

# 算法设计与分析复习笔记

## 一、概论

### 1. 求最大公约数：辗转相除：

•gcd(60,24) = ?

- ① gcd(60,24) =
- ②  $60 / 24 = 2$  余 12  $\therefore m = 24$   $n = 12$
- ③ gcd(24,12) =
- ④  $24 / 12 = 2$  余 0  $\therefore m = 12$   $n = 0$
- ⑤ gcd(12,0) = 12

**第一步：**如果  $n=0$ ，返回  $m$  的值作为结果，同时过程结束；否则，进入**第二步**。

**第二步：**用  $n$  去除  $m$ ，将余数赋给  $r$ 。

**第三步：**将  $n$  的值赋给  $m$ ，将  $r$  的值赋给  $n$ ，返回**第一步**。

最后一步  $\text{gcd}(12,0)=12\div 0=0\cdots 12$

所以  $\text{gcd}(12,0)=12$

### 2. 算法的时间复杂性分析：

#### a) 非递归算法的时间复杂度：

- i. 选择某种能够用来衡量算法运行时间的依据
- ii. 依照该依据求出运行时间  $T(n)$  的表达式
- iii. 采用渐进符号表示  $T(n)$
- iv. 获得算法的渐进时间复杂性，进行进一步的比较和分析

#### b) 递归算法的时间复杂度：

- i. 决定采用哪个(或哪些)参数作为输入规模的度量；
- ii. 找出对算法的运行时间贡献最大的语句作为基本语句；
- iii. 检查一下,对于相同规模的不同输入,基本语句的执行次数是否不同。如果不同,就需要从最好、最坏及平均三种情况进行讨论；
- iv. 对于选定的基本语句的执行次数建立一个**递推关系式** $T(n)$  与  $T(n-1)$

**的关系**，并确定停止条件；

- v. 通过**计算该递推关系式**（计算  $T(n)$  与  $T(1)$  的关系，将  $T(1)$  带入为常数  $C$ ，得到  $T(n)$  的表达式）得到算法的时间复杂度。

#### c) **主定理法求解递归算法的时间复杂性**（适用于每次递归数据规模成倍数衰减——分治策略）

第二章分治策略中，通常设计为递归算法  
其时间复杂性的递归定义一般有如下形式：

$$T(n)=\begin{cases} O(1) & n=n_0 \\ aT(\frac{n}{b})+f(n) & n>n_0 \end{cases}$$

使用master定理方法可以快速求解该方程

这里要求  $a \geq 1, b > 1$ ,  $f(n)$  是正函数

i.

## 二、Master定理方法求递归算法时间复杂度

1. 首先根据 
$$T(n) = \begin{cases} O(1) & n = n_0 \\ aT(\frac{n}{b}) + f(n) & n > n_0 \end{cases} \xrightarrow{\text{红色箭头}} n^{\log_b a}$$

2. 比较  $n^{\log_b a}$  和  $f(n)$  的阶的关系 ( $>$ ,  $=$ ,  $<$ ), 求  $T(n)$

规则1: 如果  $f(n) = O(n^{\log_b a - \epsilon})$ ,  $\epsilon > 0$  为常数, 则  $T(n) = O(n^{\log_b a})$

规则2: 如果  $f(n) = \Theta(n^{\log_b a})$ , 则  $T(n) = O(n^{\log_b a} \log n)$

规则3: 如果  $f(n) = \Omega(n^{\log_b a + \epsilon})$ ,  $\epsilon > 0$  为常数, 且存在  $n_0$ , 当  $n > n_0$  时,

$af(\frac{n}{b}) \leq cf(n)$  成立,  $c < 1$  为常数, 则  $T(n) = O(f(n))$

ii.

iii. 主定理总结: 孩子干的多 ( $n^{\log(ba)}$  大), 以孩子为主导 ( $O(n^{\log(ba)})$ );  
自己干的多 ( $f(n)$  大), 以自己为主导 ( $O(f(n))$ )  
两个一样大时, 结果乘以层数  $\log n$

d) 渐进上界:  $O$

渐进下界:  $\Omega$

渐进准确界:  $\theta$

**定理:** 若  $T(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n$  ( $a_i > 0, i=1, 2, \dots, m$ ) 是关于  $n$  的一个  $m$  次多项式, 则  $T(n) = O(n^m)$ , 且  $T(n) = \Omega(n^m)$ , 因此有  $T(n) = \Theta(n^m)$ 。

$O$  的性质:

(1).  $O(f) + O(g) = O(\max(f, g))$

补充: 当出现  $O(1)$  时直接按照常数  $C$  处理

## 二、分治法

1. 设计思想:

- (1) 将一个难以直接解决的大问题分割成一些规模较小的相同问题, 分而治之;
- (2) 反复使用分治手段, 可以使子问题规模不断缩小
- (3) 分治法解决问题通常使用递归算法
- (4) 边界条件 (保证在有限次计算后得到结果) 和递归方程是递归函数的两个要素

2. 快速幂算法:

给定实数  $a$  和非负整数  $n$ , 用分治法设计求  $a^n$  的快速算法 (递归算法)

分析:

$$a^n = \begin{cases} 0 & a = 0 \\ 1, & n = 0 \text{ 且 } a \neq 0 \\ (a^{\frac{n}{2}})^2 & n > 0, n \text{ 为偶数} \\ (a^{\frac{n-1}{2}})^2 \times a & n > 0, n \text{ 为奇数} \end{cases}$$

该问题满足四个条件

时间复杂度  $O(\log n)$

```
double exp2(double a, int n)
{
    if (a == 0) return 0;
    if (n <= 0) return 1;
    else
    {
        int x = exp2(a, n/2);
        if (n % 2) return a * x * x;
        else return x * x;
    }
}
```

(1)

3. Strassen 矩阵乘法

- (1) 设计思想：将 A,B,C 矩阵划分为 4 个大小相等的子矩阵，递归地计算子矩阵各个位置对应的值（不好——>并没有真正改进算法的时间复杂度：本质上是因为乘法运算的次数并没有真正减少）

为了降低时间复杂度，必须减少乘法的次数。

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$T(n) = O(n^{\log_2 7}) = O(n^{2.81}) \quad \checkmark \text{较大的改进} \odot$$

$$T(n) = \begin{cases} O(1) & n = 2 \\ 7T(n/2) + O(n^2) & n > 2 \end{cases}$$

$$M_1 = A_{11}(B_{12} - B_{22})$$

$$M_2 = (A_{11} + A_{12})B_{22}$$

$$M_3 = (A_{21} + A_{22})B_{11}$$

$$M_4 = A_{22}(B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$$

$$C_{11} = M_5 + M_4 - M_2 + M_6$$

$$C_{12} = M_1 + M_2$$

$$C_{21} = M_3 + M_4$$

$$C_{22} = M_5 + M_1 - M_3 - M_7$$

(2)

#### 4. 棋盘覆盖问题：

- (1) 递归：向着一个方向不断深入：

	3	4	4	8	8	9	9
3	3	2	4	8	7	7	9
5	2	2	6	10	10	7	11
5	5	6	6	1	10	11	11
13	13	14	1	1	18	19	19
13	12	14	14	18	18	17	19
15	12	12	16	20	17	17	21
15	15	16	16	20	20	21	21

①

- (2) 队列的非递归：广度优先遍历：先填充相同性质的位置


6	6	7		10	10	11	11
6	2	7	7	10	3	3	11
8	2	2	9	12	12	3	13
8	8	9	9	1	12	13	13
14	14	15	1	1	18	19	19
14	4	15	15	18	18	5	19
16	4	4	17	20	5	5	21
16	16	17	17	20	20	21	21

①

- (3) 栈的非递归：深度优先遍历：从右下角逐渐上推


21	21	20		16	16	15	15
21	17	20	20	16	12	12	15
19	17	17	18	14	14	12	13
19	19	18	18	1	14	13	13
11	11	10	1	1	6	5	5
11	7	10	10	6	6	2	5
9	7	7	8	4	2	2	3
9	9	8	8	4	4	3	3

①

- (4) 注意：

- ① 不同的算法形成的覆盖图案不同
- ② 四个子问题的处理顺序（上右下左）不同，形成的覆盖图案也不同
- ③ 栈的运行时间最短，队列的运行时间最长（深搜优于广搜）

#### 5. 线性时间选择：找出线性序列（不一定有序）第 k 小的元素

- (1) 将 n 个元素 5 个一组划分为  $\frac{n}{5}$  组，对每一组元素排序，找出每一组的中位数，再对

这些中位数排序并找出这些数的中位数，将这个最终找到的中位数作为基准值，将所有小于基准值的数放在基准值左边，将所有大于基准值的数放在基准值右边（类似于快速排序），判断  $k$  落在哪个区间（左边 or 右边），对该区间继续进行递归。

(2) 时间复杂度计算：

- ① 分组找中位数  $O(n)$ ;
- ② 在中位数组中找中位数:  $O(n/5)$
- ③ 分区时间:  $O(n)$ ——划分到左右两边
- ④ 递归最多在  $7n/10$  的一边:  $T(7n/10)$ : 中位数的中位数这样的基准值选择保证了每次至少有  $3n/10$  个元素被排除在递归之外
- ⑤  $T(n) \leq T(n/5) + T(7n/10) + O(n) \Rightarrow T(n) = O(n)$
- ⑥ 综上所述，总的时间复杂度为  $O(n)$

6. 循环赛日程表：

- (1) 重点：将左上的表 copy 到右下（最小单位为 1），递归此过程，直到表全部被填满

**例题9：循环赛日程表**

设计一个满足以下要求的比赛日程表：

- (1) 每选手必须与其他  $n-1$  个选手各赛一次
- (2) 每选手一天只能赛一次
- (3) 循环赛一共进行  $n-1$  天
- (4) 选手人数  $n=2^k$

分治策略：将选手分为两半， $n$  个选手的比赛日程表就可以通过为  $n/2$  个选手设计的比赛日程表来决定。递归地对选手进行分割，直到只剩下 1 个选手时，比赛日程表不需排

1							
2							
3							
4							
5							
6							
7							
8							

1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

(2)

### 三、动态规划

1. 基本思想：将待求解的问题分解成若干个子问题，但是分解得到的子问题往往不是互相独立的（如果使用分治法求解，有些子问题会被重复计算）——>动态规划的核心思想就是要保存已解决的子问题的答案，在后续需要时可以在常数时间内查找得到该子问题的答案。

2. 动态规划的基本步骤：

(1) 找出最优解的性质，刻画最优子结构（当原问题最优时，其子问题也是最优解）

(2) 递归地定义最优值

(3) 自底向上地计算出最优值

(4) 根据最优值信息，构造最优解

3. 矩阵连乘问题：选择一种加括号的方式，使得按照这种计算顺序需要计算的乘法次数最少。

(1) 定义状态:  $m[i][j]$  表示从矩阵  $A_i$  到矩阵  $A_j$  的最小计算代价

(2) 状态转移方程:  $m[i][j] = \min_{i \leq k < j} \{m[i][k] + m[k+1][j] + p_{i-1} \cdot p_k \cdot p_j\}$

(3) 初始条件:  $m[i][i] = 0$

(4) 最后就是按照状态转移方程填写矩阵连乘记录表

#### 4. 使用动态规划算法的前提:

(1) 求解的问题具有最优子结构性质 (一个问题的最优解可以由其子问题的最优解组合而成)

(2) 子问题可能会被反复地计算

(3) 无后效性: 某一状态的最优解只依赖于它之前的状态, 而与之后的选择无关

(4) 动态规划算法需要有特定的起点, 作为划分子结构的依据; 而贪心算法则不需要。

#### 5. 最长公共子序列:

用  $c[i][j]$  记录序列的最长公共子序列的长度。其中:

$X_i = \{x_1, x_2, \dots, x_i\}; Y_j = \{y_1, y_2, \dots, y_j\}$ 。

递归关系如下:

$$c[i][j] = \begin{cases} 0 & i = 0 \vee j = 0 \\ c[i-1][j-1] + 1 & i, j > 0; x_i = y_j \\ \max\{c[i][j-1], c[i-1][j]\} & i, j > 0; x_i \neq y_j \end{cases}$$

注意:  $c[i][j]$  中存储的是从  $X_i$  到  $Y_j$  的公共子序列的长度 (int) ——  $i$  代表的是在  $X$  序列中取前  $i$  个元素,  $j$  代表的是在  $Y$  序列中取前  $j$  个元素。

#### 6. 0-1 背包问题 (动态规划求解): 要求物品不能被拆解

(1) 状态定义: 定义  $m[i][j]$  表示前  $i$  个物品装入容量为  $j$  的背包所能获得的最大价值 ( $i$  表示考虑前  $i$  个物品;  $j$  表示此时背包的剩余容量)

(2) 状态转移方程:

对于第  $i$  个物品, 通常有两种选择:

a. 不能将  $i$  装入背包:  $m[i][j] = m[i-1][j]$

b. 可以将  $i$  装入背包 (选择装或不装):

$$m[i][j] = \max\{m[i-1][j], m[i-1][j-w[i]] + v[i]\}$$

(3) 写初值

(4) 自底向上地计算出表格中的每一项

(5) 找到最后的结果



## 四、贪心算法

- 1.通俗理解：一步步做选择，只考虑当前看来最优的解（局部最优解），不从整体考虑。
- 2.活动安排问题：找到一种**贪心选择策略**，按照该方法经过一次次选择，找出相容最多的活动。
- 3.贪心选择算法的基本要素：
  - (1) 贪心选择性质：所求问题的整体最优解（近似最优解）可以通过一系列局部最优的选择（贪心选择）来达到；贪心算法通常是**自顶往下（第一步时就根据贪心策略决定了选择，不回溯）**的方式进行，每一次贪心就将问题简化成规模更小的子问题（**问题的最优解可以转化为一部选择之后子问题的最优解**）
  - (2) 最优子结构性质：**子问题的最优解组合才能形成原问题的最优解**
- 4.多机调度问题：
  - (1) 目标：在最短时间内完成作业  
贪心策略：最长处理时间作业优先  
**首先将  $n$  个作业依其所需的处理时间从大到小排序。然后依此顺序将作业分配给占用时间少的处理机。**  
**算法所需的计算时间为  $O(n\log n)$ ——>排序的时间，分配需要的是常数级的时间。**
- 5.哈夫曼编码：按照字符出现的频率进行编码（**对一个字符的编码一定不能成为另一个字符编码的前缀**）
  - (1) 核心思想：让出现频率高的字符尽可能地出现在编码二叉树的上方（离根结点更近）
  - (2) 过程：
    - a.将所有字符按照出现频率按升序排列
    - b.将权重最小的两个节点合并到新的节点（新节点的权重为两个叶节点的权重之和），并将新节点按照权重插入到原来的有序队列中；
    - c.重复上述过程，直到有序队列中只剩下一个元素，该元素即为哈夫曼树的根结点
    - d.得到哈夫曼树后从根节点出发，左走记 0，右走记 1，直到叶节点，所有的路径标号即为该叶节点对应字符的哈夫曼编码。
- 6.单源最短路径的 Dijkstra 算法：从源开始，固定到源距离最小的点，从该点出发，更新所有可达点到源的最小距离；重复此步骤直到遍历所有的点。
- 7.最小生成树（MST）：能连接图  $G$  所有节点且边权之和最小的子图  $G'$ 称为最小生成树
  - (1) Prim 算法：  
目标：找到最小生成树  
贪心策略：找到已划分顶点集内一点到已划分顶点集外一点的最短割边，并将该点加入到已划分顶点集内，直到所有顶点加入到已划分顶点集。（**找割边的最小值**）
  - (2) Kruskal 算法：  
目标：找到最小生成树  
贪心策略：先将所有顶点均视为不连通分支；将所有边按照权重大小排序，一次检查每条边（按权重由小到大），当该边的两个顶点不属于同一连通支时，用该边将这两个顶点归为同一个连通支。

当图的边数为 $e$ 、顶点数为 $n$ 时，Kruskal算法所需的计算时间是 $O(e\log e)$ ，Prim算法是 $O(n^2)$ 。  
当 $e=\Omega(n^2)$ 时，即边数多时，Kruskal算法比Prim算法差；  
但当 $e=O(n^2)$ 时，即边数少时，Kruskal算法却比Prim算法好得多。

## 五、回溯法：通用的解题之法——一种能避免不必要搜索的穷举式搜索

1.核心思想：深度优先遍历

2.在使用回溯法解决问题时：

- (1) 首先要定义问题的解空间( $x_1, x_2, \dots, x_i$ )， $x_i$  代表对分量的不同的取值
- (2) 然后将解空间组织起来（一般组织成树或图的形式）
- (3) 采用深度优先的方法对节点进行拓展

总结：首先用树 or 图的方式列出所有的解（解空间），然后采用深度优先的方式遍历解空间（层层深入地往下走），如果走不通（剪枝），则退回到上一个节点，继续用深度优先的方式探索其他儿子节点。如果能走到解空间的叶节点，则该路径为一条可行解

3.回溯法的三种解题框架

(1) 递归回溯：使用递归实现深度优先遍历

(2) 子集树算法框架

a.问题描述：从  $n$  个元素的集合  $S$  中找出满足某种性质的子集时，相应的解空间树称为子集树，时间复杂度为  $O(2^n)$

b.注意：子集树是二叉树，代表对第  $i$  个元素，会有选或不选两种可能。

(3) 排列树算法框架

a.问题描述：当所给的问题是确定  $n$  个元素满足某种性质的排列时，相应的解空间树称为排列树。时间复杂度  $O(n!)$ 。

b.注意：排列树是一颗一般的树，表示可以从当前节点扩展到所有与其相关的子节点上（对子节点不同的访问顺序代表不同的排列方式）

4.旅行售货员问题：一位售货员要拜访若干个城市，每个城市只访问一次，最后回到出发地。

请问他应如何选择路径，使得总的旅行距离最短？（每个节点只走一次——哈密顿回路）

注意：在旅行售货员问题中，解空间树的边对应不同的城市（选择某条边代表着下一个要去某个城市）

5.一般回溯法只保存从根结点到当前扩展结点的路径，不保存整个解空间（当发现当前解到达叶节点且路径长度耗费小于当前最优解，则更新最优解——用独立于解空间树的新的数据结构保存最优解）

6.货物装载问题：将  $n$  个质量不同的物品装载到 2 艘船上

(1) 核心思想：保证一艘船装载的货物尽可能的多——使用回溯法找到尽可能接近第一艘船载重量的货物组合（子集树——对于第  $i$  个货物选 or 不选）

7. 剪枝：用约束函数在扩展结点处剪去不满足约束的子树；用限界函数剪去得不到最优解的子树（判断未到达叶节点是已经不满足最优就可以直接剪枝）。

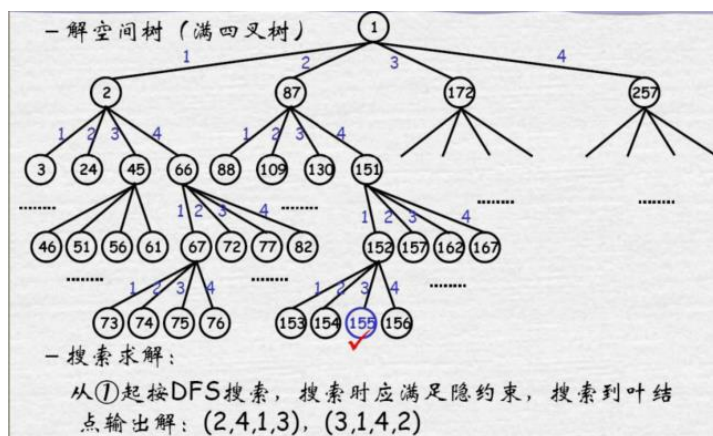
上界函数：在子集树问题中，问题的上界（已有路径的代价+剩余未选择代价的极值）已经无法使得当前路径达到已知最优解的规模时，就可以提前对当前路径进行剪枝。

注意：在引入剪枝的前提下，只要问题能进行到叶节点，就可以修正当前最优解

8.批处理作业调度：全排列问题——排列树

9.n 皇后问题：在同一行/同一列/同一斜线上不能有两个及以上棋子——完全  $n$  叉树表示解空间，树的根结点表示搜索的初始状态，从根结点到第 2 层结点对应皇后 1 在棋盘第 1 行的可能摆放位置，从第 2 层结点到第 3 层结点对应皇后 2 在棋盘第 2 行的可能摆放位置，依此类推。（每个分叉代表着不同的排列选择）

Eg.四皇后问题的解空间:



注意: 节点代表的不是这一层选择的位置, 不同的边 (分支才代表不同的选择, 比如说, 第一行就有四种可能的选择, 对应到解空间树上就是从根结点向外的 4 条分支)

四皇后问题最终的解: (2, 4, 1, 3) or (3, 1, 4, 2)

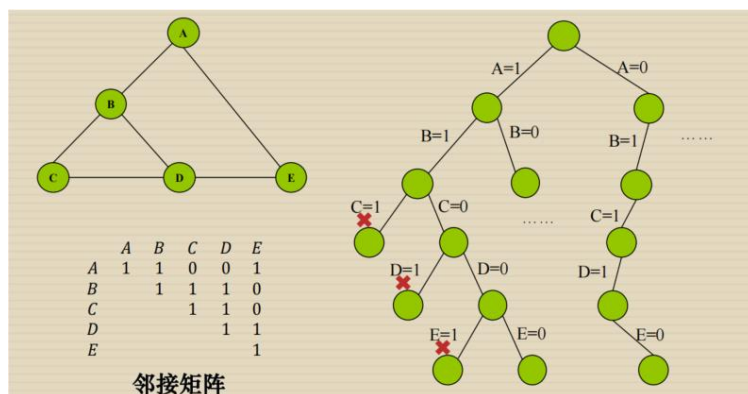
10.0-1 背包问题:

(1) 剪枝上界: 当右子树的最大价值 (按背包问题求解) 小于已知最优解, 则对右子树剪枝。

11.最大团问题:

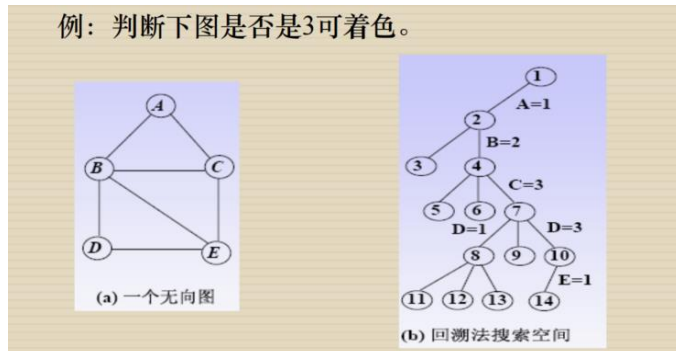
(1) 团 (完全子图): 一个点集中每对顶点之间都存在一条边

(2) 构建一个子集树: 表示第  $i$  个点是否加入已有的点集



12.图的着色问题

例: 判断下图是否是3可着色。



这是一个排列树, 不同的分支代表选择不同的颜色, 如果涂色问题产生冲突, 则剪枝

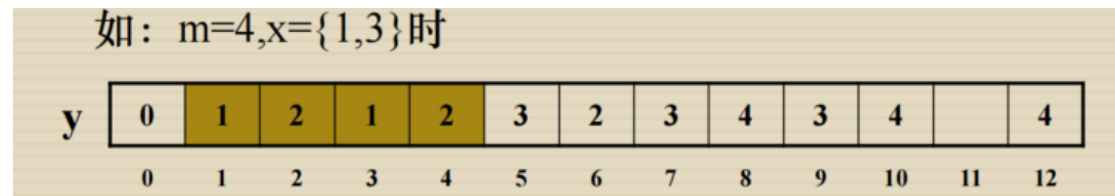


13.连续邮资问题：确定  $n$  中邮票不同的面值

重难点：计算已知的  $k$  种面值所能得到的连续邮资的最大值  $r$ ，第  $k+1$  种面值的取值范围是  $x[j-1]+1 \sim r+1$ （对应不同的分支）

计算  $r$  的值：

用已有的  $x[1:i]$  计算不超过  $m$  张时可以贴出邮资  $k$  所需要的最少的邮票数  $y[k]$



（其实在做题的过程中  $r$  可以一个一个计算出来）

## 六、分支限界法

1.基本思想：分支限界法的求解目标则是找出满足约束条件的一个解，或是在满足约束条件的解中找出在某种意义下的最优解。

2.分支限界法与回溯法的区别：分支限界法采用广度优先搜索；而回溯法采用深度优先的方式搜索解空间

3.在分支限界法中，一旦某个活结点成为扩展结点，则一次性产生其所有儿子节点，在这些儿子节点中，导致不可行解或者非最优解的儿子节点被舍弃，其余儿子节点加入活结点列表，等待被选中成为扩展结点。此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。

4.三类常用的选择活结点作为扩展结点的方法：

(1) 先进先出 (FIFO)

(2) 后进先出 (LIFO)

(3) 优先队列：按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

5.优先队列式分支限界法的基本思想：通过优先队列控制搜索结点的扩展顺序，优先扩展最有可能产生最优解的结点，结合限界策略剪枝。

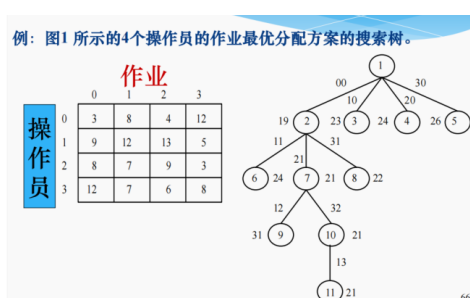
6.装载问题：尽可能地将第一艘船装满

(1) 优先队列式分支限界法：活结点  $x$  在优先队列中的优先级定义为从根结点到结点  $x$  的路径所相应的载重量再加上剩余集装箱的重量之和。

(2) 在优先队列式分支限界法中，一旦有一个叶结点成为当前扩展结点，则可以断言该叶结点所相应的解即为最优解。此时可终止算法。

7.作业分配问题：

(1) 优先队列的优先级是通过前序任务一定的情况下，计算剩余任务+前序任务的总时间（下界时间）决定的，下界时间越少，优先级越高



8.布线问题：广度优先的一种蔓延的策略

9.分支限界法与回溯法的比较：

(1) 回溯法的求解目标是找出解空间树中满足约束条件的所有解，而分支限界法的求解目标则是找出满足约束条件的一个解，或是在满足约束条件的解中找出在某种意义下的最优解。

(2) 回溯法以深度优先的方式搜索解空间树，而分支限界法则以广度优先（队列）或以最小耗费优先（优先队列）的方式搜索解空间树

## 七、概率算法

1.定义：概率算法是一类在运行过程中使用随机数，并且其行为或输出带有概率性的算法。对于同一个输入的多次运行，概率算法可能产生不同的输出或者耗时不同。

2.概率算法通常比最优选择算法省时。

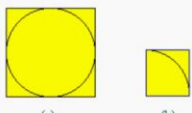
3.数值概率算法：将一个问题的计算与某个概率分布确定的事件联系起来，求得问题的近似解。近似解的精度随计算时间的增加而不断提高。

(1) 用随机投点法计算 $\pi$ 的值：

a.

**用随机投点法计算 $\pi$ 值**  
设有一半径为 $r$ 的圆及其外切正方形。向该正方形随机地投掷 $n$ 个点。设落入圆内的点数为 $k$ 。由于所投入的点在正方形上均匀分布，因而所投入的点落入圆内的概率为 $\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$ 。所以当 $n$ 足够大时， $k$ 与 $n$ 之比就逼近这一概率。从而  $\pi \approx \frac{4k}{n}$

```
double Darts(int n)
{ // 用随机投点法计算 $\pi$ 值
  static RandomNumber dart;
  int k=0;
  for (int i=1; i<=n; i++) {
    double x=dart.fRandom();
    double y=dart.fRandom();
    if ((x*x+y*y)<=1) k++;
  }
  return 4*k/double(n);
}
```

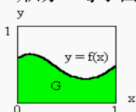


(a) (b)

(2) 计算定积分：

设 $f(x)$ 是 $[0, 1]$ 上的连续函数，且 $0 \leq f(x) \leq 1$ 。

需要计算的积分为 $I = \int_0^1 f(x) dx$  积分  $I$  等于图中的面积 $G$ 。



在图所示单位正方形内均匀地作投点试验，则随机点落在曲线下方的概率为

$$P_r \{y \leq f(x)\} = \int_0^1 \int_0^{f(x)} dy dx = \int_0^1 f(x) dx$$

假设向单位正方形内随机地投入 $n$ 个点 $(x_i, y_i)$ 。如果有 $m$ 个点落入

$G$ 内，则随机点落入 $G$ 内的概率  $I \approx \frac{m}{n}$

4.蒙特卡洛算法：在有限的时间内采用随机化的方法解决问题（可能会有错误的解），但是可以通过多次实验降低错误率。

5.拉斯维加斯算法：用随机的方法去碰撞正确的解，如果碰撞失败，则重新开始（保证一定有正解）

6.舍伍德算法：针对某个特定的算法，将输入随机打乱，从而避免了直接输入最坏情况，获得该算法的平均运行时间——虽然不能消除最坏情况，但是让它几乎不可能发生。（快速排序、线性时间选择）

## 7.产生随机数（伪随机数）的方法：线性同余法

**线性同余法**是产生伪随机数的最常用的方法。由线性同余法产生的随机序列 $a_0, a_1, \dots, a_n$ 满足：

$$\begin{cases} a_0 = d \\ a_n = (ba_{n-1} + c) \bmod m \quad n = 1, 2, \dots \end{cases}$$

其中： $b \geq 0, c \geq 0, d \geq m$ 。 $d$ 称为该随机序列的种子。如何选择常数 $b$ 、 $c$ 和 $m$ 直接关系到所产生的随机序列的随机性能。这是随机性理论研究的内容，已超出本书讨论的范围。

从直观上看， $m$ 应取得充分大，因此可取 $m$ 为机器大数，另外应取 $\gcd(m, b) = 1$ ，因此可取 $b$ 为一素数。

## 8.搜索有序表：

### 搜索有序表

用两个数组来表示所给的含有 $n$ 个元素的有序集 $S$ 。用 $\text{value}[0:n]$ 存储有序集中的元素， $\text{link}[0:n]$ 存储有序集中元素在数组 $\text{value}$ 中位置的指针。 $\text{link}[0]$ 指向有序集中第1个元素，即 $\text{value}[\text{link}[0]]$ 是集合中的最小元素。一般地，如果 $\text{value}[i]$ 是所给有序集 $S$ 中的第 $k$ 个元素，则 $\text{value}[\text{link}[i]]$ 是 $S$ 中的第 $k+1$ 个元素。 $S$ 中元素的有序性表现为，对于任意 $1 \leq i \leq n$ 有 $\text{value}[i] \leq \text{value}[\text{link}[i]]$ 。对集合 $S$ 中的最大元素 $\text{value}[k]$ 有， $\text{link}[k] = 0$ 且 $\text{value}[0]$ 是一个大数。例如，有序集 $S = \{1, 2, 3, 5, 8, 13, 21\}$ 的一种表示方式如下图：

i	0	1	2	3	4	5	6	7
Value[i]	$\infty$	2	3	13	1	5	21	8
Link[i]	4	2	5	6	1	7	0	3

如果在  $\text{value}[i]$  这个节点，下一个元素是  $\text{value}[\text{link}[i]]$ ，这种表示有序集的方法实际上是用数组来模拟有序链表。

在最坏情况下，搜索一个元素需要 $O(n)$ 的时间复杂度，提高查找效率的一个方法是采用**跳跃表**：在有序链表的部分结点处增设附加指针以提高其搜索性能。在增设附加指针的有序链表中搜索一个元素时，可借助于附加指针跳过链表中若干结点，加快搜索速度。

**跳跃表每增加一层，新的索引序列中会保留原来一半的节点**

在插入新节点后（在最底层），以 $p = 0.5$ 的概率决定当前节点是否上升一层（不一定保证每一层的节点都比前一层少一半），如此产生的新结点的级别有可能是一个很大的数，甚至远远超过表中元素的个数。为了避免这种情况，用 $\log_{1/p} n$  作为新结点的级别的上界。（新节点的级别不超过 $\log_2 n$ ）

## 八、线性规划与网络流

### 1.线性规划问题的一般形式：

- （1）求解目标：满足某个条件的最优解
  - （2）限制条件：各种约束条件
  - （3）线性规划问题是在一组**线性约束条件**的限制下,求一**线性目标函数**最大或最小的问题
- 2.单纯形算法：先找出一个基本可行解（可行极点——凸可行域的顶点），判断其是否为最优解，如果不是，则转换到相邻的基本可行解
- 3.单纯形算法的求解过程：

- （1）对于约束条件中的不等式，首先引入松弛变量将其转化为等式

(2) 构造单纯形初始表格：以松弛变量  $s_1$  和  $s_2$  作为基变量（令  $x_1$  和  $x_2$  初始时为 0，求  $s_1$  个  $s_2$ ）

(3) 选择  $z$  行对应系数最大的一个变量  $x_i$  进入基底（选择对  $z$  目标影响最大的变量进入基底）；选择比值最小的元素出基。

(没学会)

#### 4. 最大网络流

(1) 定义：在有向图（网络）中，每一条边有一个容量，希望找出从源点到汇点最多能有多少流量，使多有边都不超过其容量。

(2) 相关概念：

a. 流值：这条边当前已经占用的流量

b. 容量：这条边能供承受的最大流量

c. 流量守恒：对于除了源点和汇点之外的其他任何一个点，流量守恒（入度等于出度）

(3) 操作过程：

a. 初始化：当前流：所有边为 0；残存流：原始容量-当前流

b. 任意找到一条没有重复节点的路径——>增广路径，整个路径的流量由某一边的最大流量决定（瓶颈）

c. 在残存网络中重复 b 过程（找增广路），直到无法找到新的增广路——但是找到的流不一定是最大的！

d. 核心改进：残存图中建立双向边（更新路径时正向减少，反向增加相同的值：瓶颈）

e. 过程仍然是找增广路更新当前图和残存图（包含刚刚添加的反向流量），因为新找的增广路总是从源点到汇点，所以保证了网络中的总流量总是不断增加的。

## 九、NP 完全性理论

1. 规约：一个问题 A 可以规约为 B：可以用问题 B 的解法解决问题 A。（B 更难解）

2. 易解问题：存在多项式时间算法的问题

难解问题：在指数时间内解决的问题

3. P 类问题：可以在多项式时间内解决的问题

NP 问题：可以在多项式时间内验证一个解是否正确的问题（数独，存在哈密顿回路）

NP 难问题：任意一个 NP 问题都可以在多项式时间内归约到它的问题。（难度大于等于 NP 问题的问题）

## NP难问题 (NPH, NP-hard)

- 1、**定义：**任意NP问题可以在多项式时间内规约成该问题。
  - 对于一个问题S，满足任何NP问题都可以在多项式复杂度范围之内被规约为S，当解决问题时，所有NP问题就都被解决，可以认为这是一个比所有NP问题都难的问题。
- 2、**注意：** NP难问题与NP类问题不完全互相包含。因为一个问题规约后会变得更难，就不一定还能在多项式时间内验证答案。
- 3、**例子：**旅行商的求最短回路问题

