可逆な深さ優先探索

2015SE003 浅野 早紀　2015SE092 山口 春樹
指導教員：横山哲郎

1　はじめに

アルゴリズムとは問題解決のための手法であり，本研究ではアルゴリズムの効率化を目標とする．アルゴリズム自体の効率化を行うことで，そのアルゴリズムを扱う媒体に関わらず，実行時間の短縮を行うことができる．

可逆アルゴリズムとは，各実行状態に移る計算が単射であり，出力をもとに入力を特定できるアルゴリズムのことである．可逆化とは，非可逆もしくは非可逆の可能性があるものに可逆性をもたせることである．可逆性は，非可逆なものの出力に，入力を特定できる情報を追加することで，与えることができる．また非可逆な計算を実行すると必ず熱を発散する．こうした熱の発生量の下限が理論的には存在しないのが，可逆的な計算である [7].

可逆計算はコンピュータの並列計算において，重要な役割を担っている．特に投機的実行の際に可逆計算が行われる．投機的実行ではある仕事が必要かどうか判明する前に，予めその仕事を実行する．するとその仕事が必要と判明した後で，実行するよりも早く仕事を処理することができる．もし，その仕事が不要だった場合，可逆計算によって実行状態を元に戻すことができる．

しかし，現時点で基本的な可逆アルゴリズムの整備は，まだ不十分である．代表的なアルゴリズムの一つである線形探索は，配列版，スタック版について一般解法による可逆化が行われている．また一般解法より効率が良いアルゴリズムも，手動によって提案されている [3]．今回，整備が行われていないアルゴリズムの中で，代表的なグラフアルゴリズムの 1 つである深さ優先探索に着目する．深さ優先探索はアルゴリズムの構造がシンプルで実装が用意であるからである．可逆な深さ優先探索は先行研究である可逆線形探索の問題に当てはめることができると考えられ，この研究に取り組んだ．しかし，可逆線形探索は対象の問題のリストや配列が一直線上にあることを前提としているが可逆な深さ優先探索はグラフを対象とするので必ずしも適用できない．深さ優先探索のグラフが一直線上の場合，可逆線形探索の問題として扱うことができる．ただし，可逆線形探索は探索する対象の配列が 1 つに対して，可逆な深さ優先探索では配列が 3 つ必要である．そこで可逆な深さ優先探索ではリンクによる隣接リスト表現によって解決しようと試みる．

本研究の目的は，深さ優先探索を一般解法を用いて可逆化することである．またステップ数，メモリ使用量，ゴミ出力の観点から深さ優先探索の一般解法を改良して，解析を行うことを目標とする．

この研究のアプローチは，深さ優先探索アルゴリズムのプログラムを C 言語で実装し，そのプログラムを Janus 言語で書き換えることで可逆化することである．Janus 言語で書いたプログラムは，可逆性が保証されている．可逆化には，埋め込み，計算-コピー-逆計算法といった一般解法を用いる．一般解法の解析を行った後，ステップ数，メモリ使用量，ゴミ出力の観点から効率が良くなるように，プログラムを改良し解析する．

2　関連研究

2.1　グラフ

グラフは頂点と辺の集合で表されるデータ構造である．頂点の集合を V，辺の集合を E として，グラフは $G = (V, E)$ で表す．グラフには無向グラフと有向グラフがある．有向グラフの辺には向きがあり，無向グラフの辺には向きがない．

2.2　深さ優先探索

深さ優先探索とは，グラフ探索アルゴリズムの 1 つであり，未探索の隣接する頂点を次々に探索するものである．文献 [2] では深さ優先探索の基本的なアルゴリズムが説明されている．各頂点の深さは，グラフの始点からその頂点までの最短距離（最小の辺の数）で表す．深さ優先探索のアルゴリズムを以下に示す．

最初に始点の頂点を選択する．この時点で，始点は探索済とする．その頂点と隣接する頂点のうち，未探索の頂点を探索して，その頂点を探索済とする．隣接する頂点がすべて探索済のときは，1 つ前の頂点に戻り，その頂点を再び出発点にして探索する．探したい頂点が見つかるか，すべての頂点を探索したら探索終了となる．

2.3　深さ優先探索のプログラム

```
1  int traverse1(int k, int key)
2  {
3      int t;
4      visited[k] = 1;
5      if(k == key)
6          f = 1;
7      for (t = adj[k]; (next[t] != 0) && (f == 0)
           ; t = next[t]){
8          if (!visited[val[t]])
9              traverse1(val[t],key);
10     }
11     return f;
12 }
```

図 2.1　深さ優先探索の C プログラム

図 2.1 は，探索したい頂点が見つかるまで探索を続けるプログラムである．このプログラムは文献 [1] を参考にしている．深さ優先探索の再帰プログラムを扱う．

k は探索する頂点，key は探索したい頂点を表している．配列 visited は探索済の場合 1，未探索の場合 0 を格納する．5–6 行の if 文で，探したい頂点の値が見つかったら f に 1 を代入する．7–10 行の for 文で隣接する頂点が未探索の場合，再帰的にプログラムを繰り返し，隣接する頂点を次々に探索するようになっている．出力として f を返す．f の値は探索成功で 1，失敗で 0 である．

3 可逆

3.1 単射

単射の定義は，「任意の関数 $f{:}X \to Y$ について，任意の $a, b \in X$ に対して，$a \neq b$ ならば，$f(a) \neq f(b)$」である．この命題の対偶をとると，「任意の $a, b \in X$ に対して，$f(a) = f(b)$ ならば，$a = b$」となる．整数型をもつ変数 x が，a の値を持つことを $\{x \mapsto a\}$ と表す．

3.2 可逆プログラミング言語

文はその意味が単射であるとき，可逆であるという．また，すべての文が可逆であるようなプログラミング言語を，可逆プログラミング言語という．可逆プログラミング言語で書かれたすべてのプログラムは，可逆性が保証される．

3.3 埋め込み

埋め込みは，Landauer 法とも呼ばれる方法である．埋め込みでは，代入など非可逆な計算で失われてしまう情報を保存する．出力と保存した情報を合わせることで入力を復元して，可逆化を行う．Janus では情報が失われる前に，スタックに情報を保存することで可逆になる．ただし埋め込みでは，計算途中で情報を保存するので，出力としてゴミ情報と呼ばれる余分な情報が残ってしまう欠点がある．

3.4 計算-コピー-逆計算法

計算-コピー-逆計算法は，call で順計算をした後，必要な情報のみを保存した後，uncall で逆計算をする方法である．uncall による逆計算によって，実行前の状態に戻し，スタックにたまったゴミ情報を綺麗にすることができる．ただし，計算を 2 度行ったりデータをコピーをしたりと，実行時間は 2 倍以上になってしまう欠点がある．

4 Janus 言語

Janus は命令型可逆プログラミング言語である．可逆なプログラムのみを記述でき，非可逆なものは記述できない．

まず代入に関して記す．代入は情報を上書きする操作であり，操作前の情報を失ってしまうので，非可逆である．そのため可逆プログラミング言語でそのまま用いることができない．

Janus では代入の際，加算代入演算子 +=，減算代入演算子 -=，排他的論理和を表す ^=を使用する．ただし，代入式の左辺に現れた変数が，右辺に現れてはいけないという制約がある．これは，可逆性を保証するために設けられている．

条件式 if e1 then s1 else s2 fi e2 について記す．if に続く式 e1 が真であれば，then の後の文 s1 を実行して，偽であれば，else の後の文 s2 を実行する．その後，Janus では fi に続くアサーションを評価する．e1 が真であれば e2 は真でなければならず，e1 が偽であれば，e2 も偽でなければならない．なお Janus では真を非 0，偽を 0 で表す．

5 一般解法

書き換え規則を用いて深さ優先探索のプログラムを Janus 言語で書き換えた [3]．代入の場合は，push(x, g) を代入の前に付け加える．変数宣言の場合は，delocal の前に push(x, g) を追加する．if 文にはアサーションを用意して，then 節には push(1,g)，else 節には push(0,g) を書き加える．繰り返し文では，from 文の前に，push(1, g) を加え，from の後にアサーションを書き加え，loop 文の最後には push(0, g) を加える．

仮引数として，k, key, f, g, adj, visited, next, val を置く．adj，visited，next，val は整数型の配列である．Janus ではプログラムの可逆性を保証するために計算の過程でゴミ情報が生じる．このゴミ情報を保存するためのスタック g を用意している．

k は探索する頂点の値で，key は探索したい頂点の値である．f は探索したい値が見つかったら 1，見つからなかったら 0 が入る．変数 i を用いて説明すると，adj[i] は頂点の値を格納する val の場所を格納している．visited[i] は頂点 i が探索済の場合は 1，未探索の場合は 0 を格納する．next[i] は頂点 i に隣接する頂点の場所を格納して，val[i] は頂点の値が格納されている．頂点数を n とする．traverse が埋め込みを用いたプログラムで，traverse_b が計算-コピー-逆計算法を用いたプログラムである．

また，if 文，loop 文についてはプログラムの可逆性を保証するためにアサーションを置く必要がある．アサーションを置くことによってプログラム中の if 文，loop 文がどのようにたどってきたかを特定することができる．loop 文のアサーションとして，top(g) = 1 を置く．アサーションを満たすために，loop 文の前に push(1, g)，抜け出す際に，push(0, g) を用意する．

traverse_b では，call で関数を呼び出した後，探したい頂点が見つかったかどうかを表す値だけを保存しておく．その後，uncall で逆計算をして，スタックを綺麗にする．

一般解法のプログラムでは，代入を行う前に代入前の値を保存するため，4 箇所で push 操作をしてスタック g に値を保存している．また if 文のアサーションとして，fi top(g) = 1 を置いている．このアサーションを満たすため，push(1, g)，push(0, g) を then 節，else 節で行っている．from 文のアサーションとして top(g) = 1 を置いている．このアサーションを満たすため，アサーションの後の文で，push(1, g)，push(, g) を行っている．

S は探索が成功したか失敗したかを 1 か 0 で表し（探索成功で 1，失敗で 0 になる），L は探索を何回するかを表すこととする．

走査回数が 1 回の場合を求める．メモリ使用量について，変数を 1 つ用意するのにメモリを 1 使用とする．一般解法では，k，key，f が用意されている．またプログラム内で変数 s，t を用意している．次に長さ n の配列を用意すると，メモリを n 使用とする．配列は adj，visited，next，val の 4 つがプログラムに用意されている．頂点数を n としたとき，配列 adj と visited は n の長さ，配列 val と next は頂点数の 2 倍の長さが必要になる．またスタックに 1 回 push すると，メモリを 1 使用とする．一般解法では，12 回スタックに push している．計算した結果，走査回数 1 回の場合，最大メモリ使用量は $6n+4L-3+11$ となる．次に走査回数 2 回，すなわち計算-コピー-逆計算法の場合である．走査回数 1 回の時のメモリ使用量に加えて，計算-コピー-逆計算法を用いるので，値をコピーするための変数が新たに 1 つ必要になる．また走査回数 1 回のときのスタックへの push 操作は，uncall で呼び出した時，pop 操作になる．そのためその分メモリ使用量は減少する．走査回数 2 回のときのメモリ使用量は $6n+6$ となる．

各行について，ステップ数を求める．ステップ数は do，else を除く各行を 1 回実行することを 1 ステップと数える．走査回数 1 回の場合，各行のステップ数の合計は $T_r(L) = 11L - 6S + 16$ である．

また走査回数 2 回のときは，call と uncall の呼び出しに加えて，値をコピーするステップも存在する．走査回数 2 回のときのステップ数を $T(L)$ とすると $T(L) = 22L - 12S + 40$ となる．

ゴミ出力は，出力に使われるもの以外なので，変数 k，変数 key，配列 visited，配列 adj，配列 val，配列 next，またプログラム内にある変数 t と s とスタック g の分である．走査回数 1 回の場合のゴミ出力は，$6n+4$ である．

走査回数 2 回の場合，プログラム内の変数 s と t は，計算-コピー-逆計算法により，元の状態に戻り再利用できるので，ゴミ出力とならない．走査回数 2 回のときのゴミ出力は $6n+2$ である．表 5.1 は走査回数別にまとめたものである．

表 5.1　一般解法の走査回数による比較

	traverse	traverse_b
走査回数	1 回	2 回
最大メモリ使用量	$6n+4L-3+11$	$6n+6$
ステップ数	$11L-6S+16$	$22L-12S+40$
ゴミ出力	$6n+4L-3S+10$	$6n+2$

6　提案解法

一般解法で可逆化することができているが，一般解法では無駄な部分も存在する．そのため無駄な部分を最適化

```
1  procedure traverse(int k, int key, int f, stack
       g, int adj[],int
2   visited[], int next[], int val[]) \\ 1
3      local int t = adj[k]            \\ 1
4        visited[k] ^= 1               \\ 1
5          if k = key then             \\ S
6            f ^= 1                    \\ S
7          fi k = key                  \\ S
8        from t = adj[k] loop          \\ L+1-S
9          if visited[val[t]]=0 then   \\ L
10           call traverse(val[t], key, f, g, adj,
                 visited,next, val)    \\ L
11         push(1, g)                  \\ L
12       else
13         push(0, g)                  \\ 1-S
14       fi top(g) = 1                 \\ L
15         t ^= adj[k]                 \\ L-S
16         t ^= next[adj[k]]           \\ L-S
17     until next[t] = 0 || f != 0     \\ L+1-S
18     push(t, g)                      \\ 1
19   delocal int t = 0                 \\ 1
20
21 procedure traverse_b(int k, int key, int f,int
       adj[],int visited[], int
22  next[], int val[])                 \\ 1
23   local stack g = nil               \\ 1
24     local int x = 0                 \\ 1
25       call traverse(k, key, x, g, adj, visited
             , next, val)              \\ 1
26       f ^= x                        \\ 1
27       uncall traverse(k, key, x, g, adj,
           visited, next, val)         \\ 1
29     delocal int x = 0               \\ 1
   delocal stack g = nil               \\ 1
```

図 6.1　深さ優先探索:提案解法

する方法を考えていく．ただし，深さ優先探索は訪問印によってその頂点を探索したかどうかを確認するが，可逆であるとその訪問印も途中で元の値に戻ってしまい，訪問印があまり役に立たず，この部分の最適化が難しい．提案解法では，プログラム内の変数を減らし，アサーションに変更を加えた．これによりスタックへの push 操作の回数を減らすことができた．if 文のアサーションについて，一般解法では top(g) = 1 を置いているが，提案解法 1 では 5 行に k = key を置いている．これにより if 文内で push 操作をする必要が無くなる．変数 k と key は if 文内で一度も変更されていないので，アサーションとして使うことができる．from 文のアサーションについて，一般解法では top(g) = 1 を置いているが，提案解法では 8 行に t = adj[k] を置いている．これによりアサーションの前に push 操作をする必要が無くなる．また，from 文内の push 操作を減らすことができる．

図 6.1 のプログラムでは，変数 k，key，f が用意されている．またプログラム内で変数 t を用意している．配列は adj，visited，next，val の 4 つがプログラムに用意されている．また，図 6.1 では， 11，13，18 行でスタックに push している．図 6.1 の走査回数 1 回のときの最大メモリ使用量を計算する．計算の結果，図 6.1 の最大メモリ使用量は $6n+L-S+6$ である．

各行について，ステップ数を求める．図 6.1 の 1–19 行，各行のステップ数の合計を $T_r(L)$ として計算する．図 6.1 の走査回数 1 回の時のステップ数は $T_r(L) = 8L - 2S + 8$ である．

ゴミ出力は，出力に使われるもの以外なので，変数 `k`，変数 `key`，配列 `visited`，配列 `adj`，配列 `val`，配列 `next`，またプログラム内にある変数 `t` とスタック `g` である．スタック `g` には 11，13，18 行で `push` している．すなわち図 6.1 の走査回数 1 回のときのゴミ出力は $6n + L - S + 5$ である．

次に走査回数 2 回，すなわち計算-コピー-逆計算法の場合である．メモリ使用量について，走査回数 1 回の時のメモリ使用量に加えて，計算-コピー-逆計算法を用いるので，値をコピーするための変数が新たに 1 つ必要になる．また走査回数 1 回のときのスタックへの `push` 操作は，`uncall` で呼び出した時，`pop` 操作になる．そのためその分最終的なメモリ使用量は減少する．図 6.1 の走査回数 2 回のときのメモリ使用量は $6n + 5$ となる．

ステップ数について，走査回数 2 回のときは，25，27 行で 1–19 行をそれぞれ呼び出している．加えて値をコピーするステップも存在する．図 6.1 の走査回数 2 回のときのステップ数を $T(L)$ として計算する．図 6.1 の走査回数 2 回のときのステップ数は $T(L) = 16L - 4S + 24$ となる．

ゴミ出力は変数 `k`，変数 `key`，配列 `visited`，配列 `adj`，配列 `val`，配列 `next` である．プログラム内の変数 `t` は，計算-コピー-逆計算法により，元の状態に戻り再利用できるので，ゴミ出力とならない．すなわち図 6.1 の走査回数 2 回のときのゴミ情報量は $6n + 2$ である．表 6.1 は走査回数別にまとめたものである．

表 6.1　提案解法の走査回数による比較

	`traverse`	`traverse_b`
走査回数	1 回	2 回
最大メモリ使用量	$6n + L - S + 6$	$6n + 5$
ステップ数	$8L - 2S + 8$	$16L - 4S + 24$
ゴミ出力	$6n + L - S + 5$	$6n + 2$

また別の指標でメモリ使用量の計算を行う．元の入出力用以外のメモリ使用量を M_1，元の入出力以外のゴミ出力量を M_2 とする．

一般解法では，走査回数 1 回のとき，入力以外に変数を 2 つ用意している．また，スタックへの `push` 操作があり，$4L - 3S + 6$ 回であるので，$M_1 = 4L - 3S + 8$ である．

元の入力以外のゴミ出力は変数 `t` とスタック `g` である．スタック `g` には $4L - 3S + 6$ 回 `push` 操作が行われているので，$M_2 = 4L - 3S + 7$ である．

走査回数 2 回のとき，走査回数 1 回のときの変数 2 つに加えて，値をコピーするための変数を新たに 1 つ用意する．`push` 操作をした後，2 回目の走査では `pop` 操作をしている．そのため $M_1 = 1$ である．走査が 2 回なので，計算-コピー-逆計算法により，元の入力以外のゴミ出力をなくしているので $M_2 = 0$ となる．

提案解法では，走査回数 1 回の場合について，入力以外に変数 t を 1 つ用意している．またスタックへの `push` 操作が 11，13，18 行で行われる．そのため $M_1 = L - S + 3$ である．元の入出力以外のゴミ出力は，変数 `t` とスタック `g` の分である．スタック `g` への `push` 操作は，$L - S + 2$ 回行われるので，そのため $M_2 = L - S + 3$ である．

走査回数 2 回の場合，走査回数 1 回のときの変数に加えて，値をコピーするための変数を 1 つ用意する．走査が 2 回なので，1 回目の走査で 11，13，18 行にスタックに `push` 操作をした後，2 回目の走査では `pop` 操作をしている．そのため $M_1 = 1$ である．走査が 2 回なので，計算-コピー-逆計算法により，元の入力以外のゴミはなくなるので $M_2 = 0$ となる．

7　おわりに

本研究では可逆，深さ優先探索について記述した．C 言語で書いた深さ優先探索を可逆プログラミング言語である Janus で書き換えた．書き換えの際に，埋め込み，計算-コピー-逆計算法といった一般解法を用いた．これにより深さ優先探索の可逆化を行った．また，メモリ使用量，ステップ数，ゴミ出力の観点から，一般解法の改良を行った．

参考文献

[1] Sedgewick, R.: *Algorithms in C*, Addison-Wesley Professional, 1st edition (1990).

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., et al.（著），浅野哲夫，岩野和生，梅尾博司ほか（訳）：アルゴリズムイントロダクション，近代科学社，第 3 版 (2013).

[3] 家崎雄太，水野竣太郎：可逆線形探索，南山大学 2017 年度卒業論文 (2018).

[4] 大堀　淳，Garrigue, J.，西村　進：コンピュータサイエンス入門: アルゴリズムとプログラミング言語，岩波書店 (1999).

[5] Axelsen, H. B. and Yokoyama, T.: Programming Techniques for Reversible Comparison Sorts, *Proc. Programming Languages and Systems* (*APLAS 2015*), Feng, X. and Park, S. (Eds.), Lecture Notes in Computer Science, Vol. 9458, Springer-Verlag, pp. 407–426 (2015).

[6] Carothers, C. D., Perumalla, K. S. and Fujimoto, R. M.: Efficient Optimistic Parallel Simulations Using Reverse Computation, *ACM Transactions on Modeling and Computer Simulation*, Vol. 9, No. 3, pp. 224–253 (1999).

[7] 森田憲一：可逆計算，近代科学社 (2012).