修 士 論 文

題　目　　Cプログラム学習を支援するための
穴あきヒントの自動生成

主任指導教員　　水野　修　教授

指導教員　　崔　恩瀞　助教

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　20622003

氏　　名　　上北　裕也

令和4年2月7日提出

様式 2 号

学位論文内容の要旨（和文）

令和 4 年 2 月 7 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
令和 2 年入学
学生番号　　20622003
氏　　名　　上北　裕也　㊞

（主任指導教員　　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

C プログラム学習を支援するための穴あきヒントの自動生成

2.　論文内容の要旨（400 字程度）

プログラミングの学習においては，近年 iSnap などに代表される自動フィードバック機能，すなわち学習者の回答に応じて自動生成されたヒントを提供する機能を持つシステムの研究が進められている．適切に用いられるフィードバックは学習者の学習効果，モチベーションを向上させることや教師の負担を減らすことが期待される．

このようなフィードバックシステムにおいては，ヒントコードを生成するためにマルコフ決定過程を用いた経路探索などのアルゴリズムを用いるものもあるが，これらを用いるとシステムの規模はどうしても大きくなる．簡易なアルゴリズムを用いたシンプルな実装でも十分なヒントの提供が行えるならば，フィードバックシステムをよりスリムに構築することができ，開発にかかるリソースを減らせると考える．

本論文では複雑なアルゴリズムを用いずに，C 言語のみを対象とした簡易的な自動フィードバックのツールを提案する．このツールの採用により，どの程度正確なフィードバックがなされたかを確認した．アルゴリズムのアイディアとしては学習者のソースコードを抽象構文木に変換，過去の学生の回答データから学習者の編集部分にマッチする部分木を検索，一番望ましいと思われる部分のコードを表示する．このツールを用いて幾らかのケーススタディを行い，出力されたヒントの妥当性を検証した．

Automatic generation of perforated hints to support C program learning

2022 20622003 *UEKITA Hiroya*

Abstract

In programing language learning, there has been much research on automatic feedback systems, such as iSnap, that provide hints to learners in response to their answers. Appropriate feedback is expected to improve learners' learning effectiveness and motivation, and to reduce the burden on the teacher.

In such feedback systems, some algorithms such as path finding using Markov decision processes are used to generate hint codes, but these algorithms inevitably increase the scale of the system. If a simple implementation using a simple algorithm can provide enough hints, the feedback system can be built more slenderly and the resources required for development can be reduced. In this paper, we present a simple implementation of a feedback system using a simple algorithm.

We propose a simple automatic feedback tool for the C language only, without using any complicated algorithm, and check how accurate the feedback is. The idea of the algorithm is to convert the learner's source code into an abstract syntax tree, search for a subtree that matches the learner's edited part from the past student's response data, and display the code of the part that seems to be the most desirable. We conducted several case studies using this tool, and verified the validity of the output hints.

目　次

1. 緒言

近年，プログラミング技術は重要なものとなっており，プログラミング学習の分野においては，学習者の回答に応じて自動生成されたヒントを提供する自動フィードバック機能を持つITS（インテリジェントチュータリングシステム）の研究が進められている．例としてiSnap [1] が挙げられる．これはブロックベースのプログラミング環境Snap [2] に，詳細なログ取得やヒントの自動生成などの機能を追加したシステムである．こういったプログラミング環境とそれに関連したカリキュラムは，従来の指導に比べ，テストの得点向上 [3] ，より効果的なプログラミング編集 [4] などの効果があることが過去の実証研究によって示されている．

このようなフィードバックシステムにおいては，ヒントコードを自動生成するためにデータ駆動型のアプローチをとり，マルコフ決定過程を用いた経路探索などのアルゴリズムを用いるものもある [5]．今までに様々なアルゴリズムが研究されているが，こういったアルゴリズムは複雑なものが多くシステム構築も難しくなることが想定される．一方簡易なアルゴリズムを用いたシンプルな実装でも十分なヒントの提供が行えるならば，フィードバックシステムをよりスリムに構築することができ，システムの開発にかかるリソースを減らせると考える．よって本研究では，簡易的な自動フィードバックのツールを制作し，どの程度正確なフィードバックがなされたかについて分析を行う．

本ツールの対象言語は，C言語とする．アルゴリズムの概要としては他のデータ駆動型ヒント生成のアルゴリズムにならい，まずは学習者のソースコードを抽象構文木に変換する．抽象構文木とはソースコードの意味に関係ない情報を取り除き、関係ある情報のみを取り出した木構造である．その後過去の学生の回答データから学習者の編集部分にマッチする部分木を検索し，最も望ましいと思われる部分木をコードに再変換して表示する．このツールを用いていくつかのケーススタディを検証し，ツールが適切なヒントを提供できるかを確認した．

本論文の構成を紹介する．まず，第2章では必要となる背景知識について触れる．第3章では，本研究にあたって参考とした研究を紹介する．第4章では，本研究において開発した簡易的なフィードバックツールについて，言語等の環境やアルゴリズムについて詳細に説明する．第5章ではに，上記フィードバックツールのケーススス

タディについて記述し，そこから得られた考察を述べる．最後に第 6 章では，本研究のまとめと今後の課題を述べる．

2. 研究概要

2.1 背景

インテリジェントチュータリングシステム（ITS）は，コンピュータプログラミングの分野で有望視されている [5] システムでその大きな特徴は学習者の回答に応じて自動生成されたヒントを提供する，自動フィードバック機能を持つことである．ITS やフィードバックの利用は，学生のパフォーマンスを向上させることが示されており [6]，ITS を利用した学生と従来の指導を受けた学生とでは，実施したテストの標準偏差において前者が 2 高くなったとの研究結果もある [7].

このような ITS においては，自由形式のプログラミング課題への対応などを目的として複雑なアルゴリズムを用いたフィードバック生成システムの研究がなされている．しかし，複雑なアルゴリズムを用いればシステムも複雑になり，構築も難しくなることが想定される．簡易なアルゴリズムを用いたシンプルな実装でも十分なヒントの提供が行えるならば，フィードバックシステムをよりスリムに構築することができ，システムの開発にかかるリソースを減らせると考える．

2.2 目的

本研究の目的は，C 言語を対象とした簡易な自動フィードバックシステムを作成し，それを用いて適切なヒントが提供されることを確認することである．また，本システムのヒント提供においてヒント提供にかかる時間がどう増えるのか，また提供されるヒントの正確さが変わるのかを調べ適切に提供が可能となるパラメータ調整を見つけ出すことも目的とする．

2.3 提案するツールの概要

提案ツールは Visual Studio Code（以下 VSCode）[8] の拡張機能として実装する．提案するツールの対象言語は，C 言語を対象とする．これは C 言語が広く使用されており，多くのカリキュラムで始めに学習される言語であることが理由である．

本ツールは学習者のソースファイル，及びヒントを要求する行の位置を入力にと

り，ヒントとしてコード数行を出力するシステムである．具体的な処理としては他のデータ駆動型ヒント生成のアルゴリズムにならい，まずは学習者のソースコードを抽象構文木に変換する．その後，過去の学生の回答データから学習者の編集部分にマッチする部分木を検索，一番望ましいと思われる部分木をコードに再変換して表示する．

3. 関連研究

本章では，本研究に関連する既存研究を紹介する．

3.1 Hint Factory

Hint Factory[9] は強化学習に用いられる手法であるマルコフ決定過程（MDP）を応用し，過去の学生データから文脈に応じたヒントを自動生成するヒント生成手法である．多くのフィードバック生成研究において先行研究として参照されている研究であり，本研究においても過去の学生データからヒントを生成する，ソースコードを抽象構文木化して扱うなど多くの部分を参考にした．

3.2 CTD algorithm

Contextual Tree Decomposition (CTD) algorithm[5] は上記 Hint Factory のアルゴリズムを参考に，オープンエンドなプログラミング問題に対応できるように拡張したデータ駆動型アルゴリズムである．過去の回答コードではなく，生徒の編集を記録してネットワークを構築する手法をとっており，Hint Factory の課題であった状態空間の削減にも成功している．本研究においてはコード比較時，生徒のコードの全体で比較するのではなく，注目部分の部分木に注目して比較することで精度を上げることを参考にした．

4. 提案ツール

4.1 言語環境及び言語

2.3 節で述べたように，本研究は Visual Studio Code（以下 VSCode）の拡張機能として実装する．VSCode の拡張機能は TypeScript を用いて実装する．環境として JavaScript 実行環境の Node.js，及び JavaScript の API である VS Code API を用いる．これらの要素技術の詳細を以下に示す.

1.Visual Studio Code

Visual Studio Code[8] は Microsoft が開発しているソースコードエディタであり，デバッグ，埋め込み Git コントロールと GitHub，シンタックスハイライト，コード補完などのサポートが含まれている．またカスタマイズ性が高く，テーマやキーボードショートカット，環境設定を変更できたり，機能を追加する拡張機能をインストールすることができる．本研究に用いたツールは本エディタの拡張機能として開発した．また，VSCode の拡張機能開発において呼び出すことができる API 群が VS Code API (注 1)として提供されている．本ツールの開発ではエディタ側の情報の取得や画面への出力など全般にわたってこの API を利用している.

2.Node.js

Node.js [10] は非同期型，イベント駆動の JavaScript 環境であり，ネットワークアプリケーションの構築に適している．本ツール制作においては VSCode 拡張機能の実装に用いた.

3.TreeModel

本プログラムでは TreeModel (注 2)という JavaScript ライブラリを用いている．このライブラリは木構造データを扱うためのライブラリであり，主に木構造データをノードの検索やフィルタなどの操作を行うために用いた.

(注 1)：https://code.visualstudio.com/api/references/vscode-api
(注 2)：http://jnuno.com/tree-model-js/

4.2 Clang

今回コードの抽象構文木化にはC言語のコンパイラであるClang[11]の機能を用いた．Clangにはコンパイルするソースファイルの抽象構文木を出力する機能があり，さらに結果をJSON形式で出力することもできる．これを用い，コマンド呼び出し時に保存した学習者のソースファイルをClangを用いて抽象構文木化しこのJSONファイルを読み込んでプログラム上で扱う．また，比較に用いる過去の回答コードもClangを用いて抽象構文木化しておく．こちらは今回手動で行った．

4.3 抽象構文木

抽象構文木とはソースコードの意味に関係ない情報を取り除き、関係ある情報のみを取り出した木構造であり，そのままでは本来比較することが難しいソースコードを比較することが容易になる．構文木の節点をノードと呼称する．

本論文では，あるノードの親となるノードを親ノード，子に当たるノードを子ノード，親が同じノードを兄弟ノードと呼称する．また，あるノード及びその兄弟ノードを元コードでの行番号が若い順に並べた時，そのノードの前後に位置するノードを前ノード，後ノードと呼称する．

4.4 ツール動作

本研究ではコードのパターンマッチにおいて「注目箇所及び前後部分の兄弟ノードが一致する部分木を検索する」ことによって類似コードを見つけ出すという手法をとる. そしてパターンがマッチした場合そのマッチの具合に応じて点数をつけ，点数が一番高いものをヒントコードとして推薦する. 大まかな動作を図4.1に示す.

本拡張機能の動作として主に3つの段階に分けて考えることができる．現在のソースコードを抽象構文木化し，JSONファイルに変換して内容を取り込む入力部分，抽象構文木化したソースコード群を比較評価し，現在のカーソル部分と類似した部分木を見つけ出す比較部分，見つけた部分木から対応するヒントコードを表示する出力部分である.

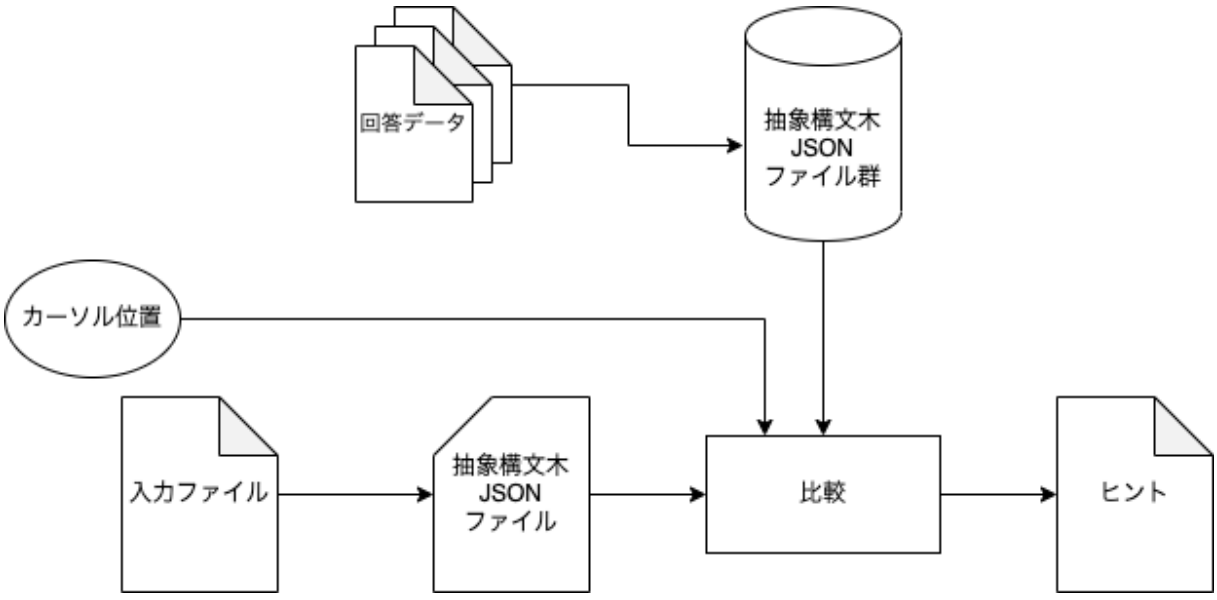

図 4.1　提案ツールの動作

4.4.1　入力部分

入力として学習者の現在編集しているコードファイル（以下入力ファイルと称す），及び入力ファイル上でのカーソル現在位置，すなわちコマンド呼び出し時にカーソルのあった行番号を取得する．入力ファイルは後述する Clang を呼び出して抽象構文木化し，JSON 形式として出力したものを扱う．

学習者は通常，現在編集しているコードの続きについてヒントを求めていると考えられるのでカーソル部分及びその前部分のコードをプログラム上で注目する．

入力ファイルの例を図 4.2，コマンドの呼び出しを図 4.3 に示す．

4.4.2　比較部分

入力部分の項でも述べたように，まずは抽象構文木化した学習者のコードを調査しコマンド呼び出し時のカーソル部分のノードを過去の学生の回答コード群から検索する．これを検索ノードとする．

Clang を用いて出力された JSON 形式の構文木には，”kind”という要素が存在する．これはノードの種類を表す要素であり，種類ごとに特定の文字列が当てはめられている．これを踏まえて，本提案ツールのパターンマッチ方法（以降検索アプローチと呼称）について述べる．

提案ツールのパターンマッチにおいてはまず，比較する構文木から検索ノードと”kind”が一致するノード群を抽出する．ただし，ファイル中に検索ノードと”kind”が一致するノードが存在しなかった場合，検索ノードの兄弟ノードを順番に検索ノードにして，一致するノードが見つかるまでやり直す．それでも該当するノードがなかった場合，その構文木の比較を終了する．

次にそれらのノードそれぞれにおいて，事前に設定した数だけ前後の兄弟ノードの”kind”を調べ，検索ノードの前後の兄弟ノードと比較する．パターンがマッチした場合そのマッチの具合に応じて点数をつけ，点数が一番高いものを残す．これを全データファイルの構文木に対して行い，最後まで残ったものをヒントコードとして推薦する．

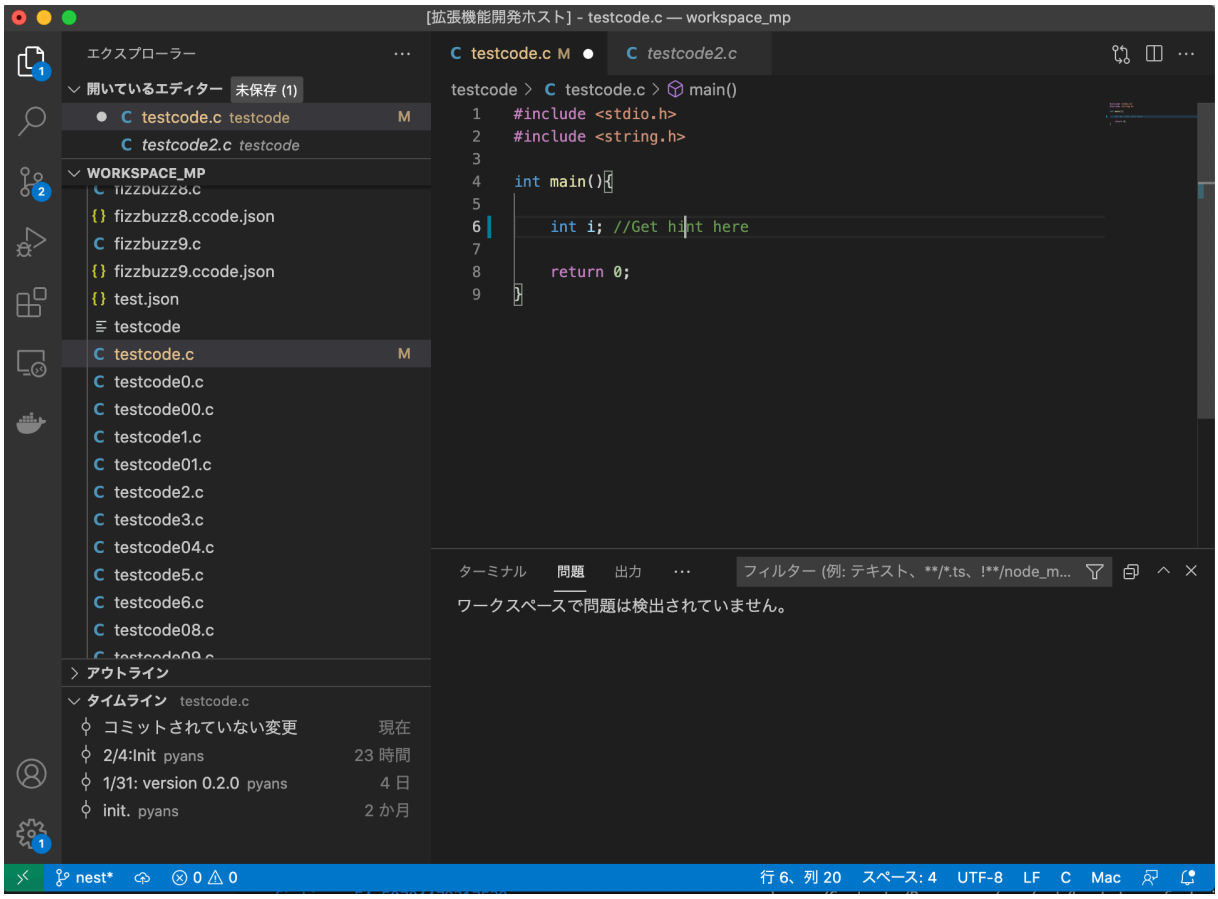
[拡張機能開発ホスト] - testcode.c — workspace_mp
エクスプローラー
開いているエディター 未保存 (1)
testcode.c testcode M
testcode2.c testcode
WORKSPACE_MP
fizzbuzz8.ccode.json
fizzbuzz9.c
fizzbuzz9.ccode.json
test.json
testcode
testcode.c M
testcode0.c
testcode00.c
testcode1.c
testcode01.c
testcode2.c
testcode3.c
testcode04.c
testcode5.c
testcode6.c
testcode08.c
アウトライン
タイムライン testcode.c
コミットされていない変更 現在
2/4:Init pyans 23 時間
1/31: version 0.2.0 pyans 4 日
init. pyans 2 か月
testcode.c M
testcode2.c
testcode > testcode.c > main()
#include <stdio.h>
#include <string.h>
int main(){
int i; //Get hint here
return 0;
}
ターミナル 問題 出力
フィルター (例: テキスト、**/*.ts、!**/node_m...
ワークスペースで問題は検出されていません。
nest*
0 0
行 6、列 20 スペース: 4 UTF-8 LF C Mac

図 4.2　入力ファイル例

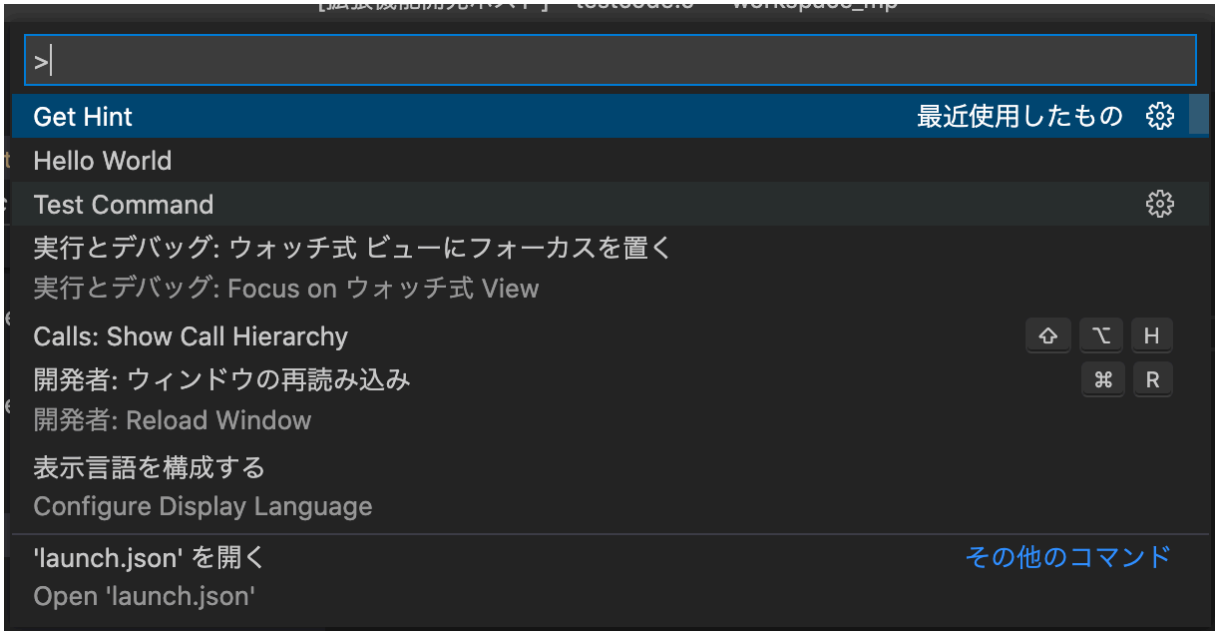

図 4.3　コマンド呼び出し

(1) 比較ノードの採点

比較時にノードに与えられる点数に関しては，ヒューリスティックな式を用いている．検索ノード及びその兄弟ノードと一つマッチするたびに一律で 500 を加算する．ただし検索ノードと”kind”が一致するノードが存在せず，検索ノードが兄弟ノードに変更された場合はその回数を n として，式 4.1 に応じて加算点は減少する．

$$K = (500 * (1 - k/(l + 8))) * s \tag{4.1}$$

$$k = (n + 1)/2 * (-1)^n \tag{4.2}$$

$$s = 0.9 * (n\%2) \tag{4.3}$$

ここで，K は加算点，l は兄弟ノードの内パターン評価に用いる前後ノードの数，n は検索ノードが変更された回数である．ただし，k は小数点以下切り捨て，n は比較する構文木ファイルが変わるたびに 0 に戻る．

(2) 追加の検索アプローチ

ノード検索に関して，本研究では上述の検索アプローチ（以後基本アプローチと呼称）に加えて，これを基にした 3 つの検索アプローチ（以後追加アプローチと呼称）をとった．3 つとも比較ノードの採点までは上述したものと同じであるが，以降追加でパターンマッチを行い，点数を増減させる．

1 つ目は検索ノードの親ノードと，マッチしたノードの親ノードの”kind”が同じ場合，さらに K 点が加算されるものである．2 つ目は検索ノードの子ノードと，マッチしたノードの子ノードを再起的に探索，どちらかのノードが見つからなくなったケースをカウントし，最終的にその合計を点数からマイナスするものである．3 つ目はこれらのアプローチを両方行うものである．

4.4.3 出力部分

マッチしたヒントコード，及びその後ろに続く部分を取り出しコンソールに出力する．ヒントコードは元のソースファイルから対応する部分を抽出し，編集せずそ

のまま出力している．if 文や for 文等明確なブロックに含まれていればその全て，なければヒント部分を先頭に含む 3 行を出力する．また，この時同時に処理開始からヒントの最終行出力までの時間を計測してコンソールに表示する．出力例を図 4.4 に示す.

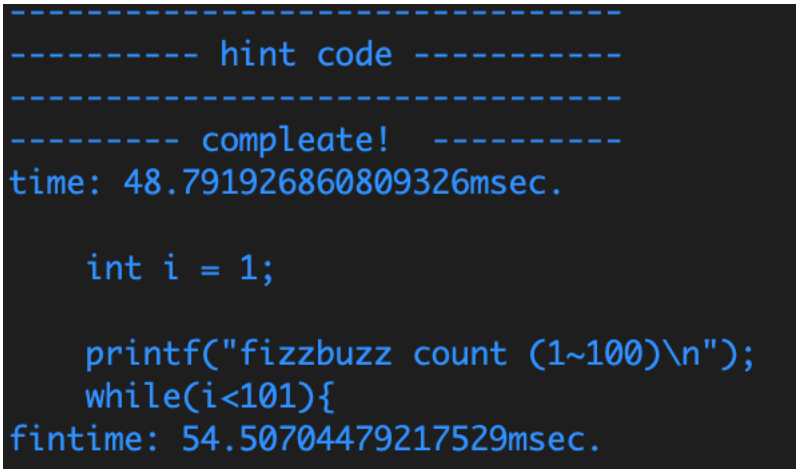
---------- hint code ----------
---------- compleate! ----------
time: 48.791926860809326msec.
int i = 1;
printf("fizzbuzz count (1~100)\n");
while(i<101){
fintime: 54.50704479217529msec.

図 4.4　出力例

5. ケーススタディ

5.1 概要

本提案ツールが具体的にどのように動作するかを示すため，ケーススタディを行った．以降，行ったケーススタディの詳細及びその実行結果を述べる．さらにその結果を受け条件を変化させた別のケーススタディとの比較を行い，それらの考察について述べる．

5.2 適用対象

今回は fizzbuzz プログラムの課題をケーススタディの適用対象として取り扱った．fizzbuzz プログラムとは，1 から 100 までのカウントアップを行うプログラムであり，数字が 3 の倍数の場合は fizz，5 の倍数の場合は buzz，15 の倍数の場合は fizzbuzz を数字の代わりに表示する．図 5.1 にソースコード実装の一例を示す．

比較用ファイル

上記の fizzbuzz プログラム課題に対する生徒の回答を想定したソースファイルを 10 個用意し，これらを抽象構文木化したものを比較用のデータベースとして用いる．内容はインターネット上で公開されているソースコードなどを参考に自作した．

入力ファイル

上述した比較用ソースファイルの特定部分を消去したものを入力ファイルとして同じ数だけ用意する．消去した部分にはコメントを差し込んでおり，この位置にカーソルを合わせてツールを実行する．

用意した入力ファイルを VSCode 上で開き，消去した部分の行にカーソルを合わせて提案ツールを実行する．検索に用いる検索ノードと前後ノードの合計数（以下検索ノード数と称する）は 3，検索アプローチは基本アプローチを用いた．

出力を見て，適切なヒントが得られたかどうかを判断する．ただし，何を持って「適切なヒント」とするかは主観による部分が大きく，定義が難しいため本研究における適切なヒントとは，先述した入力ファイルの作成時に消去した部分が表示され

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int i = 0;

    printf("fizzbuzz count (1~100)\n");
    for(i=1;i<101;i++){
        if(i % 3 == 0 || i % 5 == 0){
            if (i % 3 == 0){
                printf("fizz");
            }
            if (i % 5 == 0){
                printf("buzz");
            }
            printf("\n");
        }else{
            printf("%d\n",i);
        }
    }
    return 0;
}
```

図 5.1　fizzbuzz プログラムの例

ることとする．これら 10 個のソースファイルの出力結果について正答率を調べた。また出力時には同時にヒントの出力にかかった時間も表示させ，こちらも記録して平均値を算出した．

5.3 出力結果

正答例を図 5.2 及び図 5.3，誤答例を図 5.4 及び図 5.5 に示す．また，以下の表 5.1 に全体の実行結果を示す．表中の実行時間は処理開始からヒントの最終行出力までのミリ秒単位の実行時間，正誤はそのファイルに対して正しいヒントが生成されたかどうかである．

正答例では空欄の while 文に対して，中身に当たる if 文による分岐部分が出力されている．一方誤答例では if 文の空欄に対して，return 文周辺が出力されている．誤答のパターンとしては，if や for ループなど使うパーツは同じものの中身が異なるものをヒントとして提示する場合が多かった．また一部特定のパターンが頻繁に出力された．

実験用ファイル 10 個に対してヒントを出力させたが正答率は低く，10 ファイル中 1 ファイルのみの正答となった．平均実行時間は約 0.1 秒であり，本ケーススタディのみを考えれば十分な時間であるがデータベースに用いるファイル数の増加によっては処理に 1 秒以上の時間がかかることも考えられる．

5.4 検索ノード数による影響

5.3 節で示した出力結果における正答率の低さを改善できるかどうかを調査する為，検索に用いるノード数を変更して同様のケーススタディを行った．先述のケーススタディで使用された検索ノード数が 3 の場合に加えて，1 と 5 の場合を調査した．これは，ノード数の増加によってより正確な検索を行い，より正確なヒントが得られることを期待したものである．この場合の結果を以下の表 5.2 に示す.

出力されたヒントには多少変化が見られ，検索ノード数が増えるほど多様なコードが見られたが正答率は検索ノード数に関わらず 10%と変化しなかった．実行時間に関しては，検索ノード数 1 と 3 の間では明確な違いは見られなかったものの検索ノード数が 5 になると大幅に増加した．

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int i = 1;

    printf("fizzbuzz count (1~100)\n");
    while(i<101){
        // Please Get Hint Here!
    }
    return 0;
}
```

図 5.2　正答例 (入力ファイル)

```
--------- compleate!  ----------
time: 438.5375819206238msec.

    while(i<101){
        printf(i % 15 == 0 ?
            "fizzbuzz\n" : i % 5 == 0 ?
                "buzz\n" : i % 3 == 0 ?
                    "fizz\n" : "%d\n",i);
        i++;
    }
fintime: 447.89957666397095msec.
```

図 5.3　正答例 (出力ヒント)

```c
#include <stdio.h>
int main()
{
    int i, n ;
    printf("Enter the number :");
    scanf("%d", &n);
    for (i=1; i<=n; i++)
    {
        // number divisible by 3 and 5 will always be divisible by 15, print 'FizzB
        if (i%15 == 0)
            //please get hint here!

        // number divisible by 3, print 'Fizz' in place of the number
        else if ((i%3) == 0)
            printf("Fizz\t");

        // number divisible by 5, print 'Buzz' in place of the number
        else if ((i%5) == 0)
            printf("Buzz\t");

        else // print the number
            printf("%d\t", i);
    }
    return 0;
}
```

図 5.4　誤答例 (入力ファイル)

```
kind: ReturnStmt
line: 15
--------- output hint ---------
begin : 15   end : undefined
---------------------------------
---------- hint code -----------
---------------------------------
--------- compleate!  ----------
time: 259.0355567932129msec.

    return 0;
}
```

図 5.5　誤答例 (出力ヒント)

表 5.1　実験結果 (検索ノード数:3)

入力ファイル番号	実行時間 (ms)	正誤
0	169.65	○
1	117.01	×
2	94.25	×
3	94.33	×
4	59.85	×
5	55.11	×
6	44.48	×
7	出力なし	×
8	83.87	×
9	109.99	×
	平均実行時間 (ms)	正答率 (%)
	92.06	10

表 5.2　検索ノード数ごとのヒント生成の正答率と平均実行時間

検索ノード数	平均実行時間 (ms)	正答率 (%)
1	92.06	10.00
3	102.71	10.00
5	295.22	10.00

5.5 検索アプローチによる影響

また，ノードの検索アプローチを変更した場合に関しても調査した.「検索ノードの親も検索に含めるようにした実装」,「検索部分の親ノードを根としてに部分木全体の比較を試みた実装」「上記両方を含めた実装」の 3 つの実装について，検索ノード数を一律 3 として比較を試みた．これは，兄弟ノードだけではない情報を検索に含めることによって，より正確なヒントが得られることを期待したものである．結果を表 5.3 に示す.

こちらも出力には変化が見られ，比較的多様なヒントが出力されたが，やはりどのアプローチでも正答率は改善されず，1 割を超えるものは現れなかった．こちらは実行時間に関しては単純な親ノードの検索が最も実行時間が長くなるという結果になった.

5.6 失敗要因の考察

本研究のアプローチは，正答率の面で決して良好な結果を得られなかった．ここでは，その原因として考察されるものを述べる.

1. **子ノードの軽視**

   本研究のアプローチにおいては，発見されたノードの兄弟ノードを重視した反面，子ノードの探索及び比較は積極的に行わなかった．そのため「兄弟ノードが似ているが子ノードが違う」ノードが割り込む可能性は高いと考える.「if 文や for 文など使うべきパーツは合っているが，その中身が異なる」コードが出力される原因と考える.

2. **検索ノードの選択**

   検索するノードは「学習者がコマンドを呼んだ時のカーソル行」から取得していたが，コードによっては一行に長い処理が描かれるなどして，括弧内の記述に対してヒントが求められることもある．現状のコードでは「カーソルの行」のみを取得しているため，一番根に近いノード，すなわち行の一番外側部分を示すノードを選んでしまうと考えられる．カーソルの行だけではなく文字位置の情報も利用し，注目ノードを特定してより小さな部分木に注目することで，

表 5.3　各アプローチごとの正答率及び平均実行時間

内容	平均実行時間 (ms)	正答率
親ノード調査	196.75	10%
部分木調査	171.15	10%
両方	125.22	0%

より具体的なヒントを提示できたと考える．

3. **編集の想定不足**

学習者のコードはあくまで「編集中」であり，完成形のコードではないため過去の学生の回答コードと学習者のコードを比較した場合，完全一致する部分ではなく，学生が今後書くであろうと想定される部分と一致させなければならない．現在の兄弟コードに注目するアプローチの場合，学習者の検索ノードと過去の回答ノードの間ではほぼ確実に齟齬が出てしまう．

4. **データ数の不足**

本研究で用いた比較用のコードファイルの数は 10 と少なく，正確な比較を行うためには不十分であった可能性が高い．出力されたヒントが一部のコードに偏る要因になったと考えられる．

これらの原因から推察するに，単純なアルゴリズムを用いてフィードバックシステムの実装を試みる場合，兄弟ノードに注目するアプローチではなく，小さな部分木に注目して比較するアプローチを考慮して改修すべきであると考える．

6. 結言

本論文では簡易的な自動フィードバックのアルゴリズムを実装し，そのシステムのフィードバックの正確さについてケーススタディを行った. 入力ファイルを抽象構文木化して，注目ノード及びその兄弟ノードをパターンとして過去の回答コードファイル群から検索するアプローチをとったが，適切なヒントが得られるケースは多くなかった．その後検索ノード数等の条件を変更してケーススタディを実行したが，どの場合でも芳しい結果は得られなかった.

本研究の改善案としては，以下の 3 つが挙げられる.

1. 検索ノードをより小さな部分木に注目するように改善し，小さな部分木同士で比較を行う.
2. 兄弟ノードのみでは，子ノードの比較も適切に行う.
3. 編集の想定ができるように検索ノードの後ノード（注目部分の後に描かれるコード部分のノード）のマッチングには特別な処理を行う.

謝辞

本論文を執筆するにあたり，研究の指導から論文の校正に至るまで全面的にお支えいただいた本学情報工学・人間科学系，水野修教授，崔恩瀞助教に厚く御礼を申し上げます.

本論文の校正にご協力いただいた岡山大学大学院自然科学研究科の西浦生成特任助教，サイボウズ株式会社の國領正真さん，本論文執筆のアドバイスをいただいたソフトウェア研究室の皆様，執筆の支えとなった友人や家族に深く感謝いたします.

参考文献

[1] T.W. Price, Y. Dong, and D. Lipovac, “isnap: towards intelligent tutoring in novice programming environments,” Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, pp.483–488, 2017.

[2] D. Garcia, B. Harvey, and T. Barnes, “The beauty and joy of computing,” ACM Inroads, vol.6, no.4, pp.71–79, 2015.

[3] W. Dann, D. Cosgrove, D. Slater, D. Culyba, and S. Cooper, “Mediated transfer: Alice 3 to java,” Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, pp.141–146, 2012.

[4] T.W. Price and T. Barnes, “Comparing textual and block interfaces in a novice programming environment,” Proceedings of the eleventh Annual International Conference on International Computing Education Research, pp.91–99, 2015.

[5] T.W. Price, Y. Dong, and T. Barnes, “Generating data-driven hints for open-ended programming,” Proceedings of the International Conferance on Educational Data Mining Society, pp.191–198, 2016.

[6] A.T. Corbett and J.R. Anderson, “Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes,” Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp.245–252, 2001.

[7] A. Corbett, “Cognitive computer tutors: Solving the two-sigma problem,” Proceedings of the International Conference on User Modeling, pp.137–147, 2001.

[8] Microsoft, Visual Studio Code - Code Editing. Redefined, Visual Stdio Code（オンライン）, 入手先〈https://code.visualstudio.com/〉（参照 2022-2-6）.

[9] J. Stamper, T. Barnes, L. Lehmann, and M. Croy, “The hint factory: Automatic generation of contextualized help for existing computer aided instruction,” Proceedings of the 9th International Conference on Intelligent Tutoring Systems Young Researchers Track, pp.71–78, 2008.

[10] LLVM Developer Group, Clang: a C language family frontend for LLVM, Clang（オ

ンライン), 入手先〈https://clang.llvm.org/〉(参照 2022-2-7).

[11] OpenJS Foundation, Node.js とは, Node.js (オンライン), 入手先〈https://nodejs.org/ja/about/〉(参照 2022-2-2).