




分割統治法

計算機アルゴリズム特論：2017年度

只木進一



分割統治法(Divide and Conquer)と再帰

- 再帰で、よりサイズの小さい問題に分割
- 分割された、非常に小さい問題を解決する。
- その後、小さな問題の組み合わせを繰り返し、全体の問題を解く

分割統治法としてのMerge-Sort

- 長さ n のリストを、すでに整列済みの長さ $n/2$ のリストからの整列に、問題を分割
- 長さ $n/2$ の整列済みのリストからの整列の計算量：最悪で n
- 分割の回数 $\log_2 n$
 - 分割毎に n 個の要素の操作
- 計算量： $n \log_2 n$

待ち行列を使ったMerge Sort の基本的考え方

- 再帰的方法
 - 考え方は単純
 - エラーが発生した場合に、突き止めるのが困難
- 再帰過程を分析して、再帰を使わない方法を探る

再帰の基本的構造

- 再帰木の葉の部分まで処理を待機させる
- 再帰木を根に戻る過程で評価を実施
- 分岐している場合
 - 一つの枝の処理中、他の枝の処理は待機
- スタックや待ち行列で実装できる

Merge Sort

リストの分離

3	8	5	2	7	6	1	4
---	---	---	---	---	---	---	---

3	8	5	2
---	---	---	---

7	6	1	4
---	---	---	---

3	8
5	2

7	6
---	---

1	4
---	---

3	8	5	2
---	---	---	---

7	6	1	4
---	---	---	---

Merge Sort

リストの結合

3	8	5	2
---	---	---	---

7	6	1	4
---	---	---	---

3	8	2	5
---	---	---	---

6	7	1	4
---	---	---	---

2	3	5	8
---	---	---	---

1	4	6	7
---	---	---	---

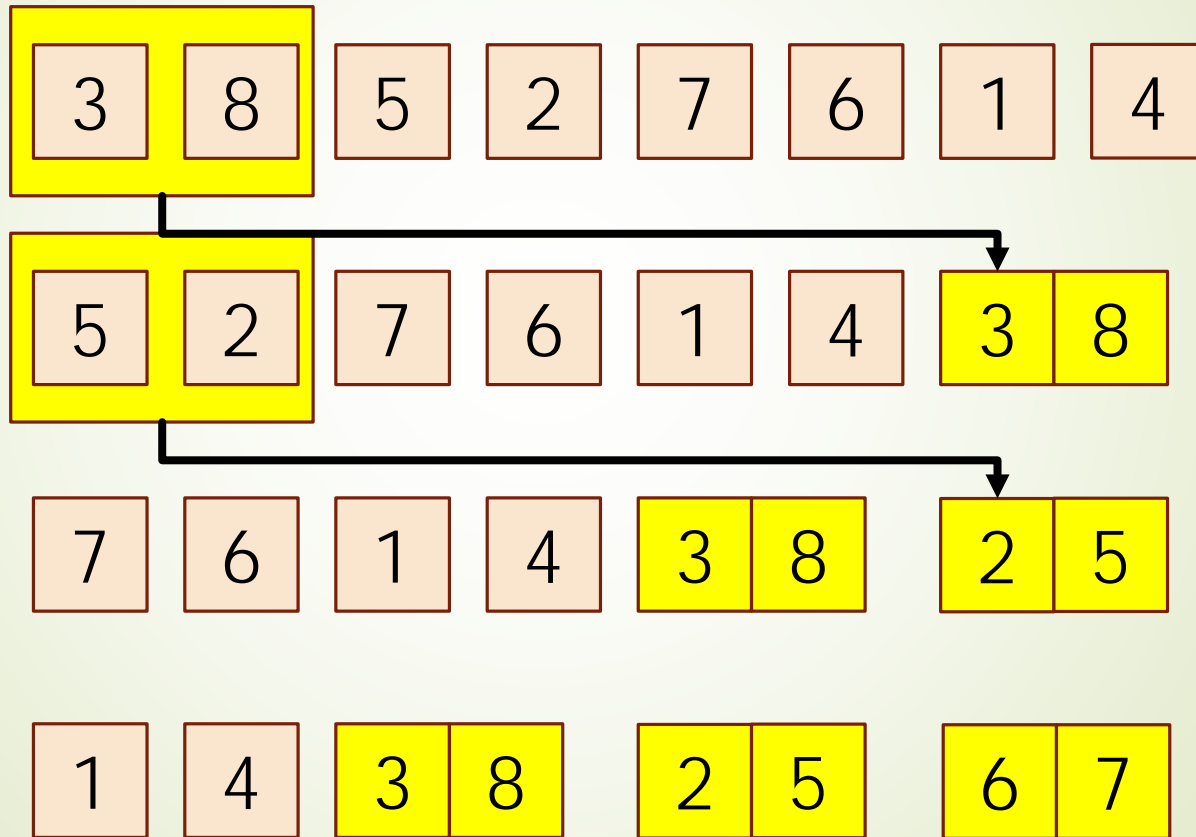
1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

- 入力リストの順序は、結果に影響しない
 - 長さ1のリストに分割して、二つ毎に整列統合すればよい
 - 二つ毎の整列統合が終わったら、さらにそれらを整列統合する

- 整列統合が終わったリストを、待ち行列で待たせて、後で処理
- 先頭から取り出した二つのリストは、必ず整列されていることが重要
- 長さが異なっても問題にはならない

Merge Sort

リストの結合



Merge Sort リストの結合

1. $\langle [3], [8], [5], [2], [7], [6], [1], [4] \rangle$
2. $\langle [5], [2], [7], [6], [1], [4], [3, 8] \rangle$
3. $\langle [7], [6], [1], [4], [3, 8], [2, 5] \rangle$
4. $\langle [1], [4], [3, 8], [2, 5], [6, 7] \rangle$
5. $\langle [3, 8], [2, 5], [6, 7], [1, 4] \rangle$
6. $\langle [6, 7], [1, 4], [2, 3, 5, 8] \rangle$
7. $\langle [2, 3, 5, 8], [1, 4, 6, 7] \rangle$
8. $\langle [1, 2, 3, 4, 6, 7, 8] \rangle$

リストの結合 長さが 2^n でない場合

1. $\langle [3], [8], [5], [2], [7], [6], [1] \rangle$
2. $\langle [5], [2], [7], [6], [1], [3, 8] \rangle$
3. $\langle [7], [6], [1], [3, 8], [2, 5] \rangle$
4. $\langle [1], [3, 8], [2, 5], [6, 7] \rangle$
5. $\langle [2, 5], [6, 7], [1, 3, 8] \rangle$
6. $\langle [1, 3, 8], [2, 5, 6, 7] \rangle$
7. $\langle [1, 2, 3, 5, 6, 7, 8] \rangle$

Merge Sortアルゴリズム 待ち行列での記述

```
protected void sortSub(int begin, int end) {  
    Queue<List<T>> queue = new ConcurrentLinkedQueue<>();  
    //全ての要素をリストにして待ち行列へ  
    for (int i = begin; i < end; i++) {  
        List<T> tt = Utils.createList();  
        tt.add(list.get(i));  
        queue.add(tt);  
    }  
    //二つ毎にマージ  
    while (queue.size() > 1) {  
        mergeListWithQueue(queue);  
    }  
    List<T> listOut = queue.poll();  
    for (int i = 0; i < list.size(); i++) {  
        list.set(begin + i, listOut.get(i));  
    }  
}
```

Merge Sortアルゴリズム 待ち行列での記述

```
protected void mergeListWithQueue(Queue<List<T>> queue) {  
    List<T> t1 = queue.poll();  
    List<T> t2 = queue.poll();  
    List<T> tOut = mergeList(t1, t2);  
    queue.add(tOut);  
}
```