5.1.3　ハノイの塔

ハノイの塔は、フランスの数学者エドゥアール・リュカが 1883 年に発売したゲーム『ハノイの塔』がルーツとされています (図 5.1 参照)。また、このゲームのリーフレットには以下のような伝説が書かれていています。

ゲームのルール

- 3 本の杭と、中央に穴の開いた大きさの異なる複数の円盤から構成される。
- 最初はすべての円盤が左端の杭に小さいものが上になるように順に積み重ねられている。
- 円盤を 1 度に 1 枚ずつどれかの杭に移動させることができるが、小さな円盤の上に大きな円盤を乗せることはできない。

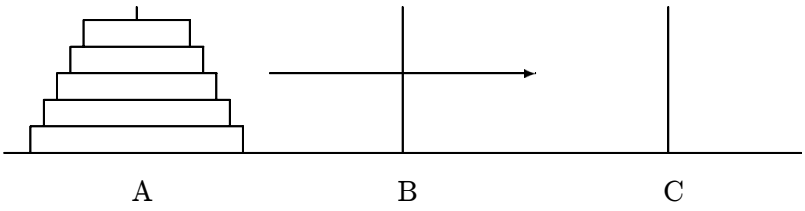

図 5.1: ハノイの塔

伝説　インドのガンジス河の畔のヴァラナシ (ベナレス) に、世界の中心を表すという巨大な寺院がある。そこには青銅の板の上に、長さ 1 キュビット、太さが蜂の体ほどの 3 本のダイヤモンドの針が立てられている。そのうちの 1 本には、天地創造のときに神が 64 枚の純金の円盤を大きい円盤から順に重ねて置いた。これが「ブラフマーの塔」である。司祭たちはそこで、昼夜を通して円盤を別の柱に移し替えている (移し変えのルールの説明は省略)。そして、全ての円盤の移し替えが終わったときに、世界は崩壊し終焉を迎える。【ウィキペディアより抜粋】

例えば、n 枚の円盤を使ってゲームを行うと、最低でも

$$2^n - 1 \text{ 回}$$

円盤を移動する必要があることが知られています (証明してみよう)。したがって、伝説のように 64 枚の円盤では $2^{64} - 1 = 18,446,744,073,709,551,615$ (1844 京 6744 兆 737 億 955 万 1615) 回も円盤を移す必要が出てきます。仮に、1 秒間に 1 枚の円盤を動かすことができたとしても、約 5,845 億年かかってしまいます。なお、現在の高速なコンピュータ (例えばクロック数が 4GHz で 8 コアの CPU を 1 個搭載したコンピュータ; 並列処理が可能で 1 枚の円盤を動かすのに 10 命令必要だと仮定する; その他のオーバーヘッドは考えない) を使っても、約 183 億年かかってしまい、たとえコンピュータが壊れなかったとしても私たちが生きている間にゲームを終わらせることはできません[4]。このように、このゲームには膨大な時間がかかるため、ゲームが終わる頃には世界が滅亡してしまうという実しやかな伝説となっています。

[4] 日本が誇るスーパーコンピュータ「京」だと、1 秒間に 1 京 ($= 10^{16}$) 回の演算が可能だから、1 枚の円盤を動かすのに 10 命令必要 (並列処理が可能で、その他のオーバーヘッドは考えない) だと仮定すると、約 5 時間ほどで終了することができる。

このゲームを最短で終わらせるための手順 (アルゴリズム) を考えて見ましょう。まず、図 5.2 のように大きく 3 つのステップに手順を分けます。

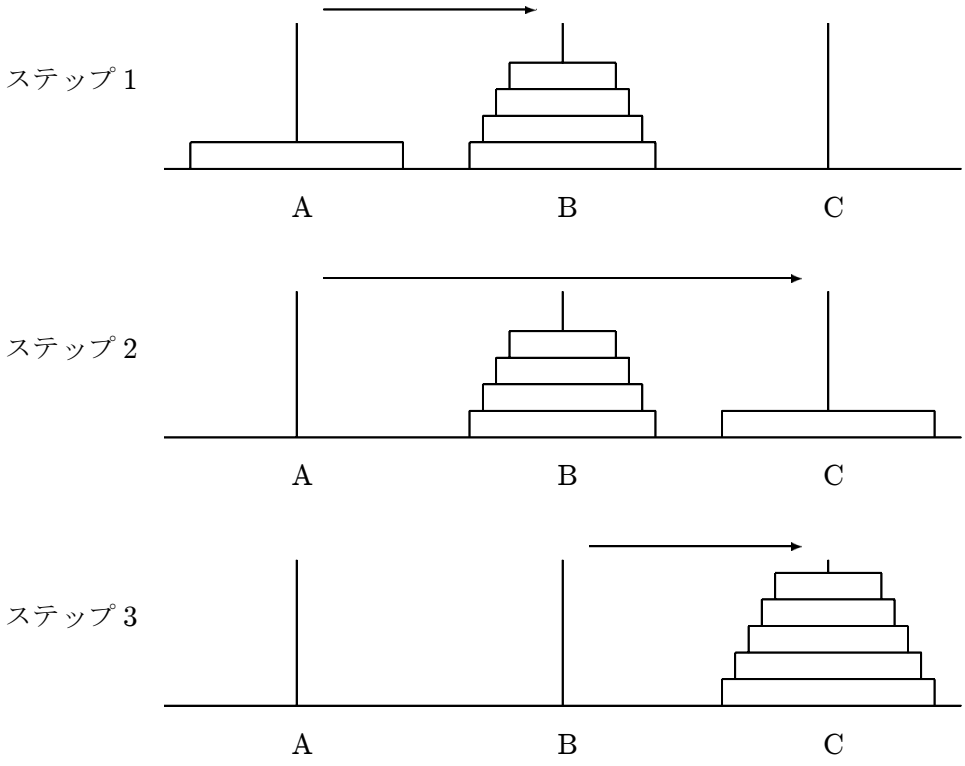

図 5.2: ハノイの塔のアルゴリズム

このとき、ステップ 1 とステップ 3 は、棒の位置は違いますが、$n-1$ 枚の円盤を移すハノイの塔のゲームに帰着できることがわかります。実際、ステップ 1 とステップ 3 を図 5.3 のように書き換えてみれば、その様子がよくわかります (ステップ 2 は円盤を動かす操作)。

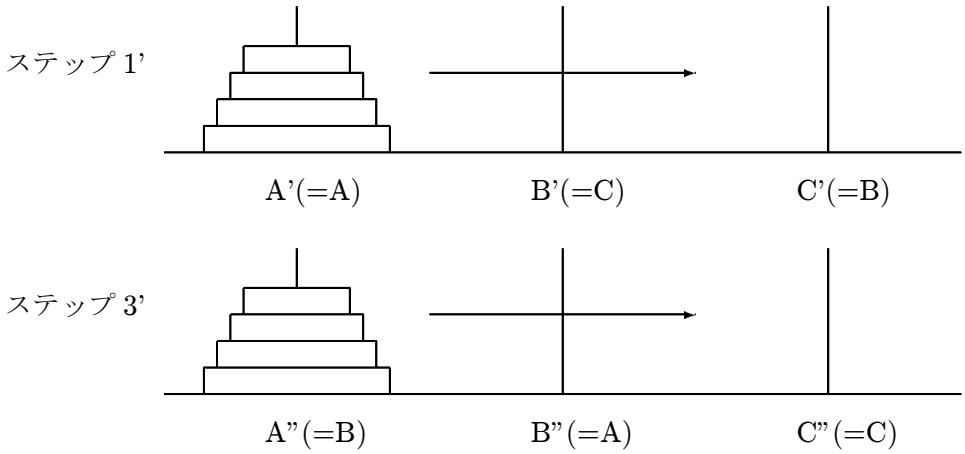

図 5.3: 帰着されたハノイの塔

このように、手順の中で同じ手順を繰り返し行うようなアルゴリズムを**再帰的アルゴリズム**と呼び、プログラミングにおいても**再帰的プログラム**の例として必ず取り上げられます。図 5.4 は、4 枚の円盤を最短の手順で移動させる様子を再帰的アルゴリズムで模式的に表したものです (これまで例として挙げてきた 5 枚の円盤の場合はもう 1 段増えることに注意してください)。なお、白い箱 □ は再帰的に呼び出される図 5.2 のステップ 1 とステップ 3 を表し、黒い箱 ■ は図 5.2 のステップ 2 を表しています。特に、黒い箱 ■ は、実際に円盤を動かす操作で、$2^4 - 1$ $(= 15)$ 回であることも確認することができます。

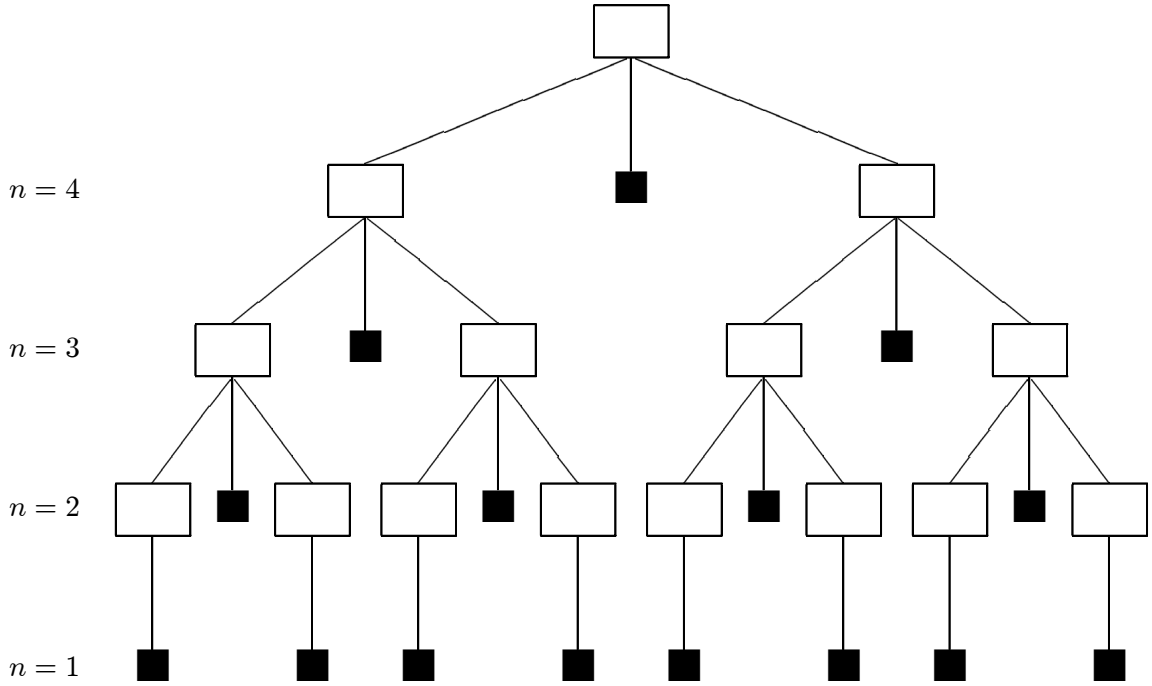

図 5.4: 再帰的アルゴリズムの展開 (ハノイの塔)

ただし、再帰的アルゴリズムは非常に便利な反面、図 5.4 からもわかるように階層を作りながらプログラムを進めていきます。そのため、実際のプログラムでは上位層の情報を記憶しておく必要があり、前節で紹介した**スタック領域**をたくさん使うことになります (階層が増えると指数関数的にスタック領域を使用する)。その結果、オペレーティングシステムから割り当てられたスタック領域を使い切ってしまい、プログラムが正常に実行できないことがあります (アルゴリズムが正しいからといってプログラムとして上手く動くわけではない)。したがって、再帰的アルゴリズムを用いたプログラムを作成する際は注意が必要なことを心に留めておいてください。