# 算法设计与分析期中复习



ps:大部分内容不是自己写的，对一位学长or学姐的复习笔记做了一些补充

## 证明正确性

肯定用**数学归纳法**，我们所见的大部分题目都是使用数学归纳法。

**算法1：**

```
EUCLID(a,b):
if b==0 then return a;
else return EUCLID(b,a mod b)
```

**定理1.1**

EUCLID算法是正确的（数学归纳法证明）

对b进行归纳，假设gcd(a,k)对于k<n恒成立

EUCLID(a,n)=EUCLID(n,a mod n)=gcd(n, a mod n) = gcd(a,n)//由最大公约数性质

#### 性能指标

最坏情况时间复杂度： W(n)

平均情况时间复杂度：$A(n)=\sum_{I\in D_n}Pr(I)\cdot f(I)$，$I$是输入，$f(I)$是该输入的具体时间复杂度

### 算法中的数学

#### 渐进增长率

$f(n)=O(g(n))$ iff $\lim_{n\to\infty}\frac{f(n)}{g(n)}=c<\infty, c>0$ $f(n)=o(g(n))$ iff $\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$

两个都说明了f(n)增长率不如g(n)，但$o(g(n))$强调f与g间存在实质性的差距（更不如g了）
$f(n)=\Theta(g(n))$ iff $\lim_{n\to\infty}\frac{f(n)}{g(n)}=c, 0<c<\infty$ $f(n)=\Theta(g(n))$ iff $f(n)=O(g(n))\wedge f(n)=\Omega(g(n))$ 说明f和g同级
$f(n)=\Omega(g(n))$ iff $\lim_{n\to\infty}\frac{f(n)}{g(n)}=c>0, c$可以为$\infty$ $f(n)=\omega(g(n))$ iff $\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$ 两个都说明了f(n)增长率优于g(n)，但$\omega(g(n))$强调f与g间存在实质性的差距（更胜于g了）

### 求解分治与递归

#### 替换法

（期中考过）

利用数学归纳法，归纳证明$T(n)$的复杂度也小于等于$cO(n)$，$c$是某常数。

例如，求$T(n)=2T(n/2)+n$，猜测为$O(n\log n)$，步骤如下：

1. 假设对于小于$n$的参数都成立
2. 证明$n$也成立，如：

$$T(n)=2T(n/2)+n\ \le\quad 2c\frac{n}{2}\log\frac{n}{2}+n\ =cn\log\frac{n}{2}-c'n+n\ \le\quad cn\log n(c\ge 1)$$

#### 递归树

用不上

如果一定要用就点了吧

#### 主定理

放置几道例题在此：

**蛮力算法**

憨憨查找、选择排序、插入排序

~~狗都会~~

（不是我写的哈哈

插入排序在元素有序的情况下时间复杂度比较低（省去了大量的移动操作

插排可以通过二分插入来改进一下比较的次数，但是不能改进移动的次数

$$O(n^2)$$

## 分治 in 排序

**快速排序**

**算法7：PARTITION(A,p,r)**

```
PARTITION(A,p,r)
pivot := A[r]
i:=p-1
for j:= p to r-1 do:
    if A[i] < pivot:
        i:=i+1;
        SWAP(A[i],A[j])
SWAP(A[i+1],A[r])
return i + 1
```

**算法8：QUICK-SORT(A,p,r)**

```
QUICK-SORT(A,p,r)
if p < r then
    q = PARTITION(A,p,r);
    QUICK-SORT(A,p,q-1);
    QUICK-SORT(A,q+1,r);
```

**归并排序**

$W(n) = 2W(\frac{n}{2}) + O(n)$

**算法9：MERGE(A,p,q,r)**

很重要的过程

**算法10：MERGE-SORT(A,p,r)**

```
MERGE-SORT(A,p,r)
if p < r then
    q = (p+r)/2;
    MERGE-SORT(A,p,q);
    MERGE-SORT(A,q+1,r);
    MERGE(A,p,q,r);
```

**决策树**

引入决策树，说明算法结果需要一步一步走到叶节点，从而证明，比较排序的最坏情况时间复杂度的下界：$\Omega(n \log n)$

**逆序对及计数**

计算逆序对数，可以在归并排序中顺便完成

```
long long merge_count(long long array[], long long start, long long end)
{
    if (start == end)
        return 0;
    long long mid = (start + end) / 2;
    long long lcount = merge_count(array, start, mid);
    long long rcount = merge_count(array, mid + 1, end);
    long long p1 = mid, p2 = end;
    long long* copyarray = new long long[end - start + 1];
    long long copy_index = end - start;
    long long count = 0;
    while (p1 >= start and p2 >= mid + 1)
        if (array[p1] > array[p2])
            count += p2 - mid, copyarray[copy_index--] = array[p1--];
        else
            copyarray[copy_index--] = array[p2--];
    while (p1 >= start)
        copyarray[copy_index--] = array[p1--];
    while(p2 >= mid+1)
```

```
        copyarray[copy_index--] = array[p2--];
    for (long long i = 0; i < end - start + 1; i++)
        array[start + i] = copyarray[i];
    return lcount + rcount + count;
}
```

### 分治例子

**整数乘法**

长度都为$n$的$xy$相乘，直接计算复杂度为$O(n^2)$ $x = x_1 \cdot 2^{n/2} + x_0, y = y_1 \cdot 2^{n/2} + y_0$ 那么计算变为：
$xy = (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \qquad = x_1y_1 \cdot 2^n + (x_1y_0 + x_0y_1) \cdot 2^{n/2} + x_0y_0 \ = x_1y_1 \cdot 2^n + [(x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_0y_0] +$
将问题分解为$x_1y_1$ $(x_1 + x_0)(y_1 + y_0)$ $x_0y_0$三个子问题 $W(n) \leq 3W(\frac{n}{2}) + O(n)$ 不知道为什么是小于等于

根据主定理 $W(n) = O(n^{\log_2 3})$

**检测异类**

期中考过

列出所有情况组合，每次都只做降低规模至一半的操作 $W(n) \leq \frac{n}{2} + \frac{n}{2^2} + \cdots$ $W(n) = O(n)$

## $O(\log)$时间算法

**二分查找**

有序下二分查找、峰值查找、$\sqrt{N}$计算

**红黑树**

红黑树性质如下：

- 节点颜色只有红色或黑色
- 根节点必黑，叶节点必黑
- 节点若有子节点，必有2子；或者节点完全无子节点
- 红色节点不能连续出现
- 所有外部节点的黑色深度（到根路径上除根的黑色节点数）相等，称为黑色高度

准红黑树即根节点是红色，但其他性质都满足

**不存在$ARB_0$**

对于$h \geq 1$，红黑树$RB_h$左右子树分别为$RB_{h-1}$或$ARB_h$

对于$h \geq 1$，准红黑树$ARB_h$左右子树都为$RB_{h-1}$

**假设$T$为一个有$n$个内部节点的红黑树，则红黑树的普通高度不超过$2\log(n+1)$，基于红黑树的查找代价为$O(\log n)$**

## $O(n)$线性选择k阶元素

> 想要找到阶为k的元素，最笨的方法是每次$O(n)$找最小（或最大）并取出，找$k$次即可，用时$O(kn)$

**期望情况**

期望下，使用快速排序的Partition每次规模减半，可在$O(n)$内找到

```
Func Partition(A[], low, high)
    pivot = A[low]
    while low < high:
        while low < high && A[high] >= pivot:
            high--
        A[low] = A[high]
        while low < high && A[low] <= pivot:
            low++
        A[high] = A[low]
    A[low] = pivot
    return low
```

一顿期望的数学计算后，反正是$O(n)$

**算法11：SELECT-ELINEAR(A,p,r,k)**

**最坏情况**

考虑最坏情况，每次都精准地选出最小或最大数作为基准，那么每次规模只$-1$，那么$T(n) = T(n-1) + O(n)$，退化成$O(n^2)$

通过将数据分组，用选出基准**组**的方式来避免过于不平衡。已知，5个一组最好（期中考过，必不可能再考）

**算法12：SELECT-WLINEAR**

1. 所有元素分5组，凑不齐一组的元素暂放（也可按课本的分为一组）
2. 找出每组中位数，共$\lfloor \frac{n}{5} \rfloor$个
3. 对这$\lfloor \frac{n}{5} \rfloor$个中位数递归地使用SELECT-WLINEAR找出其中的中位数$m*$（同时调整好了组的位置）
4. 基于$m*$的大小对所有元素（含第1步凑不齐一组的元素）进行划分，假设有$x-1$个元素小于$m*$

5. 若$k=x$，返回$m*$；若$k<x$，对小于$m*$的元素调用SELECT-WLINEAR找阶为$k$的元素；若$k>x$，对大于$m*$的元素调用SELECT-WLINEAR找阶为$k-x$的元素

$W(n) \le W(\lceil \frac{n}{5} \rceil) + W(\frac{7}{10}n+6) + O(n)$

第1项是找组中的中位数的中位数

第2项是划分结果。在所有$\lceil \frac{n}{5} \rceil$组中，至少有一半的小组要贡献3个比$m*$大的元素，其中不包括$m*$所在组以及最后凑不满的组，那么至少也淘汰掉$3(\frac{1}{2}\lceil \frac{n}{5} \rceil - 2) \ge \frac{3n}{10} - 6$个元素，子问题最大也不过$n-(\frac{3n}{10}-6) = \frac{7}{10}n+6$

第3项是杂七杂八的用时

## 简单数据结构

**堆**

- 堆结构特性：比完美二叉树在底层少若干节点，且底层左侧连续排列
- 堆偏序特性：根节点的值大于所有子节点的值（大根堆）

**堆的修复**

**算法52-FIX-HEAP(A,p)**

取出堆顶后，左右子树仍满足两性质。先恢复堆结构特性，再恢复堆偏序特性：

1. 底层最右侧的元素移至堆顶（堆结构fixed）
2. 从堆节点开始递归地，将父亲节点与两子节点中大者交换，直到叶子节点（堆偏序fixed）

修复次数不超过堆高度为$O(\log n)$，每次代价为$O(1)$，堆修复代价为$O(\log n)$

**堆的构建**

**算法53-CONSTRUCT-HEAP(A,p)**

1. 将元素摆放在堆中
2. 从叶子节点开始，子节点中的最大值若大于父亲节点则与父亲节点交换，并从该子节点位置开始向下修复堆

$W(n) = 2W(\frac{n}{2}) + 2\log n \; W(n) = O(n^{\log_2 2}) = O(n)$

**堆排序**

基于大根堆进行升序排序

堆顶与底层最右侧的叶子交换后，堆大小-1使原堆顶不在堆的处理范围；反复进行直到堆大小为0

```
func HEAP-SORT(A)
    A建堆
    for i from 1 to n:
        swap(A[1], A[n+1-i])
        堆A大小--
        FIX-HEAP(1)
```

**优先队列**

增加了INSERT插入和EDIT-KEY改权操作

- INSERT：新增叶子节点（从左到右），堆增大，新节点向上上浮同时向下修复
- EDIT-KEY：直接修改权值，向上上浮同时向下修复

## 并查集

变化的、扩增的等价关系，即**动态等价关系**

- FIND(a_i)：返回a_i所在的**等价类的代表元**
- UNION(a_i,a_j)：将a_i和a_j所在的等价类合并成一个等价类

**普通并+普通查**

慢

$O(nl)$

**加权并+普通查**

**算法：WEIGHTED-UNION**

feature: **WEIGHTED-UNION**

把节点数更少的挂到更多的上，需要在根节点记录根树的大小信息

- 基于WEIGHTED-UNION的并查集，k个节点的树高不超过$\lfloor \log k \rfloor$，证明：

$k=1$时显然成立

假设对任意$m<k$，m个节点的树高不超过$\lfloor \log m \rfloor$

假设有k个节点的树T是由k1个节点的子树T1和k2个节点的子树T2合并而成的，不妨设$k_1 \geq k_2$

此时T树高$h = \max h_1, h_2 + 1$

而$h_1 \leq \lfloor \log k_1 \rfloor \leq \lfloor \log k \rfloor$且$k_2 \leq \frac{k}{2}, h_2 + 1 \leq \lfloor \log k_2 \rfloor + 1 \leq \lfloor \log \frac{k}{2} \rfloor + 1 = \lfloor \log k \rfloor$

综上，$h \leq \lfloor \log k \rfloor$

- 采用WEIGHTED-UNION和FIND的并查集，最坏代价为$O(n + l \log n)$

树高不超过$\log n$，那么FIND代价不超过$O(l \log n)$

初始化，需要$O(n)$

WEIGHTED-UNION代价不超$O(n)$

所以n个节点长度为l的并查集程序代价为$O(n + l \log n)$

**加权并+路径压缩**

**算法：WEIGHTED-UNION + C-FIND**

在C-FIND找到祖先节点后，立即更新沿途的节点的父亲为该祖先

- 最坏情况时间复杂度为$O((n + l) \log^* n) \approx O(n + l)$

## 哈希表

哈希表实现了接近下界$O(1)$的准常数时间性能$O(1 + \alpha)$

定义**负载因子**$\alpha = \frac{n}{m}$，它反映了哈希表的拥挤程度

**直接寻址表**

键值空间U为每个元素分配了空间，查找每个元素用时$O(1)$但空间开销惊人

**简单均匀哈希**

多用$h(k) = k \mod n$等函数

**封闭寻址 close address**

又叫链式寻址

在哈希表的每个位置放的是指向一个链表头部的指针

**冲突元素会直接插入到链表头部而不是尾部**，以实现$O(1)$的插入

一次成功查找，代价为$O(1)$

一次不成功查找，平均代价为$\Theta(1 + \alpha)$。不成功查找的比较次数为找到链表尾所用次数。

**开放寻址**

$i$都是从0开始

- 线性探测：直接往后一个个挪看有没有位置

$h(k, i) = (h'(k) + i) \mod m$

- 二次探测：第i次从$+1^2, -1^2, +2^2, -2^2, +3^2, -3^2, \cdots, +n^2, -n^2$里选第i项加上（不是累加）

$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \mod m$

- 双重哈希：如果冲突，两个函数一起算

$h(k, i) = (h_1(k) + i h_2(k)) \mod m$

假设$\alpha = \frac{n}{m} < 1$:

- <u>不成功查找的平均代价不超过</u>$\frac{1}{1-\alpha}$
- <u>成功查找的平均代价不超过</u>$\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$

## 平摊分析

$C_{actual}$实际代价，直接、精准地反映了每次的代价

$C_{accounting}$记账代价，让一些操作产生的额外的代价（正）以抵消另外一些操作的代价（负的代价）

$C_{amortized} = C_{act} + Caccounting$平摊代价

运行花费较低的operations时先存credit未雨绸缪, 供未来花费较高的operations使用

**设置每个操作的平摊成本(amortized cost)后，要做valid check确保任何时刻credit不可以是0**

举几个平摊分析的例子：

### 对手论证

将算法设计者与算法分析者看作对手，同时扮演两个角色进行算法分析。

1. 算法设计者：尽量多的创造更多信息
2. 算法分析者：尽量少的给予信息，拥有着**随时合理改变取值**的能力

具体看书