アルゴリズム B 第 8 回　　2019 年 12 月 2 日

8　比較によらない整列・中間まとめ

整列の計算量の下界

前回までに取り扱った整列アルゴリズムの計算量は次の表のようになる[*1]．ただし，n 個の相異なる要素からなる列を整列するものとして，平均計算量の評価ではすべての順列が一様に現れるものとする．

	最悪計算量	平均計算量
バブルソート	$O(n^2)$	$O(n^2)$
挿入ソート	$O(n^2)$	$O(n^2)$
選択ソート	$O(n^2)$	$O(n^2)$
ヒープソート	$O(n \log n)$	$O(n \log n)$
マージソート	$O(n \log n)$	$O(n \log n)$
クイックソート	$O(n^2)$	$O(n \log n)$

これらの整列アルゴリズムはすべて 2 個の要素の比較に基づいている．このような比較に基づく整列アルゴリズムでは，最悪計算量の下界が $\Omega(n \log n)$ であることが**決定木** (decision tree) を用いて証明できる[*2]．すなわち，比較による整列アルゴリズムでは，最悪計算量を $O(n \log n)$ より小さくすることはできない．しかし，比較以外の方法を用いれば最悪計算量をより小さくできる場合もある．以下では，キー（整列に用いる項目）の値が取り得る範囲が小さい場合に有効な整列アルゴリズムを説明する．

バケットソート

バケットソート (bucket sort) では，まずキーが取り得る値の種類の数だけキューを用意する．整列する列の各要素を対応するキューに入れ，順に取り出すことで整列が完了する．

配列 a の各要素が 1 以上 m 以下の整数であるとき，バケットソートのアルゴリズムは次のように書ける．ただし，Enqueue(q, x) はキュー q の末尾に x を挿入する関数を表し，Dequeue(q) はキュー q の先頭の要素を返し，それを q から削除する関数を表す．

```
1: BucketSort(a[ ], n)
2:     q[1], ..., q[m] を空のキューとする
3:     for i = 1 to n
4:         Enqueue(q[a[i]], a[i])
5:     i = 1
6:     for j = 1 to m
7:         while q[j] が空でない
8:             a[i] = Dequeue(q[j])
9:             i = i + 1
```

[*1] より精密に，すべての $O(n^2)$, $O(n \log n)$ をそれぞれ $\Theta(n^2)$, $\Theta(n \log n)$ で置き換えられることが知られている．

[*2] 平均計算量の下界も $\Omega(n \log n)$ であることが知られている．

バケットソートの最悪計算量は，キーの値が m 種類のとき，$O(m+n)$ である．

整列アルゴリズムが**安定** (stable) であるとは，キーが等しい要素について整列前の順序を保つことをいう．キューを用いることにより，バケットソートは安定である．

計数ソート

計数ソート (counting sort) は，各要素の出現回数を数えることで整列するアルゴリズムである．

配列 a の各要素が 1 以上 m 以下の整数であるとする．要素 i が a に現れる回数を $c[i]$ とする．このとき，要素 i は a を整列した配列の $\sum_{j=1}^{i-1} c[j]+1$ 番目から $\sum_{j=1}^{i} c[j]$ 番目に現れる．このことから，計数ソートのアルゴリズムは次のように書ける．ただし，他のアルゴリズムに合わせて整列した結果を配列 a に格納する．

```
1:  CountingSort(a[ ], n)
2:      b を長さ n の配列，c を長さ m の配列として，c のすべての要素を 0 で初期化する
3:      for j = 1 to n
4:          c[a[j]] = c[a[j]] + 1
5:      for i = 2 to m
6:          c[i] = c[i] + c[i − 1]
7:      for j = n downto 1
8:          b[c[a[j]]] = a[j]
9:          c[a[j]] = c[a[j]] − 1
10:     配列 a に配列 b をコピーする
```

計数ソートの最悪計算量は，キーの値が m 種類のとき，$O(m+n)$ である．

第 7 行から第 9 行までのループにおいて，配列 a の末尾から順に要素を調べ，等しい要素が複数ある場合は後ろから配列 b に格納するようにしている．これにより計数ソートは安定である．

基数ソート

基数ソート (radix sort) は，キーが整数や文字列などの場合に，バケットソートや計数ソートなどの高速で安定な整列アルゴリズムを各桁（各文字）に繰り返し適用することで整列するアルゴリズムである．

k 桁の正の整数を整列するとき，基数ソートのアルゴリズムは次のように書ける．

```
1:  RadixSort(a[ ], n)
2:      for i = 1 to k
3:          下から i 桁目をキーとして配列 a を安定な整列アルゴリズムで整列する
```

整数が m 進法で表記されている，すなわち，各桁の数字が $0, 1, \ldots, m-1$ のいずれかであるとする．安定な整列アルゴリズムとしてバケットソートまたは計数ソートを用いるとする[*3]．このとき，基数ソートの最悪計算量は $O(k(m+n))$ である．

[*3] ここでは BucketSort, CountingSort においてキーを 1 以上 m 以下の整数としたので，基数ソートに使う際は調整が必要である．