九种排序算法分析与实现

昵称: Glory_D
园龄: 2年6个月
粉丝: 3
关注: 8
+加关注

< 2019年11月 >

日	一	二	三	四	五	六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

搜索

找找看

谷歌搜索

My Tags

Linux(8)
Computer Network(4)
环境配置(4)
虚拟机(4)
【书】TCP/IP网络编程(3)
Algorithm(3)
ARM开发(3)
Bash Shell(2)
文件I/O(2)
Data Structures(1)
更多

随笔分类

编程技法(3)
读书笔记(4)
技术杂记(5)
开发工具(6)
项目总结(1)

随笔档案

2019年11月(3)
2018年8月(1)
2018年7月(1)
2018年5月(3)
2018年4月(1)
2017年11月(2)
2017年10月(3)
2017年9月(5)

Recent Comments

1. Re:九种排序算法分析与实现
@ dhchen2012是的，多谢提醒，默认bgn=0了...
--Glory_D
2. Re:九种排序算法分析与实现
楼主，希尔排序的第二个loop里应该是 i < bgn + step 吧
--dhchen2012
3. Re:九种排序算法分析与实现
@ 天地辽阔看到你已经实现了哈...
--Glory_D
4. Re:九种排序算法分析与实现

简介：总的来说，排序算法共有八大类，即冒泡排序、选择排序、快速排序、插入排序、希尔排序、归并排序、基数排序以及堆排序等，本文另外也介绍了桶排序。编程语言使用了C/C++（其实主要用的C），3个经常出现的函数形参，arr - 待排序数组名（首元素地址）、bgn - 待排序数组起始排序元素位置（有时我们仅需要对数组中某一段元素进行排序，但通常bgn = 0，即arr首元素位置）、end - 待排序数组截止排序尾元素的下一个位置（即该位置无效，不可引用）。文中均已升序为例，降序原理相同。

时间复杂度：描述该算法在处理大量数据时，总的来说其时间效率的参考； 稳定性：描述算法对原始序列处理前后，该序列相等大小的元素前后位置是否发生改变

两个常用的函数：1、获取数组最大元素值； 2、交换两个整形元素。代码如下：

```
//获取整形数组的最大值
//NOTE: 默认arr非空
int getMaxValue(const vector<int> &arr)
{
    int max = INT_MIN;
    for (auto val : arr)
    {
        if (val > max)
            max = val;
    }
    return max;
}

/*交换两个整形值*/
void mySwap(int *pa, int *pb)
{
    int tmp = *pa;
    *pa = *pb;
    *pb = tmp;
}
```

1、**冒泡排序** - 依次比较相邻两元素，若前一元素大于后一元素则交换之，直至最后一个元素即为最大；然后重新从首元素开始重复同样的操作，直至倒数第二个元素即为次大元素；依次类推。如同水中的气泡，依次将最大或最小元素气泡浮出水面。

时间复杂度：**$O(N^2)$** 稳定性：**稳定**

```
/*冒泡排序*/
void bubbleSort(vector<int> &arr, int bgn, int end)
{
    /*isLoop用于指示依次遍历中是否发生元素交换，若没有，则已是有序数列，退出即可*/
    bool isLoop = true;
    for (int i = end; true == isLoop && i > bgn; --i)
    {
        isLoop = false;
        for (int j = bgn + 1; j < i; ++j)
        {
            if (arr[j] < arr[j - 1])
            {
                mySwap(&arr[j], &arr[j - 1]);
                isLoop = true;
            }
        }
    }
}
```

楼主，请问你这个代码折叠的功能是怎么实现的呀?

--天地辽阔

5. Re:九种排序算法分析与实现

@ 猫尼玛嗯，是的，多谢提醒，当时没考虑负值，初始化为INT_MIN即可...

--Glory_D

Top Posts

1. 九种排序算法分析与实现(41678)
2. Linux下MySQL 数据库的基本操作(40966)
3. Win7 环境下虚拟机内 Samba 服务器的安装、配置以及与主机的通信实现(1194)
4. J-Link固件烧录以及使用J-Flash向arm硬件板下载固件程序(462)
5. 如何编写Makefile，一份由浅入深的Makefile全攻略(406)

推荐排行榜

1. 九种排序算法分析与实现(3)
2. Linux常见目录及命令介绍(1)
3. J-Link固件烧录以及使用J-Flash向arm硬件板下载固件程序(1)

2、**选择排序** - 首先初始化最小元素索引值为首元素，依次遍历待排序数列，若遇到小于该最小索引位置处的元素则刷新最小索引为该较小元素的位置，直至遇到尾元素，结束一次遍历，并将最小索引处元素与首元素交换；然后，初始化最小索引值为第二个待排序数列元素位置，同样的操作，可得到数列第二个元素即为次小元素；以此类推。

时间复杂度：**O(N^2)**　　稳定性：**不稳定**

```
/*选择排序*/
void selectSort(vector<int> &arr, int bgn, int end)
{
    for (int i = bgn; i < end; ++i)
    {
        int minIndex = i;
        for (int j = i + 1; j < end; ++j)
        {
            if (arr[j] < arr[minIndex])
                minIndex = j;
        }
        if (minIndex != i)
            mySwap(&arr[i], &arr[minIndex]);
    }
}
```

3、**快速排序** - （类似于选择排序的定位思想）选一基准元素，依次将剩余元素中小于该基准元素的值放置其左侧，大于等于该基准元素的值放置其右侧；然后，取基准元素的前半部分和后半部分分别进行同样的处理；以此类推，直至各子序列剩余一个元素时，即排序完成（类比二叉树的思想，from up to down）

时间复杂度：**O(NlogN)**　　稳定性：**不稳定**

```
/*快排*/
void quickSort(vector<int> &arr, int bgn, int end)  //arr must be the reference of real param
{
    //数组arr空or仅有一个元素则退出
    if (bgn >= end - 1)
        return;

    int lindex = bgn;
    int rindex = end - 1;
    int std = arr[lindex];
    while (lindex < rindex)
    {
        while (lindex < rindex)
        {
            if (arr[rindex] < std)
            {
                arr[lindex++] = arr[rindex];
                break;
            }
            --rindex;
        }

        while (lindex < rindex)
        {
            if (arr[lindex] >= std)
            {
                arr[rindex--] = arr[lindex];
                break;
            }
            ++lindex;
        }
    }

    arr[lindex] = std;
    quickSort(arr, bgn, lindex);
```

```
    quickSort(arr, rindex + 1, end);
}
```

4、**插入排序** - 数列前面部分看为有序，依次将后面的无序数列元素插入到前面的有序数列中，初始状态有序数列仅有一个元素，即首元素。在将无序数列元素插入有序数列的过程中，采用了逆序遍历有序数列，相较于顺序遍历会稍显繁琐，但当数列本身已近排序状态效率会更高。

时间复杂度：**$O(N^2)$**　　稳定性：**稳定**

```
/*插入排序*/
void insertSort(vector<int> &arr, int bgn, int end)
{
    for (int i = bgn + 1; i < end; ++i)
    {
        /*
        * 分为1,2两部分处理，可以囊括j = beg - 1时的情况
        * 即需要将arr[i]插入到首元素前的位置，若使用一个for
        * 包括这两部分，则会在发生这种情况时退出
        */
        /*1*/
        int j = i - 1;
        for ( ; j >= bgn; --j)
            if (arr[j] <= arr[i])
                break;
        /*2*/
        if (j != i - 1)
        {
            int temp = arr[i];
            for (int k = i; k > j + 1; --k)
            {
                arr[k] = arr[k - 1];
            }
            arr[j + 1] = temp;
        }
    }
}
```

5、**希尔排序** - 插入排序的改进版。为了减少数据的移动次数，在初始序列较大时取较大的步长，通常取序列长度的一半，此时只有两个元素比较，交换一次；之后步长依次减半直至步长为1，即为插入排序，由于此时序列已接近有序，故插入元素时数据移动的次数会相对较少，效率得到了提高。

时间复杂度：**通常认为是$O(N^{3/2})$，未验证**　　稳定性：**不稳定**

```
/*希尔排序*/
void shellSort(vector<int> &arr, int bgn, int end)
{
    for (int step = (end - bgn) / 2; step > 0; step /= 2)
    {
        for (int i = bgn; i < bgn + step; ++i)
        {
            /*
            * 以下，insertSort的变异
            */
            for (int j = i + step; j < end; j += step)
            {
                int k = j - step;
                for ( ; k >= i; k -= step)
                    if (arr[k] <= arr[j])
                        break;
                if (k != j - step)
                {
                    int tmp = arr[j];
                    for (int m = j; m > k + step; m -= step)
                        arr[m] = arr[m - step];
                    arr[k + step] = tmp;
                }
            }
```

```
25          }
26      }
27  }
```

6、**桶排序** - 实现线性排序，但当元素间值得大小有较大差距时会带来内存空间的较大浪费。首先，找出待排序列中得最大元素max，申请内存大小为max + 1的桶（数组）并初始化为0；然后，遍历排序数列，并依次将每个元素作为下标的桶元素值自增1；最后，遍历桶元素，并依次将值非0的元素下标值载入排序数列（桶元素>1表明有值大小相等的元素，此时依次将他们载入排序数列），遍历完成，排序数列便为有序数列。

时间复杂度：**O(x*N)**　　稳定性：**稳定**

```
/*桶排序*/
void bucketSort(vector<int> &arr)
{
    int max = getMaxValue(arr);
    int *pBuf = new int[max + 1];

    memset(pBuf, 0, (max + 1)*sizeof(int));
    for (auto const i : arr)
        ++pBuf[i];

    for (int i = 0, j = 0; i <= max; ++i)
    {
        while (pBuf[i]--)
            arr[j++] = i;
    }
    delete []pBuf;
}
```

7、**基数排序** - 桶排序的改进版，桶的大小固定为10，减少了内存空间的开销。首先，找出待排序列中得最大元素max，并依次按max的低位到高位对所有元素排序；桶元素10个元素的大小即为待排序数列元素对应数值为相等元素的个数，即每次遍历待排序数列，桶将其按对应数值位大小分为了10个层级，桶内元素值得和为待排序数列元素个数。

时间复杂度：**O(x*N)**　　稳定性：**稳定**

```
/*基数排序*/
//1. 计数排序，按整形数值单位进行排序
void countSort(vector<int> &arr, int exp)
{
    int bucket[10] = { 0 };
    int arrSize = arr.size();
    int *pTemp = new int[arrSize];
    memset(pTemp, 0, arrSize * sizeof(int));

    //统计单位exp各数值计数值
    for (auto const val : arr)
        ++bucket[(val / exp) % 10];

    //计数分层
    for (int i = 1; i < 10; ++i)
        bucket[i] += bucket[i - 1];

    //按exp位大小用数组arr元素填充pTemp
    for (int i = arrSize - 1; i >= 0; --i)
        pTemp[ --bucket[(arr[i] / exp) % 10] ] = arr[i];
/*bugs*/
#if 0
    //bug1: bucket各层次的计数值没遍历一次相应自减1
    for (auto const val : arr)
        pTemp[bucket[(val / exp) % 10] - 1] = val;
    //bug2: arr数组元素每次排序时，下标应从大到小遍历，否则无法实现排序
    for (auto const val : arr)
        pTemp[ --bucket[(val / exp) % 10] ] = val;
#endif
```

```
    //pTemp -> arr
    for (int i = 0; i < arrSize; ++i)
        arr[i] = pTemp[i];
    delete []pTemp;
}
//2. 合并各单位计数排序结果
void radixSort(vector<int> &arr)
{
    int max = getMaxValue(arr);
    for (int exp = 1; max / exp != 0; exp *= 10)
        countSort(arr, exp);
}
```

8、**归并排序** - 采用了分治和递归的思想，递归&分治-排序整个数列如同排序两个有序数列，依次执行这个过程直至排序末端的两个元素，再依次向上层输送排序好的两个子列进行排序直至整个数列有序（类比二叉树的思想，from down to up）。

时间复杂度：**O(NlogN)**　　稳定性：**稳定**

```
/*归并排序*/
//排序两个有序数列
void mergeSortInOrder(vector<int> &arr, int bgn, int mid, int end)
{
    int *pBuf = new int[end - bgn];
    int *pTemp = pBuf;
    int lindex = bgn;
    int rindex = mid;

    while ((lindex < mid) && (rindex < end))
        *(pTemp++) = (arr[lindex] < arr[rindex]) ? arr[lindex++] : arr[rindex++];

    while (lindex < mid)
        *pTemp++ = arr[lindex++];
    while (rindex < end)
        *pTemp++ = arr[rindex++];

    //pTemp -> arr
    pTemp = pBuf;
    for (int i = bgn; i < end; i++)
        arr[i] = *pTemp++;

    delete []pBuf;
}
//UpToDown To DownToUp
void mergeSort(vector<int> &arr, int bgn, int end)
{
    //数组arr空or仅有一个元素则退出
    if (bgn >= end - 1)
        return;

    int mid = (bgn + end) / 2;
    mergeSort(arr, bgn, mid);
    mergeSort(arr, mid, end);
    mergeSortInOrder(arr, bgn, mid, end);
}
```

9、**堆排序** - 堆排序的思想借助于二叉堆中的最大堆得以实现。首先，将待排序数列抽象为二叉树，并构造出最大堆；然后，依次将最大元素（即根节点元素）与待排序数列的最后一个元素交换（即二叉树最深层最右边的叶子结点元素）；每次遍历，刷新最后一个元素的位置（自减1），直至其与首元素相交，即完成排序。

时间复杂度：**O(NlogN)**　　稳定性：**不稳定**

```
/*堆排序*/
//根节点元素自顶向下移动到合适的位置以构成最大堆
void downToMaxHeap(vector<int> &arr, int bgn, int end)
{
```

```
    int child;
    int parent = bgn;

    /*假根节点向下移动至合适的位置 --整个堆排序的核心代码块*/
    while ((child = parent * 2 + 1) < end)
    {
        if ((child < end - 1) && (arr[child] < arr[child + 1]))
            ++child;    //右孩子节点
        if (arr[child] > arr[parent])
            mySwap(&arr[child], &arr[parent]);
        else
            break;
        parent = child;
    }
}
//将数组构造为最大堆
void buildMaxHeap(vector<int> &arr, int bgn, int end)
{
    if (bgn >= end - 1)
        return;

    int parent = end / 2 - 1;
    while (parent >= 0)
    {
        downToMaxHeap(arr, parent, end);
        --parent;
    }
}
//堆排序
void heapSort(vector<int> &arr, int bgn, int end)
{
    //构造最大堆
    buildMaxHeap(arr, bgn, end);

    while (end > 1)
    {
        mySwap(&arr[0], &arr[--end]);
        downToMaxHeap(arr, 0, end);
    }
}
```

参考博文：http://www.cnblogs.com/skywang12345/p/3603935.html

点滴记录 点滴成长 未雨绸缪 不乱于心

分类：编程技法

标签：Algorithm

好文要顶 关注我 收藏该文

Glory_D
关注 - 8
粉丝 - 3
+加关注

3 推荐 0 反对

« 上一篇：Window下通过CuteFTP与Linux虚拟机连接失败的原因总结及解决方法
» 下一篇：J-Link固件烧录以及使用J-Flash向arm硬件板下载固件程序

posted @ 2017-11-23 16:54 Glory_D 阅读(41685) 评论(8) 编辑 收藏

Post Comment

#1楼 2019-09-06 14:05 | 猫尼玛

大哥你这第一个getMaxValue就有问题啊，万一数组里都是负的咋整，你得吧max初始化为数组的第一个元素才对呀亲~

支持(0) 反对(

#2楼 2019-09-08 14:05 | Y骑绝尘

大哥，你冒泡排序内部for循环条件应该是j<=i，要不然你肯定漏掉数组最后一个值没排序，很明显的错误