数据结构之排序

秦新良

2014年7月22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。

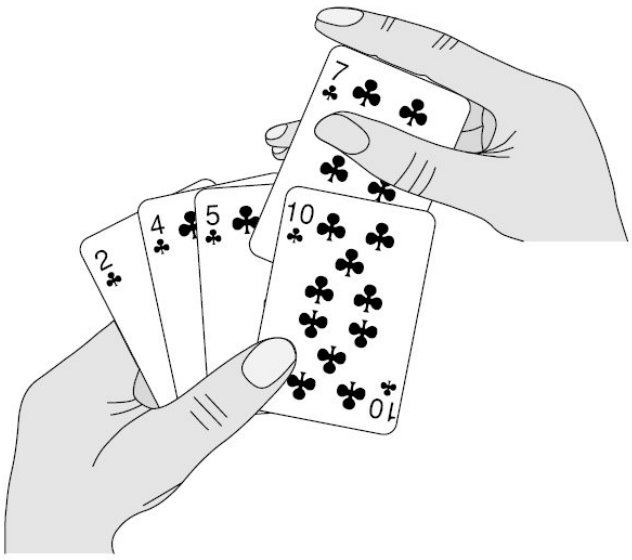


图 1: 打牌

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。

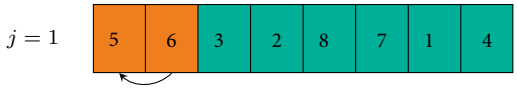
i = 0[1]时：

$j = 0$

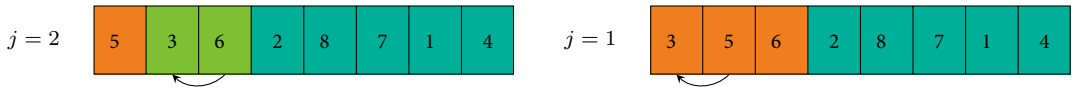
6	5	3	2	8	7	1	4

i = 1 时：

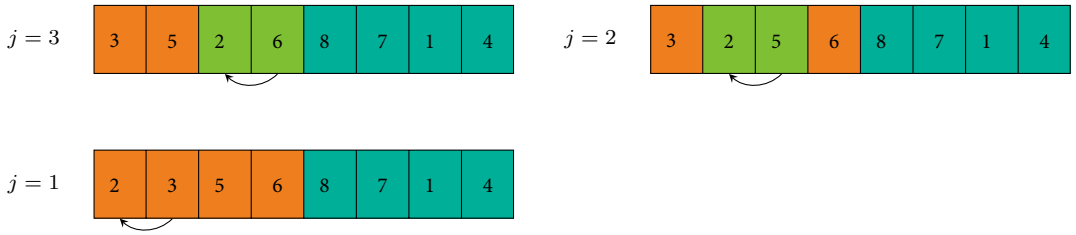
[1]真实的排序不会从i = 0开始，这里用（i = 0）&&（j = 0）表示初始未排序。



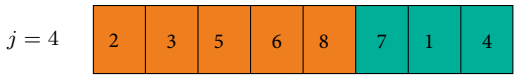
i = 2 时：



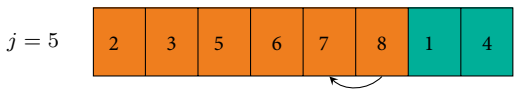
i = 3 时：



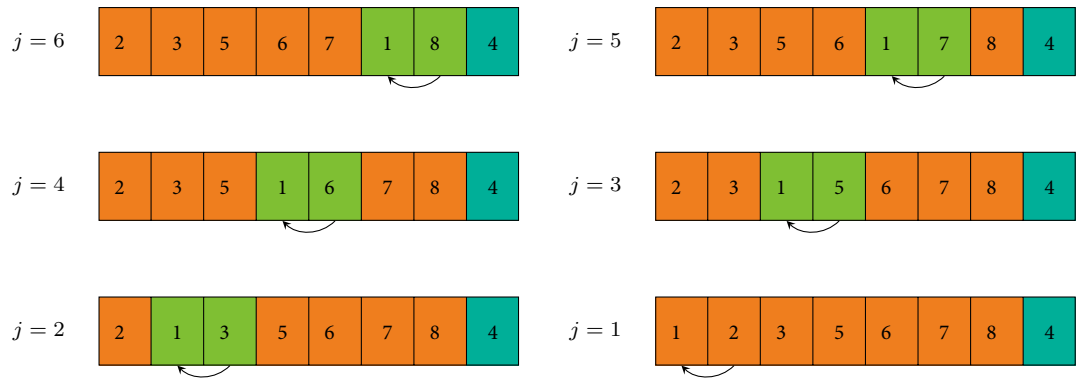
i = 4 时：



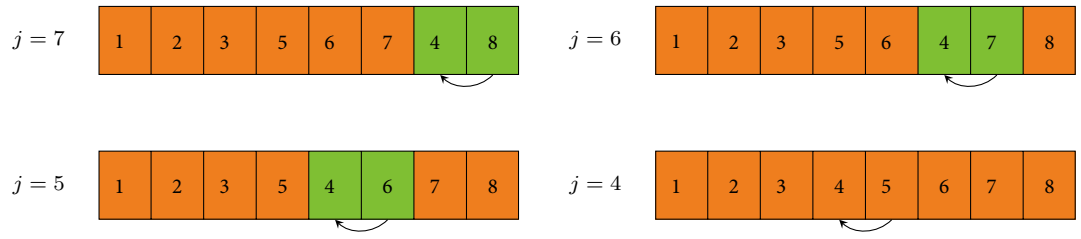
i = 5 时：



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

该场景一般是由于通信的一方异常终止，而另一方并没有感知到，这时候如果另一方继续在原有的连接上发送数据，对端就会直接将链路重置。该场景可通过down掉一台主机的网卡，然后再将该主机上的服务重启来构造，实际的网络环境中几乎不会出现。