競技プログラミングの問題は機械的に解けるか？

Kimiyuki Onaka

2019 年 8 月 4 日

0　概要

この PDF では、競技プログラミングの問題を機械的に解くことについて議論する。

まず 3 章までで競技プログラミングの問題を機械的に解くことについての概要を見る。1 章でいくつかの問題が解かれる過程を分析し、機械によって競技プログラミングの問題を解くことは有望であることを確認する。問題の形式化を取り扱わないことにすれば、AtCoder 水色程度の能力のプログラムができうるだろうというのが結論である。2 章では、機械的に解くことが難しいだろう問題の例を見る。問題自体に再帰が含まれるような複雑な問題 (たとえばゲームに関する問題) などでは困難が大きいことを見る。また、問題の形式化そのものが問われている問題 (たとえば幾何問題) は今回の手法の範囲外であることも確認する。3 章では、自動で問題を解くプログラムを作ることの意義を説明する。ここまでは一般の競技プログラマーに向けての文章である。

4 章では競技プログラミングの問題を解くプログラムの実装のための背景の理論を説明する。これはそのようなプログラムの開発者に向けての文章である。

1　いくつかの問題はプログラムによって解ける

1.1　手作業による機械的な変形で解くことができる問題がある

AtCoder Beginner Contest 100 の 4 問目 Patisserie ABC (`https://atcoder.jp/contests/abc100/tasks/abc100_d`) を例に見てみよう。これは整理すると以下のような問題である。そこまで難しくはないので、自分がどうこの問題を解いているのかを観察しながら一度自分で解いてみて、それから続きを読み進めてほしい。

問題 1.1. *[Patisserie ABC から引用]*

> 高橋君はプロのパティシエになり, *AtCoder Beginner Contest 100* を記念して,「*ABC* 洋菓子店」というお店を開いた.
>
> *ABC* 洋菓子店では, N 種類のケーキを売っている. 各種類のケーキには「綺麗さ」「おいしさ」「人気度」の 3 つの値を持ち, i 種類目のケーキの綺麗さは x_i , おいしさは y_i , 人気度は z_i である. これらの値は 0 以下である可能性もある.
>
> りんごさんは, *ABC* 洋菓子店で M 個のケーキを食べることにした. 彼は次のように, 食べるケーキ

の組み合わせを選ぶことにした.

- 同じ種類のケーキを 2 個以上食べない.
- 上の条件を満たしつつ, (綺麗さの合計の絶対値) + (おいしさの合計の絶対値) + (人気度の合計の絶対値) が最大になるように選ぶ.

このとき, りんごさんが選ぶケーキの (綺麗さの合計の絶対値) + (おいしさの合計の絶対値) + (人気度の合計の絶対値) の最大値を求めなさい.

◇

形式的に書けば以下のようになる。

問題 1.2. *[形式化された問題 1.1]* 長さ N の整数列 $x = (x_0, x_1, \ldots, x_{N-1})$, $y = (y_0, y_1, \ldots, y_{N-1})$, $z = (z_0, z_1, \ldots, z_{N-1})$ と自然数 $M \leq N$ が与えられる。集合 $X \subseteq N = \{0, 1, \ldots, N-1\}$ に対し

$$f(X) = \left|\sum_{i \in X} x_i\right| + \left|\sum_{i \in X} y_i\right| + \left|\sum_{i \in X} z_i\right|$$

と定義する。このとき $\max\{\, f(X) \mid X \subseteq N \wedge |X| = M \,\}$ を求めよ。 ◇

これは機械的に解ける。このとき求める値 y は

$$y = \max\{\, |a| + b \mid \ldots \,\}$$

の形の式で定義されている。$|a| = \max\{\, a, -a \,\}$ であるので、このような式は

$$y = \max\{\, \max\{\, a, -a \,\} + b \mid \ldots \,\}$$

を経由して

$$y = \max\{\, \max\{\, a + b \mid \ldots \,\}, \max\{\, -a + b \mid \ldots \,\} \,\}$$

と変形できる。この変形を可能な限り繰り返すと、求める式は

$$y = \max\left\{ \begin{array}{l} \max\left\{ +\sum_{i \in X} x_i + \sum_{i \in X} y_i + \sum_{i \in X} z_i \;\middle|\; \ldots X \ldots \right\}, \\ \max\left\{ +\sum_{i \in X} x_i + \sum_{i \in X} y_i - \sum_{i \in X} z_i \;\middle|\; \ldots X \ldots \right\}, \\ \max\left\{ +\sum_{i \in X} x_i - \sum_{i \in X} y_i + \sum_{i \in X} z_i \;\middle|\; \ldots X \ldots \right\}, \\ \max\left\{ +\sum_{i \in X} x_i - \sum_{i \in X} y_i - \sum_{i \in X} z_i \;\middle|\; \ldots X \ldots \right\}, \\ \max\left\{ -\sum_{i \in X} x_i + \sum_{i \in X} y_i + \sum_{i \in X} z_i \;\middle|\; \ldots X \ldots \right\}, \\ \max\left\{ -\sum_{i \in X} x_i + \sum_{i \in X} y_i - \sum_{i \in X} z_i \;\middle|\; \ldots X \ldots \right\}, \\ \max\left\{ -\sum_{i \in X} x_i - \sum_{i \in X} y_i + \sum_{i \in X} z_i \;\middle|\; \ldots X \ldots \right\}, \\ \max\left\{ -\sum_{i \in X} x_i - \sum_{i \in X} y_i - \sum_{i \in X} z_i \;\middle|\; \ldots X \ldots \right\} \end{array} \right\}$$

となる。次に絶対値記号が取れた $\max\{\, \pm\sum x_i \pm \sum y_i \pm \sum z_i \mid \ldots X \ldots \,\}$ の形の 8 本の式の計算が必要である。$x'_i = -x_i$ とおけば $-\sum x_i = \sum x'_i$ であるなど、符号の正負は同様に扱える。つまり次の式の計算だけを考えればよい。

$$\max\left\{ \sum_{i \in X} x_i + \sum_{i \in X} y_i + \sum_{i \in X} z_i \;\middle|\; X \subseteq N \wedge |X| = M \right\}$$

$\sum_{\phi(i)} f(i) + \sum_{\psi(i)} g(i)$ は $\phi(i) \leftrightarrow \psi(i)$ のとき $\sum_{\phi(i)} (f(i) + g(i))$ で置き換えられる。$w_i = x_i + y_i + z_i$ とおいて次の式の計算だけを考えればよい。

$$\max\left\{ \sum_{i \in X} w_i \;\middle|\; X \subseteq N \wedge |X| = M \right\}$$

この形の式「N 個のものから $M \leq N$ 個選んだときの重み総和の最大値」は十分典型的でかつこれ以上分解できそうにないものであり、これを $O(NM)$ で解くための変形の残りの部分はすべて埋め込んでおいてよいだろう [*1]。よってこれは計算できる式であり、機械的な変形により y が $O(NM)$ で計算できることが分かった。N, M の上限と計算時間制限が与えられていれば、これが時間内に計算できることも機械的に判定できる。よってこの問題は機械的に解けると言える。

1.2 このような手動での機械的な変形は、プログラムによっても可能である

引き続き Patisserie ABC を例に見ていこう。「このようにして機械的に解ける」と主張した上の説明を分析して裏に隠されたものも明示すれば、さらにいくつかの要素に分解することができる。主には以下のような 3 段階のものとして説明できる。

(1.) 自然言語で入力された問題を形式的な形に変形する
(2.) 適用するべき変換規則を選択し適用する
(3.) 十分に簡単な形に変形された問題を実際の実装として出力する

順番に見ていこう。

1.2.1 プログラムは形式的な入力のみを受け付けることとする

まず (1.) について。これはプログラムによっては扱わず、プログラムの利用者に行わせるものとする。Haskell や ML 系言語 [*2] に似た文法のできるだけ高級な言語を用意し、問題をそのままこの言語の上に翻訳したものを入力させることになる。入力されたプログラムを解釈した結果は、たとえば図 1.2.1 のような構文木として理解できる。

我々は汎用人工知能を作りたいのでも自然言語処理をしたいのでもない。もしいま無理にこれを行おうとしても、たいへんな苦労の後に実用に耐えない精度のものができあがるだけだろう。

1.2.2 変換規則は人間が列挙し、どのような順序で適用するかは機械に探索させる

次に (2.) について。

問題 1.1 を機械的に解く過程で出現した変形は主に以下の 5 種であった [*5]。

- $|a| = \max\{ a, -a \}$ という等式に基づく変換
- $\max\{ \max\{ a, b \} + c \mid \dots \} = \max\{ \max\{ a + c \mid \dots \}, \max\{ b + c \mid \dots \} \}$ という等式に基づく変換
- $-\sum_{i \in X} a_i = \sum_{i \in X} b_i$ という等式に基づく変換 (ただし $b_i = -a_i$ とおく)
- $\sum_{i \in X} a_i + \sum_{i \in X} b_i = \sum_{i \in X} c_i$ という等式に基づく変換 (ただし $c_i = a_i + b_i$ とおく)
- $\max\left\{ \sum_{i \in X} w_i \;\middle|\; X \subseteq N \wedge |X| = M \right\}$ から適切な式 [*6] への変換

変換規則と構文木が与えられたとき「変換が適用できるかを判定すること」および「変換を適用した結果を

[*1] $O(N \log N)$ や $O(N)$ でもよい。
[*2] F# や Standard ML など。
[*5] $a + (b + c) = (a + b) + c$ や、$a = b$ ならば $f(a) = f(b)$ であるといった自明な等式については省略した。
[*6] たとえば列 $w' = \mathrm{reverse}(\mathrm{sort}(w))$ を計算した後に $\sum_{i<M} w'_i$ を計算するような式。

得ること」はプログラムによって可能である。実際、このような規則は図 1.2.2 に示すような木から木への組み換えとして理解できる。

ただしもちろん、これらの変換規則を人の手によって事前に組み込んでおくことが前提である。「$\cdots = \ldots$ という等式に基づく変換」と表記したことから明らかであるように、「競技プログラミングをする上で有用な等式」を「変換の向きと共に」列挙することとなる。これらの変換規則を自動で獲得することは、個々の問題を解くよりはるかに難しい課題であるので行わない。

変換の個々の適用が可能であることだけでは十分でない。考えられる変換はこれらだけではないため「適用する変換を適切に選択できること」もまた必要だからである。この変換の選択の部分は自明ではない。いまのところ式の複雑さが減少するにように貪欲に変換すればたいていの場合で上手くいくだろうと予想しているが、もしそれが偽ならヒューリスティックな探索を行うことになるだろう。

1.2.3　変形結果の式をソースコードとして出力することは難しくない

最後に (3.) について。そもそも操作対象である形式的な数式はそれ自体がある種のソースコードである。これを C++ のような高級言語に変換することは難しくない。

1.3　そのようにして解ける問題の数は少なくはない

実際に AtCoder Beginner Contest の第 100 回から第 104 回までの 20 問[*7]を、その問題が機械的に解けそうであるかを判断[*8]した結果を表 1 に示す。ここから直ちに言えることは次の 2 点だろう。

- 解けるか解けないかは問題との相性の面が大きい
- AtCoder Beginner Contest の C, D 問題が 5 問に 1 問ぐらいは解ける

2　すべての問題が解けるわけではない

形式化との相性が悪い問題がいくらか存在する。次のような問題を解くことは難しい。

1. 形式的に証明を書くことが難しい問題
2. 形式化する能力が問うている問題

2.1　妥当性の説得は簡単であっても、証明として書き下すことが難しい解法がある

競技プログラミングにおいて、解法は必ずしも証明を伴なわない。たとえば貪欲法を用いた解法の証明を数学的に厳密に与えることは難しい。小さい入力の値を表示した結果から帰納的に解法を推測することもある。しかしわれわれの手法は必然的に解法に証明を伴う。真であることが知られている等式を用いて自明な式変形を繰り返すことを基本にしているので「証明はできないがおそらく正しいであろう解法[*9]」は基本的に[*10]生成できない。

[*7] 問題をどこから持ってくるかによっても結果は変わるだろう。

[*8] 主観的な基準で

[*9] 乱択アルゴリズムのことではない

[*10] 実験結果の先頭 n 個を使って OEIS を引くとか、無根拠に貪欲だと決め付けるなどは可能だが、まったく有意義でない

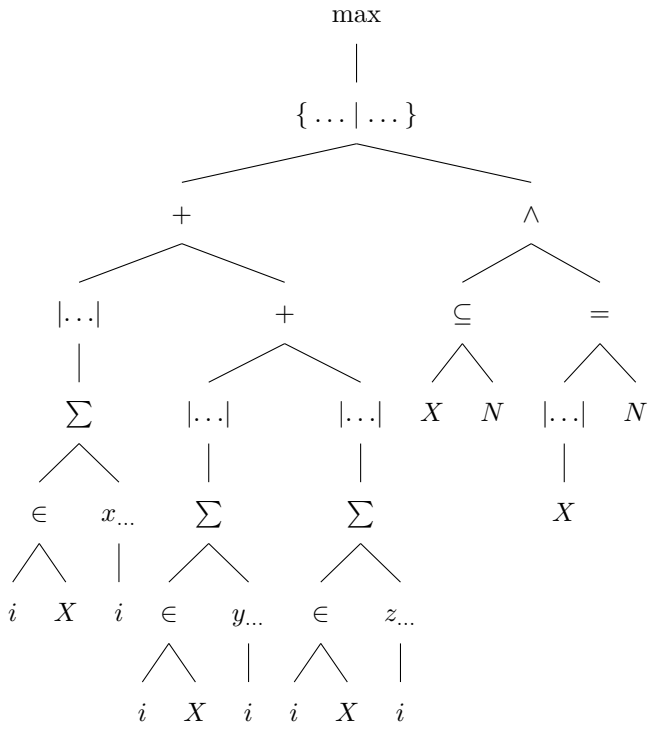

図 1　形式化された問題 1.2 を表現する構文木の一例 [*4]

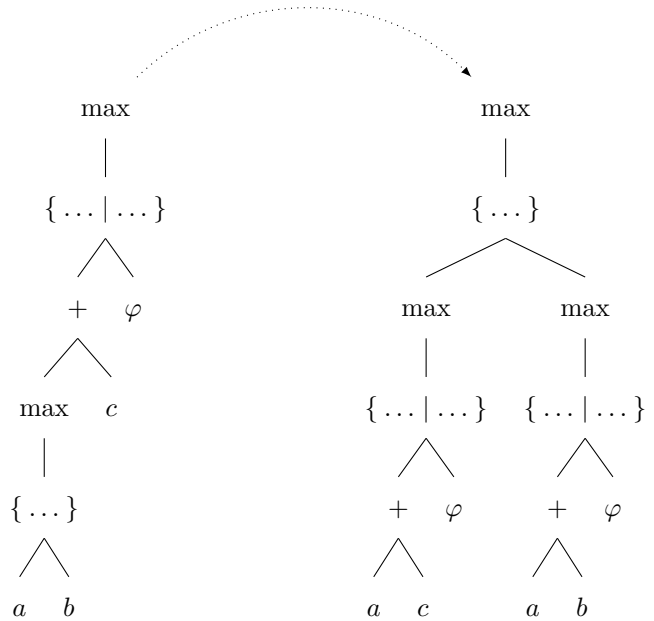

図 2　等式 $\max\{\max\{a,b\}+c \mid \dots\} = \max\{\max\{a+c \mid \dots\}, \max\{b+c \mid \dots\}\}$ に基づく変換規則の、構文木の書き換えとしての表現

AtCoder Beginners Selection の 7 問目 Card Game for Two (`https://atcoder.jp/contests/abs/tasks/abc088_b`) がその例である。これは以下のような問題である。

問題 2.1. *[Card Game for Two から引用]*

> N 枚のカードがあります. i 枚目のカードには, a_i という数が書かれています. *Alice* と *Bob* は, これらのカードを使ってゲームを行います. ゲームでは, *Alice* と *Bob* が交互に 1 枚ずつカードを取っていきます. *Alice* が先にカードを取ります. 2 人がすべてのカードを取ったときゲームは終了し, 取ったカードの数の合計がその人の得点になります. 2 人とも自分の得点を最大化するように最適な戦略を取った時, *Alice* は *Bob* より何点多く取るか求めてください.

◇

この問題は人間にとっては簡単である。単純な貪欲法でよい。その時点で残っているカードの中から最大のものを選ぶことが誰にとっても最適である。つまり列 A を降順にソートし (1 から数えて) 奇数番目の数の総和から偶数番目の数の総和を引いたものが答えである。計算量は $O(N \log N)$ となる。

この問題を機械によって解くにはどうすべきだろうか？ そもそも問題を機械に入力する時点ですでに自明ではない。単純には minimax 法のようにして、以下のように形式化できる。

問題 2.2. *[形式化された問題 2.1]* 整数の多重集合 [*11] $A \in \mathbb{N}^{\mathbb{Z}}$ [*12] に対し関数 $f : \mathbb{N}^{\mathbb{Z}} \to \mathbb{Z}$ を

$$f(A) = \begin{cases} 0 & (A = \emptyset) \\ \max \{\, a - f(A \setminus \{\, a \,\}) \mid a \in A \,\} & (A \neq \emptyset) \end{cases}$$

と定義する。多重集合 A が入力される。値 $f(A)$ を出力せよ。 ◇

このような式は複雑である。式の形だけを見ても (1.) 集合を引数とするような (2.) 再帰関数であって (3.) その漸化式に集合の最大値を得る操作を含むためである[*13]。人間であってもこのような再帰的な式だけから「貪欲に取ればよい」という解法に至るのは手間がかかるはずである[*14]。

そしてこれに対し証明を与えるのはなお難しい。実際に証明を書き下してみると以下のようになる。機械的に解く場合は解法と同時に証明が出力されるようなものであるべきだが、これを自動で導出するのは困難だろう。

問題 *2.1* の貪欲解の正当性の証明. 重複がある場合も以下の議論はほとんど土曜である。簡単のため、入力される重複集合に重複はないとする。つまり問題は有限集合 $A \subseteq_{\text{fin}} \mathbb{Z}$ が与えられるものとみなしてよい。

関数 $g : \{\, A \subseteq \mathbb{Z} \mid A \text{ is finite} \,\} \to \mathbb{Z}$ を

$$g(A) = \begin{cases} 0 & (A = \emptyset) \\ \max A - f(A \setminus \{\, \max A \,\}) & (A \neq \emptyset) \end{cases}$$

[*11] 多重集合とは重複を許す集合のこと。それぞれの要素 x からそれがいくつ含まれているか (重複度) を表す自然数 $m(x) \in \mathbb{N}$ への関数 $m : X \to \mathbb{N}$ のことだとして理解できる。

[*12] Y^X は X から Y への関数全体の集合 $\{\, f \mid f : X \to Y \,\}$ のことである。ただし計算不可能な関数などの面倒は考えないこととする

[*13] 展開すれば N 段の max と min の交互の繰り返しになっている、つまりある種の Π_N 文のような性質を持つと思うと難しさが分かりやすい。

[*14] このことから「この事態は適切な形式化ができていない証拠であり、形式言語の表現力がとぼしいことが原因である」とみることができるかもしれない。

と定義する。任意の A に対し $f(A) = g(A)$ を示せば、貪欲解が最適であると言える。

集合 A の大きさ $|A| \in \mathbb{N}$ に関する帰納法を用いる。

- $|A| \leq 1$ のときは明らか。
- n 未満の場合の成立を仮定し $|A| = n \geq 2$ での場合を示す。$a = \max A$ とする。$b \in A$ であって $b - f(A \setminus \{ b \}) \geq a - f(A \setminus \{ a \})$ なものをとる。$b = a$ を示せば $f(A) = g(A)$ であることが示せる。帰納法の仮定から左辺は $b - f(A \setminus \{ b \}) = b - g(A \setminus \{ b \}) = b - a + g(A \setminus \{ a, b \})$ である。どんな $c \in A$ についても $c - f(A \setminus \{ c \}) \leq f(A)$ であることを使って、右辺は $a - f(A \setminus \{ a \}) \geq a - b + f(A \setminus \{ a, b \}) = a - b + g(A \setminus \{ a, b \})$ である。あわせると $b - a + g(A \setminus \{ a, b \}) \geq a - b + g(A \setminus \{ a, b \})$ が得られる。これを整理すると $b \geq a$ が得られ、つまり $b = a$ である。

□

2.2 形式化する部分が問われている問題には役に立たない

こちらは手法の枠組みに起因する難しさである。「形式的に表現された式を自動で変形しアルゴリズム的な改善をする」ことは「そもそも形式化することが難しい」問題には無力である。

たとえば ICPC 2019 年度国内予選の 5 問目 立方体表面パズル (`https://onlinejudge.u-aizu.ac.jp/challenges/sources/ICPC/Prelim/1636`) を見てみよう。問題文は長いので詳細は該当ページやその図解を直接見てほしいが、簡潔には以下のようなものである。

問題 2.3. 立方体表面パズルとは、次のようなパズルである*:*

- $1 \times 1 \times 1$ の大きさの単位立方体 $n^3 - (n-2)^3$ 個で構成された $n \times n \times n$ の中空の立方体がある。この立方体を 6 分割したもの *(*ピース*)* が与えられる。これを組み立てて元の立方体を復元する遊びが立方体表面パズルである。

さて、立方体表面パズルの 6 個のピースが入力される。ただし必ずしも立方体が復元できるものとは限らない。立方体が復元できるならば復元した結果を、できないならできないと出力せよ。 ◇

実際に解こうとしてみれば、この問題が「アルゴリズムを用いて計算量を落とすこと」でなくむしろ「問題文の内容を正確にプログラミング言語に翻訳すること」を問うていることが分かる。特に、この問題が始めて出題されたコンテストでは実行時間制限は実質 1 時間程度であり、アルゴリズム的な工夫はほとんど不要であった。

3 自動で問題を解くプログラムは有用である

次のような用途が考えられる。

(1.) 自分では解けない問題を解くために用いる。
(2.) 学習のために用いる。
(3.) 考察や実装の仮定で手間を省くために用いる。

この節の議論の多くは「電卓があれば算数は不要か？」「Google などの検索エンジンが使えるのに単語を覚

える必要はあるか？」のような議論と類似していることを注意しておく。

3.1 自分では解けない問題を解くために用いることは難しいだろう

まず (1.) について。ここまで議論してきたような方法で自動で競技プログラミングの問題を解くプログラムを、自分では解けないような問題を解くために用いることは難しいだろう。

原因は次の 3 つである。

(a.) 形式化する作業の難しさ
(b.) 機械的に解ける問題の判定の難しさ
(c.) コンテスト環境への影響

今回の手法では問題を形式化することはユーザの役割である。一般に、なにか与えられたものを正確に読みとってその結果を形式的に (あるいは厳密に) 表現することは、表現される対象と表現のための手法への理解が必要である。そのような理解を持つ人間であれば、プログラムによって機械的に解けてしまうような問題 (つまり比較的単純な問題) の多くを手動で機械的に解くことができるはずである。たとえば $\max\{ f(X) \mid X \subseteq N \wedge |X| = M \}$ のような表現を読み書きできるほど数学に慣れている人間であれば、$|a| = \max\{ a, -a \}$ を使うような簡単な式変形にも同様に慣れているだろう。

プログラムですべての問題が解けるわけではないので、ユーザは「問題が機械的に解けるのはどのような場合であるか」を大まかに判断できる必要がある。機械的に解けうる問題であっても入力の与え方次第では解けないことも多いので、この判断を行なえることはプログラムの利用のために必要である。しかしそのような判断ができるには、やはりユーザ自身の能力が要求される。

コンテスト環境への影響も無視できない。問題を機械的に解くプログラムが十分に普及すれば、機械的には解けない問題のみが出題されるようになることは明らかである。

3.2 競技プログラミングの初心者にとっては、学習のために役に立つだろう

次に (2.) について。競技プログラミングの問題を解くプログラムは「競技プログラミングをする上で有用な等式や定理やデータ構造を集めたもの」であり、また、規則に従って変換を繰り返すことから「問題を解く過程をギャップの小さい変形の列として出力できる」という性質を持つ。これらは学習のためのリソースとして利用できるだろう。

3.3 競技プログラミングの上級者ならば、機械的には解けない問題の考察や実装の道具として用いることができる

最後に (3.) について。これが最も重要であり、主に意図される用途である。

競技プログラミングの簡単な問題を解くプログラムがあったとしても、実際の難しい問題は人が解くしかないだろう。しかしそのような場合でも、難しい部分を人間が解決したあとに帰着されて残る簡単な部分は機械によって処理可能である。つまりわれわれは、問題のより本質的な部分にのみ注力することができるようになる。

単に実装の手間を省くのみならず、考察の手段としても利用することができる。式を入力すればそれをアルゴリズムに改善した結果の計算量を出力してくれるシステムとしても利用できるからである。立てた式の計算

量が瞬時に分かることは、高度なアルゴリズムの設計の効率化に寄与する。

3.4　これらの用途は総合的に、競技プログラミングの界隈のレベルの上昇に寄与する

(1.) により出題される問題の質が上昇し、(2.) と (3.) により多くの競技者の能力が上昇することが期待できる。これらから、機械的に問題を解くプログラムは競技プログラミングの界隈にとって有益であると言える。

4　この機械的な手法はどのような理論に基づくものか

4.1　事前知識として何が必要か

今回の手法は主に数理論理学の、特に証明に関する範囲の知識に基づくものである。これに関する比較的平易な和書として、次の 2 冊をおすすめしておく。

- 寛晰小野. 情報科学における論理. 日本評論社
- 一成照井. コンピュータは数学者になれるのか? -数学基礎論から証明とプログラムの理論へ-. 青土社

実際の開発においては、型システムやコンパイラの実装についての理解も必要である。対象言語の設計には λ計算や集合論の知識も役に立つだろう。これには次の 3 冊を紹介しておく。

- Benjamin C. Pierce, 英二郎住井. 型システム入門 −プログラミング言語と型の理論−. オーム社
- Andrew W. Appel, 靖神林, 宗宏滝本. 最新コンパイラ構成技法. 翔泳社
- Kenneth Kunen. キューネン数学基礎論講義. 日本評論社

アルゴリズムや競技プログラミングについての知識が、ドメイン知識として必要である。データ構造も数学に寄せて扱うので永続データ構造に関する知識もあるとよい。これには次の 2 冊をおすすめしておく。

- 拓哉秋葉, 陽一岩田, 宜稔北川. プログラミングコンテストチャレンジブック [第 2 版] ～問題解決のアルゴリズム活用力とコーディングテクニックを鍛える～. マイナビ, 第 2 版
- Chris Okasaki. 純粋関数型データ構造. KADOKAWA

一方で統計学に関連する知識は不要である。ここまでの議論から明らかなように、いま考えている手法は統計的手法ではない。ビッグデータは用いず、ディープラーニングは行わない、基本的に一切の学習を行わないシステムである。エキスパートシステムという名前で説明されるのが適切だろう。

4.2　推論を形式化した体系としてのシークエント計算を援用する

シークエント計算は単なる形式的な証明の体系としてのみならず、型推論や定理証明支援系のための道具としても利用されている。われわれの手法も、明示的にであれ暗黙的にであれシークエント計算に類似した構造を利用することになるだろう。これは「どのような値について計算が終わっているか」などの情報を保持することが重要であることによる。

「式」や「型」がどのようなものであるかについては、事前に適切に定義をしておくとする。「式 e は型 T を持つ」という言明を $e \in T$ と書くことにする。

定義 4.1. *[シークエント]*「言明の集合 $\Gamma = \{\, e_0 \in T_0, \ldots, e_{n-1} \in T_{n-1} \,\}$ に含まれる式 $e_0, \ldots, e_{n-1}$ の値がす

べて与えられているとき、型 T を持つ式 e が時間計算量 $O(f)$ で計算できる」という言明を $\Gamma \vdash e \in T \cap O(f)$ と書くこととする[*15]。このような形で書かれた言明をシークエントと呼ぶ。 ◇

たとえば「自然数 $K \in \mathbb{N}$ と関数 $f \in \mathbb{Z}^{\mathbb{N}}$ が与えられているとき式 $\sum_{i<n} f(i)$ の値は整数であって計算量 $O(K)$ で計算できる」という言明は $K \in \mathbb{N}, f \in \mathbb{Z}^{\mathbb{N}} \vdash \sum_{i<n} f(i) \in \mathbb{Z} \cap O(K)$ というシークエントで書かれる。あるシークエントが表現する言明が正しいとき、そのシークエントを妥当であるという。

定義 4.2. *[推論規則]*「シークエント $\Gamma \vdash e \in T \cap O(f)$ が妥当であるならば、シークエント $\Gamma' \vdash e' \in T' \cap O(f')$ も妥当である」という言明を

$$\frac{\Gamma \vdash e \in T \cap O(f)}{\Gamma' \vdash e' \in T' \cap O(f')}$$

と書くこととする。ただし条件となるシークエントは複数 *(0 以上の有限個)* あってもよく、その場合は

$$\frac{\Gamma_0 \vdash e_0 \in T_0 \cap O(f_0) \quad \Gamma_1 \vdash e_1 \in T_1 \cap O(f_1) \quad \dots \quad \Gamma_{n-1} e_{n-1} T_{n-1} O(f_{n-1})}{\Gamma' \vdash e' \in T' \cap O(f')}$$

のように書く。このような形で書かれた言明を推論規則と呼ぶ。 ◇

分かりやすさのために推論規則の名前を横棒の右に書いて示すことがある。条件が 0 個の推論規則は、始式あるいは公理とも呼ばれる。ある推論規則が表現する言明が正しいとき、その推論規則は健全であるという。

たとえば「自然数 $n \in \mathbb{N}$ と関数 $f : \mathbb{N} \to \mathbb{Z}$ が与えられているとする。このとき $-\min_{i<n} f(i)$ が $O(N)$ で計算できるならば、$\max_{i<n} -f(i)$ も $O(N)$ で計算できる」という言明は

$$\frac{n \in \mathbb{N}, f \in \mathbb{Z}^{\mathbb{N}} \vdash -\min_{i<n} f(i) \in \mathbb{Z} \cap O(N)}{n \in \mathbb{N}, f \in \mathbb{Z}^{\mathbb{N}} \vdash \max_{i<n} -f(i) \in \mathbb{Z} \cap O(N)}$$

と書かれる。「式 e が $O(f)$ で計算でき、e の計算結果があれば式 e' が $O(f')$ で計算できるならば、全体としては式 e' は $O(f + f')$ で計算できる」という言明は

$$\frac{\Gamma \vdash e \in T \cap O(f) \quad e \in T, \Gamma \vdash e' \in T' \cap O(f')}{\Gamma \vdash e' \in T' \cap O(f + f')}\ \text{cut}$$

と書ける[*16]。

定義 4.3. *[推論木]* ある判断したいシークエントから始めて、推論規則を逆向きに使って木を組み上げることを考える。そのような木を推論木と呼ぶ。推論規則の集合を固定した上で、次のように定義される。

(1.) シークエント S は推論木である。そのような推論木の終式とはシークエント S 自体である。

(2.) 未完成の推論木 $\Delta_0, \dots, \Delta_{n-1}$ に対し、その終式 $S_0, \dots, S_{n-1}$ を前提としシークエント S' を結論とする推論規則があるならば、これらを $\dfrac{\vdots \quad \dots \quad \vdots}{S_0 \quad \dots \quad S_{n-1}}{\scriptstyle S'}$ と繋げて書いたものも推論木である。その終式とはシークエント S' である。

[*15] 言明を正確に表現できさえすれば具体的な記法は重要でないことに注意せよ。たとえば $R(\Gamma, e, T, f)$ や

Γ	e
T	f

と書くということにしても、以下の議論にまったく影響はない

[*16] この形の規則はカット規則と呼ばれる

◇

推論木の定義の (1.) を使わず構成された推論木を完成した推論木と呼ぶ。推論規則が適切であれば完成した推論木の終式は妥当なシークエントであり、推論木を逆から読むと形式的な証明とみなせる。また、そこからプログラムを抽出することができる。

4.3 シークエント計算の利用の具体例

たとえば AtCoder Beginner Contest 134: C - Exception Handling (`https://atcoder.jp/contests/abc134/tasks/abc134_c`) を例に見てみよう。これは次のように形式化できる問題である。

問題 4.1. *[形式化された Exception Handling]* 自然数 N と長さ N の数列 A が与えられる。関数 $f(i) = \max\{\, A(j) \mid j < n \wedge j \neq i \,\}$ の値を $O(N)$ ですべて計算せよ。 ◇

これは終式を $N \in \mathbb{N},\ A \in \mathbb{N}^N \vdash \lambda i.\max\{\, A(j) \mid j < n \wedge j \neq i \,\} \in \mathbb{N}^N \cap O(N)$ とする完成した推論木を構築せよという問題に等しい。次の推論規則が用意されているとする。以下の推論規則はすべて健全である。

- 「式 e が $O(f)$ で計算でき、e の計算結果 a があれば式 e' が $O(f')$ で計算できるならば、全体としては式 e' は $O(f + f')$ で計算できる」

$$\frac{\Gamma \vdash e \in T \cap O(f) \qquad e,\ \Gamma \vdash e' \in T \cap O(f')}{\Gamma \vdash e' \in T \cap O(f + f')}\ \text{cut}$$

- 「$\max\{\, \max\{\, e(j) \mid j < i \,\}, \max\{\, e(j) \mid i < j < n \,\} \,\}$ が計算できるならば $\max\{\, e(j) \mid j < n \wedge j \neq i \,\}$ も計算できる」

$$\frac{\Gamma \vdash \max\{\, \max\{\, e \mid j < i \,\}, \max\{\, e \mid i < j < n \,\} \,\} \in T \cap O(f)}{\Gamma \vdash \max\{\, e \mid j < n \wedge j \neq i \,\} \in T \cap O(f)}\ \text{split-max}$$

- 「自然数 N 未満の自然数 $i < N$ を使って値 $e(i) \in T$ が $O(f)$ で計算できるならば、$O(N)$ で関数 $e : N \to T$ のすべての値を計算できる」

$$\frac{N \in \mathbb{N},\ i \in N, \Gamma \vdash e \in T \cap O(f)}{\Gamma \vdash \lambda i.e \in T \cap O(Nf)} \to \text{右}$$

- 「関数 $A : N \to \mathbb{Z}$ のすべての値が計算されているとき関数 $\lambda i.\max\{\, A(j) \mid j < i \,\}$ は $O(N)$ ですべての値を計算できる」

$$\frac{}{N \in \mathbb{N},\ A \in \mathbb{Z}^N \vdash \lambda i.\max\{\, A(j) \mid j < i \,\} \in \mathbb{Z}^N \cap O(N)}$$

- 「関数 $A : N \to \mathbb{Z}$ のすべての値が計算されているとき関数 $\lambda i.\max\{\, A(j) \mid i < j < N \,\}$ は $O(N)$ ですべての値を計算できる」

$$\frac{}{N \in \mathbb{N},\ A \in \mathbb{Z}^N \vdash \lambda i.\max\{\, A(j) \mid i < j < N \,\} \in \mathbb{Z}^N \cap O(N)}$$

- 「$\max\{\, l(i), r(i) \,\}$ のような自明な式は $O(1)$ で計算できる」

$$\frac{}{i \in N,\ l = \ldots,\ r = \ldots,\ \Gamma \vdash \max\{\, l(i), r(i) \,\} \in \mathbb{Z} \cap O(1)}$$

$\Gamma = \{\, N \in \mathbb{N}, A \in \mathbb{N}^N \,\}$ と書くことにし、これらの推論規則を使えば問題 4.1 のシークエントを終式とす

る完成した推論木が図 4.3 のように書ける。そして、これらの推論規則がどのようにして健全であったかに基づいて、推論木からプログラムを復元できる。実際、たとえばカット規則「式 e が $O(f)$ で計算でき、e の計算結果 a があれば式 e' が $O(f')$ で計算できるならば、全体としては式 e' は $O(f + f')$ で計算できる」はつまり「まず e を計算し、それを使って e' を計算する」ようなプログラムの断片を指示している。

表 1　AtCoder Beginner Contest の問題が機械的に解けそうか判断した結果の一覧

コンテスト	問題	機械的に解けうるか	それはなぜか
AtCoder Beginner Contest 100	A - Happy Birthday!	Yes	自明
AtCoder Beginner Contest 101	A - Eating Symbols Easy	Yes	自明
AtCoder Beginner Contest 102	A - Multiple of 2 and N	Yes?	自明ではあるが面倒
AtCoder Beginner Contest 103	A - Task Scheduling Problem	Yes	自明
AtCoder Beginner Contest 104	A - Rated for Me	Yes	自明
AtCoder Beginner Contest 100	B - Ringo's Favorite Numbers	Yes	自明
AtCoder Beginner Contest 101	B - Digit Sums	Yes	自明
AtCoder Beginner Contest 102	B - Maximum Difference	Yes	自明
AtCoder Beginner Contest 103	B - String Rotation	Yes	自明
AtCoder Beginner Contest 104	B - AcCepted	Yes	自明
AtCoder Beginner Contest 100	C - *3 or /2	No	機械的な手法とあまり相性が良くない
AtCoder Beginner Contest 101	C - Minimization	No	機械的な手法とあまり相性が良くない
AtCoder Beginner Contest 102	C - Linear Approximation	Yes?	難しいが可能なはず
AtCoder Beginner Contest 103	C - Modulo Summation	No	剰余は意外と面倒。難しそう
AtCoder Beginner Contest 104	C - All Green	No	貪欲を含むので難しい
AtCoder Beginner Contest 100	D - Patisserie ABC	Yes	機械的な手法が有効に働く問題である
AtCoder Beginner Contest 101	D - Snuke Numbers	No	機械的な手法と明らかに相性が悪い
AtCoder Beginner Contest 102	D - Equal Cut	No	単純に問題が難しい
AtCoder Beginner Contest 103	D - Islands War	No	形式化の形次第だが難しそう
AtCoder Beginner Contest 104	D - We Love ABC	Yes?	難しいが可能なはず

$$\dfrac{\dfrac{}{\Gamma \vdash \lambda i.\max\{\, A(j) \mid j < i \,\} \in \mathbb{N}^N \cap O(N)} \qquad \dfrac{\dfrac{}{l = \ldots,\ \Gamma \vdash \lambda i.\max\{\, A(j) \mid i < j < N \,\} \in \mathbb{N}^N \cap O(N)} \qquad \dfrac{\dfrac{}{i \in N,\ l = \ldots,\ r = \ldots,\ \Gamma \vdash \max\{\, l(i), r(i) \,\} \in \mathbb{N} \cap O(1)}}{l = \ldots,\ r = \max\{\, A(j) \mid i < j < N \,\},\ \Gamma \vdash \lambda i.\max\{\, l(i), r(i) \,\} \in \mathbb{N}^N \cap O(N)}\ \to \text{右}}{l = \max\{\, A(j) \mid j < i \,\},\ \Gamma \vdash \lambda i.\max\{\, l(i), \max\{\, A(j) \mid i < j < n \,\} \,\} \in \mathbb{N}^N \cap O(N)}\ \text{cut}}{\dfrac{\Gamma \vdash \lambda i.\max\{\, \max\{\, A(j) \mid j < i \,\}, \max\{\, A(j) \mid i < j < N \,\} \,\} \in \mathbb{N}^N \cap O(N)}{\Gamma \vdash \lambda i.\max\{\, A(j) \mid j < N \wedge j \neq i \,\} \in \mathbb{N}^N \cap O(N)}\ \text{split-max}}\ \text{cut}$$

図 3　問題 4.1 に関する推論木

4.4 利用可能な既存のソルバについて

個別的に解ける問題のクラスがいくつか知られている。

(1.) データ構造によるもの
(2.) 線型計画問題
(3.) 燃やす埋める問題
(4.) 正規表現に関連するもの
(5.) 構文解析に関連するもの
(6.) Robinson 算術の式として書かれるもの
(7.) 実閉体の言語の式として書かれるもの
(8.) 数列に関連するもの

(1.) のような、データ構造を利用できる問題はその汎用性から重要である。利用可能であるかどうかの判定が比較的容易であり、他のソルバと違ってユーザが陽に指定しなくても自然に利用しやすいだろう。segment tree や convex hull trick のような汎用的なデータ構造や wavelet matrix や van Emde Boas tree や top tree などの強力なデータ構造がすべて自動で利用されるようにしておくべきである。

(2.) の線型計画問題は有名な問題クラスである。単体法や Karmarkar のアルゴリズムなど解ける。(3.) は競技プログラミングのコミュニティ内で特有の呼称であり、Project Selection Problem とも呼ばれる。整理すると「$x_0 \wedge \cdots \wedge x_{s-1} \to x_s \vee \cdots \vee x_{s+t-1}$ の形の命題論理式と自然数の対の集合 $\{ (\varphi_0, c_0), \ldots, (\varphi_{n-1}, c_{n-1}) \}$ が与えられる。付値 $v : \mathrm{PropVar} \to \{ \top, \bot \}$ に対する関数 $f(v) = \sum_{v(\varphi_i)=\bot} c_i$ を最小化する付値 v を求めよ」という形の問題のクラスに等しい。少なくない問題をこれに帰着させることができ、最大流アルゴリズムで解ける。

(4.) の正規表現は扱いやすい対象のひとつである。たとえば「ある正規表現 γ に受理されるような文字列の個数を数えよ」と等しい問題はしばしば出題され、これは syntactic monoid [8] などを使えば解ける。(5.) はよりやさしい。構文解析については Lex や Yacc のようなツールが整備されており、これを利用することができる。

(6.), (7.) は量化子除去アルゴリズムを用いて自明な形に変換可能である。国立情報学研究所の東ロボくんプロジェクトでも実際に用いられたそうである [9]。

(8.) には OEIS と呼ばれるデータベース (The On-Line Encyclopedia of Integer Sequences®, `https://oeis.org/`) を利用することができる。

参考文献

[1] 寛晰小野. 情報科学における論理. 日本評論社.
[2] 一成照井. コンピュータは数学者になれるのか? -数学基礎論から証明とプログラムの理論へ-. 青土社.
[3] Benjamin C. Pierce, 英二郎住井. 型システム入門 −プログラミング言語と型の理論−. オーム社.
[4] Andrew W. Appel, 靖神林, 宗宏滝本. 最新コンパイラ構成技法. 翔泳社.
[5] Kenneth Kunen. キューネン数学基礎論講義. 日本評論社.
[6] 拓哉秋葉, 陽一岩田, 宜稔北川. プログラミングコンテストチャレンジブック [第 2 版] 〜問題解決のアルゴ

リズム活用力とコーディングテクニックを鍛える〜. マイナビ, 第 2 版.
[7] Chris Okasaki. 純粋関数型データ構造. KADOKAWA.
[8] John E. Hopcroft, Jeffrey D. Ullman, and Rajeev Motwani. オートマトン言語理論 計算論〈1〉. サイエンス社, 第 2 版.
[9] 紀子新井, 竜一郎東中. 人工知能プロジェクト「ロボットは東大に入れるか」: 第三次 AI ブームの到達点と限界. 東京大学出版会.