タイムラインを活用した、
プログラムの動作を可視化する学習システムの開発

開発駆動コース 川合ゼミ 2023年度 丸山拓真(34Dk)
2024年1月 第5回イベント

1 自己紹介

丸山 拓真

Maruyama Takuma

大学1年生。19さい
Sechack365 2023年 開発駆動

2 プロジェクト概要

こんなことありませんか・・・？

何が原因かよくわからない論理エラーに苦しむ
気合でprintデバッグをしてしまう

突然出てくるSegmentation fault (セグメンテーション違反)
の特定に苦慮

過去の自分が書いたとんでもソースコードと格闘する
昨日の自分は赤の他人

2 プロジェクト概要

こんなことありませんか・・・？

教科書・技術本に乗っている不思議なアルゴリズム
まるで魔法で動いていそう

眠気と格闘しながら頑張って書いたプログラム
全然動かない（そんな時に限って締め切りが近い）

数時間かけて原因調査
結局原因は添え字のｉとｊの書き間違え

2 プロジェクト概要

プログラム入門者ならなおさら・・・

わけがわからないよ～！

2 プロジェクト概要

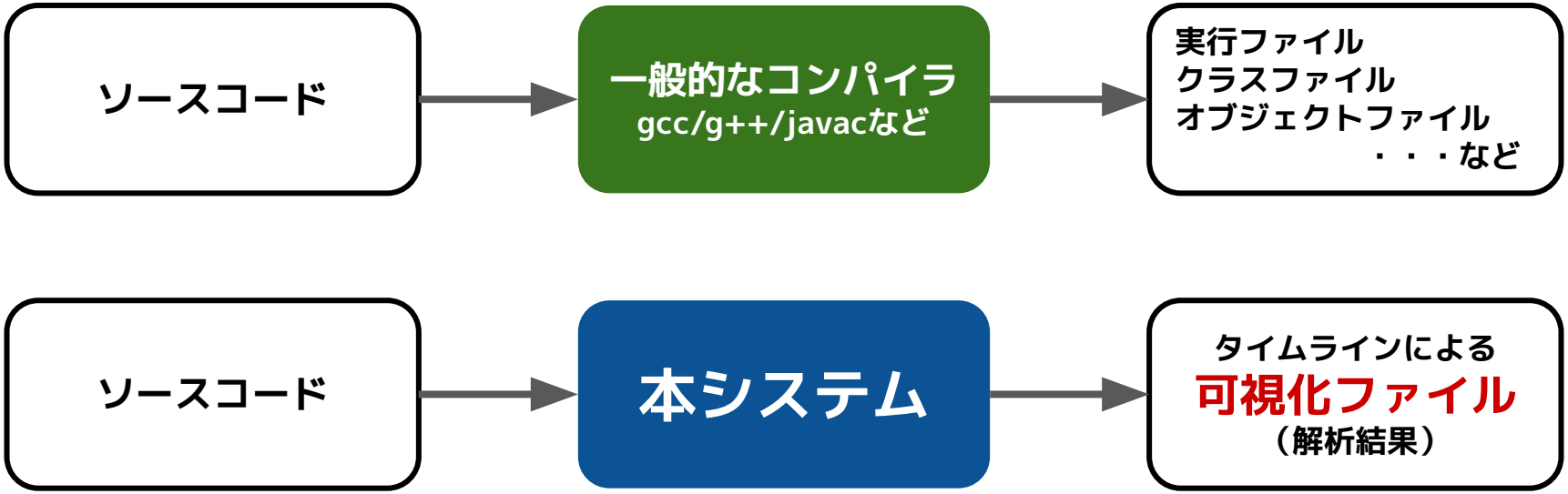
そんな長年の問題（？）を
一歩解決するツールを作った

2 プロジェクト概要

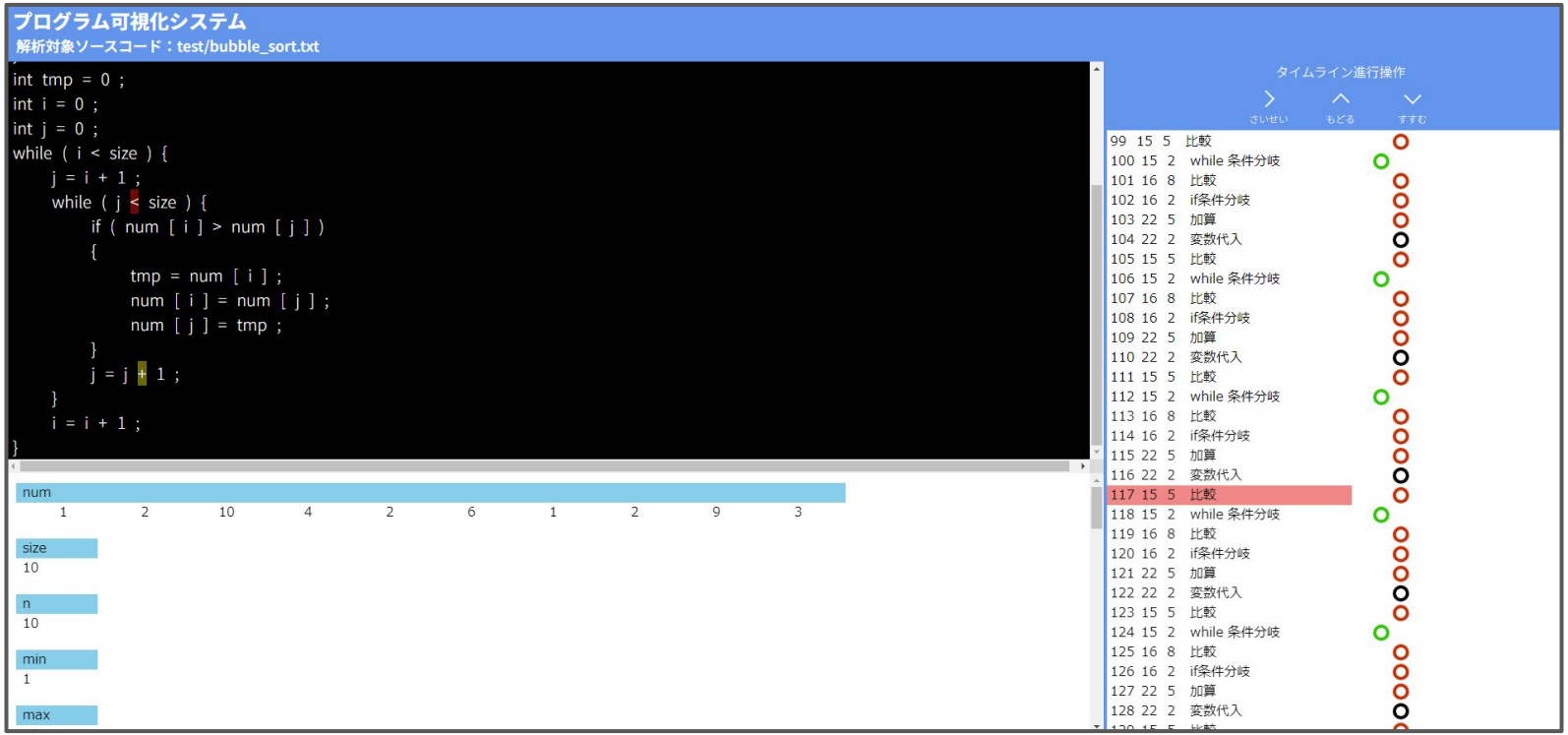
タイムラインを活用した
プログラムの動作を可視化する学習システム

プログラムの動作
ロジック・アルゴリズムの実行を可視化

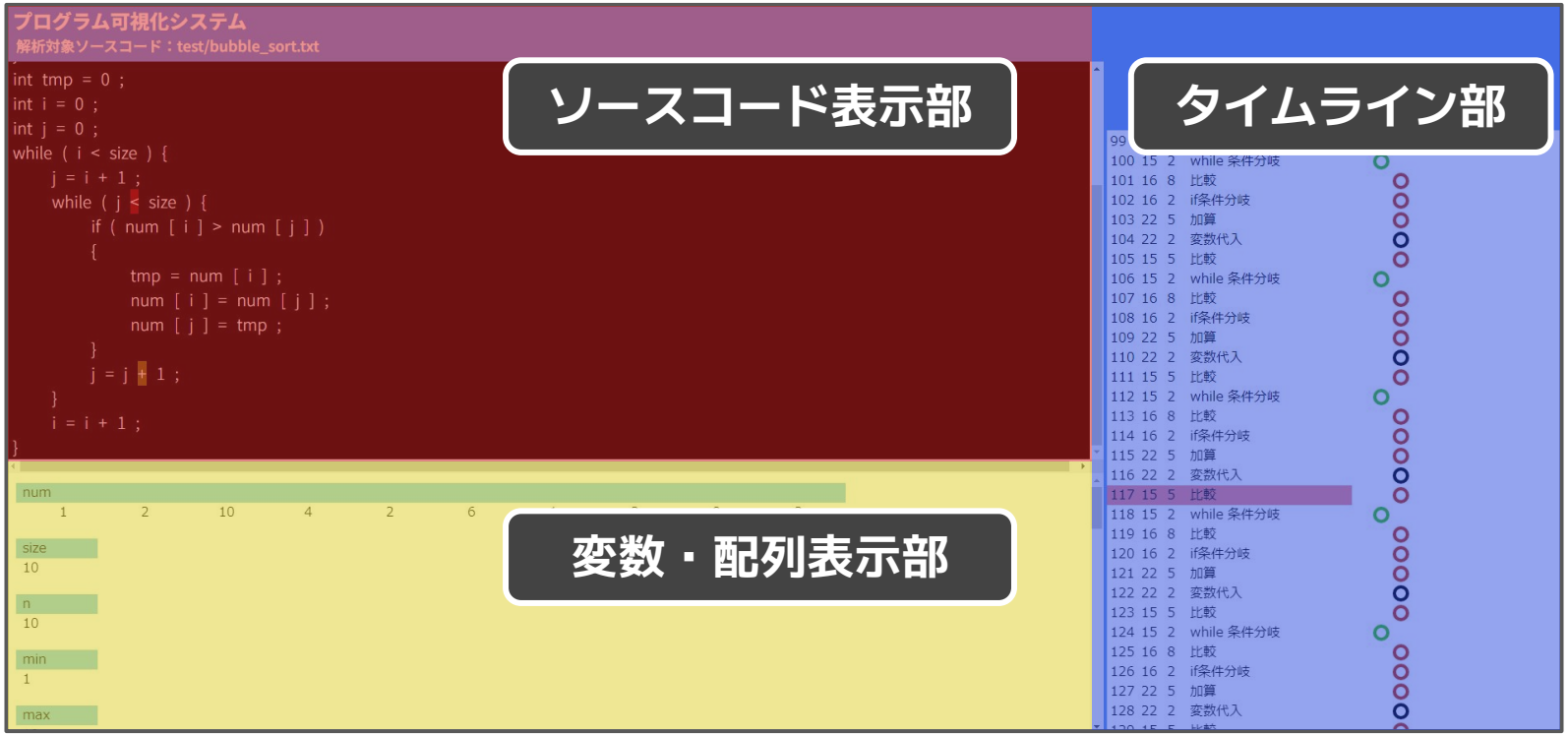
2 プロジェクト概要


ソースコード
一般的なコンパイラ
gcc/g++/javacなど
実行ファイル
クラスファイル
オブジェクトファイル
・・・など
ソースコード
本システム
タイムラインによる
可視化ファイル
（解析結果）

2 可視化ファイル（解析結果表示画面）について



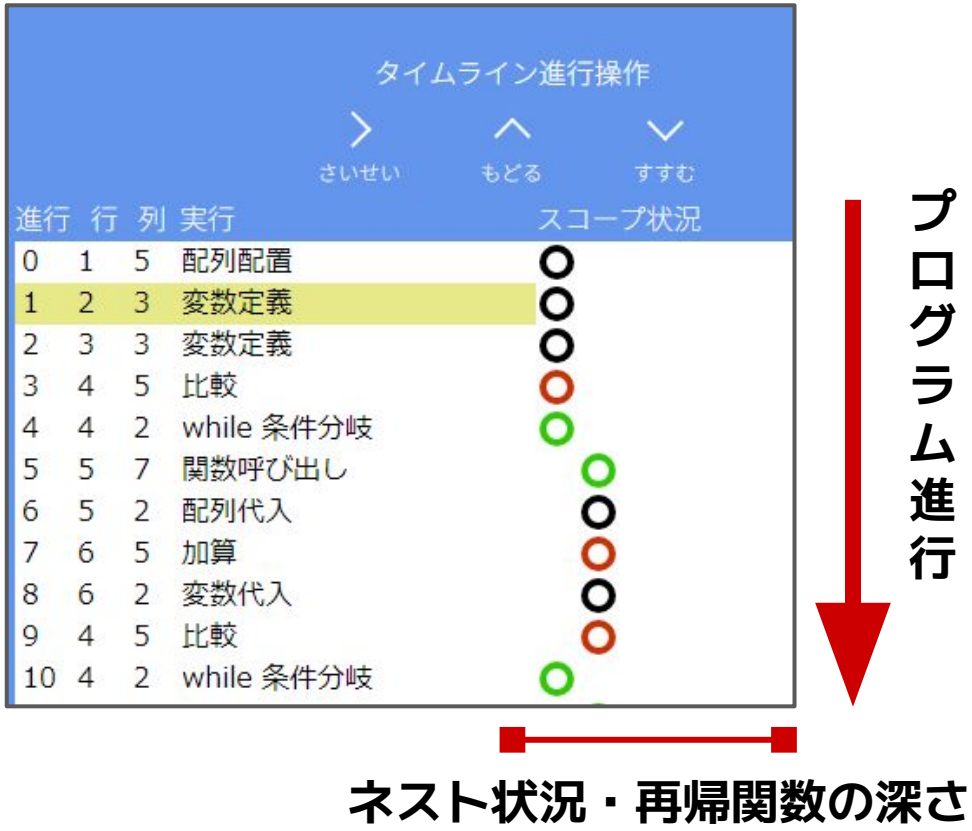
2 可視化ファイル（解析結果表示画面）について



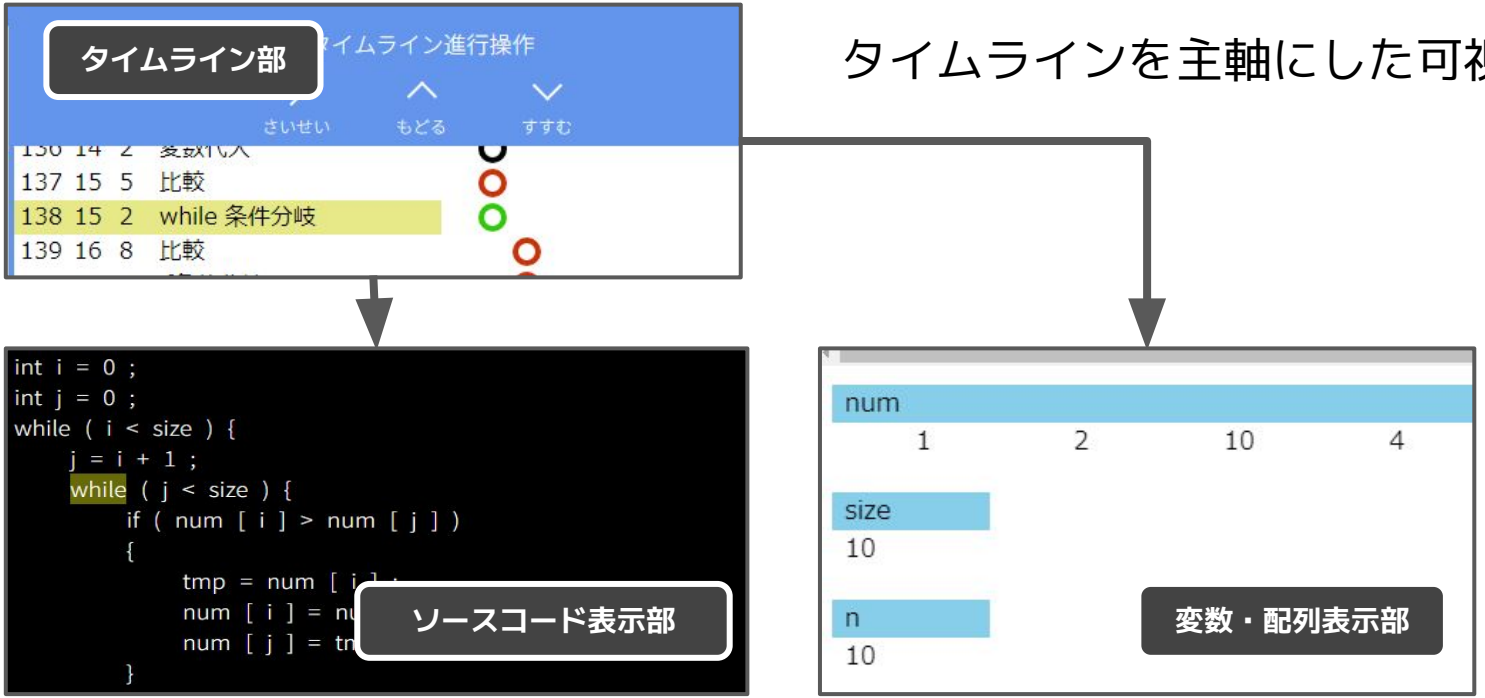
3つの表示部より構成

2 可視化ファイル（解析結果表示画面）について

タイムライン部



2 可視化ファイル（解析結果表示画面）について



タイムラインを主軸にした可視化

2 解析対象となる言語について

CやC++、Javaなどの言語の
基本データ構造に関する機能を参考にした独自言語

現時点でも、再帰関数や多次元配列に対応しているため、
一定の高度なアルゴリズムを可視化できる体制が整っている

```
int i = 0 ;
int function ( ) {
    if ( i < 3 ) {
        i = i + 1 ;
        function ( ) ;
    }
}
function ( ) ;
```

2 プロジェクト概要・デモ

タイムラインを活用した
プログラムの動作を可視化する学習システム

早速実演＞＞＞
題材は「バブルソート」「ランレングス圧縮」

3 特徴 (1/7)

自由に試したいソースコードを可視化できる

自分が書いたプログラムを動かせる！
作って遊ぶことができる！楽しい！

プログラム可視化システム
解析対象ソースコード：test/bubble_sort.txt

```
int tmp = 0 ;
int i = 0 ;
int j = 0 ;
while ( i < size ) {
    j = i + 1 ;
```

3 特徴 (2/7)

セキュリティ教育をアルゴリズム面からサポート

セキュリティで重要な暗号理論・符号理論には
様々なアルゴリズムが使用されている！

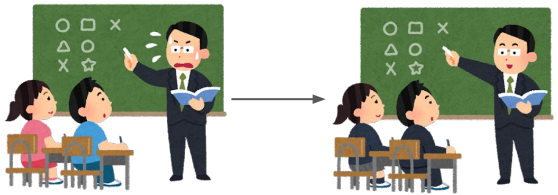
本システムでは、解析器、解析木走査、文法定義が許す限り
符号理論を含めたさまざまなアルゴリズムの可視化が可能

すでにランレングス圧縮(連長圧縮)の可視化に成功！

ランレングス圧縮可視化ファイルを、バブルソート可視化ファイルと共に
スライド掲載場所に掲示しておきますので、ぜひ遊んでみてください

3 特徴 (3/7)

小回りが利く可視化ファイル



がんばってパワーポイントを使って
動作の説明をする資料を作らなくてよい
このシステムで一発！

Google Classroomに可視化ファイルを添付して
資料として配布することもできる

3 特徴 (4/7)

実行エラーの強調表示によるエラー対処の練習

```
int [ ] array = new int [ 10 ] ;
int a = array [ 11 ] ;
```

プログラムは思った通りには動かない！
書いた通りに動く！

タイムライン進行操作

さいせい　もどる　すすむ

進行	行	列	実行	スコープ状況
0	1	5	配列配置	○
1	2	8	配列範囲外アクセス(要素)	○
2	2	9	言語処理系異常検出	○

実行エラーの強調表示、
このシステムを開発しているときに欲しかった（小声）

3 特徴 (5/7)

柔軟な構文解析・文法定義が行える仕組みの開発

```
<S> ::= <EL>
<EL> ::= <E> *
<E> ::= <left> ";" | <right> ";" | <left> "=" <right> ";" | <ifexpr> | <while> | <function> | <return> ";"

<left> ::= <value_definition> | <array_definition> | <value_name> | <array_name>
<right> ::= <comparison_equal> | <function_message_passing> | <new_array>

<comparison_equal> ::= <comparison_equal> "==" <comparison> | <comparison>
<comparison> ::= <comparison> "<" <expr> | <comparison> "<=" <expr> | <comparison> ">" <expr> | <comparison> ">=" <expr> | <expr>

<expr> ::= <expr> "+" <term> | <expr> "-" <term> | <term>
<term> ::= <term> "*" <factor> | <term> "/" <factor>  | <term> "%" <factor> |  <factor>
<factor> ::= <number> | "(" <right> ")" | <value_name> | <array_name> | <text>

<number> ::= NUM
<value_definition> ::= <type_name> <value_name>
<array_definition> ::= <type_name> <array_length> <value_name>

<type_name> ::= "int" | "string" | "void"
<value_name> ::= TEXT

<text> ::= "\"" TEXT "\"" | "'" TEXT "'" | "\"" NUM "\"" | "'" NUM "'" | "\"" "\"" | "'" "'"

<array_name> ::= <value_name> <array_length>
<array_length> ::= "[" <right> "]" <array_length> | "[" "]" <array_length> |

<new_array> ::= new <type_name> <array_length>

<ifexpr> ::= <if>
<if> ::= "if" <right> <block>
<while> ::= "while" <right> <block>
<function> ::= <value_definition> "(" <argument> ")" <block>
<function_message_passing> ::= <value_name> "(" <value_enumeration> ")"
<argument> ::= <value_definition> "," <argument>  | <value_definition> |
<value_enumeration> ::= <right> "," <value_enumeration>  | <right> |
<return> ::= "return" | "return" <right>

<block> ::= "{" <EL> "}"
```

LR(1)構文解析法に基づいた解析器の実装

BNF準拠の構文定義ファイル、
および構文木走査モジュールを組み変えるだけで、
可視化の根拠となる言語仕様の変更が可能

3 特徴 (6/7)

多次元配列の可視化もばっちり

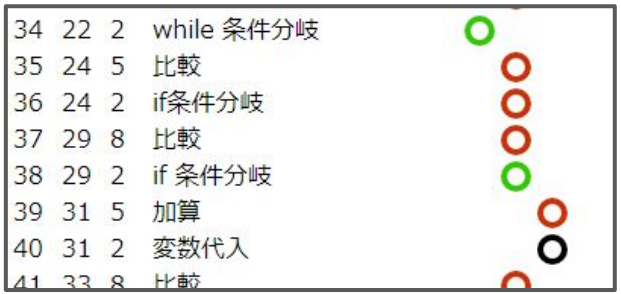
```
int [] array2 = new int[2][2];
int [] array3 = new int[2][2][2];
int [] array4 = new int[2][2][2][2];
int [] array5 = new int[2][2][2][2][2];
int [] array6 = new int[2][2][2][2][2][2];
```

何次元でも可視化が可能な表現機能

3 特徴 (7/7)

再帰関数が理解しやすい可視化

```
int i = 0 ;
int function ( ) {
    if ( i < 3 ) {
        i = i + 1 ;
        function ( ) ;
    }
}
function ( ) ;
```



whileとifの組み合わせによるネスト

再帰関数の仕様を直感的に学ぶことができる　再帰関数に親しくなれる

再帰的アルゴリズムが怖いとはもう言わせない！

4 フィードバックの実施・評価

フィードバックを実施

自分が開発したものを、どういった形でもいいから、
「誰かに見てもらいたい！」「使ってもらいたい」と考え、実施

可視化ファイルを配布し、それを閲覧・体験してもらい、
Googleフォームに回答してもらう形式
依頼のうち、資料作成までに3名の方に協力を頂きました。

Sechack365外　社会人/情報系学校
X(旧Twitter)などのダイレクトメッセージで依頼

4 フィードバックの実施・評価

フィードバック結果の概要

- **コンセプトについては概ね高評価**
- 可視化ファイルのデザイン・UIについては改善の余地が大きくあり
- 初学者のみならず、中級者を含めたアルゴリズム学習に役立ちそうだ
- 競技プログラミングを意識したコメントも頂きました

4 フィードバックの実施・評価（抜粋して紹介）

コンセプトについて

グラフィカルに表現され、視覚的に非常に面白く、
抽象的な概念に対する親しさを覚える

初心者から中級者以上まで幅広く役立てられそうなので、
このプロダクトの秘めるポテンシャルはかなり大きそう

4 フィードバックの実施・評価（抜粋して紹介）

可視化について　評価点

- ifやwhile文の処理が色で分かる点も分かりやすい
- コードの中で今どこが実行されていて、ループで戻る場所がわかる
- タイムライン上の処理の説明と
  ソースコード上のハイライトが同期しており、対応関係がわかりやすい
- 配列が実際にソートされている様子が反映されていたため、
  どの数字がどこに移動するのか把握しやすい

4 フィードバックの実施・評価（抜粋して紹介）

可視化について　改善点

- タイムラインやプログラム同様、変数や配列にも色が反映されると、
どの部分が変更されたかもっと分かりやすくなりそう
- 配列のswapなど重要な処理に関して、
アニメーションでその様子が強調されたら教材レベルになりそう
- コードの中の変数にカーソルを合わせた時、変数の中身を見たい

4 フィードバックの実施・評価（抜粋）

アルゴリズムについて

- 再帰処理（なんと、対応済み！）
- 探索、リスト構造について
- 木やグリッド形式で与えられる問題
- 幅優先探索や深さ優先探索
- 動的計画法（レーベンシュタイン距離の導出アルゴリズムに言及）

おわり

開発で参考にした主な文献

・コンパイラ入門―構文解析の原理とlex/yacc、C言語による実装
　　サイエンス社 山下義行 2008年
・低レイヤを知りたい人のためのCコンパイラ作成入門
　　https://www.sigbus.info/compilerbook#
・LR(1)パーサジェネレータを自作して構文解析をする 第1回:かんたん構文解析入門
　　http://tatamo.81.la/blog/2016/12/22/lr-parser-generator-implementation/
・LR(0)構文解析を行いながら、コンパイラの構文解析を学ぶ
　　https://memo.yammer.jp/posts/compiler-lr0
・LR parsing
　　https://www.slideshare.net/ichikaz3/lr-parsing