数据结构

第二十一讲 分配与归并排序 & 课程总结

孙猛

http://www.math.pku.edu.cn/teachers/sunm

2017年12月29日

课程内容

- 分配排序
- 归并排序
- 课程总结

分配排序

- 基本思想：分配排序是把排序码分解成若干部分，然后通过对各个部分排序码的分别排序，最终实现对整个排序码的排序。
- 定义：假设文件F里有n个记录，$F=(R_0,R_1,\ldots,R_{n-1})$，且记录 R_i 的排序码包含 d 个部分 $(k_i^0,k_i^1,\ldots,k_i^{d-1})$，文件 F 对排序码 $(k^0,k^1,\ldots,k^{d-1})$ 有序是指：对于文件中的任意两个记录 R_i 和 R_j $(0\le i\le j\le n-1)$，其排序码都满足字典序：

$$(k_i^0, k_i^1,\ldots, k_i^{d-1}) \le (k_j^0, k_j^1,\ldots, k_j^{d-1})$$

- 这里k_i^0 称为最高位排序码，k_i^{d-1} 称为最低位排序码。

实现分配排序的第一种方法

- 从最高位排序码k^0开始，逐个针对各个排序码排序，这种方法称为高位优先法。
  - 采用高位优先排序，按一位排序码排序，就是将文件分割成若干子文件。要完成整个文件的排序，后续工作是做各子文件独立排序。
  - 这样逐层分割，d数目大时分许多层，数据组织比较麻烦。
- 例如扑克牌排序(花色和点数)：
  - 先按花色分成4组，
  - 然后每组按点数排序，
  - 最后把4组收集到一起。

实现分配排序的第二种方法

- 从最低位排序码k^{d-1}开始排序，称为低位优先法。
  - 低位优先排序不划分子文件，对每位排序码K^i工作时都以整个文件作为排序对象，通过若干次“分配”和“收集”实现。
- 例如扑克牌排序(花色和点数)：
  - 先按点数分13组，收集到一起；
  - 后按花色分4组，再收集到一起。
- 低位优先法的实现比高位优先法简单。
- 但是要求针对各 K^i 的排序必须用稳定的排序方法。

基数排序：一种低位优先法分配排序

- 把排序码看成 d 元组：

$$K_i = (K_i^0, K_i^1, \ldots, K_i^{d-1})$$

其中每个 K_i^j都是集合$\{C_0, C_1, \ldots, C_{r-1}\}$ 中的元素，这些元素满足：

$C_0 < C_1 < \ldots < C_{r-1}$

r 称为排序码的基数。

排序过程：

- 首先按 K_i^{d-1} 把记录从小到大分配到 r 个组里，后依次收集；
- 再按 K_i^{d-2} 把记录分配到 r 个组里，……；
- 如此反复d 遍，直到完成对 K_i^0 的分配、收集，便得到完全排好序的序列。

基数排序示例

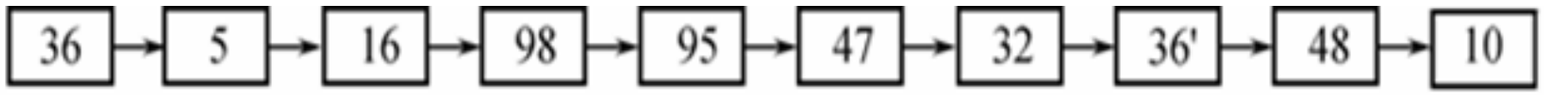
- 假定排序码为3位4进制数：
  - 301, 212, 031, 320, 122, 203, 113, 020, 333, 001, 100, 311
- 按最低位排序，得到4组：
  - 320, 020, 100 | 301, 031, 001, 311 | 212, 122 | 203, 113, 333
    - 收集：320, 020, 100, 301, 031, 001, 311, 212, 122, 203, 113, 333
- 按第二位分为4组：
  - 100, 301, 001, 203 | 311, 212, 113 | 320, 020, 122 | 031, 333
  - 收集：100, 301, 001, 203, 311, 212, 113, 320, 020, 122, 031, 333
- 按最高位分为4组：
  - 001, 020, 031 | 100, 113, 122 | 203, 212 | 301, 311, 320, 333
  - 收集：001, 020, 031, 100, 113, 122, 203, 212, 301, 311, 320, 333

存储结构

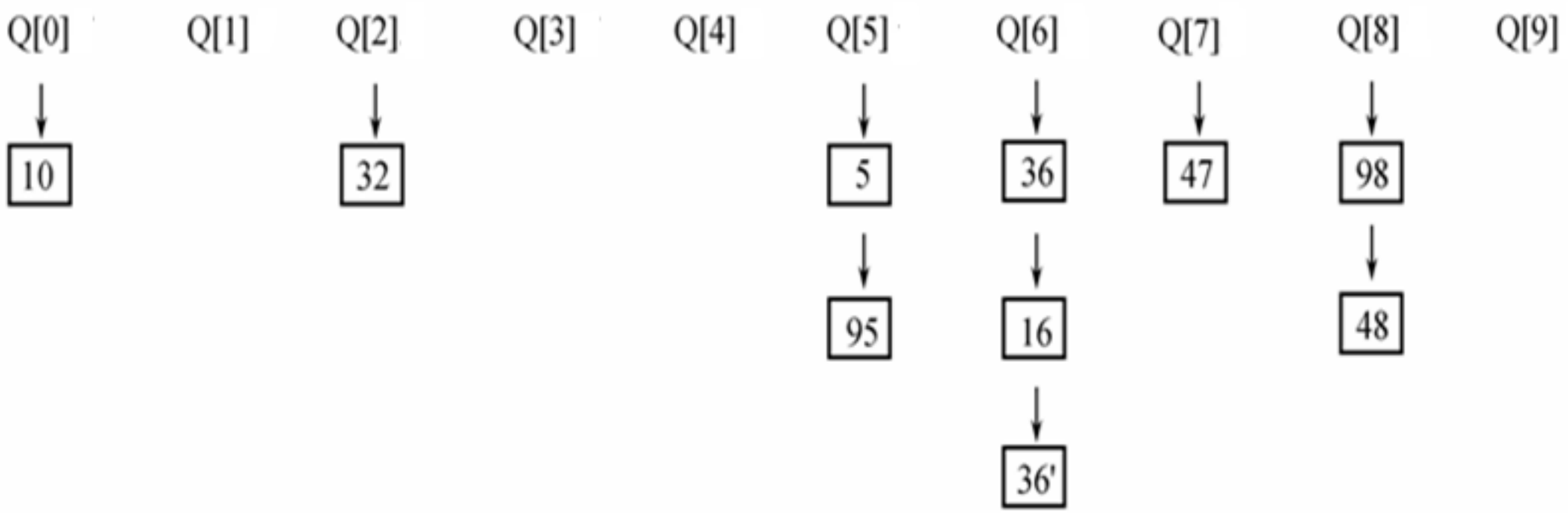
- 假设r是排序码的基数，d是排序码的位数，每位的类型是KeyType。
- 待排序的文件采用带表头结点的链接表示，类型为RadixList；
- 为了便于实现记录的分配和收集，建立r个链接表（队列），每个队列的类型为Queue。每次分配完后把所有的表连接起来。

排序过程

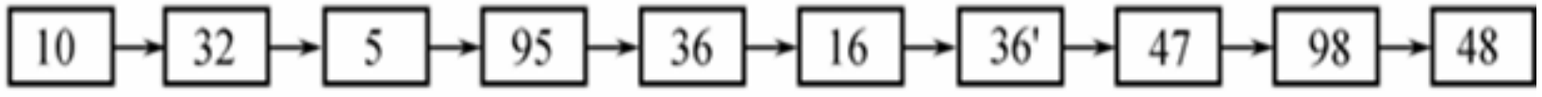
(1) 初始状态



(2) 第一趟分配后

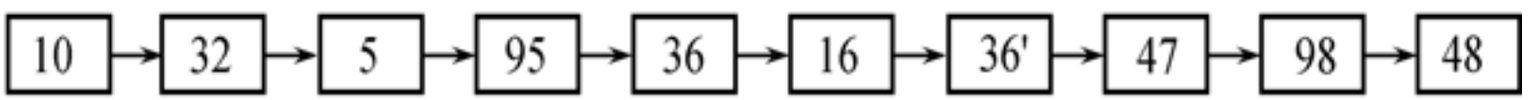


(3) 第一趟收集后

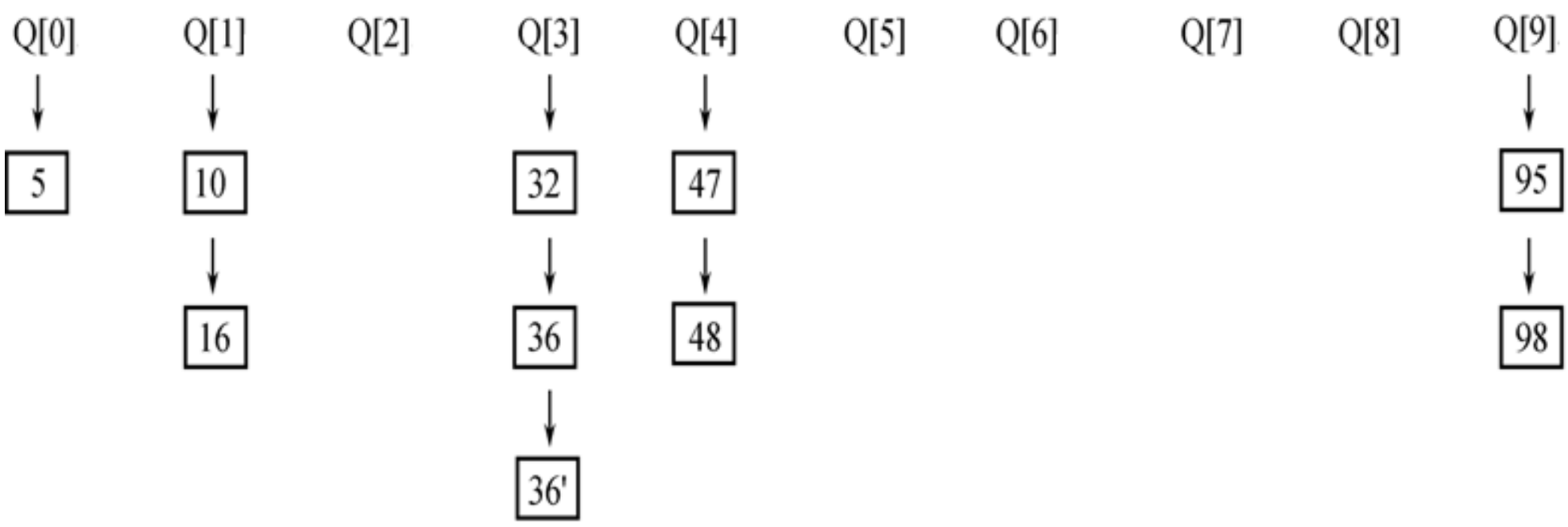


排序过程

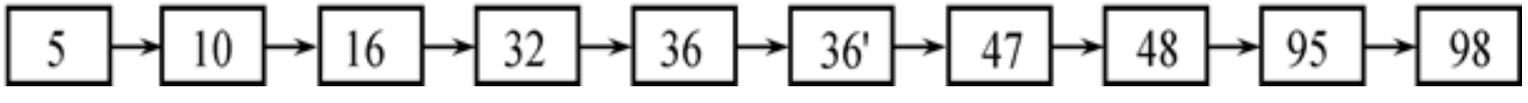
(3) 第一趟收集后



(4) 第二趟分配后



(5) 第二趟收集后



存储结构定义

```
struct Node
typedef struct Node * PNode;
struct Node {
    KeyType key[D];                          /* 排序码是数组 */
    DataType info;
    PNode  next;
};
typedef struct  {
    PNode f;                                 /* 队列的头指针 */
    PNode e;                                 /* 队列的尾指针 */
} Queue;                                     /* 用队列表示各个子序列 */

typedef struct Node *RadixList;              /* 待排序文件类型 */
```

基数排序算法

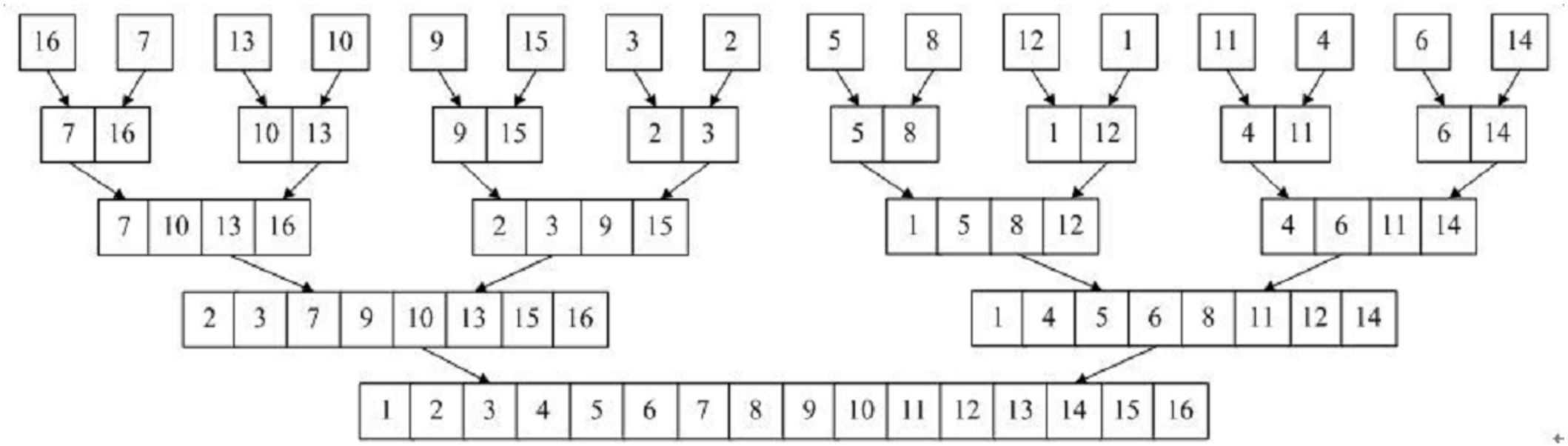
```
void radixSort(RadixList * plist, int d, int r) {
    int i,j,k;    Pnode p, head = (*plist)->next;
    Queue queue[r];                                          /* 队列数组 */
    for(j = d-1; j >= 0; j--) {                              /* 进行d次分配和收集*/
        for(i = 0; i < r; i++)queue[i].f = queue[i].e = NULL;   /* 清空队列 */
        for (p = head; p != NULL; p = p->next) {
            k = p->key[j];                                   /* 按排序码的第j个分量分配*/
            if (queue[k].f == NULL) queue[k].f = p;          /*队列为空设队头指针 */
            else (queue[k].e)->next = p;                     /* 接到队列k 尾*/
            queue[k].e = p;
        }
        for (i = 0; queue[i].f == NULL; i++) ;               /*找到第一个非空队列*/
        p = queue[i].e;  head = queue[i].f;                  /* head为收集链表的头指针*/
        for(i++; i < r; i++)                                 /* 收集其他非空队列 */
          if (queue[i].f != NULL) {p->next = queue[i].f;  p = queue[i].e;  }
        p->next = NULL;
    }
    (*plist)->next = head;
}
```

基数排序算法分析

- 基数排序算法中，时间耗费主要在修改链接上；
- 一趟排序的时间为$O(r+n)$，总共要进行d趟排序，基数排序的时间复杂度是$T(n) = O(d*(r+n))$；
- 采用链表排序方法只修改链接指针，操作效率不受记录的信息量大小的影响;
- 排序中增加了一个next字段(链表指针), 还增加了一个queue 数组，故辅助空间为$S(n)=O(n+r)$；
- 基数排序是稳定的。

归并排序

- 归并排序的思想: 把待排序的文件分成若干个子文件，先将每个子文件内的记录排序，再将已排序的子文件合并，逐步得到完全排序的文件。



归并排序

- 归并排序的思想：把待排序的文件分成若干个子文件，先将每个子文件内的记录排序，再将已排序的子文件合并，逐步得到完全排序的文件。
- 二路归并排序
  - 初始时，可以把待排序序列的 n 个记录看成 n 个有序子序列，每个子序列的长度为 1；
  - 把当时的有序子序列两两归并，可得到长度加倍，数目减少一半的一组有序子序列；
  - 重复做子序列的两两归并，最终得到长度为 n 的有序序列。
- 类似还可考虑三路归并或多路归并。

二路归并排序示例

- 初始序列为25, 57, 48, 37, 12, 82, 75, 29, 16，用二路归并排序法完成序列的排序:

初始序列: 25 57 48 37 12 82 75 29 16

第一趟归并后: 25 57 37 48 12 82 29 75 16

第二趟归并后: 25 37 48 57 12 29 75 82 16

第三趟归并后: 12 25 29 37 48 57 75 82 16

第四趟归并后: 12 16 25 29 37 48 57 75 82

排序后的结果为：12，16，25，29，37，48，57，75，82

二路归并排序

- 数组归并排序分三层实现（用一个同样大的辅助数组）：

  1) 最下层：实现数组中相邻的两组有序序列的归并，归并结果存入另一数组里的相同位置；

  2) 第二层：基于1)实现对整个数组中各对有序序列的归并，所有归并结果存入另一数组；

  3) 最高层：在两个数组之间往复进行操作2)；完成一遍归并后交换两数组的地位并重复这一操作，直至数组里只有一个有序序列时排序完成。

二路归并排序示例

- 初始序列为25, 57, 48, 37, 12, 82, 75, 29, 16，用二路归并排序法完成序列的排序:

初始序列: 25 57 48 37 12 82 75 29 16

第一趟归并后: 25 57 37 48 12 82 29 75 16

第二趟归并后: 25 37 48 57 12 29 75 82 16

第三趟归并后: 12 25 29 37 48 57 75 82 16

第四趟归并后: 12 16 25 29 37 48 57 75 82

排序后的结果为：12，16，25，29，37，48，57，75，82

两组归并算法

```
/*  r[low]到r [m]和r[m+1]到r[high]是两个有序段，把它们归并到 r1[low]
    到r1[high] 一段中 */
void merge (RecordNode  *r, RecordNode  *r1,  int low,  int m, int
    high ) {
    int i = low, j = m + 1, k = low;
    while ( i <= m && j <= high ) { /* 反复复制两段首记录中较小的一个 */
        if (r[i].key <= r[j].key)  r1[k++] = r[i++];
        else r1[k++] = r[j++];
    }
    while (i <= m)                                  /* 复制第一段剩余记录 */
        r1[k++] = r[i++];
    while (j <= high)                               /* 复制第二段剩余记录 */
        r1[k++] = r[j++];
}
```

稳定的?!

一趟归并算法

```
/* 对 r 做一趟归并, 结果放入 r1; 当前有序段长 length */
void mergePass(RecordNode *r, RecordNode *r1, int n, int
    length) {
    int i = 0, j;
    while(i + 2*length - 1 < n) {   /* 归并长length的两个子段*/
        merge(r, r1, i, i+length-1, i + 2*length - 1);
        i += 2*length;
    }
    if (i + length - 1 < n - 1)       /* 剩下两段，后段长度小于 length */
        merge(r, r1, i, i+length-1, n-1);
    else                              /* 剩下一段，将它复制到数组r1 */
        for(j = i; j < n; j++)  r1[j] = r[j];
}
```

二路归并算法

```
/*二路归并排序就是多次调用“一趟归并”过程, 每趟归并后有序文件的
    长度length扩大一倍.*/
void mergeSort(SortObject * pvector) {
    RecordNode record[MAXNUM];   int  length = 1;
    while (length < pvector->n) {
        /* 一趟归并，结果存入辅助数组record */
        mergePass(pvector->record, record, pvector->n, length);
        length *= 2;
        /* 另一趟归并，结果存回原位 */
        mergePass(record, pvector->record, pvector->n, length);
        length *= 2;
    }
}
```

二路归并算法分析

- **时间复杂度:** $T(n)=O(n*log_2n)$
  - 做了第 i 遍归并后，有序子序列的长度为 2^i，因此完成排序需要做不多于log_2n+1遍归并，每遍归并做$O(n)$次比较，总比较和移动次数都为$O(n*log_2n)$；
- **空间复杂度:** $S(n)=O(n)$
  - 和待排记录序列等量的辅助空间(空间开销大)
- **二路归并算法是稳定的:**
  - 这里关注了归并相同排序码的记录时的选择顺序。

外排序

- 若待排序的记录存在外存文件，由于数据量太大无法同时全部放入内存，就需要考虑适合这种情况的排序算法。
- 归并排序的特点是顺序处理一系列(排序)记录并顺序生成更长的排序序列，适合外存顺序访问效率高、随机访问效率低的特点，因此成为实现外排序算法的基础。

外排序

- 若待排序的记录存在外存文件，由于数据量太大无法同时全部放入内存，就需要考虑适合这种情况的排序算法。
- 归并排序的特点是顺序处理一系列(排序)记录并顺序生成更长的排序序列，适合外存顺序访问效率高、随机访问效率低的特点，因此成为实现外排序算法的基础。
- 最简单的外排序算法是照搬前面介绍的两路归并算法:
  - 准备：读入一组适当数量的页块，在内存完成页块组内记录排序，将一组页块写入一个临时已排序文件(也称为顺串)；
  - 归并：同时读入两个顺串，用两路归并方法生成一个更大的顺串。一遍归并使顺串的长度（包含的记录数）加大一倍，数量减少一半。
  - 反复归并，直至所有记录都在一个顺串里。

外存二路归并排序的问题

- 需要解决两个充分大的顺串（也可以看成是两个有序文件）的合并问题。

- 如何提高外排序的速度，减少对于外存的访问次数——外排序的核心问题。

- 如何减少访外时的定位时间，提高主机与外存的并行能力等。

排序小结

- 各种排序方法各有优缺点。排序算法之间的比较主要考虑以下几个方面：
  - 算法的时间复杂度
  - 算法的辅助空间与被排序对象的表示方式
  - 排序算法的稳定性
  - 算法结构的复杂性
  - 适合参加排序的数据的规模

排序方法比较

- 根据平均时间复杂度可知，Shell排序、堆排序、快速排序及归并排序速度较快，其它排序方法的速度较慢；一般情况下，从算法结构的简单性看，速度慢的排序算法比较简单、直接。Shell排序法，快速排序法、堆排序法及归并排序法可以看作是对某一种排序方法的改进，算法结构一般都比较复杂。

数据结构

第二十讲(2) 课程总结

孙猛

http://www.math.pku.edu.cn/teachers/sunm

2016年12月29日

器？术？道？

- 紫薇软剑，三十岁前所用，误伤义士不祥，悔恨无已，乃弃之深谷。
- 重剑无锋，大巧不工。四十岁前恃之横行天下。
- 四十岁之后不滞于物，草木竹石均可为剑。自此精进，渐入无剑胜有剑之境。

学习目的

- 理解数据结构和算法的概念
- 掌握设计数据结构和算法的主要原理和方法
- 比较不同数据结构和算法的特点，使用适当的数据结构和算法解决具体问题
- 掌握计算思维，提高使用计算机解决问题的能力

科学与科学思维

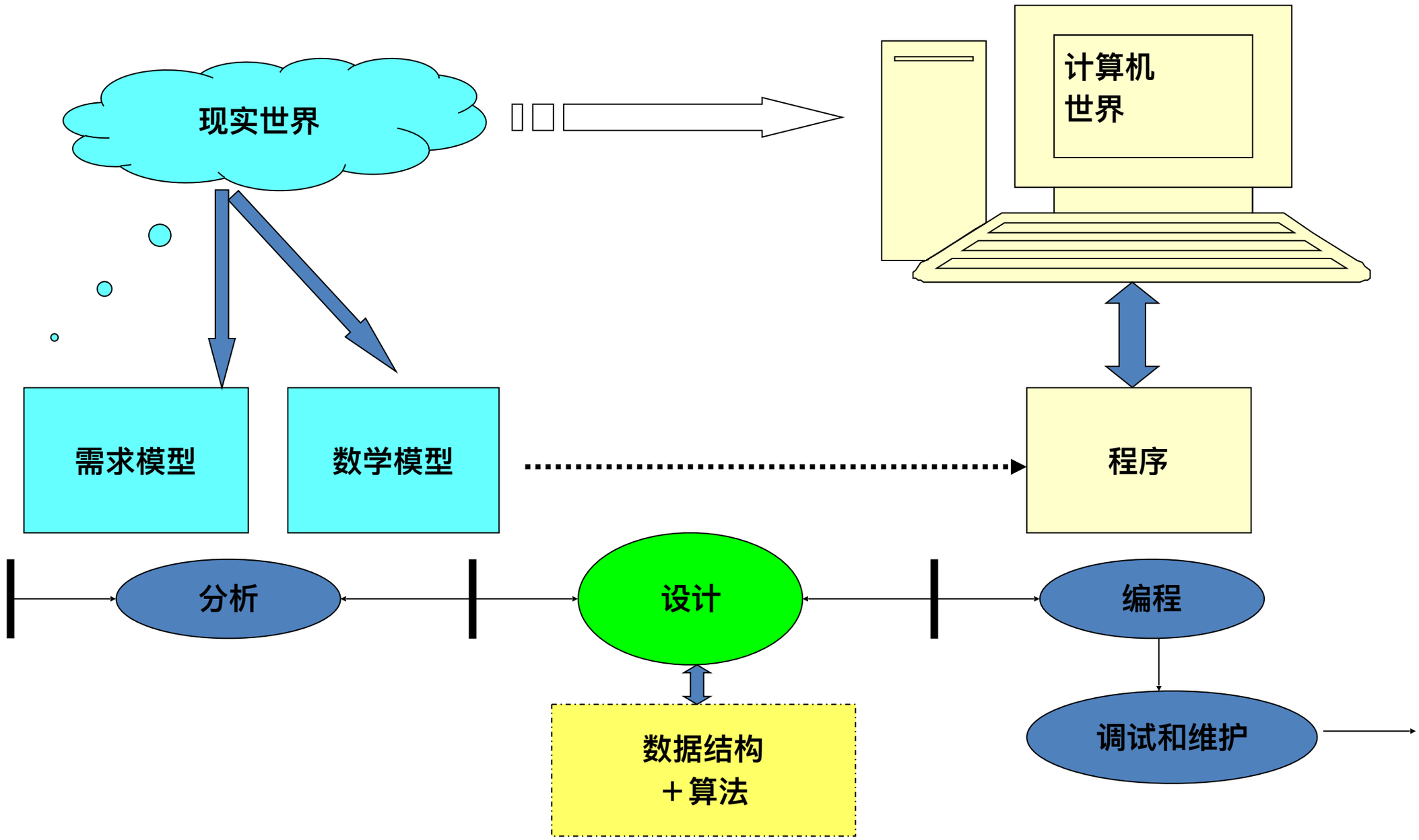
- 科学与思维
  ①科学就是整理事实，从中发现规律，作出结论。
  ②思维是高级的心理活动形式，人脑对信息的处理包括分析、抽象、综合、概括。
- 人类科学发现的三大支柱对应着三种思维方式：
  - 理论科学↔推理思维，以推理和演绎为特征，以数学学科为代表。
  - 实验科学↔实证思维，以观察和总结规律为特征，以物理学科为代表。
  - 计算科学↔计算思维，以设计和构造为特征，以计算机学科为代表。

计算思维的特征

- 概念化，不是程序化
- 基础的，不是机械的技能
- 人的，不是计算机的思维
- 思想，不是人造品
- 数学与工程思维的互补与融合，而不只是数学性思维
- 面向所有人，所有地方的思维，而不仅仅是计算机科学家的思维

——引自CMU周以真教授CACM2006年论文

用计算机求解问题



算法 + 数据结构 = 程序

- 程序就是在数据的某些特定的结构和表示的基础上对于算法的描述。
- 算法与数据结构是程序设计中相辅相成、不可分割的两个方面。

抽象数据类型

- **有一定行为（操作）的抽象（数学）类型。**

- **抽象出数据类型的使用要求，而把它的具体表示方式和运算的实现细节都隐藏起来。**
- **支持数据类型的实现与使用分离的原则，是一种十分有效的对问题进行抽象与分解的思维工具。**
- **允许独立地考虑数据类型的外部接口和内部的实现。**
- **对于系统的分解、设计、维护和修改十分有利，是面向对象技术与方法的主要理论基础。**

抽象数据类型

- 线性表
- 字符串
- 栈
- 队列
- 二叉树
- 树
- 优先队列
- 图
- 集合
- 字典

数据结构

- 传统的概念：数据结构是计算机中表示(存储)的、具有一定逻辑关系和行为特征的一组数据

- 根据面向对象的观点：数据结构是抽象数据类型的物理实现。

- 主要解决两个问题：
  如何具体表示抽象数据类型中的数学模型；
  如何给出抽象数据类型中需要操作的实现。

数据结构的三个方面

- 逻辑结构（数学模型）
  - 指数学模型(集合，表，树和图等)中的基本元素（结点）之间的相互关系（在抽象数据类型中这些关系隐含在数学名称中）。
  - 描述方式：B=<K,R>，K是结点的有穷集合，R是K上的一个关系。
- 存储结构（物理结构）
  - 指数学模型的具体表示方式，包括结点的表示和关系的表示。
  - 数据的逻辑结构在计算机存储器中的映射(或表示)
- 操作（行为）
  - 指抽象数据类型关心的各种行为在不同的存储结构上的具体实现(算法或程序)

数据结构

- 线性表
  - 链表、顺序表
- 字符串
  - 链接表示、顺序表示
- 栈
  - 链接栈、顺序栈
- 队列
  - 链接队列、顺序队列
- 二叉树
  - 链接表示、顺序表示、线索二叉树
- 树
  - 子结点表示、父结点表示、长子兄弟表示、子表表示
- 图
  - 邻接表表示、邻接矩阵表示
- 集合
  - 单链表表示、位向量表示
- 字典
  - 顺序表示、散列表示、索引表示、二叉排序树、平衡二叉排序树、最佳二叉排序树、B树、B+树、多分树

算法

- 算法是由有穷规则构成（为解决某一类问题）的运算序列。
- 算法设计的方法
  - 贪心法
  - 分治法
  - 回溯法
  - 动态规划法
  - 分支界限法

贪心法

- 当追求的目标是一个问题的最优解时，设法把对整个问题的求解工作分成若干步骤来完成。在其中的每一个阶段都选择从局部看是最优的方案，以期望通过各阶段的局部最优选择达到整体的最优。
- 示例解着色问题时就是采用的贪心法。
- 贪心法实际上不能保证都成功地产生一个全局性最优解，但是通常可以得到一个可行的较优解。

分治法

- 把一个规模较大的问题分成两个或多个较小的与原问题相似的子问题。首先对子问题进行求解，然后设法把子问题的解合并起来，得出整个问题的解，即对问题分而治之。如果一个子问题的规模仍然比较大，不能很容易地求得解，就可以对这个子问题重复地应用分治策略。
- 二分法检索就是用分治策略的典型例子。

动态规划法

- 与分治法相似都是把一个大问题分解为若干较小的子问题，通过求解子问题而得到原问题的解。
- 不同点是：
  - 分治法每次分解的子问题数目比较少，子问题之间界限清楚，处理的过程通常是自顶向下进行；
  - 动态规划法分解的子问题可能比较多，而且子问题相互包含，为了重用已经计算的结果，要把计算的中间结果全部保存起来，通常是自底向上进行。

回溯法

- 有一些问题，需要通过彻底搜索所有可能情况寻找一个满足某些预定条件的最优解。由于彻底搜索的运算量通常非常大，所以采取一步一步向前试探，当有多种选择时可以任意选择一种，只要目前可行就继续向前，一旦发现问题或失败就后退，回到上一步重新选择，借助于回溯技巧和中间增加判断，这样常常可以大大地减少搜索时间。

- 常见的迷宫问题以及八皇后问题都可以用回溯方法来解决

分支界限法

- 与回溯法相似，也是一种在表示问题解空间的树上进行系统搜索的方法。
- 所不同的是，回溯法使用了深度优先策略，而分枝界限法一般采用广度优先策略或者采用最大收益（或最小损耗）策略，并且利用最优解属性的的上下界来控制搜索的分枝。

不同算法的应用

- 对于某一问题来说，能用不同的设计技巧得到不同的算法。0/1背包问题就是一个典型的例子。
- 一个算法中，也可以结合使用几种算法设计技巧，如快速排序算法，就是分治和回溯的技巧的结合，分治的技巧是主要的。

检索／查找

- 集合和字典
  - 顺序、链接、散列、索引
- 线性表
  - 顺序 / 二分法
- 字符串
  - 模式匹配
- 二叉树周游
  - 广度优先
  - 深度优先（先根、对称、后根）
- 树周游
  - 广度优先
  - 深度优先（先根、后根）
- 树林周游
  - 后根、先根
- 图周游
  - 广度优先
  - 深度优先

排序算法

- 插入排序
  - 直接插入排序
  - 二分法插入排序
  - Shell排序
- 选择排序
  - 堆排序
  - 直接选择排序
- 交换排序
  - 快速排序
  - 起泡排序
- 分配排序
  - 基数排序
- 归并排序
  - 二路归并

其它重要算法

- KMP匹配算法
- Huffman 算法、Huffman 树和Huffman 编码
- 二叉排序树插入和删除
- AVL树插入删除（平衡恢复）
- B树和B+树的插入删除（结点分裂和合并，方法和原则）
- 图的基本算法
  - 最小生成树（Prim算法和Kruskal算法，解决同样问题）
  - 最短路径（Dijkstra算法和Floyd算法，解决不同问题）
  - 拓扑排序，关键路径

算法要求

- 对于算法功能和过程的理解，分为三个层次：
  1. 理解算法所解决的问题和算法的基本思想，
  2. 了解算法的基本过程（能自己按算法的过程处理具体实例，得出相应的解。自己操作）
  3. 清楚地知道算法的实现，包括算法实现中需要（可能用）的辅助数据结构，以及如何用程序语言严格描述。
- 此外，需要理解算法的性质（优劣、比较）
- 主要的算法性质：时间复杂性和空间复杂性

考试题目形式

- 基本概念
  - 与基本定义和性质有关的问题（填空、选择、判断、简答）
  - 手工完成一些操作，对具体实例实现算法过程
  - 画图，说明一些数据结构，或一些操作的过程和结果
  - 根据定义和性质完成某些计算或证明
  - 完成某些分析和比较工作
- 算法及程序题：数据结构描述和算法实现，在试卷上写出程序，包括程序填空
  - 应给出适当解释，尽可能写清楚，使改卷时容易辨认
- 综合题，包含上面两类题目的要素

考试之后……

- 《程序设计技术与方法》、《软件形式化方法》
- 出国希望我写推荐信的同学联系我之前请先阅读下面的说明：

  http://www.math.pku.edu.cn/teachers/sunm/goabroad.html
- 欢迎有意向跟我做本科生科研的同学和我联系。


期末考试：
1月4日上午8:30-10:30,
三教203
Merry Christmas
WINTER