卒　業　研　究　報　告　書

令和6年度

研究題目

解き方を維持した正答ソースコード
作成支援のための類似ソースコードの提示

指導教員　　上　野　秀　剛　　准教授

氏　　名　　澤　田　直　樹

令和7年2月28日　提出

奈 良 工 業 高 等 専 門 学 校　　情 報 工 学 科

解き方を維持した正答ソースコード作成支援のための類似ソースコードの提示

上野研究室　澤田直樹

プログラミング系科目の講義では学習者に対して指定の動作を実現するソースコードを作成する課題が出されることが多い．ある動作を実現するプログラムの作成方法は複数存在するため，正解と見なされるソースコードも複数存在する．一方で，すべての解き方に対して教員が解説を行うことは講義時間の制約上難しく，少数の正答例のみが提示される場合も多い．そのため，正答を作成できなかった学習者のうち，教員による解説がされなかった解き方でプログラムを作成した学習者は，自力で正答を作成するか，解説された解き方に基づいてプログラムを作成し直す必要がある．正答を自力で作成することを求めるのは難しく非効率であり，解説された解き方に基づいて作り直しても，自分の解答がどこまで正答に近かったかや不足点がわからず，効果的なフィードバックが得られない．本研究は正答を作成できなかった学習者を支援するために，学習者が採用した解き方と類似した解き方を持つ正答例を自動的に検索，提示するシステムの開発を長期的な目的とする．その第一段階として，ソースコードの分散表現のcos類似度，階層型クラスタリングによってソースコード間の解き方の類似性を評価できるか実験を行う．コンパイル不可能なソースコードも対象とするために，ソースコードの分散表現の取得に一般的に用いられるcode2vecではなく，word2vecを用いてソースコード中のトークンの分散表現を算出し，平均値をソースコードの分散表現として扱う．実験では，Project CodeNetのデータセットを使用してword2vecモデルの学習を行い，ある課題に対する解答ソースコードの分散表現を取得し，4つの解き方で解かれた8つの対象ソースコードに対してcos類似度の高い上位5つのソースコードを課題に対する全解答ソースコードの中から計算し，それらの解き方について分析した．分析の結果，一定以上ソースコード間の解き方の類似性を分散表現によって評価できることがわかったが，分散表現に反映されやすいデータ構造やアルゴリズムに対しての課題や，類似したソースコード間の解き方を厳密に区別することに対しての課題などが見られた．今後の展望として，doc2vecを用いた順序関係を反映した分散表現の取得やトークンに対する重み付けなどが有効な手法であると考えられた．

目 次

1 はじめに

プログラミング系科目の講義では学習者に対して指定の動作を実現するソースコードを作成する課題が出されることが多い. ある動作を実現するプログラムの作成方法（解き方）は複数存在するため, 正解と見なされるソースコード（正答）も複数存在する. 一方で, すべての解き方に対して教員が解説を行うことは講義時間の制約上難しく, 少数の正答例のみが提示される場合も多い. そのため, 正答を作成できなかった学習者のうち, 教員による解説がされなかった解き方でプログラムを作成した学習者は, 自力で正答を作成するか, 解説された解き方に基づいてプログラムを作成し直す必要がある. 正答を自力で作成することを求めるのは難しく非効率であり, 解説された解き方に基づいて作り直しても, 自分の解答がどこまで正答に近かったかや不足点がわからず, 効果的なフィードバックが得られない. これらは現時点でのプログラミング教育における問題点であると言える.

本研究は正答を作成できなかった学習者を支援するために, 学習者が採用した解き方と類似した解き方を持つ正答例を自動的に検索, 提示するシステムの開発を長期的な目的とする. 開発するシステムは学習者が作成した未完成のソースコードを入力として, 他の学習者が提出した正答から類似度の高いものを検出し, 提示する. 類似度の高い正答ソースコードを提示することで, 学習者の解き方を維持しながら正答への到達を支援する. 本研究において, プログラミング課題の解き方とは使用するデータ構造とアルゴリズムの組み合わせによって表現されるものとして扱う. その上でシステムが対象とする未完成のソースコードは図1に示すように正しい解き方にマッチしないソースコードが大半を占める. 正しい解き方にマッチしないソースコードとは図2に示すような正しい解き方とは違っている, もしくは正答に対し満足でない解き方であるものである. 対して正答ソースコードは正しい解き方のどれかとマッチすることが保証されている. この正しい解き方というものは概念に過ぎないが, 実際に提示したい正答ソースコードの解き方は全て正しい解き方に内包されている. しかし, 未完成のソースコードの解き方は正しい解き方に内包されていない解き方を含むため, これらの未完成のソースコードに対して解き方が類似した正答ソースコードを提示することは, 正答ソースコードに対して解き方が類似した正答ソースコードを提示することに比べて困難である.

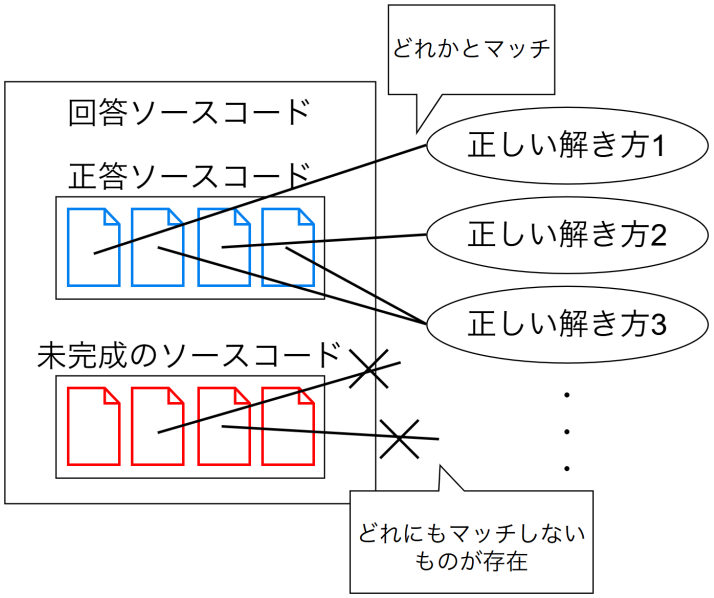

図 1 正答ソースコード, 未完成のソースコードの正しい解き方との関係

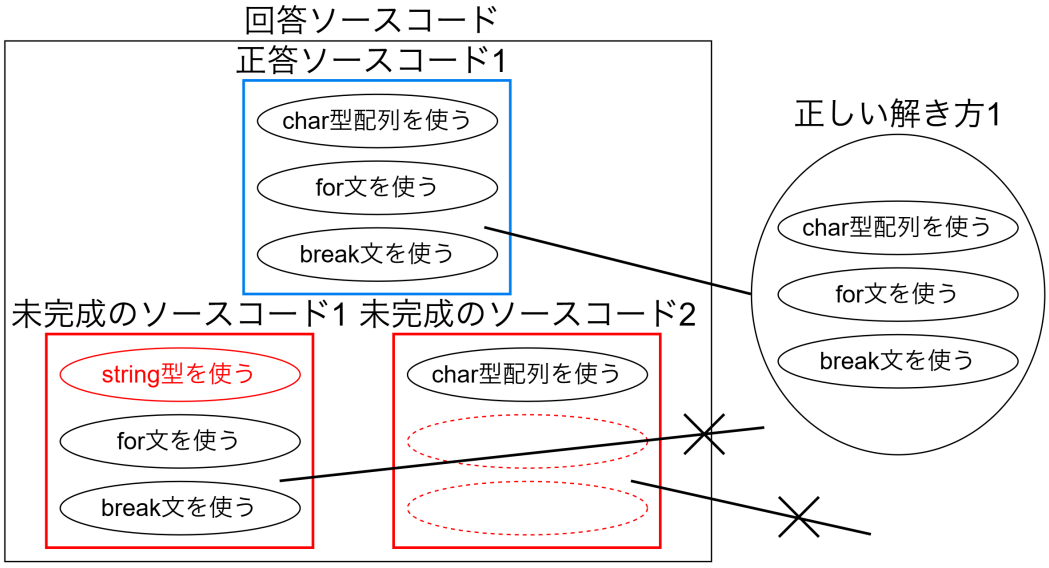

図 2 正答ソースコード, 未完成のソースコードの正しい解き方との関係の具体例

また, 前提として解き方を機械上で捉えることができるかは明らかではない. よって検証した手法で解き方を機械上で捉えられなかったとき, 手法が不適切なだけで改善すれば成功するのか, 本質的に解き方を機械上で捉えることが不可能なのか区別できない問題がある. 本質的に解き方が捉えられない場合はどのソースコードを対象にしてどんな手法をとったとしてもこの問題に陥るが, 上記で示したように, 未完成のソースコードを対象にすると難易度が高く手法にも工夫が必要であるため, 適切な手法をとることが難しく, この問題に陥る可能性

が正答ソースコードを対象にする場合よりも高くなる. そのため, 難易度の高い未完成のソースコードを対象にしてこの問題に陥る可能性を増やすのではなく, まず難易度の低い正答ソースコードを対象にして解き方を機械上で捉えることができるのかを明らかにし, 可能な場合手法を評価するという段階を踏んだ後に対象を未完成のソースコードまで広げるのが妥当である. よって, 本研究では上記のシステムを開発する第一段階として, ソースコードの分散表現のcos類似度, 階層型クラスタリングによって正答ソースコード間の解き方の類似性を評価できるか実験を行う. 分散表現とは文章や単語, ソースコードなどの意味を固定長のベクトルとして表現することである. 意味はただの記号としての単語などとは違い, 意味同士で相対的な距離を持ち, それは以下に示すように様々な側面からの距離となる.

man-king(性別の側面で近い, 偉さの側面で遠い)

man-woman(性別の側面で遠い, 偉さの側面で近い)

そのような意味の距離はスカラー量では表現できないため, 多次元ベクトル空間で表現する. 例えば単語であれば, 意味の近い単語は同じような文脈で使用されやすいという前提のもと, 周辺単語の出現率をベクトルに反映することにより意味を捉える手法が存在する(word2vec[1]). このような分散表現にソースコード内の意味を持った単語(型名や制御フローを表すifやforなど)のデータ構造やアルゴリズムに関する情報を反映することができれば, データ構造やアルゴリズムの組み合わせによって表現できる解き方の類似性をベクトルの類似度により表現できると考えられるため, 本実験では分散表現を用いたシステムの開発を試みる. ソースコードの分散表現と解き方の関係を分析することで, プログラミング問題の解き方の類似性を評価するための新たな知見(分散表現に反映されやすい, 反映されにくい解き方等の情報)も得られると考える. ソースコードを分散表現に変換する手法として, 抽象構文木を用いてソースコードの構造を分散表現に反映できるcode2vec[3]が挙げられるが, 本研究で最終的に入力の対象とする未完成のソースコードはコンパイル可能な状態であるとは限らず, 抽象構文木への変換が行えることが保証できない. そのため単語ごとの分散表現を得ることができる手法であるword2vecを用いてソースコード中のトークン(プログラム内で意味を成す最小の文字列)の分散表現を算出し, 平均値をソースコードの分散表現として扱う. また, word2vecの学習モデル生成, 分析のための検証データには, Project CodeNet[8]のデータを使用する.

以下, 2章では関連研究について述べ, 3章で準備として分散表現, 階層型クラスタリング, Project CodeNetについて説明し, 4章で実験設定と手順を示す. 5章では実験の結果と考察を論じ, 6章では本研究のまとめと今後の発展について記述する.

2 関連研究

プログラミングの学習支援を目的としてソースコードの分散表現を用いた研究として北庄司らの研究[2]がある．北庄司らはソースコードから生成される抽象構文木を元に分散表現を出力する手法であるcode2vec[3]を用いて，入力したソースコードの構造に類似した解答コードを提示し，プログラミングの学習支援を行うシステムを構築した．構築したシステムを用いたソースコードの改善効率や理解しやすさを被験者実験により評価した結果，システムを用いたコード改善が必ずしも難易度を下げるわけではないことが示され，類似したソースコードを提示することが改善の容易さに直結しない場合があることが分かった[2].

北庄司らの研究は，ソースコードの分散表現を用いたプログラミングの学習支援システムを構築している点で本研究と類似している．一方で，学習者がソースコードを作成する際の解き方については言及されておらず，分散表現がソースコード内のどの特徴を反映しているのか分析されていないため，類似度は高いが解き方が異なるソースコードを提示している可能性がある．また，北庄司らが分散表現の算出に用いたcode2vecは，抽象構文木への変換が可能な（コンパイルが可能な）ソースコードしか対象とできない．このことは文法エラー等でコンパイル不可能なコードの作成しかできなかった学習者をシステムの適用範囲外としてしまう問題がある．

本研究は学習者が作成したコンパイル不可能なコードも含めた未完成のソースコードを対象に，解き方の類似性に着目したソースコードの提示を行うシステムの開発を行うことを長期的な目標としている．解き方の類似性に着目したソースコードの提示を目的としている点と，code2vecではなくword2vecを用いることによりコンパイル不可能な未完成のコードも対象とする点で，北庄司らの研究とは異なる．

また，北庄司らの研究で分かった類似したソースコードを提示することが改善の容易さに直結しない問題について，上記で述べたように類似度は高いが解き方が異なるソースコードを提示している可能性があるため，その場合本研究で最終的に構築するシステムにこの問題は関わらない．加えて類似度は高いが解き方が異なるソースコードの提示でない場合にこの問題が起こっていたという仮定を置いた上でも本研究の価値はなくならない．改善の効率が下がったとしても，解き方の類似したソースコードを示すことで自分の解答がどこまで正答に近かったかや不足点がわかるというメリットがあるためである．このような効果的なフィードバックを行うためには解き方の類似したソースコードを示すことが有効となる．また，授業の補助としてこのシステムを使用し，授業で解説された正答例と入れ替えるのではなく，新たな正答例も加えて示す，という形をとることで改善の効率を求める場合は必要に応じて自分の解き方と授業で解説された

解き方を見て，コードの長さや単純さで実装効率の良い方を選択することができる．したがって，北庄司らの研究で分かった問題は本研究において考慮する問題にならないと考えられる．

本研究はcode2vecを用いず，word2vecを用いることでコンパイル不可能なソースコードへの適用の道を潰さずに，正答ソースコードの解き方に関する情報を機械上で捉える手法を提案し，その有用性を確認するために分散表現を用いて算出した類似度と解き方の関係性を分析する．本実験でcode2vecを用いないソースコードの分散表現の取得法の有用性が確認できれば，コンパイル不可能なソースコードを含む未完成のソースコードに対して，解き方が類似した正答ソースコードの提示に繋げることができる．これによって文法エラー等で正答コードの作成ができなかった学習者に対しても開発したシステムを適用できるため，北庄司らの開発したシステムに比べ，システムを適用できない学習者の数を減らすことができる．

3 準備

3.1 分散表現

3.1.1 概要

分散表現とは単語や文，ソースコードなどの意味を持ったドキュメントに関して，その意味をベクトル空間で表現することである．1章でも説明したように意味同時は様々な側面における距離を持つ．図3に1章の例と同様にman,woman,king,queenの意味を表したベクトルを示す．

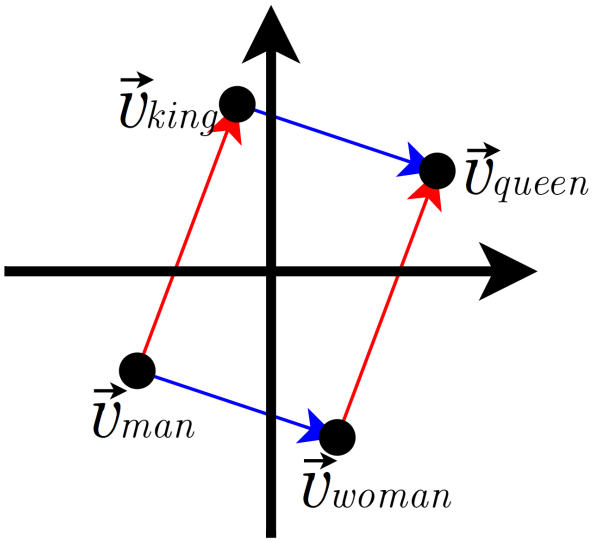

図3 分散表現の例

赤いベクトルは$\vec{v}_{king} - \vec{v}_{man}$で得られるベクトルであり，偉さの側面における距離を表す．青いベクトルは$\vec{v}_{woman} - \vec{v}_{man}$で得られるベクトルであり，性別の側面における距離を表す．このように様々な側面における意味がベクトルとして数値で表現できるため，加算減算の処理ができ，$\vec{v}_{woman} + \vec{v}_{king} - \vec{v}_{man} = \vec{v}_{queen}$のように意味の抽出や意味の付与が行える．主に自然言語処理の分野で用いられるが，プログラミング系の分野においても，code2vecによって得られた分散表現を利用することで，ある関数の振る舞いを示す名前の予測[3]や，コードクローン検出[4]などで良い結果を示している．

3.1.2 word2vec

word2vec[1]はニューラルネットワークを介して単語に対する分散表現を取得する手法である．この手法は単語の文字自体は自身の意味を形成せず，その周辺の単語により意味が形成されるという分布仮設に基づいている．そのため，この手法は周辺単語の出現頻度を基に単語ベクトルの学習を行う．周辺単語の予測タスク

には，ある1つの単語の予測に周辺単語を使用するCBOWと，ある1つの単語からその周辺単語を予測するskip-gramの2種類のタスクが存在する．図4に本研究でも使用しているCBOWのプロセスの一例を示す.

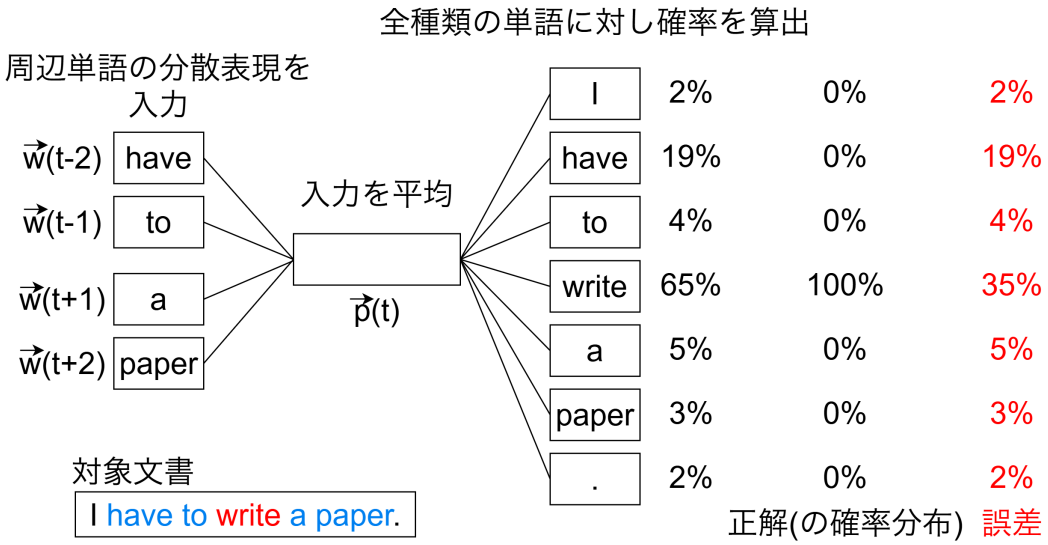

図 4 CBOW のプロセス例

ここでは，”I have to write a paper”を全体の文書とし，”write”の予測を行う．但し，前後2単語までを周辺単語として扱う．左の層では周辺単語の(現状で学習している)分散表現を入力しており，中間の層では入力された分散表現を平均し，予想単語の分散表現とする．右の層では，まずこの分散表現と文書内の全種類の単語の分散表現でそれぞれ内積をとり，その結果をsoftmax関数で確率分布に変換する．softmax関数とは以下の式で定義され，数値が大きいほど高い確率に変換される．

$$\mathrm{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \quad (i = 1, 2, \ldots, n) \tag{3.1.1}$$

分散表現が近い(意味が近い)単語の内積が大きくなるため，予想単語に分散表現が近い単語の確率が高くなる．このようにして予想された各単語の確率と正解の確率(writeが100%，他が0%)の誤差を0に近づけるように入力の周辺単語の分散表現を調整することで学習を進めていく.

Mikolovらの論文[1]では，CBOWは学習における計算量が小さく，頻出単語のベクトル表現を正確に行え，Skip-gramは小規模なデータセットにおいても高いパフォーマンスを示し，出現数の少ない単語のベクトル表現をより正確に行えることがわかっている．3.1章でも示したように単語の意味を表す分散表現に対して加算減算の処理ができるため，言語間の距離を計算して他言語の同じ意味の単語を計算することも可能となり，機械翻訳にも応用されている[5]．また，プログラミング系の分野においてもデザインパターン認識で良い結果を示している[6]．ここ

ではソースコード内の各単語の分散表現を取得した後にその平均をソースコードの分散表現として扱う手法が採られており, 本研究においてもこの手法を用いてソースコードの分散表現を得る.

3.2 階層型クラスタリング

階層型クラスタリングとは, データを階層的にグループ化するクラスタリング手法である. この手法は, クラスタ間の距離に基づいてクラスタを逐次的にクラスタにまとめていき, 最終的にすべてのデータが1つのクラスタにまとめられるまで進める. 各データ(データ数1のクラスタ)の距離はユークリッド距離で定義されるが, データ数が1同士以外のクラスタ間の距離を測る手法は様々ある. 以下に一般的に用いられる手法であるウォード法[7]のアルゴリズムを示す.

1. 各データをデータ数1のクラスタとする.
2. データ数1の各クラスタ間の距離を, 各クラスタ内のデータ間のユークリッド距離として求める. ユークリッド距離は以下の式で定義され, $\vec{x}$, $\vec{y}$は各クラスタ内のデータであり, nはデータの次元数, x_i, y_iは$\vec{x}$, $\vec{y}$のi番目の成分である.

$$\mathrm{d}(\vec{x},\vec{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.2.1}$$

3. クラスタ間の距離が最小のペアを結合し, 新たなクラスタとする.
4. 新たなクラスタと他のクラスタ間での距離を以下の式で求める. 但し, X, Y, Zはクラスタを表し, $X \cup Y$はX, Yが結合された新たなクラスタを表す. n_XはクラスタXのデータ数を表し, $\mathrm{D}_{X,Y}$はクラスタX, Yの距離を表す.

$$\mathrm{D}_{X\cup Y,Z} = \sqrt{\frac{n_X + n_Z}{n_X + n_Y + n_Z}\mathrm{D}_{X,Z}^2 + \frac{n_Y + n_Z}{n_X + n_Y + n_Z}\mathrm{D}_{Y,Z}^2 - \frac{n_Z}{n_X + n_Y + n_Z}\mathrm{D}_{X,Y}^2} \tag{3.2.2}$$

5. 最終的にクラスタが1つになるまで3, 4を繰り返す.

この手法はクラスタ内のデータのばらつきを表す指標である誤差平方和SSEを抑えるように結合が行われる. 誤差平方和SSEは以下の式で定義される. 但し, Xはクラスタ, nはクラスタのデータ数, $\vec{x}_i$はクラスタ内のデータ, $\vec{\mu}_X$はクラスタの重心を表す.

$$\mathrm{SSE}_X = \sum_{i=1}^{n}\mathrm{d}_(\vec{x}_i, \vec{\mu}_X)^2 \tag{3.2.3}$$

このプロセスの例を示す. 図5のような2次元データ群を考える.

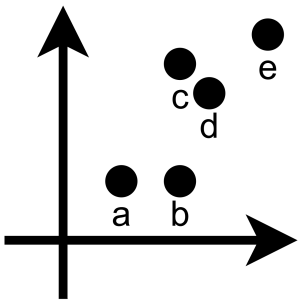

図 5 2次元データ群

この階層型クラスタリングのプロセスを図示したものを図6に示す. 但し, {a,b}はデータにa,bを持つクラスタを表す.

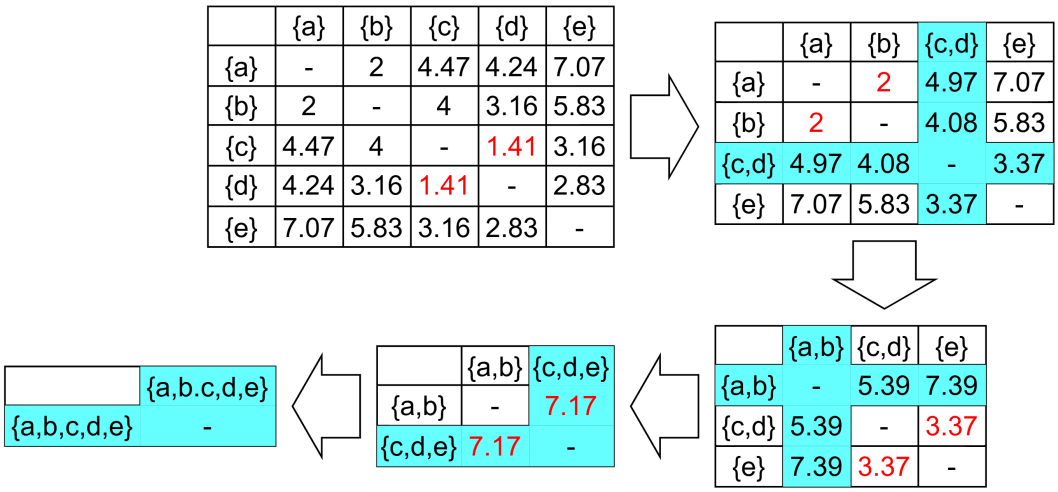

図 6 ward法を用いた階層型クラスタリングの例

まず, 手順1で求めた距離(ユークリッド距離)を左上の6×6の表に示している. この表から, 赤字で示したc,d間の距離が一番小さいことがわかるため, 結合して新たなクラスタc,dを作り, 次の表に新たなクラスタc,dと他のクラスタ間の距離を式(3.2.2)で求めて更新する. この処理を繰り返すと図に示すように最終的に1つのクラスタに全てのデータが結合されるため, これで階層型クラスタリングは終了である.

このような階層型クラスタリングのプロセスを1つの図で表したものがデンドログラム(樹形図)である. 図6のプロセスを表したデンドログラムを図7に示す. 縦軸はクラスタ間の距離を表し, 横軸は結合前のデータ数1のクラスタ(各データ)を表している. その横軸の数字から伸びている枝が他の枝と交わっている箇所がクラスタ結合箇所であり, その箇所の縦軸の値が結合されるクラスタ間の距離である. 他の枝と交わった箇所の上に伸びる一本の枝は結合後のクラスタを表

す. 結合箇所を下から上に見ていくと, 図6での結合の順番に対応している.

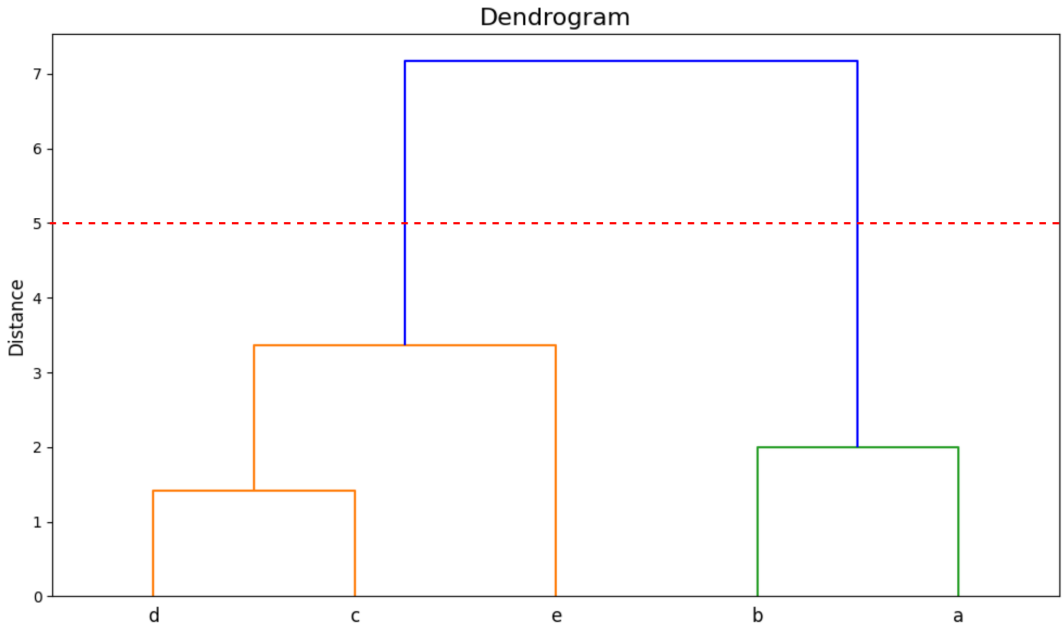

図7デンドログラムの例

また, 最終的なクラスタリング結果を得るためには上記で説明した手順に加えて閾値を与える必要がある. 閾値には結合するクラスタ間の距離の上限を与える. 例えば閾値を5とすると図6では5つめの結合(距離7.17)が行われないため, {a,b},{c,d,e}というクラスタリング結果が得られる. また, 図7では縦軸の5の箇所に横軸に平行な線(図7の赤の点線)を引くと, その交点がクラスタを表し, その交点に枝を伸ばす横軸のデータがクラスタに含まれるデータとなる.

本研究では, 解き方の間の類似性と分散表現の近さの関係について分析するためにデンドログラムを使用する.

3.3 Project CodeNet

Project CodeNet[8]は約1,400万のソースコードを含む大規模なデータセットである. 各ソースコードは2つのオンラインジャッジシステムAIZU Online JudgeとAtCoderに出題された4,000問のプログラミング問題に対する, 利用者の回答である. ソースコードのstatusとして「正解ソースコード」,「不正解ソースコード」(実行は可能),「実行時エラー」,「実行時間超過」,「コンパイルエラー」等が定義されソースコードに割り振られている. 前述しなかったstatusをその他として図8にソースコードに割り振られたstatusの比率を示す.

ソースコードは50を超えるプログラミング言語で記述されており, 主な言語はC++, Python, Java, C, Ruby, C#である. 図9に全ソースコードに対する各言語

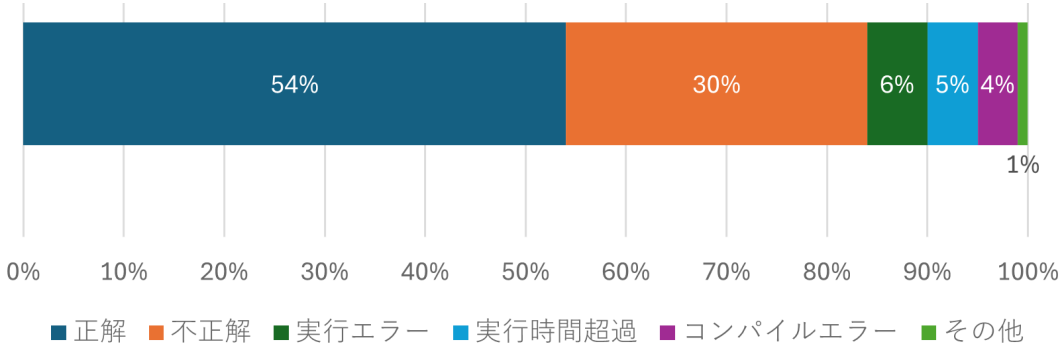

図 8 ソースコードの status の比率

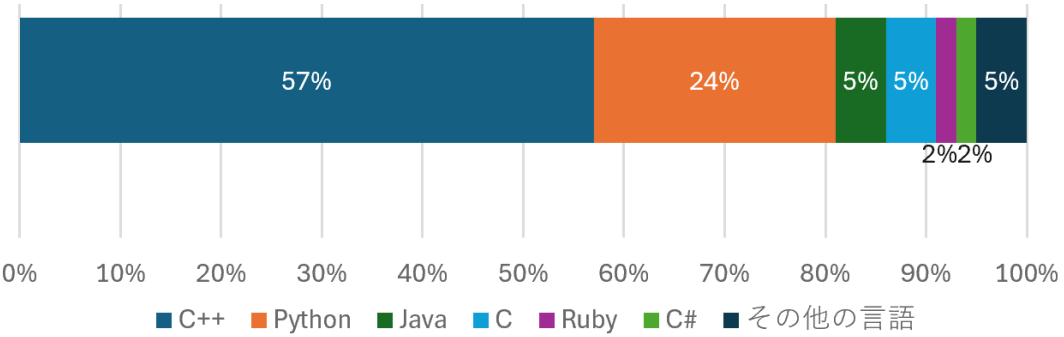

図 9 各言語の比率

の比率を示す. 本研究では Java で記述され status が「正解ソースコード」,「不正解ソースコード」のソースコードを使用する.

Hit and Blow

Let's play Hit and Blow game. A imagines four numbers and B guesses the numbers. After B picks out four numbers, A answers:

・The number of numbers which have the same place with numbers A imagined (Hit)
・The number of numbers included (but different place) in the numbers A imagined (Blow)

For example, if A imagined numbers:

9 1 8 2

and B chose:

4 1 5 9

A should say 1 Hit and 1 Blow.

Write a program which reads four numbers A imagined and four numbers B chose and prints the number of Hit and Blow respectively. You may assume that the four numbers are all different and within from 0 to 9.

Input

The input consists of multiple datasets. Each dataset consists of:

$$\begin{array}{cccc} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \end{array}$$

where $a_i\,(0 \leq a_i \leq 9)$ is the i-th number A imagined and $b_i\,(0 \leq b_i \leq 9)$ is the i-th number B chose.

The input ends with EOF. The number of datasets is less than or equal to 50.

Output

For each dataset, print the number of Hit and Blow in a line. These two numbers should be separated by a space.

Sample Input

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

Output for the Sample Input

```
1 1
3 0
```

図 10 学習データ内の課題例 (p00025)

4 実験

4.1 学習データ

CodeNet のソースコードのうち Java で記述され, かつ, status が「正解ソースコード」,「不正解ソースコード」のいずれかであるソースコード, 3299 個の課題に対する回答ソースコード 551,640 ファイルを学習データとして使用する. Java 言語は本校 (奈良高等工業専門学校) の情報工学科のカリキュラムで初めてプログラミング課題が課されるような講義が行われる言語であり, 扱える学生が多いことによる本校の情報工学科での被験者実験のしやすさ, 講義への適用のしやすさから選択した. 今回の学習データの例として, 課題 p00025 とそれに対する回答ソースコードを図 10, 図 11 に示す. この課題 p00025 に対する回答コード数は 220, 平均行数は 39.89 行である.

```java
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Main
{
    public static void main(String[] args) throws IOException
    {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    String s, s1;
    while((s=br.readLine())!=null)
    {
      int[] a = new int[4], b = new int[4];
      int hit=0, blow=0;
      for(int i=0; i<4; i++) a[i] = Integer.parseInt(s.split(" ")[i]);
      s1=br.readLine();
      for(int i=0; i<4; i++) b[i] = Integer.parseInt(s1.split(" ")[i]);

      for(int i=0; i<4; i++){
        for(int j=0; j<4; j++){
          if(a[i] == b[j] && i==j) hit++;
          else if(a[i] == b[j])   blow++;
        }
      }
      System.out.println(hit + " " + blow);
    }
    }
}
```

図 11 p00025 に対する回答例 (s197886488.java)

4.2 前処理

モデル学習を行うために学習データの全 Java ファイルを 1 つの txt ファイルに変換する. 前処理の実行例を図 12 に示す. 学習データであるすべてのソースコードに対して以下の処理を行う. まずソースコードを単語トークンに分割し, 動作に影響を与えない空白文字と改行文字は, 動作に関する意味を含むトークンの分散表現をソースコードの分散表現により強く反映するために削除する. (空白文字は単語トークンの分割には使用されるが, それ自体を単語トークンとしては使用しない.) 次に, リテラル (定数) について, 異なるソースコード間で同じ単語が現れることが多い一方で, ソースコードごとに意味が異なる可能性が高い. 例えば, bool 型のリテラルである True に対して, while 文の判定式に使われていればループ処理を続ける意味になるが, for 文の中で break 文を内包する if 文の判定式に使われていればループ処理を終了する意味になり, 真逆の意味となる. しかし, 今回の単語のソースコード内の各単語の分散表現を取得した後にその平均をソースコードの分散表現として扱う手法では, 分散表現は 1 種類のトークンで共通して同じものが学習され, ソースコードの分散表現を得るのに使用するため, トークンに対して普遍的な学習が必要となる. よってリテラルを削除する. そして, コメントについてはプログラミング言語というより自然言語の意味を持ち, ソースコードの動作に影響を与えないため削除する. 最後に, 残ったすべてのトークンを半角スペース区切りで txt ファイルに書き込み, ソースコード毎に改行文字を追加する.

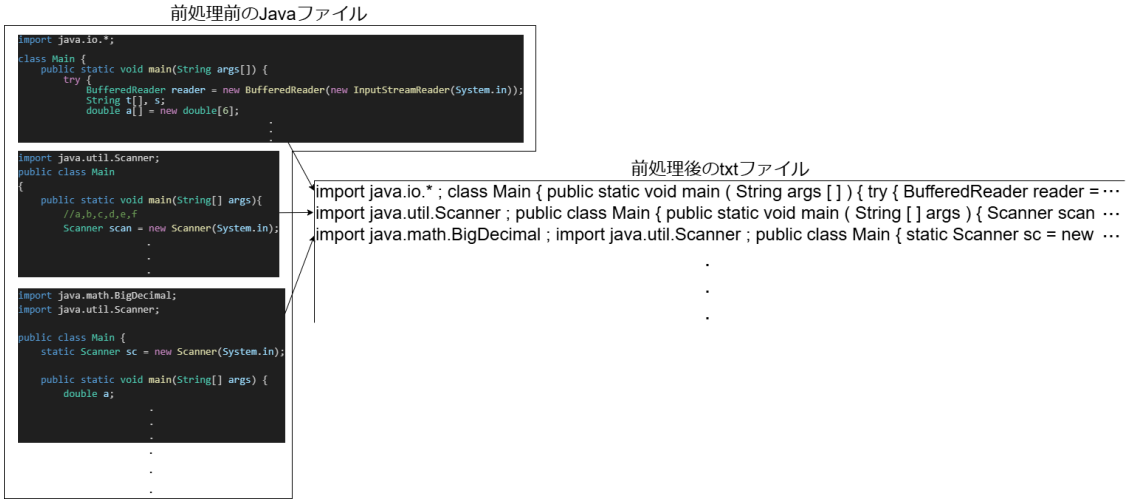
前処理前のJavaファイル
import java.io.*;
class Main {
public static void main(String args[]) {
try {
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
String t[], s;
double a[] = new double[6];
import java.util.Scanner;
public class Main
{
public static void main(String[] args){
//a,b,c,d,e,f
Scanner scan = new Scanner(System.in);
import java.math.BigDecimal;
import java.util.Scanner;
public class Main {
static Scanner sc = new Scanner(System.in);
public static void main(String[] args) {
double a;
前処理後のtxtファイル
import java.io.* ; class Main { public static void main (String args [ ]) { try { BufferedReader reader = …
import java.util.Scanner ; public class Main { public static void main (String [ ] args) { Scanner scan …
import java.math.BigDecimal ; import java.util.Scanner ; public class Main { static Scanner sc = new …

図 12 前処理の実行例

4.3 学習モデルの生成

本実験ではgensimライブラリ(vesion4.3.3)のword2vecモジュールを使用して学習モデル生成を行う. モデル生成時に学習のためのパラメータの設定を行う. パラメータとは処理の内容を動的に決める目的で外側から与える値である. ライブラリを使用して学習を行うとき, ライブラリ内の処理はカプセル化されているため, パラメータの調整で動作を制御する. 各パラメータの説明と設定値を表1, 表2に示す.

表1 各パラメータの説明と設定値

パラメータ	説明	設定値
sentences	学習に使用するデータ. スペースで単語が区別され, 改行で別データとして区別される.	前処理後の学習データ
vector_size	単語ベクトルの次元数.	100(デフォルト値)
window	対象単語に対して周辺単語として扱う最大単語距離.	5(デフォルト値)
shrink_windows	このパラメータがTrueの場合, 対象単語に対して周辺単語として扱う単語距離が[1,window]の範囲からランダムに選択される. falseの場合, 対象単語に対して周辺単語として扱う単語距離はwindowの値に固定される.	True(デフォルト値)
min_count	この数値以下の出現数の単語は無視する.	5(デフォルト値)
sg	学習アルゴリズムの選択. 0がCBOW, 1がskip-gram.	0(デフォルト値)
cbow_mean	CBOWでの予測単語のベクトルを周辺単語の合計を用いて出力するか, 平均を用いて出力するかを決める値. 0で合計, 1で平均が用いられる. sg=0で学習アルゴリズムにCBOWを使用する際のみ適応される.	1(デフォルト値)
hs	各単語の確率を出力するための活性化関数の選択. 0が通常のsoftmax関数, 1がHierarchical Softmax関数を使用.	0(デフォルト値)

表2各パラメータの説明と設定値(続き)

パラメータ	説明	設定値
negative	ネガティブサンプリングで分散表現の更新に使用する不正解単語の数. 1つの正解タスクに対してこの数値の不正解単語がサンプリングされ使用される. 0の場合ネガティブサンプリングの考え自体を使用しないため, 分散表現の更新に使用するのは正解, 不正解関係なく全語彙となる.	5(デフォルト値)
ns_exponent	不正解単語のサンプリングの確率密度関数の定義に使用される数値. 1のとき, 不正解単語の学習データ全体での出現頻度に正確に比例してサンプリング確率が決定され, 0のときランダムにサンプリングされ, 負の値の場合不正解単語の出現頻度が低い単語が出現頻度が高い単語よりも高い確率でサンプリングされる.	0. 75(デフォルト値)
alpha	学習率の初期値.	0.025(デフォルト値)
min_alpha	最終的な学習率. 学習の進行に合わせalphaの値からこのパラメータの値に線形に減少する.	0.0001(デフォルト値)
sample	頻度の高すぎる単語をダウンサンプリングする(間引いて使用しない)確率を決定する値. この値が小さいほどダウンサンプリングの確率が上がり, 頻度の高すぎる単語情報の過学習を抑制する.	0.001(デフォルト値)
hashfxn	重みをランダムに初期化するために使用するハッシュ関数.	pythonに組み込まれたハッシュ関数(デフォルト値)
epochs	学習データに対して, 繰り返し学習する回数. (エポック数)	15
batch_words	学習時の単語単位のバッチサイズ	10000(デフォルト値)

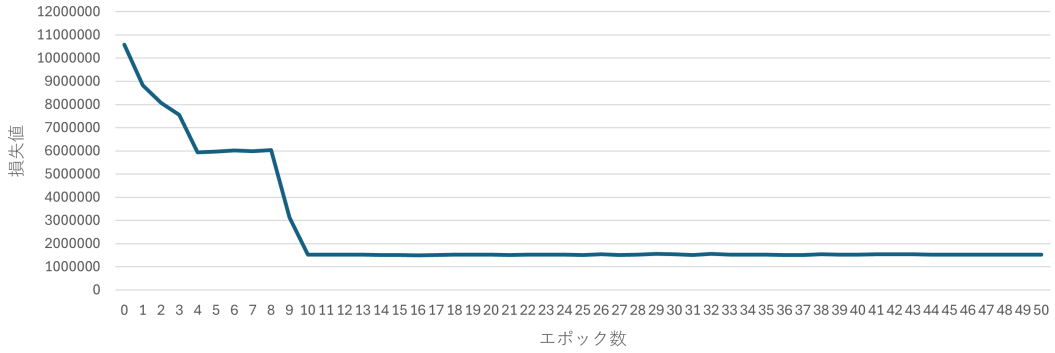

図 13 各エポック数における損失値の変化

本研究では学習データである sentences に 4.1 節，4.2 節で述べた前処理を行った txt データを指定する．エポック数はデフォルト値では学習総データ量 (学習データ × エポック数) が十分な量であるかが不明である．そのため，エポック数を 1 から 1 ずつ 50 まで変化させたときの学習時の損失値の変化を記録し，変化が十分に収束した値を用いる．各エポック数における損失値の変化を図 13 に示す．図 13 より，エポック数 10 以降で損失値の変化が十分に収束していることが確認できる．しかし，word2vec の学習における損失値はいつ学習させても一定というわけではないため，収束する前の epoch=9 に近い値で収束することの再現性は担保できない．また，epoch 数を増やしすぎることによるモデルの過学習を避けるための適当な値として epochs=15 を採用する．他のパラメータについては gensim の公式ドキュメントに記載されたデフォルト値を設定する．

National Budget

A country has a budget of more than 81 trillion yen. We want to process such data, but conventional integer type which uses signed 32 bit can represent up to 2,147,483,647.

Your task is to write a program which reads two integers (more than or equal to zero), and prints a sum of these integers.

If given integers or the sum have more than 80 digits, print "overflow".

Input

Input consists of several datasets. In the first line, the number of datasets N (1 $\leq$ N $\leq$ 50) is given. Each dataset consists of 2 lines:

The first integer
The second integer

The integer has at most 100 digits.

Output

For each dataset, print the sum of given integers in a line.

Sample Input

```
6
1000
800
99999999999999999999999999999999999999
1
99999999999999999999999999999999999999999999999999999999999999999999999999999999
1
99999999999999999999999999999999999999999999999999999999999999999999999999999999
0
10000000000000000000000000000000000000000000000000000000000000000000000000000000
1
10000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Output for the Sample Input

```
1800
10000000000000000000000000000000000000
overflow
99999999999999999999999999999999999999999999999999999999999999999999999999999999
overflow
overflow
```

図 14 検証データとして採用した課題

4.4 検証データ

本研究で開発するシステムが対象とするプログラム課題は，指定の動作を実現する解き方が複数存在し，一部の解き方の解説だけでは学習者の効率的な学習に不十分となる課題である．実験では条件を満たす課題としてCodeNet内に存在する図14の課題National Budgetを採用する．この問題は最大100桁の整数の加算を行い，合計が80桁を超える場合に「overflow」を出力するプログラムを作成するものである．Javaのint型やlong型では扱うことのできない桁数の数値を処理する方法を考えてコーディングする必要がある．

表3 正答に対する解き方によるラベルの定義

ラベル	解き方
BigInteger	BigIntegerを使用する.
BigDecimal	BigDecimalを使用する.
桁ごと	桁ごとに分割して計算する.
garner	garnerのアルゴリズムを使用して計算し, BigIntegerで出力する.

National Budgetの課題には大きく分けて以下の解き方が存在する.

1. 多倍長演算のライブラリを使用する.
   (a) BigIntegerを使用する.
   (b) BigDecimalを使用する.
2. 桁ごとに分割して計算する.
3. garnerのアルゴリズムを使用して計算し, BigIntegerで出力する.

これらの解き方をした回答に対して表3のラベルを定義する. この課題は, 指定の動作を実現する解き方が複数存在し, 多倍長演算のライブラリを使用する「BigInteger」と「BigDecimal」のラベルのグループ,「桁ごと」のラベル,「garner」のラベルの間で大きく実装方法が異なり一部の解き方の解説だけでは学習者の効率的な学習に不十分となる. よってこの課題は本研究で開発するシステムが対象とするプログラム課題であると言え, このデータを用いた検証は最終的な研究目標であるシステムの開発に繋がると言える. 本実験では課題National Budgetに提出され, 正解した228個のJavaファイルを検証データとして使用する. 本稿では以降, このラベルを使用した記述を行う.

4.5 分析

4.5.1 cos類似度による分析

cos類似度による分析ではすべての検証データに対して分散表現を求めた後，4種類の解き方で解かれたソースコードそれぞれ2個を著者が目視で抽出し，計8個のソースコードとすべての検証データの分散表現のcos類似度を計算する．抽出した8個のソースコードに対してcos類似度が高い上位5つのソースコードの解き方を記録し，その傾向について分析する．ここで抽出するコード数や解き方を記録するソースコードの数は多い方が正確な分析が行えるが，目視が必要なことを考慮し，研究期間中に分析可能な値としてサンプル数を増やし過ぎずに上記の値を採用した．具体的な手順は以下の通りである．

1. 4.1節で示した学習データ, 4.4節の検証データに対して4.2節で述べた前処理を行う.
2. gensimのword2vecモジュールを利用して4.3節のパラメータ(sentences=前処理後の学習データ)でモデルの学習を行う.
3. 学習したモデルで前処理後の4.4節の検証データの各ソースコードに対して以下の処理を行う.
   (a) 2で学習したモデルでソースコードの各トークンの分散表現を取得する．
   (b) 各トークンの分散表現の平均をソースコードの分散表現として保存する．
4. ラベル「BigInteger」,「BigDecimal」,「桁ごと」,「garner」のソースコード2つをそれぞれ検証データの中から用意する．
5. 用意した解答の分散表現に対し，自分以外の全ての検証データの分散表現とのcos類似度を計算する．
6. 用意したソースコードそれぞれに対し，cos類似度が高い上位5つのソースコードの解き方を記録する．

3から6までの流れを抽出したBigIntegerのコード2つのうち1つに適用した分析例を図15に示す. 但し，分散表現は実際は100次元であるが表現できないため2次元で表しており, 実際のベクトル空間には検証データ全体に対する228個の分散表現が存在する．

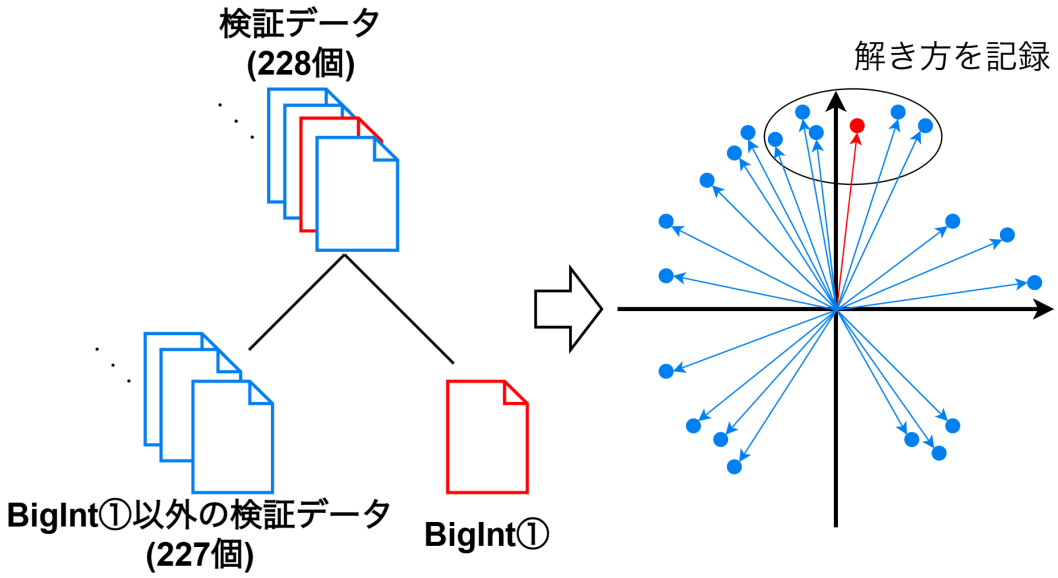

図 15 分析例

4.5.2 階層型クラスタリングによる分析

階層型クラスタリングによる分析ではソースコードの分散表現による階層型クラスタリングを行い，そのプロセスを表すデンドログラムを用いてどのような解き方のソースコードがどの順番で同じクラスタに結合されるのかを確認する．その結果を用いて分散表現の類似性によって解き方の類似性を評価できるのかについての分析を行う．しかし，全検証データのソースコード228個の解き方を目視で確認した上で，その結合の様子を分析することは時間の見積もりが立たない上に，どこまで細かい解き方の差異を目視で捉えられるかは不明のため，効果的な分析が行える保証はない．よってクラスタ数は分析に十分で現実的な数で，解き方を目視するは各クラスタから現実的な数のサンプル(5つ)に対して行う方法をとる．その上でクラスタの結合を観測するのに十分なクラスタ数を考えると，ラベル数に合わせた4のクラスタ数が挙げられる．よってクラスタ数4以上かつ，現実的な時間，難易度で分析が行えるクラスタ数として，クラスタ数10前後となる閾値を決定する．この閾値の決定をするために，一度クラスタリングを実行し，閾値0までのデンドログラムを得る．そのデンドログラムでクラスタ数が10前後となっているところの縦軸の値を閾値としてクラスタリングを再度実行し，クラスタリング結果とその閾値以降のデンドログラムを得る．この方法でサンプリングしたソースコードの解き方の傾向を基にそのクラスタに属するソースコードの傾向を取得し，それ以降のクラスタの結合の様子について分析する．具体的な手順は以下の通りである．但し，検証データの各ソースコードの分散表現を得るまでの手順は4.5.1と同様であるため，同様のものを使用する．

1. 検証データの各ソースコードの分散表現を閾値0でウォード法を用いた階層型クラスタリングを行い，プロセス全体を表したデンドログラムを得る．
2. 得られたデンドログラムから，クラスタ数が10前後となっている箇所の縦軸の値を閾値として再度クラスタリングを行い，クラスタリング結果とその閾値より上の結合に関するデンドログラムを得る．
3. 各クラスタに属するソースコードの中から代表コードとしてランダムに5つのソースコードをサンプリングする．クラスタ内のデータ数が5つ未満の場合，クラスタ内のデータ全てを代表コードとして扱う．
4. 代表コードの解き方を著者が目視で確認し，各クラスタに属するソースコードの傾向を取得する．
5. デンドログラムからどのクラスタがどのクラスタとどの順番で結合するかを確認する．

5 結果と考察

5.1 cos類似度

5.1.1 BigInteger

BigIntegerのソースコードを図16, 図17に示す. 図16のソースコードは, まず加算対象の整数の読み込み回数を入力からint型で受け取り, その回数分入力からBigInteger型で2つの整数を受け取り, その加算結果が80桁を超えていれば”overflow”を出力, そうでなければ加算結果を出力するプログラムである. 図17のソースコードも基本は同様の処理だが, 加算対象の整数を入力からstring型で受け取った後, BigInteger型に変換する点と, 加算結果をstring型に変換して加算結果が80桁を超えているのかを判定している点, mainメソッドに処理を直接記述するのではなく, doItメソッドに処理を記述し, それをmainメソッドら呼び出している点が図16のソースコードと異なる. 表4に抽出した2つのソースコードそれぞれに対してcos類似度が高い上位5つのソースコードとラベルを示す. どちらのソースコードにおいても, cos類似度の高いすべてのソースコードのラベルがBigIntegerであり, ソースコードの分散表現によって解き方の類似したソースコードを識別できた.

```
1  import java.math.*;
2  import java.io.*;
3  import java.util.*;
4
5  public class Main{
6
7    public static void main(String []args)
8    {
9      Scanner cin=new Scanner(System.in);
10     int n = cin.nextInt();
11     while(n--!=0){
12       BigInteger b=cin.nextBigInteger(), a=cin.nextBigInteger();
13       if(a.add(b).toString().length()>80){
14         System.out.println("overflow");
15       }else System.out.println(a.add(b).toString());
16
17     }
18   }
19 }
```

図 16 BigIntegerのソースコード1(s622946479.java)

```java
import java.math.BigInteger;
import java.util.*;

public class Main {
  Scanner sc = new Scanner(System.in);
  void doIt() {
    int n = sc.nextInt();
    for(int i = 0;i < n;i++){
      String a = sc.next();
      String b = sc.next();
      BigInteger numa = new BigInteger(a);
      BigInteger numb = new BigInteger(b);
      BigInteger all = numa.add(numb);
      String allnum = all.toString();
      if(allnum.length() > 80){
        System.out.println("overflow");
      }else{
        System.out.println(all);
      }
    }
     }
    public static void main(String[] args) {
      // TODO Auto-generated method stub
      new Main().doIt();
    }
}
```

図 17 BigInteger のソースコード 2(s205975462.java)

表 4 cos 類似度の高いソースコード (BigInteger)

対象ソースコード	類似ソースコード	ラベル	cos 類似度
BigInteger 1 (s622946479.java)	s358352316.java	BigInteger	0.989
	s593233163.java	BigInteger	0.987
	s712674763.java	BigInteger	0.987
	s172166742.java	BigInteger	0.984
	s297896256.java	BigInteger	0.983
BigInteger 2 (s205975462.java)	s174218924.java	BigInteger	0.991
	s412711051.java	BigInteger	0.991
	s156248176.java	BigInteger	0.991
	s532254573.java	BigInteger	0.989
	s461638579.java	BigInteger	0.989

5.1.2 BigDecimal

BigDecimalのソースコードを図18に，表5に抽出した2つのソースコードそれぞれに対してcos類似度が高い上位5つのソースコードとラベルを示す. 図18のソースコードは，まず加算対象の整数の読み込み回数を入力からstring型で受け取った後int型に変換してその回数for文を実行する判定式に使用する．for文内では入力からstring型で2つの整数を受け取った後，BigDecimalに変換して，その加算結果を再度string型に変換する．string型の加算結果を判定式に使用し，加算結果が80桁以内であれば加算結果を出力，そうでなければ”overflow”を出力するプログラムである．BigDecimalの対象ソースコード2つに対して8つはBigIntegerのソースコードだった．BigIntegerとBigDecimalの解き方はどちらも多倍長の数の型を用いた解き方であり, 必要となる処理が類似しているため, 出現するトークンも類似し，分散表現としても類似度が高くなったと考えられる．また，BigDecimalとBigIntegerのソースコード数を目視で確認すると，BigDecimalが22，BigIntegerが152だったことから，BigDecimalのデータ数不足による結果ではないかという疑念が生まれる．しかし，BigDecimalのソースコードに対してBigDecimalのソースコードよりも類似度の高いBigIntegerのソースコードが存在する時点で，BigIntegerとBigDecimalのどちらのデータの型を使うのかという解き方の差はデータの型以外の解き方の差よりも分散表現に与える影響が少ないと考えられる．よってデータ数不足によって結果が大きく変わると考えられるのはむしろBigIntegerの結果である．BigDecimal1に対するcos類似度1位は0.989でBigIntegerのソースコードであり，表4の上位5位の類似度の水準と大差がない．BigIntegerとBigDecimalの分散表現の間に大きな距離はないとわかるため，BigDecimalのソースコードを増やすと表4のBigInteger1，BigInteger2に対してデータの型以外の解き方の類似度の高いBigDecimalのコードが表れる可能性がある．したがって，データ数がBigIntegerとBigDecimalで同等の数であれば，お互いの類似度の高いソースコードにお互いが混じり合う形になると考えられるため，今回の手法に関してはBigIntegerとBigDecimalの使用するデータの型に関する解き方の違いを分散表現で評価することは難しいと考察できる．この原因に対する仮説として，BigIntegerやBigDecimalの周辺単語に，扱えるデータが整数か少数かの差を表す情報が含まれておらず, 単語の分散表現に差が生まれなかった説や，データの型を宣言時以外に使用していないかつ，トークン数が多く1トークンが全トークンの平均のソースコードの分散表現に対して与える影響が少なくなり，ソースコードの分散表現が近くなってしまった説が考えられる．一方で，解き方が大きく異なる桁ごと，garnerは上位5件に現れておらず，分散表現のcos類似度でソースコード間の解き方の類似性を一定以上評価できたと言える．

```java
import java.math.BigDecimal;

import java.util.Scanner;
public class Main {

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String n = sc.nextLine();
    for(int i = 0 ; i < Integer.parseInt(n) ; i++){
      String x = sc.nextLine();
      String y = sc.nextLine();
      BigDecimal a = new BigDecimal(x);
      BigDecimal b = new BigDecimal(y);
      String str = a.add(b).toString();
      if(str.length() <= 80){
        System.out.println(str);
      } else {
        System.out.println("overflow");
      }
    }
    sc.close();
  }

}
```

図 18 BigDecimal のソースコード 1(s072033427.java)

表 5 cos 類似度の高いソースコード (BigDecimal)

対象ソースコード	類似ソースコード	ラベル	cos 類似度
BigDecimal 1 (s072033427.java)	s800890666.java	BigInteger	0.989
	s205975462.java	BigInteger	0.988
	s909785568.java	BigDecimal	0.987
	s174218924.java	BigInteger	0.985
	s461638579.java	BigInteger	0.983
BigDecimal 2 (s092763241.java)	s055966518.java	BigDecimal	0.987
	s620443514.java	BigInteger	0.978
	s830364643.java	BigInteger	0.975
	s242122975.java	BigInteger	0.976
	s452663738.java	BigInteger	0.967

5.1.3 桁ごと

桁ごとのソースコードを図 19 に，表 6 に抽出した 2 つのソースコードそれぞれに対して cos 類似度が高い上位 5 つのソースコードとラベルを示す．図 19 のソースコードは，まず加算対象の整数の読み込み回数を入力から int 型で受け取り，その回数次の処理を行う．

1. 入力から string 型で 2 つの整数を受け取り，桁ごとに分割して string 型の配列に入れる．
2. どちらかの整数が 80 桁を超えていたらこの時点で”overflow”を出力し，この整数に対する処理を終了する．

3. そうでない場合，string型の配列に入った整数を桁ごとにint型に変換し，int型の配列(長さ81)のインデックス0からstring型の配列とは逆順に入れる．

4. 桁ごとに入ったint型の配列をインデックス0の値から加算し，桁ごとの加算結果を10で割ってint型に代入(小数点以下切り捨て)した値を次の桁に繰り上げとして加算し，桁ごとの加算結果を10で割ったあまりを最終的なその桁の数とする．

5. 最終的なその桁の数が0でないとき，加算結果の桁数としてその桁を更新する．

6. 5の処理をint型配列のインデックス0から79まで行い，加算結果とその桁数を得る．

7. インデックス79の加算結果に繰り上がりが起こっていてインデックス80の値が0でないとき，”overflow”を出力し，この整数に対する処理を終了する．

8. そうでない場合，加算結果のint型配列をインデックス0の値からstring型の変数の左に追記する処理を加算結果の桁数分行い，string型の変数に入った加算結果を出力し，この整数に対する処理を終了する．

どちらのソースコードにおいても，cos類似度の高いすべてのソースコードのラベルが桁ごとであり，ソースコードの分散表現によって解き方の類似したソースコードを識別できた．

```java
import java.util.*;
public class Main {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int n = s.nextInt();
    for (int i=0;i<n;i++) {
      int [] a = new int[81];
      int [] b = new int[81];
      Arrays.fill(a,  0);
      Arrays.fill(b,  0);
      String [] as = s.next().split("");
      String [] bs = s.next().split("");
      if (as.length > 80 | bs.length > 80) {
        System.out.println("overflow");
        continue;
      }
      for (int j=0;j<as.length;j++) {
        a[j] = Integer.valueOf(as[as.length - 1 - j]);
      }
      for (int j=0;j<bs.length;j++) {
        b[j] = Integer.valueOf(bs[bs.length - 1 - j]);
      }
      int ansLen = 0;
      for (int j=0;j<80;j++) {
        int temp = a[j] + b[j];
        int c = temp / 10;
        int r = temp % 10;
        a[j] = r;
        a[j + 1] += c;
        if (a[j] != 0) ansLen = j;
      }
      if (a[80] != 0) {
        System.out.println("overflow");
        continue;
      }
      String ans = "";
      for (int j=0;j<=ansLen;j++) {
        ans = a[j] + ans;
      }
      System.out.println(ans);

    }
  }
}
```

図 19 桁ごとのソースコード 1(s836663828.java)

表 6 cos 類似度の高いソースコード (桁ごと)

対象ソースコード	類似ソースコード	ラベル	cos 類似度
桁ごと 1 (s836663828.java)	s706626404.java	桁ごと	0.990
	s015207243.java	桁ごと	0.988
	s123911032.java	桁ごと	0.988
	s996705606.java	桁ごと	0.987
	s327971512.java	桁ごと	0.985
桁ごと 2 (s198796521.java)	s780705380.java	桁ごと	0.986
	s582187920.java	桁ごと	0.983
	s706626404.java	桁ごと	0.982
	s600008956.java	桁ごと	0.981
	s016089106.java	桁ごと	0.981

5.1.4 garner

garnerのソースコードを図20に，表7に抽出した2つのソースコードそれぞれに対してcos類似度が高い上位5つのソースコードとラベルを示す．2つの対象ソースコードの上位2つがいずれもgarnerのソースコードであり，cos類似度が0.999以上と極めて高い．ソースコードの内容を確認すると3つのラベル「garner」のソースコード(付録: 図24，図25，図26，図27，図28，図29)は分散表現には影響を与えないコメントの内容まで一致している箇所が多くあり，同じコピー元を持つコードクローン（コピーとペースト等で意図的に類似または一致したコード）であると考えられる．コードクローンがcos類似度で上位となることに問題はないが，同じラベル「garner」かつコードクローンではないコードがコードクローン以下の順位で確認できない．そのため，この結果から今回のモデルでラベル「garner」の解き方で解かれた対象ソースコードを他のラベルのソースコードと区別して提示できるシステムを開発できるとは言えない．ラベル「garner」のソースコードがこのコードクローン以外に存在しない可能性もあるため，データセットとしてコードクローンではないラベル「garner」のソースコードを追加して検証する必要がある．

```java
import java.io.*;
import java.util.*;
import java.math.BigInteger;

class Lib{
    static long extgcd(long a, long b, long[]x) {
        for (long u = x[1] = 1,  v = x[0] = 0; a!=0;){
            long q = b / a;
            long r=x[0]-q*u;
            x[0]=u;
            u=r;
            r=x[1]-q*v;
            x[1]=v;
            v=r;
            r=b-q*a;
            b=a;
            a=r;
        }
        return b;
    }

    static long mod_inv(long a, long m) {
        long[]x=new long[2];
        extgcd(a,  m,  x);
        return (m + x[0] % m) % m;
    }

    static BigInteger garner(long[]a, long[]m){
        int ms=a.length;
        long[]coffs=new long[ms], constants=new long[ms];
        long[]digs=new long[ms];
        Arrays.fill(coffs, 1);
        for(int i=0;i<ms;++i){
            long v = (a[i] - constants[i]) * mod_inv(coffs[i],  m[i]) % m[i];
            if(v<0)v+=m[i];
            digs[i]=v;
            for (int j = i + 1; j < ms; j++) {
                constants[j] += coffs[j] * v;
                constants[j] %= m[j];
                coffs[j] *= m[i];
                coffs[j] %= m[j];
            }
        }
        BigInteger ans=BigInteger.valueOf(0), c=BigInteger.valueOf(1);
        for(int i=ms-1;i>=0;--i){
            c=c.multiply(BigInteger.valueOf(m[i]));
            ans=ans.multiply(BigInteger.valueOf(m[i]));
            ans=ans.add(BigInteger.valueOf(digs[i]));
        }
        if(ans.compareTo(c.divide(BigInteger.valueOf(2)))>0)
            ans=ans.subtract(c);
        return ans;
    }
}
```

図 20 garner のソースコード 1(s678994851.java, 一部)

次に，garner 以外で cos 類似度の高いソースコードについて考察する．garner のアルゴリズムはデータ構造として配列を使用し，繰り返し参照する（図 24, 30〜43 行目）．同様に，桁ごとのソースコードも配列を繰り返し参照する作りになっており（図 19, 17〜39 行目），一部の BigInteger のソースコード（例えば，図 21）についても配列を用いている．ソースコード中に配列が現れる場合，トークンとして”[”と”]”の出現数が増加し，ソースコード全体の分散表現に影響を与える．そのため，garner のソースコードと同様に配列を多用したソースコードが，より類似したソースコードとして識別された可能性がある．これは，トークン単位で分散表現を取得し，ソースコードの分散表現へ変換する際にどのトークンにも重みを付けずに変換することが原因となり，データ構造やアルゴリズムの中で特徴を

表 7 cos 類似度の高いソースコード (garner)

対象ソースコード	類似ソースコード	ラベル	cos 類似度
garner 1 (s678994851.java)	s728758016.java	garner	1.000
	s964736393.java	garner	0.999
	s328143254.java	桁ごと	0.976
	s599188314.java	桁ごと※	0.971
	s599968744.java	桁ごと	0.969
garner 2 (s964736393.java)	s728758016.java	garner	0.999
	s678994851.java	garner	0.999
	s328143254.java	桁ごと	0.972
	s358656626.java	BigInteger	0.972
	s599188314.java	桁ごと※	0.971

※:8 桁ごとに分割後，int 型で加算し桁上げ処理．

持ったトークンが繰り返し出現するものが分散表現への影響を強めている可能性を示唆している．これは分散表現によるデータ構造やアルゴリズムの判別のしやすさに差をもたらす要因となり，解き方の類似性を評価するにあたって最適であるとは言えない．特定の単語への重みの調整やトークンごとではなくソースコードから直接の分散表現の取得などで改善する必要がある．

```java
import java.math.BigInteger;
import java.util.Scanner;


public class Main{

  static Scanner sc = new Scanner(System.in);

  static int n;
  static BigInteger a[], b[];

  public static void main(String[] args) {

    while(read()){
      solve();
    }

  }

  public static boolean read(){
    if(!sc.hasNext()) return false;

    n = sc.nextInt();

    a = new BigInteger[n];
    b = new BigInteger[n];

    for(int i=0; i < n; i++){
      a[i] = sc.nextBigInteger();
      b[i] = sc.nextBigInteger();
    }

    return true;
  }

  public static void solve(){

    for(int i=0; i < n; i++){
      if(a[i].add(b[i]).toString().length() > 80){
        System.out.println("overflow");
      }
      else{
        System.out.println(a[i].add(b[i]));
      }
    }
  }

}
```

図 21 配列を用いている BigInteger のソースコード (s358656626.java)

5.2 階層型クラスタリング

まず, 検証データの各ソースコードに対する分散表現に閾値 0 で階層型クラスタリングを実行して得られたデンドログラムを図 22 に示す.

図 22 から, 閾値 6.5 をとればクラスタ数が 11 になることがわかったため, 閾値 6.5 で再度階層型クラスタリングを実行し, 得られたクラスタリング結果の各クラスタからサンプリングした代表コードの解き方のラベルを表 8 に示す. 1 のクラスタに含まれる 4 個のソースコードの内, 3 個が garner, 2, 3, 4 のクラスタに属する代表コードのすべてが桁ごとと, 特定の解き方がクラスタに分類されている. 5, 6, 7, 8, 9 のクラスタは大部分の代表コードが Biginteger であるが, 5, 7, 8 には BigDecimal も含まれている. 10, 11 のクラスタには BigInteger, BigDecimal, 桁ごとの 3 種類の異なる解き方のソースコードが含まれており, BigInteger の比率が高い.

図 23 に階層型クラスタリングで得られたクラスタ距離 6.5 以上のデンドログラ

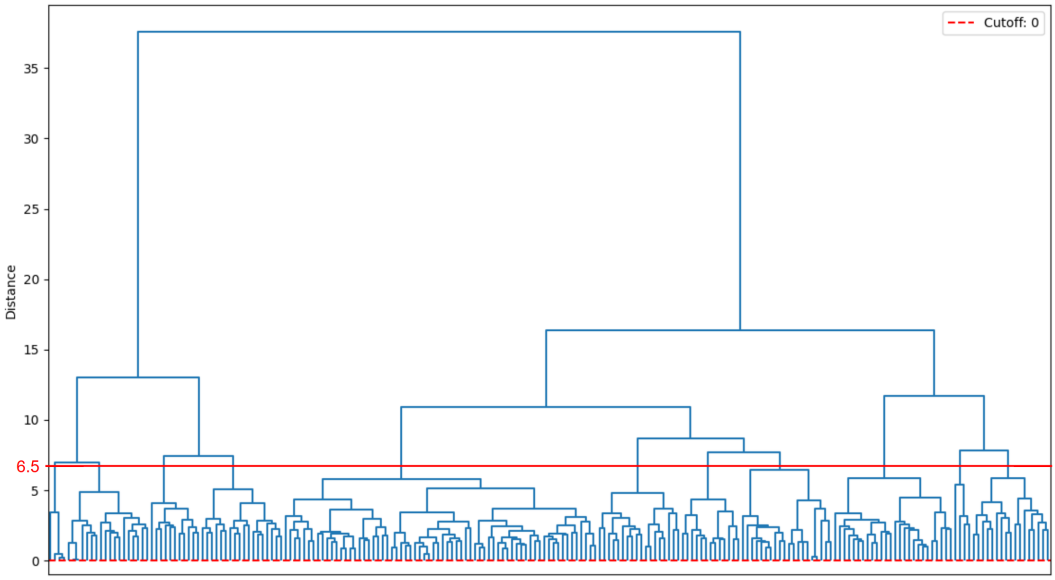

図 22 プロセス全体を表すデンドログラム

ムを示す. 縦軸はクラスタ距離, 横軸はクラスタ番号である. また, クラスタが結合する順番を図内の①から⑩の番号で示している.

まず, 全体像に着目すると, クラスタ 1〜4 とクラスタ 5〜11 は⑩まで結合が起きておらず, もっとも距離があるクラスタであった. クラスタ 1〜4 は garner と桁ごとからなり, クラスタ 5〜11 は主に BigInteger と BigDecimal を多く含むため, garner と桁ごと, BigDecimal と BigInteger はそれぞれの分散表現が近い一方で, 各組み合わせ間の分散表現が離れているといえる. BigDecimal と BigInteger はデータの型以外の処理が一致している点で解き方に類似性があるため, 適切な分類といえる. 一方で, garner と桁ごとは処理として一致していると言える箇所は少ないため, 解き方に類似性があるとは言えず, 今後の改善が必要な点と言える.

次に, クラスタ 1〜4 に着目すると, 桁ごとのみから構成されるクラスタ 2,3,4 よりも先にクラスタ 1 と 2 が先に結合されている. クラスタ 2 が garner から構成されるクラスタ 1 に距離が近くなった原因として, 5.1.4 節でも述べた, "["と"]"の出現数が影響している可能性がある. ソースコード中に配列が現れる場合, トークンとして"["と"]"の出現数が増加し, ソースコード全体の分散表現に影響する. 表 9 にクラスタ 1〜4 の代表コードの全トークン数に対するトークン"["の割合を示す. 但し, 代表コードのトークン"["と"]"の数が一致していたため, "["についての情報で分析を行う. クラスタ 1, 2 の代表コードは総じて全トークンに対する"["の割合が他と比べて高い一方で, クラスタ 3,4 の代表コードはほとんどの場合で割合が低く, この特徴の差が距離に反映された可能性がある. 一方で, クラスタ 4 の代表コード 1 は"["の割合が高く, クラスタ 1,2 の代表コードの割合に近い. ソースコード間の距離は"["の割合以外の様々な要因から決定されており, クラスタ 4 の代表

表8 各クラスタに属する代表コードの解き方

クラスタ	データ数	代表1	代表2	代表3	代表4	代表5
1	4	garner	garner	garner	桁ごと[a]	
2	18	桁ごと	桁ごと	桁ごと	桁ごと	桁ごと
3	14	桁ごと	桁ごと	桁ごと	桁ごと	桁ごと
4	18	桁ごと[b]	桁ごと[c]	桁ごと	桁ごと	桁ごと
5	70	BigInteger	BigInteger	BigInteger	BigDecimal	BigInteger
6	20	BigInteger	BigInteger	BigInteger	BigInteger	BigInteger
7	14	BigDecimal	BigInteger	BigInteger	BigInteger	BigInteger
8	21	BigInteger	BigDecimal	BigInteger	BigInteger	BigInteger
9	28	BigInteger	BigInteger	BigInteger	BigInteger	BigInteger
10	4	BigDecimal	BigInteger	BigInteger	桁ごと[d]	
11	17	BigInteger	桁ごと[e]	BigInteger	BigDecimal	BigInteger

[a]:課題に無関係の関数も記述
[b]:10桁ごとに分割後，long型で加算し桁上げ処理．
[c]:8桁ごとに分割後，int型で加算し桁上げ処理．
[d]:9桁ごとに分割後，int型で加算し桁上げ処理．
[b]:17桁ごとに分割後，long型で加算し桁上げ処理．

表9 全トークンに対する"["の割合

クラスタ	代表1	代表2	代表3	代表4	代表5
1	4.73%	4.74%	4.61%	6.05%	
2	5.35%	4.57%	5.37%	5.05%	5.73%
3	0.26%	0.27%	0.92%	0.42%	0.32%
4	6.00%	3.38%	3.46%	3.38%	0.38%

コード1は"["の割合以外の要因からクラスタ4に分類されていると考えられる．

次に，クラスタ5〜11に着目する．これらのクラスタの中で代表コードにBigDecimalのソースコードを含むのは5, 7, 8, 10, 11のクラスタである．しかし，7と8の結合クラスタ(③で結合)はクラスタ5, 10, 11よりも代表コードにBigDecimalを含まないクラスタ6と距離が近い．このことから，BigDecimalとBigIntegerの解き方の差が分散表現の距離にあまり反映されていないことが考えられる．2つの解き方BigDecimalとBigIntegerは，主に使用するデータの型のみが異なり，他の処理方法などは同一であっても実装が可能である．そのため，2つの解き方を区別するためには，データの型名に対する重み付けを他の要素と変えるなど，分散表現に基づいた距離の算出方法を検討する必要がある．また，前述したようにBigDecimalとBigIntegerはデータの型以外は類似している要素が多い．学習者の解き方を反映した完成例を提示するという本研究の最終的な目標を達成する為には，BigIntegerで実装を進めている学習者にBigDecimalを用いた実装例を提示しても十分機能す

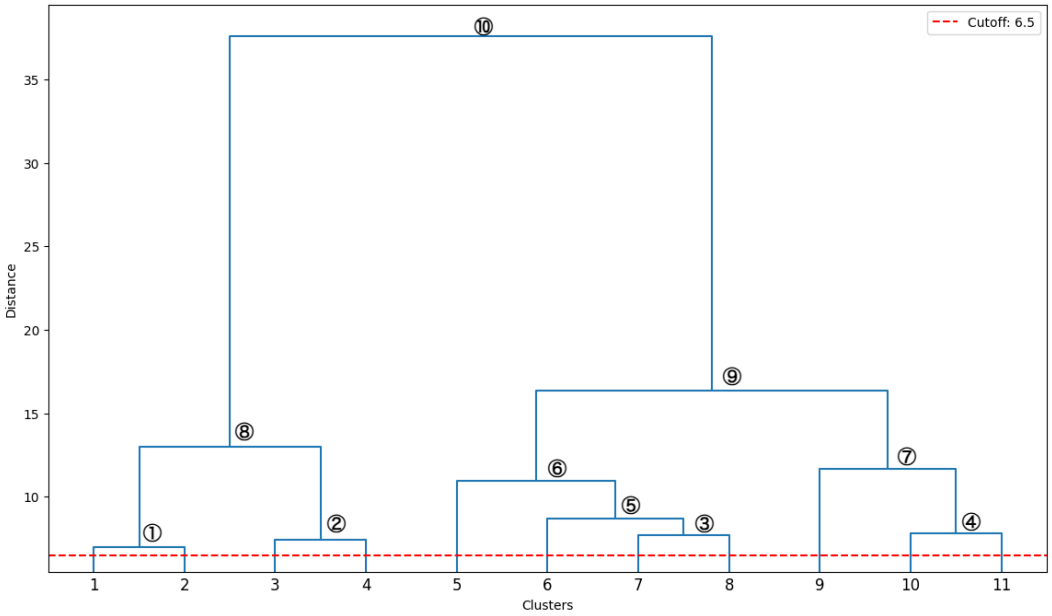

図 23 クラスタ距離 6.5 以上に関するデンドログラム

る可能性もあり, 今後の検討が必要である.

6 おわりに

本研究は正答を作成できなかった学習者を支援するために, 学習者が採用した解き方と類似した解き方を持つ正答例を自動的に検索, 提示するシステムの開発を長期的な目的とした. その第一段階としてソースコードの分散表現のcos類似度と階層型クラスタリングによってソースコード間の解き方の類似性を評価できるかword2vecを用いた分散表現の取得によって実験を行った.

5.1節の結果について, ラベル「BigInteger」,「桁ごと」の対象ソースコードに対しては同じラベルのソースコードを分散表現のcos類似度によって他のソースコードと区別することができた. また, ラベル「BigDecimal」の対象ソースコードと同じラベルのソースコードを分散表現のcos類似度によって他のソースコードと完全に区別することができなかったが, 類似した解き方を含めて区別することができ, 分散表現のcos類似度によってソースコード間の解き方の類似性を一定以上評価できた. しかし, ラベル「garner」の対象ソースコードに対しては, コードクローン以外のラベル「garner」のソースコードが上位5つで確認できなかった. そのため, ラベル「garner」の対象ソースコードに対しては分散表現のcos類似度によって他のソースコードと区別可能であるかが確認できず, 更なる検証が必要である可能性が示唆された. 加えてラベル「garner」の対象ソースコードに対する上位のソースコードではすべてデータ構造として配列を使用していたことから, 参照のたびに特徴的なトークンが出現するようなデータ構造は今回のような方法で取得した分散表現のに反映されやすい可能性があるという新たな知見が得られた. 5.2節の結果として, 分散表現による階層型クラスタリングでは, ラベル「BigInteger」と「BigDecimal」,「garner」と「桁ごと」のそれぞれの組の中で分散表現が近く, 各組の間で分散表現が離れていた. ラベル「BigInteger」と「BigDecimal」の組についてはデータの型以外の処理の一致を反映した適切な分類が行えたが,「garner」と「桁ごと」の組については解き方に類似性があるとは言えず, 適切な分類が行えなかった.「garner」と「桁ごと」の組について, 特に「garner」に分散表現が近かった「桁ごと」のクラスタと「garner」のクラスタは, 他の「桁ごと」のクラスタよりもトークン"["の割合が高く, この特徴の差が「garner」と「桁ごと」の分散表現の距離を近づけた可能性が示唆された. また,「BigDecimal」と「BigInteger」で明確な分散表現の距離が確認できず,「BigDecimal」と「BigInteger」の解き方の差が分散表現の距離にあまり反映されていないことが考えられた. しかし, 研究の最終目標を叶えるために「BigInteger」で実装している学習者に「BigDecimal」のソースコードを提示しても, 他の処理の一致性から十分に機能する可能性についても検討が必要である.

今後の展望として, 今回のword2vecを用いた分散表現の取得方法では, トークンの順序関係が反映されないという問題があるため, トークンの順序関係を分

散表現に反映することができるdoc2vec[9]を用いることにより解決できると考える．また，今回は最終的なシステムが対象とするソースコードをコンパイル不可能なソースコードにまで広げるためにcode2vecを採用しなかったが，code2vecにはソースコードの構造を捉えたうえで学習が行えるという強みがある．今回のようにword2vecを用いた分散表現の取得ではソースコードの構造情報を完全に反映することは難しいが，Javaのような言語では";"や"{"，"}"がトークンをして残っているため，doc2vecなどで順序関係を分散表現に反映させることができれば，ソースコードの構造情報も反映させることができると考えられる．pythonのような言語でも．改行文字やインデントの情報が存在するため，適切な処理をすればJavaのような言語と同様に構造情報を扱えると考えられる．加えて，word2vecを用いた今回の手法では単語ごとに分散表現を出力した後でソースコードの分散表現を算出するため，同じ単語に対する分散表現はどのソースコードにおいても同じである．よってモデルは単語の普遍的な意味を学習する必要があり，ソースコードごとに意味が異なるリテラル(定数)のようなトークンを前処理で削除した．しかし，doc2vecやcode2vecではソースコードごとに分散表現を出力することができるため，ソースコードごとに意味が異なるリテラル(定数)のようなトークンが持つ解き方に関する情報も反映することができると考えられる．次に，ラベル「BigDecimal」の対象ソースコードと同じラベルのソースコードを分散表現のcos類似度によって他のソースコードと完全に区別することができなかったことについて，ラベルごとのソースコード数がわかり，ラベルごとの比率を調整できるほど大規模な検証データを用意することができれば，分散表現の類似性と解き方の類似性に対するさらに厳密な分析が行えると考えられる．また，モデル学習の際のパラメータやソースコードの分散表現を得る方法などを変更し，頻出しないトークンや指定のトークンに重みを付けることでラベル「BigDecimal」を「BigInteger」と区別できなかったような問題の解決に繋がると考えられる．このような手法はTF-IDF[10](ある単語が特定の文書に特徴的かつ重要かを評価する方法)として多くの分野で利用されており，本研究の長期的な目標を叶えるために重要な観点であると思われる．さらに今回，データに対する解き方のラベル付け方法が確立されておらず目視によるラベル付けを行ったため，全データに対するラベル付けを行わず，ラベルの粒度も細かい解き方の違いを無視した荒い4つのラベルしか定義しなかった．しかし，実際にはソースコードの分散表現には細かい解き方の違いも反映されており，どんな解き方の違いが分散表現に反映されるのか，されないのかを明らかにするためには，細かい解き方の違いを示すラベルに基づく分析も必要であると考えられる．

謝辞

本研究を進めるにあたり, 多くの方々のご助力をいただきました. この場を借りてお礼を申し上げます. 指導教員である上野秀剛准教授には, 常日頃から研究の方向性や実験設定の妥当性等の相談に加えて, 論文執筆に関してもご指導いただきました. 心から感謝申し上げます. 査読対応をしていただいた岩田大志准教授には, 本稿の記述の論理性や正確性を上げる指摘などの多くの貴重なご意見をいただきました. 誠にありがとうございました.

参考文献

[1] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, "Efficient Estimation of Word Representations in Vector Space", International Conference on Learning Representations (2013)

[2] 北庄司亮, 松下誠, 肥後芳樹, "機械学習を用いて模範コードを提示する初学者向けプログラミング学習システムの構築", 研究報告ソフトウェア工学(SE), vol.213, No.7, pp.1-8 (2023)

[3] URI ALON, MEITAL ZILBERSTEIN, OMER LEVY, ERAN YAHAV, "code2vec: Learning Distributed Representations of Code", Proc. ACM Program. Lang., Vol.3, Issue POPL, pp.1–29 (2019)

[4] Anupriya Prasad, "Code Clone Detection using Code2Vec", UC Irvine Electronic Theses and Dissertations, University of California, Irvine, pp.1-47 (2020).

[5] T. Mikolov, Q. V. Le, I. Sutskever, "Exploiting similarities among languages for machine translation", ArXiv abs/1309.4168 (2013).

[6] Sivajeet Chand, Sushant Kumar Pandey, Jennifer Horkoff, Miroslaw Staron, Miroslaw Ochodek, Darko Durisic, "Comparing Word-Based and AST-Based Models for Design Pattern Recognition", ACM International Conference Proceeding Series, pp.44-48 (2023).

[7] Ward, J. H., "Hierarchical grouping to optimize an objective function", Journal of the American Statistical Association, Vol.58, No.301, pp.236-244 (1963).

[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, Frederick Reiss, "CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks", NeurIPS Datasets and Benchmarks (2021)

[9] Le, Q., Mikolov, T., "Distributed Representations of Sentences and Documents", Proceedings of the 31st International Conference on Machine Learning, Vol.32, No.2, pp.1188-1196 (2014).

[10] Spärck Jones, K., "A Statistical Interpretation of Term Specificity and Its Application in Retrieval", Information Storage and Retrieval, Vol.17, No.5, pp.391-402 (1972).

付録

5.1.4 節でコードクローンであることを確認したラベル「garner」の 3 つのソースコード (s964736393.java, s728758016, s678994851) を図 24, 図 25, 図 26, 図 27, 図 28, 図 29 に示す.

```java
import java.io.*;
import java.util.*;
import java.math.BigInteger;

class Lib{
    /* Verify http://codeforces.com/contest/903/problem/D */
    /* ?????? http://math314.hateblo.jp/entry/2015/05/07/014908 */
    static long extgcd(long a, long b, long[]x) {
        for (long u = x[1] = 1,  v = x[0] = 0; a!=0;){
            long q = b / a;
            x[0] -= q * u;
            long t=x[0];
            x[0]=u;
            u=t;
            x[1] -= q * v;
            t=x[1];
            x[1]=v;
            v=t;
            b -= q * a;
            t=b;
            b=a;
            a=t;
        }
        return b;
    }

    static long mod_inv(long a, long m) {
        long[]x=new long[2];
        extgcd(a,  m,  x);
        return (m + x[0] % m) % m;
    }
    static BigInteger garner(long[]a, long[]m){
        int ms=a.length;
        long[]coffs=new long[ms], constants=new long[ms];
        long[]digs=new long[ms];
        Arrays.fill(coffs, 1);
        for(int i=0;i<ms;++i){
            long v = (a[i] - constants[i]) * mod_inv(coffs[i],  m[i]) % m[i];
            if(v<0)v+=m[i];
            digs[i]=v;
            for (int j = i + 1; j < ms; j++) {
                constants[j] += coffs[j] * v;
                constants[j] %= m[j];
                coffs[j] *= m[i];
                coffs[j] %= m[j];
            }
        }
        BigInteger ans=BigInteger.valueOf(0), c=BigInteger.valueOf(1);
        for(int i=ms-1;i>=0;--i){
            c=c.multiply(BigInteger.valueOf(m[i]));
            ans=ans.multiply(BigInteger.valueOf(m[i]));
            ans=ans.add(BigInteger.valueOf(digs[i]));
        }
        if(ans.compareTo(c.divide(BigInteger.valueOf(2)))>0)
            ans=ans.subtract(c);
        return ans;
    }
}
class Main {
    public static void main(String[] args) {
        MyScanner sc = new MyScanner();
        out = new PrintWriter(new BufferedOutputStream(System.out));
        Random rnd=new Random(0x314159265L);
        int t=sc.nextInt();
        BigInteger lim=BigInteger.TEN.pow(80);
        int ms=12;//(10^9)^12=10^108
        long[]m=new long[ms];
```

図 24 ラベル「garner」のソースコード (s964736393.java)

```
68          for(int i=0;i<ms;++i){
69              boolean ok;
70              do{
71                  long p=BigInteger.probablePrime(31, rnd).longValue();
72                  ok=true;
73                  for(int j=0;j<i;++j)
74                      if(m[j]==p){
75                          ok=false;
76                          break;
77                      }
78                  if(ok)m[i]=p;
79              }while(!ok);
80          }
81          while(t-->0){
82              String a=sc.next();
83              String b=sc.next();
84              BigInteger ba=new BigInteger(a);
85              BigInteger bb=new BigInteger(b);
86              long[]ans=new long[ms];
87              for(int i=0;i<ms;++i){
88                  long ss=ba.mod(BigInteger.valueOf(m[i])).longValue();
89                  long tt=bb.mod(BigInteger.valueOf(m[i])).longValue();
90                  ans[i]=(ss+tt)%m[i];
91              }
92              BigInteger r=Lib.garner(ans, m);
93              if(r.compareTo(lim)>=0)
94                  out.println("overflow");
95              else
96                  out.println(r);
97          }
98          out.close();
99      }
100     // http://codeforces.com/blog/entry/7018
101     //-----------PrintWriter for faster output---------------------------------
102     public static PrintWriter out;
103     //-----------MyScanner class for faster input----------
104     public static class MyScanner {
105         BufferedReader br;
106         StringTokenizer st;
107         public MyScanner() {
108             br = new BufferedReader(new InputStreamReader(System.in));
109         }
110         String next() {
111             while (st == null || !st.hasMoreElements()) {
112                 try {
113                     st = new StringTokenizer(br.readLine());
114                 } catch (IOException e) {
115                     e.printStackTrace();
116                 }
117             }
118             return st.nextToken();
119         }
120         int nextInt() {
121             return Integer.parseInt(next());
122         }
123         long nextLong() {
124             return Long.parseLong(next());
125         }
126         double nextDouble() {
127             return Double.parseDouble(next());
128         }
129         String nextLine(){
130             String str = "";
131             try {
132                 str = br.readLine();
133             } catch (IOException e) {
134                 e.printStackTrace();
135             }
136             return str;
137         }
138     }
139 }
```

図 25 ラベル「garner」のソースコード (s964736393.java)(続き)

```java
import java.io.*;
import java.util.*;
import java.math.BigInteger;

class Lib{
    /* Verify http://codeforces.com/contest/903/problem/D */
    /* ?????? http://math314.hateblo.jp/entry/2015/05/07/014908 */
    static long extgcd(long a,long b,long[]x) {
        for (long u = x[1] = 1, v = x[0] = 0; a!=0;){
            long q = b / a;
            x[0] -= q * u;
            long t=x[0];
            x[0]=u;
            u=t;
            x[1] -= q * v;
            t=x[1];
            x[1]=v;
            v=t;
            b -= q * a;
            t=b;
            b=a;
            a=t;
        }
        return b;
    }

    static long mod_inv(long a,long m) {
        long[]x=new long[2];
        extgcd(a, m, x);
        return (m + x[0] % m) % m;
    }
    /*
      x==a[i] (mod
          m[i])???????????x????????????? ¶ ?????????? ° ??????????????????????
      ??????????? ¶  a.length==m.length
      ? ¨ ??????? O(a.length^2)
    */
    static BigInteger garner(long[]a,long[]m){
        int ms=a.length;
        long[]coffs=new long[ms],constants=new long[ms];
        long[]digs=new long[ms];
        Arrays.fill(coffs,1);
        for(int i=0;i<ms;++i){
            long v = (a[i] - constants[i]) * mod_inv(coffs[i], m[i]) % m[i];
            if(v<0)v+=m[i];
            digs[i]=v;
            for (int j = i + 1; j < ms; j++) {
                constants[j] += coffs[j] * v;
                constants[j] %= m[j];
                coffs[j] *= m[i];
                coffs[j] %= m[j];
            }
        }
        BigInteger ans=BigInteger.valueOf(0),c=BigInteger.valueOf(1);
        for(int i=ms-1;i>=0;--i){
            c=c.multiply(BigInteger.valueOf(m[i]));
            ans=ans.multiply(BigInteger.valueOf(m[i]));
            ans=ans.add(BigInteger.valueOf(digs[i]));
        }
        if(ans.compareTo(c.divide(BigInteger.valueOf(2)))>0)
            ans=ans.subtract(c);
        return ans;
    }
}

class Main {
    public static void main(String[] args) {
        MyScanner sc = new MyScanner();
        out = new PrintWriter(new BufferedOutputStream(System.out));
        Random rnd=new Random(0x314159265L);
        int t=sc.nextInt();
        BigInteger lim=BigInteger.TEN.pow(80);
        int ms=12;//(10^9)^12=10^108
        long[]m=new long[ms];
```

図 26 ラベル「garner」のソースコード (s728758016.java)

```
74          for(int i=0;i<ms;++i){
75              boolean ok;
76              do{
77                  long p=BigInteger.probablePrime(31,rnd).longValue();
78                  ok=true;
79                  for(int j=0;j<i;++j)
80                      if(m[j]==p){
81                          ok=false;
82                          break;
83                      }
84                  if(ok)m[i]=p;
85              }while(!ok);
86          }
87          while(t-->0){
88              String a=sc.next();
89              String b=sc.next();
90              long[]ans=new long[ms];
91              for(int i=0;i<ms;++i){
92                  long ss=0,tt=0;
93                  for(char c:a.toCharArray())
94                      ss=(10*ss+c-'0')%m[i];
95                  for(char c:b.toCharArray())
96                      tt=(10*tt+c-'0')%m[i];
97                  ans[i]=(ss+tt)%m[i];
98              }
99              BigInteger r=Lib.garner(ans,m);
100             if(r.compareTo(lim)>=0)
101                 out.println("overflow");
102             else
103                 out.println(r);
104         }
105         out.close();
106     }
107     // http://codeforces.com/blog/entry/7018
108     //-----------PrintWriter for faster output---------------------------------
109     public static PrintWriter out;
110     //-----------MyScanner class for faster input----------
111     public static class MyScanner {
112         BufferedReader br;
113         StringTokenizer st;
114         public MyScanner() {
115             br = new BufferedReader(new InputStreamReader(System.in));
116         }
117         String next() {
118             while (st == null || !st.hasMoreElements()) {
119                 try {
120                     st = new StringTokenizer(br.readLine());
121                 } catch (IOException e) {
122                     e.printStackTrace();
123                 }
124             }
125             return st.nextToken();
126         }
127         int nextInt() {
128             return Integer.parseInt(next());
129         }
130         long nextLong() {
131             return Long.parseLong(next());
132         }
133         double nextDouble() {
134             return Double.parseDouble(next());
135         }
136         String nextLine(){
137             String str = "";
138             try {
139                 str = br.readLine();
140             } catch (IOException e) {
141                 e.printStackTrace();
142             }
143             return str;
144         }
145     }
146 }
```

図 27 ラベル「garner」のソースコード (s728758016.java)(続き)

```java
import java.io.*;
import java.util.*;
import java.math.BigInteger;


class Lib{
    /* Verify http://codeforces.com/contest/903/problem/D */
    /* ?????? http://math314.hateblo.jp/entry/2015/05/07/014908 */
    static long extgcd(long a,long b,long[]x) {
        for (long u = x[1] = 1, v = x[0] = 0; a!=0;){
            long q = b / a;
            long r=x[0]-q*u;
            x[0]=u;
            u=r;
            r=x[1]-q*v;
            x[1]=v;
            v=r;
            r=b-q*a;
            b=a;
            a=r;
        }
        return b;
    }


    static long mod_inv(long a,long m) {
        long[]x=new long[2];
        extgcd(a, m, x);
        return (m + x[0] % m) % m;
    }
    /*
      x==a[i] (mod
          m[i])????????????x??????????????¶??????????°?????????????????????
      ??????????¶ a.length==m.length
      ?¨??????? O(a.length^2)
    */
    static BigInteger garner(long[]a,long[]m){
        int ms=a.length;
        long[]coffs=new long[ms],constants=new long[ms];
        long[]digs=new long[ms];
        Arrays.fill(coffs,1);
        for(int i=0;i<ms;++i){
            long v = (a[i] - constants[i]) * mod_inv(coffs[i], m[i]) % m[i];
            if(v<0)v+=m[i];
            digs[i]=v;
            for (int j = i + 1; j < ms; j++) {
                constants[j] += coffs[j] * v;
                constants[j] %= m[j];
                coffs[j] *= m[i];
                coffs[j] %= m[j];
            }
        }
        BigInteger ans=BigInteger.valueOf(0),c=BigInteger.valueOf(1);
        for(int i=ms-1;i>=0;--i){
            c=c.multiply(BigInteger.valueOf(m[i]));
            ans=ans.multiply(BigInteger.valueOf(m[i]));
            ans=ans.add(BigInteger.valueOf(digs[i]));
        }
        if(ans.compareTo(c.divide(BigInteger.valueOf(2)))>0)
            ans=ans.subtract(c);
        return ans;
    }
}


class Main {
    public static void main(String[] args) {
        MyScanner sc = new MyScanner();
        out = new PrintWriter(new BufferedOutputStream(System.out));
        Random rnd=new Random(0x314159265L);
        int t=sc.nextInt();
        BigInteger lim=BigInteger.TEN.pow(80);
        int ms=12;//(10^9)^12=10^108
        long[]m=new long[ms];
```

図 28 ラベル「garner」のソースコード (s678994851.java)

```
74          for(int i=0;i<ms;++i){
75              boolean ok;
76              do{
77                  long p=BigInteger.probablePrime(31,rnd).longValue();
78                  ok=true;
79                  for(int j=0;j<i;++j)
80                      if(m[j]==p){
81                          ok=false;
82                          break;
83                      }
84                  if(ok)m[i]=p;
85              }while(!ok);
86          }
87          while(t-->0){
88              String a=sc.next();
89              String b=sc.next();
90              long[]ans=new long[ms];
91              for(int i=0;i<ms;++i){
92                  long ss=0,tt=0;
93                  for(char c:a.toCharArray())
94                      ss=(10*ss+c-'0')%m[i];
95                  for(char c:b.toCharArray())
96                      tt=(10*tt+c-'0')%m[i];
97                  ans[i]=(ss+tt)%m[i];
98              }
99              BigInteger r=Lib.garner(ans,m);
100             if(r.compareTo(lim)>=0)
101                 out.println("overflow");
102             else
103                 out.println(r);
104         }
105         out.close();
106     }
107     // http://codeforces.com/blog/entry/7018
108     //-----------PrintWriter for faster output---------------------------------
109     public static PrintWriter out;
110     //-----------MyScanner class for faster input----------
111     public static class MyScanner {
112         BufferedReader br;
113         StringTokenizer st;
114         public MyScanner() {
115             br = new BufferedReader(new InputStreamReader(System.in));
116         }
117         String next() {
118             while (st == null || !st.hasMoreElements()) {
119                 try {
120                     st = new StringTokenizer(br.readLine());
121                 } catch (IOException e) {
122                     e.printStackTrace();
123                 }
124             }
125             return st.nextToken();
126         }
127         int nextInt() {
128             return Integer.parseInt(next());
129         }
130         long nextLong() {
131             return Long.parseLong(next());
132         }
133         double nextDouble() {
134             return Double.parseDouble(next());
135         }
136         String nextLine(){
137             String str = "";
138             try {
139                 str = br.readLine();
140             } catch (IOException e) {
141                 e.printStackTrace();
142             }
143             return str;
144         }
145     }
146 }
```

図 29 ラベル「garner」のソースコード (s678994851.java)(続き)