可逆アルゴリズムのゴミ出力量の最適化
―深さ優先探索とハフマン符号化を対象として―

M2020SE006 田島嘉人
指導教員：横山哲郎

1　はじめに

全過程が単射なステップで構成される計算は，その出力から入力を一意に決定することができる．この性質は計算の可逆性と呼ばれ，消費エネルギー最適化や量子コンピュータの実現などに用いられる．

非単射な計算を対象とする計算機科学の知見には，計算の可逆性の下でそのまま適用することができないものが存在する．本研究が対象とするアルゴリズムの分野はその代表例である．現在提案されている多くの非可逆アルゴリズムは非単射な計算を含むので，計算の可逆性の下では直接的には実行できない．非単射な計算を含むアルゴリズムを可逆制約下でシミュレーションする一般解法 [1, 2, 3] がいくつか知られているものの，これらの手法はオーバヘッドをもたらす．したがって，少なくとも現在は，効率的な可逆アルゴリズムを得るには，問題固有の知識を用いて勘と経験によって開発する必要がある．実際，一般解法と比較し効率的な可逆アルゴリズムが，問題の性質を利用することで開発されてきた [4].

本研究では，深さ優先探索とハフマン符号化の効率的な可逆化を行う．深さ優先探索について，既に知られているよりも時間及び空間効率の良いアルゴリズムを開発できることを示す．また，ハフマン符号化について，入力文字の頻度表が整列されている場合に，ハフマン木の構築をゴミ出力ゼロで行う可逆アルゴリズムを開発できることを示す．深さ優先探索とハフマン符号化は基本的なアルゴリズムであり，本研究の知見が他のアルゴリズム構成において用いられることが期待される．

2　関連研究

2.1　可逆アルゴリズムの計算量

非可逆アルゴリズムの可逆シミュレーションでは，時間計算量と空間計算量に加えて，ゴミ出力量が性能を解析する指標として用いられる．ゴミ出力とはアルゴリズムの実行後の出力の内，シミュレーションをする対象の非可逆アルゴリズムに存在しない出力のことである．一般的にゴミ出力は，計算を単射化するために出力されるものであり，問題の本質的な解決には関わらない．また，可逆システム中で自由にゴミ出力を消去することはできない．従って，ゴミ出力は可能な限り少ないことが望ましい．

また，入力データ構造及び出力の違いによる漸近的計算量の差が存在しない非可逆アルゴリズムについて，その可逆シミュレーションでは計算量に違いが現れることが分かっている [4].

2.2　可逆シミュレーションの一般解法

可逆シミュレーションの代表的な一般解法に，Landauer 法 [1], Bennett 法 [2], LMT 法 [3] がある．Landauer 法は全ての計算の履歴をゴミ出力とする方法である．Bennett 法は，Landauer 法によって実行された非可逆なアルゴリズムの計算結果を保存した後，計算結果と一緒に出力されたゴミ出力を，計算を逆実行することによって消去する方法である．LMT 法は，計算が必ず終了し決定的である非可逆アルゴリズムについて，その計算過程が木構造になることを利用して可逆シミュレーションを行う手法である．一般解法を用いて，入力 x に対して時間計算量 $T(x)$ 及び空間計算量 $S(x)$ で動作する非可逆アルゴリズムを可逆シミュレーションした場合の計算量を，表 1 に示す．ここで，LMT 法のゴミ出力量は x，T 及び S に依存しないので，記述していない．

表 1　一般解法による可逆シミュレーションの計算量

解法	時間計算量	空間計算量	ゴミ出力量
Landauer[1]	$\mathrm{O}(T)$	$\mathrm{O}(S \cdot T)$	$\mathrm{O}(S \cdot T)$
Bennett[2]	$\mathrm{O}(2 \cdot T)$	$\mathrm{O}(S \cdot T)$	x
LMT[3]	$\mathrm{O}(2^T)$	$\mathrm{O}(S)$	–

3　アプローチ

本研究では，ゴミ出力量最適化の観点から，可逆アルゴリズムを設計する．ゴミ出力が可能な限り少なく，漸近的な時間計算量が可逆化の対象であるアルゴリズムと等しくなるような，可逆アルゴリズムの開発を目指す．

可逆な深さ優先探索は，入力を木構造に制限することによって，節点数 n に対して時間計算量 $\Theta(n)$，空間計算量 $\mathrm{O}(n)$，入力のみをゴミ出力として実現できることが分かっている [4]．本研究では，入力を木構造に制限することに加えて，番兵を配置することで，時間計算量 $\mathrm{O}(n)$，入力と出力を除く空間計算量 $\Theta(1)$，ゴミ出力が入力のみとなる深さ優先探索を構成できることを示す．

ハフマン符号化は，ハフマン木を構築して文字列を符号語列に変換するアルゴリズムである．文献 [5] では，ゴミ出力ゼロかつ非可逆なアルゴリズムと漸近的に同じ時間及び空間計算量でハフマン木を構築する可逆アルゴリズムが示された．しかし，この解法には可逆性制約を満たしていない箇所がある．本研究では，文献 [5] の解法に，ゴミ出力が必要であることを示す．加えて，入力文字の頻度表が整列されている場合は，ゴミ出力ゼロでハフマン木を構築

できることを示す．

アルゴリズムの実装には，プログラミング言語 ROOPL++[6] を用いる．ROOPL++ はメタレベルで可逆なプログラミング言語であり，この言語で記述されたプログラムが可逆性を持つことを保証する．また，オブジェクト指向言語であり，データ構造を柔軟に定義できる．従って，研究の対象とするアルゴリズムで登場する木構造などの複雑なデータ構造を扱いやすい．例えば，後述するハフマン木の節点は，プログラム 1 のように定義される．紙面の都合上，本稿では ROOPL++ を模した擬似コードを使用する．

プログラム 1　ハフマン木の節点

```
1 class Node
2     Node left // 左の子への参照
3     Node right // 右の子への参照
4     int value // 文字
5     int freq // 出現数
```

4　結果

4.1　木構造の深さ優先探索

深さ優先探索は，グラフ中に任意の値を持つ要素が存在するかどうかを，グラフのより深い部分を優先的に辿ることで探索するアルゴリズムである．木構造における深さ優先探索では，与えられた木は一意に定まる順序 $v_1, v_2, \ldots, v_n$ で走査され，任意の走査中の節点 v_i とその直前に走査した節点 v_{i-1} からその更に直前に走査した節点 v_{i-2} と走査中の節点の次に走査する節点 v_{i+1} はそれぞれ一意に定まる．例として，図 1 を考える．この図において，走査は $v_1 \to v_2 \to v_3 \to v_2 \to v_4 \to v_2 \to v_1 \to v_5 \to v_1$ の順で行われる．ここで，走査中の節点が v_2 であり，直前に走査した節点が v_1 の場合，次に走査する節点は v_3 ただ一つに決まる．また，走査中の節点が v_2 であり，直前に走査した節点が v_3 の場合，次に走査する節点は v_4 ただ一つに決まる．つまり，木構造における深さ優先探索では，任意の時点において走査中の節点とその直前に走査してきた節点を探すのに，どの節点を走査してきたかの履歴を用いる必要がない．従って，走査中に保持する情報は，走査中の節点とその直前に走査していた節点のみでよい．さらに，根の親節点に番兵を配置することで，探索終了時点において，結果となる節点の直前に走査した節点の情報を消去することが可能である．以上より，提案アルゴリズムは元の入出力以外の出力を持たず，漸近的な時間計算量は元のアルゴリズムと同じである．ここで，出力が節点への参照である場合のみ衛生であることに注意する必要がある．2.1 節で述べたように，可逆アルゴリズムは出力の種類によって，その計算量が変化する．出力が存在を表すフラグなどの場合，探索終了時に探索終了時点において，結果となる節点の直前に走査した節点の情報を消去できないので，ゴミ出力量は $\Theta(1)$ となる．

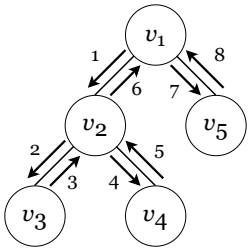

図 1　木構造の深さ優先探索の走査過程

4.2　ハフマン符号化

ハフマン符号化は，あるデータを構成する各文字を表現するための，効果的な可変長符号を設計するアルゴリズムである．いま，表 2 のように，各文字に対して符号長 3 の固定長符号が割り当てられている場合を考える．この固定長符号を用いて，文字列「eaceabbbdbadddabddd」を符号語列で表した場合，長さは $3 \cdot 19 = 57$ となる．ここで，この文字列中の各文字の出現数が異なることに注目し，出現数が多い文字に短い符号語を割り当てる可変長符号を考える．ハフマン符号化は，この文字列から，ハフマン木と呼ばれる図 2 の 2 分木を構築する．ハフマン木において，各文字はある葉に対応している．0 を左の子，1 を右の子に対応させ，表 2 の可変長符号を得る．この可変長符号を用いて先の文字列を表すと，符号語列の長さは 41 となる．これは，固定長符号で表したときより 16 短い．

表 2　文字と出現数及びその固定長符号と可変長符号

	a	b	c	d	e
出現数	4	5	1	7	2
固定長符号	000	001	010	011	100
可変長符号	111	10	1100	0	1101

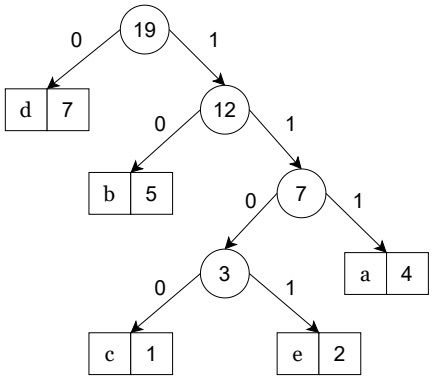

図 2　表 2 から構築したハフマン木

文献 [7] で紹介されている，ハフマン木を構築する代表的なアルゴリズムを，プログラム 2 に示す．プログラム中で，C は入力文字の頻度表を表す集合であり，集合の各要

素 c ∈ C は文字を表す value と出現数を表す freq を持つ．また，Q は出現数をキーとする最小優先度付きキューである．constructMinHeap(C) は集合 C から最小優先度付きキューを構築する操作．extractMin(Q) は Q から最小の値を取り出す操作，insert(Q, z) は Q に値 z を追加する操作である．node はハフマン木の節点を表し，左の子を指す left，右の子を指す right，左の子と右の子の出現数の合計の値を表す freq を持つ．最小優先度付きキューを最小ヒープで実現する場合，constructMinHeap は時間計算量 $O(n \lg n)$ で，insert と extractMin はどちらも時間計算量 $O(\lg n)$ で実現される [7]．ここで，最小ヒープは，常に節点の値が必ずその親の値以下であるという条件を満たす，完全 2 分木である．以上より，プログラム 2 の非可逆アルゴリズムは時間計算量 $O(n \lg n)$，入力と出力を除く空間計算量 $\Theta(n)$ で動作する．

プログラム 2　最小優先度付きキューを用いてハフマン木を構築する非可逆アルゴリズム（文献 [7] より引用）

```
 1 huffmanTree(C)
 2     n = |C| // 要素数
 3     Q = constructMinHeap(C)
 4
 5     for i = 1 to n − 1
 6         allocate a new node z // ハフマン木に追加する節点
 7         z.left = extractMin(Q) // 左の子
 8         z.right = extractMin(Q) // 右の子
 9         z.freq = z.left.freq + z.right.freq
10         insert(Q, z) // 追加
11
12     return extractMin(Q)
```

文献 [5] は，プログラム 2 をベースとする可逆アルゴリズムを，時間計算量 $O(n \lg n)$，入力と出力を除く空間計算量 $\Theta(n)$，ゴミ出力ゼロで実現できると述べた．このアルゴリズムの入出力例を図 3 に示す．文献 [5] の可逆アルゴリズムは，文献 [8] で提案された最小ヒープを用いて，最小優先度付きキューを実現している．文献 [8] の最小ヒープは，時間計算量 $\Theta(\lg n)$，入力と出力を除く空間計算量 $\Theta(\lg n)$，ゴミ出力量 $\Theta(1)$ で値を可逆的に追加できる．従って，これを用いた最小優先度付きキューへの値の追加には，ゴミ出力が必要である．文献 [5] は，プログラム 3 に示す方法で，このゴミ出力のゼロクリアを試みている．しかし，この方法はループの開始条件が不適切であり，可逆ではない．

プログラム 3　ゴミ出力の可逆消去（文献 [5] より）

```
1 method resetCounter(int counter)
2     from 0 < counter loop // 開始条件が不適切である
3         counter −= 1
4     until counter = 0
```

また，文献 [8] では，前述した最小ヒープに値を追加する操作の入力を工夫し逆実行することで，最小ヒープから最小の値を取り出す操作を時間計算量 $\Theta(1)$，入力と出力を除く空間計算量 $\Theta(1)$，ゴミ出力ゼロで可逆的に実現できるとした．しかし，この方法には可逆性制約を満たしていない箇所がある．加えて，値を取り出した後の木が最小ヒープ条件を満たさないことがある．文献 [8] の最小ヒープへの値の追加操作の効率性を維持したまま，最小ヒープから最小の値を取り出すためには，$O(\lg n)$ のゴミ出力量が必要である．また，その場合の時間計算量は $O(\lg n)$，入力と出力を除く空間計算量は $\Theta(\lg n)$ となる．以上より，文献 [5] で提案されたアルゴリズムの計算量は，時間計算量 $\Theta(n \lg n)$，入力と出力を除く空間計算量 $\Theta(n)$，ゴミ出力量 $\Theta(n \lg n)$ である．

入力文字の頻度表が整列されているとき，即ち図 3 中の C が freq でソートされているときを考える．この場合，時間計算量 $\Theta(n)$，入力と出力を除く空間計算量 $O(n)$ でハフマン木を構築する非可逆アルゴリズムが知られている [9]．このアルゴリズムをベースとした可逆アルゴリズムを，プログラム 4 に示す．このアルゴリズムの入出力は図 3 と同様である．ただし，C はソートされているとして扱う．

プログラム 4　C がソートされている場合の可逆解法

```
 1 method huffmanTree(Queue C, Node root)
 2     local int n = C.length // 要素数
 3     local Queue Q = nil // 節点を格納する待ち行列
 4     new Queue Q
 5
 6     local int i = 0
 7     from i = 0 loop
 8         local Node node = nil // ハフマン木に追加する節点
 9         new Node node
10
11         if C.head.value.freq < Q.head.value.freq then
12             call C::dequeue(node.left)
13         else
14             call Q::dequeue(node.left)
15         fi node.left.left = nil && node.left.right = nil
16
17         if C.head.value.freq < Q.head.value.freq then
18             call C::dequeue(node.right)
19         else
20             call Q::dequeue(node.right)
21         fi node.right.left = nil && node.right.right = nil
22
23         node.freq ^= node.left.freq + node.right.freq
24         call Q::enqueue(node) // キューに追加
25         delocal Node node = nil
26         i += 1
27     until i = n − 1
28     delocal int i = n − 1
29
30     call Q::dequeue(root)
31     delete Queue Q
32     delocal Queue Q = nil
33     uncall root::leaf(n) // ゼロクリア
34     delocal int n = 0
35     delete Queue C // ゼロクリア
```

プログラム 4 に示すアルゴリズムは，入力文字列の頻度表 C と初期値の root を入力とする．簡単のため，C を先頭の要素への参照 head と要素数 length を持つ待ち行列によって実現する．C の各要素は Node 型の値 value を持つ．アルゴリズム実行後，C はゼロクリアされ，root は構築したハフマン木の根への参照となる．プログラム中に現れる Q は，ハフマン木の節点を一時的に格納するための待ち行列である．プログラム 7–27 行目のループでは，ハフマン木の新たな節点が作成され，Q に追加される．プログラム 11–15 行目で，C の先頭の要素と Q の先頭の要素のより小さい出現数の要素が待ち行列から取り出され，新たな

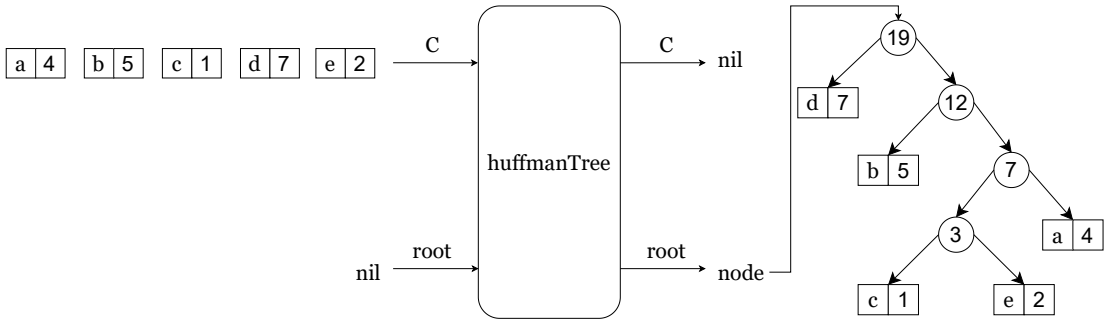

図 3 huffmanTree メソッドの入出力例

節点の左の子となる．同様の処理を，プログラム 17–21 行目において右の子について行う．ここで，節点に追加した子が葉であるかどうかの結合条件により制御流の結合後にどちらの分岐を辿ったかが一意に定まることに注意して欲しい．ループ終了後，Q の要素はただ一つであり，その値はハフマン木の根への参照である．プログラム 33 行目の leaf は，ハフマン木の葉の個数を n の値から引く．leaf は，時間計算量 $\Theta(n)$，入力と出力を除く空間計算量 $\mathrm{O}(n)$，ゴミ出力ゼロで実行される．文献 [8] によって，待ち行列から要素を取り出す dequeue 及び要素を追加する enqueue は，どちらも時間計算量 $\Theta(1)$，入力と出力を除く空間計算量 $\Theta(1)$，ゴミ出力ゼロで実行できることが明らかにされている．以上より，プログラム 4 は，時間計算量 $\Theta(n)$，入力と出力を除く空間計算量 $\mathrm{O}(n)$，ゴミ出力ゼロで動作する．この時間及び空間計算量は，文献 [9] の非可逆アルゴリズムと漸近的に等しい．

5 おわりに

本研究では，基本的な非可逆アルゴリズムをゴミ出力量最適化の観点から可逆化した．第 1 に，入力データ構造を木構造に制限し，番兵を配置することによって，時間計算量 $\mathrm{O}(n)$，入力と出力を除く空間計算量 $\Theta(1)$，ゴミ出力が入力のみとなる可逆深さ優先探索法を提案した．ただし，節点への参照以外を出力とする場合，定数のゴミ出力が追加で必要となる．これは，出力データの種類によって可逆アルゴリズムの漸近的な計算量が変化することの一つの証拠である．第 2 に，ハフマン符号化について，文献 [5] の直接的な可逆化は，非可逆なアルゴリズムと漸近的に同じ時間及び空間計算量ではあるものの，ゴミ出力量 $\Theta(n \lg n)$ が必要であることを確かめた．また，入力文字の頻度表が整列されている場合，時間計算量 $\Theta(n)$，入力と出力を除く空間計算量 $\mathrm{O}(n)$，ゴミ出力ゼロでハフマン木を構築できることを示した．このような同一の計算を行う非可逆アルゴリズムにおいて，直接的に効率的な可逆化の可否があるという知見は，今後の可逆アルゴリズムの設計と解析に役立つことが期待される．

ハフマン符号化のように，最小優先度付きキューのような複雑なデータ構造を扱うアルゴリズムにおいて，データ構造とその操作の性能は重要である．しかし，可逆アルゴリズムにおいてそれらはほとんど議論されていない [8]．データ構造とその操作の効率的な可逆化を進めることは，可逆アルゴリズム研究の今後の課題である．

参考文献

[1] Landauer, R.: Irreversibility and heat generation in the computing process, *IBM journal of research and development*, Vol.5, No.3, pp.183–191 (1961).

[2] Bennett, C.H.: Logical reversibility of computation, *IBM journal of Research and Development*, Vol.17, No.6, pp.525–532 (1973).

[3] Lange, K.J., McKenzie, P. and Tapp, A.: Reversible space equals deterministic space, *Journal of Computer and System Sciences*, Vol.60, No.2, pp.354–367 (2000).

[4] 増田大輝：効率的な可逆線形探索と木構造の可逆深さ優先探索の設計と解析，南山大学 2019 年度修士論文 (2020).

[5] Hay-Schmidt, L.: Investigating the Reversibilization of Irreversible Algorithms, Master's thesis, University of Copenhagen (2021).

[6] Cservenka, M.H.: Design and implementation of dynamic memory management in a reversible object-oriented programming language, Master's thesis, University of Copenhagen (2018).

[7] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.: *Introduction to algorithms*, MIT press (2009).

[8] Cservenka, M.H., Glück, R., Haulund, T. and Mogensen, T.Æ.: Data Structures and Dynamic Memory Management in Reversible Languages, *Reversible Computation* (Kari, J. and Ulidowski, I., Eds.), Springer, pp.269–285 (2018).

[9] Van Leeuwen, J.: On the Construction of Huffman Trees, *ICALP*, pp.382–410 (1976).