数据结构之排序

秦新良

2014年7月25日

目　录

1 排序

本篇介绍常用的排序算法并给出其实现，最后对各种排序算法作比较。代码1给出了排序算法链表实现的链表定义。

```
#ifndef __LIST_H__
#define __LIST_H__

typedef struct node_s  node_t;
typedef struct node_s *list_t;

struct node_s
{
    int     data;
    node_t *next;
};

#endif /* __LIST_H__ */
```

代码 1: 链表

1.1 插入排序

插入排序的一个非常形象的说明就是打牌。起牌的过程中，每次我们摸起一张牌后，都会和手中已有的牌做比较，然后将牌插到相应的位置，如图1所示。

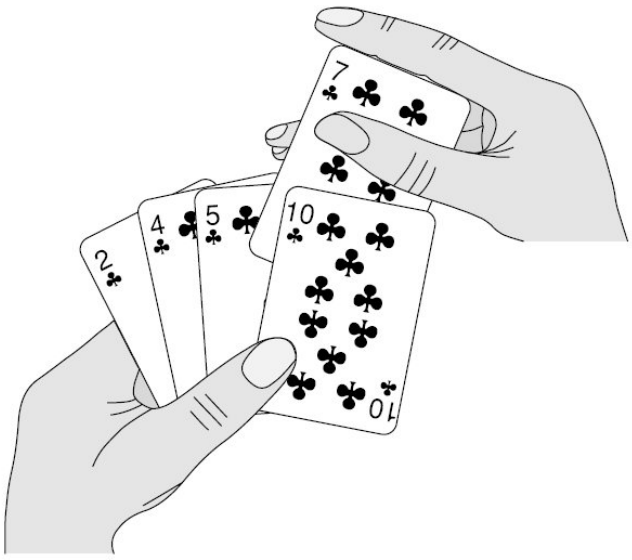


图 1: 打牌

整个起牌的过程，就是对手中的牌的一次插入排序过程。当然，也不排除有的人对手中的牌并不排序，因为这样可以有效的防止“邻居偷窥“，该场景就不在我们的举例范围之内了。

插入排序的时间复杂度为O(n^2)，排序过程中只需O(1)个元素的额外空间即可完成整个数组的排序。代码2给出了插入排序的C代码实现。

```c
void insert_sort(int array[], int size)
{
    int i = 0;
    int j = 0;
    int temp = 0;

    for (i = 1; i < size; ++i)
    {
        for (j = i; j > 0; --j)
        {
            if (array[j] < array[j-1])
            {
                temp = array[j];
                array[j] = array[j-1];
                array[j-1] = temp;
            }
            else
            {
                break;
            }
        }
    }

    return;
}
```

代码 2: 插入排序

为了更清楚的理解该算法，下面以排序6、5、3、2、8、7、1、4这一组数为例来说明插入排序的算法原理。

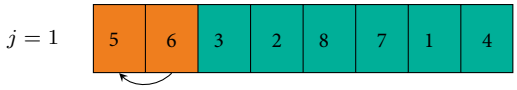
i = 0[1]时：

$j = 0$

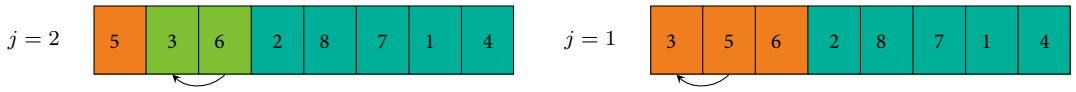
6	5	3	2	8	7	1	4

i = 1 时：

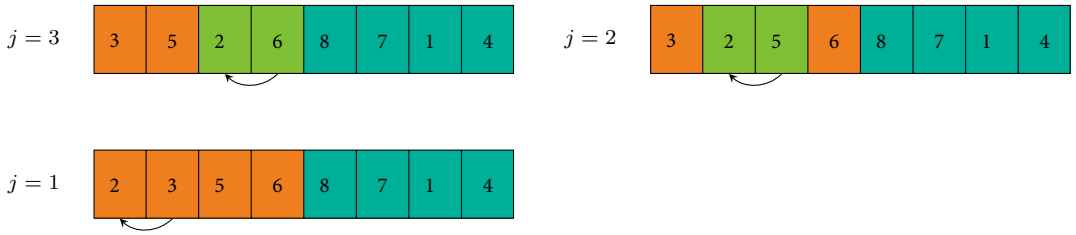
[1]真实的排序不会从i = 0开始，这里用（i = 0）&&（j = 0）表示初始未排序。



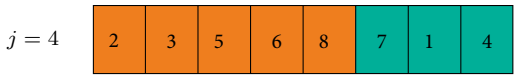
i = 2 时：



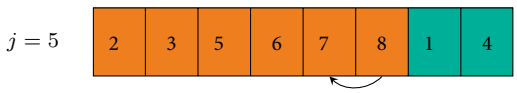
i = 3 时：



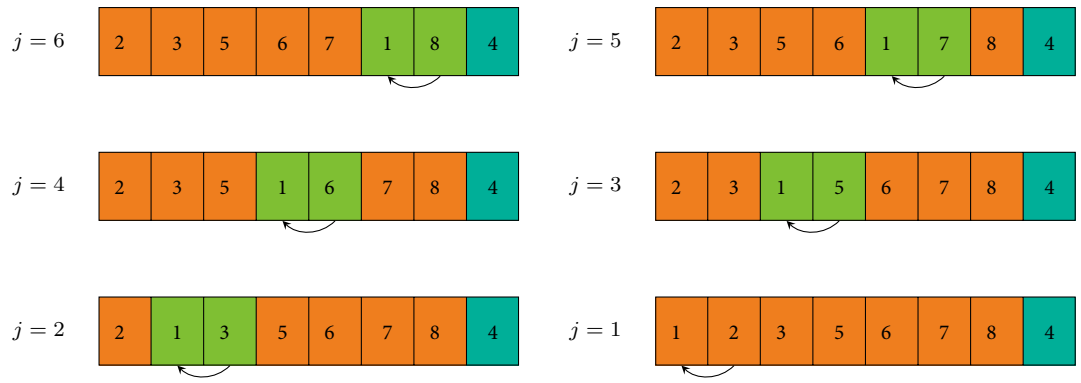
i = 4 时：



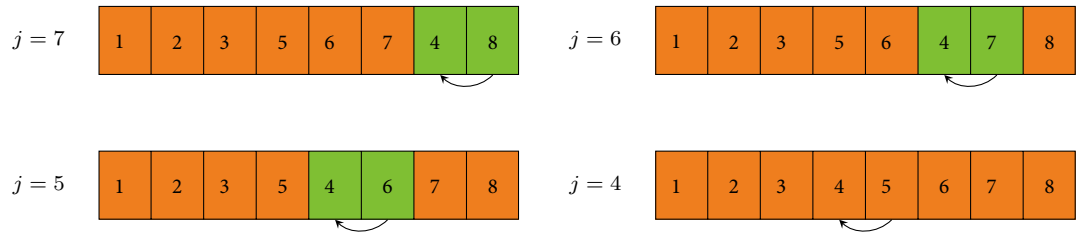
i = 5 时：



i = 6 时：



i = 7 时：



插入排序是稳定的[2]排序算法，在待排序元素基本有序的情况下，其最好可达到接近O(n)的时间复杂度。在元素基本有序或元素不多的情况下，可选择插入排序。

最后，我们以插入排序的链表实现来结束本小节，见代码3。

```
1  void list_insert_sort(list_t *list_head)
2  {
3      if ((NULL == list_head) || (NULL == *list_head))
4      {
5          return;
6      }
7
8      node_t *p = NULL;
9      node_t *c = NULL;
10     node_t *n = NULL;
11     node_t *t = NULL;
12
13     n = (*list_head)->next;
14     (*list_head)->next = NULL;
```

[2]假定在待排序的记录序列中，存在多个具有相同的关键字的记录，若经过排序，这些记录的相对次序保持不变，即在原序列中，ri=rj，且ri在rj之前，而在排序后的序列中，ri仍在rj之前，则称这种排序算法是稳定的；否则称为不稳定的。

```
15
16      while (NULL != n)
17      {
18          p = c = *list_head;
19          do
20          {
21              if (n->data < c->data)
22              {
23                  t = n;
24                  n = n->next;
25
26                  if (p == c)
27                  {
28                      t->next = c;
29                      *list_head = t;
30                  }
31                  else
32                  {
33                      t->next = c;
34                      p->next = t;
35                  }
36
37                  break;
38              }
39              else
40              {
41                  if (p == c)
42                  {
43                      c = c->next;
44                  }
45                  else
46                  {
47                      p = c;
48                      c = c->next;
49                  }
50
51                  if (NULL == c)
52                  {
53                      t = n;
54                      n = n->next;
55                      t->next = NULL;
56                      p->next = t;
57                      break;
58                  }
59              }
60          } while (1);
61      }
62
63      return;
64  }
```

代码 3: 插入排序

1.2 冒泡排序

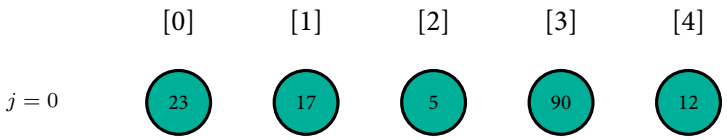
冒泡排序可以说是所有排序算法中最“有名”的一个，因为其名字好记、原理简单：循环对数组中相邻的元素两两比较，每次循环结束后，都会有一个元素在其正确的位置上，即已排序。代码4给出其C语言的实现。

```c
void bubble_sort(int array[], int size)
{
    int i = 0;
    int j = 0;
    int temp = 0;

    for (i = size - 1; i > 0 ; --i)
    {
        for (j = 0 ; j < i; ++j)
        {
            if (array[j + 1] < array[j])
            {
                temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
        }
    }

    return;
}
```

代码 4: 冒泡排序

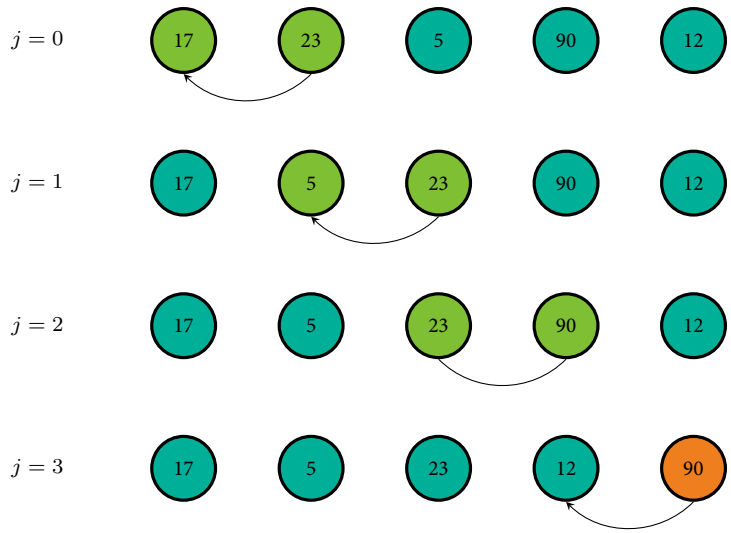
从代码可以看出，冒泡排序的时间复杂度为$O(n^2)$。同插入排序一样，排序过程中只需要一个额外的元素空间。算法虽然简单，我们还是通过对23、17、5、90、12这一组数排序来说明该算法的原理。

i = 5[3]时：

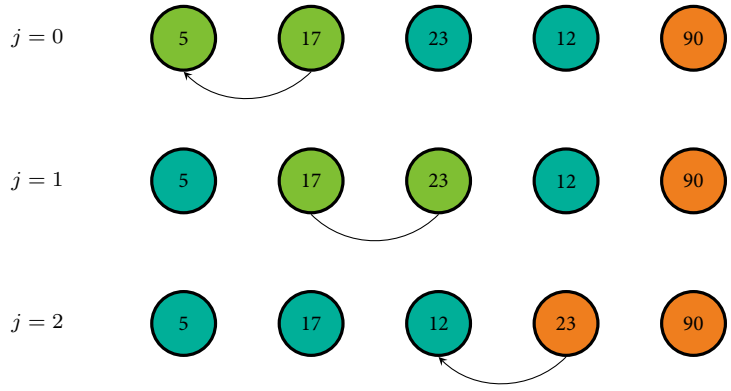


i = 4 时：

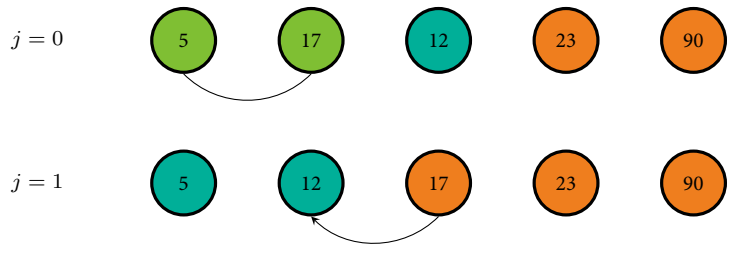
[3]初始状态，实际从i = 4 开始。


$j=0$
17 23 5 90 12
$j=1$
17 5 23 90 12
$j=2$
17 5 23 90 12
$j=3$
17 5 23 12 90

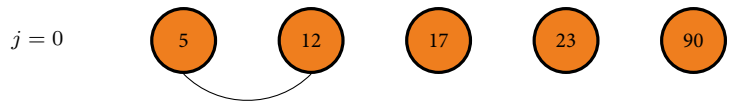
i = 3 时：


$j=0$
5 17 23 12 90
$j=1$
5 17 23 12 90
$j=2$
5 17 12 23 90

i = 2 时：


$j=0$
5 17 12 23 90
$j=1$
5 12 17 23 90

i = 1 时：



冒泡排序是一种稳定的排序算法，前提是排序过程中如果两个元素相同，即array[j + 1] == array[j]，不对两个元素交换。最后同样给出冒泡排序的链表实现来作为本小结的结尾，见代码5。

```
/* with the bubble sort, after each iteration, the last
   element is on the right place */
void list_bubble_sort(list_t *list_head)
{
    if ((NULL == list_head) || (NULL == *list_head))
    {
        return;
    }

    node_t *c = NULL;
    node_t *p = NULL;
    node_t *n = NULL;
    node_t *e = NULL;

    while (e != (*list_head)->next)
    {
        p = c = *list_head;
        n = c ->next;
        while (e != n)
        {
            if (n->data < c->data)
            {
                if (p == c)
                {
                    c->next = n->next;
                    n->next = c;
                    *list_head = n;
                    p = n;
                    n = c->next;
                }
                else
                {
                    p->next = n;
                    c->next = n->next;
                    n->next = c;
                    p = n;
                    n = c->next;
                }
            }
            else
```

```
            {
                if (p == c)
                {
                    c = n;
                    n = c->next;
                }
                else
                {
                    p = c;
                    c = n;
                    n = c->next;
                }
            }
        }

        e = c;
    }

    return;
}
```

代码 5: 冒泡排序