
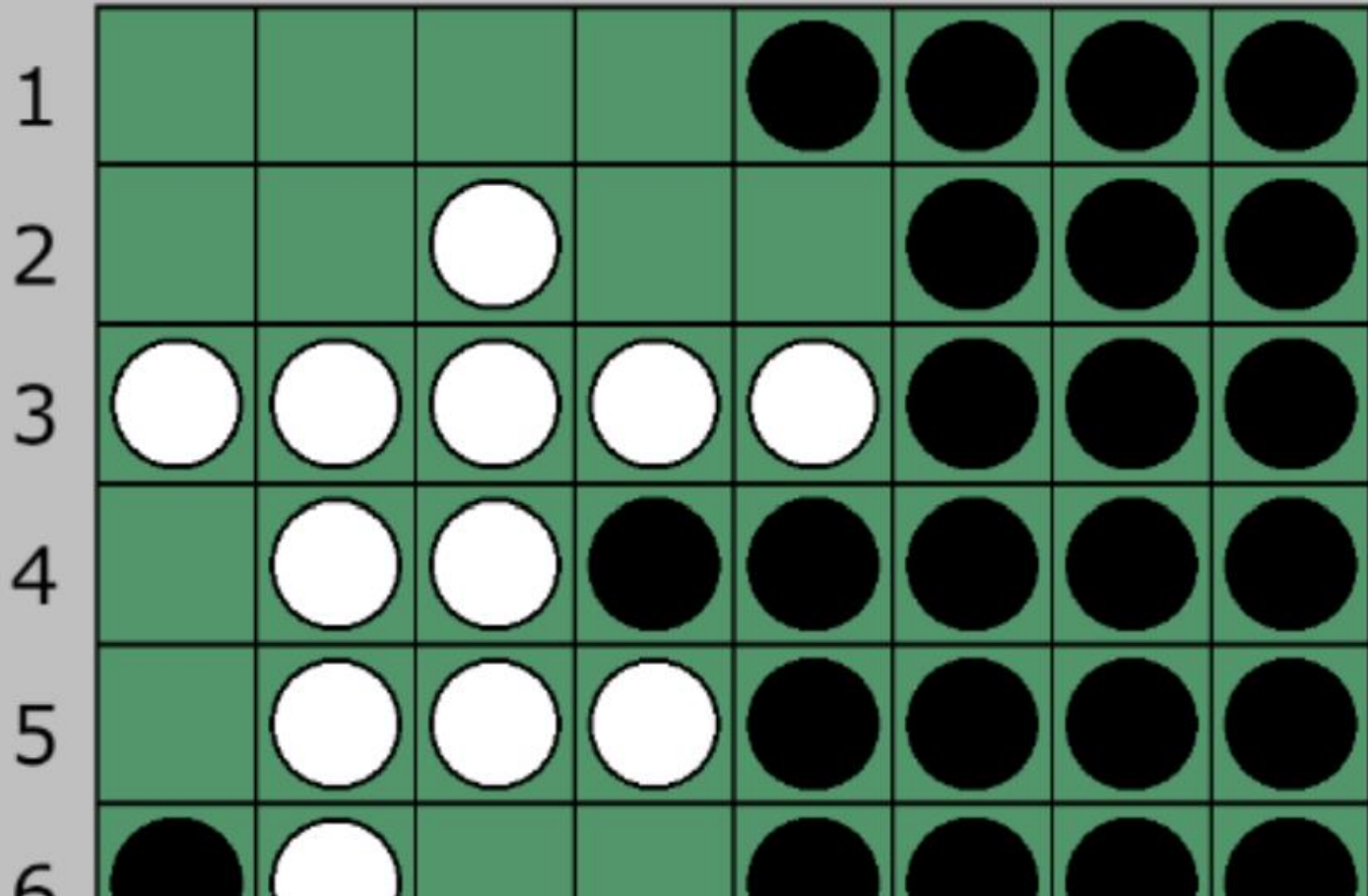


square1001



a8

● 36  
+2



Default



b8

○ 13  
-1

# 問題 1 「碁石ならべ 2」 解説


*Problème 1 "Disposer des Pierres 2"*

解説: 米田 寛峻 (よねだ ひろたか) / square1001

# 1 問題の説明

$N$  個の碁石を左から順に並べます

1



$i$  番目に置く石の色は  $A_i$  です

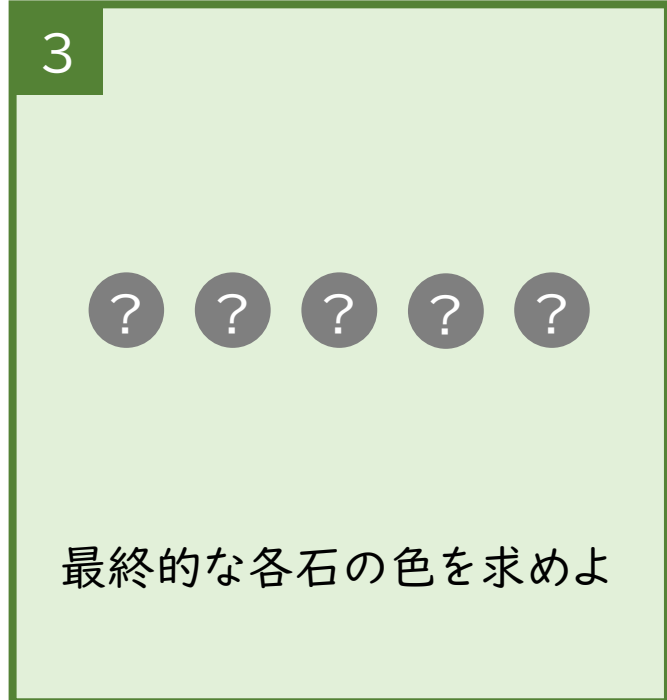
2



New!

同色の 2 つの石に挟まれる  
と同じ色に変化します

3



最終的な各石の色を求めよ

# 1

# 制約

$$N \leq 2000$$
$$A_i \leq 10^9$$

小課題 1 (25 点)

$$N \leq 200\,000$$
$$A_i = 1, 2$$

小課題 2 (60 点)

$$N \leq 200\,000$$
$$A_i \leq 10^9$$

小課題 3 (100 点)

# Chapter II

---

## 小課題 1 の解法

*Sous-tâche 1*

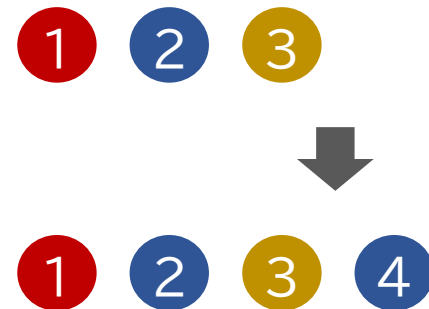
## 2 小課題 1 の解法

小課題 1 の制約は  $N \leq 2000$  です

単純にシミュレーションすることを考えましょう

### 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える



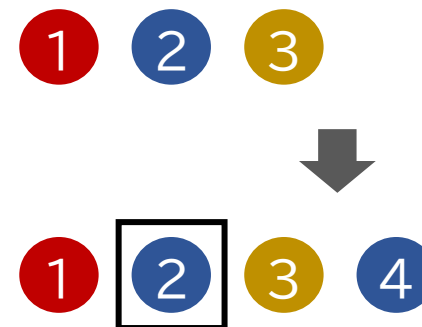
## 2 小課題 1 の解法

小課題 1 の制約は  $N \leq 2000$  です

単純にシミュレーションすることを考えましょう

### 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える



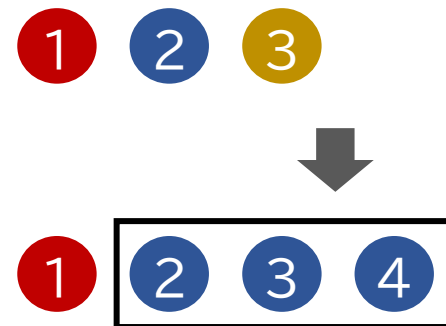
## 2 小課題 1 の解法

小課題 1 の制約は  $N \leq 2000$  です

単純にシミュレーションすることを考えましょう

### 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える



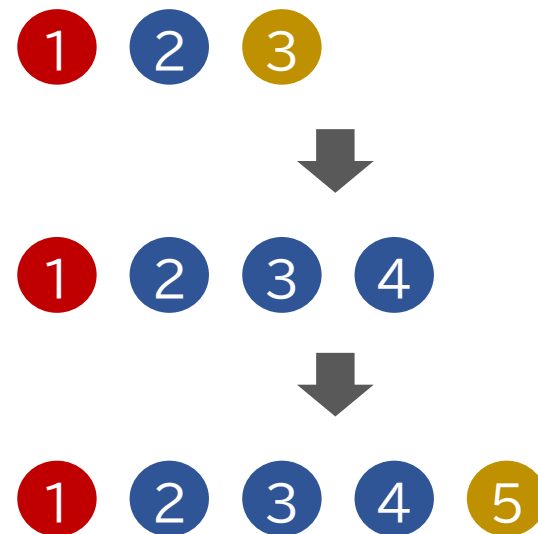
## 2 小課題 1 の解法

小課題 1 の制約は  $N \leq 2000$  です

単純にシミュレーションすることを考えましょう

### 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える





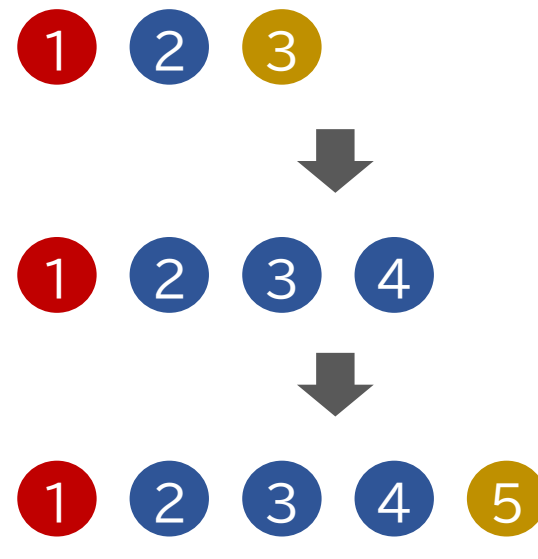
## 2 小課題 1 の解法

小課題 1 の制約は  $N \leq 2000$  です

単純にシミュレーションすることを考えましょう

### 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える



黄色の石は存在しない!

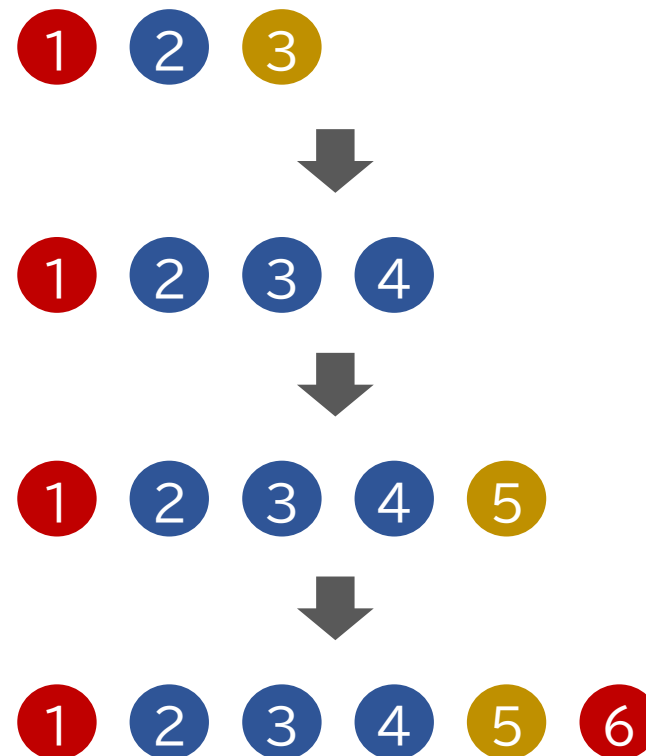
## 2 小課題 1 の解法

小課題 1 の制約は  $N \leq 2000$  です

単純にシミュレーションすることを考えましょう

### 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える



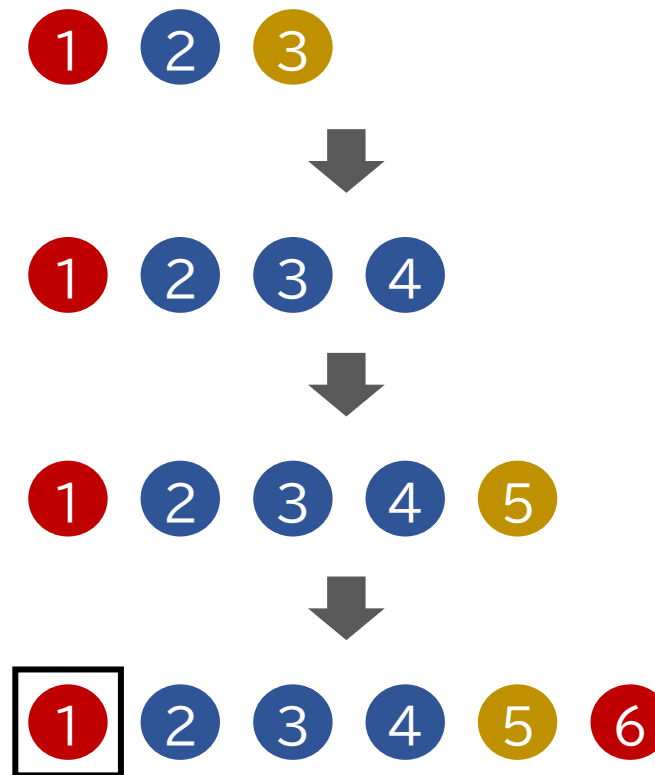
## 2 小課題 1 の解法

小課題 1 の制約は  $N \leq 2000$  です

単純にシミュレーションすることを考えましょう

### 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える



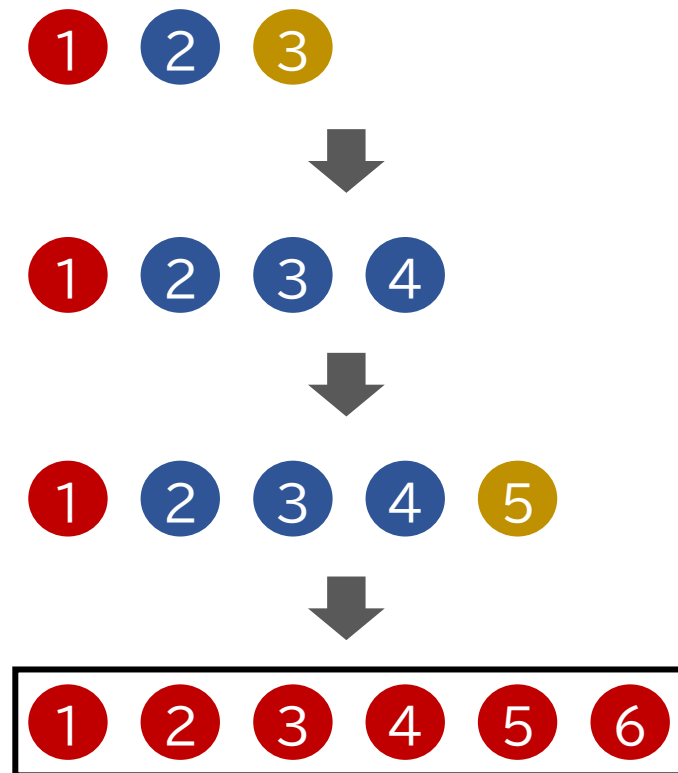
## 2 小課題 1 の解法

小課題 1 の制約は  $N \leq 2000$  です

単純にシミュレーションすることを考えましょう

### 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える



# 2 小課題 1 の解法

解答例を右に示します

手順ごとに分けて考えると実装しやすいです

## 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える

```
int main() {
    // step #1. read input
    int N;
    cin >> N;
    vector<int> A(N);
    for (int i = 0; i < N; i++) {
        cin >> A[i];
    }

    // step #2. simulation
    vector<int> stones;
    for (int i = 0; i < N; i++) {
        int pos = -1;
        for (int j = 0; j < i; j++) {
            if (stones[j] == A[i]) {
                pos = max(pos, j);
            }
        }
        if (pos != -1) {
            for (int j = pos + 1; j < i; j++) {
                stones[j] = A[i];
            }
        }
        stones.push_back(A[i]);
    }

    // step #3. output
    for (int i = 0; i < N; i++) {
        cout << stones[i] << endl;
    }

    return 0;
}
```

# 2 小課題 1 の解法

解答例を右に示します

手順ごとに分けて考えると実装しやすいです

## 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える

入力部分

```
int main() {  
    // step #1. read input  
    int N;  
    cin >> N;  
    vector<int> A(N);  
    for (int i = 0; i < N; i++) {  
        cin >> A[i];  
    }  
  
    // step #2. simulation  
    vector<int> stones;  
    for (int i = 0; i < N; i++) {  
        int pos = -1;  
        for (int j = 0; j < i; j++) {  
            if (stones[j] == A[i]) {  
                pos = max(pos, j);  
            }  
        }  
        if (pos != -1) {  
            for (int j = pos + 1; j < i; j++) {  
                stones[j] = A[i];  
            }  
        }  
        stones.push_back(A[i]);  
    }  
  
    // step #3. output  
    for (int i = 0; i < N; i++) {  
        cout << stones[i] << endl;  
    }  
  
    return 0;  
}
```

# 2 小課題 1 の解法

解答例を右に示します

手順ごとに分けて考えると実装しやすいです

## 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える

```
int main() {  
    // step #1. read input  
    int N;  
    cin >> N;  
    vector<int> A(N);  
    for (int i = 0; i < N; i++) {  
        cin >> A[i];  
    }
```

```
    // step #2. simulation  
    vector<int> stones;
```

```
    for (int i = 0; i < N; i++) {
```

```
        int pos = -1;  
        for (int j = 0; j < i; j++) {  
            if (stones[j] == A[i]) {  
                pos = max(pos, j);  
            }  
        }
```

```
        if (pos != -1) {  
            for (int j = pos + 1; j < i; j++) {  
                stones[j] = A[i];  
            }  
        }
```

```
        stones.push_back(A[i]);  
    }
```

```
    // step #3. output  
    for (int i = 0; i < N; i++) {  
        cout << stones[i] << endl;  
    }
```

```
    return 0;  
}
```

手順 1 の部分

# 2 小課題 1 の解法

解答例を右に示します

手順ごとに分けて考えると実装しやすいです

## 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える

```
int main() {
    // step #1. read input
    int N;
    cin >> N;
    vector<int> A(N);
    for (int i = 0; i < N; i++) {
        cin >> A[i];
    }

    // step #2. simulation
    vector<int> stones;
    for (int i = 0; i < N; i++) {
        int pos = -1;
        for (int j = 0; j < i; j++) {
            if (stones[j] == A[i]) {
                pos = max(pos, j);
            }
        }
        if (pos != -1) {
            for (int j = pos + 1; j < i; j++) {
                stones[j] = A[i];
            }
        }
        stones.push_back(A[i]);
    }

    // step #3. output
    for (int i = 0; i < N; i++) {
        cout << stones[i] << endl;
    }

    return 0;
}
```

手順 2 の部分



# 2 小課題 1 の解法

解答例を右に示します

手順ごとに分けて考えると実装しやすいです

## 各ステップで行う操作

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える

```
int main() {  
    // step #1. read input  
    int N;  
    cin >> N;  
    vector<int> A(N);  
    for (int i = 0; i < N; i++) {  
        cin >> A[i];  
    }  
  
    // step #2. simulation  
    vector<int> stones;  
    for (int i = 0; i < N; i++) {  
        int pos = -1;  
        for (int j = 0; j < i; j++) {  
            if (stones[j] == A[i]) {  
                pos = max(pos, j);  
            }  
        }  
        if (pos != -1) {  
            for (int j = pos + 1; j < i; j++) {  
                stones[j] = A[i];  
            }  
        }  
        stones.push_back(A[i]);  
    }  
  
    // step #3. output  
    for (int i = 0; i < N; i++) {  
        cout << stones[i] << endl;  
    }  
  
    return 0;  
}
```

出力部分

## 2 小課題 1 の解法

解答例を右に示します

手順ごとに分けて考えると実装しやすいです

各ステップで行う操作

# 25 点獲得!

- 1 色  $A_i$  の石の中で最も右のものを求める
- 2 1 で求めた場所より右を色  $A_i$  で塗り替える

```
int main() {  
    // step #1. read input  
    int N;  
    cin >> N;  
    vector<int> A(N);  
    for (int i = 0; i < N; i++) {  
        cin >> A[i];  
    }  
  
    // step #2. simulation  
    int pos = 0;  
    for (int i = 0; i < N; i++) {  
        int j = pos;  
        while (j < N; j++) {  
            if (A[j] == A[i]) {  
                stones[j] = A[i];  
            }  
        }  
        stones.push_back(A[i]);  
    }  
  
    // step #3. output  
    for (int i = 0; i < N; i++) {  
        cout << stones[i] << endl;  
    }  
  
    return 0;  
}
```

出力部分

# Chapter III

---

## 小課題 2 の解法

*Sous-tâche 2*

# 3 小課題 2 の解法

小課題 2 の制約は「 $A_i = 1, 2$ 」です

つまり、囲碁やオセロのように「黒と白しかない」ということです



# 3 小課題 2 の解法

どのような性質があるのでしょうか？

これを探るため、小さいケースで実験してみましよう



# 3 小課題 2 の解法

どのような性質があるのでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合

①

# 3 小課題 2 の解法

どのような性質があるのでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合

①

① ②

# 3 小課題 2 の解法

どのような性質があるのでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合

①

① ②

① ② ③



# 3 小課題 2 の解法

どのような性質があるでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合

①

① ②

① ② ③

① ② ③ ④

# 3 小課題 2 の解法

どのような性質があるのでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合

①

① ②

① ② ③

① ② ③ ④

① ② ③ ④ ⑤

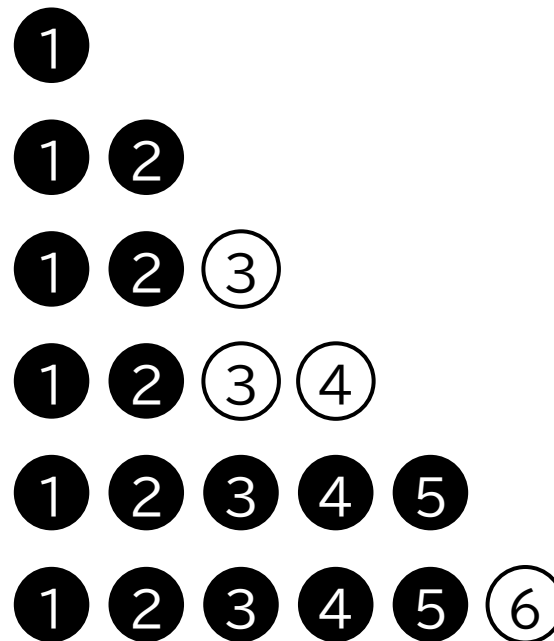
# 3 小課題 2 の解法

どのような性質があるでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合



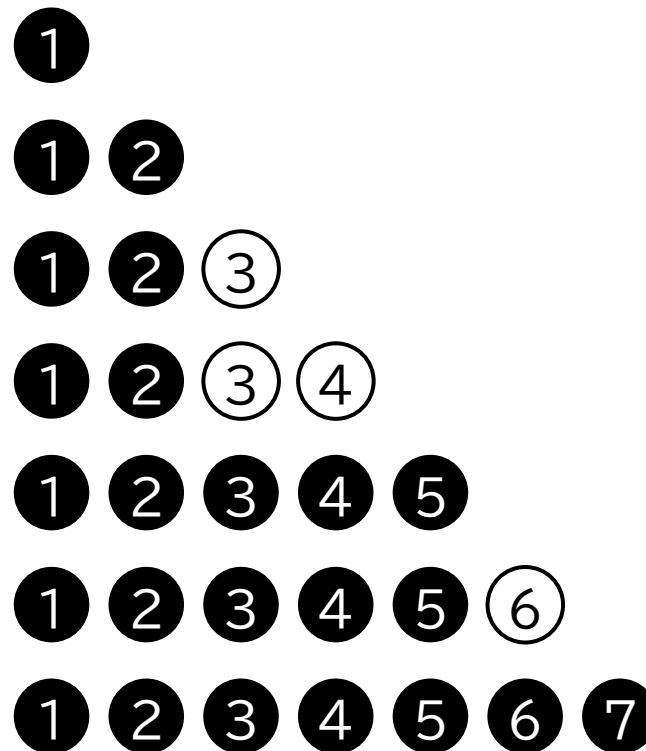
# 3 小課題 2 の解法

どのような性質があるでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合



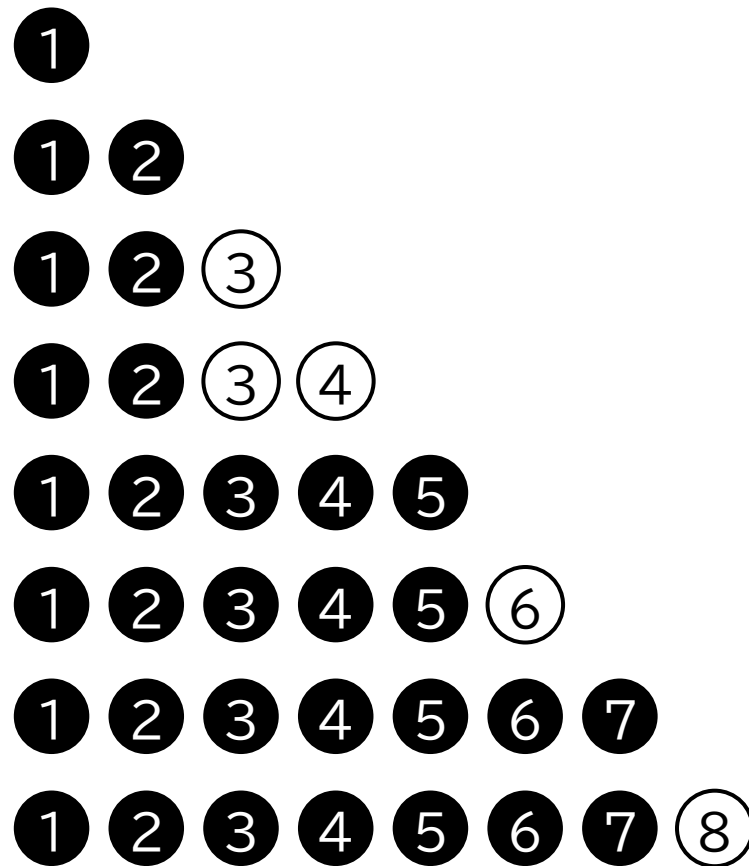
# 3 小課題 2 の解法

どのような性質があるでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合



# 3 小課題 2 の解法

どのような性質があるのでしょうか？

これを探るため、小さいケースで実験してみましよう



例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合

①

① ②

① ② ③

① ② ③ ④

① ② ③ ④ ⑤

① ② ③ ④ ⑤ ⑥

① ② ③ ④ ⑤ ⑥ ⑦

① ② ③ ④ ⑤ ⑥ ⑦ ⑧

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

# 3 小課題 2 の解法

どのような性質があるのでしょうか？

これを探るため、小さいケースで実験してみましよう

## 結論

どんな場合でも「黒, 黒, ..., 黒 の後に  
白, 白, ..., 白」という状態になる!

例:  $A = (1, 1, 2, 2, 1, 2, 1, 2, 2)$  の場合

①

① ②

① ② ③

① ② ③ ④

① ② ③ ④ ⑤

① ② ③ ④ ⑤ ⑥

① ② ③ ④ ⑤ ⑥ ⑦

① ② ③ ④ ⑤ ⑥ ⑦ ⑧

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨





# 3 小課題 2 の解法

## 解法

一番左が黒であれば、一番右の黒まで全部黒になる（それより右は全部白）

入力: ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨



出力: ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

# 3 小課題 2 の解法

## 解法

一番左が黒であれば、一番右の黒まで全部黒になる（それより右は全部白）

入力: ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨



出力: ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

実際にプログラムを書くときは…

$A_1 = A_i$  となる最大の  $i$  まで、  
全部色  $A_1$  で塗られる

ということになります

当然、計算量は  $O(N)$

# 3 小課題 2 の解法

解法

一番左が黒であれば、一番右の黒まで全部黒になる（それより右は全部白）

実際にプログラムを書くときは…

# 60点獲得!

1からNまでの最大のkまで、k番目まで塗られるというようになります

入力: ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨



出力: ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

当然、計算量は  $O(N)$

# Chapter IV

---

## 小課題 3 (満点) の解法

*Sous-tâche 3*

# 4

# 満点解法

実は、一般の場合でも、小課題 2 と同じような性質がある

## 性質

最も右の色  $A_1$  の石までが、全部色  $A_1$  に塗り替わる

入力: ① ② ③ ④ ⑤ ⑥

①

① ②

① ② ③

① ② ③ ④

① ② ③ ④ ⑤

① ② ③ ④ ⑤ ⑥

# 4

# 満点解法

性質

最も右の色  $A_1$  の石までが、全部色  $A_1$  に塗り替わる

この性質を使って、答えを求めよう!



# 4

# 満点解法

性質

最も右の色  $A_1$  の石までが、全部色  $A_1$  に塗り替わる

この性質を使って、答えを求めよう!



最も右の赤色はここなので...

# 4

# 満点解法

性質

最も右の色  $A_1$  の石までが、全部色  $A_1$  に塗り替わる

この性質を使って、答えを求めよう!



これより左が全部赤とわかる!



# 4

# 満点解法

性質

最も右の色  $A_1$  の石までが、全部色  $A_1$  に塗り替わる

この性質を使って、答えを求めよう!



これより左が全部赤とわかる!

Question. 残りの部分はどうやって求める?

# 4

# 満点解法

性質

最も右の色  $A_1$  の石までが、全部色  $A_1$  に塗り替わる

一旦 5 番目までの石の存在を忘れることにしよう（後の結果には関係ない）



# 4

# 満点解法

性質

最も右の色  $A_1$  の石までが、全部色  $A_1$  に塗り替わる

一旦 5 番目までの石の存在を忘れることにしよう（後の結果には関係ない）



最も右の黄色はここなので…

# 4

# 満点解法

性質

最も右の色  $A_1$  の石までが、全部色  $A_1$  に塗り替わる

一旦 5 番目までの石の存在を忘れることにしよう（後の結果には関係ない）

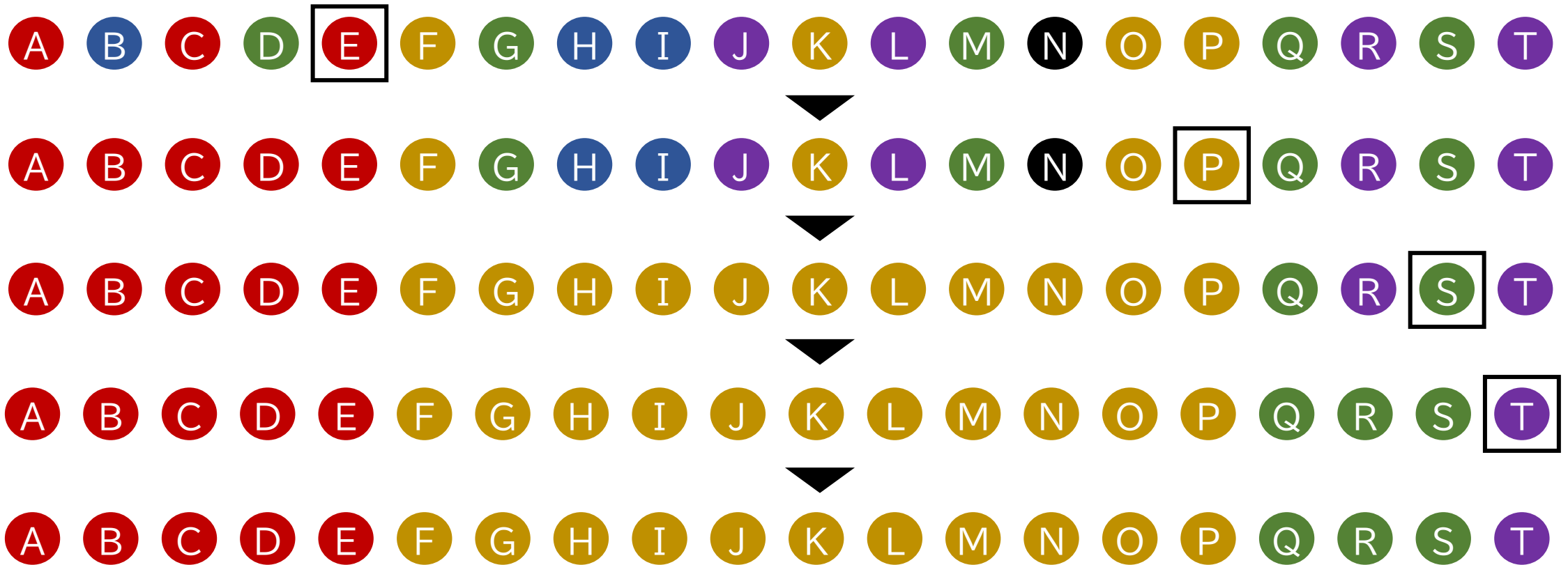


これより左が全部黄色とわかる!

# 4

# 満点解法

残りも同様にやっていると、次のようになります



# 4 満点解法

残りも同様にやっていくと、次のようになります

しかし、最悪ケースでは  $N$  ステップかかる

計算量  $O(N^2)$  になってしまう！

A B C D E F G H I J K L M N O P Q R S T

# 4

# 満点解法

Question. 各ステップで「どこまで塗り替わるか」は、どうやって高速計算できるか？

# 4

# 満点解法

Question. 各ステップで「どこまで塗り替わるか」は、どうやって高速計算できるか？

アイデア

各色に対して、最も右のものを前計算する



(この例だと、赤: E、青: I、黄: P、緑: S、紫: T、黒: N)

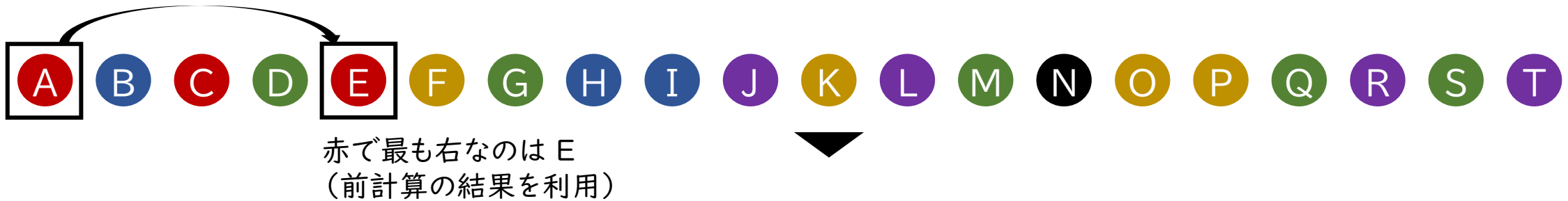
C++ では map を使って実装すると便利



# 4

# 満点解法

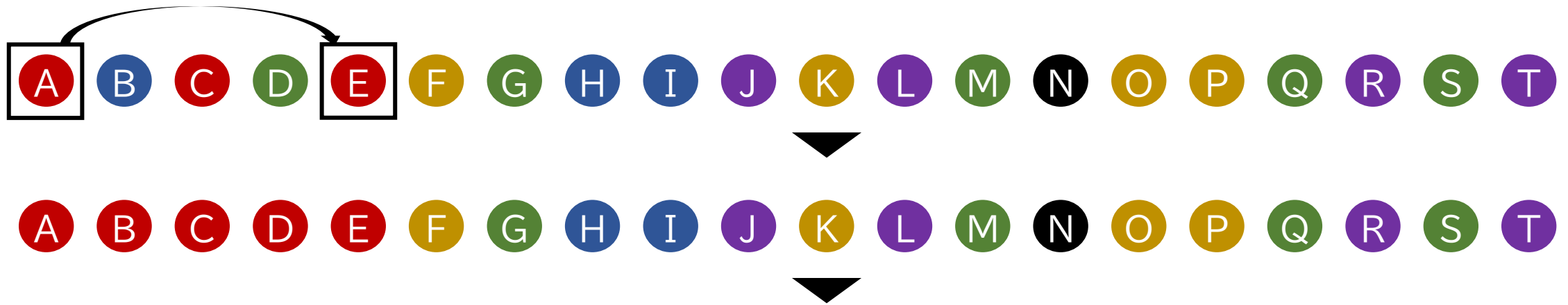
すると…? (最も右は 赤: E、青: I、黄: P、緑: S、紫: T、黒: N)



# 4

# 満点解法

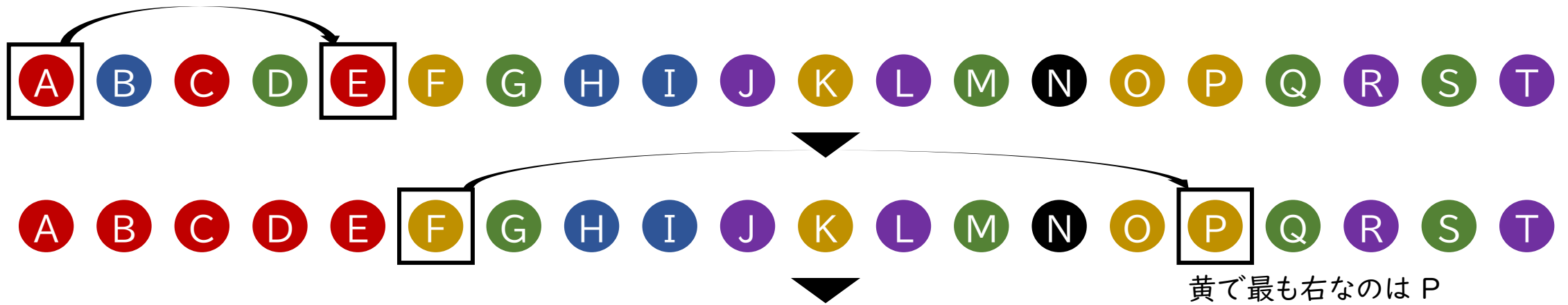
すると…? (最も右は 赤: E、青: I、黄: P、緑: S、紫: T、黒: N)



# 4

# 満点解法

すると…? (最も右は 赤: E、青: I、黄: P、緑: S、紫: T、黒: N)

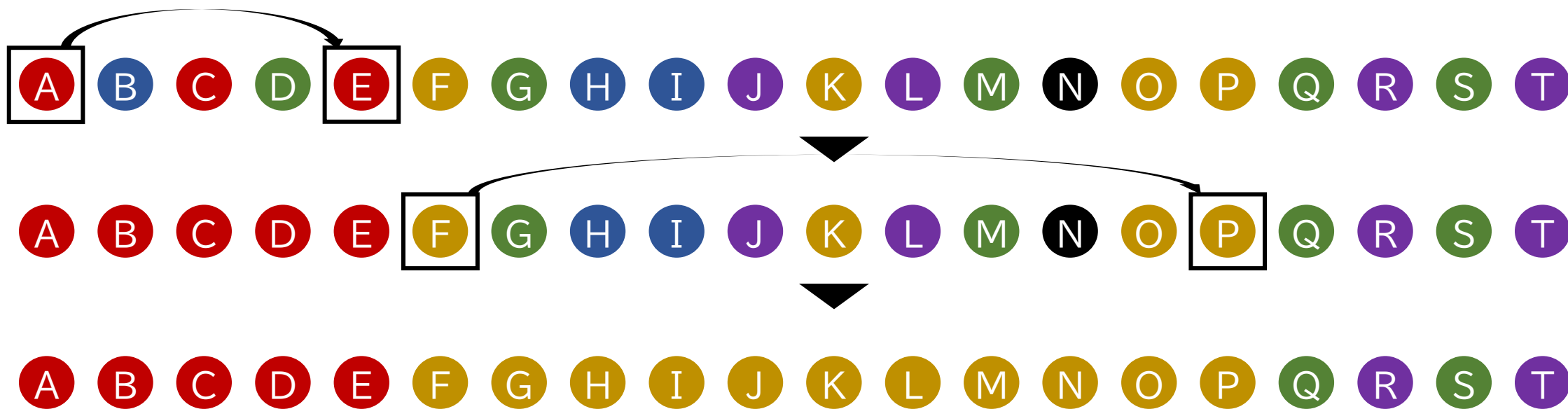


黄で最も右なのは P  
(前計算の結果を利用)

## 4

## 満点解法

すると…? (最も右は 赤: E、青: I、黄: P、緑: S、紫: T、黒: N)



# 4 満点解法

すると…? (最も右は 赤: E、青: I、黄: P、緑: S、紫: T、黒: N)

前計算の結果を利用すれば

1ステップをすぐに終わられる!

# 4 満点解法

## 計算量について

- map を使うとアクセスに  $O(\log n)$  かかります
- そのため、全体計算量  $O(N \log N)$  で解けます

## 別解について

- ハッシュマップ (unordered\_map) を使うと計算量  $O(N)$  になります
- 圧縮を使って同じ計算量で解く解法もあります

## 4 満点解法

計算量について

- map を使うとアクセスに  $O(\log n)$  かかります

- そのため、全体計算量  $O(N \log V)$  で解けます

別解について

- ハッシュマップ (unordered\_map) を使うと計算量  $O(N)$  になります
- 圧縮を使って同じ計算量で解く解法もあります

# 1000点獲得!

# Chapter V

---

## 得点分布

*Répartition des scores*



# 5

# 得点分布

## 得点分布

- 100 点: 135 人
- 60 点 (小課題 1+2): 25 人
- 35 点 (小課題 2): 3 人
- 25 点 (小課題 2): 6 人

# 5 宣伝

CodinGame というサイトでオセロ AI を作って遊べます

- 打倒 square1001 !

The screenshot displays the CodinGame Othello interface. The main game board is an 8x8 grid with columns labeled a-h and rows 1-8. The board shows a game in progress with white and black pieces. The player 'square1001' is in the 1st position with a score of d7 (4 to -2), and the AI 'Default' is in the 2nd position with a score of c3 (6 to +3). The interface includes a sidebar with navigation options like BACK, FORUM, LAST BATTLES, HISTORY, LEADERBOARD, and SETTINGS. The bottom section shows a console output, game information (Standard Output Stream: EXPERT c3), and player information for 'square1001' and 'Default'. The code editor on the right shows C++ code for a random number generator and a bitline base class.

```
1 #include <cmath>
2 #include <ctime>
3 #include <chrono>
4 #include <random>
5 #include <string>
6 #include <vector>
7 #include <cassert>
8 #include <iostream>
9 #include <algorithm>
10 #include <functional>
11 #include <immintrin.h>
12 #pragma GCC target("avx2")
13 #pragma GCC optimize("ofast")
14
15
16 namespace myrandom {
17     std::mt19937_64 mt((uint64_t)time(NULL));
18     uint64_t rand_seed = mt();
19     uint64_t xorshift4() {
20         rand_seed ^= rand_seed << 13;
21         rand_seed ^= rand_seed >> 7;
22         rand_seed ^= rand_seed << 17;
23         return rand_seed;
24     }
25     int64_t rand_int(int64_t l, int64_t r) {
26         return l + int64_t(xorshift4() % (r - l));
27     }
28 }
29
30
31 // ----- I. BOARD CLASS & FUNCTIONS ----- //
32
33 inline int popcount(uint64_t bit) {
34     return __builtin_popcountll(bit);
35 }
36
37 class bitline_base {
38     ...
39 }
```