

C言語初心者教育のための空欄補充問題の生成

—TF-IDF と正規表現を用いた出題方式—

2023 年 2 月 14 日

1 はじめに

プログラミング演習問題を自動生成、自動採点する方法を確率して教員の労力削減を実現しようと試みる研究がある。[1], [2] 当分野での演習問題の形式は様々であり、プログラムに空欄を生成する空欄補充問題型、プログラムに間違った構文を生成する誤りプログラム型、プログラムを一から作成させる全記述型などがある。

当研究では、空欄補充問題型のプログラムを TF-IDF と正規表現を用いて生成することで、労力の削減を達成できるか分析する。

2 目的

空欄補充問題型のプログラムを TF-IDF と正規表現で生成し、空欄補充問題の生成で有効打となるか分析する。さらに、解答された空欄補充問題を自動採点できるように設計する。上記を実現することで教員の労力削減を目指す。

3 TF-IDF について

TF-IDF とは、文書内の単語の重要度を示す値である。この値が高いほど文書内で重要な単語となり、逆にこの値が低いとそこまで重要ではないということになる。本来は機械学習など AI の分野で広く使われている手法だが、プログラムを文書に見立てることで同じように TF-IDF を求める。TF-IDF は N_i を文書内の任意の単語と定義し

次のような式で得る。

$$TF\text{-}IDF = TF \times IDF$$

TF と IDF は次の値である。

TF : N_i の出現回数

IDF : N_i が何個の文書に出現したかを出現回数で表す。

実際は単語の数だけ繰り返すので次の式となる。

$$TF(t, d) = \frac{n_{t,d}}{\sum_{s \in d} n_{s,d}}$$

TF(t,d) : 文書 d 内のある単語 t の TF 値

$n_{t,d}$: ある単語 t の文書 d 内での出現回数

$\sum_{s \in d} n_{s,d}$: 文書 d 内のすべての単語の出現回数の和

$$IDF(t) = \log \frac{N}{df(t)}$$

IDF(t) : ある単語 t の IDF 値

N : 全文書数

df(t) : ある単語 t が出現する文書の数

$$TF_IDF = TF(t, d) \times IDF(t)$$

TF_IDF : TF-IDF の値

以上のことから、TF-IDF は文書が複数存在しないと求めることができない。そこで、本研究では文書集合として 2019 年度プログラミング A 演習のプログラム計 36 個を使用する。

3 空欄補充問題の生成方法

教員が用意した正解プログラムから空欄補充問題を生成して演習問題を得る。
生成手順は以下のとおりである。

1. 正解プログラムを用意
2. 母集合の全プログラムを分かち書き
3. 各単語の TF-IDF を導出
4. 正規表現で空欄生成エリアを導出
5. 4 からランダムに空欄生成

正規表現とランダム選択法を組み合わせることで不適切な空欄を除去できる。さらに一つのプログラムから複数の空欄補充問題が生成できる。

4 実験

3 章の生成手順を実際に行い有効性を評価する。

4-1 手順 1

正解プログラムを用意する。今回は図 3 のプログラムを空欄対象プログラムとする。
複数の構造体を使用して長方形の面積と長さ、対角線を表示するプログラムである。
プログラミング A 演習は構造体を主に学習するため例題に最も適していると考えた。

```
struct rectangle inputr(void) {
    struct rectangle r;

    printf("Input the coordinate of the lower left corner: ");
    scanf("%d%d", &r.ll.x, &r.ll.y);
    printf("Input the coordinate of the upper right corner: ");
    scanf("%d%d", &r.ur.x, &r.ur.y);

    if ( (r.ll.x >= r.ur.x) || (r.ll.y >= r.ur.y) ) {
        fprintf(stderr, "Inputs are invalid.\n");
        exit(1);
    }

    return r;
}

double area(struct rectangle r) {
    return (r.ur.x - r.ll.x) * (r.ur.y - r.ll.y);
}

double lengthd(struct rectangle r) {
    double vertical = ((r.ur.x - r.ll.x) * (r.ur.x - r.ll.x));
    double side = ((r.ur.y - r.ll.y) * (r.ur.y - r.ll.y));
    return sqrt(vertical + side);
}

int issquare(struct rectangle r) {
    return r.ur.x - r.ll.x == r.ur.y - r.ll.y;
}
```

図 3 正解プログラム

4-2 手順 2

母集合のプログラム計 36 個を分かち書きする。

```
1 include stdio h struct cityTN char name 20 int
2 include stdio h struct cityLN char name 20
3 include stdio h include stdlib h include string t
4 include stdio h include stdlib h define MAX 100
5 include stdio h include stdlib h typedef struct _r
6 include stdio h include stdlib h include string t
7 include stdio h include stdlib h struct node int
8 include stdio h include stdlib h struct node int
9 include stdio h include stdlib h define N typedef
10 include stdio h struct cityTN char name 20 int
11 include stdio h include stdlib h void printarray ir
12 include stdio h include stdlib h include string t
13 include stdio h void moveone int size char dep char
14 include stdio h int kaijou int n int r
15 include stdio h int fn_roop int a int b int c int n
16 include stdio h include stdlib h int calculate int
17 include stdio h define MAX 100 int main void
18 include stdio h int main void int i j tmp
```

図 4 分かち書きの例 (1~18 行まで)

4-3 手順 3

母集合に属する各単語の TF-IDF を導出する。対象プログラムの TF-IDF は表 1 となった。
r,ur,ll など対象プログラムでのみ使用されている単語は TF-IDF 値が高くなっている。対象プログラムでは「double」「struct」が多く出現しているため「int」「void」よりも TF-IDF 値が高くなっている。

単語名	TF-IDF
r	0.6335
ur	0.3690
ll	0.3690
rectangle	0.3122
y	0.2805
x	0.2247
struct	0.1686
double	0.1584
point	0.0851
issquare	0.0851
inputr	0.0851
a	0.0594
vertical	0.0568

side	0.0568
lengthd	0.0568
area	0.0432
-	0.0419
return	0.0318
sqrt	0.0284
math	0.0284
*	0.0253
scanf	0.0210
void	0.0202
>=	0.0198
int	0.0186
include	0.0186
h	0.0186
>	0.0186
<	0.0186
	0.0184
&	0.0174
exit	0.0155
stdlib	0.0130
if	0.0127
+	0.0095
==	0.0077
stdio	0.0064
main	0.0062

表 1 対象プログラムでの TF-IDF

	int	double	char	void	struct
Target	3	8	0	3	14
Total	303	21	73	84	112

表 2 「主要な型」の出現頻度表

-	>=		*	%	<	>	&	+	==
4	1	1	3	2	3	4	2	1	1
40	4	5	135	31	116	210	36	37	77

表 3 「主要な数式記号」の出現頻度表

表 1 には赤い箇所と青い箇所があるが、赤い箇所は対象プログラムでしか使用していない単語であり、青い単語はどのプログラムでも一般的によく使用される単語である。単語名 h は `#include <stdio.h>` で使用されている。

表 2, 表 3 の Target, Total は以下である。

Target: 対象プログラムでの出現回数

Total: 全プログラムでの出現数の合計

「主要な」となっているが、これは母集合のプログラム計 36 個に限定した範囲での話であり、一般的なプログラムを対象としている訳ではない。

「double」の出現回数が全体的に少ないことから、2019 年度のプログラミング A 演習では実数の計算をメインとした問題が少ないことが分かる。

数式記号では TF-IDF が高く設定されたものは無かった。-, >=, || が中間程度に設定されており、== が最も低く設定された。数式記号はどのプログラムでも使用されるため変数名と比較すると TF-IDF は高くなりづらいと考察している。

「*」に関してはポインタの場合と掛け算の場合を区別せずに算出している。このように、同じ単語でも意味が異なる場合は対策できていないので欠点となる。

表 1 より、下位 50% の単語はどのプログラムでもよく見る単語となっていることから、以下の方法で空欄にする単語を決定する。

表1の変数名の中から上位50%の変数名を空欄に置き換える。

表1の数式記号の中から上位50%の数式記号を空欄に置き換える。

4-4 手順 4

空欄にする変数名と数式記号が決定したら空欄にしない構文を決定する。このようにすることで明らかに不適切な空欄を生成しないようにする。

- ・ コメント文
- ・ #で始まる定義
- ・ print 関数の中
- ・ 関数名のプロトタイプ
- ・ 典型的な for 文
- ・ 配列の要素の初期値
- ・ “文字列”

上記の構文中で使用されている変数名を空欄にすると、空欄補充問題としての効果が殆どなかったり、事前に答えを知っていないと分からなかったりするため空欄の対象から外している。

```

# 空欄にしたい領域をインデックスする
def escape(line):
    コメント
    if "/" in line:
        return 1
    elif "<"/>" in line:
        return 1
    #define, printf などの定義
    elif "# " in line:
        return 1
    elif "#print" in line:
        return 1
    関数のプロトタイプ
    elif re.compile(r'([int|void|char|struct|double] (\w*)[<=>Z0-9_]\w*)\.search(line):
        return 1
    elif re.compile(r'([int|void|char|struct|double] (\w*)[<=>Z0-9_]\w*)\.search(line):
        return 1
    典型の for 文を書き for(i=0; i<len; i++) -> ここを空欄にするなら別の場所を空欄にしたいので出題しない
    elif re.compile(r'for \(\w* (int)* (\d+)\) \{\w* \{<=>Z0-9_]\w*\} \w* (\d+)\} <"/>"; i++)' in line):
        return 1
    配列の要素の初期値 x = {1,2,3}; -> 空欄になると推測不可能なので出題しない
    elif re.compile(r'(\w*) = {<"/>};' in line):
        return 1
    "文字列" = "Canada", "America", "Brazil"; -> 空欄になると推測不可能なので出題しない
    elif r'(\w*) in line' and ("donan" not in line):
        return 1
    elif r'(\w*) in line' and ("donan" not in line):
        return 1

```

図5 バインド構文一覧

4-5 手順 5

手順 4 の空欄生成エリアから、空欄にする単語をランダムで 1 箇所空欄にする。

しかし、空欄にした行で同一の単語が複数使用されている場合は全て空欄にする。

最終的に、図 3 の正解プログラムから図 6 の空欄補充問題が生成された。

どの空欄も「不適切である」とは言えない箇所が空欄になっており、演習問題の質を保つことに成功した。

```

struct rectangle input(void) {
    struct rectangle r;

    printf("Write the coordinate of the lower left corner: ");
    scanf("%d %d", &r.ll.x, &r.ll.y);
    printf("Write the coordinate of the upper right corner: ");
    scanf("%d %d", &r.ur.x, &r.ur.y);

    if ( (r.ll.x > r.ur.x) || (r.ll.y > r.ur.y) ) {
        fprintf(stderr, "Inputs are invalid.\n");
        exit(1);
    }

    return r;
}

double area(struct rectangle r) {
    return (r.ur.x - r.ll.x) * (r.ur.y - r.ll.y);
}

double length(struct rectangle r) {
    double vertical = ((r.ur.x - r.ll.x) * (r.ur.x - r.ll.x));
    double side = ((r.ur.x - r.ll.x) * (r.ur.x - r.ll.x)) + ((r.ur.y - r.ll.y) * (r.ur.y - r.ll.y));
    return sqrt(side);
}

int issquare(struct rectangle r) {
    return r.ur.x - r.ll.x == r.ur.y - r.ll.y;
}

```

図 6 生成された空欄補充問題 1

空欄補充問題の仕組みを示す。

以下の項目を全て満たした空欄補充問題が生成される。

項目一覧

- ・ □ □ □ は変数名のいずれかが答えとなる。
- ・ ○ ○ ○ は図 15 に示す数式記号のいずれかが答えとなる。
- ・ 空欄箇所はランダムで生成するので生成する度に異なる問題を得る。(6 章で説明)
- ・ 一つの行に同じ変数名や数式記号が複数存在する場合は全て空欄にする。
- ・ 出題数は変数名がソースコード中の変数名の総数/2、数式記号はソースコード中の数式記号の総数/2 を目安とする。

5 空欄補充問題生成アプリ作成

本研究の目的は教員の労力削減である。そのため、質の高い演習問題を作成できても可用性が低いのであれば本末転倒となってしまう。そこで、3 章の生成方法を一つに統合してアプリを作成した。

空欄補充問題作成アプリ

空欄対象プログラムを選択

9-3.c

空欄補充問題を生成

↓ 生成ボタンを押す

空欄補充問題作成アプリ

空欄対象プログラムを選択

9-3.c

空欄補充問題を生成

ソースコード

空欄の行:計16行

空欄補充問題

ファイルをダウンロード

図 7,8 生成アプリの画面構成

空欄プログラムの選択では、計 36 個のプログラムを登録しており、スライドして選択

することができる。(図 9)

「ソースコード」を押すと正解プログラムが表示される。(図 10)

「空欄の行」を押すと空欄が生成された箇所が表示される。(図 11)

「空欄補充問題」を押すと空欄補充問題が表示される。(図 12)

「ファイルをダウンロード」を押すと空欄補充問題がダウンロードされる(prob.c)。

空欄補充問題作成アプリ

空欄対象プログラムを選択

9-3.c

2-1.c

2-2.c

2-3.c

3-2.c

3-3.c

4-1.c

4-2.c

4-3.c

図 9 空欄対象プログラム選択画面

ソースコード

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

struct rectangle inputr(void);
double area(struct rectangle r);
double lengthd(struct rectangle r);
int issquare(struct rectangle r);

struct point
{
    double x;
    double y;
};

struct rectangle
{
    struct point ll;
    struct point ur;
};

int main(void)
{
    struct rectangle a;
    printf("A:\n");
    a = inputr();
    printf("The area of A is %f\n", area(a));
    printf("The length of the diagonal of A is %f\n", lengthd(a));
    printf("A is ");

    if (!issquare(a))
        printf("NOT ");
}
```

図 10 「ソースコード」表示画面

空欄の行: 計15行	
(行番号,置換文字):	(10,point)
(行番号,置換文字):	(12,x)
(行番号,置換文字):	(13,double)
(行番号,置換文字):	(19,ur)
(行番号,置換文字):	(25,rectangle)
(行番号,置換文字):	(27,inputr)
(行番号,置換文字):	(32,issquare)
(行番号,置換文字):	(42,struct)
(行番号,置換文字):	(45,ll)
(行番号,置換文字):	(47,r)
(行番号,置換文字):	(49,>=)
(行番号,置換文字):	(67,*)
(行番号,置換文字):	(68,y)
(行番号,置換文字):	(69,vertical)
(行番号,置換文字):	(75,-)

図 11 「空欄の行」表示画面

```

struct rectangle inputr(void)
{
    struct rectangle r;

    printf("Input the coordinate of the lower left corner: ");
    scanf ("%lf,%lf", &r.ll.x, &r.ll.y);
    printf("Input the coordinate of the upper right corner: ");
    scanf ("%lf,%lf", &r.ur.x, &r.ur.y);

    if( (r.ll.x />= r.ur.x) || (r.ll.y />= r.ur.y) )
    {
        fprintf(stderr, "Inputs are invalid.\n");
        exit(1);
    }

    return r;
}

double area(struct rectangle r)
{
    return (r.ur.x - r.ll.x) * (r.ur.y - r.ll.y);
}

double lengthd(struct rectangle r)
{
    double vertical = (r.ur.x - r.ll.x);
    double side = ((r.ur.y - r.ll.y) * (r.ur.y - r.ll.y));
    return sqrt(side);
}

int issquare(struct rectangle r)
{
    return r.ur.x - r.ll.x == r.ur.y - r.ll.y;
}

```

図 13 生成された空欄補充問題 2

```

ソースコード

#include<stdio.h>
#include<stdlib.h>
#include<math.h>

struct rectangle inputr(void);
double area(struct rectangle r);
double lengthd(struct rectangle r);
int issquare(struct rectangle r);

struct rectangle
{
    double ll.x;
    double ll.y;
    double ur.x;
    double ur.y;
};

struct rectangle
{
    struct point ll;
    struct point ur;
};

int main(void)
{
    struct rectangle a;
    printf("A:\n");
    a = inputr();
    printf("The area of A is %f\n", area(a));
    printf("The length of the diagonal of A is %f\n", lengthd(a));
    printf("A is ");

    if (issquare(a))
        printf("NOT ");
}

```

図 12 「空欄補充問題」表示画面

```

struct rectangle inputr(void)
{
    struct rectangle r;

    printf("Input the coordinate of the lower left corner: ");
    scanf ("%lf,%lf", &r.ll.x, &r.ll.y);
    printf("Input the coordinate of the upper right corner: ");
    scanf ("%lf,%lf", &r.ur.x, &r.ur.y);

    if( (r.ll.x />= r.ur.x) || (r.ll.y />= r.ur.y) )
    {
        fprintf(stderr, "Inputs are invalid.\n");
        exit(1);
    }

    return r;
}

double area(struct rectangle r)
{
    return (r.ur.x - r.ll.x) * (r.ur.y - r.ll.y);
}

double lengthd(struct rectangle r)
{
    double vertical = (r.ur.x - r.ll.x);
    double side = ((r.ur.y - r.ll.y) * (r.ur.y - r.ll.y));
    return sqrt(side);
}

int issquare(struct rectangle r)
{
    return r.ur.x - r.ll.x == r.ur.y - r.ll.y;
}

```

図 14 生成された空欄補充問題 3

生成する度に異なる箇所が空欄になるため一つのプログラムから複数の空欄補充問題が生成できる。本論文の図 6,図 13,図 14 は空欄箇所が異なっている。

ダウンロードした .c ファイル(prob.c)をそのまま演習問題として使用するため空欄箇所をコメントアウトして解答の際に見やすくしている。

6 空欄補充問題の採点方法

学習者が提出した解答プログラムを回収し、実行することで実行結果を得る。本研究では Linux で for 文を組み、全員の解答プログラムをまとめて実行することで新たな実行結果ファイルを生成している。これは、実行結果のねつ造に対する対抗策でもある。上記の解答プログラムと実行結果から以下の採点内容を出力するプログラムを作成した。

- ・相違点
- ・判定
- ・総合点数
- ・各空欄の正誤判定

相違点

提出されたプログラムと模範解答の相違点（‘ ’と¥t は相違点としない）

判定

実行結果が同じなら正解,異なるなら不正解としている。

総合点数

(正解した空欄の合計点数÷総合点)×100

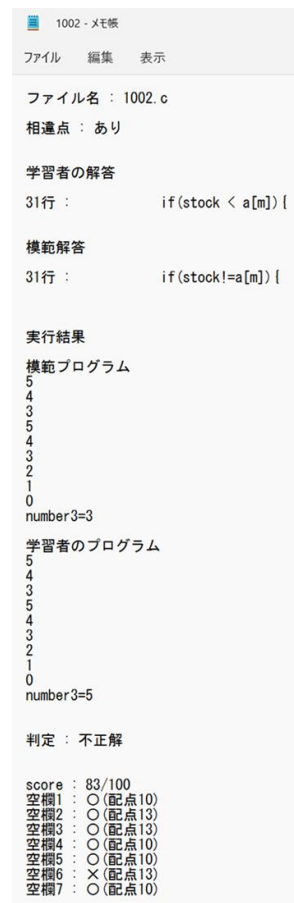
各空欄の配点は以下の式で設定した

空欄の配点 = 10 + (間違えた学習者の人数 ÷ 学習者の人数) × 10

各空欄の正誤判定

模範解答と同じ解答なら○, 異なるなら×

生成されるファイルは図 14 のとおりである。



```
1002 - メモ帳
ファイル 編集 表示

ファイル名 : 1002.c
相違点 : あり

学習者の解答
31行 :          if(stock < a[m]){

模範解答
31行 :          if(stock!=a[m]){

実行結果
模範プログラム
5
4
3
5
4
3
2
1
0
number3=3

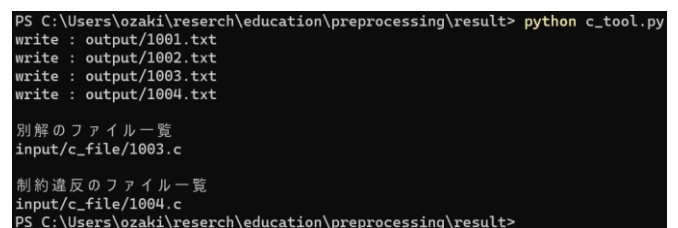
学習者のプログラム
5
4
3
5
4
3
2
1
0
number3=5

判定 : 不正解

score : 83/100
空欄1 : ○ (配点10)
空欄2 : ○ (配点13)
空欄3 : ○ (配点13)
空欄4 : ○ (配点10)
空欄5 : ○ (配点10)
空欄6 : × (配点13)
空欄7 : ○ (配点10)
```

図 14 生成された採点ファイル

別解の可能性がある場合・空欄以外の箇所
が変更されている場合は該当ファイル名を
表示する。空欄以外の箇所が変更されてい
るファイルは不正扱いとしている。



```
PS C:\Users\ozaki\reserch\education\preprocessing\result> python c_tool.py
write : output/1001.txt
write : output/1002.txt
write : output/1003.txt
write : output/1004.txt

別解のファイル一覧
input/c_file/1003.c

制約違反のファイル一覧
input/c_file/1004.c
PS C:\Users\ozaki\reserch\education\preprocessing\result>
```

図 15 コマンドプロンプト表示内容

7 空欄補充問題採点アプリ作成

採点プログラムに関しても可用性を考慮してアプリ化した。

空欄補充問題採点アプリ

ans.c (提出ファイル数上限: 1)

Drag and drop file here
Limit 200MB per file • C

Browse files

ans.txt (提出ファイル数上限: 1)

Drag and drop file here
Limit 200MB per file • TXT

Browse files

prob.c (提出ファイル数上限: 1)

Drag and drop file here
Limit 200MB per file • C

Browse files

student_c.zip (提出ファイル数上限: 1)

Drag and drop file here
Limit 200MB per file • ZIP

Browse files

student_txt.zip (提出ファイル数上限: 1)

Drag and drop file here
Limit 200MB per file • ZIP

Browse files

↓ ファイルを提出

student_c.zip 508.0B

student_txt.zip (提出ファイル数上限: 1)

Drag and drop file here
Limit 200MB per file • ZIP

Browse files

student_txt.zip 508.0B

採点を開始

↓ 採点ボタンを押す

student_txt.zip (提出ファイル数上限: 1)

Drag and drop file here
Limit 200MB per file • ZIP

Browse files

student_txt.zip 508.0B

採点を開始

別解のファイル一覧

input/c_file/1003.c

制約違反のファイル一覧

input/c_file/1004.c

ファイルをダウンロード

図 16,17,18 採点アプリの画面構成

必要なファイルを全て提出したときに採点ボタンが出現する。

採点ボタンが押されると採点ファイルが生成され、ダウンロードボタンが出現する。

ダウンロードボタンを押すと人数分の採点ファイルが zip ファイルでダウンロードされる。zip ファイルの中には図 14 の txt ファイルが人数分格納されている。

このアプリでは 5 つのファイルを提出する。

- ans.c
- ans.txt
- prob.c
- student_c.zip
- student_txt.zip

ファイル名や拡張子が異なるとエラーメッセージを表示する。

ans.c

空欄補充問題の元としたプログラムを提出

ans.txt

ans.c の実行結果を提出

prob.c

ans.c から生成した空欄補充問題を提出

student_c.zip

学生が解答した空欄補充問題を提出

人数が多いため zip ファイル形式でまとめて提出する。

zip ファイルの中に.c 以外のファイルがある場合はエラーメッセージを表示する。

student_txt.zip

student_c の実行結果を提出

人数が多いため zip ファイル形式でまとめて提出する。

zip ファイルの中に.txt 以外のファイルがある場合はエラーメッセージを表示する。

8. 当研究のまとめ

当研究では、空欄対象プログラムでのみ頻繫に使用されている変数名を優先的に空欄にする手法を用いた。空欄生成箇所の選択では正規表現とランダム選択法を組み合わせることで演習問題の使い回しができるようになった。空欄補充問題の採点では、実行結果を txt ファイルに格納しておくことで効率的に自動採点する仕組みを実装した。しかし、別解の自動採点は困難であった。

9. 当研究の課題

TF-IDF は変数の字面しか見ていないので、変数の意味まで考えることができていない。プログラムでは別の関数で同じ名前の変数名が使用されることが多く、その場合は字面が同じでも役割は全く異なることがある。関数ごとに分けて TF-IDF を算出した方がよさそうだが、TF の母数が少なくなり振幅が異常な程大きくなってしまう。構文ごとに TF-IDF を算出する方法を実現できるとこの問題を改善できるだろう。[3] 空欄補充問題の採点においては、ソースコードの構文は模範プログラムと異なっているが、実行結果は同じだったときに、別解となりえるか判別できない点が挙げられる。当研究では別解を極力減らすために、空欄は一つの行で最大でも「1 変数名」または「一つの数式記号」にしか生成しないようにしているが、それでも別解となりえる解が発生する可能性が僅かながら存在する。現状では別解の可能性あることを表示しているが、別解に対して完璧に対応する方法は課題となる。

10. 参考文献

- [1] 蜂巢 吉成, 吉田 敦, 阿草 清滋 :
プログラムの誤り修正課題および正誤判定プログラムの自動生成(情報処理学会論文誌 教育とコンピュータ Vol.3 No.1 64-78 Feb.2017)
- [2] 柏原 昭博, 久米井 邦貴, 梅野 浩司, 豊田 順一 : プログラム空欄補充問題の作成とその評価
- [3] 小林 勇揮, 水野 修 : N-gram IDF を利用したソースコード内の特徴的部分抽出手法(ソフトウェア・シンポジウム 2017 in 宮崎)