

## 特集

## 1 Part 1

アルゴリズムの  
威力を知る

山本哲史



「アルゴリズムなんて知らなくても困らない」内心こう思っている方は意外に多いのではないのでしょうか。確かに、多少のプログラミング経験を持つ方なら、普段、必要なロジックをコードに落とし込むのに困ることはめったにないかもしれません。でも、良いアルゴリズムを知っているのとそうでないのでは、ときとして非常に大きな差が生じるのです。

それは、良いアルゴリズムと悪いアルゴリズムでは、処理速度やメモリー使用量が極端に異なるからです。ここで、「処理速度なんて重要じゃない」とか「最近のパソコンは高速だから…」などと考えるはいけません。アルゴリズムによる処理速度の違いは、いわゆるハードの性能向上やプログラムのチューニングとまるで次元の異なるものだからです。

アルゴリズムによる速度差は、扱うデータの量などが増えるにしたがってどんどん大きくなる場合があります。あるアルゴリズムなら1秒で済む処理が、別のアルゴリズムだと1年かかっても終わらない。実際にそうした例がいくつでもあるのです。

この特集では、ソートや探索などの基本的なものからバズルの解法まで、さまざまなアルゴリズムを紹介いたします。以下Part1では、アルゴリズムを知ることがなぜ重要なのか、いくつか例を挙げな

がら解説します。Part2では、プログラマとして最低限知っておくべきあるアルゴリズムを9個紹介します。アルゴリズムにあまりなじみがない方は、ここでしっかり学んでおきましょう。

一方、教科書に載っている話は聞き飽きたという人向けには、Part3でバズルを解くアルゴリズムを、Part4でゲームの思考ルーチンの作り方を解説します。アルゴリズムの楽しさをじっくり味わってください。

## アルゴリズムがマズイと

## 事実上問題が解けないことがある

アルゴリズムによる処理速度とメモリー使用量の違いについてもう少し詳しく説明しておきましょう。一般に、アルゴリズムが対象とする問題（処理）には、その問題の特徴付ける「サイズ」があります。例えば要素をソート（並べ替え）する処理なら要素の個数がサイズになります。文字列を検索するなら、検索対象の文字列の長さがそのサイズになるでしょう。

プログラムの処理に要する時間や必要なメモリー量は、このサイズが増加すると増えるのが普通です。したがって、サイズをNとしたときにNの関数として記述できるはずですが、現実にはそれを求めるのは不可能に近い。例えば、パソコンの構成や環境などによって変わり得るために正確に求めてもあまり役に立ちません。

そこで、代わりによく使うのがO記法です。O記法は、Nが大きくなったときに、プログラムが大体どのように振る舞うかを表すものです。例えば、処理時間がO(N<sup>2</sup>)であるとは、N

表1 アルゴリズムの違いにより、Nが増えた場合に処理時間が増えていく割合がまったく異なる

アルゴリズム	N=10	N=100	N=1000	N=100万
O(log N)	3.3	6.6	10	20
O(N)	10	100	1000	100万
O(N log N)	33	660	10000	2000万
O(N <sup>2</sup> )	100	10000	100万	1兆
O(N <sup>3</sup> )	1000	100万	10億	100京
O(2 <sup>N</sup> )	~10 <sup>3</sup>	~10 <sup>30</sup>	~10 <sup>300</sup>	~10 <sup>30万</sup>

図1 「安定な結婚の問題 (N = 5の場合)」。0 - 4の“背番号”をつけた男女が集団お見合いをして、それぞれの参加者が相手を好感度によって希望順位を付けた結果

男性	第1希望	第2希望	第3希望	第4希望	第5希望
0	3	2	1	4	0
1	2	1	4	0	3
2	3	1	4	2	0
3	3	4	2	0	1
4	1	2	3	0	4

女性	第1希望	第2希望	第3希望	第4希望	第5希望
0	3	0	2	1	4
1	0	2	4	1	3
2	2	4	0	1	3
3	3	1	0	2	4
4	3	2	1	0	4

### 「安定な結婚の問題」

アルゴリズムによってどれだけ処理時間が変わるのか、実際に調べてみましょう。例として、次のような問題を考えます。

男性N人と女性N人が集団でお見合いをしました。男女それぞれの参加者が、異性の参加者全員を好感度によって順位付けした結果が図1です。参加者は、男女双方とも互いに対する好感度が自分の婚約相手よりも高い、という状況になると浮気してしまいます。図1を基に、浮気する組が一つも生じないようにすべての参加者を結婚させる組み合わせを考えてください。一般に、条件を満たす組み合わせは複数存在しますが、ここでは一つ見つけたいものとします。

これは、「安定な結婚の問題」と呼ばれる昔からよく知られた問題です。単純に考えると、ループですべての組み合わせを生成し、それぞれに対して浮気する組があるかどうかをチェックする、というアルゴリズムが思いつきません。いささか「力任せ」の感がありますが、

が増加するときに、処理時間が大体 $N^2$ に比例することを表します。処理時間が $O(N^2)$ であるアルゴリズムのことを、単に「 $O(N^2)$ のアルゴリズム」と言うこともあります。

ある問題を解く場合、採用するアルゴリズムによって処理時間が $O(\log N)$ 、 $O(N)$ 、 $O(N \log N)$ 、 $O(N^2)$ 、 $O(2^N)$ などさまざまに変わることがあります。 $O(N \log N)$ のアルゴリズムを選んだ場合と $O(2^N)$ を選んだ場合では、Nが大きくなると処理時間の差は気が遠くなるほど大きくなります(表1)。

特に、 $O(2^N)$ の場合、N = 10のときに1000分の1秒で終わる処理でも、N = 100のときには10の27乗秒 = 3000京年かかる計算です。これは解けないのと同じです。

処理に必要なメモリの量についても事情は同じです。パソコンが搭載するメモリの量には限りがありますし、32ビットWindowsでアプリケーションが使用できるアドレス空間は2GBしかありません。効率が悪いアルゴリズムだと、Nがちょっと大きくなるだけで、すぐにメモリ不足になってしまいます。良いアルゴリズムと悪いアルゴリズムの差は、目的とする処理をそもそも実現できるかどうかを決めることだってあるのです。

```

int male_priority[N][N];
int female_priority[N][N];
int male_fiancee[N];
int female_fiancee[N];

int is_stable(void)
{
    int i, j, male_fiancee_priority;
    for(i = 0; i < N; i++){
        male_fiancee_priority =
            male_priority[i][male_fiancee[i]];
        for(j = 0; j < N; j++){
            if(male_priority[i][j] < male_fiancee_priority &&
                female_priority[j][i] <
                    female_priority[j][female_fiancee[j]])
                return FALSE;
        }
    }
    return TRUE;
}

int stable_marriage(void)
{
    int i, j;

    for(i = 0; i < N; i++){
        male_fiancee[i] = 0;
        female_fiancee[i] = -1;
    }
}

```

```

}
/* 組み合わせを系統的に生成する */
i = j = 0;
while(1){
    if(i >= N){
        if(is_stable()) return TRUE;
        j = N;
    }
    else if(female_fiancee[j] == -1){
        male_fiancee[i] = j;
        female_fiancee[j] = i;
        i++;
        j = 0;
    }
    else{
        j++;
    }
    if(j >= N){
        if(i == 0) return FALSE;
        i--;
        j = male_fiancee[i];
        female_fiancee[j] = -1;
        j++;
    }
}
}
}

```

リスト1 「安定な結婚の問題」を、すべての組み合わせについて安定かどうかをチェックして解くコード