

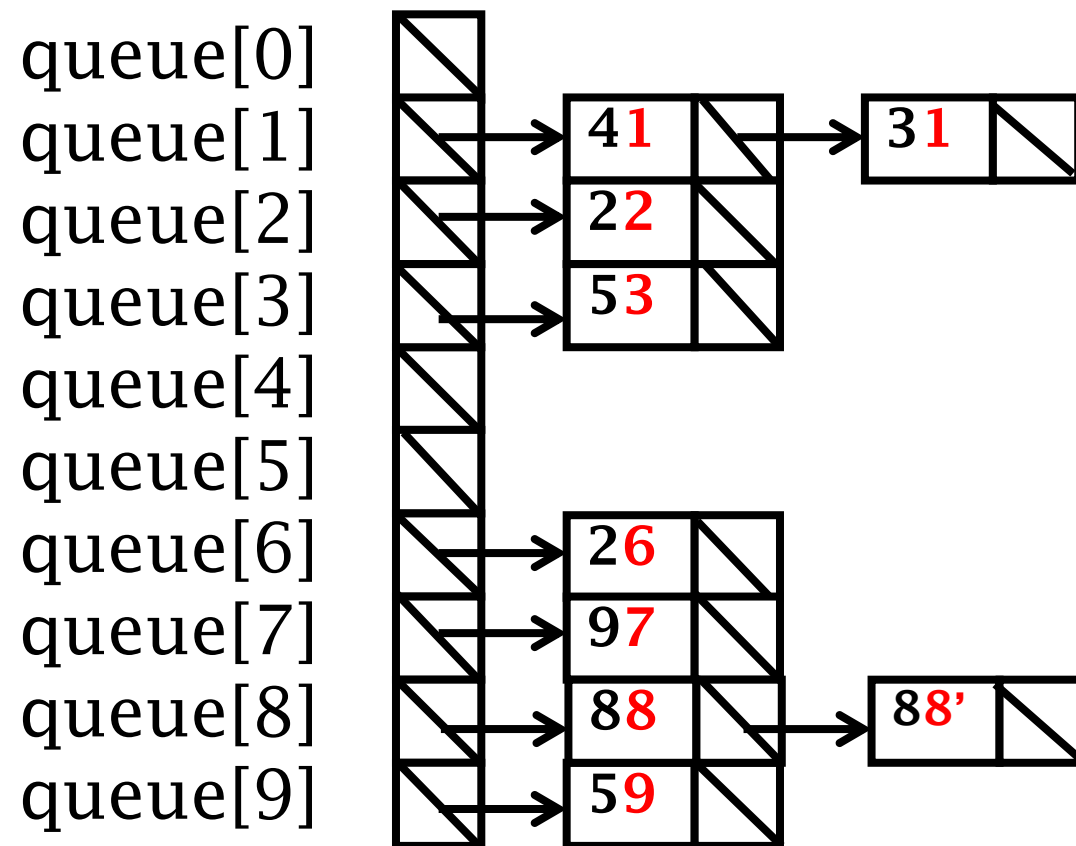


基于静态链的基数排序

- 将分配出来的子序列存放在 r 个 (静态链组织的) 队列中
- 链式存储避免了空间浪费情况

97	53	88	59	26	41	88'	31	22
----	----	----	----	----	----	-----	----	----

(a) 初始链表内容

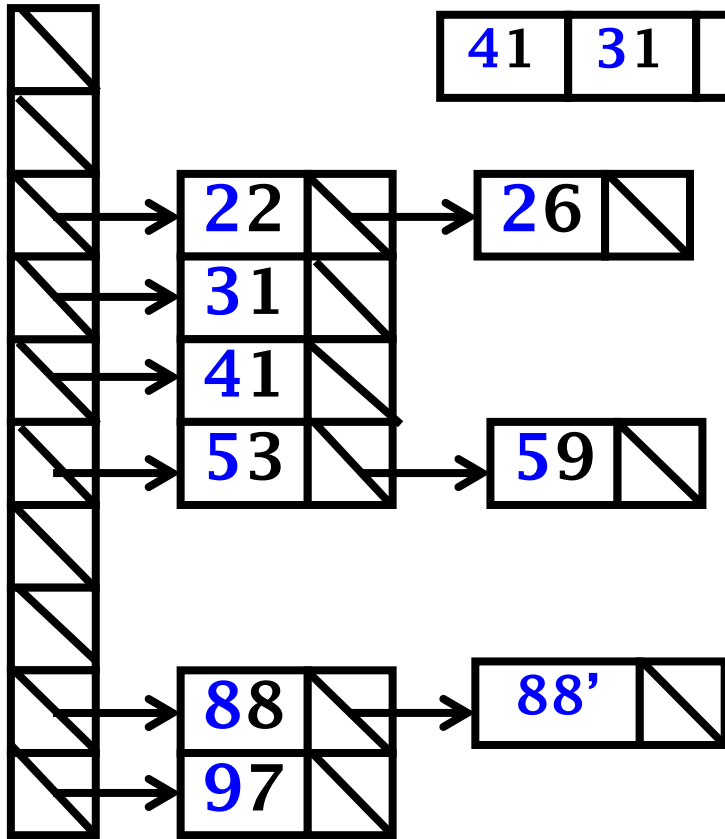


(b) 第一趟分配

(c) 第一趟收集

41	31	22	53	26	97	88	88'	59
----	----	----	----	----	----	----	-----	----

queue[0]
queue[1]
queue[2]
queue[3]
queue[4]
queue[5]
queue[6]
queue[7]
queue[8]
queue[9]



41	31	22	53	26	97	88	88'	59
----	----	----	----	----	----	----	-----	----

(d)第二趟分配

(e) 第二趟收集结果
(最终结果)

22	26	31	41	53	59	88	88'	97
----	----	----	----	----	----	----	-----	----

8.6.2 基数排序

静态队列定义

```
class Node {           // 结点类
public:
    int key;            // 结点的关键码值
    int next;           // 下一个结点在数组中的下标
};

class StaticQueue { // 静态队列类
public:
    int head;
    int tail;
};
```

8.6.2 基数排序

基于静态链的基数排序

```
template <class Record>
void RadixSort(Record *Array, int n, int d, int r) {
    int i, first = 0;           // first指向第一个记录
    StaticQueue *queue = new StaticQueue[r];
    for (i = 0; i < n-1; i++)
        Array[i].next = i + 1; // 初始化静态指针域
    Array[n-1].next = -1;       // 链尾next为空
    // 对第i个排序码进行分配和收集，一共d趟
    for (i = 0; i < d; i++) {
        Distribute(Array, first, i, r, queue);
        Collect(Array, first, r, queue);
    }
    delete[] queue;
    AddrSort(Array, n, first); // 整理后，按下标有序
}
```



```
template <class Record>
void Distribute(Record *Array, int first, int i, int r,
StaticQueue *queue) {
    int j, k, a, curr = first;
    for (j = 0; j < r; j++) queue[j].head = -1;
    while (curr != -1) { // 对整个静态链进行分配
        k = Array[curr].key;
        for (a = 0; a < i; a++) // 取第i位排序码数字k
            k = k / r;
        k = k % r;
        if (queue[k].head == -1) // 把数据分配到第k个桶中
            queue[k].head = curr;
        else Array[queue[k].tail].next = curr;
        queue[k].tail = curr;
        curr = Array[curr].next; // curr移动, 继续分配
    }
}
```



```
template <class Record>
void Collect(Record *Array, int& first, int r, StaticQueue
*queue) {
    int last, k=0;           // 已收集到的最后一个记录
    while (queue[k].head == -1) k++; // 找到第一个非空队
    first = queue[k].head; last = queue[k].tail;
    while (k < r-1) {        // 继续收集下一个非空队列
        k++;
        while (k < r-1 && queue[k].head == -1)
            k++;
        if (queue[k].head != -1) { // 试探下一个队列
            Array[last].next = queue[k].head;
            last = queue[k].tail; // 最后一个为序列的尾部
        }
    }
    Array[last].next = -1;    // 收集完毕
}
```



链式基数排序算法代价分析

- 空间代价
 - n 个记录空间
 - r 个子序列的头尾指针
 - $\Theta(n + r)$
- 时间代价
 - 不需要移动记录本身，
只需要修改记录的 next 指针
 - $\Theta(d (n+r))$

基数排序效率

- 时间代价为 $\Theta(d n)$, 实际上还是 $\Theta(n \log n)$
 - 没有重复关键码的情况, 需要 n 个不同的编码来表示它们
 - 也就是说, $d \geq \log_r n$, 即在 $\Omega(\log n)$ 中



思考：排序该排什么？

```
Template <class Elem >
void BucketSort(int n,int min,int max) {
    int i, num;
    int count[max-min];
    for (i=min; i<max; i++) // 计数器初始为0
        count[i]=0;
    for (i=0; i<n; i++) {    // 开始统计计数
        cin >> num;
        count[num]++;
    }
    for (i=min; i<max; i++) //按顺序输出
        while (count[i]>0) {
            cout << i;
            count[i]--;
        }
}
```




思考

1. 桶排事先知道序列中的记录都位于某个小区间段 $[0, m)$ 内。m 多大合适？超过这个范围怎么办？
2. 桶排中，count 数组的作用是什么？为什么桶排要从后往前收集？

3. 顺序和链式基数排序的优劣？

4. 链式基数排序的结果整理？

index	0	1	2	3	4	5	6	7	8
key		49	38	65	97	76	13	27	52
next	6	8	1	5	0	4	7	2	3



有头结点的单链表的插入算法

链式基数排序的结果



数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”

<http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/>

张铭，王腾蛟，赵海燕

高等教育出版社，2008.6。“十一五”国家级规划教材