第十二讲

Sorting

薛浩

2023 年 5 月 16 日

www.stickmind.com

今日话题

话题 1：编程基础 初学编程的新手，一般应该熟练使用函数和库处理字符串相关的编程任务。

话题 2：抽象数据类型的使用 在尝试实现抽象数据类型之前，应该先熟练使用这些工具解决问题。

话题 3：递归和算法分析 递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

话题 4：类和内存管理 使用 C++ 实现数据抽象之前，应先学习 C++ 的内存机制。

话题 5：常见数据结构和算法 在熟练使用抽象数据类型解决常见问题之后，学习如何实现它们是一件很自然的事情。

话题 3：递归和算法分析

递归是一种强有力的思想，一旦掌握就可以解决很多看起来非常难的问题。

- 递归过程
- 算法分析
- 递归回溯
- 排序算法

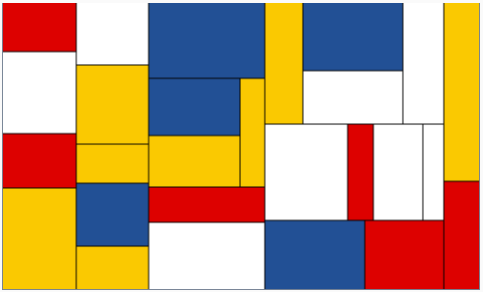


Figure 1: 递归艺术

递归在排序问题中的应用

目录

复习：算法分析

算法分析

递归思维结合分治、回溯等策略经常会得到更高效的算法，相比迭代设计过程，效率的提升有时甚至达到成千上万倍的数量级。

为了评估不同算法的效率，可以使用多种算法分析手段，常见的有计时法、统计操作数、大 O 分析等。

体会算法分析重要性的最好方法就是考虑一个问题域，其中最有趣的问题就是**排序**（sorting）问题。

复习：算法分析

方法 1：作业 1 中，我们通过秒表计时来判断程序执行的快慢。

方法 2：如果每次操作都需要固定的时间，那么统计整体操作的次数也可以评估程序的效率。

```
double averageOf(const Vector<int> &vec) {
    double total = 0.0;
    for (int i = 0; i < vec.size(); i++) {
        total += vec[i];
    }
    return total / vec.size();
}
```

方法 3：大 O 表示法是表示时间复杂度最常见的方式，旨在提供定性评估。

练习：findInVector

```
bool findInVector(const Vector<int>& vec, int key) {
    if (vec.size() == 1) {
        return (key == vec[0]) ? true : false;
    } else {
        if (vec[0] == key) {
            return true;
        } else {
            Vector<int> rest = vec.subList(1);
            return findInVector(rest, key);
        }
    }
}
```

练习：findInVectorDivide

```
bool findInVectorDivide(const Vector<int>& vec, int key) {
    if (vec.size() == 1) {
        return (key == vec[0]) ? true : false;
    }

    int mid = vec.size() / 2;
    Vector<int> firstPart = vec.subList(0, mid);
    Vector<int> secondPart = vec.subList(mid);

    bool result1 = findInVectorDivide(firstPart, key);
    bool result2 = findInVectorDivide(secondPart, key);
    return (result1 || result2) ? true : false;
}
```

排序 Sorting

排序 Sorting

在计算机科学家研究的所有算法问题中，具有最广泛实际影响的就是排序问题，即按顺序排列序列中的元素。

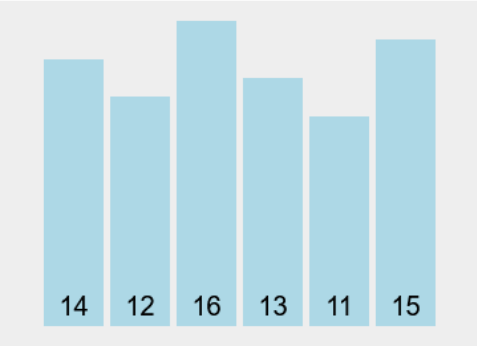
例如，按字母顺序排列电话簿、按目录号排列图书馆记录、按邮政编码整理邮件时，就会出现排序问题。

有许多算法可以用来对数组进行排序。由于这些算法的效率差异很大，因此选择一个好的算法至关重要，特别是在处理大型数组或向量的情况下。

练习：Sorting

如何按从小到大顺序排列以下序列：

```
{14, 12, 16, 13, 11, 15}
```



选择排序 Selection Sorting

选择排序 Selection Sorting

选择排序（Selection Sorting）是所有排序算法中，最容易描述的。可以参考相关在线网站，动画展示这个过程。

```
void sort(Vector<int> & vec) {
  int n = vec.size();
  for (int lh = 0; lh < vec.size(); lh++) {
      int rh = lh;
      for (int i = lh + 1; i < vec.size(); i++) {
        if (vec[i] < vec[rh]) rh = i;
      }
      int temp = vec[lh];
      vec[lh] = vec[rh];
      vec[rh] = temp;
  }
}
```

选择排序 Selection Sorting

为了便于算法分析，可以使用辅助函数改写成以下形式：

```
void sort(Vector<int> & vec) {
  for (int lh = 0; lh < vec.size(); lh++) {
    int rh = findSmallest(vec, lh, vec.size() - 1);
    swap(vec[lh], vec[rh]);
  }
}

int findSmallest(Vector<int> & vec, int p1, int p2) { ... }
void swap(int & x, int & y) { ... }
```

选择排序 Selection Sorting

```
int findSmallest(Vector<int> & vec, int p1, int p2) {
  int smallestIndex = p1;
  for ( int i = p1 + 1 ; i <= p2 ; i++ ) {
    if (vec[i] < vec[smallestIndex])
      smallestIndex = i;
  }
  return smallestIndex;
}

void swap(int & x, int & y) {
  int temp = x;
  x = y;
  y = temp;
}
```

归并排序 Merge Sorting

归并排序 Merge Sorting

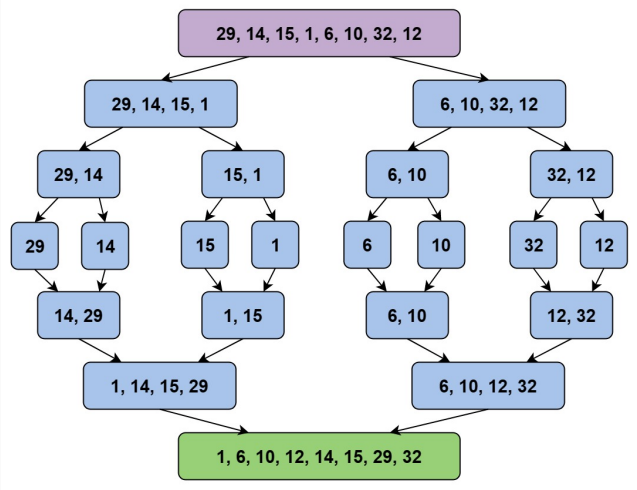
归并排序（Merge Sorting）一种高效、通用、基于比较的排序算法，属于分治算法，由约翰・冯・诺依曼于 1945 年发明。

基于递归的算法如下：

基本情况 如果是空或单元素序列，则已排序。

递归分解 将序列拆分成两半，降低问题规模；
递归处理，分别排序两个部分；
整理两个部分的排序结果，进行归并。

归并排序 Merge Sorting



归并排序 Merge Sorting

```
void merge(Vector<int>& vec, Vector<int>& v1, Vector<int>& v2) {
    int n1 = v1.size(), n2 = v2.size();
    int p1 = 0, p2 = 0;
    while (p1 < n1 && p2 < n2) {
        if (v1[p1] < v2[p2]) {
            vec.add(v1[p1++]);
        } else {
            vec.add(v2[p2++]);
        }
    }
    while (p1 < n1) vec.add(v1[p1++]);
    while (p2 < n2) vec.add(v2[p2++]);
}
```

归并排序 Merge Sorting

```
void mergeSort(Vector<int>& vec) {
    int n = vec.size();
    if (n <= 1) return;
    Vector<int> v1, v2;
    for (int i = 0; i < n; i++) {
        if (i < n / 2) {
            v1.add(vec[i]);
        } else {
            v2.add(vec[i]);
        }
    }
    mergeSort(v1); mergeSort(v2);
    vec.clear(); merge(vec, v1, v2);
}
```

快速排序 Quick Sorting

快速排序 Quick Sorting

快速排序（Quick Sorting）最早由东尼·霍尔提出，相比其他排序算法，其内部循环可以充分利用硬件缓存，从而达到更快的效率。

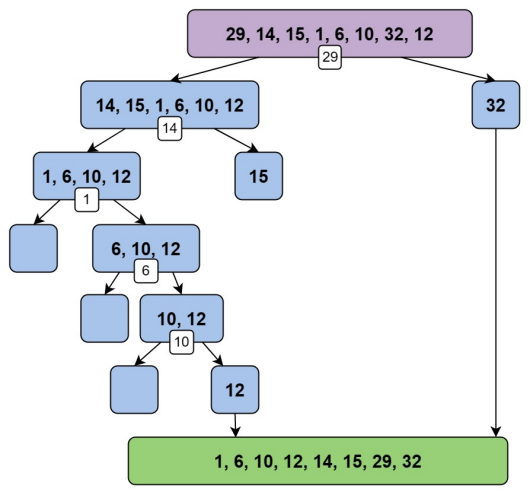
快速排序的算法大致如下：

分割 选择一个**基准**将序列拆分成三个部分；
小于基准的元素，放在前面；
等于基准的元素，放在中间；
大于基准的元素，放在后边。

递归 递归地将小于和大于基准的两个序列再次按同样的逻辑排序。

连接 经递归处理后，重新连接以上三个部分序列即完成排序。

快速排序 Quick Sorting



快速排序 Quick Sorting

```
int partition(Vector<int>& vec, int start, int finish) {
    int pivot = vec[start];
    int lh = start + 1, rh = finish;
    while (true) {
        while (lh < rh && vec[rh] >= pivot) rh--;
        while (lh < rh && vec[lh] < pivot) lh++;
        if (lh == rh) break;
        int tmp = vec[lh]; vec[lh] = vec[rh]; vec[rh] = tmp;
    }
    if (vec[lh] >= pivot) return start;
    vec[start] = vec[lh];
    vec[lh] = pivot;
    return lh;
}
```

快速排序 Quick Sorting

```
void quickSortRec(Vector<int>& vec, int start, int finish) {
    if (start >= finish)
        return;
    int boundary = partition(vec, start, finish);
    quickSortRec(vec, start, boundary - 1);
    quickSortRec(vec, boundary + 1, finish);
}

void quickSort(Vector<int>& vec) {
    quickSortRec(vec, 0, vec.size() - 1);
}
```

递归在排序问题中的应用