Analyzing Trade-offs in Reversible Linear and Binary Search Algorithms

1. 研究分野

可逆コンピューティング
ー可逆アルゴリズム

2. 目的

効率的な可逆線形探索・可逆二分探索アルゴリズムを提案する．特に，出力の形式及びデータ構造が可逆アルゴリズムの効率に及ぼす影響を考慮する.

3. 背景

可逆アルゴリズムは，可逆システムの構築において，その計算性能を高めるために重要である．近年可逆システムは，並列離散イベントシミュレーションや量子回路の設計に取り入れられており，可逆システムの計算性能を効率化することが求められている.

4. アプローチ

可逆アルゴリズムの表現には，Janusを用いる．Janusは可逆高級言語であり，これにより記述されたアルゴリズムは可逆である．アルゴリズムの表現に高級言語を用いることで，チューリング機械やセルオートマトンなどの計算モデルを使用するよりも，直感的な理解が可能となる.

この研究では，探索問題を次のような問題であるとした.

(a) 与えられたキーを持つ要素のレコード上に存在する数を明らかにする
(b) 与えられたキーを持つ要素のレコード上での位置(複数あれば最初の)を明らかにする
(c) 与えられたキーを持つ要素がレコードに存在するかどうかを明らかにする

また、M 及び G を次のように定めた.

M: 計算中に使用する，入力を除いたメモリの使用量
G: 計算後に使用する，入力を除いたメモリの使用量(ゴミ出力)

線形探索
線形探索とは，指定されたレコードを最初から最後まで逐次探索する探索アルゴリズムを指す．不可逆では，このアルゴリズムはO(n)及びΩ(1)の時間計算量で，空間計算量Θ(n)で動作する(入力を除けばΘ(1))．それ以外では，キーが存在しない場合や，レコード数を返す場合は時間計算量Θ(n)である.

可逆アルゴリズムの最適化は，そのアルゴリズムが用いられる状況によって異なるため，ここでは，以下の3パターンを用意する．ここで，1 は 2 及び 3 の特別な場合であり，時間計算量と空間計算量にトレードオフがある.

1. 走査回数1回　かつ　G=0 で計算する

2. 走査回数1回　かつ　G>=0 で計算する
3. 走査回数1回または2回　かつ　G=0 で計算する.

＊G=0 の場合，ゴミ情報を保存することはできない．また，走査回数が1回の場合，Bennett法(call-copy-uncall)は使用できない.

また，レコードを次のように定義する.

(i) ：変数を持つ配列
(ii) : 双方向リスト
(iii): 単方向リスト

以下が，上記条件を組み合わせたそれぞれのケースにおける，効率の良い可逆線形探索アルゴリズムである．.

・Case
パターン1で，レコードの形式が(i)及び(ii)であり， 探索結果が(a)及び(b)の場合
時間計算量Θ(n)、M=Θ(1)で計算可能
(i)の場合，情報を配列のサイズによって消去できる
(ii)の場合，最後のレコードへのポインタによって消去できる

・Case
パターン1で，レコードの形式が(iii)であり，探索結果が(c)の場合
可逆線形探索を行うことはできない
リストの全探索では，どのセルを全探索したかに関する情報を保存する必要がある

・Case
パターン1で，レコードの形式が(i)及び(ii)であり， 探索結果が(c)の場合
時間計算量Θ(n)，M=Θ(1)で計算可能
検索の成否に関わらず走査し，キーの値が見つかったときにフラグを設定することが可能

・Case
パターン2で，レコードの形式が(i)及び(ii)の場合
検索が成功して終了した場合→場所をゴミ情報として返す(M=G=Θ(1))
探索が失敗した場合→場所の情報なし(M=Θ(1)，G=0)

・Case
パターン2で，レコードの形式が(iii)の場合
走査するレコードの数に比例したゴミ情報が必要 M=G
(b)または(c)の場合，走査は途中で終了できる(G=O(n))
(a)の場合，全てのレコードを走査(G=Θ(n)) *レコードが成形されている場合はG=O(n)
リストが変更可能である場合，入力を消去できる(ポインターを挿し替えて)

・Case
パターン3で，レコードの形式が(i)及び(ii)であり，探索結果が(a)の場合
時間計算量O(n)で計算可能
call-copy-uncallメソッドを使う

・Case

パターン3で，レコードの形式が(i)及び(ii)であり，探索結果が(b)及び(c)の場合

探索が成功する場合，call-copy-uncallメソッドを用いる (時間計算量O(n))
探索が失敗する場合，時間計算量Θ(n)
* 与えられたレコードがソートされていた場合は、どちらもO(n)

・Case

パターン3で，レコードの形式が(iii)の場合

リストの走査では，直前の走査のセルの位置を保存する必要がある (空間使用量Θ(n))
→2回目の走査でこの情報を消去することを試みる

ーリストが変更可能：ポインタを変更し中間リストを作成することで逆計算を可能にする (M=Θ(1))
ーリストが変更不可：レコードがソートされている　(M=Θ(n log n))
　　レコードがソートされていない (M=O(n log n))

二分探索

二分探索は，キーとソート済みのレコードの中央値を比較し，一致しない場合，キーが含まれる可能性のある半分のレコードを探索する探索アルゴリズムである．不可逆では，このアルゴリズムはO(log n)の時間計算量で動作する．非可逆二分探索アルゴリズムでは，ステップの探索範囲から，直前のステップの探索範囲を計算することはできない．この情報をゴミ情報として出力することを考えるとG=O(log n) である．

可逆アルゴリズムへの最適化は，範囲が2の累乗である場合，以前の範囲を一意に決定できるという特性の元行われた．つまり，前回のステップの範囲は現在のステップの範囲の2倍である．また，j回ステップが行われる場合，⌈log n⌉- j=0 となる．この特性を用いて，時間計算量Θ(log n)，M=Θ(1)，G=0の可逆二分探索アルゴリズムが完成した．

* bsrch1において，m=k であっても i=0 になるまでステップを繰り返す必要があるため時間計算量Θ(log n)

```
1  // set the ceiling of logarithm of n to v
2  procedure log2ceil(int n, int v)
3    from v = 0 loop v+=1 until 2**v >= n
4
5  procedure bsrch1(int in[], int u, int k, int len)
6    local int l = 0
7      local int i = len
8        u ^= 2**i
9        from l=0 && u=2**len loop
10         i -= 1
11         local int m = 0
12           m ^= l + (u-l)/2
13           if size(in)<=m || in[m]>k then
14             u ^= (m-l)*2 + l        // zero clear u
15             u <=> m                 // zero clear m
16           else
17             l ^= 2*m - u            // zero clear l
18             l <=> m                 // zero clear m
19           fi (l&(2**i)) = 0
20         delocal int m = 0
21       until i = 0
22     delocal int i = 0
23     u -= 1
24   delocal int l = u
25
26 procedure bsrch(int in[], int k, int u)
27   local int len = 0
28     call log2ceil(size(in),len)    // set len
29     call bsrch1(in,n,u,k,len)
30     if u >= size(in) || in[u]!=k then
31       uncall bsrch1(size(in),n,u,k,len)
32                                    // zero clear u
33       u ^= -1
34     fi u = -1
35     uncall log2ceil(size(in),len) // zero clear len
36   delocal int len = 0
```

5. 結果

線形探索

(1) One traversal and $G = 0$.

M	(a) number	(b) location	(c) flag
(i) array/(ii) dlist	$\Theta(1)$	$\Theta(1)$	NA
(iii) list	NA	NA	NA

(2) One traversal and $G \geq 0$.

M, G	(a) number	(b) location	(c) flag
(i) array/(ii) dlist	$\Theta(1), O(1)$	$\Theta(1), O(1)$	$\Theta(1), O(1)$
(iii) list	$\Theta(n), \Theta(n)$	$O(n), O(n)$	$O(n), O(n)$

(3) One or two traversals and $G = 0$.

M	(a) number	(b) location	(c) flag
(i) array/(ii) dlist	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
(iii) (mutable) list	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
(iii) (immutable) list	$\Theta(n)$	$O(n)$	$\Theta(1)$

二分探索

時間計算量Θ(log n)，M=Θ(1)，G=0

6. 有用性

可逆アルゴリズムにおいては，出力の種類とデータ構造が，アルゴリズムの効率に影響を与えることが確かめられた．この性質は，非可逆アルゴリズムとは異なるものであり，可逆アルゴリズムの設計に，従来のアルゴリズム設計に対する考え方とは異なる原理が必要であることを示している．

7. 限界・短所

可逆アルゴリズムにおける，動的計画法や貪欲アルゴリズムなどのような一般的な戦略を提案することが，この分野における一つの目標である．
また，可逆アルゴリズムの性能を測定する指標として，どういったものが適切なのか，依然として明確ではない．

8. 次に何を読めばいいか？

可逆計算(前方・後方決定性)の利点
—Buhrman, H., Tromp, J., and Vitányi, P. 2001. Time and Space Bounds for Reversible Simulation. In ICALP (LNCS), Orejas, F., Spirakis, P.G., and van Leeuwen, J. (Eds.), Vol. 2076. Springer-Verlag, 1017–1027.
—Morita, K. 2017. Theory of Reversible Computing. Springer-Verlag.