可逆グラフアルゴリズム

2016SE085 鳥居大樹　2016SE098 吉田翔亮
指導教員：横山哲郎

1　はじめに

本研究では，可逆アルゴリズムの効率化の整備が進んでいないグラフアルゴリズムの可逆化および効率化を目標とする．また，本研究では，グラフアルゴリズムの中でも基本的なアルゴリズムである深さ優先探索と幅優先探索を取り扱う．

1.1　背景

計算過程においての可逆とは，直前の状態が高々一意に定まるもので，可逆な場合，状態から状態への遷移が 1 : 1 の関係になっている．アルゴリズムは計算機上で問題を解くための手法である．アルゴリズムには線形探索アルゴリズムやバブルソートをはじめとした多くのアルゴリズムが存在し．その中に連結無向グラフの深さ優先探索や幅優先探索が存在する．実際にプログラムを記述して実行させると計算過程で情報を失うことによって，エネルギーを消費し熱を発生させる．これは非可逆計算を行うと必ず発生してしまうことが分かっている．反対に，可逆計算では計算過程で情報を失わないのでエネルギーの消費や熱を発生の下限が存在しない．非可逆なアルゴリズムを可逆化させ，可逆アルゴリズムにする．すなわち，各実行状態に移る計算が単射であり，直前の状態が高々一意に定まるプログラムで記述すると非可逆なプログラムで本来発生するはずだったエネルギーの消費や熱の発生が抑えられる可能性がある．また，基本的な探索アルゴリズムの一つである線形探索では，提唱されている一般解法を用いた可逆化が行われている．加えて，手動で可逆化することで一般解法よりも時間計算量の効率の良いアルゴリズムも提案されている [2]．また，深さ優先探索や様々な比較ソートでも可逆化が行われている [1][3]．しかし，深さ優先探索の可逆化は手動による効率化が不十分であると考えられる．そのため，効率のいい手動の解法の提案が求められている．加えて，似たケースで使われる幅優先探索の可逆化も求められている．

1.2　目的

本研究では，連結無向グラフの深さ優先探索の既に可逆化された一般解法と比較してより効率的な解法を提案すること，連結無向グラフの幅優先探索の一般解法による可逆化と手動による効率的な解法を提案することをを目標とする．また可逆化したプログラムと効率化したプログラムの解析を行い，トレードオフの関係を明確にする．

1.3　アプローチ

はじめに，C 言語のプログラムを用意し，一般解法を用いて可逆化を行う．そしてその可逆化されたプログラムの解析を行い，ステップ数やメモリ使用量がどの程度かかるのかを把握する．その後可逆化されたプログラムの改良を行いステップ数やメモリ使用量の効率化を図る．最後に改良したプログラムの解析を行い，どの程度効率化できたかを明確化する．

2　関連研究

2.1　探索アルゴリズム

探索アルゴリズムとは，入力されたデータの集合の中から目的となる値や状態を見つけるアルゴリズムである．探索アルゴリズムの分類としてリスト探索，木探索・グラフ探索，文字列探索といったものが挙げられ，本研究で扱う深さ優先探索及び幅優先探索は木探索・グラフ探索の分類に属する．また木探索・グラフ探索の分類には他にも，双方向探索や反復深化深さ優先探索等が挙げられ，その中で深さ優先探索と幅優先探索は基本的なアルゴリズムといえる．

2.2　線形探索

通常の線形探索では探したいキーの値と配列の先頭の値が一致するか確認し，一致すれば終了し，一致しなければ，確認した値が配列の最後であるかを確認し，最後の値でなければ配列の次の値を確認し，キーの値と一致するかもしくは配列の最後の値を探索するまでこの探索をつづける．探索成功の場合 1，失敗の場合 0 を返し成否を判定する．通常の線形探索の他に番兵や，整列リストを用いた線形探索がある．番兵法は配列の最後にキーと同じ値を加え，通常線形探索と同じ方法で探索を行い，キーと同じ値を見つけたときに配列の最後に加えた値かどうかの確認を行い，加えた値ならば失敗とし，加えた値でないならば，成功とする．通常の線形探索と比べて，キーと同じ値を見つけたときにだけ，配列の最後かどうかの確認を行えばよいので，比較回数を減らすことができる．整列リストを用いた線形探索はあらかじめ配列の値が昇順または降順にソートされている場合に用いることができる．昇順の場合で考えるとする．まず番兵法のように配列の最後に∞を加え，先頭から探索を行い，探索している配列の値がキーよりも小さい値ならば次の値を探索し，キー以上の値だった場合キーと一致するかどうかを確認し，一致した場合成功，しなかった場合は失敗とする．

2.3　グラフ

グラフは隣接リストの集合による表現と隣接行列による表現の 2 種類の表現方法がある．本研究では隣接リストによる表現を用いる．隣接リストによる表現は $G = (E, V)$ で表現される．V は頂点の集合 E は頂点間を結ぶエッジ

の集合を表す．グラフには，エッジが有向と無向の 2 種類がある．また，エッジに重みを追加することで，最短経路を求めるアルゴリズムに用いられる．このようにグラフを用いることによって，様々なアルゴリズムを表現することができ，本研究で取り扱う深さ優先探索や幅優先探索にも用いられている．本研究では，無向グラフ表現によるグラフアルゴリズムを取り扱う．

2.4 グラフアルゴリズム

グラフアルゴリズムには様々な探索方法が存在し，探索方法を分類すると，深さ優先探索，幅優先探索，双方向探索，分枝限定法の 4 つの基本的な探索方法に分類される．深さ優先探索を応用したものには，深さ制限探索，反復深化深さ優先探索がある．幅優先探索を応用したものには，最良優先探索，均一コスト探索，$A*$ がある．基本的な探索方法である深さ優先探索と幅優先探索の効率化できれば同様のアプローチもしくは，それに近い方法で応用した探索は効率化できる可能性が高い．そのため深さ優先探索と幅優先探索は，グラフアルゴリズムの中でも基本的なアルゴリズムだといえる．本研究では，グラフアルゴリズムの中の基本的なアルゴリズムの効率化を行うことによって，応用的なグラフアルゴリズムの効率化にも貢献できると考える．

2.5 深さ優先探索

深さ優先探索は縦型探索とも呼ばれ，始点からの距離が大きい未探索の頂点を優先的に探索するアルゴリズムである．手順は，はじめに，与えられた始点と隣接する頂点を探索済みにし隣接する頂点がすべて探索済みになるまで行う．隣接済みの頂点がすべて探索済みの場合，ひとつ前の頂点に戻る．これを繰り返し，すべての頂点が探索済みになるか，探索したい頂点が見つかったときに探索が終了する．始点からの距離が無限にあるグラフでは無限の深さまで探索してしまうため，解が必ず見つけられるとは限らない．また始点からの距離が大きい頂点を優先的に探索する性質があるため，解が複数個ある時最小経路が最初に見つかるとは限らない。

2.6 幅優先探索

深さ優先探索とは反対に横型探索とも呼ばれ，始点から距離が小さい頂点を優先的に探索する．手順は，始点からの距離が小さい頂点を順に探索する．これを繰り返し，すべての頂点が探索済みになるか，探索したい頂点が見つかったときに探索が終了する．深さ優先探索とは反対に始点からの距離が小さい頂点を優先的に探索する性質があるため，解が複数個ある時最小経路が最初に見つかる．

2.7 一般解法

可逆化を行う一般解法には幾つかの方法が存在し，その中で Landauer 法と Bennett 法と呼ばれるものがある [4]．Landauer 法は，代入などの非可逆な計算では情報の消失が起こる．そこで非可逆な計算が行われる前に非可逆な計算によって消失する情報を別の値に保存することで可逆化を実現した．しかしこの方法は元の非可逆な計算の失われる情報を保存するため，実行時間に比例したメモリ使用量が必要となる．Landauer 法の欠点として出力が必要以上に出てしまうということが挙げられる．Bennett 法は Landauer 法の欠点を解消する解法で，プログラムを実行した後に出力に有効な情報を保存し，その後逆実行を行う．逆実行を行うことでスタックを空にし，メモリ使用量を抑えることができる．しかし，逆実行を行うことによって実行時間が約 2 倍になってしまうという欠点がある．

2.8 Janus

命令型可逆プログラミング言語 Janus で書かれたプログラムは可逆性が保証されている．非可逆な言語において条件分岐文や繰り返し文は可逆性を持たない．しかし，Janus ではこれらの文も可逆性が保証されている．本研究では可逆なプログラムを作成する際に，Janus で実装する．

2.9 O 記法

計算量を漸近的に表す記法で O で表される．$f(x) = 3x + 2(x \to \infty)$ で計算量を表せるプログラムがあるとすると，$O(x)$ と表す．O は関数 $f(x)$ を実行するのに，x の定数倍の時間程度しかかからず．関数 f は漸近的に x によって上から抑えられることを表す．反対に Ω 記法と呼ばれる $\Omega(x)$ では，漸近的に x によって下から抑えられることを表す．

3 グラフアルゴリズムと可逆化

3.1 アルゴリズムの解析

実際にアルゴリズムをプログラムで実装するとなると，同じアルゴリズムでも実装の方法によってプログラムの動き方が変わってしまう．そのため，プログラムをある指標を用いて数値に表し，同じ指標を用いて数値に表した別のプログラムと比較することでそれぞれのプログラムの良さを把握することができる．このプログラムの良さを見積もることをアルゴリズムの解析と呼ぶ．本研究ではプログラムの良さを表すための指標として時間計算量，空間計算量，ゴミ出力量の 3 つを用いる．

3.2 時間計算量

時間計算量はアルゴリズムを実行した際にどの程度時間がかかるかを表す．しかし，アルゴリズムの実行時間はプログラムや実装する計算機，入力されたデータの量によって変化する．そのため，一般的にはオーダー記法という考え方を用いてアルゴリズムそのものの効率を漸近的に考える．具体的にはそのアルゴリズムにとって最も重要とされる演算を基本演算と呼び，基本演算が実行される回数を求めるだけで良いとされる [5]．アルゴリズム同士ではオーダーを用いて比較することが望ましいが，同じアルゴリズムを別々のプログラムで実装した場合だと，大抵は同じ結

果が出てきてしまう．そのため，本研究では，可逆プログラムは全ての行を同一の時間で計算するものとし，行毎に実行回数を評価して時間計算量を求める．

3.3 空間計算量

空間計算量はプログラムを実行した際にどの程度記憶領域を必要とするかを表す．ここでは変数を 1 つ新たに用意すると記憶領域を 1 使用するものとする．すなわち変数や配列を宣言するか，スタックに push した場合に空間計算量が増える．

3.4 ゴミ情報

ゴミ情報は可逆アルゴリズム特有の考え方で，出力データの中で問題解決に必要な情報を保持している出力データ以外の本来は必要でない出力データをゴミ情報と呼ぶ．非可逆アルゴリズムでは必要な情報のみを出力すれば良いが，可逆アルゴリズムでは逆実行する際の入力として必要な情報も出力される場合があるためゴミ情報と呼ばれるデータが出てくる．また，実行途中で出てくるゴミ情報を中間ゴミ，実行終了時に出てくるゴミ情報を最終ゴミと呼ぶ．

4 既存の可逆線形探索の解析

グラフ探索アルゴリズムの解析を行う前に，可逆アルゴリズムや，アルゴリズムの解析に慣れる必要がある．したがって，探索アルゴリズムの基本的なアルゴリズムにあたる文献 [?] の可逆線形探索の実装と解析を行い．さらに，解析の方法を工夫し別の視点で見ることでグラフ探索アルゴリズムの可逆化のための糸口を模索した．

4.1 ウォークスルー

はじめに，記述されているプログラムに適当な main 関数を付け加えて Janus に実装することで全てのすべてプログラムが正常に動作することが確認できた．すなわち，文献 [2] に記述されているプログラムは可逆化が行われていることが確認できたということである．次に時間計算量の解析を行った．具体的には，全ての行で同じ時間が掛かるものと定義をし，プログラムを実行した時に何行走査したのかを変数を用いて表すという作業を行い，文献 [2] に書かれている計算量と同じ値になることが確認できた．

4.2 効率化に出る特徴

効率化をするとき，一般的に時間計算量を削減すると空間計算量が増加し，空間計算量を削減すると時間計算量が増加するというトレードオフがある．文献 [2] ではそのトレードオフ性が見られず両方とも削減できるか，一方だけ削減しもう一方は変化しないというような結果が出ている．トレードオフ性が出てこなかった原因として二つ考えられた．一つは，一般解法による可逆化は時間計算量と空間計算量のどちらにおいても非常に効率の悪いものとなっている可能性がある．もう一つは，計算量の求め方が悪い可能性がある．時間計算量で全ての行を走査したが，そのアルゴリズムにおいてあまり重要でない部分も走査してしまっていた．また，キーを見つけても最後まで走査するという条件に制限することで効率化を行ったものがあり，より狭い条件でのみ効率化が行えることもあるということがわかった．

4.3 別の方法での解析

上の節を踏まえて，時間計算量の基本演算を制限しつつ，解析し直してみた．今回の基本演算は，探索している要素が探索したいものと一致するかを比較する演算，探索している要素が最後の要素かを比較する演算と定義し，それぞれ key，index と呼ぶことにした．具体的なプログラム上でどの部分を指すのかは図 1 に示す．

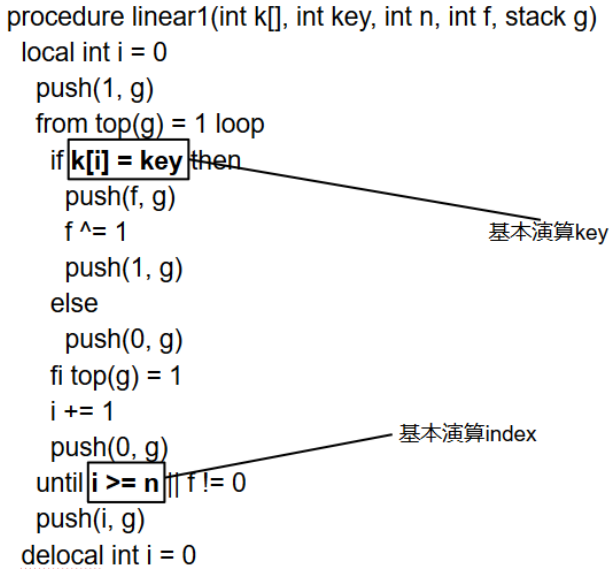

図 1 基本演算の説明

また比較するために，文献にある通常の線形探索 1 の一般解法と提案解法，番兵法の一般解法，のプログラムを用いる．また探索する値が存在する位置を変数 M，探索が成功したら 1 失敗したら 0 を表す変数 S を用意する．以上の条件で解析を行った結果を表 1 に示す．

表 1 基本演算別に見た時間計算量

基本演算	全ての行	key	index
通常の一般解法	$14M+18S+22$	$2M$	$2M-2S+2$
通常の提案解法	$5M+6S+6$	$M+S$	$M+S$
番兵法の一般解法	$8M+4S+32$	$2M-2S+2$	2

まず通常の線形探索の一般解法と提案解法では key と index のどちらの値も大体 M 改善出来たのに対して key と index 以外の部分では $(14M-5M)-M-M=7M$ となり，$7M$ ほど改善できていたことがわかった．この key と index によって改善された合計 $2M$ という値は，線形探索アルゴリズムにおいて重要な演算を行う部分であり，それ以外の部分で改善された $7M$ という値と比較して大きなものとしてみる必要があると考えられる．また，通常の線

形探索の一般解法と番兵法の一般解法では key の値はあまり変化がないのに対して index の値が非常に小さくなっているという特徴が見ることができた．基本演算を幾つか定めて解析を行うことはアルゴリズムの特徴を発見することができるため，有効な手段だと考えられる．

4.4 現状整理

可逆線形探索の追試を行い，アルゴリズムの解析に必要な手順を確認することができた．更に，アルゴリズムの効率化をする際はアルゴリズムの動きや，効率化をする指標を制限することで改善がされやすいことがわかった．アルゴリズムの解析においては基本演算を定めることが重要だと発見できたが，どの演算が基本演算にあたるのか慎重に定める必要がある．文献 [?] では深さ優先探索を一般解法を用いて，可逆化を行い，また，改良も行っている．提案解法 1 と 2 が示されている．より効率化が行われている提案解法 2 と一般解法とを比較して，現状どの程度深さ優先探索が効率化されているかを知る．文献 [?] の結果を以下の表 2，3 に示す．一般解法 a と提案解法 $2a$ は Landauer 法，一般解法 b と提案解法 $2b$ は Bennett 法を用いて可逆化が行われている．

表 2 一般解法 a と提案解法 $2a$ の比較 [1]

比較項目	一般解法 a	提案解法 $2a$
メモリ使用量	$6n+4L-3S+11$	$6n+L-S+6$
ステップ数	$11L-6S+16$	$8L-2S+8$
ゴミ出力	$6n+4L-3S+10$	$6n+L-S+5$

表 3 一般解法 b と提案解法 $2b$ の比較 [1]

比較項目	一般解法 b	提案解法 $2b$
メモリ使用量	$6n+6$	$6n+5$
ステップ数	$22L-12S+40$	$16L-4S+24$
ゴミ出力	$6n+2$	$6n+2$

表中の n は頂点の数，L は探索したい頂点は何番目にあるか，S は探索成功なら 1，失敗なら 0 を返すとこれらを定義する．この定義から $n \geq L$ となるので，効率化を行う上で優先的に減らす必要があるのは，n である．表 2 の一般解法 a は Landauer 法を用いて可逆化が行われ，一般解法 a を改良したものが，提案解法 a である．表 3 の一般解法 b は Bennett 法を用いて可逆化が行われ，一般解法 b を改良したものが，提案解法 b である．これらの結果をどの程度効率化できたかを可視化するために，比較項目を一般解法 − 提案解法したものを下記の表 4 に示す．

表 4 より提案解法 a はどの項目もおよそ $3L$ 程度減らすことができている．しかし n に着目すると効率化はできていない．また提案解法 b はステップ数は $6L$ 程度減らすことができているがメモリ使用量はほとんど変わらず，ゴミ出力においては一般解法と変わらない．これらの結果から

表 4 提案解法 $2a$，$2b$ がどの程度効率化できているか [1]

比較項目	$2a$	$2b$
メモリ使用量	$3L-2S+5$	1
ステップ数	$3L-4S+8$	$6L-8S+16$
ゴミ出力	$3L-2S+5$	0

現状 Landauer 法を用いた解法ではメモリ使用量，ステップ数，ゴミ出力の三点を $3L$ 程度，Bennett 法を用いた解法ではステップ数を $6L$ 程度減らすことができたということである．この結果から本研究では，Landauer 法を用いた解法では n に着目した場合前述した 3 点を n の定数倍もしくは L に着目して 3L 以上減らすことが必要であると考える．Bennett 法を用いた解法では n に着目した場合 n の定数倍，もしくは L に着目して $6L$ 以上減らすことが必要であると考える．

5 おわりに

本研究では，深さ優先探索を n に着目したときにより効率化することを第 1 の目標とし，第 2 に幅優先探索の効率化を目標とする．現状先行研究である可逆線形探索の効率化の手順，解析の方法を学び解析を実際に行った．今後の課題は，同様の方法を用いて深さ優先探索と幅優先探索の可逆化，効率化を行うことである．

参考文献

[1] 浅野早紀，山口春樹：可逆な深さ優先探索，南山大学 2018 年度卒業論文 (2019).

[2] 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).

[3] 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, pp.407–426 (2015).

[4] Frank, M.P.: Reversibility for Efficient Computing, PhD Thesis, MIT (1999).

[5] 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，pp.3–95，岩波書店 (1999).