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

平成 31 年 9 月 20 日

1　はじめに

1.1　背景

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．これは非可逆と呼ばれる性質をもった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的な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的な解法よりも時間計算量などの様々な観点から効率の良いアルゴリズムも提案されている．[2] 同様に，連結無向グラフの深さ優先探索でも一般解法を用いた可逆化が行われている [1] が，手動による効率化は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物が挙げられ，本稿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るが，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 通常の線形探索

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，最後の値出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 を返し成功かどうかを判定する．

2.3 他の線形探索

2．2 の線形探索の他に番兵法，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．

2.4 Landauer 法

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．

2.5 Bennett 法

この方法では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増える欠点がある．

2.6 Janus

命令型プログラミング言語の一つで，Janus で書かれたプログラムは可逆であるといえる．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が，Janus ではこれらの文も可逆性が保証されている．本稿では可逆なプログラムを作成する際に，Janus で実装する．

2.7 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 →∞) 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 は定数倍を除いて，関数を上から抑えることができる．反対に下から抑えることができるΩ記法がある．

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，プログラムを実行した場合の所要時間と使用する記憶領域の量の 2 つの観点で比較することでプログラムの良さを把握することができる．本稿では，プログラムの良さを表す言葉としてプログラムを実行した場合の所要時間を時間計算量，プログラムを実行した場合に使用する記憶領域の量を空間計算量，これら二つの計算量と可逆計算で出てくるゴミのいずれかを減らしたことを表すために効率という言葉を用いる．例として，あるプログラム A の時間計算量が別のプログラム B の時間計算量よりも少ない数値であるならば，プログラム A はプログラム B よりも時間計算量の観点での効率が良いと言える．アルゴリズムの効率の良さを量るためにプログラムの時間計算量等を調べることをアルゴリズムの解析と呼ぶ．

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．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，本稿では，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データと定義すれば良いが，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データとして定義す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 現状把握

参考文献 [1] 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 が示されているが，一般解法と提案解法 2 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．参考文献 [1] の結果を以下の表 1，2 に示す．

表 1: 一般解法 a と提案解法 2a の比較

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 2: 一般解法 b と提案解法 2b の比較

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≥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 1 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 2 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 3 に示す．

表 3: 提案解法 a，b がどの程度効率化できているか

比較項目	a	b
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

表 3 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．これらの結果から現状 Landauer 法を用いた解法では 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 2n から 3n 程度もしくは 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 4n から 6n 程度もしくは L に着目して 6L 以上減らすことが必要である．

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(2015)

[4] Michael P. Frank : Reversibility for Efficient Computing ， pp. 64-70(1999)