数学のためのプログラミング入門・学習案内

相転移 P, @phasetrbot[1]

2022-04-30

[1] phasetr@gmail.com

1 はじめに

2 微分積分・線型代数系

3 アルゴリズム系

4 メモ

1 はじめに
2 微分積分・線型代数系
3 アルゴリズム系
4 メモ

注意: 吃音あり

- 吃音 (いわゆるどもり) があるので聞き取りにくい可能性があります
- 適当な形で「筆談」なども併用します
- 講演資料ページへのリンク
  - つどいのスケジュールページにリンクあり

議論のポイント・流れ

- 数学・物理でプログラミングを勉強する
- 学習案内・バッドノウハウ集
  - 「こんなところがつらい」
  - 「ここに地雷が埋まっているから気をつけて」
  - 具体的な本やコンテンツとその選び方案内
- 流れ
  - 導入・いろいろなつらい点の説明
  - 微分積分・線型代数・微分方程式のつらさ
    - 面白いが難しい
  - アルゴリズム学習: 今のお勧め
    - 詳細はリンク先の資料参照

自己紹介

- 学部は物理, 修士は数学
- 職業: (Web 系の) プログラマー
- 趣味で数学と物理をやっている
- 勉強も兼ねてプログラミングで遊びたい
- **どうやって?**
- 何にどう困ったか?
- 困った最果てでいま何をどうしているか?

「宣伝」

- 今回は私の教育関係の活動の「宣伝」
- まさに研究: 色々な人が色々なことを考えてうまくいかずに大半のアイデアが死ぬ
- どう行動してどう死ぬかを見せる
- 自分なりの最善を尽くしてなおうまくいかない
  - 例えば今も試験的に通信講座を展開中
- 後続が二の轍を踏まないように

世間でのプログラミング学習

- 代表的なアプローチ
  - 「アプリやサービスを作ってみよう」
  - 「ゲームを作ってみよう」
  - 「ゲームしながら勉強してみよう」
- どうしても興味が持てない
  - 作りたいモノがあるわけでもない
  - 数学や物理がしたい
- コンピューターは計算機
  - プログラムは計算指示
- 我々は計算し倒せばいいのでは?
  - 微分積分・線型代数・微分方程式・場合の数

とてもつらい

- 「こういうのが子供の頃・学生時代にほしかった」
  - 積分アニメ・波動方程式・スターリングの公式
- 今の自分の腕と環境があればとりあえず簡単なプログラムは書ける
- つらいこと
  - 使える情報を探すのが大変
    - プログラムが腐る・壊れる問題 (後述)
  - 作るのはよくてもメンテナンスが大変
    - cf. ソシャゲのメンテタイムの延長
    - プログラム・システムのメンテナンスは本当に大変

対象設定

- 今回の対象
  - 主に数学・物理系
  - 単純にプログラミングを勉強したい
  - 数学・物理に関わる内容を／で勉強したい
  - 「役に立つ」ことにもつながればいいな
- 既存のプログラミング教育には馴染めない
- 二つの方向性
  - 微分積分・線型代数とアルゴリズム学習
  - アルゴリズム学習は競プロ利用
    - ちょっとした情報科学基礎
    - 純粋な数学学習にも役立つ

応用計算の最果て

- 純粋な数学
  - 数式処理: グレブナー基底
  - 計算数論: 横山俊一さんのサイト, プログラム
  - 微分方程式, 計算機援用解析: 2004 年度解析学賞
- 微分方程式の応用
  - 数値計算ソフトウェア市場
  - Mathematica, 流体系の工学
- 数理工学
  - よいアルゴリズムの研究開発: 機械学習
  - 電車の運賃の高速計算
  - プロ野球のスケジューリング
- これで食べているプロもいる

プログラミングへのスタンス

- プログラミングは証明である
  - そう思える部分に着目する
- 微分積分・線型代数のプログラミングの特徴
  - 多くの大定理を酷使するタイプ
  - 一つ一つの議論 (プログラム) が長め
  - 詳細まで全て把握したい人には修羅の道
- 競プロ的アルゴリズム学習の特徴
  - 入門レベルは徒手空拳 (大定理不要)
  - プログラムも短め: 読み書きが楽
  - 例の確認・反例対策の意味での数学らしさあり
  - まずはこちらをお勧め

スタンスを明確に

- 何がしたいのか?
  - プログラミングの知識・技術の深掘りか?
  - とにかくプログラミングで何かしたいのか?
- イメージ
  - 集合・位相から数学したいのか?
  - 数学を応用して何かしたいのか?
  - 参考
    - 物理: 学部一年で超関数のフーリエ変換
    - 化学: 学部一年前期でベクトル解析からの電磁気
    - 応用化学: 学部一年からシュレディンガー
- 今回は後者向けの話

プログラムが腐る・壊れる

- プログラムは適当なプログラミング言語で書く
- 古文を思い出そう
  - 現代と文法が一致しない
  - 現代と単語の意味も一致しない
  - 現代文の感覚では読めない
- プログラミング言語ではこのサイクルが速い
  - 数年・数ヶ月のプログラムは古文
- 古文が読めない＝プログラムが腐る・壊れる

コンテンツが腐る問題

- 古い本に書かれたプログラムは古文書
  - 古文書の現代語訳が必要
    - 既に使われていない文法や単語が使われているかも?
    - 適切な書き換えが必要
  - 一年前の本でさえ寿命を迎えている可能性あり
  - 事情を知らずに数学の気分で古い本を読むと爆死

腐る原因: プログラムの存在

- 具体的なプログラムを書かなければ腐りにくい
  - 擬似言語で気分だけ示してある
    - もっと言えば抽象的にアルゴリズムだけ書いてある
  - 略証しか載っていない数学書のようなモノ
  - 学習者の視点からは不親切極まりない

ここまでのまとめ

- 具体的なプログラムを書くと腐る
- 抽象的なアルゴリズムだけだと勉強が大変
- どう両立させるか?
  - 競プロ的なアルゴリズム学習
  - 場合の数ベースの具体的な問題演習
  - 書き捨て覚悟で微分積分・線型代数・微分方程式
    - 微分幾何のお絵描きもできればなお楽しい
  - (偏) 微分方程式の解法で学べること
    - 大量のデータ処理
    - 高速計算アルゴリズムの重要性

プログラミングの学習環境

- 学習環境構築 (インストール) も大変
  - cf. TeX のインストールと PDF 生成
  - 「Overleaf を使おう」みたいな話
- Google Colaboratory
  - インストール不要
  - クラウド環境の Python が使える
  - Python は機械学習を含めた数学系のライブラリがたくさんある
  - 何より情報がたくさんある
  - それも新しい情報が
- 言語もとりあえず Python でよい
  - 最新の情報が多い
  - 数学系ユーザーも多い

コンテンツ選びの注意

- アルゴリズムだけ書いてあって実装がない
  - 特に物理や数値シミュレーション (の古い本)
  - 実装があっても古くて動かない
  - または動く環境が (簡単に) 用意できない
- 実装もあって動くが写経が大変
  - 古い本はプログラムが配布されていない
  - コンピューターは融通がきかない
  - cf. TeX
    - 一文字間違えても破滅: 全体が動かない
    - 括弧を抜かしても破滅: 全体が動かない
  - 添字を間違えると計算結果が破滅: 微分方程式
    - 例: 有限差分での熱方程式の失敗

コンテンツ選びの注意2

- 実装はあり，プログラムそのものが配布されていても問題は終わらない
  - 最近はプログラム配布・取得自体もかなり楽
- プログラムが今の (最新の) 環境で動くかわからない
- どう直せばいいかもわからない
- 数学の本の誤植修正ともまた違う
- 正解そのものが変わる
  - 文法・単語の意味自体が変わっているから

コンテンツ選びの注意3

- プログラミング言語の変化
  - 数年・数ヶ月で変わる
  - 一年前に正しく動いたプログラムでも動かない
  - 自然言語と同じ
- 対して数学
  - 昔書いた正しい証明は正しいまま
    - もちろん読みにくいかもしれないが
  - 他の分野によくある (らしい)「学説の更新」とも違う
  - このギャップを認識しよう

補足: 高速化のご利益1

- cf. 組み合わせ爆発お姉さん問題
- 速くすると何が嬉しいか?
- 大量のデータを処理する場面がある
  - いわゆるビッグデータの処理
  - スパコン利用での微分方程式の計算
- 「10倍速くなりました」
  - どのくらいのインパクトかイメージつくか?
  - スパコンレベルの計算が必要な前提
  - よいアルゴリズムを組む意義

補足: 高速化のご利益2

- 0.1 秒か 1 秒か?
  - 大した違いではない
- 1 時間か 10 時間か?
  - 一日の中で何度か試せるか一日待つか?
  - 帰り際に計算を仕込む
  - 朝来てみたら途中で失敗していた
- 1 日か 10 日か?
  - 毎朝のルーティンで結果チェックできるか?
- これがアルゴリズムの改善の意義

1 はじめに

2 微分積分・線型代数系

3 アルゴリズム系

4 メモ

何を計算するか?

- 教養数学・物理ネタ
  - 指導教員層に浸透させたい
  - (同年代になりつつある) 新人教員世代に定着させたい
  - 非物理の人向けの物理教育, 微分方程式の数値計算・シミュレーション
    - 方程式の解が確かに現象を近似する
    - 厳密解以外の方法で視覚的に表す
- いろいろな図示
  - 変な関数の挙動
  - 近似の挙動: cf. スターリングの公式
  - 微分方程式の挙動・数値実験

目で見える世界をきちんと描く

- 原理的に 1-3 次元までしか描けない
- 適切に使えば／使えれば効果は大きい
  - アニメーションさえ簡単に作れる
  - 区分求積法などは顕著にわかりやすくなる
    - 高校の頃本当にほしかった
- 何より楽しい
  - 大学受験の図形問題遊びもいいかもしれない
  - 微分幾何お絵描き・PDE シミュレーション

統計勉強会での一幕

- スターリングの公式 $n! \sim \left(\frac{n}{e}\right)^n$
  - 近似の精度が想像以上によい
  - グラフを描くと一目瞭然
  - 「学部一年から知ってはいたが改めて見ると趣がある」
  - お絵描きだけでも楽しい
- 数値的・視覚的検証の意義
  - cf. Wikipedia
  - スターリングの公式は小さい N からかなりの精度を持つ.
  - 自力での計算はつらいのでとても便利

手計算の代替

Proposition

三次元のレビ-チビタテンソルは次の関係式をみたす.

$$\sum_{k=1}^{3} \varepsilon_{ijk}\varepsilon_{lmk} = \delta_{il}\delta_{jm} - \delta_{im}\delta_{jl}. \tag{1}$$

- ベクトル積に関する議論で出てくる
- 力学・電磁気・流体力学でも登場

証明の難しさを計算でカバー

- (多分) よくある証明は結構大変
  - かなりの慣れが必要
- 全ての i, j, k, l, m を具体的に突っ込んで計算してもよい
  - $3^5 = 243$ 通り
  - 手計算では大変
  - わざわざ全部挙げたくない
  - プログラムに計算してもらおう
- 一種の計算機援用証明
  - 数学の証明の代替にしてもいいが, プログラムの演習問題にしてもいい

微分方程式・計算量・アルゴリズム

- 次元が上がると計算量がひどくなる
- 微分の差分近似

$$\frac{df(t)}{dt} \simeq \frac{f(t+\Delta t) - f(t)}{\Delta t}. \tag{2}$$

- 時間発展: 時間 $I = [0, 1]$ とし, $\Delta t = 0.01$ で $t = 0$ から順次計算
- 偏微分方程式では空間領域も同じように分割

データ量の感覚

- 一次元 (常微分方程式)
  - $[0,1]$ を 100 分割: それなりによくわかる
- 三次元の時間発展: 波動方程式
  - 空間領域 $[0,1]^3$ で時間 $[0,1]$ の発展を追う
  - 時空ともに 100 分割すると?
    - 各 x, y, z, t を全て 100 分割
  - データ量は $100^4 = 10^8 \to$ 10 億=100M
- スパコン計算のデータ規模: 参考
  - 2013 年, 5 億-7 億原子規模
  - 高速計算用のアルゴリズムを使わないと研究が終わらない

参考: 微分方程式の解法1

- 非線型の方程式でも線型代数に帰着
- 出てくる行列の次数
  - だいたい空間データ量と一致
    - 先の例では 100 分割で $10^6 = 100$ 万
  - 100 万次行列の処理
- 高度な効率化・高速化が必要
  - 高性能なコンピューター (ハード) の開発
  - よい数値計算アルゴリズム (ソフト) の開発
- 微分積分・線型代数でもアルゴリズムは大事
  - 一般次元の線型代数が必要

補足: PDEの近似についていろいろ

- 適切な近似・良い近似と良いアルゴリズム
  - 対象に応じて変わる
  - 物理がわかっていないと良い近似・良いアルゴリズムもわからない
- 物理・シミュレーションでのポイント
  - 物理的にリーズナブルな仮定に基づく近似
  - 具体的な問題でのデータに基づく仮定・近似
    - 微分作用素の近似行列はほとんどの要素が 0: 疎行列
    - アルゴリズム・データ構造と直結
- 例: 多様体の線型近似問題
  - リー群: リー環 (単位元の接空間)
  - 一般の多様体: 接束・余接束・テンソル積束
    - ジェット束: 高階微分を見る

数学・物理系の本の紹介

- Python 本
  - 新しめの「Python で数学」系の本
    - 機械学習と絡めて探すとたくさん見つかる
    - 古いとプログラムが腐っているので勧められない
  - Anders, Elementary Mechanics Using Python
  - プログラミングで数学を 中高数学虎の穴
    - 案内ページにいろいろ書いたので勉強の参考に
  - Geometry for Programmers
    - 名前の通り既にある程度プログラミングできる人向け
- github, phasetr/mathcodes
  - いろいろな数値計算コードが置いてある

その他

- Walck, Functional Programming for Physics Geeks
  - Learn Physics by Programming in Haskell
  - Haskell なので結構大変
- Functional Differential Geometry
  - オリジナルは Scheme
  - Clojure 版 (のライブラリ) あり: sicmutils
  - 参考, 開発は活発
- Mathematica での微分幾何みたいな本もある

1 はじめに

2 微分積分・線型代数系

3 アルゴリズム系

4 メモ

競技プログラミングで遊ぶ

- 「作りたいモノがない」
- 「微分積分・線型代数の計算は難しすぎる」
  - プログラミングの技術として厳しい
  - 微分方程式だと物理の知見もいる
  - グラフ・アニメーション部分もまたつらい
  - 書いたコードがすぐ腐る
- いろいろな問題が絡むからつらい
  - 数学・物理・情報科学の総合格闘技
  - 解きほぐして単純なところからはじめよう
- 情報科学の知見に突撃してみよう
  - アルゴリズムとデータ構造
  - 特に競技プログラミング (競プロ)

何を計算するか

- 特に競技プログラミングに絞って
  - 場合の数・数え上げ
  - (初等) 数論
    - 素数判定
    - 約数列挙
    - 素因数分解
    - 約数の個数
    - オイラー関数
- 特に場合の数が主戦場
  - 約数列挙などはまさに数え上げ

競技プログラミングとは何か

- クイズの早解き選手権
- 提出する解答がプログラム
- 過去問が公開されている
  - 選手権に出なくても過去問演習はいくらでもできる
  - 他の人の解答も公開されている
  - これを見ながら勉強しよう

現実的な例

- 競プロの背後にあるアルゴリズムの世界
- 凄まじい量の場合の数からほしいモノを見つけたい
  - 鉄道運賃計算と最適経路探索
  - オペレーションズリサーチ
    - 野球のスケジューリング
    - ドームのスケジューリング
  - etc
- これの簡単な場合が競技プログラミングのネタ

例: 鉄道運賃計算と最適経路探索

- 東京の鉄道・路線図を想定
  - 乗車駅と降車駅しかわからない
  - 途中で変な経路を通ってきているかもしれない
    - いくらでも変な経路は考えられる
  - 運賃はいくらにすべきか?
  - 安い経路を通ったのに高い値段ではまずい
  - ピッとタッチした瞬間に計算が終わってほしい
- 最適化問題として定式化
  - グラフ上の適当な極値問題

スケジューリング

- 野球のゲームマッチ
  - 球場 (特にドーム) をどうおさえるか?
  - 球場はプロ野球をするだけではない
    - コンサート・高校野球などのイベント
    - 関係者多数
  - 移動の都合
    - あるチームが北海道から九州に移動するようなハードな旅程ばかりになってもいけない
    - 各チームの旅費・宿泊費問題
- 最適化
  - 適当な評価基準を作って計算してみる
  - 評価基準自体も調整したい
  - 高速化できればいろいろ試せる

問題の特徴

- 尋常ではないパラメータ数
  - 球場の数: 10 個
  - シーズン: 100 日
  - チーム数: 12
  - 野球以外のイベント主催団体: 数百
  - これらの組み合わせ: 階乗レベルで効いてくる
  - 階乗のオーダーは指数オーダー
- 数学的には高々有限だがそういう話ではない
- 時間はかけたくない・かけられない
- 究極的には場合の数

お勧めの勉強方針1

- Project Euler
  - 名前からしてもより数学チック
  - 勉強の観点からは**勧められない**
  - 解答が読めないから
- AtCoder と AOJ
  - これがお勧め
  - どちらも解答が読める

お勧めの勉強方針2

- まずはプログラムを読めるようになろう
- cf. 語学学習
  - 英語が読めても書けない・話せない
    - そういう訓練をしていないからでもある
  - 証明が読めても自力で構成できない
- プログラミングも同じ
  - 読めるプログラムを自力で書けない
  - まずは読めるようになろう
  - 読めたプログラムを少しずついじろう

プログラミング上の負荷の低さ

- 各問題で書くプログラムの行数も短い
  - 言語によるが数行から十数行
- 難しい言語機能も使わない
  - 微分積分・線型代数は可視化も必要
  - 必要なプログラミング上の知識・技術は広く深い
  - ライブラリも含めれば万単位のプログラムになりうる

コンテンツ紹介: 初等数論

- AtCoder 版！マスター・オブ・整数 (素因数分解編)
- 中高数学駆け込み寺 素数判定の巻 現代の新たな遊びの形を探る
  - ここにも学習案内あり
  - 上記 AtCoder 版含めて数学科向けの記述が足りない
  - もっと計算例や物の見方の説明が必要
- アルゴ式
- 素数夜曲, 参考: 私の GitHub
  - Scheme を勧めるのはどうかという気分もある

アルゴリズム・競プロ系

- 一般的なアルゴリズムの本
- 競技プログラミング系の本
- 競プロ系サイト
  - AtCoder 本体, AtCoder C++入門
  - AOJ
  - アルゴ式
  - Project Euler
- パズルで鍛えるアルゴリズム力: 未読
  - 数独や編集距離の問題
- phasetr/AlgorithmsAndDataStructureByFSharp
  - 私のアルゴリズム・データ構造の学習ログ
  - F#, Haskell メインであまり勧められない

雑多な本・コンテンツ

- 京大 OCW, 計算数論入門
- プログラミングしながらゲームを進める (やったことなし)
  - 【2022 年最新】簡単にプログラミングが学べるゲーム 10 選！
  - JavaScript を打ち込んで対戦, Screeps: Arena
- その他調べるといろいろある

計算機科学入門

- Sannella, Fourman, Peng, Wadler, Introduction to Computation Haskell, Logic and Automata
- アンダースタンディングコンピュテーション
  - Ruby で書かれている

まとめ

- 微分積分・線型代数は難しい
  - だがいいスパイスではある
  - プログラミング的には割り切りつつ遊ぶと楽しい
  - 大量の計算が出てくるので高速化が大事
  - アルゴリズムとデータ構造の意義を横目で学ぶ
- アルゴリズムとデータ構造
  - プログラミング的にはシンプル
  - これでプログラミングに慣れると良さそう

1 はじめに

2 微分積分・線型代数系

3 アルゴリズム系

4 メモ

言語の更新で処理が変わる例

- ツイート URL
- (ドイツ語の)lasst と laSSt は同じ単語と判定するか?

何が大変か

- 微分積分・線型代数
  - ふつうライブラリを使う
    - ライブラリは Fact または大定理
    - 中身も読めるが難しい: 認めて進もう
  - このライブラリが「腐る・壊れる」
    - 古い本に書いてあるプログラムは動かない
    - 壊れてもまず自力で直せない
    - 自力で書くのも大変
    - 壊れたら自分のプログラムを都度調整
- アルゴリズム系
  - バカみたいな量をさばく
    - 組み合わせ爆発お姉さん問題
    - 微分積分・線型代数でも同じ

アルゴリズム系を勧めたい

- 微分積分・線型代数系はプログラムの蓄積が難しい
  - すぐ腐る・壊れる
  - 数値計算でもアルゴリズムは必要で大事
- アルゴリズム系: 特に競技プログラミング
  - 競プロ: 与えられた問題を解くプログラムを速く正確に書く
  - 「証明を書く」テイストで楽しめる
  - 反例で証明の穴を突かれる体験
    - cf. エッジケース・コーナーケース
  - 変なプログラム (証明) を落とす例が仕込まれている

二つの処方箋

- 言語の更新に合わせてプログラムをメンテし続ける
  - できるだけやりたくない
    - プログラムが書きたいわけではない
  - とてもつらい
  - 微分積分・線型代数系はこれとの戦い
    - ライブラリを使う (使わざるを得ない) から
    - 独学の素人にはお勧めしにくい
- 言語の変化によらないプログラムを書く
  - アルゴリズム系
  - 今回の最終的なお勧め
  - 「ちょっとやってみたい」程度ならこちら

目で見える世界をきちんと描く

- 変な図を描くと引っ張られる
  - cf. 平面幾何での問題
    - 小旅行 3 目で見える世界に頼ることの危うさ
    - 「全ての三角形は二等辺三角形である」
- 適切に使えば／使えれば効果は大きい
  - アニメーションさえ簡単に作れる
  - 区分求積法などは顕著にわかりやすくなる
    - 高校の頃本当にほしかった
- 何より楽しい
  - 大学受験の図形問題遊びもいいかもしれない
  - 微分幾何お絵描き・PDE シミュレーション

参考: 微分方程式の解法2

- きちんと考えないと保存系でエネルギーが保存しない
- エネルギーが保存する数値計算法がほしい
- なぜエネルギーを保存させたいのか?
  - 物理のモチベーションがないとわからない
- シンプレクティック数値積分法
  - 必ずしも難しいわけではないがすぐに細かく面倒な話になりがち

補足: プログラミング言語のアップデート

- セキュリティの問題
  - いわゆる脆弱性の解決
  - ふつうはアップデートしないとまずい
- プログラム実行速度の問題
  - 一般に新バージョンの方が速い
    - プログラムを書き換えなくても処理系が速くなる
    - cf. 「英語力があがって昔は辞書・文法書を引きながらでないと読めなかった英文がすらすら読めるようになった」
  - 速度問題は大事: cf. 組み合わせ爆発お姉さん問題

補足: ライブラリとは何か?

- 数学で言えばお役立ち補題
  - 大定理レベルのすごいのもたくさんある
  - 「証明は知らないが使い倒す」系の命題・理論
- 微分積分・線型代数で重要
  - (やるとわかるが) 実装が死ぬ程面倒
  - 計算するべき量が尋常ではなく高速化したい
  - 専門家が日々ブラッシュアップしている
- 教育・勉強用: お絵描き・アニメーション
  - これも専用のツールやライブラリがある
- バージョンアップでよく「壊れる」
  - 特に Python

補足: 図示できない世界をどう図示するか

- 統計学では現実的な問題
- 例: 病気の統計分析での数十の因子
  - 何がどう影響しているかどう調べるか
  - 因子間の相関の具合をどう調べるか
  - 数値だけで見ていてもわからない
- 平面に三次元以上の情報を盛る
  - 点の大きさを変える
  - 点の色を変える
- cf. 変なグラフによる「統計のウソ」

参考: コンピューターの資源

- 昔のゲームソフトの容量
  - スーパーマリオブラザーズ=40KB
  - 今では PDF 一つも作れない
- 今は解くべき問題も巨大化している
  - 小さい問題なら個人の創意工夫が強く効く
  - 大きい問題はチーム戦
  - 戦い方も変わる
- 資源に関する根本的な事情は変わらない

具体例

- なかなかいい例を作るのが難しい
- 数学版
  - 空集合は連結か?
  - 0^0 は何か?
  - 部分環ともとの環の単位元は一致するか?
- プログラミング
  - 数のリストを考える: $[1, 2, 3, 4]$
  - 空リスト (空集合) の先頭要素は何?
    - `List.head` の返り値
    - 典型的には例外 (エラー) で処理終了
    - エラーにしない対処もある: `List.tryHead`

数学教育上のご利益

- 手を動かす訓練
  - 一般論ばかりではなく具体的な計算対応も大事
  - cf. 先程のレビ-チビタ
- アルゴリズムは証明・エッジケースは反例
  - 証明を書く訓練
  - 反例の構成
  - 変な例でも動くか?
    - 条件をみたすモノがないときに正しく「非存在」を返すプログラムが書けるか?
  - 「0 割り算やってない?」
    - 要素が 0 個のとき, 1 つのときでも動くか?

Project Eulerから

- Problem 18, Maximum path sum I
  - **15** 段の三角形型経路に並んだ数の和の最大値
  - 16384 経路
  - 既に手計算で処理できるレベルではない
  - コンピュータなら腕力で処理できる
- Problem 67, Maximum path sum II
  - **100** 段の三角形型経路に並んだ数の和の最大値
  - 経路数は 10^{12}
  - 高々 100 段でここまで膨れ上がる
  - どう効率化・高速化するか?

プログラミングと数学的訓練

- 素数判定・素因数分解のプログラムを書こう
  - 後者はかなり難しい
  - バグを仕込まずきちんと書くのは本当に大変
- プログラムがきちんと動かない=証明がおかしい
  - プログラムのどこがおかしいかを見抜く=証明の粗探し
  - プログラミングではデバッグと言う
- 具体例で確認しながら進める
  - テスト駆動開発・ふるまい駆動開発

アルゴリズムと競プロ

- 競プロの視点・気分: **証明の提出**
- どんな視点から証明をチェックするか?
  - いくつかの具体例で成り立つ
  - きわどいケースでも成り立つ: 「証明の反例」
    - cf. エッジケース・コーナーケース
  - データが大量にあっても短時間に処理しきれるか?
    - 組み合わせ爆発お姉さん問題
- 単に正しければいいだけではない
  - これを面白いと捉えられるかどうか
  - やってみると結構楽しい (と思う人もいる)