链滴

冒泡排序之优化策略

作者：zhou-tao

原文链接：https://ld246.com/article/1571455004234

来源网站：链滴

摘要：冒泡排序应该是大部分人学到的第一个排序算法, 它思想简单, 是入门排序算法的好选择. 然而于它的时间复杂度为O(n^2), 所以除了学习它的时间之外我们就很少的想到它了, 通常提到更多的还快速排序等时间复杂度更低的排序算法. 然而, 在对经典的冒泡排序进行改善之后, 在一定的条件之下, 然有它的用武之地.

接下来我们首先介绍冒泡的普通实现，再通过对基本冒泡实现代码的缺陷分析进行一步步针对性的优
。

基本冒泡实现

先简要概述一下冒泡的基本实现原理：通过相邻元素两两对比将最大或最小的那一个元素往另一边移，这样下来从左到右每一轮都能得到一个最大或最小元素，再通过一定轮次的比较即实现有序数列。

这里以从小到大排序为例，冒泡排序经典代码实现如下：

```
//一、基本冒泡
let base = arr => {
  for(let i = 0;i < arr.length - 1;i++){
    for(let j = 0;j < arr.length - i - 1;j++){
      if(arr[j + 1] < arr[j]){
        swap(arr, j, j + 1);
      }
    }
  }
  console.log('baseSort:',arr)
}
```

其中swap()函数是用来交换数据的，由于之后的优化策略均要使用到数据交换，秉承程序猿界的DRY则，故封装之~

```
//数据交换
let swap = (array,index1,index2) => {
  [array[index1],array[index2]] = [array[index2],array[index1]]      //es6解构
}
```

优化一：提前结束有序

由于冒泡排序是相邻两元素依次相比，则若出现一轮未发生元素更换的情况下：即为数组已经有序,则提前结束。这样的好处是当数组在前几轮对比已经有序之后可以节省不必要的元素对比次数，比如需排序的数组为【3,1,2,4,5】这样第一大轮比较之后数组已经有序，则会再比一轮发现元素没有发生过换即退出排序结束。

代码实现如下：

```
//二、优化 1
//简介: 由于冒泡是相邻两元素依次相比，则若出现一轮未发生元素更换的情况下：即为数组已经有序则可提前结束
let op1 = arr => {
  for(let i = 0;i < arr.length - 1;i++){
    //每大轮给与一个标记
    let flag = true
    for(let j = 0;j < arr.length - i - 1;j++){
```

```
      if(arr[j + 1] < arr[j]){
        swap(arr, j, j + 1);
    //若有交换赋值
        flag = false;
      }
    }
    //没交换即有序可提前结束
    if(flag) break;
  }
  console.log('op1:',arr)
}
```

优化二：跳过部分有序

结合优化一中讨论的，我们要使得整轮都有序的情况下才可以提前结束排序操作。但当我们遇到比如3,2,1,4,5,6】这种数组时，后面的4,5,6已经有序的情况下我们在每大轮的比较里可以跳过这部分的小两两相比，这样又可以提高一部分的效率。

```
//三、优化 2(在上一步优化的基础上)
//简介: 记录最后一个更换元素时的索引值，下一轮的比较可以自动跳过之后的有序队列
let op2 = arr => {
  let flagIndex;
  let lastIndex = arr.length - 1;
  while(true){
    let flag = true
    //跳过后面有序的部分
    for(let j = 0;j < lastIndex;j++){
      if(arr[j + 1] < arr[j]){
        swap(arr, j, j + 1);
        flag = false;
        //记录最后一个交换时的索引
        flagIndex = j;
      }
    }
    lastIndex = flagIndex
    if(flag) break;
  }
  console.log('op2:',arr)
}
```

优化三：两级反转

以上的优化总的来说都是分大小两轮，每一大轮排出一个最值到边界。而接下来我们有一个想法，就一大轮我们同时在左右两边各排出一个最大值与最小值，这样我们的排序效率又会更进一步了。

```
//四、优化 3
//简介: 每大轮从左到右选出最大的同时从右到左选出最小的并结合前两次优化
let op3 = arr => {
  //左边开始索引
  let start = 0;
  //右边开始索引
  let end = arr.length - 1;
  while(start < end){
```

```
    //每大轮向右排出最大值
    for(let i = start;i < end;i++){
      if(arr[i] > arr[i + 1]) swap(arr, i, i + 1)
    }
    //排出一个最大值，终点向左移一位
    end--;
    //每大轮向左排出最小值
    for(let i = end;i > start;i--){
      if(arr[i] < arr[i - 1]) swap(arr, i, i - 1)
    }
    //排出一个最小值，起点向右移一位
    start++;
  }
  console.log('op3',arr)
}
```

到这里，我们的优化基本就到极致，但我们没有利用到前两步的优化思想，故 综上：

```
//五、综上
let op4 = arr => {
  let start = 0,startPos = start;
  let end = arr.length - 1,endPos = end;
  while(start < end){
    //let flag = true
    for(let i = start;i < end;i++){
        if(arr[i] > arr[i + 1]){
          swap(arr, i, i + 1);
          endPos = i;
          //flag = false;
        }
    }
    //利用endPos与end位置关系来替代flag变量
    if(endPos === end){
      console.log(arr)
      return;
    }
    end = endPos;
    for(let i = end;i > start;i--){
      if(arr[i] < arr[i - 1]){
        swap(arr, i, i - 1);
        startPos = i;
        // flag = false;
      }
    }
    if(startPos === start){
      console.log(arr)
      return;
    }
    start = startPos;
  }
}
```

最后提醒： 注意这里的注释，上面判断一大轮中没有交换元素的办法是新建一个变量来辨别，这里利一大轮前后的位置关系来辨别更加简介，OVER~