前順走査及び中順走査から二分木を構築するアルゴリズムの解析に関する一考察

2020SE056 澤田 一輝
指導教員: 横山 哲郎

1 はじめに

ラベルなし二分木(以下, 二分木)の再構築問題[1]を解くアルゴリズムの改良が半世紀以上に渡って続けられている. 1980 年代には$O(n^2)$, $O(n \log n)$のアルゴリズムや$O(n)$の非線形空間アルゴリズムが知られていた. 再帰的アルゴリズムは任意の入力に対して比較回数が$2n+1$の線形空間アルゴリズムが知られている[3]. 本稿では反復的な命令型プログラムによる線形時間かつ線形空間のアルゴリズムに焦点を当てる. 2000 年以前に提案された線形時間アルゴリズムで最悪比較回数の定数係数がそれぞれ 7, 5, 4, 3 のもの[4,5,6,7]については, 最良比較回数と共に, 二分木のサイズが 1 から 14 までの平均線形係数(ALC: Average Linear Coefficient)が計算されて比較されている[5].

この報告の後に, 最悪比較回数$3n-2$で最良比較回数が$2n-1$のアルゴリズム M[2]と最悪比較回数$2n+\lceil n/2 \rceil$で最良比較回数が$n+2$のアルゴリズム YG が提案された[8]. YG はそれまでに知られているどのアルゴリズムよりも最悪比較回数と最良比較回数が少なく, 入力サイズが 6 未満の場合を除く全ての場合で M より平均比較回数が少ないことが知られている. しかし, ALC は実際の運用環境でのアルゴリズムの性能示唆する重要な指標であるが, これらのアルゴリズムの ALC の報告は行われていない.

本研究では, サイズが 1 から 14 までの二分木を入力としたときの YG と M の比較回数の ALC を計算して比較を行う. また, YG が M より効率が悪くなるような 3 以下のサイズの二分木の特定, 効率が同等であるような 5 以下のサイズの二分木の特定を行う.

XML 文書や LISP の S 式のような木構造とストリームを相互に変換して処理する手法は広く使用されている. 本研究で対象とするアルゴリズムは, ラベルなし二分木から, 属性や要素をもつ XML 文書の構造をもつ木構造に置き換えても適用が可能である.

2 二分木再構築アルゴリズム

i-p 列とは, 中順でラベル付けした二分木のノードを前順で並べ替えた配列である. 二分木再構築問題は i-p 列から対応する二分木を再構築する問題である.

i-p 列から再構築される二分木は, 以下の(1), (2)の性質を持つ.

(1) i-p 列の接頭辞の末尾が bc であり, $b > c$ であるときcはbの左の子となる.
(2) i-p 列の接頭辞が$\ldots b \ldots c$ であり, $b < c$ であり, c より大きいものが無い c より前の最長の列の中で b が最大であるとき c は b の右の子となる.

M は i-p 列の先頭から末尾に達するまで以下を繰返す. ただし初期スタックは任意のラベルより大きい仮想のラベルをもつとする.

(1) i-p 列の接頭辞が$\ldots bc$ で $b > c$ の場合, c を左の子にし, c をプッシュする.
(2) i-p 列の接頭辞が$\ldots bc$ で $b < c$ の場合, ポップして以下の (2a) と (2b) を実行する.
(2a) スタックのトップより c が大きいときポップをすることを繰り返す.
(2b) c を直前のトップのノードの右の子にして, c をプッシュする.

YG は, i-p 列の先頭と末尾に i-p 列中の任意のラベルより大きい仮想のラベルXと$X+1$それぞれ追加する. ただし, i-p 列の末尾に達したかという終了判定は繰り返しのたびに行うのではなく, 右の子どもを継ぎ木したときのみに行う.

3 平均線形係数の比較

サイズ n が 1 から 14 までの全ての二分木を入力として M と YG のプログラム(https://github.com/tetsuo-jp/tree-reconstruction)の比較回数を求めて結果をまとめたものを表 1 に示す. なお, 入力サイズが 14 を超えると二分木については実行時間が膨大になり ALC を求められていない.

表 1 アルゴリズム M とアルゴリズム YG の ALC

n	C_n	M_n	YG_n	ALC_n^M	ALC_n^{YG}	YG_n/M_n
1	1	1	3	1.0000000000	3.0000000000	3.00000000
2	2	7	9	1.7500000000	2.2500000000	1.28571429
3	5	31	32	2.0666666667	2.1333333333	1.00000000
4	14	126	111	2.2500000000	1.9821428571	0.88095238
5	42	498	406	2.3714285714	1.9333333333	0.81526104
6	132	1947	1506	2.4583333333	1.9015151515	0.77349769
7	429	7579	5643	2.5238095238	1.8791208791	0.74455733
8	1430	29458	21307	2.5750000000	1.8625000000	0.72330097
9	4862	114478	80938	2.6161616162	1.8496732026	0.70701794
10	16796	445094	308958	2.6500000000	1.8394736842	0.69414101
11	58786	1731926	1184118	2.6783216783	1.8311688312	0.68370011
12	208012	6745532	4553654	2.7023809524	1.8242753623	0.67506225
13	742900	26298660	17562156	2.7230769231	1.8184615385	0.66779661
14	2674440	102631635	67901060	2.7410714286	1.8134920635	0.66159971

ここで, 各変数は以下の通りである

C_n:n 番目のカタラン数. つまり, n 個のノードを持つ二分木の数
M_n: M によって, n 個のノードを持つすべての二分木を構築するために必要な比較回数
YG_n: YG によって, n 個のノードを持つすべての二分木を構築するために必要な比較回数
ALC_n^M: n個のノードを持つ二分木を構築するときの M の ALC
ALC_n^{YG}: n個のノードを持つ二分木を構築するときの YG の ALC

アルゴリズム X のサイズnの入力に対する ALC は，$X_n/(nC_n)$ である．二分木のサイズが大きくなるにつれて，M は ALC が増加する傾向， YG は ALC が減少する傾向にあった．二分木のサイズが 3 の場合までは M の方が ALC の値が小さいが，サイズ 4 以降は YG の方が ALC の値が小さくなった．

4 入力データセットの特定

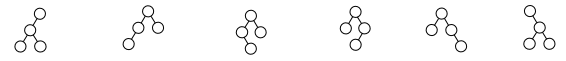
表 2 サイズ 3 までの i-p 列ごとの比較回数

	M			YG			
i-p	labelM	endM	sumM	labelYG	endYG	sumYG	sumM<sumYG
1	0	1	1	2	1	3	True
1,2	2	2	4	4	1	5	True
2,1	1	2	3	3	1	4	True
1,2,3	4	3	7	6	1	7	False
1,3,2	3	3	6	5	1	6	False
2,1,3	4	3	7	6	2	8	True
3,1,2	3	3	6	5	1	6	False
3,2,1	2	3	5	4	1	5	False

M が YG よりも比較回数が小さく，より効率的であるといえる入力データセットを特定するため，サイズ 1 から 3 の二分木の i-p 列ごとにそれぞれのアルゴリズムを使用した場合の比較回数をまとめたものが表 2 である．表 2 中の label，end，sum はそれぞれラベルの比較回数，終了判定の比較回数，それらの合計比較回数を表す．右端の列は M が YG よりも合計比較回数が少ないかを表す．

M が YG よりも比較回数が少なく効率的であった入力データセットは，今回未検証のサイズ 7 以上の場合を除くとサイズ 3 までに存在した入力の i-p 列{1}，{1,2}，{2,1}，{2,1,3}の 4 パターンであった．また， M と YG の比較回数が同じで効率が同等と言えるパターンはサイズ 3 で 4 パターン，サイズ 4 で 6 パターン，サイズ 5 で 6 パターンの合計 12 パターンであった．サイズ 6 以降は，今回検証した最大サイズであるサイズ 14 まで M が YG よりも比較回数が少なく効率的であるパターンも同等であるパターンも存在しなかった． M が効率的であるパターンを YG が効率的でないパターンとして考えた時の二分木はサイズ 4 を例にすると図 1 のようなものとなる．

図 1 サイズ 4 で YG が効率的でない場合の二分木



5 考察

二分木のサイズ 1 から 14 のそれぞれのアルゴリズムを使用した場合の ALC を比較した表 1 からは，サイズ 3 までは M の平均線形係数が YG のものよりも少なく，効率的であるという結果が得られた．つまり，少なくとも今回検証した最大サイズであるサイズ 14 まではサイズ 4 以上の場合， YG が M より平均的に効率的である．

文献[5]によると[4,5,6]のアルゴリズムはサイズが大きくなるほど ALC は大きくなる傾向にあり，サイズ 14 の ALC は YG よりも大きい．したがって，入力サイズが大きいとき，[4,5,6]のアルゴリズムよりも YG の方が平均的に効率が良いといえる．

図 1 の例には，全ノードが子を 1 つもつような場合が現れていない．直前のノードではなく先祖のノードに接ぎ木をすることが多いときに効率が悪くなっている可能性がある．

6 おわりに

本研究では， M と YG の効率性を，比較回数をサイズ 1 から 14 までの二分木について調べることで比較した．この比較から，サイズ 6 以下の二分木ではサイズ 3 までは M が，サイズ 6 より大きい二分木では YG がより平均的に効率的であることが分かった． また， YG が M と同等以上の効率であるようなサイズ 4，5 の二分木の特定とサイズ 4 の時なぜ効率が悪くなるかを考察した．

効率が悪くなるときの原因の特定は今後の課題である．また，サイズ 6 を超える任意の入力サイズに対して YG が M より効率的であることを，サイズ 6 を基底ケースとする数学的帰納法を用いて示すことは今後の課題となっている．

参考文献

[1] Knuth, D. E.: *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley Professional, 1st edition (1968).

[2] Mäkinen, E.: Constructing a binary tree efficiently from its traversals, *International Journal of Computer Mathematics*, Vol. 75, No. 2, pp. 143–147 (2000).

[3] Glück, R. and Yokoyama, T.: Constructing a binary tree from its traversals by reversible recursion and iteration, *Information Processing Letters*, Vol. 147, pp. 32–37 (2019).

[4] Andersson, A. and Carlsson, S.: Construction of a tree from its traversals in optimal time and space, *Information Processing Letters*, Vol. 34, No. 1, pp. 21–25 (1990).

[5] Xiang, L., Lawi, A. and Ushijima, K.: On Constructing a Binary Tree from Its Traversals, Technical report, Faculty of Information Science and Electrical Engineering, Kyushu University, Vol. 5, No. 1, pp. 13–18 (2000).

[6] Mäkinen, E.: Constructing a binary tree from its traversals, *BIT Numerical Mathematics*, Vol. 29, No. 3, pp. 572–575 (1989).

[7] Chen, W. and Udding, J.T.: Program inversion: More than fun! *Science of Computer Programming*, Vol. 15, No. 1, pp. 1–13 (1990).

[8] Yokoyama, Y. and Glück, R : Faster Construction of Binary Trees from Their Traversals, Preprint (2023).