修士学位論文

題目

入力コードに応じた模範解答を提示する
プログラミング学習支援システムの構築

指導教員
肥後芳樹 教授

報告者
北庄司亮

令和 5 年 2 月 1 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

令和 4 年度 修士学位論文

入力コードに応じた模範解答を提示するプログラミング学習支援システムの構築

北庄司亮

内容梗概

コンピュータが内蔵されたものが身近に増えたため，コンピュータを効果的に利用するためにプログラミング学習に対する需要は高まっている．そのためプログラミング学習者は増加しており，文部科学省がプログラミング学習を小学校から取り入れていることからも学習を支援する機会は増えている．

学習においてインプット，アウトプット，フィードバックを受けることは重要である．しかしプログラミング初学者は知識や経験が不足しているために，自身でコードをフィードバックすることが困難である．さらにプログラミングは模範解答がひとつとは限らないため，他者が適切にフィードバックすることも容易ではない．

本研究では，既存研究を利用して入力したコードを改善すべきか判定した上で，改善が必要な場合は入力したソースコードに構造が類似した模範解答となるソースコードと，その要約を推薦するシステムを構築した．このシステムは code2vec などの学習モデルを用いることで入力したソースコードと類似した模範解答を提示できる．

評価実験には定量的な評価とアンケートによる評価を行った．まず Project CodeNet のデータセットを用いてプログラミングコンテストの問題を用意し，用意した問題を被験者に解答させた．その後システムを利用して改善した場合と，システムを利用せずに改善した場合を比較し，定量的な評価を行った．またその被験者にはアンケートを回答してもらい，システムの有無による改善の難易度について調査した．

結果として，システムを利用した場合に平均で 2 分の改善時間の短縮が見られたが，入力が複雑なソースコードの場合には改善が十分でないと予想された．またアンケートの結果からシステムを使うことで改善の難易度が下がる場合もあるが，上がる場合もあり，単純なソースコードにすることと改善の難易度が反比例することがわかった．

主な用語

プログラミング学習
ソフトウェア品質
コード要約

目 次

1　はじめに

コンピュータはスマートフォンやパソコンなどの身近なものに内蔵されており，生活を豊かにするために重要な要素である．そのためコンピュータを効果的に利用することを目的としてプログラミング学習に対する需要が高まっている [15]．これらの内容を受け，文部化科学省は小学校からプログラミング教育を必修化しており [16]，プログラミング学習者は増加している．また，プログラミング学習者の増加にともなって学習を支援する機会も増加する．

学習にはインプット，アウトプット，フィードバックを受けることが重要である [12]．しかし，プログラムはアルゴリズムや設計の問題などが理由で，模範解答をひとつに絞れるとは限らないため，フィードバック先のソースコードが複数存在することもありうる．そのため適切にフィードバックすることは容易でない．

本研究では学習モデルを用いて入力したソースコードの構造に類似した模範解答を提示し，プログラミングの学習支援を行うシステムを構築する．このシステムは入力をソースコードとし，そのソースコードに改善の余地がある場合には入力したソースコードの構造と類似する模範解答，入力したソースコードの要約，模範解答の要約を提示するシステムである．

このシステムの評価方法は実験とアンケートにより行った．まず Project CodeNet のデータセットを基に問題とそれに対する模範解答群を用意し，被験者が問題を解いたあと，提案するシステムを用いて改善した場合と改善しない場合での改善状況を計測する．この計測結果によりシステムの有無による改善度の違いを評価した．その後被験者には改善しやすさに関するアンケート調査を行った．

以下，2 章では本研究の背景として，プログラミングコンテストとソフトウェアの評価指標，そしてシステムに用いた既存研究であるソースコードを自動評価する手法，ソースコードをベクトル化するための code2vec, ソースコードを要約するための code2seq について説明する．3 章で提案するシステムの設計方法について説明する．4 章では提案するシステムの評価方法となる実験とアンケートの内容について説明し，5 章では 4 章で説明した実験の結果と考察を行う．最後に 6 章ではまとめについて述べる．

2 背景

本研究では，プログラミング学習支援システムを提案する．この章では研究の背景として，データセットとして利用した Project CodeNet[1]，プログラミングコンテストとその採点に利用するオンラインジャッジシステム，ソフトウェアの品質指標，システムに用いた既存研究について説明する．

2.1 オンラインジャッジシステム

オンラインジャッジシステムとは，オンラインで演習問題に関するソースコードを自動採点するシステムである．プログラマが書いたプログラムをテストすることで自動的に評価し，正誤に関して判定できる．これによりプログラミングの能力を評価するためのオンラインテストやコンテストを開催することができる．採点基準は以下のようなものがある．

- コンパイルできるか
- 出力が適切か
- メモリ使用量
- 実行時間

このような基準に対して満たさない場合はその箇所を，すべて満たす場合は正解であることを示す．具体的なオンラインジャッジシステムとしては AIZU Online Judge[1]，AtCoder[2]，Topcoder [3] などがある．

2.2 プログラミングコンテスト

プログラミングコンテストは，プログラミングの能力を競うコンテストのことである. 参加者は，与えられた問題を解決するためのプログラムを記述し，その結果を評価するためにオンラインジャッジシステムに提出する．その後，正解した問題数などの様々な基準で参加者の順位を決定する．大学，プログラマ，企業などの様々な団体によって開催され、個人で参加するものからチームで参加するものまである．コンテストはオンラインで開催されることが多く，世界中の人々が参加できる．競技は以下の五種類に分類される．

- アルゴリズム，コンピュータサイエンス，数学に関する問題を解く早さ
- ソフトウェア, ネットワークのセキュリティ上の問題点を発見すること
- ソースコードの小ささ

[1] https://github.com/IBM/Project_CodeNet

- 人工知能 (AI) を作成
- 作品製作

本研究では，個人で参加し，アルゴリズム，コンピュータサイエンス，数学に関する問題を解く早さを競うコンテストを扱う．

2.3 Project CodeNet

Project CodeNet[8] は，ImageNet[6] が Computer Vision の分野に与えた影響を，AI for Code の分野にも与えることを目的としており，オンラインジャッジシステムである AIZU Online Judge と AtCoder[2] から収集された 4000 を超える問題とそれに対応して様々な言語で解答されたソースコードやメタデータがまとめられている．言語の例としては Java, Python3，C, C++, Ruby などがあり，このデータセットはソースコードに関連する機械学習のトレーニングに対してよく使用される．データセットのメタデータは以下のようなものがある．

- 問題番号
- 参加者の id
- 解答日時
- プログラミング言語
- コンパイル情報
- 実行時間
- 使用メモリ量
- コードのサイズ
- プログラムの正誤

本研究では，プログラム言語，実行時間，プログラムの正誤に関する情報を利用し，模範解答群を作成する．

2.4 ソフトウェアの品質指標

ソフトウェアの品質を高く保つことは重要な要素である．品質には可読性，一貫性，拡張性，再利用性などの様々な要素があり，高品質なコードを作成することは高品質なソフトウェアの開発に役立つ．以下では評価に利用した指標について述べる．

2.4.1 Cyclomatic Complexity

Cyclomatic Complexity[10]（以下，CC）はプログラムの構造の複雑さに関する指標であり，この指標が高いほどプログラムは複雑で，バグが発生しやすくなることから保守性が低くなりやすい．制御フローグラフを解析し，そのグラフの辺の数に基づいて計算され,CC はグラフのエッジ数 E, グラフのノード数 N, 連結成分の個数 P を用いて次式で表される．

$$CC = E - N + 2P$$

制御フローグラフは図 1 のように，プログラムの中のすべてのフロー制御構造を表したグラフで，ノードは逐次コード，辺は制御フロー（if 文、for 文、while 文など）を用いる．

```
public class Main {
  public static void main(String[] args) {
    if(args.length > 0) System.out.print(args[0]);
  }
}
```

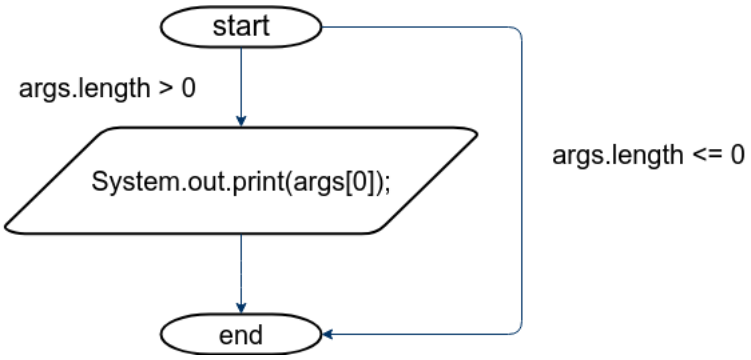


図 1: 制御フローグラフの具体例

2.4.2 Lines of Code

Lines of Code(以下，LOC) はソフトウェア開発の規模や複雑度を評価するために使用される指標であり，プログラムの行数をコメントや空行も含めて数えたものである．LOC はプログラミング言語や開発環境に依存するため，この指標だけでソフトウェアの質を決定することができない．そのため，他の指標と組み合わせて使用する．

2.5 システムに利用した既存研究

2.5.1 品質の自動評価

松井ら [13] は堤 [14] によって作成されたプログラミングコンテストのデータセット[2]を用いて，3 値分類によるソースコードの良さを判定するモデルを作成している．判定するまでの流れが大きく分けて 3 段階になっており，それぞれの段階を図 2，図 3，図 4 に示す．

まずデータセットとなるソースコード群を上級者コード群・中級者コード群・初級者コード群で分割する．このとき，上級者・中級者・初級者の定義は解答者のレートを基準として 4 つに分割し，最もレーティングが高いソースコード群を上級者，最もレーティングが低いソースコード群を初級者，残りの 2 つを中級者コード群とする．

次に分割したソースコードに対して表 1 に示した予約語の利用頻度や 2 メトリクスの値などのソースコード特徴量を取得し，機械学習の説明変数とする．目的語はソースコードの良さであり，上級者コード群を”良”，上級者コード群を”良くない”，中級者コード群を”どちらでもない”として学習する．これによりソースコードの品質を評価することができる．

表 1: 特徴量に使用した予約語

asm	break	case	catch
class	continue	decltype	do
else	enum	extern	for
friend	goto	if	namespace
operator	private	public	return
struct	switch	template	try
typedef	typeid	typename	using
while			

[2] https://sites.google.com/site/miningprogcodeforces/

表 2: 特徴量に使用したメトリクス

メトリクス	説明
avg complexity	各関数の CC の平均値
max complexity	各関数の CC の最大値
avg depth	各関数のネスト深さの平均値
max depth	各関数のネスト深さの最大値
methods per class	クラス当たりのメソッド数
n classes	クラス数
n func	関数の数
n lines	物理行数
n statements	セミコロンで区切られた論理行数
percent branch statements	全体の論理行数に占める分岐文の割合
percent comments	全体の物理行数に占めるコメントの割合
avg statements per method	各メソッドの論理行数の平均値
statements at block level 0	深さ 0 の論理行数
statements at block level 1	深さ 1 の論理行数
statements at block level 2	深さ 2 の論理行数
⋮	⋮
statements at block level 8	深さ 8 の論理行数
statements at block level 9	深さ 9 以上の論理行数

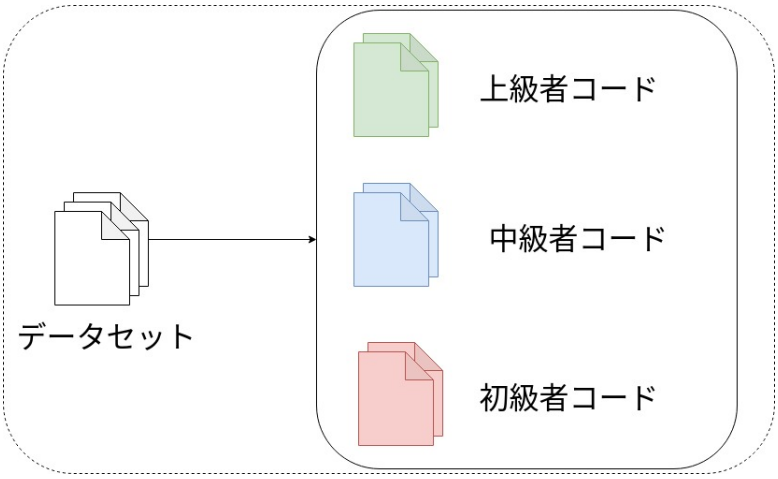


図 2: ソースコードの分類

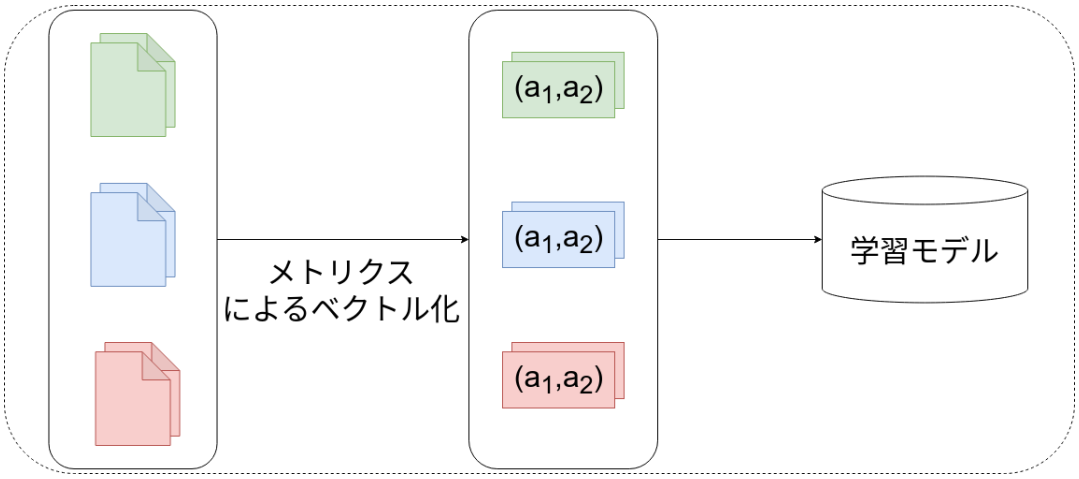


図 3: メトリクスを用いた学習

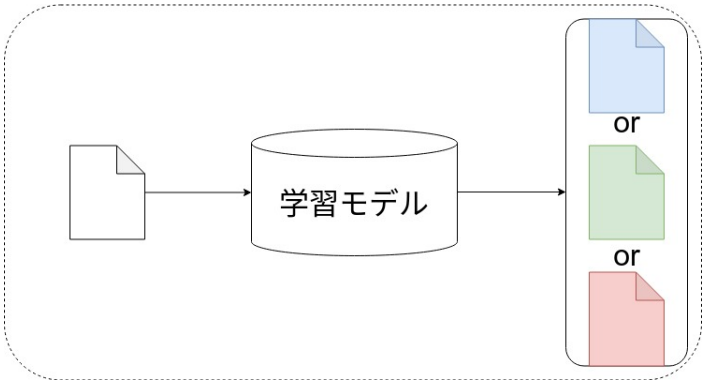


図 4: ソースコードの判定

2.5.2 code2vec

code2vec[5] はソースコードのメソッドに対する分散表現を作るニューラルモデルであり，Uri らは Allamanis ら [11] が作成した Java のデータセットを基に作成したトレーニング済みモデル[3]や，モデルを利用するためのソースコード[4]をオンライン上に公開している．このモデルの学習までの手順を説明する．まず図 5 のようにソースコードのメソッドを Abstract syntax tree(以下，AST) に変換する．AST はソースコードをコンパイル際にできる構文情報を表現した中間表現であるため，AST を利用することでソースコードの構造に着目した分散表現が作れる．

[3] https://code2vec.s3.amazonaws.com/model/java-large-released-model.tar.gz
[4] https://github.com/tech-srl/code2vec

```
public class Main {
  public static void main(String[] args) {
    if(args.length > 0) System.out.print(args[0]);
  }
}
```

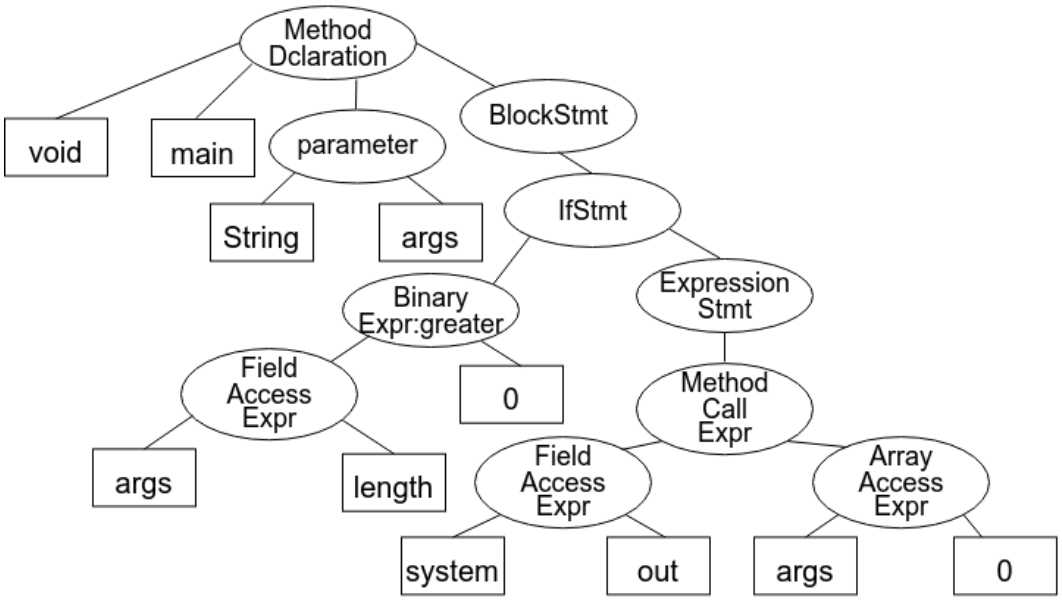


図 5: AST の具体例

次に AST から Bag of Path-Contexts という分散表現を作成する過程を図 6 に示す．AST の葉から葉までのパスを表現したものを AST-Path とし，(始点, AST パス, 終点) を Path-Context とする．Path-Context は一つの AST に対して複数存在するため，それらをまとめて Bag of Path-Contexts という．

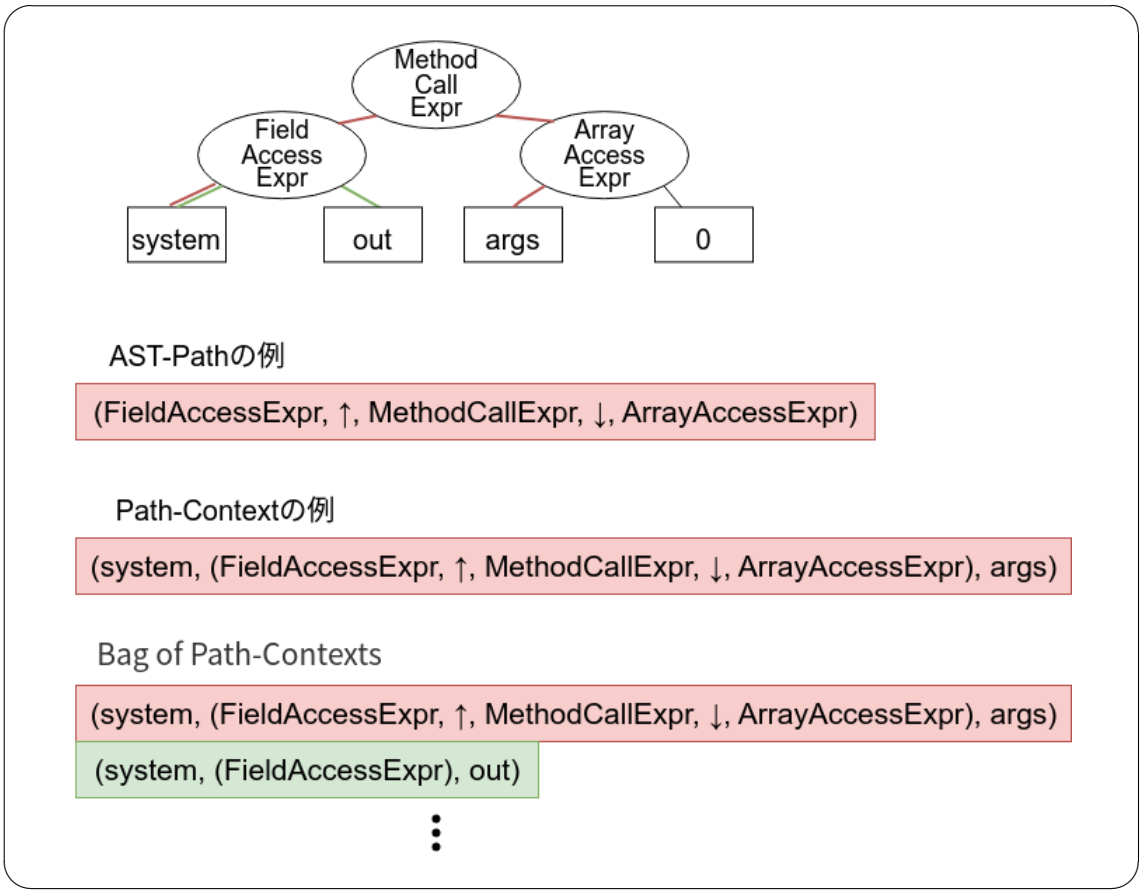


図 6: Bag of Path-Contexts までの流れ

最後に Bag of Path-Contexts をベクトルに変換する．まず Path-Context の始点，終点，AST パスを図 7 のように One-Hot ベクトルに変換した後，埋め込みを行うことで次元を固定長に変換する．さらに作成した Path-Context の埋め込みベクトルを活性化関数 tanh と線形変換する行列を用いて一つのベクトルに結合する．

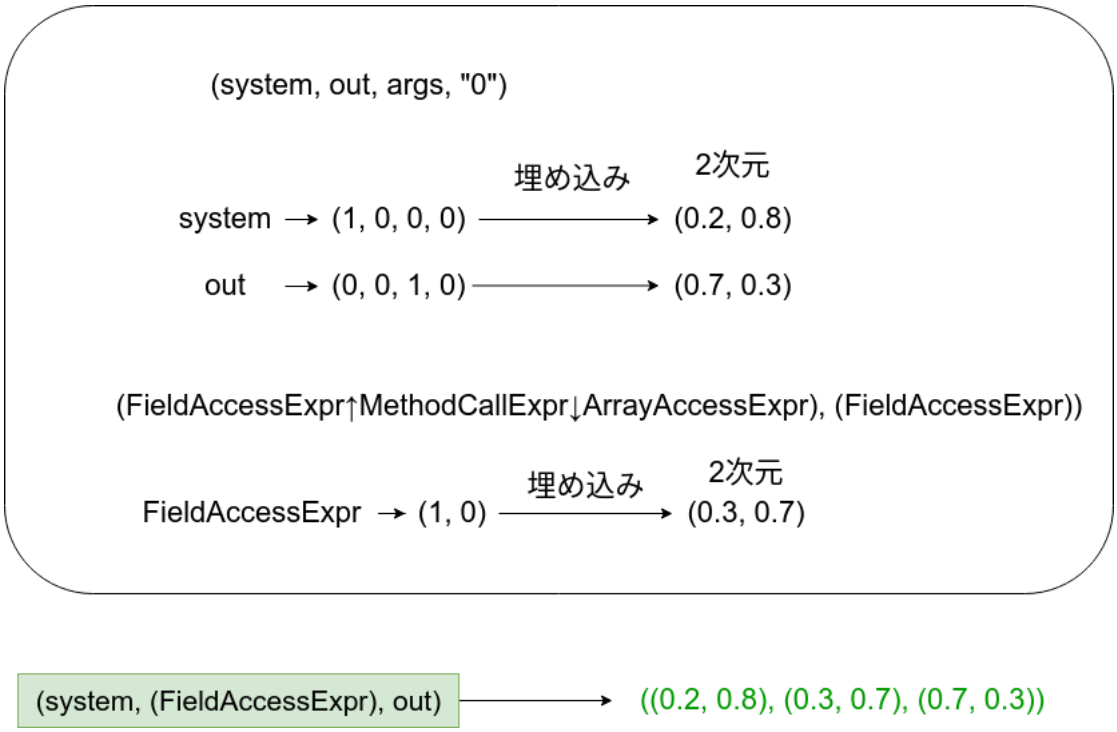


図 7: Path-Context のベクトル化

モデルは Bag of Path-Contexts を基に学習するため，ソースコードの構造に着目したベクトルとなる．著者の Alon らは code2vec を用いて以下のようなことができると述べている．

- メソッド名の提案
- 機能の類似性を基にしたコード検索
- プログラムが I/O を実行するか予測
- プログラムの依存関係を予測
- マルウェアの検知

研究ではソースコードをベクトル化し，類似したコードの探索を行うことに利用している．

2.5.3 code2seq

code2seq[4] は code2vec を改良し，メソッド名の提案に特化させるために設計されたニューラルモデルであり，Uri らは code2vec のモデルを作成したときと同様のデータセットを用いて作成したモデル[5]と，モデルを利用するためのソースコード[6]をオンライン上に公開して

[5] https://s3.amazonaws.com/code2seq/model/java-large/java-large-model.tar.gz
[6] https://github.com/tech-srl/code2seq

いる．code2seq でも 2.5.2 項と同様に AST から AST-path を作成するが，code2vec とは異なり code2seq では学習の精度を上げるために葉をサブトークンに分割する．具体例としては $linear_search$ を $linear, search$ のように変化させることで固有の言葉を減少させる．また AST-Path をベクトル化する際に bi-directional LSTM[9] という入力の時系列に依存した学習に有効な学習方法を利用している．これはメソッド名を単語の羅列として出力できるように前後の関係を学習させるためである．さらに Attention 機構 [7] を用いることにより，入力のどの単語が重要かを重み付けしている．これらの工夫により code2seq はメソッド名を連続した単語の羅列で提案でき，この手法を利用して研究ではソースコードの要約を行った．このようにして作成したモデルも 2.5.2 項で説明した code2vec と同様に bag of Path-Contexts を基にベクトル化するため，ソースコードの構造に依存した要約となる．

3　提案システム

この章では，提案するシステムの機能と設計について説明する．システムの全体像を図 8 に示す．まずシステムにはソースコードを入力し，そのソースコードに改善する必要がなければシステムは終了する．改善の余地がある場合は，入力とソースコードの構造が類似した模範解答を出力するため，ソースコードを構造に着目してベクトル化し，類似した模範解答を選定する．そして選定した模範解答と入力を基に以下，4 種類の情報を出力する．

- 入力したソースコードの要約情報
- 模範回答となるソースコードのファイル名
- 模範解答の要約情報
- 入力と模範解答の類似度

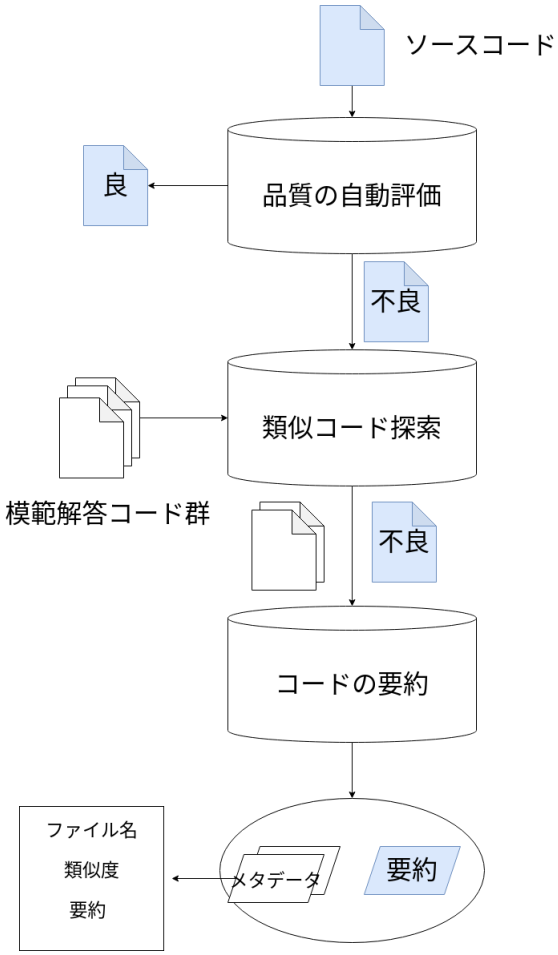


図 8: システムの全体像

3.1 事前準備

事前準備としてソースコードを自動で評価する学習モデル，code2vec の学習モデル, code2seq の学習モデル, プログラミングの問題，問題に対応した解答コード群とメタデータを用意する．このときメタデータには正誤と実行時間に関する情報を用いる．本研究ではソースコードを自動で評価する学習モデルを松井らの既存研究と同様に作成し，code2vec と ode2seq はともに Uri らが作成した学習済みモデルを利用し，プログラミングの問題はやそのメタデータなどは Project CodeNet を利用した．

3.2 模範解答群と付随する情報の作成手順

模範解答は入力と独立しているため，あらかじめソースコードの構造に基づいたベクトル化やコードスニペットの予測によって要約することができる．以下では模範解答の条件と要約，ベクトル化について述べる．

3.2.1 模範解答群の条件

解答コード群とメタデータを基に，模範解答群を作成する．模範解答は解答コード群の中から正解とされており，表 3 の要素を満たすものとした．

この指標を用いた理由としては LOC と CC が低いものを指標にすることで，模範解答のソフトウェア品質を高く保ち，実行時間が短いソースコードを選択することで，模範解答を時間的計算量の高いソースコードとするためである．

表 3: 模範解答の条件

メトリクス	閾値 (%)
LOC	50
CC	50
実行時間	50

3.2.2 模範解答の要約とベクトル化

code2vec のモデルを用いて図 9 のように模範解答のベクトル化を行い，同様に code2seq のモデルを用いて図 10 のように模範解答の要約情報を得る．このときのベクトル数は 100 次元で，要約情報はソースコードのメソッドを単語の羅列で予測したものである．この情報を基に入力に類似した模範解答と，その要約情報を提示する．

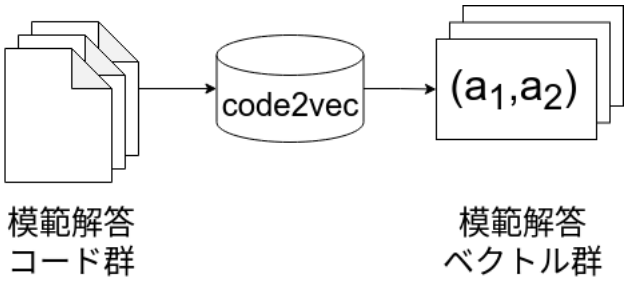


図 9: ソースコードのベクトル化

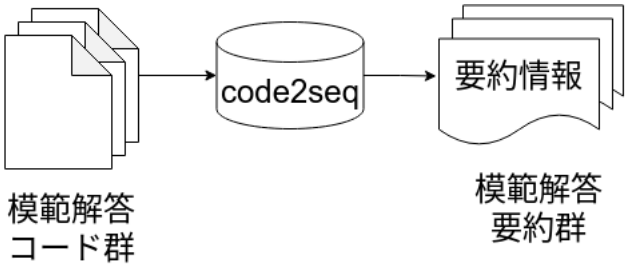


図 10: ソースコードの要約

3.3 入力の評価

入力の自動評価には 2.5.1 項で述べたソースコードの評価手法を図 11 のように利用する．入力されたソースコードが' 良' と評価されたソースコードは改善の余地がないとし，システムを終了させるが，その他の判定は改善の余地があると判定し，次の段階へ移行する．

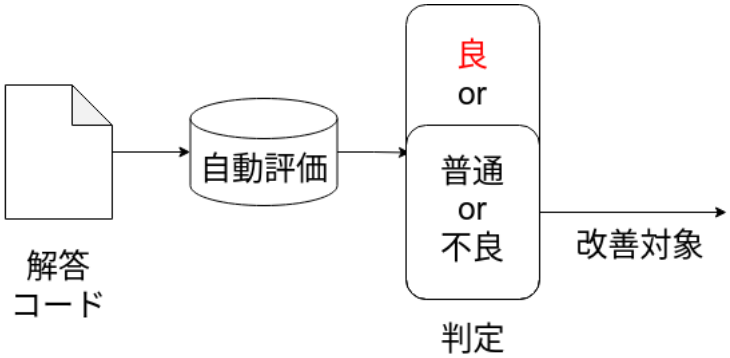


図 11: 入力の自動評価

3.4 入力の要約とベクトル化

入力したソースコードのベクトル化は事前に実行できないため，模範解答群の要約やベクトル化とは別に行う必要がある．モデルは 3.2.2 項と同様のものを用いて，入力したソースコードのベクトル化と要約情報を得る．このときのベクトル数も 100 次元であり，要約情報はソースコードのメソッドを単語の羅列で予測したものである．

3.5 入力と類似した模範解答の選択

入力したソースコードのベクトルと各模範解答とのコサイン類似度をとり，類似度の高いもの 3 つを模範解答として出力する．このときコサイン類似度はベクトルの大きさ $|V|$ を用いて以下のように定義され，具体例を図 12 に示す．図 12 よりコサイン類似度はベクトル間の角度に依存することがわかる．類似度は 1 から-1 までの値を取り，機械学習によく用いられる類似度である．

$$cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| \cdot |\vec{d}|} = \frac{\vec{q}}{|\vec{q}|} \cdot \frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \cdot \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

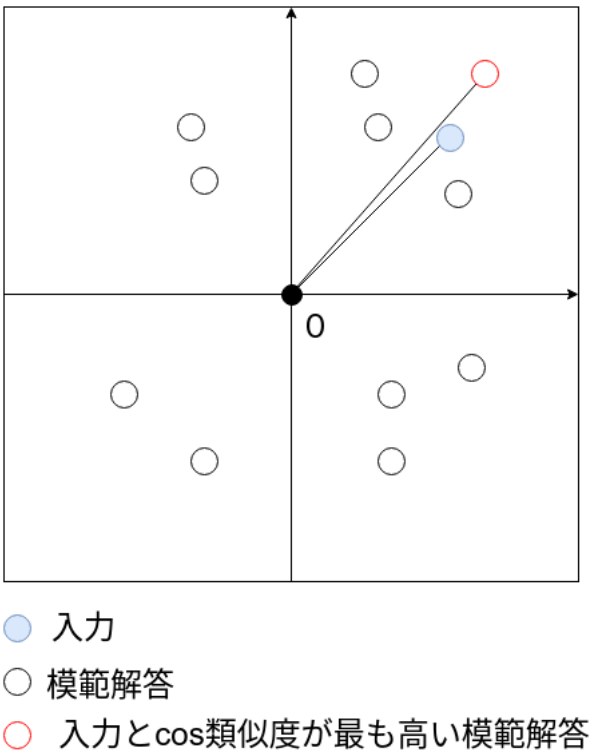


図 12: 類似した模範解答の選択例

3.6 出力

実際の入力と出力結果を図 13，図 14 にそれぞれ示す．図 14 より入力された Input.java の要約内容が main, 最も類似したソースコードが s000127.java, どの要約が main であることがわかる．出力するファイルの個数は変更可能であり，例では出力するファイルの数を 3 としている．

Input.java

```
public class Main {
  public static void main(String[] args) {
    if(args.length > 0) System.out.print(args[0]);
  }
}
```

図 13: 入力の具体例

```
Input.java
predict: main

s000127.java
predict: main
similarity: 0.16

s000182.java
predict: echo
similarity: 0.12

s000102.java
predict: main, echo
similarity: 0.08
```

図 14: 出力の具体例

4　評価方法

この章では，提案システムの評価方法について説明する．評価は実験を行い，定量的評価とアンケート評価の 2 種類で評価した．

4.1　実験内容

実験は図 15 のように行った．事前に Project CodeNet の表 4 の問題を用意し，被験者に実験の概要と注意事項を説明する．被験者は表 4 に示したそれぞれの問題に対して Java で解答し，片方の解答は，正解とされた解答群を基に自身で模範解答を決定して改善する．もう片方の解答は，システムによって提示された模範解答を基に改善する．その後，システムに関するアンケートに回答する．アンケート調査には Google が運営している Google Forms[7] を用いた．

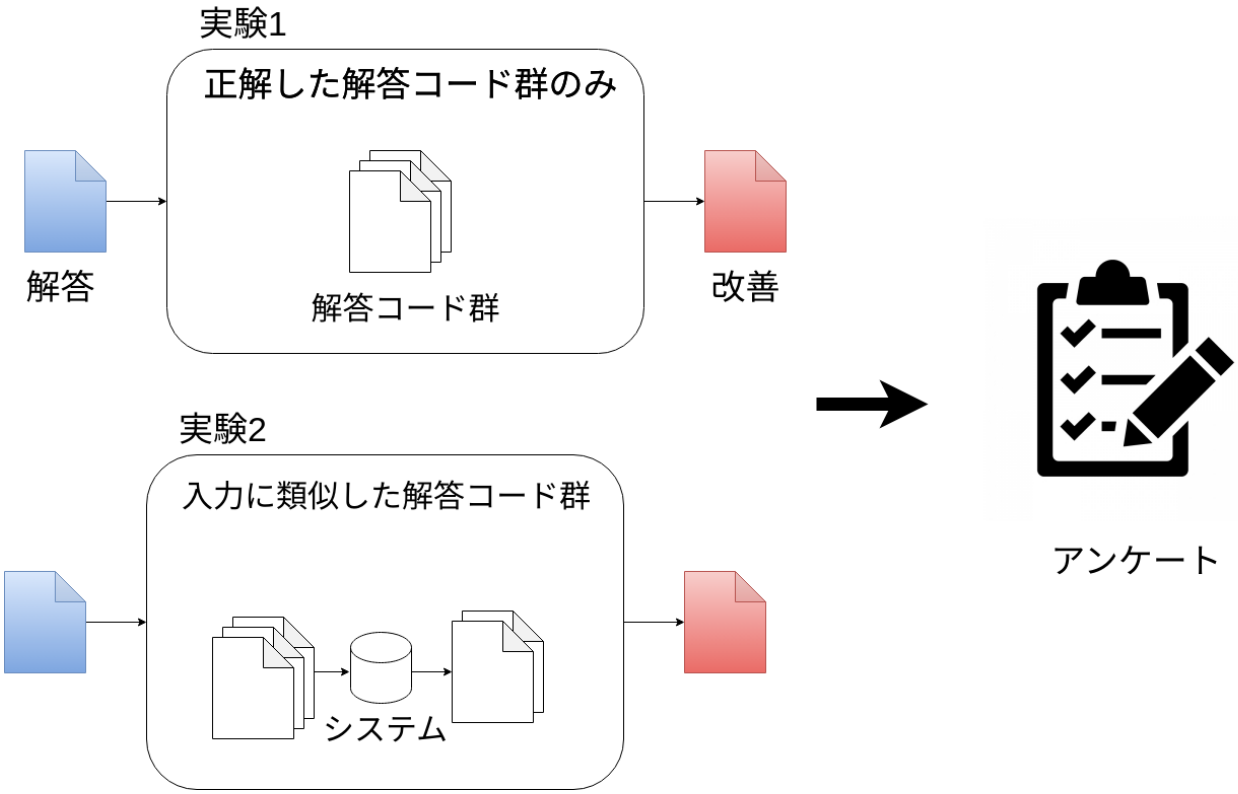


図 15: 実験の流れ

[7] https://docs.google.com/forms/

表 4: 利用した問題

問題番号	問題名
1	玉
2	平面上の図形

4.2 定量的評価

システムの有無により，以下の指標がどのように変化したか確認することでシステムを評価した．

- CC
- LOC
- 改善時間

4.3 アンケート評価

アンケートでは被験者の妥当性とシステムの利便性について確認する．被験者は以下の内容に対する理解度を，表 5 を基準に 5 段階で回答する．

- Java
- プログラミングコンテスト
- Visual Studio Code

これは被験者に関する妥当性を確認するものであり，Java は解答する言語が Java のため，プログラミングコンテストは問題がプログラミングコンテストのため，Visual Studio Code は解答する環境が Visual Studio Code のためである．この理解度を 1 と回答した被験者はシステム以外の要因で結果が変わることを考慮し，今回の実験の対象外とする．

表 5: 理解度の指標

level	内容
1	理解していない
2	あまり理解していない
3	どちらともいえない
4	理解している
5	普段から使用している

また，以下の難易度について表 6 を基準に 5 段階で回答する．

- 問題
- 手動による改善
- システムを用いた改善

これにより問題の妥当性とシステムの利便性について確認する．

表 6: 難易度の指標

level	内容
1	簡単だった
2	やや簡単だった
3	普通
4	やや難しかった
5	難しかった

5　結果と考察

この章では被験者２名による４章の実験を行った結果と、結果に対する考察に関して説明する．

5.1　実験結果とその考察

実験結果を表７に示す．表７から問題番号１ではシステムを用いた方の CC が低くなっているが，問題番号２ではシステムを用いた方が CC が高くなっていることがわかる．これはシステムが類似度による模範解答を出力するため，入力されたソースコードの構造が影響されたと考察する．

次に，改善時間の変化からシステムを用いてることで改善時間の短縮ができていることがわかる．これは模範解答を提示することで被験者が確認したソースコードの量が少なくなったためであると考察する．

最後に，アンケート結果ではシステムを利用することで改善の難易度が下がる場合と，難易度が上がる場合がある．これはシステムの評価基準が．ソースコードの構造のみに着目しているため，コーディングルールのようなその他の要因によるものであると考察する．またこの結果は CC の改善とトレードオフになっているため，CC を小さくすることで改善の難易度が上がると予想される．

表 7: 実験結果

	問題番号１		問題番号２	
システム	あり	なし	あり	なし
CC	4.5	6	16	7
LOC	36	34	53	60
改善時間 (min)	10	13	13	14
改善の難易度	2	1	2	3

5.2　今後の課題

今回の内容からソースコードの構造に関する類似度を用いて模範解答を推薦すると，より品質の高い構造に気づかず，根本的な修正ができない場合がある．そのため類似度の高さ以外にも新たな指標を用いて模範解答を推薦し，その際には CC を下げつつ改善の難易度が上がらないような工夫をしたい．

6　まとめ

本研究では，増加するプログラミング初学者の学習を支援するために，入力したソースコードを基に，入力と構造が類似している模範解答を提示するプログラミング学習支援システムを構築した．このシステムの主な機能はソースコードの自動評価，ソースコードの要約，ソースコードのベクトル化と類似コードの選定である．

提案したシステムを Project CodeNet のデータセットを利用して定量的評価とアンケート評価を行ったところ，システムを用いたほうが改善時間を短縮できることがわかった．しかし，入力したソースコードの構造が複雑な場合，模範解答の構造も複雑になるため，ソフトウェア品質の高いソースコードに改善できなかった．

またアンケートの結果から，システムを用いて改善する方が改善の難易度が上がることがあり，ソースコードの類似度が高いだけでは必ずしも理解しやすいとは限らないことがわかった．この結果からプログラミング学習者にとって理解しやすい模範解答を推薦するために，プログラムの構造以外の指標も含めたシステムの改良を今後の課題とする．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後芳樹 教授には，研究活動において貴重な御指導及び御助言を賜りました．肥後 教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下誠 准教授には，研究の方針から本論文の執筆に至るまで，直接の御指導及び御助言を賜りました．松下 准教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 神田哲也 助教には，研究活動において適切な御助言を賜りました．神田 助教に心より深く感謝いたします．

最後に，その他様々な御指導及び御助言を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻肥後研究室の皆様ならびに事務職員 軽部 瑞穂氏に心より深く感謝いたします．

参考文献

[1] Aizu online judge: Programming challenge. `https://judge.u-aizu.ac.jp/onlinejudge/`. (Accessed on 02/07/2023).

[2] Atcoder. `https://atcoder.jp`. (Accessed on 02/07/2023).

[3] Topcoder. `https://www.topcoder.com/`. (Accessed on 02/07/2023).

[4] Uri Alon, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. 2019.

[5] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. Code2vec: Learning distributed representations of code. *Proc. ACM Program. Lang.*, Vol. 3, No. POPL, pp. 40:1–40:29, January 2019.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. pp. 248–255, 2009.

[7] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation.

[8] Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. 2021.

[9] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, Vol. 45, No. 11, pp. 2673–2681, 1997.

[10] Martin Shepperd. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, Vol. 3, pp. 30 – 36, 04 1988.

[11] アラマニス, ミルティアディス, ペン, ハオ, サットン, チャールズ. ソース コードの極端な要約のための畳み込み注意ネットワーク.

[12] 中原淳. 経験学習の理論的系譜と研究動向. 10 2013.

[13] 松井智寛. 判定対象の拡大を目的とした 3 値分類によるソースコードの良さの判定手法. 大阪大学基礎工学部情報科学科卒業論文, 2020.

[14] 堤祥吾. プログラミングコンテスト初級者・上級者におけるソースコード特徴量の比較. 大阪大学大学院情報科学研究科修士論文, 2018.

[15] 文部科学省. 教育の情報化の手引き-追補版-. 06 2019.

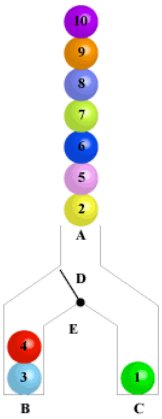
[16] 文部科学省. 小学校プログラミング教育の手引. 2020.

付録

評価に利用した問題とアンケートの例を以下にに示す

2023/02/08 8:49 p00033.html

玉



図のように二股に分かれている容器があります。1 から 10 までの番号が付けられた10 個の玉を容器の開口部 A から落とし、左の筒 B か右の筒 C に玉を入れます。板 D は支点 E を中心に左右に回転できるので、板 D を動かすことで筒 B と筒 C のどちらに入れるか決めることができます。

開口部 A から落とす玉の並びを与えます。それらを順番に筒 B 又は筒 Cに入れていきます。このとき、筒 B と筒 C のおのおのが両方とも番号の小さい玉の上に大きい玉を並べられる場合は YES、並べられない場合は NO と出力するプログラムを作成してください。ただし、容器の中で玉の順序を入れ替えることはできないものとします。また、続けて同じ筒に入れることができるものとし、筒 B, C ともに 10 個の玉がすべて入るだけの余裕があるものとします。

Input

複数のデータセットが与えられます。１行目にデータセット数 N が与えられます。つづいて、N 行のデータセットが与えられます。各データセットに 10 個の番号が左から順番に空白区切りで与えられます。

Output

各データセットに対して、YES または NO を１行に出力して下さい。

Sample Input

```
2
3 1 4 2 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
```

Output for the Sample Input

```
YES
NO
```

file:///home/ryo/Downloads/Project_CodeNet/problem_descriptions/p00033.html 1/1

平面上の図形

縦 8、横 8 のマスからなる図 1 のような平面があります。

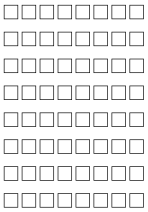
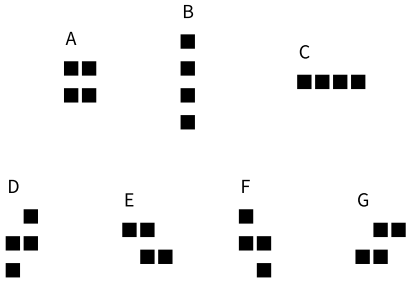


図 1

この平面上に、以下の A から G の図形のどれかが一つだけ置かれています。



たとえば、次の図 2 の例では E の図形が置かれています。

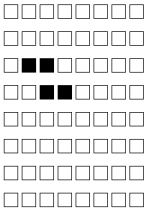


図 2

平面の中で図形が占めているマスを 1、占めていないマスを 0 で表現した数字の列を読み込んで、置かれている図形の種類（A〜G）を出力するプログラムを作成してください。

ただし、ひとつの平面に置かれている図形は必ず 1 つで、複数の図形が置かれていることはありません。また、A〜G で表される図形以外のものが置かれていることはありません。

Input

入力は複数のデータセットからなります。

１つのデータセットとして、平面の中で図形が占めているマスを 1、占めていないマスを 0 で表現した 8 文字からなる 8 つの文字列が与えられます。例えば、図 2 に対応する文字列の並びは次のようになります。

00000000
00000000
01100000
00110000
00000000
00000000
00000000
00000000

データセットの間は１つの空行で区切られています。データセットの数は 50 を超えません。

Output

各データセットごとに、平面に与えられた図形の種類（A〜G のいずれか）を１行に出力してください。

Sample Input

```
00000000
00000000
01100000
00110000
00000000
00000000
00000000
00000000

00011110
00000000
00000000
00000000
00000000
00000000
00000000
00000000

00000000
00000000
00110000
00110000
00000000
00000000
00000000
00000000
```

Output for the Sample Input

```
E
C
A
```

名前 *

回答を入力

メールアドレス *

回答を入力

以下の項目の理解度 *

	普段から使用している	理解している	どちらとも言えない	あまり理解していない	理解していない
Java	◉	○	○	○	○
プログラミングコンテスト	◉	○	○	○	○
Visual Studio Code	◉	○	○	○	○

システムを利用した問題 *

◉ 玉

○ 平面状の図形

問題：玉

難易度について *

	難しかった	やや難しかった	普通	やや簡単だった	簡単だった
問題	☑	☐	☐	☐	☐
参考にした解答コード	☑	☐	☐	☐	☐

問題：平面上の図形

難易度について *

	難しかった	やや難しかった	普通	やや簡単だった	簡単だった
問題	☑	☐	☐	☐	☐
参考にした解答コード	☑	☐	☐	☐	☐

その他の感想

回答を入力

送信　　フォームをクリア