Vol.2015-SE-190 No.16
2015/12/16

プログラミング初学者のためのソースコード再利用支援

藤原 新[1,a)]　畑 秀明[1,b)]　門田 暁人[2,c)]　松本 健一[1,d)]

概要：プログラミング学習者は，既存のソースコードを参考にすることで，より良い実装方法やアルゴリズムを学べることがある．Web ベースの競技プログラミングサイトでは，過去に提出された回答ソースコードが大量に保存されているが，その中からプログラミングの学習に繋がる回答ソースコードを見つけることは難しい．本研究では，学習者の既存ソースコード再利用を支援するため，問題に対応する回答ソースコードをアルゴリズムや実装方法等の解法の違いによって分類し，ソースコードを見つけやすくする．本稿では，簡単な問題に対応する回答ソースコードをソースコードの特徴で分類した結果，解法毎に分類できたことを報告する．

1. はじめに

競技プログラミングとは，出題されるプログラミング問題を解く速さや，正確さなどを競い合う競技であり，スキル向上を目指す多くのプログラミング学習者が問題に挑んでいる．

競技プログラミングの多くは時間制限を設けたコンテスト形式で行われるが，コンテスト後にも競技プログラミングサイトに問題とテストケースが公開されるため，学習者は常時問題に挑戦できる．また，競技プログラミングサイトでは，他人が過去に提出した回答ソースコードが公開されており，学習時に参考にできる．

既存の回答ソースコードは問題毎に管理されており，問題の ID で検索すると，問題に対する回答ソースコードの一覧が閲覧できる．しかし，より細かい情報，例えば，使用しているアルゴリズムなどの情報から回答ソースコードを検索する手段はない．

競技プログラミングの問題の中には，複数の解法で解ける問題がある．そのため，学習者が一つの解法で問題に正解したとしても，他の解法を調べることで，より効率の良い書き方や，異なるアルゴリズムについて学ぶことができることがある．例えば，競技プログラミングの問題に頻出するアルゴリズムであるソートアルゴリズムにはクイックソートやバブルソートをはじめ，様々な実装方法があり，計算効率などに差はあるが入力と出力は基本的に同じである．ソートアルゴリズムを用いる問題において，バブルソートのみを習得している学習者がバブルソートを用いて問題に正答したとき，それよりも計算効率の良いクイックソートなどで書かれた回答ソースコードを参照することで，学習者はより効率の良いソートアルゴリズムを知ることができる．その結果，より制約が厳しい問題に対応できるようになり，プログラミングやアルゴリズムへの理解が深まることが考えられる．

このような状況では，学習者は問題に正解した全ての回答ソースコードからあらゆる解法で解かれた回答ソースコードを探すが，大量の回答ソースコードの中からあらゆる解法について書かれた回答ソースコードを見つけるのは，時間のかかる困難な作業である．

そこで，本研究では既存の回答ソースコードから，解法によって回答ソースコードを検索するシステムを構築することで回答ソースコード再利用を支援する．

本稿では，その前調査として既存の回答ソースコードをソースコードメトリクスを用いて解法毎に分類できるか試みる．

2. 提案手法

2.1 解法の違い

競技プログラミングサイトに提出された回答ソースコードは問題が同じであっても，その記述方法は様々である．ここでは，回答ソースコードの何に着目して分類するべきかを説明する．

学習者が既存の回答ソースコードから得たい知見は，計算効率や実装方法などであることが考えられる．そのため，この知見とは直接関係しない条件式に若干の差異があ

[1] 奈良先端科学技術大学院大学
[2] 岡山大学
[a)] fujiwara.shin.fe5@is.naist.jp
[b)] hata@is.naist.jp
[c)] monden@okayama-u.ac.jp
[d)] matumoto@is.naist.jp

るプログラムや，処理の手順は同じだが使用している変数名が違うプログラムなどは同じ解法として分類したい．反対に，アルゴリズムやデータ構造が異なるプログラムは計算効率や実装方法も異なると考えられるので，異なる解法として分類したい．

そこで本研究では，アルゴリズムやデータ構造，標準ライブラリの使用の有無を解法の違いと定義する．アルゴリズムとは，解を求めるための具体的な手順や操作のことである．

2.2 特徴量による回答ソースコードの分類

我々は，既存の回答ソースコードを解法から検索するシステムを構築する．そのためには，まず，回答ソースコードを解法毎に分類し，管理する必要がある．そこで，回答ソースコードの構文情報に関するメトリクスを用いて解法の分類に試みる．

異なる解法の回答ソースコードは，その実装方法についても当然異なる．実装方法の異なる回答ソースコードは，特徴も異なることが考えられる．また，同じ解法で解いている回答ソースコードは実装方法も類似したものとなるので，回答ソースコードの特徴も類似すると考えられる．

そこで，本手法では提出された複数の回答ソースコードのメトリクスを計測し，特徴が類似する回答ソースコードをまとめることで，回答ソースコードを解法毎に分類する．

2.3 回答ソースコードの特徴量

競技プログラミングに提出された回答ソースコードは一般のソフトウェア開発で作成されたソースコードと異なる性質が幾つかある．回答ソースコードは 200 行未満の短いソースコードが多く，クラスも少ない．また，競技プログラミングには決まった命名規則やコーディングスタイルがないため，回答ソースコードは様々なスタイルで記述されている．加えて，競技プログラミングではアルゴリズムやデータ構造に関する問題が多く出題されるので，回答ソースコードは複雑なコントロールフローを持つものが多い．

回答ソースコードから特徴量を取得するには，回答ソースコード特有の性質を考慮する必要がある．例えば，回答ソースコードはクラス数が少ないため，クラスの依存関係などのメトリクスは使えない．また，コーディングスタイルも様々であるため，ソースコード中に含まれる変数名などの言語情報を特徴とすることも難しい．反対に，配列の数や変数の数などプログラムの構造から取得できるようなメトリクスには，回答ソースコードの特徴が顕著に現れると考えられる．

そこで我々は，回答ソースコードを構文解析し，プログラムの構造からメトリクスを取得する．

3. 関連研究

一般のソフトウェア開発を支援するために，ソースコードの検索技術は数多く研究されている．Inoue らは Component Rank モデルを用いたソースコード検索ツール SPARS-J を開発した [4]．Sadowski らは Google における検索ツールの使用目的や使用方法を調査した [6]．

Zimmerman らはプログラミング初学者のために，部分的なソースコードから所望のソースコードを推薦するフレームワークを提案した [8]．

一方，ソースコードの再利用や知識の共有を目的に，ソフトウェアを分類する研究もされている．Kawaguchi らはオープンソースプロジェクトをソースコード中の識別子から LSA により分類するツール MUDAblue を開発した [5]．Tian らはプログラムに含まれる言語情報から LDA によりドメインごとにソフトウェアを分類した [7]．

我々は競技プログラミングの回答ソースコードを対象としており，解法毎の分類を試みている点で従来研究とは異なる．

4. 実験

4.1 データセット

実験には競技プログラミングサイト AIZU ONLINE JUDGE: Programming Challenge（AOJ）[1] の提出データを用いる．AOJ は情報オリンピック，パソコン甲子園，ACM-ICPC などの過去問をはじめ，様々な問題が登録されており，多くのプログラミング学習者に利用されている．

本実験では AOJ に公開されている問題の内， Reverse Sequence と List of Top 3 Hills に対して提出された回答ソースコードに対して分析を行う．Reverse Sequence と List of Top 3 Hills は多くの人が挑戦している比較的難易度が低い問題である．本来は，難易度が高い問題についても分析するべきであるが，難易度が高い問題は様々なアルゴリズムが複合的に使われており，分類した結果に与える要因を特定しにくいと考えられる．そこで本実験では，分類結果の要因を明確にするため，比較的簡単な 2 つの問題を選択した．

AOJ では C，C++，Java など様々な言語を用いて問題に挑戦できる．AOJ に提出された回答ソースコードは，複数のテストケースが実行され，問題の仕様を満たしているか自動的に判定される．本実験では，AOJ に提出された回答ソースコードの内，Java で記述されており，全てのテストケースを通過した回答ソースコードのみを対象とする．AOJ に提出された実験対象の回答ソースコードは，Reverser Sequence に対して 144 件，List of Top 3 Hills に対して 227 件あった．

表 1　メトリクス一覧
Table 1　Metrics list.

メトリクス名	説明
#var	変数名の数
#import	import の数
#Method	メソッドの数
#Method Call	メソッドの呼び出し回数
#Object Creation	オブジェクトの生成数（new の数）
Nest	制御文の最大ネスト数
Array Accese	配列の参照回数
#(for, while, if, break)	制御文の使用回数
#Integer	int 型変数の使用回数
#String	String 型変数の使用回数
#Boolean	Boolean 型変数の使用回数
#Char	Char 型変数の使用回数
#Double	Double 型変数の使用回数
#(!,&&,||)	論理演算子の数
#(==,>,>=,<,<=,!=)	比較演算子の数
#(+,-*,/,\%,++,--)	算術演算子の数

4.2　ソースコードメトリクス

回答ソースコードから特徴を抽出するために，回答ソースコードの構文木情報から表 1 に示す 31 個のメトリクスを取得した．構文解析には javaparser[2] を用いた．

取得したメトリクスはそれぞれ値の範囲が異なっている．例えば，制御文の最大ネスト数は 2 から 4 までの値を取る回答ソースコードが多く，配列の参照回数は 0 から 12 までの値を取る回答ソースコードが多い．このようなデータをそのまま実験に使用すると，それぞれのメトリクスが分類結果に与える影響に偏りが起きる．

そこで，使用するメトリクスはそれぞれの平均値が 0，分散が 1 になるように線形変換し，正規化した．

4.3　リサーチクエスチョン

回答ソースコードの解法毎の分類が実現可能か確かめるため，2 つのリサーチクエスチョンから調査を行った．

RQ1：解法を分類する上で重要なメトリクスは何か．

RQ1 では，どのメトリクスを使用すれば解法毎に回答ソースコードを分類できるかを調査する．

回答ソースコードから 31 個のメトリクスを取得したが，全てのメトリクスが解法に関係するメトリクスとは限らない．また，解法に関係のないメトリクスを使うと，解法以外の指標で分類される恐れがある．

そこで，解法と関係しているメトリクスを事前に調査し，解法を知る上で重要なメトリクスを特定する．

重要なメトリクスを調査するために，回答ソースコード毎に手動で解法のラベル付けを行い，回答ソースコードの特徴からその解法を予測する機械学習モデルを構築する．更に，機械学習モデルが解法を予測する際に寄与する度合いをメトリクスの重要度として，調査する．

機械学習モデルは Random Forest[3] を用いた．Random Forest は決定木を弱学習器とするアンサンブル学習方式である．アンサンブル学習では複数の学習器を組み合わせることで学習器の汎化能力を向上させる．

Random Forest は，使用している説明変数の重要度を算出でき，どの説明変数が目的変数に寄与しているのかが分かる．本実験では，目的変数を解法，説明変数をメトリクスとしているので，Random Forest により，どのメトリクスが解法に寄与しているのかが分かる．

RQ2：回答ソースコードは解法毎に分類できるか．

RQ2 では RQ1 で得られた重要なメトリクスを用いて実際に回答ソースコードを解法毎に分類できるかを調査する．RQ1 では回答ソースコードに予め手動で解法をラベル付けをして予測モデルを構築したが，本来であれば，メトリクスの情報だけで自動的に回答ソースコードを分類したい．そこで，RQ2 では解法のラベルは使わず，メトリクスの情報のみを使って，回答ソースコードの分類に試みる．

メトリクスの情報のみで回答ソースコードを分類するために，クラスタリング手法を適用する．クラスタリングとは，複数のデータをデータ間の距離に基いて幾つかの部分集合に分ける教師なし学習手法である．

クラスタリングには様々な手法があるが，本実験では階層クラスタリングを用いる．階層クラスタリングとは，それぞれのデータを 1 つのクラスタとし，最も距離の近い 2 つのクラスタを 1 つのクラスタに併合する処理を繰り返すことにより，部分的な集合に切り分けていく手法である．階層クラスタリングでは，クラスタができていく過程を可視化できるので，任意のクラスタの数に分類した結果を確認できる．

RQ2 では解法が正しく分類できているか評価するために，RQ1 で使用した解法のラベルを用いる．我々の目的は解法毎に分類することなので，解法をラベル付けした回答ソースコードがそれぞれ解法毎に回答ソースコードが分類されることが好ましい．

4.4　解法のラベル付け

RQ1，RQ2 の分析をするにあたって，予め，回答ソースコードに対して解法のラベル付けをする必要がある．この節では解法のラベル付けの方法について説明する．

本実験では，Reverse Sequence と List of Top 3 Hills に対して提出された回答ソースコードをそれぞれランダムに 50 件選択し，解法のラベル付けをする．Reverse Sequence は文字列を入力したとき，その文字列を逆順にした文字列を出力するプログラムを記述する問題である．Reverse Sequence に対する回答ソースコードからは，次の 3 種類の解法を確認できた．与えられた文字列を逆順に参照し，表示する方法 (方法 A)，StringBuffer あるいは StringBuilder の reverse 関数を使用する方法 (方法 B)，文字列の両端か

表 2　解法に関係のあるメトリクス
Table 2　Important metrics.

List of Top 3 Hills		Reverse Sequence	
メトリクス名	重要度	メトリクス名	重要度
#var	0.15	#Integer	0.20
#Array Accese	0.14	#var	0.12
#Integer	0.10	#Array Accese	0.09
#if	0.10	#Object Creation	0.08
#for	0.08	#for	0.07
Nest	0.08	Nest	0.05
#--	0.07	#>	0.05
#>	0.05	#Method Call	0.05
#++	0.05	#!=	0.04
#<	0.03	#if	0.03

ら内側にかけて順に前と後の文字をスワップする方法 (方法 C) である．方法 A，方法 B，方法 C の回答ソースコードの代表例を図 3，図 4，図 5 に示す．

List of Top 3 Hills は与えられる複数の整数値から大きい順に 3 つ出力する問題である．List of Top 3 Hills に対する 50 件の回答ソースコードからは次の 4 種類の解法が確認できた．標準ライブラリのソートメソッドを使う方法 (方法 D)，入力を順番に参照し，値が大きなものを変数に保存する方法（方法 E)，バブルソートによる解法（方法 F)，クイックソートによる解法（方法 G）である．

Reverse Sequence と List of Top 3 Hills に提出された回答ソースコードの解法を分類した結果，方法 A は 21 件，方法 B は 22 件，方法 C は 7 件，方法 D は 19 件，方法 E は 17 件，方法 F は 8 件，方法 G は 5 件の回答ソースコードがあった．

4.5　RQ1:解法を分類する上で重要なメトリクスは何か．

List of Top 3 Hills と Reverse Sequence から得られた，解法を予測する上で重要度の高いメトリクスを表 2 に示す．

表 2 から変数の数や Integer の使用回数，配列のアクセス回数，変数の数などのメトリクスはどちらの問題においても重要度が高く，問題に共通して重要なメトリクスがあることが分かる．一方，比較演算子やメソッドの呼び出し回数，オブジェクトの作成数などのメトリクスはどちらか片方の問題にのみ上位にあり，問題固有で重要なメトリクスであることが分かる．

4.6　RQ2:回答ソースコードは解法毎に分類できるか

RQ2 では，RQ1 で調査した重要なメトリクスのそれぞれ上位 5 件のみを使い，回答ソースコードをクラスタリングする．クラスタリングには，2 つの問題に対して提出された全ての対象の回答ソースコードを用いる．

Reverse Sequence を階層クラスタリングした結果を図 1 に示す．図 1 の縦軸はクラスタが併合された距離を示しており，各終端ノードは各回答ソースコードを示している．それぞれの回答ソースコードに対して評価用にラベル付けした解法は終端ノードの位置にそれぞれ A，B，C で表示している．下に何も表示していない終端ノードは，本実験で解法のラベル付けをした 50 件以外の回答ソースコードである．

図 1 の距離 1.9 の破線に注目すると，回答ソースコードは 7 個のクラスタに分かれており，解法をラベル付けした全ての回答ソースコードが解法毎に同じクラスタに集まっていることがわかる．これより，Reverse Sequence では，7 つのクラスタに回答ソースコードを分類することで，複数の解法が混ざったクラスタがなくなる．この結果から，Reverse Sequence では高い精度で回答ソースコードがクラスタリングできていることが読み取れる．

List of Top 3 Hills を階層クラスタリングした結果を図 2 に示す．この結果では，距離 1.3 の破線に注目すると，回答ソースコードのクラスタは 13 個あり，方法 F は他の解法のクラスタと混在しているものの，他の解法については正しく分類できていることが分かる．方法 F は方法 E や方法 D のクラスタと混在しており，正しいクラスタリングはできなかった．

4.7　考察

RQ1 では 2 つの競技プログラミングの問題に対して，解法を特定する上で重要なメトリクスを調査した．調査の結果，メトリクスの中には問題に共通して重要なメトリクスがあることが分かった．この結果は，問題に共通して重要なメトリクスを使うことで，問題固有の性質に関係なく，解法を分類できることを示唆している．

RQ2 では 2 つ問題に対して，実際に解法毎に回答ソースコードが分類できるか調査した．調査の結果，Reverse Sequence に対して提出された回答ソースコードは高い精度で分類ができており，この問題においては，実用可能なレベルで分類できていることが分かった．一方，List of Top 3 Hills では，方法 F が他の解法のクラスタと混在しており，正しくクラスタリングできていないことが分かった．図 2 の右側を見ると特に方法 E との判別ができていない．方法 E は入力を順番に参照し，値が大きなものを変数に保存する方法であり，方法 F はバブルソートによる解法である．この 2 つの解法はどちらも配列を順に参照しつつ，大きさを比較して値を入れ替えるという点で共通しており，類似した解法である．この結果より，本実験で用いたメトリクスでは，類似した解法の判別ができないことが分かった．

より細かい解法の違いを分類するためには，プログラムの構造や，処理の手順，データの流れを考慮したメトリク

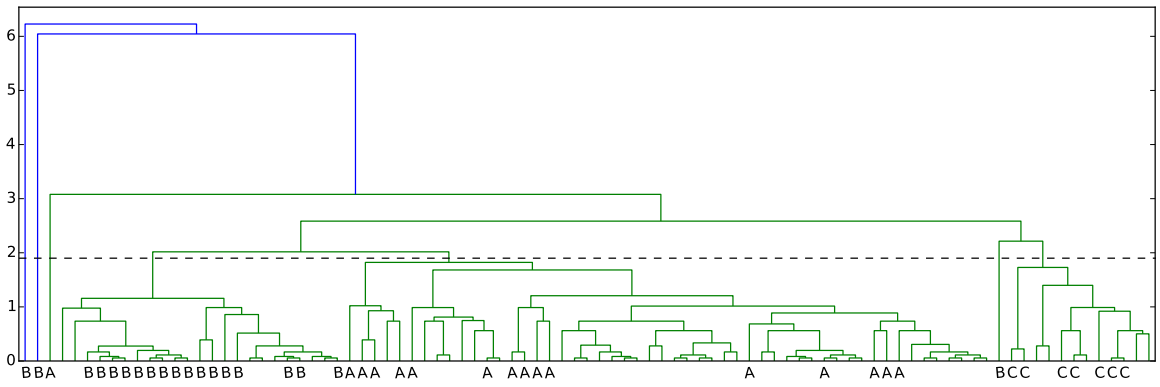

図 1 Reverse Sequence のクラスタリング結果

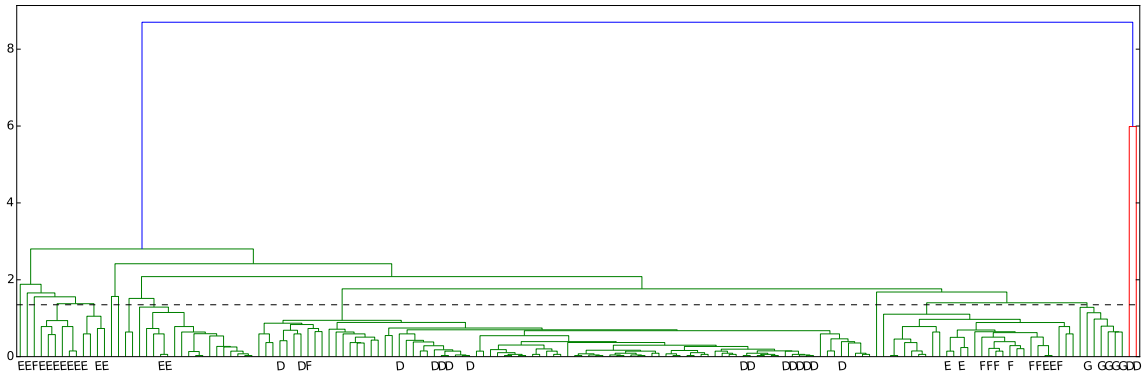

図 2 List of Top 3 Hills のクラスタリング結果

スを抽出する必要がある．具体的には文の前後関係や変数の依存関係などの情報を取ることで，精度の高い分類ができるのではないかと考えられる．

これらの実験結果から，競技プログラミングの簡単な問題において，類似しない解法の回答ソースコードは解法毎に分類できることが分かった．また，簡単な問題において，解法による検索システムを構築できる可能性を示せた．

5. おわりに

競技プログラミングにおいて，解法による回答ソースコードが検索できないという問題に対して，回答ソースコードを解法毎に分類することで検索する手法を提案した．また，実際にその手法が実現可能か確かめるために，回答ソースコードを解法毎に自動分類する実験を行った．

実験の結果，簡単な問題に対して提出された回答ソースコードはおおよそ解法毎に分類可能であることが分かった．しかし，類似した解法は判別できない場合もあり，メトリクスの取得方法を変えるなど，工夫が必要であることが分かった．

本実験では，比較的簡単な問題に対して分析を行った．今後，様々な問題に対しても同様に提案手法を試していき，汎用性を高めていく予定である．

今後，回答ソースコードの検索システムを構築していく上で，自動的に解法に意味付けし，学習者に提示するシステムが必要であると考えている．学習者が解法から回答ソースコードを検索する際，学習者は解法を選択する必要があるが，現段階では解法を選ぶ基準がない．そこで，解法に含まれる回答ソースコードの特徴を要約して学習者に提示することで，解法を選ぶ基準にできないか調査する予定である．

謝辞

本研究の遂行にあたり，Aizu Online Judge のデータを提供していただきました会津大学渡部有隆准教授及び，データ分析についてご助言いただきました奈良先端科学技術大学院大学小田悠介氏に心から感謝いたします．

本研究は JSPS 科研費 26540029 と頭脳循環を加速する戦略的国際研究ネットワーク推進プログラムの助成を受けた．

参考文献

[1] Aizu online judge: Programming challenge. `http://judge.u-aizu.ac.jp/onlinejudge/`. Accessed: 2015-11-12.

[2] Java parser and abstract syntax tree. `https://github.com/javaparser/javaparser`. Accessed: 2015-11-12.

[3] Leo Breiman. Random forests. *Mach. Learn.*, Vol. 45, No. 1, pp. 5–32, October 2001.

[4] Katsuro Inoue, Reishi Yokomori, Tetsuo Yamamoto, Makoto Matsushita, and Shinji Kusumoto. Ranking significance of software components based on use relations. *IEEE Trans. Softw. Eng.*, Vol. 31, No. 3, pp. 213–225, March 2005.

[5] Shinji Kawaguchi, Pankaj K. Garg, Makoto Matsushita, and Katsuro Inoue. Mudablue: An automatic categorization system for open source repositories. In *In Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC.04*, pp. 184–193. IEEE Computer Society, 2004.

[6] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. How developers search for code: A case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pp. 191–201, New York, NY, USA, 2015. ACM.

[7] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, MSR '09, pp. 163–166, Washington, DC, USA, 2009. IEEE Computer Society.

[8] Kurtis Zimmerman and Chandan R. Rupakheti. An automated framework for recommending program elements to novices. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2015, pp. 283–288. ACM, 2015.

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);
        System.out.println(new StringBuilder(input.next()).reverse());
    }
}
```

図 3 解法 A の代表例

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class Main {
    public static void main(String args[]) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        char[] c = br.readLine().toCharArray();
        char[] rev = new char[c.length];
        for(int i=c.length-1, j=0;i>=0;i--,j++){
            rev[j]=c[i];
        }
        System.out.println(String.valueOf(rev));
    }
}
```

図 4 解法 B の代表例

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String line = "";
         while ((line = br.readLine()) != null && !line.isEmpty()) {
            char[] str = line.toCharArray();
            char c;
            int u;
            for (int i = 0; i < str.length / 2; i++) {
                c = str[i];
                u = str.length - 1 - i;
                str[i] = str[u];
                str[u] = c;
            }
            System.out.println(str);
        }
    }
}
```

図 5 解法 C の代表例