「アルゴリズム」資料
18. 動的計画法とメモ化

奈良女子大学理学部情報科学科 鴨浩靖

2010 年 6 月 30 日 初版
2014 年 1 月 20 日 第二版

例 1

次の式で定義される数列の第 n 項を求める。

$$a_0 = 1$$

$$a_n = a_{\lfloor n/3 \rfloor} + a_{\lfloor n/2 \rfloor} \quad (n > 0 \text{ のとき})$$

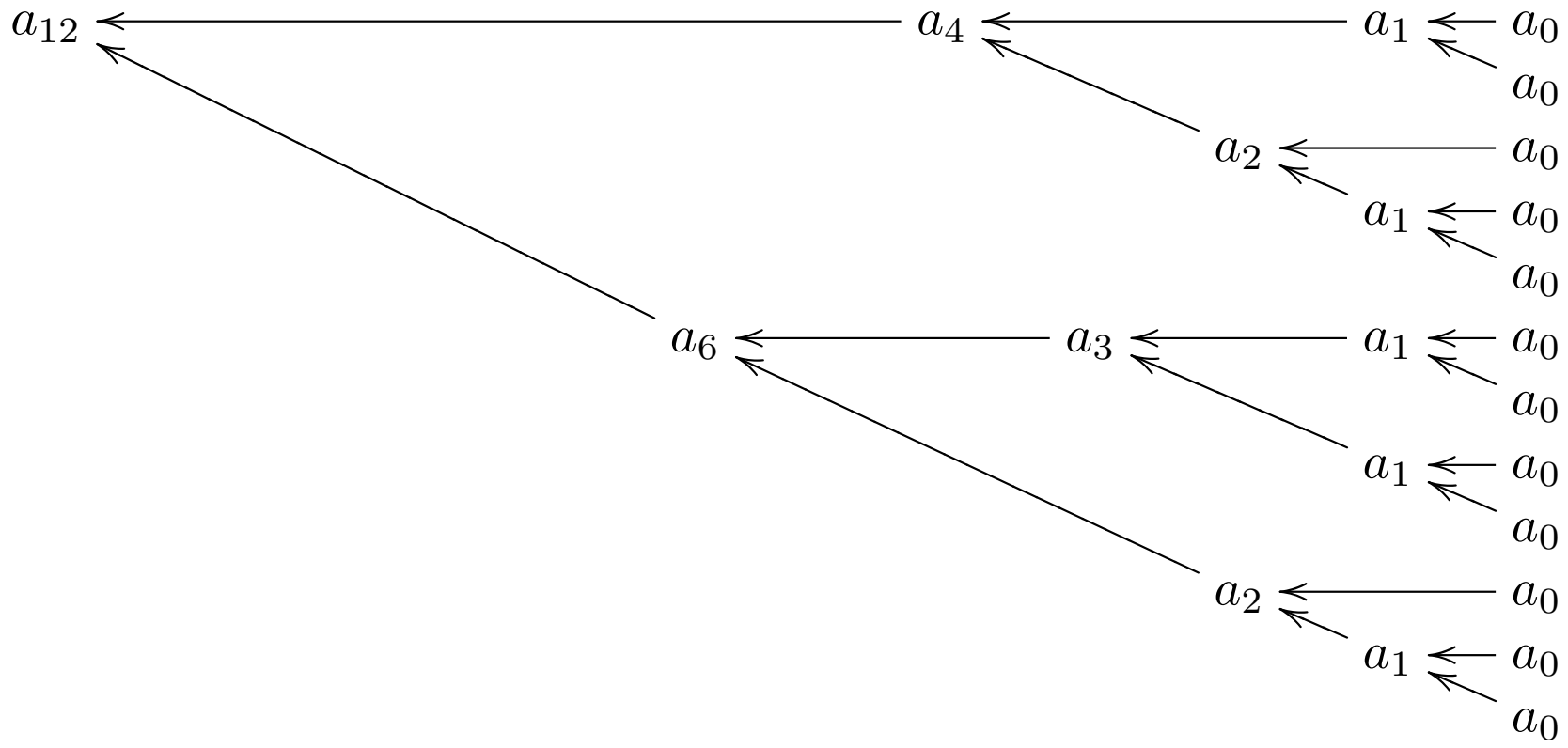
素朴な再帰呼び出し

漸化式の通りに再帰呼び出しで実装する。

素朴な再帰呼び出しによる例 1 の実装

```
int mikan(int x)
{
  if (x > 0) {
    int y3, y2;
    y3 = mikan(x / 3);
    y2 = mikan(x / 2);
    return y3 + y2
  } else {
    return 1;
  }
}
```

素朴な再帰呼び出しの実行例



同じ計算を何度も行っている。

動的計画法

小さいほうから計算し、計算結果は記録し再利用する。

動的計画法による例 1 の実装

```
int mikan_array[SIZE];
void init_mikan_array()
{
  int i;
  mikan_array[0] = 1;
  for (i = 1; i < SIZE; i ++) {
    mikan_array[i] = mikan_array[i / 2] + mikan_array[i / 3];
  }
}
int mikan(int x)
{
  return mikan_array[x];
}
```

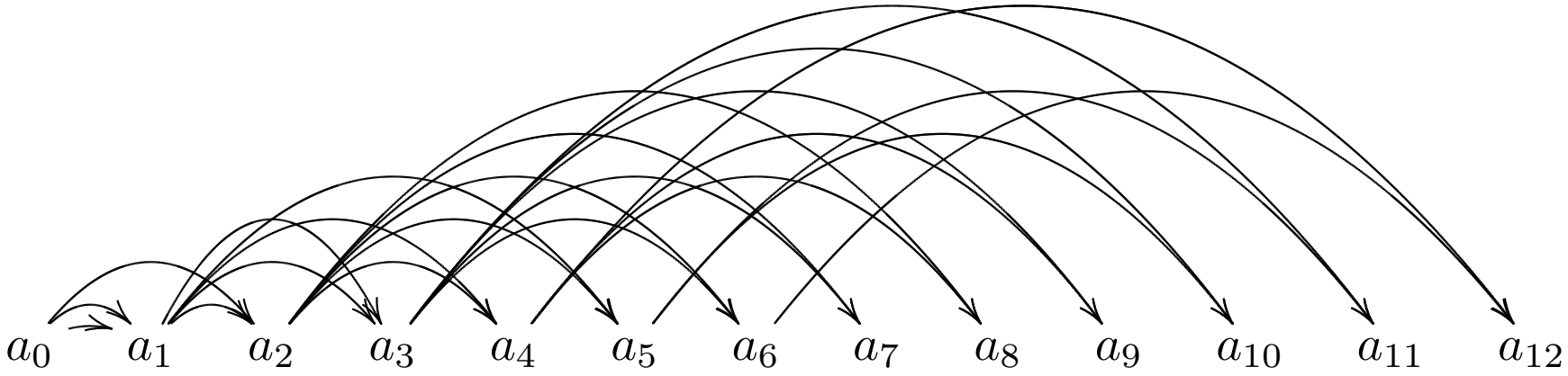
動的計画法による例 1 の（ちょっとトリッキーな）実装 1

```
int mikan(int x)
{
  static char initialized;
  static int array[SIZE];
  if (!initialized) {
    int i;
    array[0] = 1;
    for (i = 1; i < SIZE; i ++) {
      array[i] = array[i / 3] + array[i / 2];
    }
    initialized = 1;
  }
  return array[x];
}
```

動的計画法による例 1 の（ちょっとトリッキーな）実装 2

```
int mikan(int x)
{
  static int *array = NULL;
  if (array == NULL) {
    int i;
    array = malloc(SIZE * sizeof(int));
    array[0] = 1;
    for (i = 1; i < SIZE; i ++) {
      array[i] = array[i / 3] + array[i / 2];
    }
  }
  return array[x];
}
```

動的計画法の実行例



不要なものまで計算している。

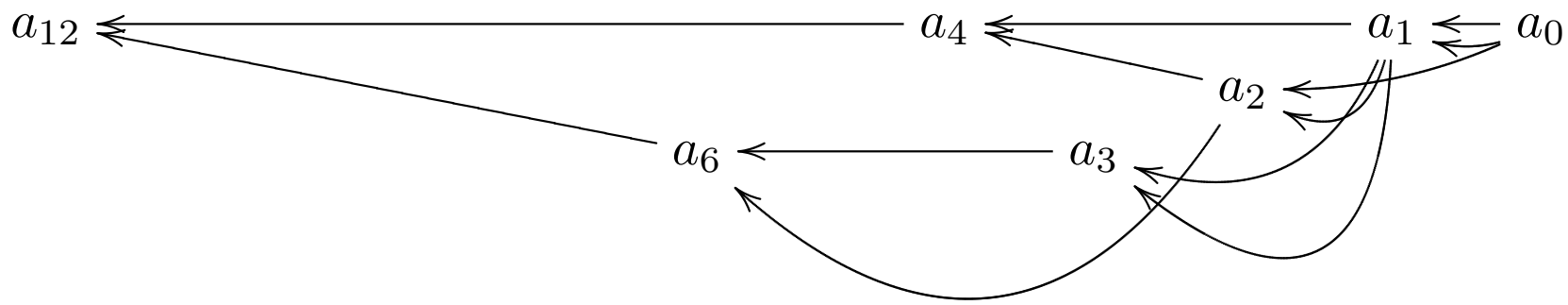
メモ化

途中の計算結果を記録しながら計算し、同じ計算が必要になったら以前の計算結果を再利用する。

メモ化による例 1 の実装

```
int mikan(int x)
{
  static char marks[SIZE]; static int array[SIZE];
  if (!marks[x]) {
    if (x > 0) {
      int y3 = mikan(x / 3); int y2 = mikan(x / 2);
      array[x] = y3 + y2;
    } else {
      array[x] = 1;
    }
    marks[x] = 1;
  }
  return array[x];
}
```

メモ化の実行例



アルゴリズムの選び方

- 計算の重複が起きる心配がないか、気にならないほど頻度が低い場合は、素朴な再帰呼び出しで十分。
- 計算に必要な値がとびとびの場合は、メモ化が計算時間を節約できて良い。
- 計算に必要な値がぎちぎちの場合は、動的計画法が計算時間を節約できて良い。

うまく使い分けると良い。

ACM ICPC 問題

1999 年アジア地区予選京都大会問題 A Square Coins

素朴な再帰でもメモ化でも動的計画法でも可。

2007 年アジア地区予選東京大会問題 C Minimal Backgammon

動的計画法が最適。