排 序 技 术 研 讨

目录/CONTENTS

01

TOP K问题实例

TOP K问题实例

什么是 Top K 问题?
简单来说就是在一组数据里面找到频率出现最高的前 K 个数，或前 K 大（当然也可以是前 K 小）的数。
下题便是经典的TOP K 问题。
剑指 Offer 40. 最小的k个数
输入整数数组 arr，找出其中最小的 k 个数。例如，输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。
1.
输入：arr = [3,2,1], k = 2
输出：[1,2] 或者 [2,1]
2.
输入：arr = [0,1,2,1], k = 1
输出：[0]
下面以该题为例，介绍 Top K 问题的解法。

02

排序算法的使用

排 序 算 法 的 使 用

1.全局排序

算法思想：

直接利用c++头文件algorithm，利用sort函数给自动排序。

求解过程：

sort() 函数是基于快速排序实现，具体不再赘述，后面会提及快排。

算法步骤：

sort(arr,arr+n);

return arr[1,k];

性能分析：

O(n*lg(n))

额外补充：

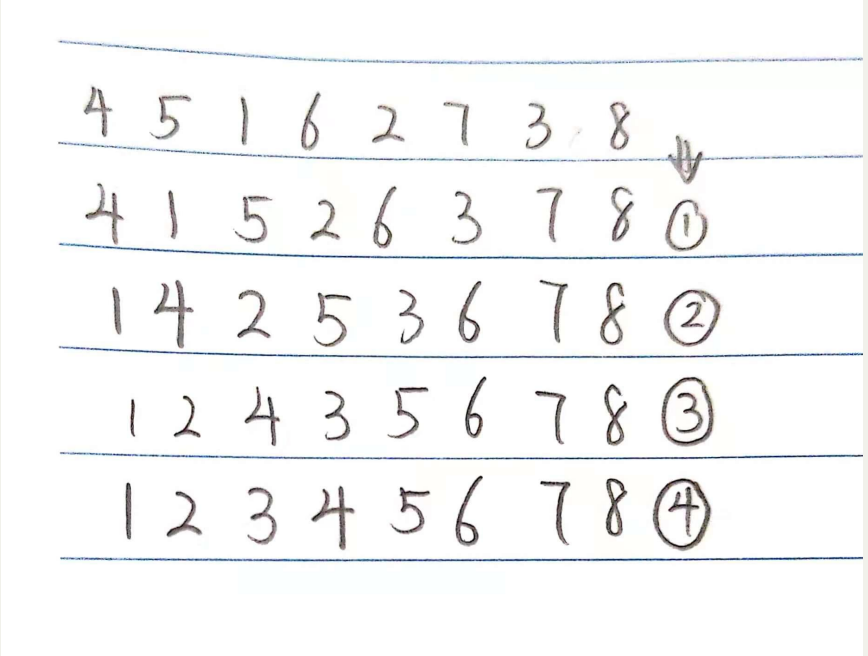
这种排序问题在于，明明只需要局部排序，却对全部元素进行了排序，提升了时间复杂度，可以进一步优化。

排 序 算 法 的 使 用

2.冒泡排序

算法思想：属于交换排序中比较简单的一种排序方法，对所有相邻记录的关键字值进行比较，如果逆顺（不符合顺序），则将其交换。

求解过程：



排 序 算 法 的 使 用

算法步骤：

```
int findKthsmallest (arr, k) { // 进行k轮冒泡排序
    bubbleSort(arr, k)
    return arr[k]
}

void bubbleSort (arr, k) {
    for (int i = 0; i < k; i++) { // 提前退出冒泡循环的标识位
        bool flag = false;
        for (int j = 0; j < arr.length() - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                flag = true;
                // 表示发生了数据交换
            }
        }
        // 没有数据交换
        if(!flag) break;
    }
}
```

性能分析：

时间复杂度：最好时间复杂度 O(n)，平均时间复杂度 O(n*k)

空间复杂度：O(1)

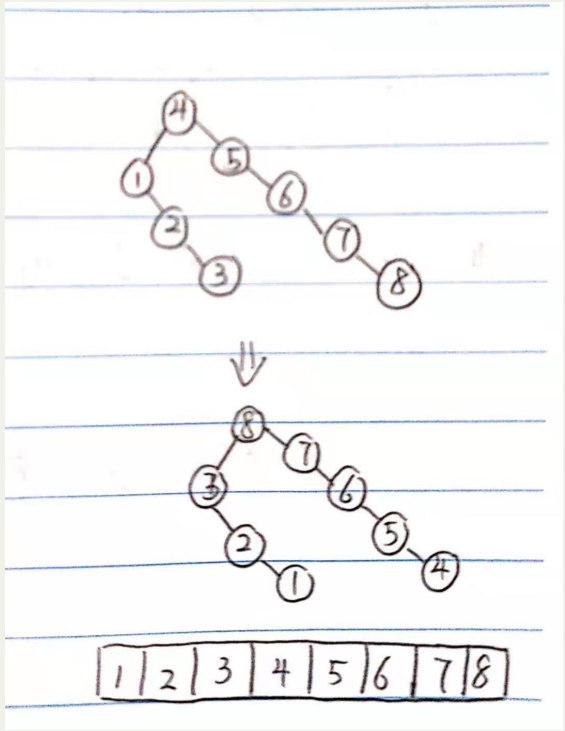
排 序 算 法 的 使 用

3.堆排序

算法思想：

我们在BST中学习过最小值堆最大值堆，因此我们可以构造最大值堆来获得k个最小值。我们先将数组转化为一个满足堆定义的序列，然后将堆顶的最大元素取出，再对剩下的元素排成堆并取出堆顶元素，如此下去，直到堆为空。将第一个堆顶元素存储到数组的第n-1个位置上，将第二个堆顶元素存储到第n-2个位置上，如此得到了一个由小到大排列的数组。

求解过程：



排 序 算 法 的 使 用

算法步骤：

```
void headsort(int A[],int n){
int mval;
maxheap<int,Comp> H(A,n,n);//建堆
for(int i=0;i<n;i++)
H.removemax(mval);//移除堆顶元素
}
```

性能分析：

时间复杂度：建堆要用O(n)时间，并且n次取堆的最大元素要用O(logn)时间，因此时间代价为O(nlogn)

空间复杂度：O(n)

额外补充：

我们还可以用另一种堆排序，虽然时间复杂度相差不大，但也可以参考借。其核心思想为从数组中取前 K 个数，构造一个最小值堆，从 K+1 位开始遍历数组，每一个数据都最小值堆的堆顶元素进行比较，如果大于堆顶元素，则不做任何处理，继续遍历下一元素；如果小于堆顶元素，则将这个元素替换掉堆顶元素，然后再堆化成一个最小值堆。遍历完成后，堆中的数据就是前 K 小的数据。

堆排序的优点在于，在一个动态数组中求 Top K 元素时，我们可以使用堆，维护一个 K 大小的最小值堆，当有数据被添加到数组中时，就将它与堆顶元素比较，如果比堆顶元素小，则将这个元素替换掉堆顶元素，然后再堆化成一个最小值堆；如果比堆顶元素大，则不做处理。这样，每次求 Top K 问题的时间复杂度仅为 O(logK)。

排 序 算 法 的 使 用

4.快速排序

算法思想:

为了避免全局排序和堆排序中额外的操作，节省时间和空间，我们可以选择快速排序。快排利用了分治策略，降一个复杂的问题分解成两个或多个相似的问题，不断分解，直至更小的问题可以简单求解，求解子问题，将原问题的解为子问题的合并。

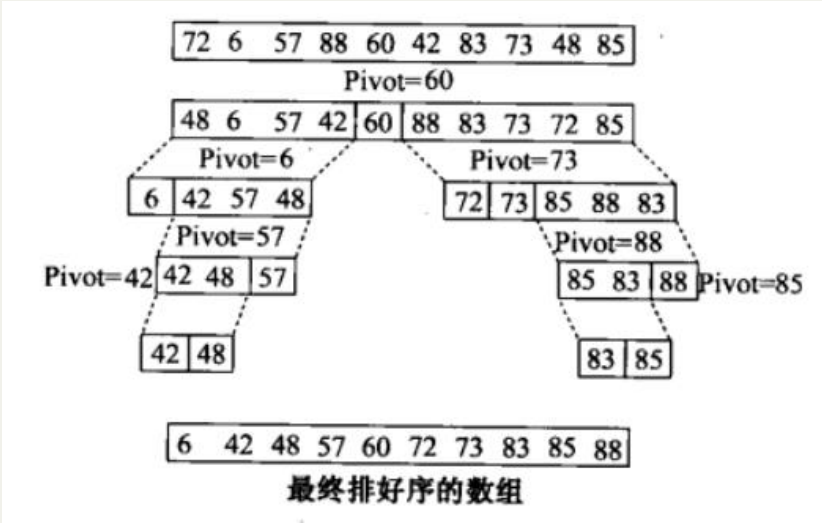
求解过程:

快排可以简单分为三步。

首先从序列中选取一个数作为基准数。

将比这个数大的数全部放到它的右边，把小于或者等于它的数全部放到它的左边。

然后分别对基准左右两边重复以上操作，直到数组完全排序。（基准可以每次都选择最左边的元素作为基准）



排 序 算 法 的 使 用

算法步骤：

```
void qsort(int A[],int i,int j){
if(j<=i) return;//不对下标为0或1元素排序
int pivotindex=findpivot(A,i,j);
swap(A,pivtindex,j);//将pivot放到末尾，k是右区域第一个位置
int k=pivotindex<int,Comp>(A,i-1,j,A[j]);
swap(A,k,j);
qsort<int,Comp>(A,i,k-1);
qsort<int,Comp>(A,k+1,j);
}

int findpiont(int A[],int i,int j){return (i+j)/2;}//寻找合适的pivot（轴值，或称为基准）

int partition(int A[],int l,int r,int& pivot){//用来寻找多少个结点比轴值小，关键码值比轴值小的结点放到数组低端
//大的放高端
do{while(Comp::lt(A[++l],pivot));//交换记录直到数组两段下标相遇为止
while(r!=0)&&Comp:gt(A[--r],pivot);//l向左r向右
swap(A,l,r);}while(l<r);
swap(A,l,r);
return l;//返回右边第一个位置
}
```

性能分析：

时间复杂度：当轴值不能很好的分割数组时，即一个子数组中没有结点，另一个子数组中有n-1个结点，这种情况下处理的子问题只比原问题规模-1，因此时间代价为O(n*n)。而在正常情况下，轴值能较好的分割数组，此时时间代价为O(nlogn)

空间复杂度：O(nlogn)

排 序 算 法 的 使 用

5.快速选择法

算法思想：

本质上是基于快排对TOP K问题进行了优化，由于我们只需要TOP K，因此我们可以在每执行一次快排的时候，比较基准值是否在n-k位置上。

求解过程：

如果小于 n-k ，则第 k 个最大值在基准值的右边，我们只需递归快排基准值右边的子序列即可；
如果大于 n-k ，则第 k 个最大值在基准值的做边，我们只需递归快排基准值左边的子序列即可；
如果等于 n-k ，则第 k 个最大值就是基准值

算法步骤：

```
void findKthsmallest(int nums[], int k) {
    return quickSelect(nums, nums+k);}
void quickSelect (int arr[], int k)  {
  return quick(arr, 0 , arr.length - 1, k);}
void quick(int arr[], int left, int right, int k) {
  int index;
  if(left < right) {// 划分数组
    index = partition(arr, left, right) // Top k
    if(k == index) {
        return arr[index];
    } else if(k < index) {// Top k 在左边
        return quick(arr, left, index-1, k);
    } else {// Top k 在右边
        return quick(arr, index+1, right, k);}
  }
return arr[left];}
```

排 序 算 法 的 使 用

```
void partition (int arr[], int left,int right) {// 取中间项为基准
  int datum = arr[Math.floor(Math.random() * (right - left + 1)) + left];
      i = left;
      j = right;// 开始调整
  while(i < j) {// 左指针右移
    while(arr[i] < datum) { i++; }// 右指针左移
    while(arr[j] > datum) { j--;}// 交换
    if(i < j) swap(arr, i, j);// 当数组中存在重复数据时，即都为datum，但位置不同，继续递增i，防止死循环
    if(arr[i] == arr[j] && i != j) { i++;}
  }
  return i;}
void swap (int arr[],int i ,int j) {// 交换
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;}
```

性能分析：

时间复杂度：平均时间复杂度O(n)，最坏情况时间复杂度为O(n*n)

空间复杂度：O(1)

6.中位数的中位数（BFPRT）算法

算法思想：

修改快速选择算法的主元选取方法，提高算法在最坏情况下的时间复杂度。在BFPTR算法中，仅仅是改变了快速选择算法中 Partion 中的基准值的选取，在快速选择算法中，我们可以选择第一个元素或者最后一个元素作为基准元，优化的可以选择随机一个元素作为基准元，而在 BFPTR 算法中，每次选择五分中位数的中位数作为基准元（即pivot），这样做的目的就是使得划分比较合理，从而避免了最坏情况的发生。

求解过程：

选取主元

将 n 个元素按顺序分为 n/5 个组，每组 5 个元素，若有剩余，舍去

对于这 n/5 个组中的每一组使用插入排序找到它们各自的中位数

对于上一步中找到的所有中位数，调用 BFPRT 算法求出它们的中位数，作为主元；

以主元为分界点，把小于主元的放在左边，大于主元的放在右边；

判断主元的位置与 k 的大小，有选择的对左边或右边递归

排 序 算 法 的 使 用

算法步骤：

```
void findKthsmallest(int nums[],int k) {
    return nums[bfprt(nums, 0, nums, nums+k)];
}

void bfprt(int arr[], int left , int right, int k) {
 int index;
  if(left < right) {// 划分数组
    index = partition(arr, left, right);// Top k
    if(k == index) {return index;}
else if(k < index) {  return bfprt(arr, left, index-1, k);} // Top k 在左边
else {return bfprt(arr, index+1, right, k);}// Top k 在右边
  }
  return left;}

int partition(int arr[],int left,int right)  {
    int datum = arr[findMid(arr, left, right);]// 基准
      i = left;
      j = right;// 开始调整
while(i < j) { while(arr[i] < datum) { i++;} // 左指针右移
                  while(arr[j] > datum) { j--;}// 右指针左移
                  if(i < j) swap(arr, i, j);// 交换
// 当数组中存在重复数据时，即都为datum，但位置不同
 // 继续递增i，防止死循环
if(arr[i] == arr[j] && i != j) {i++;}
}
  return i;}
```

排 序 算 法 的 使 用

```
/*数组 arr[left, right] 每五个元素作为一组，并计算每组的中位数，
最后返回这些中位数的中位数下标（即主元下标）。
末尾返回语句最后一个参数多加一个 1 的作用其实就是向上取整的意思，
 这样可以始终保持 k 大于 0。*/
int findMid(int arr[],int left,int right) {
    if (right - left < 5)
        return insertSort(arr, left, right);
    int n = left - 1;
    // 每五个作为一组，求出中位数，并把这些中位数全部依次移动到数组左边
    for (int i = left; i + 4 <= right; i += 5)
    { int index = insertSort(arr, i, i + 4);
      swap(arr[++n], arr[index]);}
      return findMid(arr, left, n);}// 利用 bfprt 得到这些中位数的中位数下标（即主元下标）
int insertSort  (int arr[],int left,int right) {//对数组 arr[left, right] 进行插入排序，并返回 [left, right]的中位数。
    int temp, j;
    for (int i = left + 1; i <= right; i++) {
        temp = arr[i];
        j = i - 1;
        while (j >= left && arr[j] > temp)
        {arr[j + 1] = arr[j];j--;}
        arr[j + 1] = temp;}
    return ((right - left) >> 1) + left;
}
void swap (int arr[],int  i ,int  j)  {// 交换
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;}
```

排 序 算 法 的 使 用

性能分析：

时间复杂度：最坏时间复杂度为O(n)，T(n)<=T(n/5)+T(7n/10)+c*n，若设T(n)=t*n，则有t*n<=t*n/5+t*7n/10+c*n---->t<=10c，所以T(n)=O(n)

空间复杂度：O(1)

额外补充：

之所以选择5作为分组，是因为对于奇数而言，中位数更容易计算，在3，5，7，9中选择，由于选用3，有素个数仍是n，选用7，9或者更大，在插入排序时耗时增加，常数c变大，因此得不偿失。

参　考　文　献

https://github.com/sisterAn/JavaScript-Algorithms/issues/73
前端进阶算法10：别再说你不懂topk问题了
https://blog.csdn.net/z50L2O08e2u4afToR9A/article/details/82837278
拜托，面试别再问我TopK了！！！

感 谢 观 看