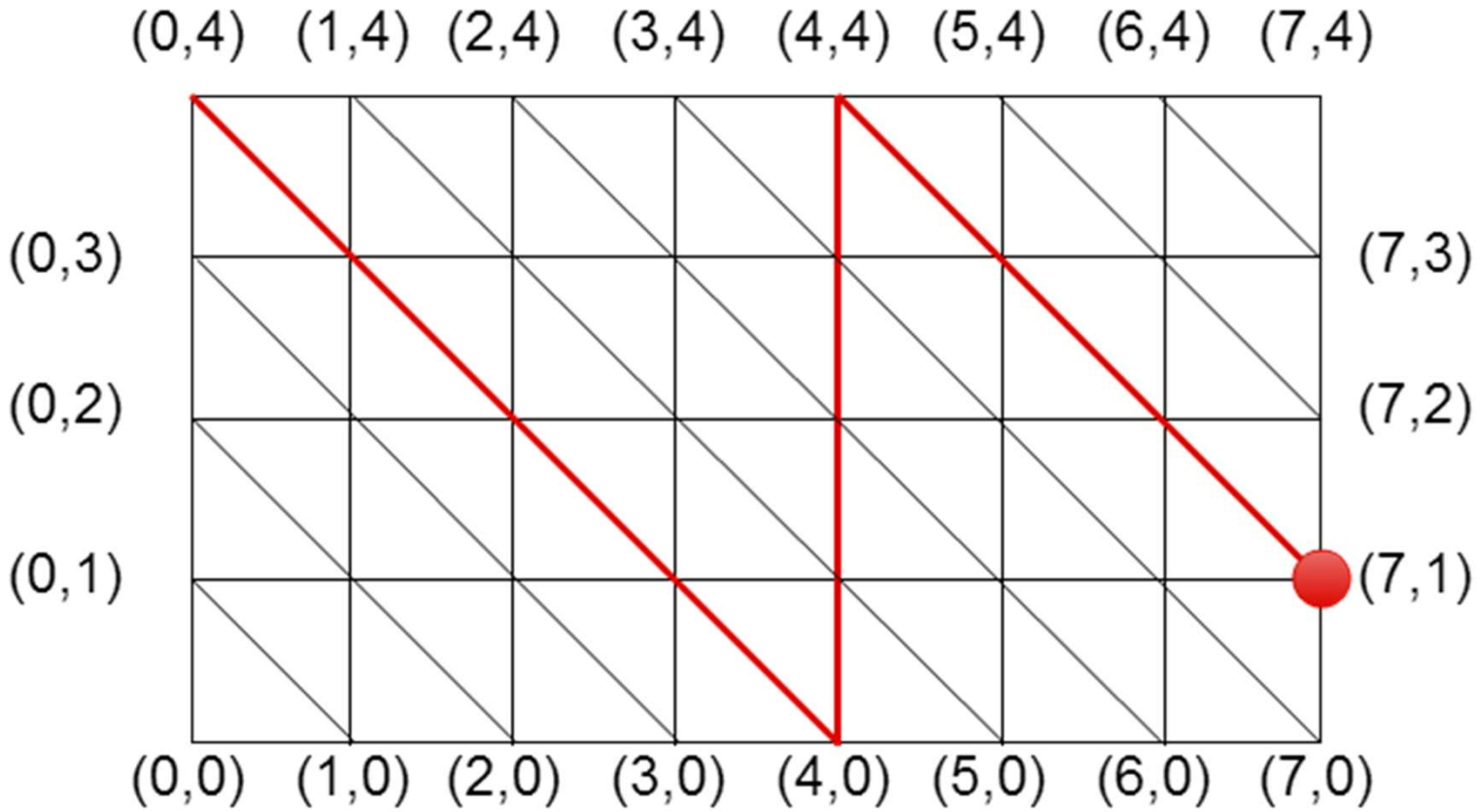
- (4,4)



プロット

AtCoder

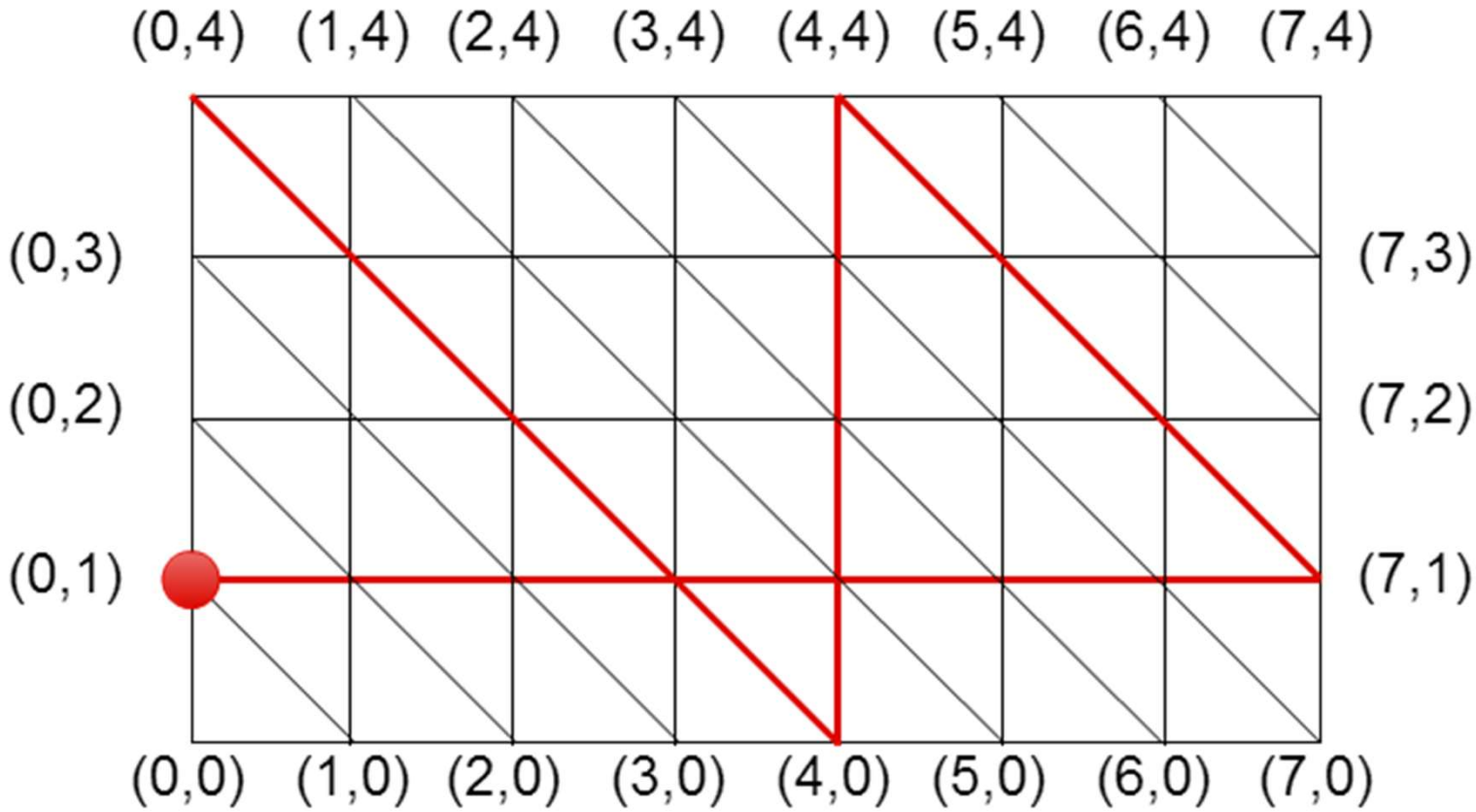
- (7,1)



プロット

AtCoder

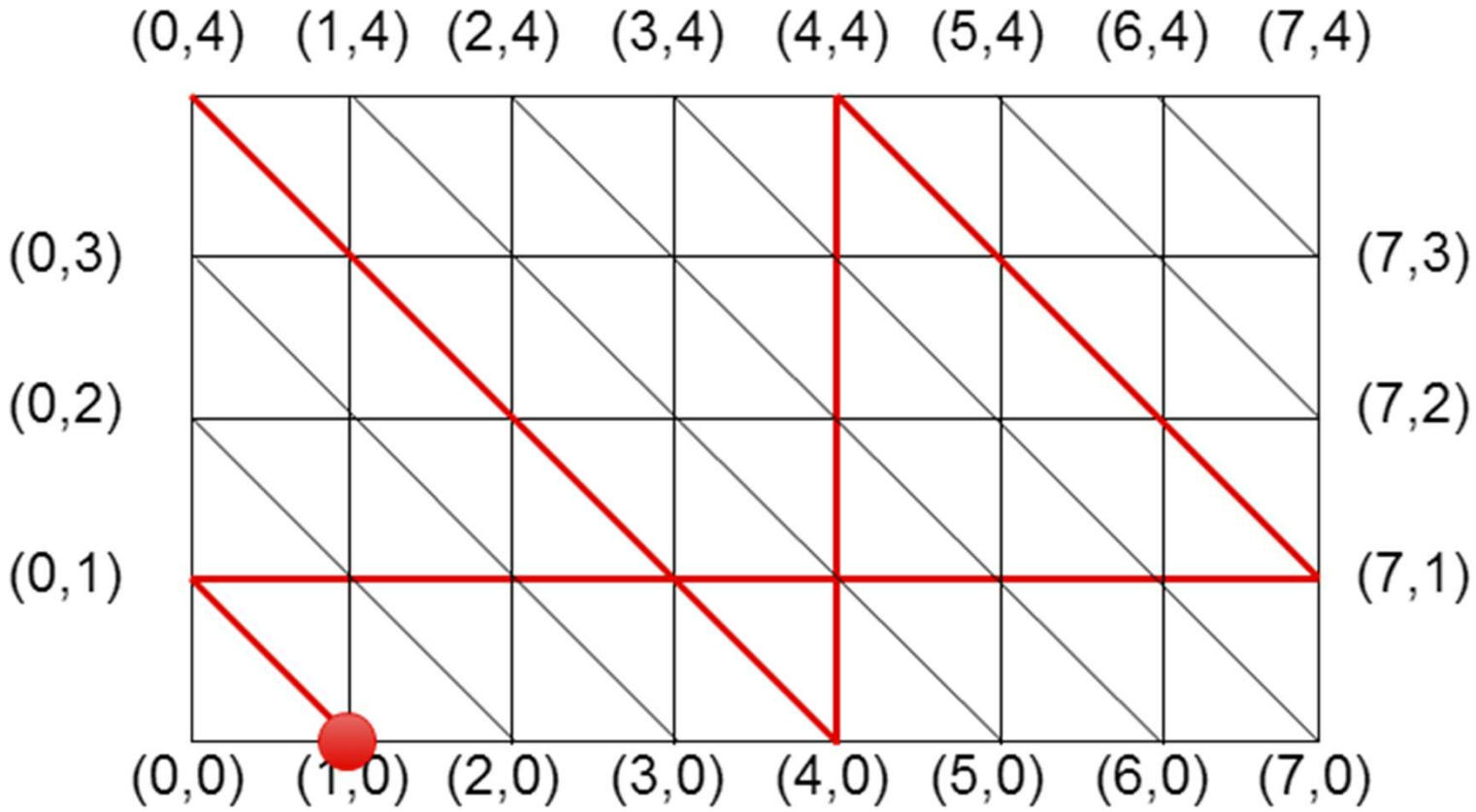
- (0,1)



プロット

AtCoder

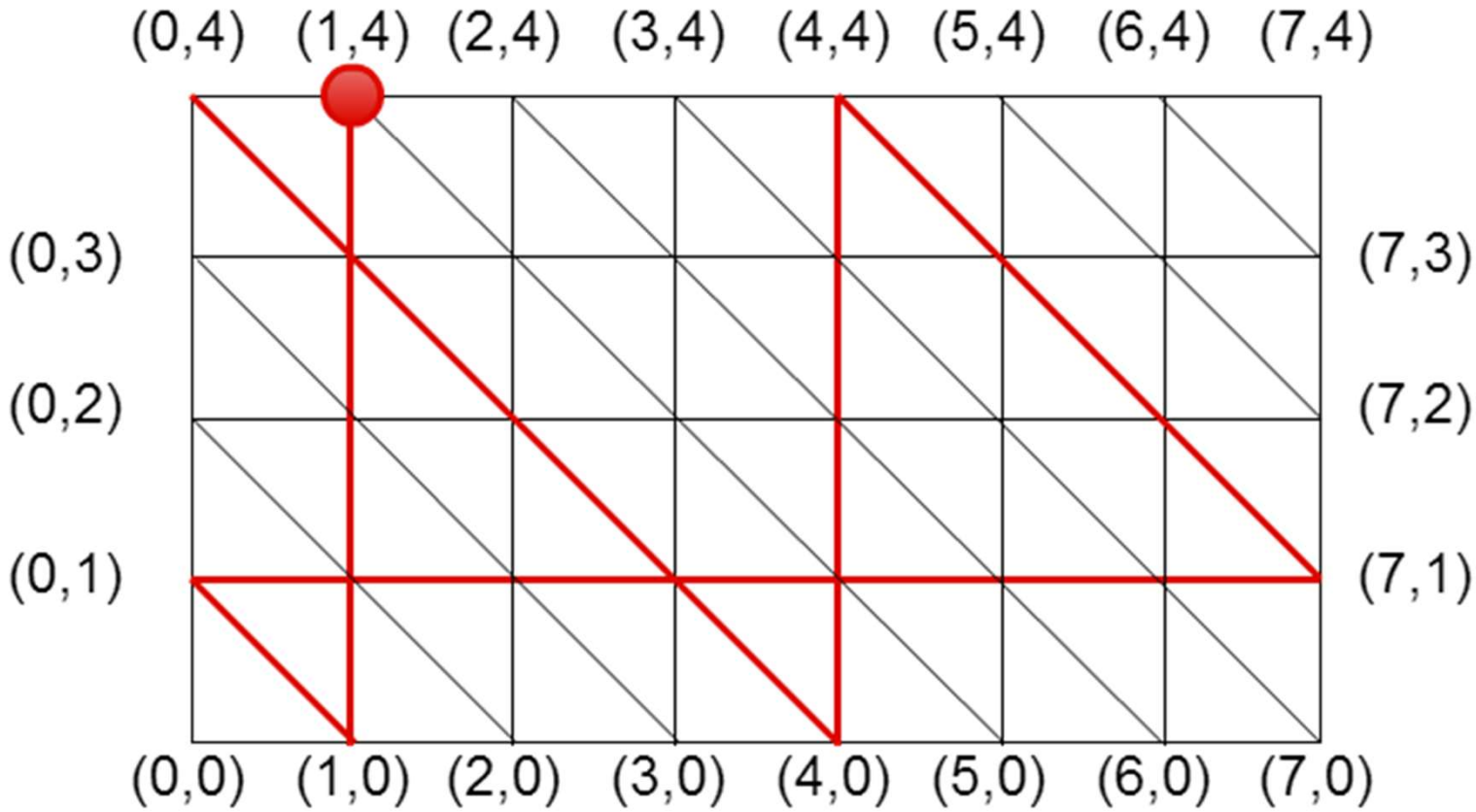
- (1,0)



プロット

AtCoder

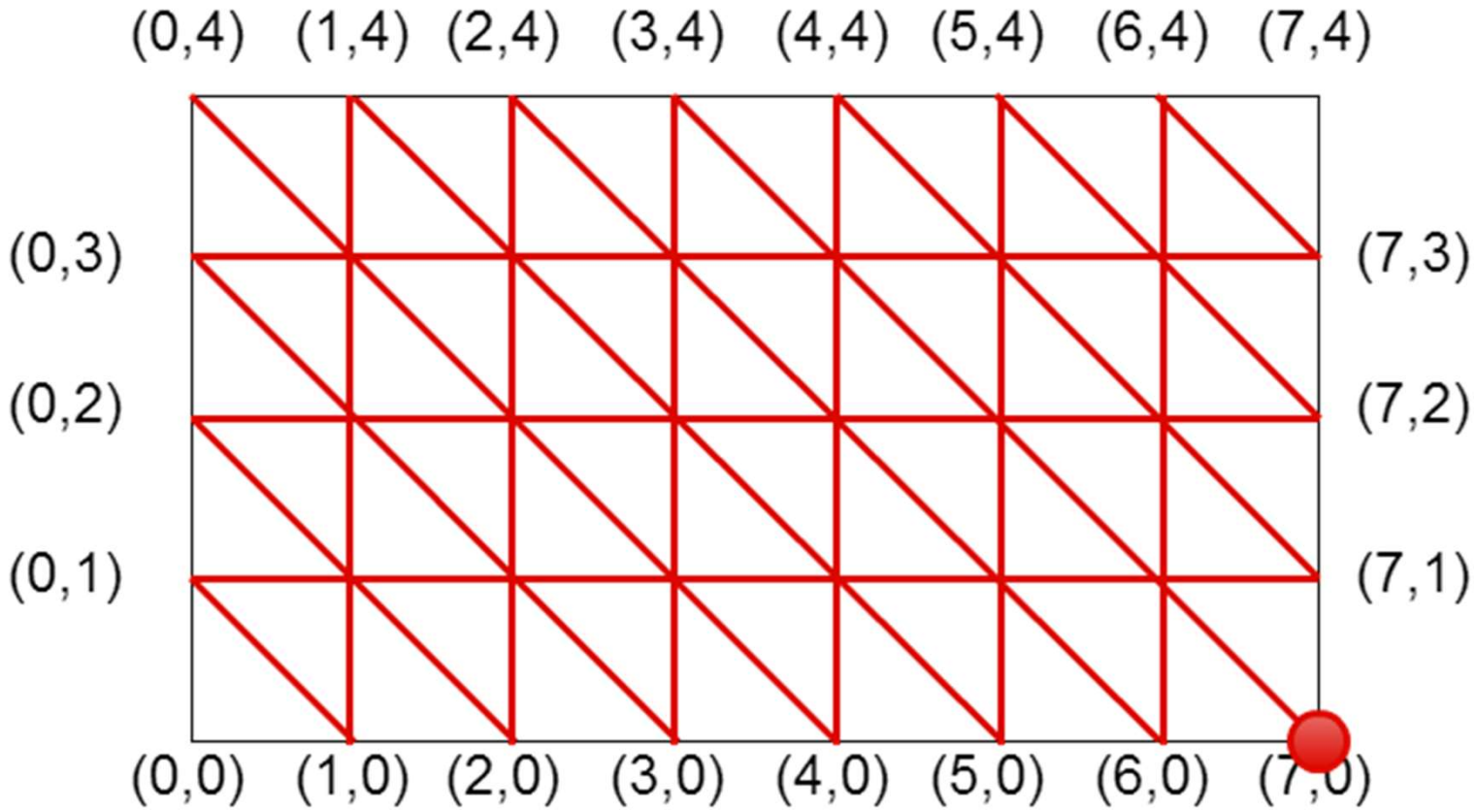
- (1,4)



プロット

AtCoder

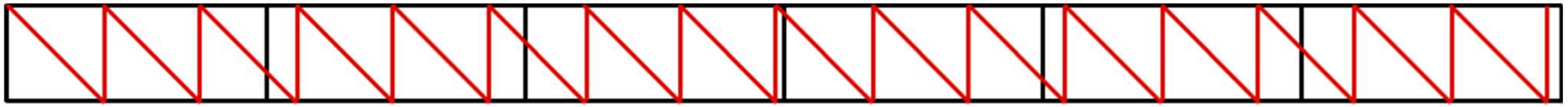
- 以下同様



考察

AtCoder

- この様子を以下のように図示してみる

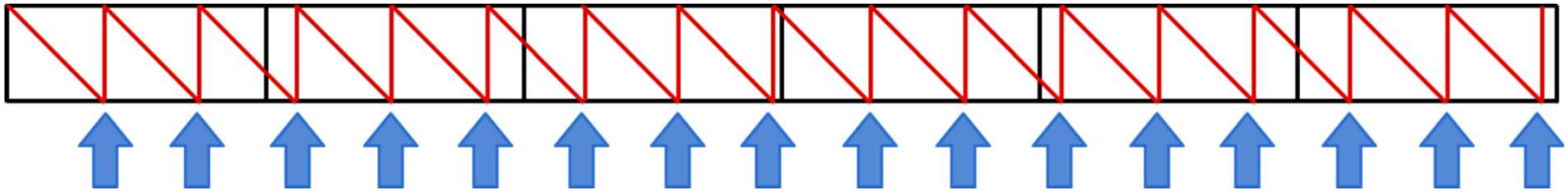


- (A,*) → (0, *)という遷移の代わりに横に長方形をつなげたもの

考察

AtCoder

- この様子を以下のように図示してみる

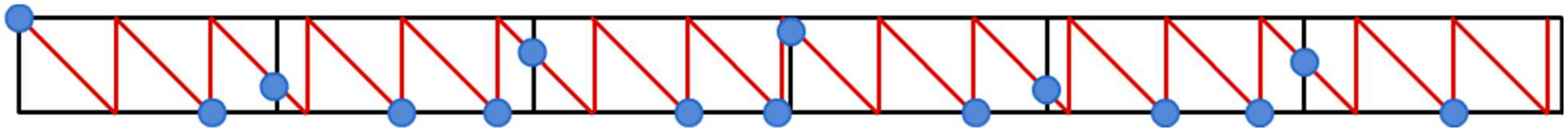


- A,Bは互いに素なので上図の矢印において(nB%A,0)の形で新たなC(=nA%B)が増えるように見える

考察

AtCoder

- この様子を以下のように図示してみる



- A,Bは互いに素なので上図の矢印において(nB%A,0)の形で新たなC(=nB%A)が増えるように見える
- nB/Aの整数部分が変わるところ(黒線)では少し事情が変わり、上図の青丸の場所で新たなCが増える

アルゴリズム

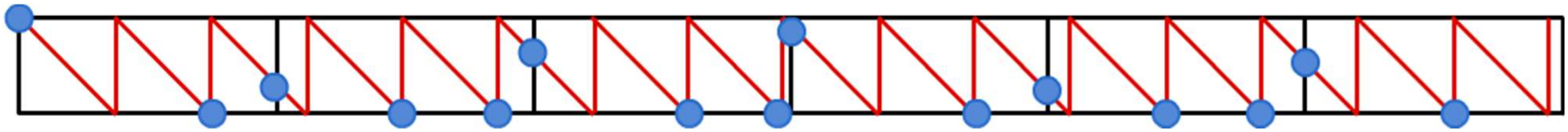
AtCoder

- (0,B)から有効な遷移のみを行うとき、N種類の値を作るのに最低何回の操作が必要か？という問いを考える。

アルゴリズム

AtCoder

- (0,B)から有効な遷移のみを行うとき、N種類の値を作るのに最低何回の操作が必要か？という問いを考える。

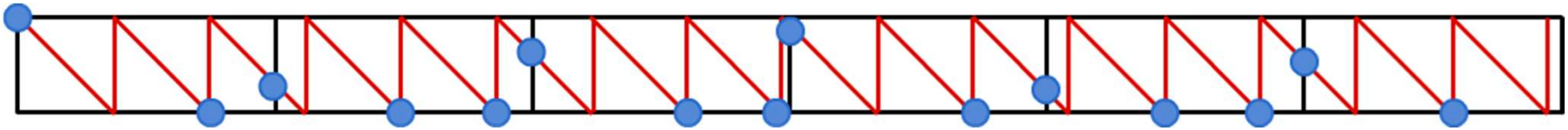


- 上図で左上隅から赤線をたどって左からN番目の青丸まで行く時にたどる線分の数+横切る黒線の数×2が答え
- ただしN番目の青丸が黒線上にある場合は、その黒線を横切らないとする

アルゴリズム

AtCoder

- (0,B)から有効な遷移のみを行うとき、N種類の値を作るのに最低何回の操作が必要か？という問いを考える。



- 式で表すと
  - N=1ならば1
  - [NB/A] = [(N-1)B/A]ならば2N+2[NB/A]
  - [NB/A] ≠ [(N-1)B/A]ならば2N+2[NB/A]-2
  - ただし[x] = xを超えない最大の整数

考察

AtCoder

- (0,B)から有効な遷移のみを行うとき、N種類の値を作るのに最低何回の操作が必要か？という問いを考える。
- これを求めることが出来るようになったので、二分探索で(0,B)から始めてK回で実現できるCの個数が求めることが出来る
  - 適当な逆算でO(1)でも計算可能

アルゴリズム

AtCoder

- (A,0)から始める場合についても同様に考える
- (0,B)から始める場合とほとんど同じ議論が可能。

アルゴリズム

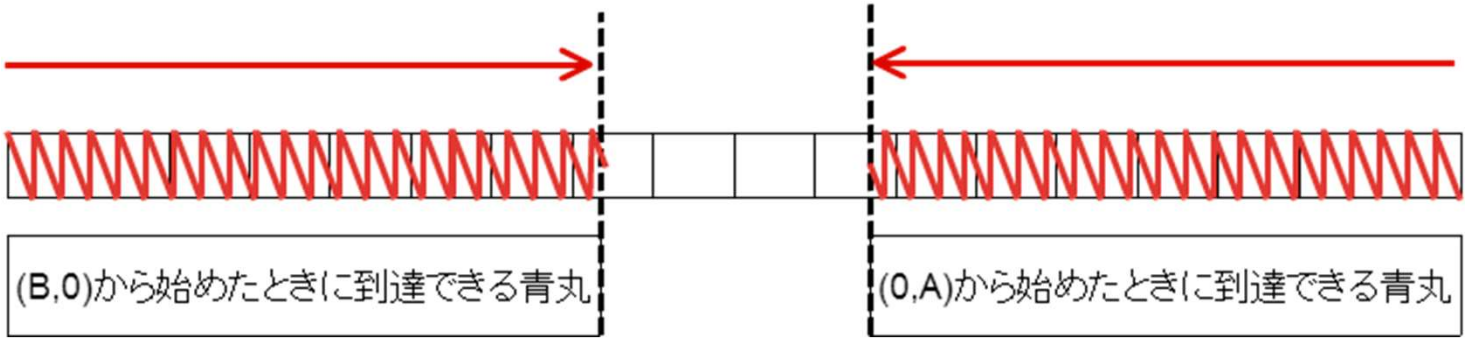
AtCoder

- (A,0)から始める場合についても同様に考える
- (0,B)から始める場合とほとんど同じ議論が可能。
- (A,0)から始める場合の実現可能なCの個数と(0,B)から始める場合の実現可能なCの個数の総和+1が答え
  - +1はC=0の分

アルゴリズム

AtCoder

- (A,0)から始める場合の実現可能なCの個数と(0,B)から始める場合の実現可能なCの個数の総和+1が答え
  - +1はC=0の分
  - (A,0)から始まる遷移はすべての状態を通り(0,B)に到達するのでこの2つの範囲は下図のようになる



アルゴリズム

AtCoder

- (A,0)から始める場合の実現可能なCの個数と(0,B)から始める場合の実現可能なCの個数の総和+1
- この値がA+1を超えるときは遷移の列を全て網羅できている時なので、答えはA+1
- よって答えは「(A,0)から始める場合の実現可能なCの個数と(0,B)から始める場合の実現可能なCの個数の総和+1」と(A+1)のうち小さい方
- 二分探索解法:O(logN)
- 逆算を使う解法:O(1)