動的計画法入門

kya(@tsaayk)
TMU-CS B4
July 02, 2022

目次

- フィボナッチ数を求めてみよう
  - 再帰関数による実装
  - メモ化
- 動的計画法とは？
- 余談
- 典型問題集
- 典型問題: EDPC A - Frog 1

フィボナッチ数を求めてみよう

- いきなりですがこの問題は解けますか？

以下で定義される数列 F はフィボナッチ数列と呼ばれます.

- $F_1 = 1$
- $F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}\ (n \geq 3)$

正整数 n が与えられたとき, フィボナッチ数列の第 n 項を返す関数を作成してください.

フィボナッチ数を求めてみよう

- 漸化式通りにプログラムを書いてみると…

```python
# Python の場合
def F(n):
    if n <= 2:
        return 1
    else:
        return F(n-1) + F(n-2)
```

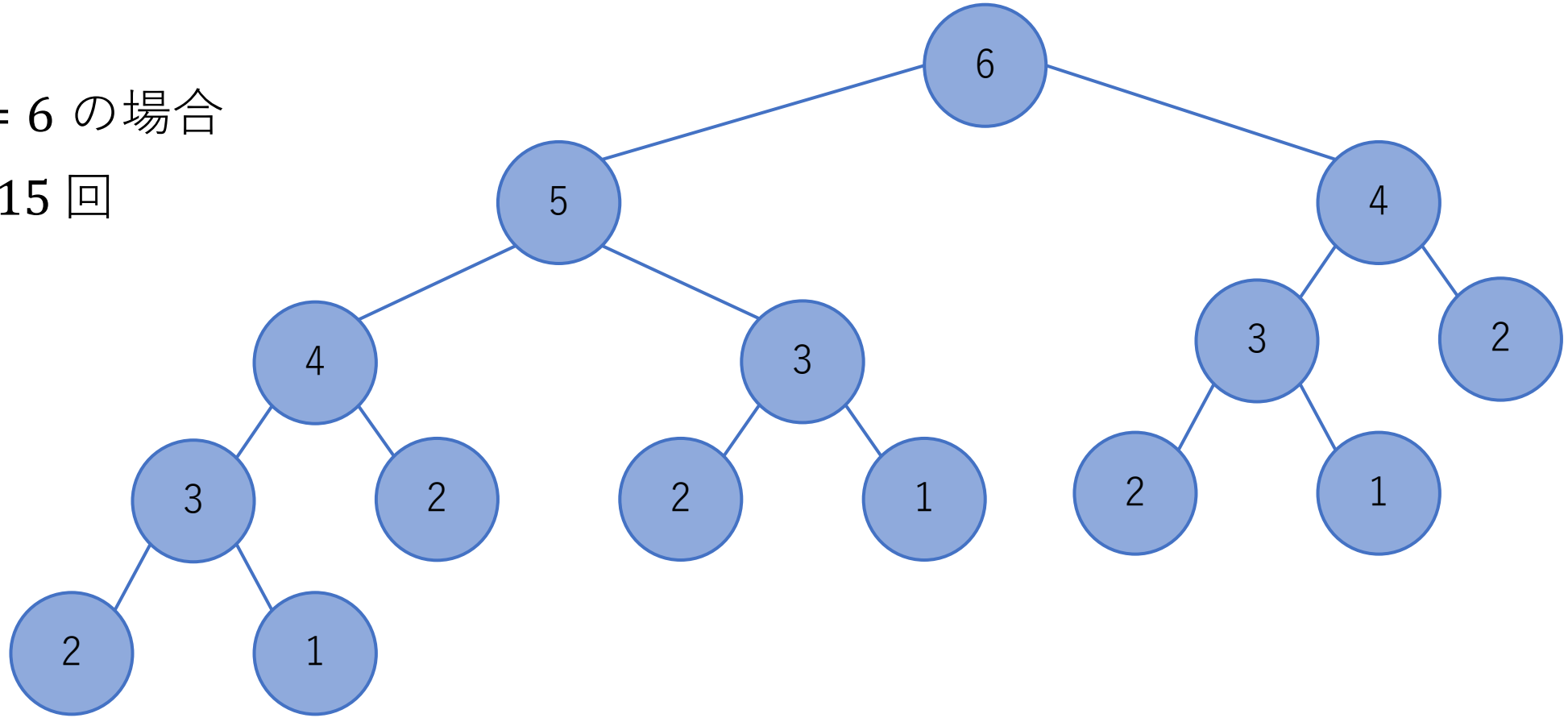
```cpp
// C++ の場合
int F (int n) {
    if (n <= 2) {
        return 1;
    } else {
        return F(n-1) + F(n-2);
    }
}
```

フィボナッチ数を求めてみよう

- 計算量（関数の呼び出し回数）はどのくらい？

$n = 6$ の場合

→ 15 回



フィボナッチ数を求めてみよう

- この方法だと n が大きくなったときに計算量が爆増しそう

- 直観的には 2^{n-1} 回くらい
  - 各ステップで関数の呼び出しが 2 回行われる

- 実際の計算量は $O\left(\left(\frac{1-\sqrt{5}}{2}\right)^n\right)$
  - だいたい $O(2^n)$ くらいだと思って OK

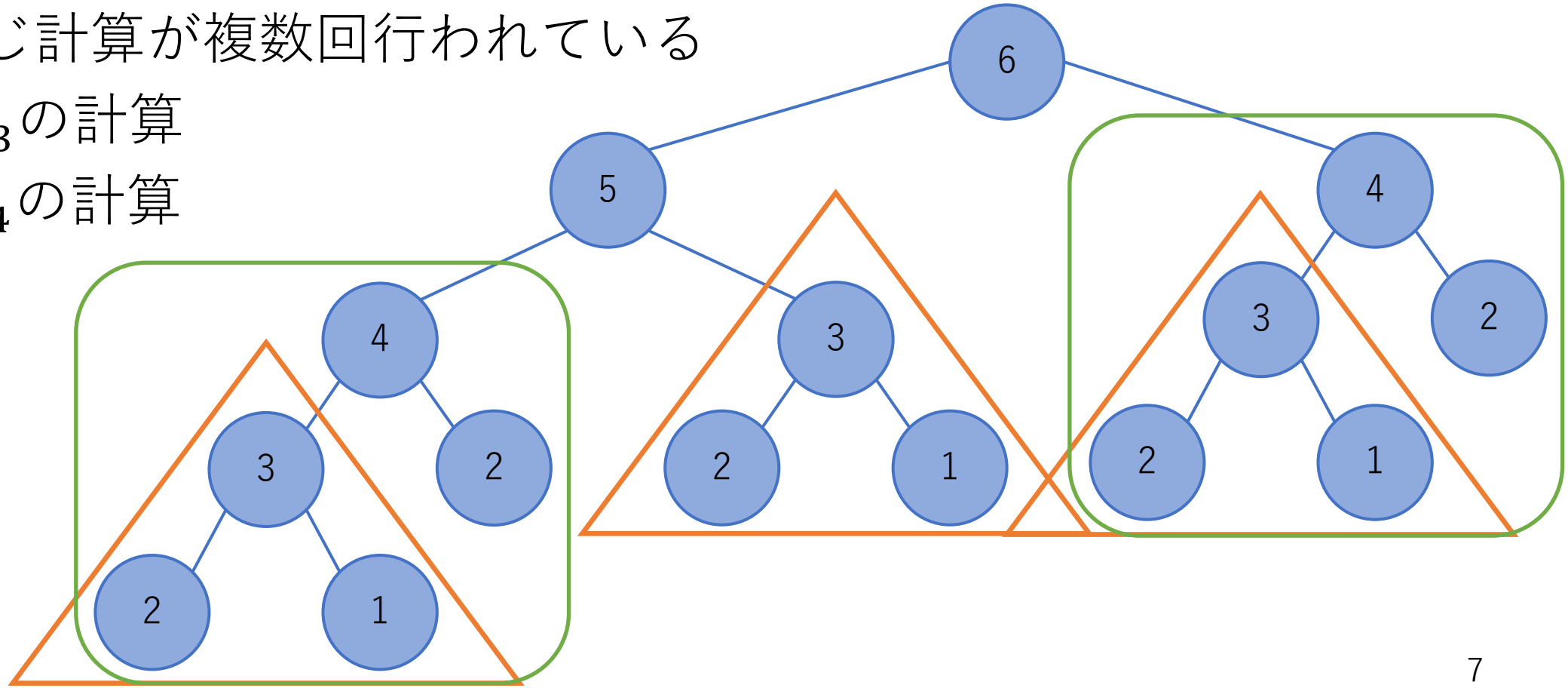
フィボナッチ数を求めてみよう

- 計算量を削減できないか？

→ 同じ計算が複数回行われている

△：F_3の計算

▢：F_4の計算



フィボナッチ数を求めてみよう

- 計算結果を配列などに格納して同じ計算をしないように工夫する
→メモ化などと呼ばれる

```python
# Python の場合
memo = [-1] * 100
def F(n):
    if memo[n] != -1:
        return memo[n]
    elif n <= 2:
        return 1
    else:
        memo[n] = F(n-1) + F(n-2)
        return memo[n]
```

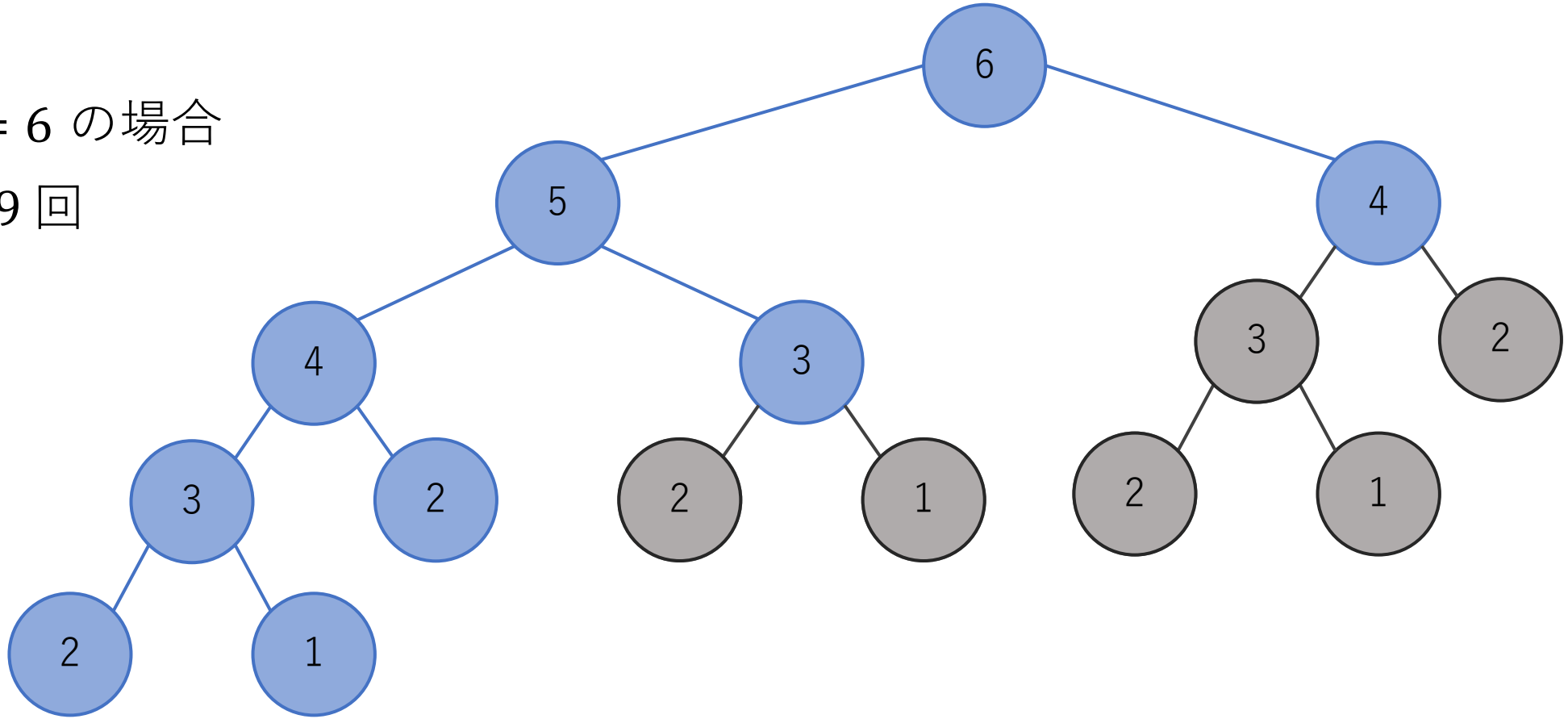
```cpp
// C++ の場合
std::vector<int> memo(100, -1);
int F (int n) {
    if (memo[n] != -1) {
        return memo[n];
    } else if (n <= 2) {
        return 1;
    } else {
        memo[n] = F(n-1) + F(n-2);
        return memo[n];
    }
}
```

フィボナッチ数を求めてみよう

- 改善後の計算量（関数の呼び出し回数）はどのくらい？

$n = 6$ の場合

→ 9 回



フィボナッチ数を求めてみよう

- n が小さいケースでは実感しづらいが, 大幅な削減になっている

- 以下のページでどのくらい計算量が減るのかを確認できます
  - https://wandbox.org/permlink/Tw9BpOQJHV3hqR4z
  - ページ下部の「コピーして編集」をクリックするとプログラムが編集できるので, n の値を自由に変更してみてください

再帰関数の呼び出し回数

n	naive	memo
1	1	1
2	1	1
3	3	3
4	5	5
5	9	7
6	15	9
7	25	11
8	41	13
9	67	15
10	109	17
11	177	19
12	287	21
13	465	23
14	753	25
15	1219	27
16	1973	29
17	3193	31
18	5167	33
19	8361	35
20	13529	37

動的計画法とは？

- 本題に戻って……
- 以下の 2 つの条件を満たすアルゴリズムを動的計画法 (Dynamic Programming) と呼ぶ
  1. 帰納的な関係の利用：より小さな問題例の解や計算結果を帰納的な関係を利用してより大きな問題例を解くのに使用する。
  2. 計算結果の記録：小さな問題例、計算結果から記録し、同じ計算を何度も行うことを避ける。帰納的な関係での参照を効率よく行うために、計算結果は整数、文字やその組みなどを見出しにして管理される。

(wikipediaより引用)

動的計画法とは？

- 帰納的な関係の利用

→ フィボナッチ数列には $F_n = F_{n-1} + F_{n-2}$ という帰納的な関係があり, F_nを求める際に F_{n-1}, F_{n-2} の答えを利用した

- 計算結果の記録

→同じ計算が何度も行われることを避けるために計算結果をメモした

動的計画法とは？

- 「動的計画法」はあくまでもアルゴリズム全体におけるカテゴリの名前
  - 条件を満たすアルゴリズムは全て動的計画法に含まれる
- 「二分探索」や「bit全探索」などとは違って特定のアルゴリズムを指すわけではない！

アルゴリズム名

- 二分探索
- ユークリッドの互除法
- ダイクストラ法
- エラトステネスの篩

など

カテゴリ名

- 動的計画法
- グラフアルゴリズム

など

余談

- フィボナッチ数は単純に for-loop を回して求めることもできます.
- やってることはメモ化して再帰関数で計算するのと同じ

```
# for-loop でフィボナッチ数を求める場合
def F(n):
    fibonacci = [1] * n
    for i in range(2, n):
        fibonacci[i] = fibonacci[i-1] + fibonacci[i-2]
    return fibonacci[n-1]
```

DP が解けるようになるためには？

- 「動的計画法」はカテゴリ名
  - 問題の数だけアルゴリズムが存在する
  - ABC-C 程度の簡単なものから AGC-F くらい難しいものまで多数存在

- たくさん問題を解いてコツを掴む必要がある

- しかし全ての問題を解くのは不可能…

→ まずは典型問題を抑えれば OK
  - 典型問題の解法（アルゴリズム・考え方）を理解すると多くの問題で応用できるようになる

典型問題集

- Educational DP Contest (EDPC)
  - DP に関する典型問題が 26 問含まれたコンテスト
  - まずは H 問題くらいまで解いてみるのがおすすめ

- Typical DP Contest (TDPC)
  - 同じく典型問題が 20 問含まれているコンテスト
  - EDPC より難しめ

- 競技プログラミングにおける動的計画法問題まとめ
  - hamayanhamayan さんのブログ

典型問題: EDPC A - Frog 1

- 足場 i にたどり着くまでの最小を求めるには？

→ 足場 $i-1, i-2$ にたどり着くまでの最小が分かれば計算可能

- 足場 i にたどり着くまでの最小コストを dp_i とすると

$$dp_i = \min(dp_{i-1} + |h_i - h_{i-1}|, dp_{i-2} + |h_i - h_{i-2}|)$$

- $i = 1, 2$ の時に h_{i-1}, h_{i-2} が存在しない可能性があることに注意

典型問題: EDPC A - Frog 1

- 典型的な DP の考え方として「i 番目までの答えが分かっている時に $i+1$ 番目の答えを計算することは可能か？」というのがある
  - 数列に関する問題や文字列に関する問題で頻出

- これができれば部分問題の答えを利用して元の問題の答えを求めることが可能
- 計算結果のメモはだいたいできる

→ 動的計画法が使える