問題5：エゴイ展　解説

情報オリンピック女性部門 2022 本選

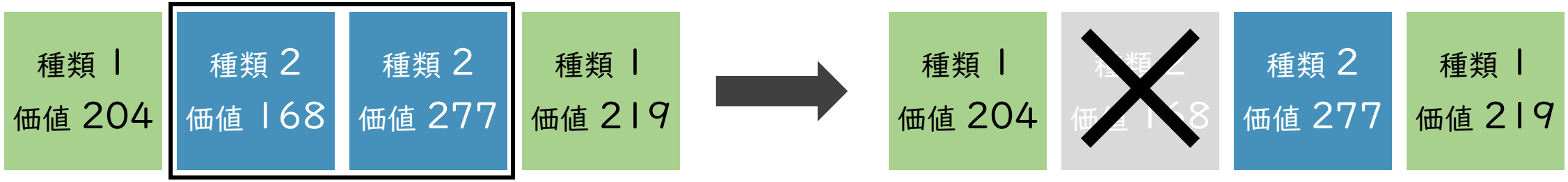
米田 優峻（@e869120）

0 問題概要

- 横一列に N 個の絵が並べられています。
- i 個目の絵の種類は A_i、価値は V_i です。
- 隣り合う絵が同じ種類にならないように絵を撤去するとき、残った絵の価値合計の最大値はいくつですか？

0 入力例

$N = 4, (A_i, V_i) = (1, 204), (2, 168), (2, 277), (1, 219)$ の場合



隣り合う絵が同じ色なので
撤去しなければならない

このようにすれば条件を満たす
価値合計は 204+277+219=700

小課題 1

$$M = 1, V_i \geq 1 \ (1 \leq i \leq N)$$

1 小課題 1 の解法

- 制約が $M = 1$ なので、全部の絵の色が同じです。
- 絵を 1 つしか残すことができません。
- 価値が最大となる絵を残すのが最適なので、求めるべき答えは $\max(V_1, V_2, \ldots, V_N)$ となります。

1 小課題 1 の実装

- このように、簡単に実装することができます。
- 情報オリンピックでは、取れる部分点は全部取りましょう。

```
# 入力
N = int(input())
M = int(input())
A = [None] * N
V = [None] * N
for i in range(N):
    A[i], V[i] = map(int, input().split());
# 出力
print(max(V))
```

4点

小課題 2

$V_i \geq 1 \ (1 \leq i \leq N)$

2 小課題 2 の解法

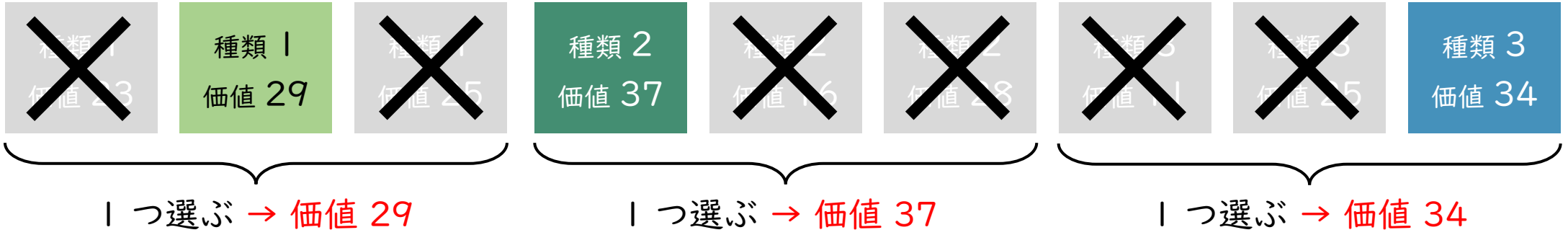
- まずは以下のケースでの答えを考えてみよう
- 1～3 番目の中からは、最大 1 つしか選べない
- 4～6 番目の中からは、最大 1 つしか選べない
- 7～9 番目の中からは、最大 1 つしか選べない

2 小課題２の解法

- まずは以下のケースでの答えを考えてみよう
- 1～3 番目の中からは、最大 1 つしか選べない
- 4～6 番目の中からは、最大 1 つしか選べない
- 7～9 番目の中からは、最大 1 つしか選べない

選べる最大のものを選ぼう！

→ 答え：29+37+34=100



2 小課題２の解法

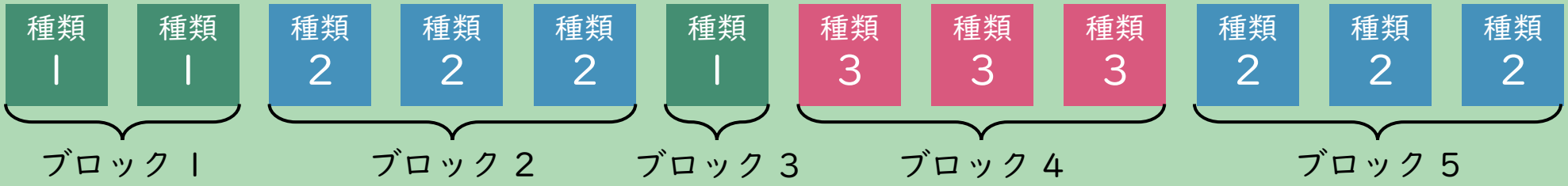
- まずは以下のケースでの答えを考えてみよう
- 1～3 番目の中からは、最大 1 つしか選べない
- 4～6 番目の中からは、最大 1 つしか選べない
- 7～9 番目の中からは、最大 1 つしか選べない

他のケースでも同じ方法で
答えが求められる

2 小課題 2 の解法

以下のアルゴリズムにより、答えが求められる

- N 個の絵を「同じ種類からなるブロック」に分割する
- 各ブロックについて、価値の最大値 (*) を求める
- 求める答えは (*) の合計



2 小課題 2 の実装


```python
# 入力
N = int(input())
M = int(input())
A = [None] * N
V = [None] * N
for i in range(N):
    A[i], V[i] = map(int, input().split());

# 答えを求める
Answer = 0
BlockMax = 0
for i in range(N):
  BlockMax = max(BlockMax, V[i])
  if (i == N-1) or (A[i] != A[i+1]):
    Answer += BlockMax
    BlockMax = 0

# 出力
print(Answer)
```

実装はやや大変だが、左のような実装例が考えられる

※Python での実装例

21 点

小課題 3

$N \leq 15$

3 全探索できるのか？

まず、絵の残し方は全部で何通りあるのかを考えよう

- 絵 1 の残し方： Yes/No の 2 通り
- 絵 2 の残し方： Yes/No の 2 通り
- 絵 3 の残し方： Yes/No の 2 通り

  ：

- 絵 N の残し方： Yes/No の 2 通り

積の法則 (*)より、残し方は全部で

$$2 \times 2 \times \cdots \times 2 = 2^N \text{ 通り}$$

たとえば $N = 3$ のときは 8 通り

(*) 知らない人は、 https://manabitimes.jp/math/1256 を確認しましょう！

3 全探索できるのか？

まず、絵の残し方は全部で何通りあるのかを考えよう

- 絵 1 の残し方： Yes/No の 2 通り
- 絵 2 の残し方： Yes/No の 2 通り
- 絵 3 の残し方： Yes/No の 2 通り
- ：
- 絵 N の残し方： Yes/No の 2 通り

積の法則 (*)より、残し方は全部で

$2 \times 2 \times \cdots \times 2 = 2^N$ 通り

たとえば $N = 3$ のときは 8 通り

小課題 3 の制約は $N \leq 15$

全部の残し方を調べることは可能か？

(*) 知らない人は、 https://manabitimes.jp/math/1256 を確認しましょう！

3 全探索できるのか？

- $N = 15$ の場合、全部で $2^{15} = 32768$ 通り を調べる必要がある
- 一方、コンピュータの計算速度は 1 秒当たり 10 億回程度
- したがって、全探索をすると小課題 3 が解ける

↓

問題：どうやって実装するか？

3 単純に実装すると…


```
// C++ での実装
for (int p1 = 0; p1 <= 1; p1++) {
  for (int p2 = 0; p2 <= 1; p2++) {
    for (int p3 = 0; p3 <= 1; p3++) {
      for (int p4 = 0; p4 <= 1; p4++) {
        for (int p5 = 0; p5 <= 1; p5++) {
          for (int p6 = 0; p6 <= 1; p6++) {
            :
          }
        }
      }
    }
  }
}
```

- 絵 i を残したかどうかを P_i とする
- ここで P_i は 0 または 1
- 単純に for 文ループを書いて全探索することも可能だが、N 重ループとなり実装が大変

3 実装の工夫：ビット全探索

- この問題を解消するため、ビット全探索という方法が用いられる
- for 文で $0 \leq t \leq 2^N - 1$ の範囲を全探索し、t の下から i 桁目が 1 のとき絵 i を選ぶとする手法（以下は $N = 3$ の例）

i の値	0	1	2	3	4	5	6	7
絵 3 を選ぶか P_3	0	0	0	0	1	1	1	1
絵 2 を選ぶか P_2	0	0	1	1	0	0	1	1
絵 1 を選ぶか P_1	0	1	0	1	0	1	0	1

3 参考資料

- ビット全探索に関する詳しい説明
  - 「アルゴリズム×数学」p.66
  - https://drken1215.hatenablog.com/entry/2019/12/14/171657
- 10 進数を 2 進数に変換する方法
  - 「アルゴリズム×数学」p.21
  - https://webkaru.net/clang/decimal-to-binary/

42点

小課題 4

$N \leq 100$

4 小課題 4 の解法

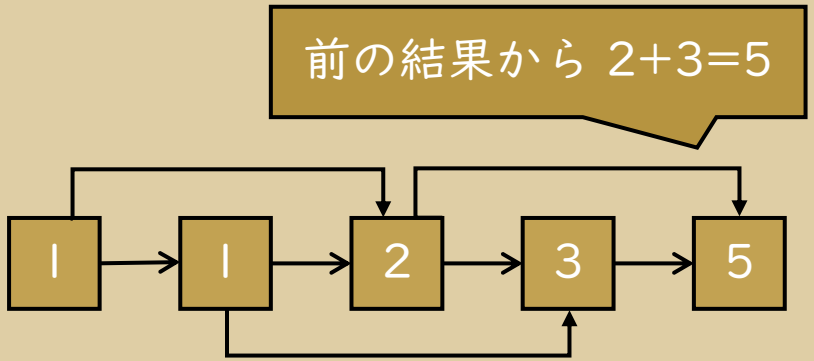
- 小課題 3 と同様に全探索をすると、大変なことになる
- 全部で $2^{100} = 1267650600228229401496703205376$ 通りを探索しなければならない
- コンピュータの計算速度は 1 秒に 10 億回程度であるため、単純計算すると計算に 40 兆年かかる

4　小課題 4 の解法

- ではどうするか？　→　以下の動的計画法を使おう！

動的計画法（DP）とは

前の結果を利用して問題を解く設計技法。フィボナッチ数などがイメージしやすい。詳しくは https://qiita.com/drken/items/dc53c683d6de8aeacf5a



4 小課題 4 の解法

- この問題で動的計画法を適用させよう
- $\mathrm{dp}[x]$:= (絵 $1, 2, \ldots, x$ まで考えたとき、絵 x を取った場合の価値合計の最大値) とする
- 絵 i の隣に絵 x を置ける条件は $A_i \neq A_x$ であるため、$dp[x]$ は $A_i \neq A_x$ を満たすような i における $dp[i] + V_i$ の最大値である

➡ 難しいので、例を考えてみよう！

4 具体例

- たとえば、以下のようなケースを考える

4 具体例

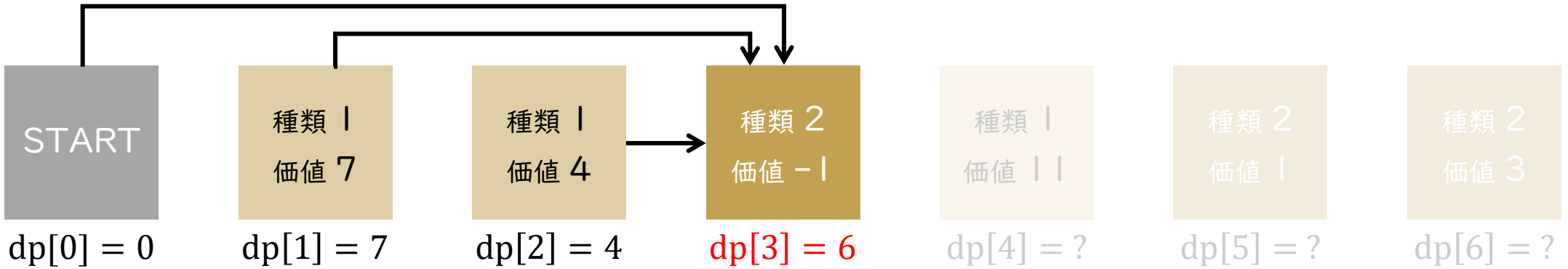
- たとえば、以下のようなケースを考える
  - dp[1] は「絵 1 を残すときの合計価値の最大値」
  - 絵 1 を取るしか選択肢がないので、$\mathrm{dp}[1] = \mathrm{dp}[0] + V_1 = 7$

4 具体例

- たとえば、以下のようなケースを考える
  - dp[2] は「絵 1 を残すときの合計価値の最大値」
  - 絵 2 は絵 1 の隣に置くことができないので、$\mathrm{dp}[2] = \mathrm{dp}[0] + V_2 = 4$

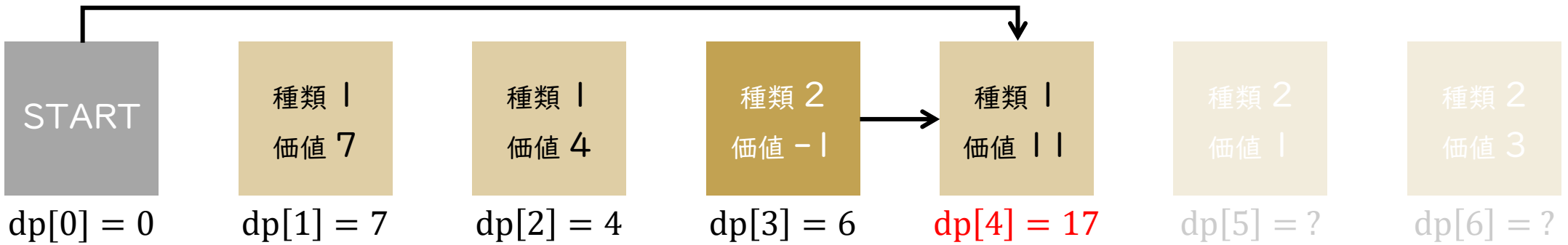
4 具体例

- たとえば、以下のようなケースを考える
  - dp[2] は「絵 1 を残すときの合計価値の最大値」
  - 絵 3 は一番左に置くか、絵 1・2 の隣に置くことができる
  - よって、$\mathrm{dp}[3] = \max(\mathrm{dp}[0], \mathrm{dp}[1], \mathrm{dp}[2]) + V_3 = 6$



4 具体例

- たとえば、以下のようなケースを考える
  - dp[2] は「絵 1 を残すときの合計価値の最大値」
  - 絵 4 は一番左に置くか、絵 3 の隣に置くことができる
  - よって、$\mathrm{dp}[4] = \max(\mathrm{dp}[0], \mathrm{dp}[3]) + V_4 = 17$



4 具体例

- たとえば、以下のようなケースを考える
  - dp[2] は「絵 1 を残すときの合計価値の最大値」
  - 絵 5 は一番左に置くか、絵 1・2・4 の隣に置くことができる
  - よって、$\mathrm{dp}[5] = \max(\mathrm{dp}[0], \mathrm{dp}[1], \mathrm{dp}[2], \mathrm{dp}[4]) + V_5 = 18$

4 具体例

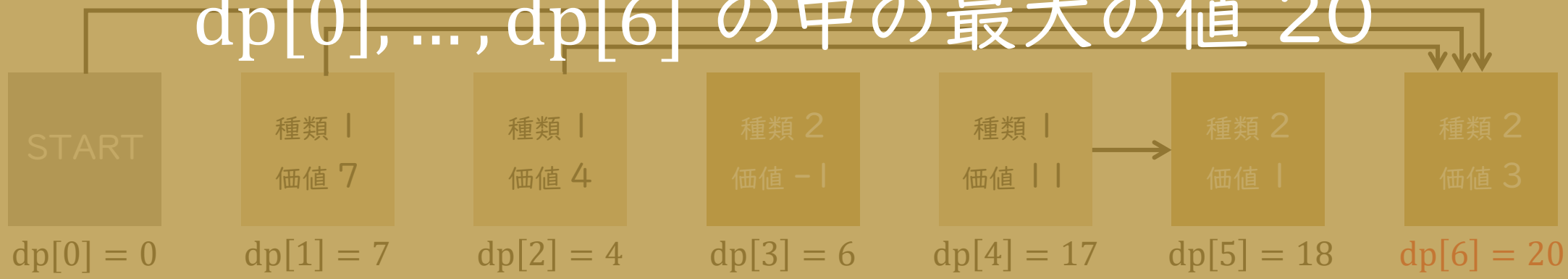
- たとえば、以下のようなケースを考える
  - dp[2] は「絵 1 を残すときの合計価値の最大値」
  - 絵 6 は一番左に置くか、絵 1・2・4 の隣に置くことができる
  - よって、$\mathrm{dp}[6] = \max(\mathrm{dp}[0], \mathrm{dp}[1], \mathrm{dp}[2], \mathrm{dp}[4]) + V_6 = 20$

4 具体例

- たとえば、以下のようなケースを考える
  - dp[2] は「絵 1 を残すときの合計価値の最大値」
  - 絵 6 は一番右に置くため、絵 1, 2, 4 の隣に置くことができる
  - よって、$\mathrm{dp}[6] = \max(\mathrm{dp}[0], \mathrm{dp}[1], \mathrm{dp}[2], \mathrm{dp}[4]) + V_6 = 20$

求める答え（残す絵の合計価値の最大）は

dp[0], …, dp[6] の中の最大の値 20



4 小課題 4 の解法


```cpp
// C++ での実装
// 動的計画法
dp[0] = 0;
for (int i = 1; i <= N; i++) {
    dp[i] = dp[0] + V[i];
    for (int j = 1; j < i; j++) {
        if (A[j] != A[i]) {
            dp[i] = max(dp[i], dp[j] + V[i]);
        }
    }
}

// 答え Ans を求める
int Ans = 0;
for (int i = 0; i <= N; i++) Ans = max(Ans, dp[i]);
cout << Ans << endl;
```

- 一般のケースでも、左のようなプログラムに従って計算すれば良い
- 計算量は $O(N^2)$ なので、$N \leq 100$ ならば通る

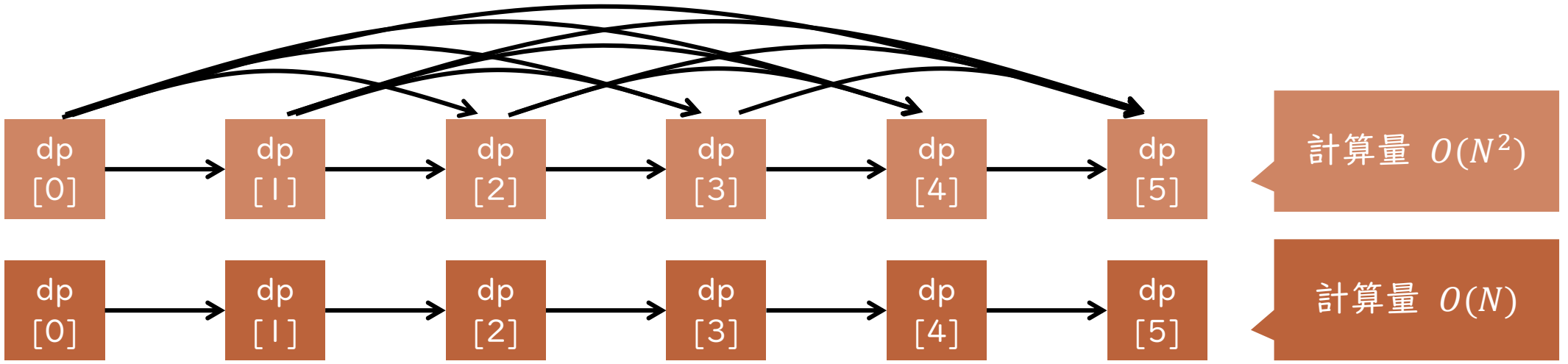
69点

小課題 5

$M \leq 100$

5 計算量を減らす方法

- 小課題 4 の解法の計算が遅い理由は、一つの pos に対して $\mathrm{dp}[pos]$ を計算するのに $O(N)$ 時間かかるから
- 上の状態遷移を、下の状態遷移のようにできないか？



5 小課題 5 の解法

- まずは $M = 2$ の場合について、以下の DP 配列を考えよう
  - $\mathrm{dp1}[x]$: 絵 $1, \ldots, x$ のみを考えたとき、最後に残した絵の種類が 1 の場合の合計価値最大値
  - $\mathrm{dp2}[x]$: 絵 $1, \ldots, x$ のみを考えたとき、最後に残した絵の種類が 2 の場合の合計価値最大値

※最後に残した絵：絵 $1, 2, \ldots, x$ の中で残したもののうち、最も番号が大きい絵（一番右の絵）を指す。

- 同じ種類の絵を隣に置くことはできないので、答えは以下のように計算できる
  - $A_i = 1$ のとき： $\mathrm{dp1}[i] = \max(\underbrace{\mathrm{dp1}[i-1]}_{\text{直前が種類 1 の絵で絵 } i \text{ を残さない}}, \underbrace{\mathrm{dp2}[i-1] + V_i}_{\text{直前が種類 2 の絵で絵 } i \text{ を残す}})$, $\mathrm{dp2}[i] = \underbrace{\mathrm{dp2}[i-1]}_{\text{直前が種類 2 の絵で絵 } i \text{ を残さない}}$
  - $A_i = 2$ のとき： $\mathrm{dp2}[i] = \max(\underbrace{\mathrm{dp2}[i-1]}_{\text{直前が種類 2 の絵で絵 } i \text{ を残さない}}, \underbrace{\mathrm{dp1}[i-1] + V_i}_{\text{直前が種類 1 の絵で絵 } i \text{ を残す}})$, $\mathrm{dp1}[i] = \underbrace{\mathrm{dp1}[i-1]}_{\text{直前が種類 1 の絵で絵 } i \text{ を残さない}}$

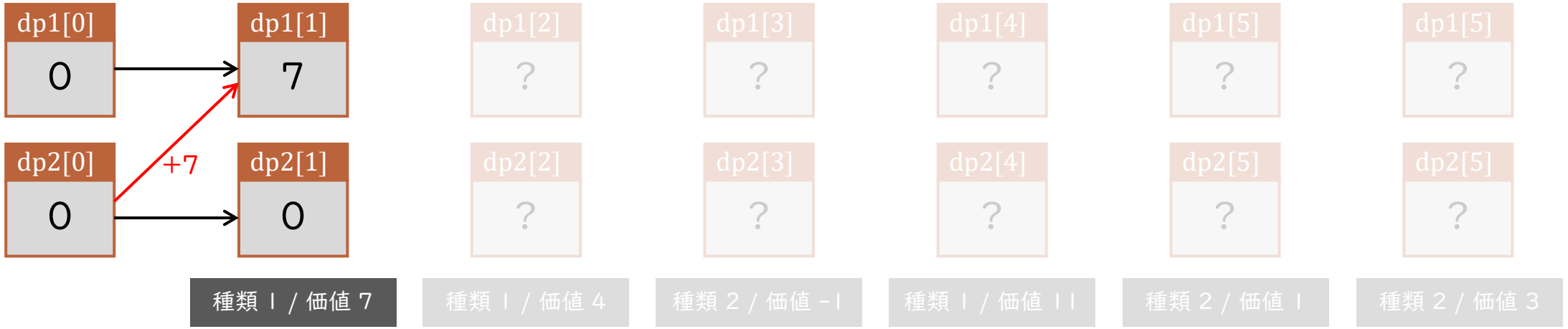
5 小課題 5 の解法

- 難しいので、たとえば以下のようなケースを考えよう

dp1[0]	dp1[1]	dp1[2]	dp1[3]	dp1[4]	dp1[5]	dp1[5]
0	?	?	?	?	?	?
dp2[0]	dp2[1]	dp2[2]	dp2[3]	dp2[4]	dp2[5]	dp2[5]
0	?	?	?	?	?	?
	種類 1 / 価値 7	種類 1 / 価値 4	種類 2 / 価値 -1	種類 1 / 価値 11	種類 2 / 価値 1	種類 2 / 価値 3

5 小課題 5 の解法

- 難しいので、たとえば以下のようなケースを考えよう
  - $\mathrm{dp1}[1] = \max(\mathrm{dp1}[0], \mathrm{dp2}[0] + 7) = 7$　　[最後に残した絵が種類 1 のときの合計価値最大]
  - $\mathrm{dp2}[1] = \mathrm{dp2}[0] = 0$　　[最後に残した絵が種類 2 のときの合計価値最大]

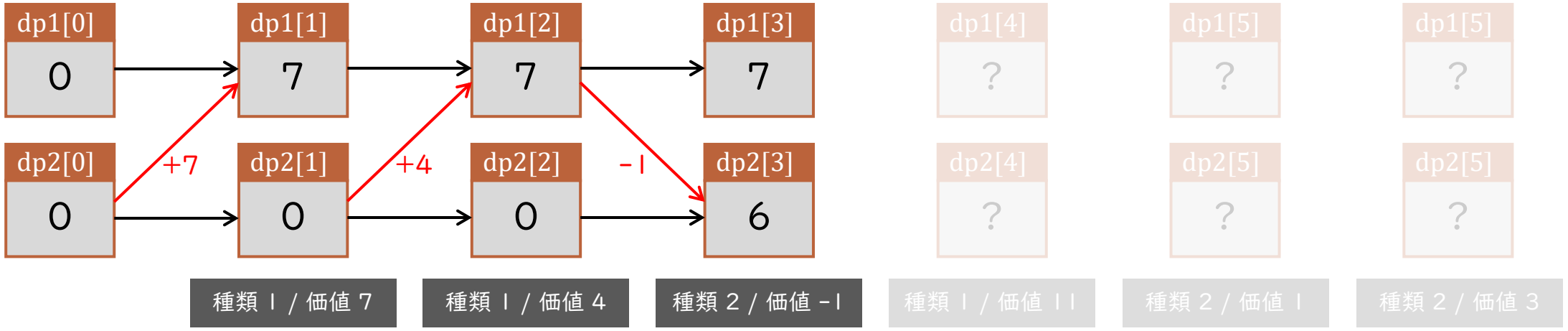


5 小課題 5 の解法

- 難しいので、たとえば以下のようなケースを考えよう
  - $\mathrm{dp1}[2] = \max(\mathrm{dp1}[1], \mathrm{dp2}[1] + 4) = 7$　［最後に残した絵が種類 1 のときの合計価値最大］
  - $\mathrm{dp2}[2] = \mathrm{dp2}[1] = 0$　［最後に残した絵が種類 2 のときの合計価値最大］

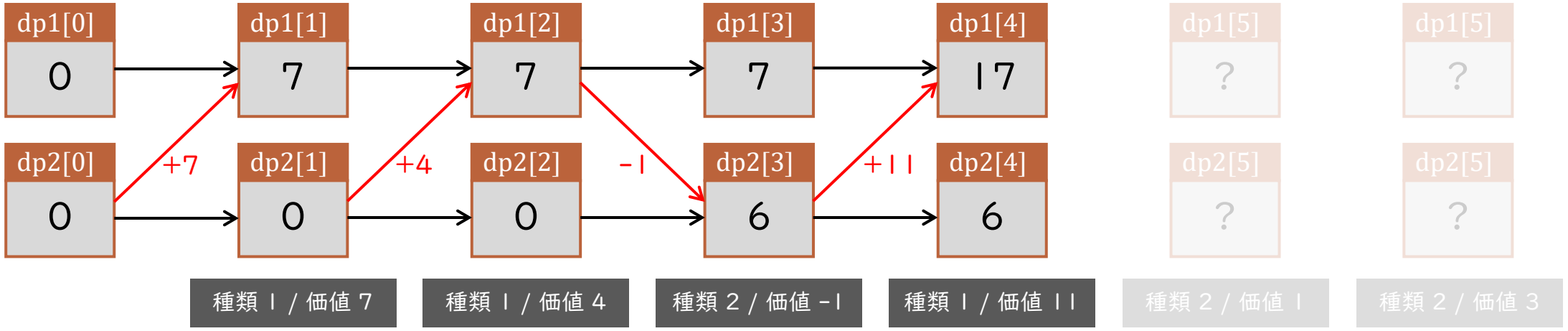
5 小課題 5 の解法

- 難しいので、たとえば以下のようなケースを考えよう
  - $\mathrm{dp1}[3] = \mathrm{dp1}[2] = 7$　　［最後に残した絵が種類 1 のときの合計価値最大］
  - $\mathrm{dp2}[3] = \max(\mathrm{dp2}[2], \mathrm{dp1}[2] - 1) = 6$　　［最後に残した絵が種類 2 のときの合計価値最大］



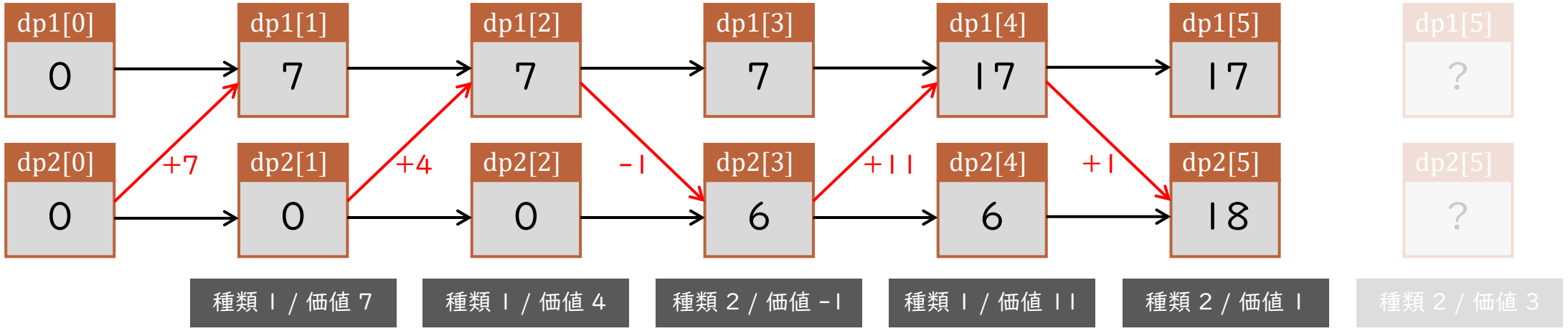
5 小課題 5 の解法

- 難しいので、たとえば以下のようなケースを考えよう
  - $\mathrm{dp1}[4] = \max(\mathrm{dp1}[3], \mathrm{dp2}[3] + 11) = 17$ ［最後に残した絵が種類 1 のときの合計価値最大］
  - $\mathrm{dp2}[4] = \mathrm{dp2}[3] = 6$ ［最後に残した絵が種類 2 のときの合計価値最大］



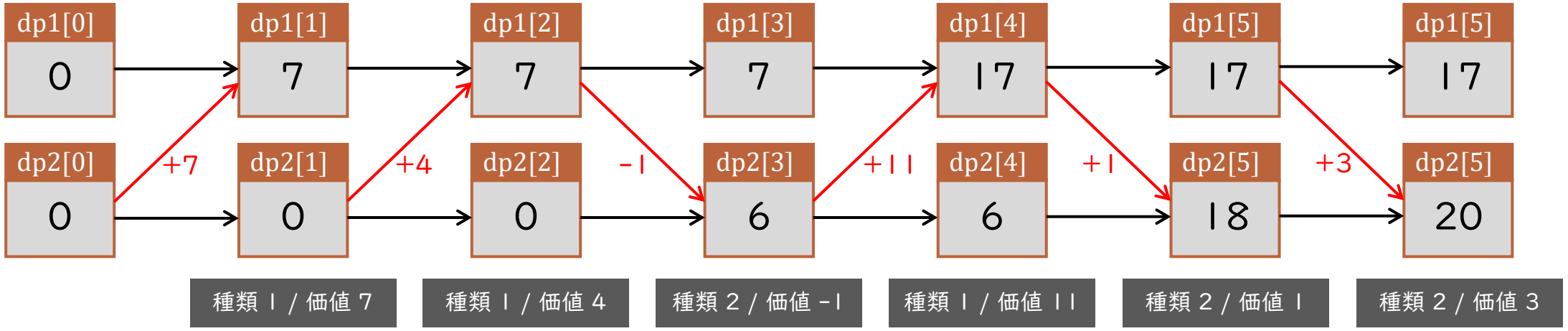
5 小課題 5 の解法

- 難しいので、たとえば以下のようなケースを考えよう
  - $\mathrm{dp1}[5] = \mathrm{dp1}[4] = 17$　　［最後に残した絵が種類 1 のときの合計価値最大］
  - $\mathrm{dp2}[5] = \max(\mathrm{dp2}[4], \mathrm{dp1}[4] + 1) = 18$　　［最後に残した絵が種類 2 のときの合計価値最大］



5 小課題 5 の解法

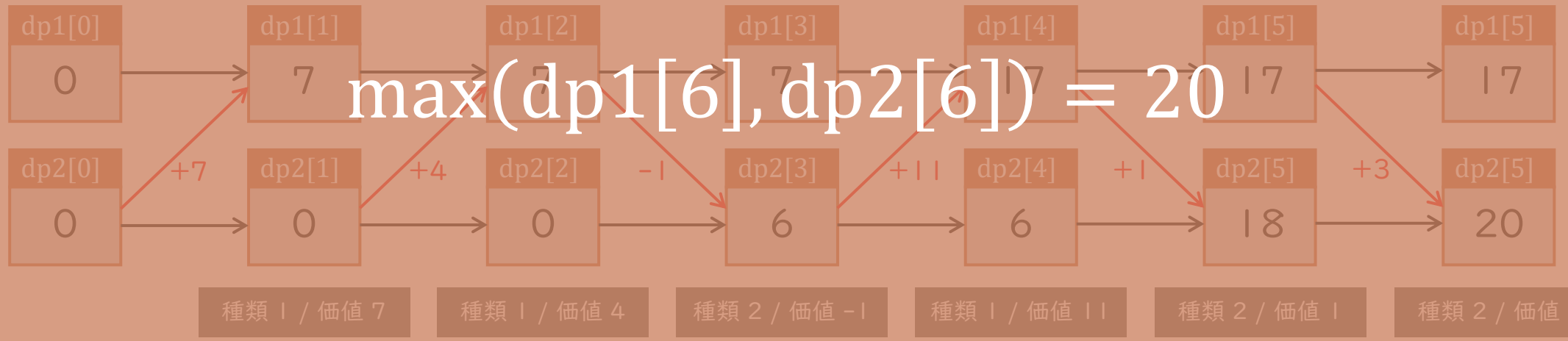
- 難しいので、たとえば以下のようなケースを考えよう
  - $\mathrm{dp1}[6] = \mathrm{dp1}[5] = 17$　　［最後に残した絵が種類 1 のときの合計価値最大］
  - $\mathrm{dp2}[6] = \max(\mathrm{dp2}[5], \mathrm{dp1}[5] + 3) = 20$　　［最後に残した絵が種類 2 のときの合計価値最大］



5 小課題 5 の解法

- 難しいので、たとえば以下のようなケースを考えよう
  - $dp1[6] = dp1[5] = 17$ ［最後に残した絵が種類 1 のときの合計価値最大］
  - $dp2[6] = \max(dp2[5], dp1[5] + 3) = 20$ ［最後に残した絵が種類 2 のときの合計価値最大］

求める答え（残す絵の合計価値の最大）は

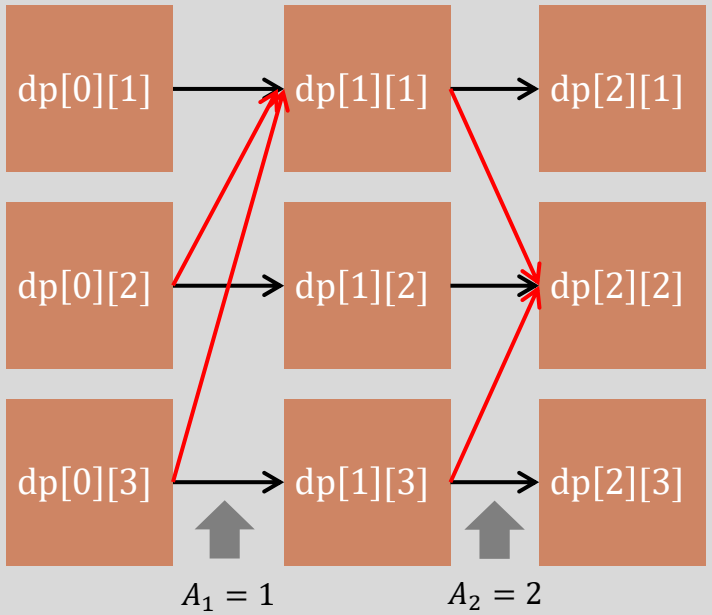


$$\max(dp1[6], dp2[6]) = 20$$

5 小課題 5 の解法

M が一般の場合も、同じようにして解ける（詳しい方針は以下の通り）

- $\mathrm{dp}[pos][col]$ を「絵 $1, 2, \ldots, pos$ を考えたときに、最後に残した絵の種類が col である場合の合計価値最大」とする
- 最終的な答えは $\mathrm{dp}[pos][1], \ldots, \mathrm{dp}[pos][M]$ の最大値となる
- イメージ図は右の通り



5 小課題 5 の実装


```cpp
// C++ での実装
// 動的計画法
for (int i = 1; i <= M; i++) dp[0][i] = 0;
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= M; j++) dp[i][j] = dp[i-1][j];
    for (int j = 1; j <= M; j++) {
        if (j == A[i]) continue;
        dp[i][A[i]] = max(dp[i][A[i]], dp[i-1][j] + V[i]);
    }
}

// 答えを出力
int Answer = 0;
for (int i = 1; i <= M; i++) Answer = max(Answer, dp[N][i]);
cout << Answer << endl;
```

- このアルゴリズムを実装すると、左のようになる
- 計算量は $O(NM)$ なので、$M \leq 100$ ならば通る

88点

小課題 6

追加の制約はない

6 小課題 6 の解法

- 絵 i を置いた場合、$\mathrm{dp}[i][A_i]$ の値は以下のうち最大の値
  - $\mathrm{dp}[i-1][1]+V_i$
  - ：
  - $dp[i-1][A_i-1]+V_i$
  - $dp[i-1][A_i+1]+V_i$
  - ：
  - $dp[i-1][M]+V_i$

そこで最大の値となり得るのは

$\mathrm{dp}[i-1][1], \ldots, \mathrm{dp}[i-1][N]$ のうち上位 2 つだけ

注：1 位が $\mathrm{dp}[i-1][A_i]$ である場合に限り、左の最大の値が 2 位のものになる

6 小課題 6 の解法

- したがって、$\mathrm{dp}[i][1], \ldots, \mathrm{dp}[i][M]$ 全部をメモする必要はない
- メモする必要があるのは上位 2 つだけ
  - $\mathrm{dp1}[i]$: $\mathrm{dp}[i][1], \ldots, \mathrm{dp}[i][M]$ の中で最も大きい値
  - $\mathrm{dp2}[i]$: $\mathrm{dp}[i][1], \ldots, \mathrm{dp}[i][M]$ の中で 2 番目に大きい値

- そうすると、考えるべき状態の数が $2N$ 通りに減る
- 計算量は $O(N)$ [実装はやや大変です]

100点

6 小課題 6 の実装例


```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int N, M;
int A[150009], V[150009];
pair<int, int> dp1[150009];
pair<int, int> dp2[150009];

int main() {
    // Step #1. 入力
    cin >> N >> M;
    for (int i = 1; i <= N; i++) {
        cin >> A[i] >> V[i];
    }

    // Step #2. 配列の初期化
    for (int i = 1; i <= N; i++) {
        dp1[i] = make_pair(-2000000000, -1);
        dp2[i] = make_pair(-2000000000, -1);
    }
    dp1[1] = make_pair(V[1], A[1]);
    dp2[1] = make_pair(0, -1);

    // Step #3. 動的計画法
    for (int i = 2; i <= N; i++) {
        vector<pair<int, int>> vec;
        if (dp1[i - 1].second != A[i]) vec.push_back(make_pair(dp1[i - 1].first + V[i], A[i]));
        if (dp2[i - 1].second != A[i]) vec.push_back(make_pair(dp2[i - 1].first + V[i], A[i]));
        vec.push_back(make_pair(dp1[i - 1].first, dp1[i - 1].second));
        vec.push_back(make_pair(dp2[i - 1].first, dp2[i - 1].second));
        sort(vec.begin(), vec.end());
        reverse(vec.begin(), vec.end());

        dp1[i] = vec[0];
        for (int j = 1; j < (int)vec.size(); j++) {
            if (vec[j].second == vec[0].second) continue;
            dp2[i] = vec[j];
            break;
        }
    }

    // Step #4. Output
    cout << dp1[N].first << endl;
    return 0;
}
```