N-008

図式操作を用いたアルゴリズムの理解度把握手法
A Method to Grasp Understanding of Algorithm with Diagram Operations

稲葉 大祐† 原田 史子‡ 島川 博光‡
Daisuke Inaba Fumiko Harada Hiromitsu Shimakawa

1. はじめに

現在，情報教育を実施している大学では，プログラミング教育においてC言語が広く採用されている．C言語は，コーディングにおいて自由度の高いプログラミング言語であり，多様な手続きが記述できる．その反面，C言語で実用的な機能を実現するためには，基礎的な文法だけでなく，種々のアルゴリズムの実現方法を正しく習得する必要がある．それゆえ，C言語教育では，文法だけでなく，アルゴリズムの教授にも重点が置かれるが，前者よりも後者の学習過程で行き詰まる学習者が多い．

教員が学習者の理解度をきめ細かく把握する手段として，頻度高くテストすることが有効である．一般的に，プログラミング技法の理解度は，プログラミングの実技テストによって判断されることが多い．しかし，プログラミングによるテストでは，プログラミング言語の基礎的な文法の理解度によっても結果が左右されるため，アルゴリズムの純粋な理解度を把握することが難しい．

アルゴリズムの理解度を把握するために，教育現場で使用されているシステムとして，MA&DA[1] とTRAKLA2[2] が挙げられる．MA&DA と TRAKLA2 は，共にアルゴリズムの学習環境を学習者に提供し，アルゴリズムを図式で表現する．しかし，MA&DA は多機能さのゆえに操作が複雑になり，アルゴリズムの理解度を把握するテストには不向きである．TRAKLA2 は短時間での実施は可能だが，フィードバックにおいて，間違えた場合は間違えた部分が表示されるだけなので，間違えた原因を特定するためには，内容が不十分である．

そこで本論文では，図式の操作を用いてアルゴリズムの理解度を把握するテスト手法を提案する．本手法では，機能を単純にしたツールを用いてテストを実施し，採点を自動化することにより，短時間でテストを実施できる．さらに，フィードバックにおいて，図式に対して行った操作を分類することにより，間違えた場合にその原因を特定するための情報を提示することができる．

2. C言語プログラミング教育

2.1 プログラミング教育の現状

大学のC言語プログラミング教育において一般的に，学習者はC言語の文法を一通り学習した後，C言語を用いて種々のアルゴリズムの実現方法を演習する．演習において，学習者はアルゴリズムを理解しながらソースコードを作成し，教員は学習者が作成したソースコードを採点する．つまり，教員は現在ソースコードだけで学習者の理解度を判断している．

学習者はソースコードを作成するさい，しばしばつまずく．つまずきの原因はひとりひとり違い，学習者はときおり「何が分からないかが分からない」といった状態に陥る．その状態を解決し理解するために，学習者は自身の理解度を知る必要がある．しかし，教員ひとりに対する学習者の人数が多いため，教員が学習者ひとりひとりの理解度を把握し，通知するためには莫大な時間と手間がかかる．さらに，学習者の理解度は日々変化していくので，テストなどにより学習者の理解度を把握し，テスト後に素早く結果を学習者に示すことを頻繁に実施しなければならない．

学習者がソースコードを作成するさい，理解する必要のある内容は以下の2つに分類できる．

- アルゴリズムの仕組み
- ソースコードの書き方

現状，前者の理解度はソースコードからのみ判断されている．しかし，学習者が後者を理解できていない場合，教員が学習者の前者の理解度を把握し，学習者に通知することは困難である．つまり，アルゴリズムの理解度をソースコードからのみ判断することは難しい．したがって，アルゴリズムの理解度を把握するためには，アルゴリズムをソースコードと切り離して考える必要がある．

2.2 図式によるアルゴリズムの可視化

アルゴリズムをソースコードと切り離して考える方法として，図式を用いてアルゴリズムを可視化する方法が考えられる．学習者の理解を促進させるためにアルゴリズムを可視化することは，有効な手段である [3]．たとえば，プログラミングの場合，アルゴリズムを図式により可視化することで，複雑な処理の流れを一目で見ることが可能になり，ソースコードを1行ずつ読むよりプログラム全体の把握が容易になる．また，図式を用いて学習者のアルゴリズムの理解を支援するシステムも開発されている [4] [5]．そして，教員は講義において，しばしば図式を用いてアルゴリズムを表現し，アルゴリズムの流れを説明する．したがって，学習者に図式を操作させることにより，ソースコードと切り離してアルゴリズムの理解度を把握できると考えられる．しかし，図式を用いて可視化することは，アルゴリズムを図式という別のものに置き換えるため，使用者に誤解を与えてしまう可能性を否定できない．したがって，図式を用いてアルゴリズムを可視化するさい，使用者に誤解を与えないための仕組みが必要となる．

3. アルゴリズムの理解度把握

3.1 図式操作を用いたテスト手法

本論文では，アルゴリズムの理解度を把握するためのテスト手法を提案する．対象者は，C言語を用いて種々のアルゴリズムを実現する方法を学習している学生である．本手法は，アルゴリズムの講義とソースコードの作成方法を学ぶ演習の間に位置する．演習の前にアルゴリ

† 立命館大学大学院 理工学研究科
‡ 立命館大学 情報理工学部

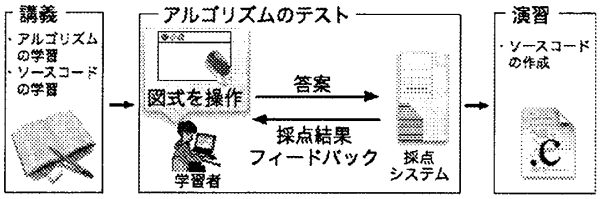

図 1: 手法の概要

ズムのテストを実施することにより，学習者は各々のアルゴリズムの理解度を確認した上でソースコードの作成方法の学習に取り組むことができる．

図 1 に本手法の概要を示す．本手法では，種々のアルゴリズムを図式を用いて表現する．アルゴリズムのテストにおいて，答案作成用のツールを用いて，図式を操作することにより答案を作成し，採点を行う採点システムに提出する．採点後には，学習者にテストの正誤結果を示す採点結果と，学習者の答案に基づくフィードバックが提示される．これにより，頻度の高いテストが可能となり，きめ細かく理解度を把握できる．

3.2 制約内での操作によるテスト

図式を用いたテストをするためには，図式表現による誤解を防ぐために，独自の制約が必要になる．制約を設けない場合，本来不正解であるところを正解と判断してしまうことがある．たとえば単方向リストにおいて，要素を 1 つ作成した後，既存のリストの途中に要素を追加する場合，図 2 の①のような順番で操作を行う．操作の内容は以下の通りである．

(1) 新要素の作成
(2) 新要素のリンクをリスト内のリンクに接続
(3) リスト内の要素のリンクを新要素へ接続

いま，図 2 の②のように，上記の (2) と (3) の操作の順番を逆にしたとする．図式上では新要素とリスト内の要素の双方向が見えているため，新要素からリスト内にあった最後の要素へリンクを接続できる．しかし，プログラム上は (3) の操作を先に実施すると，リスト内の最後の要素のアドレス情報が消失しているので，(2) の操作を実施することができない．つまり，プログラム上では不正解であるが図式上は正解となってしまう．したがって，以下のような制約を付ける必要がある．

- 灰色で塗られた要素：アドレスが既知である
- 白色で塗られた要素：アドレスはリンクをたどって求めなければならない

これら制約を付けることにより，プログラムと図式の両方で図 2 の②は不正解となる．

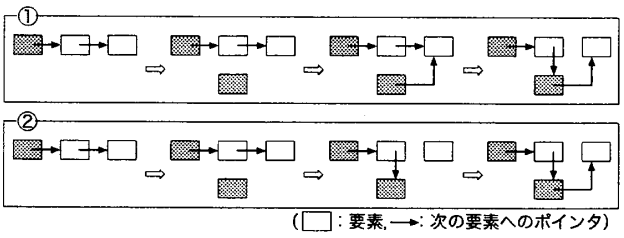

図 2: 単方向リストに要素を追加する例

3.3 図式操作の記録

本論文では，図式の操作によって生成された状態の連接をプロセスと定義する．問題で与えられる図式をプロセスの 0 番目の状態とし，プロセスの i 番目の状態を $p(i)$ と表現する．本手法における答案は，プロセスの状態を記録したものであり，答案の作成は学習者が図式を操作することにより行われる．

単方向リストの途中に要素を 1 つ追加する問題の解答例を図 3 に示す．答案において，各要素は名前付けをすることにより識別され，次の要素へのポインタは矢印を用いて表現される．たとえば $p(0)$ では，要素 A, B, C があり，それぞれの要素のポインタが指す要素はそれぞれ，B, C, null である．このように本手法では，学習者が図式に対して行った操作を要素の名前付けにより文字で表現する．

3.4 採点手法

採点は，学習者が作成した答案と模範解答を比較することにより行われる．採点するために，まず学習者が図式に対して行った操作を答案から抽出する．本論文では，$p(i+1)$ の操作を $p(i)$ から $p(i+1)$ の変化分と定義し，[変化前：変化後] と表現する．たとえば図 3 の答案において，$p(0)$ から $p(1)$ では D → null が新たに増えているので，$p(1)$ の操作は [空白：D → null] である．そして，$p(1)$ から $p(2)$ では要素 D のポインタの指す要素が null から C に変化しているので，$p(2)$ の操作は [D → null：D → C] である．操作の抽出が終わると，抽出された操作ひとつひとつを問題の制約を満たす操作と満たさない操作に分類されつつ，問題の制約を満たす操作だけが $p(0)$ の図式に順次適用され，採点システム上で答案が作成される．その後，採点システム上で作成された答案と模範解答におけるプロセスの最後の状態を比較し，それらが一致して，かつ学習者の作成した答案に，問題の制約を満たさない操作が 1 つも含まれていない場合，正解とする．

3.5 結果の開示とフィードバック

採点後，学習者は採点結果とフィードバックを受け取り，それらの内容をもとに復習をする．フィードバックでは，学習者の答案と模範解答が並べて表示され，図式に対して行った操作の意味が表 1 に基づき分類され，表示される．そして，学習者の答案と模範解答，さらに間違えた場合はその原因を特定するための情報が提示される．たとえばリストにおいて，「アドレスの情報が消失した要素に対してリンクを接続する」という操作を行っている場合，「誤った操作：ポインタの指す先に特定できない要素を設定しています」と表示される．

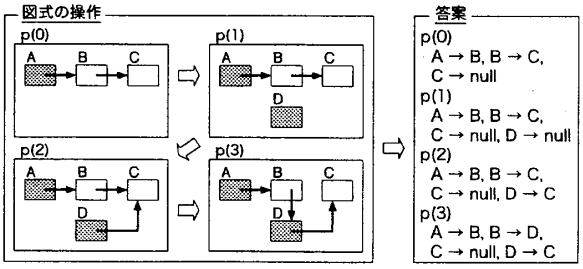

図 3: 操作の記録

表 1: 操作の分類

操作名	意味
正しい操作	問題の制約を満たし，模範解答から抽出された操作に含まれる操作
不要な操作	問題の制約を満たすが，模範解答から抽出された操作に含まれない操作
誤った操作	問題の制約を満たさない操作

4. 評価

4.1 授業への適用内容

本手法をシステムとして実装し，本学の情報理工学部情報システム学科の 1 回生 121 名を対象に実際の授業への適用を行った．適用内容は，リスト構造を講義で学習した後，講義で扱った例題を演習の開始後 20 分でテストするといったものである．問題の内容を以下に示す．

問題 要素が 4 つポインタによって接続されている単方向リストの途中に，要素を追加することを考える．単方向リストの先頭を 1 番目としたとき，単方向リストの 2 番目と 3 番目の要素の間に 1 つ要素を追加する．

制約
- リストの先頭の要素と新たに生成した要素：アドレスが既知である
- その他の要素：アドレスはリンクをたどって求めなければならない

テスト終了後には，テストで使用したツールの使いやすさ，リスト構造の理解度，フィードバックの内容の満足度，アルゴリズムのテストの再度実施の必要性の 4 点について，5 段階評価欄をもつアンケートを実施し，学習者から回答を得た．

4.2 適用結果

テストの結果は，121 人中 80 人が正解という結果であり，不正解であった 41 人中 29 人が，図 3 において，$p(2)$ と $p(3)$ の順番が入れ替わっているような，ポインタのつなぎかえの順番間違いであった．アルゴリズムテストにおいて学習者は，答案作成用のツールを初めて使用したが，ツールの誤操作と思われる答案は 1 つであった．アンケートでは 112 人から回答を得た．表 2 にアンケート結果を示す．5 段階評価において③がどちらでもないことを示し，①に近づくほど否定的な回答，⑤に近づくほど肯定的な回答を示している．テスト後に提示されるフィードバックについて約半数の学生から役に立ったという評価を受けた．そして，再度のテスト実施については，62%の学生から肯定的な評価を得た．この結果から，ソースコードを作成する前にアルゴリズムの理解度を判断し，学習者が自身の理解度を把握することは，学習者にとって重要であることが分かる．

表 2: アンケート結果

	①	②	③	④	⑤
ツールの使いやすさ	4.4%	18.6%	20.4%	43.4%	12.4%
リスト構造の理解度	5.3%	18.6%	13.3%	47.8%	14.2%
フィードバックの内容の満足度	1.8%	8.9%	38.9%	40.7%	8.9%
アルゴリズムのテストの再度実施の必要性	1.8%	8.0%	27.4%	49.6%	12.4%

本手法を実際の授業に適用することにより，既存のリストの途中に要素を追加するといった，リスト構造を用いた基本的なアルゴリズムについて，演習開始 20 分で 109 人の理解度を大雑把に把握することができた．以下の内容について，121 人中 80 人の学生は理解できており，29 人の学生は理解できていないことが分かる．

- リスト構造はリンクによって要素が接続されている
- リンクが切れた要素はアドレス情報が消失してしまうため，リンクの切れた要素に対してリンクを張ることができない

大雑把ではあるが 20 分という短時間で学習者の理解度を把握できる本手法は，テスト後に短時間で結果を学習者に提示できるという点において有効である．また，教員の負荷が大きく削減されるため，テストの頻度を高めることも可能となる．

5. おわりに

本論文では，図式操作によりソースコードに依存することなく，アルゴリズムの理解度を把握する手法を提案した．本学の情報理工学部情報システム学科の 1 回生 121 人に対して本手法を適用した結果，112 人のテスト受験者からアンケートの回答を得られ，そのうち 49.6%の学生がフィードバックの内容に満足し，アルゴリズムの理解を確認するうえで役に立ったと回答した．そして，121 人のうち 109 人のアルゴリズムの理解度を把握できた．

今後は，アルゴリズムのテストに複数の難易度を設け，学習者の理解度を詳細に把握することを目指す．

参考文献

[1] Krebs, M., Lauer, T., Ottmann, T. and Trahasch, S.: Student-built Algorithm Visualizations for Assessment: Flexible Generation, Feedback and Grading, *ITiCSE '05*, pp. 281–285 (2005).

[2] Malmi, L., Karavirta, V., Korhonen, A. and Nikander, J.: Experiences on Automatically Assessed Algorithm Simulation Exercises with Different Resubmission Policies, *J. Educ. Resour. Comput.*, Vol. 5, No. 3, p. 7 (2005).

[3] Blumenkrants, M., Starovisky, H. and Shamir, A.: Narrative algorithm visualization, *SoftVis '06: Proceedings of the 2006 ACM symposium on Software visualization*, pp. 17–26 (2006).

[4] Grissom, S., McNally, M. F. and Naps, T.: Algorithm Visualization in CS Education: Comparing Levels of Student Engagement, *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, pp. 87–94 (2003).

[5] Carlisle, M. C., Wilson, T. A., Humphries, J. W. and Hadfield, S. M.: RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving, *SIGCSE Bull.*, Vol. 37, No. 1, pp. 176–180 (2005).