アイテムが 2 種に限定された物理的バケットソートにおけるアルゴリズムの改善

小崎　仁美　(指導教員：長尾篤樹)

1　はじめに

バケットソートはソートしたいアイテムの種類数が k, アイテムの個数が n であるとき、平均的に $O(k+n)$ で動くことが知られている．これは電子的な操作であるため, どの箱にアイテムを入れる場合も単位時間のコストで行える．しかし現実でこのソートを行うには箱間の移動にもコストがかかる．一つ隣の箱に移動するたびに単位時間のコストがかかるとしたとき, このコストも合わせたバケットソートが, 物理的バケットソートとして紹介されている.[1] このソートで処理するアイテムは, キューの形で与えられ, 先頭アイテムから処理しなければならなく, ソート作業とは別に先頭アイテムを末尾に移動させることもでき, これも単位時間で行える．

物理的バケットソートには二つのモデルがあり, 上記の 3 つの操作のみ行うモデルがポストモデルと定義されている．ポストモデルで行える操作に加え, 目の前の箱に置かれているアイテムを全て取り出しキューにエンキューする操作を認めたモデルがトレイモデルと定義されている．どちらのモデルも計算量は全てのコストの合計で表される．ポストモデルは $O(n\sqrt{k})$ 時間, トレイモデルは $O(n\log k)$ 時間で処理する効率的なアルゴリズムが与えられている [1]．これらのアルゴリズムは入力のアイテムが先頭以外わからない状態で処理することを前提としている．これにより, ある入力に対して最適なソート手順を示すものではない．

2　既存アルゴリズム

先行研究ではキュー全体の情報を得て, さらに種類数 k を 2 に限定したポストモデルの最適アルゴリズムが紹介されている [2]．以下に物理的バケットソートの具体的な定義と操作を紹介する．

2.1　定義

Definition 1. *[1]* 物理的バケットソートとは以下の手続きである．種類数を k とし, 入力としてのアイテムの入った長さ n のキュー $I=\{x_1,...,x_n\},(1\leq x_i\leq k,1\leq i\leq n)$ とソート済みのアイテムを入れる箱を示すスタック集合 $S=\{s_1,...,s_k\},s_j=\emptyset,(1\leq j\leq k)$, 現在どの箱の前に立っているかを示すポインタ $p=l,(1\leq l\leq k)$ が与えられる．これに対し, キュー I が空になり, 各箱を示すスタック s_j に $x_i=j$ となるアイテム全てが含まれる状況にするための一連の操作を求める．

Definition 2. *[1]* ポストモデルでの物理的バケットソートにおける操作は以下の通りである．

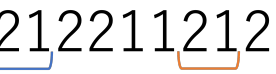
- **操作 place*(p)***:I に対してデキューを行い, 対し取り出したアイテムをポインタ p が指すスタック s_p にプッシュする．
- **操作 pass*()***:I に対してデキューを行い, 取り出したアイテムを I にエンキューする．
- **操作 move*(δ)***: ポインタ p を $p+\delta$ へと更新する．ここで $\delta\in\{-1,1\}$ とし, 更新後は $(1\leq p\leq k)$ でなくてはならない．

これらの操作の総数が物理的バケットソートを行った際の計算量として定義されている．

種類数 2 に限定した場合の物理的バケットソートには最適アルゴリズムが提唱されており, 以下で定義される独立列という概念を用いている．

Definition 3. *[2]* キューの先頭アイテム x_1 が p の値と異なり x_2 とも異なるとき,x_1 は独立であるという．またキューの i 番目のアイテム $x_i,(1<i<n)$ が x_{i-1} と異なり x_{i+1} とも異なるとき,x_i は独立であるという．ただし,x_i がキューの末尾であるなら、x_i は独立ではない．

Definition 4. *[2]* 独立なアイテムのみから成るアイテム列の部分列を独立列と定義する．また独立列の長さの偶奇によって**偶数列**または**奇数列**のどちらかに属す．つまり, 偶数列は正規表現で $(ab)^+$ とマッチし, 奇数列は正規表現で $a(ba)^*$ とマッチする．



青：奇数長の独立列(奇数列)
オレンジ：偶数長の独立列(偶数列)
＊注意：最後の入力は独立にはならない

図 1　独立列の例

2.2　既存アルゴリズムの問題点

付録 A.1 に既存アルゴリズムを記載する．このアルゴリズムには, 分岐の条件が煩雑であり分岐数も多いということが問題点として挙げられる．

3　改善案とその正当性

本研究では, より可読性の高いアルゴリズムを構成するため、既存アルゴリズムの各手続きにおける最適な挙動の解析を行った．それにより得られたアルゴリズムを付録 A.2 に挙げ, 本章ではその正当性を確認する．

3.1　提案アルゴリズムの正当性

Theorem 5. アイテム列の先頭に奇数長の独立列がある場合,i 番目の奇数列における先頭を $x_i(1\leq i)$ とする．ここで $x_1\neq$ 末尾かつ x_2 が存在し,$x_1\neq x_2$ のとき pass 操作,move 操作いずれも最適である．

Proof. i 番目の奇数列における先頭を $x_i(1\leq i)$ とする．$x_1\neq x_2$ の時,$x_1x_2...x_n=(st)^mW$ となる最大の m を求める．ここで $s,t=\{1,2\},s\neq t,W=\{1,2\}^*$ である．W の先頭である W_{head} によって場合分けを考える．W の長さが 0 であるか $W_{\mathrm{head}}=t$ なら move が最適であることが先行研究により証明されている.[2] よって $W_{\mathrm{head}}=s$ のときに move を行ってもコストに差がないことを示せば良い．この条件に合うアイテム列を正規表現で表すと $(a(ba)^*Xb(ab)^*X)^+Yb$ となる．ここで X は奇数列を含まないアイテム列とする．このアイテム列のうち,$(a(ba)^*Xb(ab)^*X)^+$ は各繰り返しに対し同一の操作を行うことが最適であることが知られている.[2] 残りの部分である Y は

二種類考えられ, **(1)**$a(ba)^*X$, **(2)**$a(ba)^*Xa(ba)^*Z$ である. ここで,Z は任意のアイテム列とする. よって $p = b, a(ba)^*Xb(ab)^*XYb$ に対し pass から一連の操作を行う場合と move から一連の操作を行う場合との pass と move の回数の総和を比較し, それらに相違が無いことを確認する.

まず **(1),(2)** で共通する接頭辞までを考える. すなわち、先頭の奇数列までを処理をすることを考える. 初手 pass から一連の操作を行うと $p = b, I = Xb(ab)^*XYba^*$ となる ...(∗). ここで Y の末尾に b から始まる偶数列を持つ場合, 末尾の b が独立になることで, それが奇数列となる.(図 1 参照) このケースは別に取り扱い, まずは Y の末尾に新たに奇数列が発生しない状況を考える. 初手 move から一連の操作を行うと $p = a, I = Xb(ab)^*XYbb^*$ となる ...(†). よって, 初手に pass を行った方が初手に move を行った場合より pass 回数が 1 回多い. 一方で初手に move を行った場合のみ move を 1 度実行するため, ここまでの操作ではコスト差が無いと言える.

以上の状況の後, それぞれの先頭の X を処理するときのコストを考える. 独立列の直後に X が存在するため、X の先頭は b である. 処理 (∗) から引き続き同様の処理を行うと $p = a, b(ab)^*XYba^*$, (†) に続く操作は move から始まり、処理後は $p = a, b(ab)^*XYbb^*$ となる. よって (†) の方がコストは 1 多くなる.

続いて Y の内容について場合分けをする. $Y = a(ba)^*X$ の時,(∗) の処理後は $p = a, b(ab)^*Xa(ba)^*Xba^*$, (†) の処理後は $p = a, b(ab)^*Xa(ba)^*Xbb^*$ となる. ここで先頭の奇数列に対し (∗) の処理後に対しては (∗) が最適であるという仮定の下同様の手順を行い,(†) の処理後に対しては (†) が最適であるという仮定の下同様の手順を行う. その結果, 独立列を一つ処理し終わった状況は, 先頭に move 操作を行う (∗) では $p = b, Xa(ba)^*Xba^*$ となり, 先頭に pass 操作を行う (†) では $p = a, Xa(ba)^*Xbb^*$ となる. (†) の方が (∗) より pass を 1 回多く行い,(∗) のみ move しているのでコスト差は無いと言える.

同様に各状況における先頭の X の処理を考える. ここで X は a 始まりであることに注意すると, (∗) に続く操作は move から始まり,$p = b, a(ba)^*Xba^*$, (†) は $p = b, a(ba)^*Xbb^*$ となる. よって (∗) の方がコストが 1 多くなる.

さらに各状況における先頭の奇数列と X の処理を考えると, どちらも pass から始まり, (∗) は $p = a, ba^*$,(†) は $p = a, bb^*a^*$ となり, どちらも同じ動きをしているのでコスト差は無いと言える. また, ここで (∗)(†) いずれもキューが同じ形になったのでこれ以降でコスト差が生じることは無い.

ここまでのコスト差を総計すると,(∗) と (†) のコストには差が無いことが言える.

(∗) のはじめの処理で Y の末尾に奇数列が発生した場合を考えると,(∗) の処理後は $p = a, b(ab)^*Xa(ba)^*X'b(ab)^+a^*$ となる. これと (†) の比較を先程と同様に行うと, X の処理がどちらも同じ動きとなりコスト差無し, X' の処理では.X' が b から始まるため (∗) の方がコストが 1 多くなることが確認できる. 残り全てを処理する場面はコスト差が同様であると言え, やはり全体でもコスト差がないことがわかる.

以上より,$Y = a(ba)^*X$ の時はいずれの操作から処理を始めてもコストは同じであることが示せた.

$Y = a(ba)^*Xa(ba)^*Z$ の時,(∗) の共通する接頭辞の処理後は $p = a, b(ab)^*Xa(ba)^*Xa(ba)^*Zba^*$, (†) は $p = a, b(ab)^*Xa(ba)^*Xa(ba)^*Zbb^*$ となっている. 以上の状況で, 先頭の奇数列に対しこれまでの手順が最適であるという仮定の下同様の手順を行いを行う. 独立列を一つ処理し終わった状況は, 先頭に move を行う (∗) では $p = b, Xa(ba)^*Xa(ba)^*Zba^*$ であり, 先頭に pass を行う (†) では,$p = a, Xa(ba)^*Xa(ba)^*Zbb^*$ である. (†) の方が (∗) より pass を 1 回多く行い,(∗) のみ move しているのでコスト差は無いと言える.

同様に各状況における先頭の X の処理を考える. ここで X は a 始まりであることに注意すると,(∗) に続く操作は move から始まり,$p = b, a(ba)^*Xa(ba)^*Zba^*$, (†) は $p = b, a(ba)^*Xa(ba)^*Zbb^*$ となり, よって (∗) の方がコストが 1 多いと言える.

さらに各状況における先頭の奇数列と X の処理を考えると, どちらも pass から始まり, (∗) は $p = b, a(ba)^*Zba^*$,(†) は $p = b, a(ba)^*Zbb^*a^*$ となり, どちらも同じ操作をしているのでコスト差は無いと言える. また, ここで (∗)(†) いずれもキューが同じ形になったのでこれ以降でコスト差が生じることは無い. また,**(2)** の場合でも末尾に奇数列が生じることはあるが,**(1)** と同様の議論でコスト差がないと言える. 以上より,$Y = a(ba)^*Xa(ba)^*Z$ の時はいずれの操作から処理を始めてもコストは同じであると示せた.

(1)(2) どちらもコスト差がないことが示せたため, アイテム列の先頭に奇数長の独立列がある場合,i 番目の奇数列における先頭を $x_i(1 \leq i)$ とする. ここで $x_1 \neq$ 末尾かつ x_2 が存在し,$x_1 \neq x_2$ のとき pass 操作,move 操作いずれも最適である. □

4 まとめと今後の課題

アイテムが 2 種類に限定された物理的バケットソートの最適アルゴリズムを修正した. 今後はアイテム数を 3 に増やした場合が考えられる. 一方で直線上にスタックを置くと,move のコストが 1 の場合と 2 の場合が存在し容易にアルゴリズムを構築できない. このため, まずは各スタックを三角形状に置くことで move() のコストを 1 に固定できる事実を用い, この状況下での最適アルゴリズムの構築が考えられる. もしくは, これらの最適アルゴリズムを求めることが NP 困難であるかどうかの解析も考えられる.

参考文献

[1] ヤコノジョン, 伊藤大雄, 長尾篤樹, 西野順二, ラパポートデイビッド. 物理的バケットソート. 電子情報通信学会技術研究報告; 信学技報, Vol. 116, No. 17, pp. 29–34, 2016.

[2] 吉澤修平, 伊藤大雄. アイテムが 2 種に限定された物理的バケットソートの 最適アルゴリズム. 電気通信大学 平成 28 年度卒業論文, 2017.

付録 A　アルゴリズム記述上の定義について

本章ではそれぞれのアルゴリズム内で利用される文字の定義を行う. 現在のアイテム列の先頭アイテムがどの種類であるかを top(I),2 番目のアイテム, 末尾のアイテムに対しても同様に second(I), end(I) とする. ただし、2 番目のアイテムが存在しないときは second(I) = NULL とする.i 番目の奇数列における先頭アイテムをそれぞれ $y_i(1 \leq i)$ とし, 該当するものがない場合 y_i = NULL とする. 先頭に偶数列があることを even(I), 先頭に奇数列があることを odd(I) とする.

既存アルゴリズムの 33 行目の処理に関して NULL ではない y_i を全て並べたものを考え,$y_1y_2...y_l = (ab)^mW$ となる最大の m を求める. ここで $a, b = \{1, 2\}, a \neq b, W = \{1, 2\}^*$ である.

A.1 既存アルゴリズム

Algorithm 1 既存アルゴリズム [2]

Input: 初期ポインタ $p \in \{1,2\}$, キュー $I : x_1, x_2, ..., x_n$, ただし, $x_i \in \{1,2\}$, スタック $s_1, s_2, \forall s_j = \emptyset$
Output: 正しくバケットソートを行う. つまり,$I = \emptyset, \forall i : x_i = j \rightarrow x_i \in s_j$

```
1: while |I| > 0 do
2:   if top(I) = p then
3:     place(p)
4:   else if |I| = 1 ∨ top(I) = second(I) then
5:     move(p − x_1)
6:   else if even(I) then
7:     if y_1 = NULL then
8:       if top(I) = end(I) then
9:         pass()
10:      else
11:        move(p − top(I))
12:      end if
13:    else
14:      y_1 より前に存在するアイテム列を一旦無視し,y_1 以降のアイテム列に対し p = 2 − y_1 として,29 行目以降
         を実行し y_1 に対する最適手順を確認する.
15:      if 上記計算結果が pass 操作となる then
16:        if top(I) = y_1 then
17:          pass()
18:        else
19:          move(p − top(I))
20:        end if
21:      else
22:        if top(I) = y_1 then
23:          move(p − top(I))
24:        else
25:          pass()
26:        end if
27:      end if
28:    end if
29:  else if odd(I) then
30:    if top(I) = end(I) ∨ y_2 = NULL ∨ y_1 = y_2 then
31:      pass()
32:    else if y_1 ≠ y_2 then
33:      if W が a もしくは aa{a, b}* にマッチ then
34:        pass()
35:      else if W が ∅ もしくは b{a, b}* にマッチ then
36:        move(p − top(I))
37:      end if
38:    end if
39:  end if
40: end while
41: return
```

A.2　修正後のアルゴリズム

Algorithm 2 修正後のアルゴリズム

Input: 初期ポインタ $p \in \{1,2\}$, キュー $I : x_1, x_2, ..., x_n$, ただし, $x_i \in \{1,2\}$, スタック $s_1, s_2, \forall s_j = \emptyset$

Output: 正しくバケットソートを行う. つまり,$I = \emptyset, \forall i : x_i = j \rightarrow x_i \in s_j$

while $|I| > 0$ **do**
  if $\mathrm{top}(I) = p$ **then**
    $\mathrm{place}(p)$
  else if $|I| = 1 \vee \mathrm{top}(I) = \mathrm{second}(I)$ **then**
    $\mathrm{move}(p - \mathrm{top}(I))$
  else if $\mathrm{odd}(I)$ **then**
    if $y_1 \neq \mathrm{end}(I) \wedge y_2 \neq \mathrm{NULL} \wedge y_1 \neq y_2$ **then**
      $\mathrm{move}(p - \mathrm{top}(I))$
    else
      $\mathrm{pass}()$
    end if
  end if
  if $y_1 = \mathrm{NULL} \wedge \mathrm{top}(I) = \mathrm{end}(I)$ **then**
    $\mathrm{pass}()$
  else if $y_1 \neq \mathrm{NULL} \wedge y_1 = \mathrm{top}(I) \wedge (y_1 = \mathrm{end}(I) \vee y_2 = \mathrm{NULL} \vee y_1 = y_2)$ **then**
    $\mathrm{pass}()$
  else if $y_1 \neq \mathrm{NULL} \wedge y_1 \neq \mathrm{top}(I) \wedge y_1 \neq \mathrm{end}(I) \wedge y_2 \neq \mathrm{NULL} \wedge y_1 \neq y_2$ **then**
    $\mathrm{pass}()$
  else
    $\mathrm{move}(p - \mathrm{top}(I))$
  end if
end while
return

図 2　オレンジの文字列が修正部分を表す