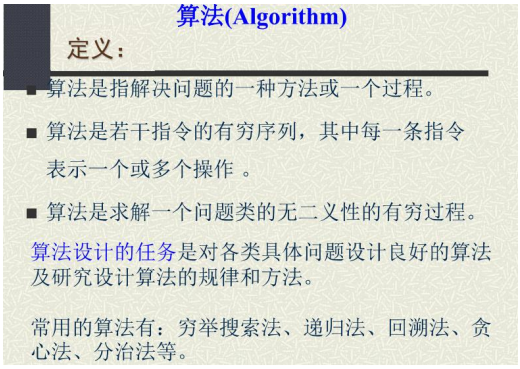
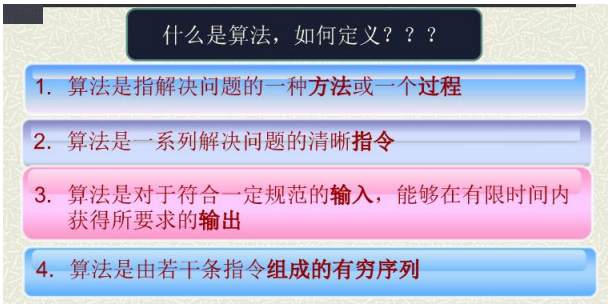
第一章 算法和算法分析

1、 算法的定义、特征

定义：指在解决问题时，按照某种机械步骤一定可以得到问题结果的处理过程，通俗讲算法是指解决问题的一种方法或一个过程。算法是若干指令的有穷序列。

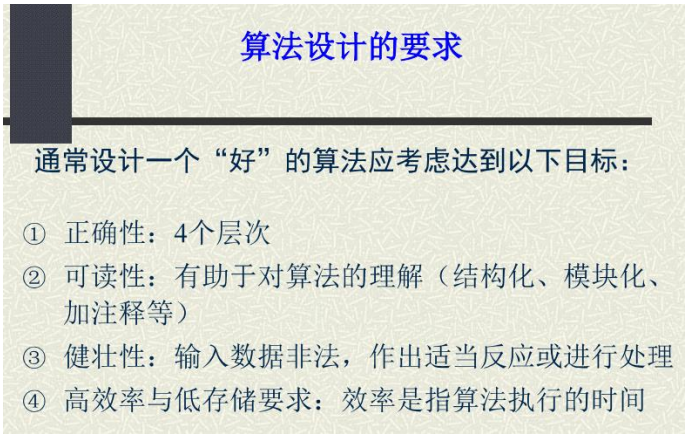




特征：①**有穷性**：算法的有穷性是指算法必须能在执行有限个步骤之后终止；②**确定性**：算法的每一步骤必须有确切的定义；③**输入项**：一个算法有 0 个或多个输入，以刻画运算对象的初始情况，所谓 0 个输入是指算法本身定出了初始条件；④**输出项**：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；⑤**可行性**：算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步骤，即每个计算步骤都可以在有限时间内完成（也称之为有效性）。

2、 好的算法的标准

评价一个好的算法应该从四个方面去考虑：算法的正确性、算法的易读性、算法的健壮性、算法运行的时空效率。



3、 算法的复杂性分析

算法复杂性 = 算法所需要的计算机资源。

影响程序运行时间的主要因素 ：

（1）程序的输入。

（2）由编译系统所产生的代码程序的质量。

（3）执行程序的计算机机器指令的性能与速度。

（4）程序所依据的算法的时间复杂度。

算法的复杂性测度主要有两个方面:

(1) 时间复杂度 T(n)

(2) 空间复杂度 S(n)

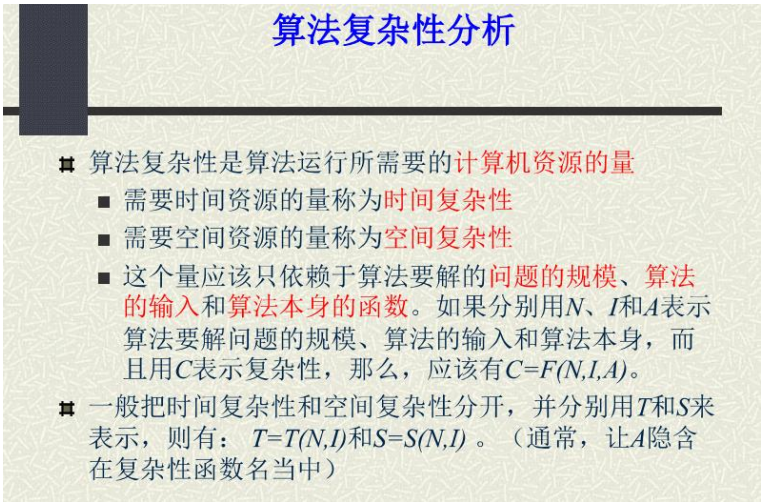
其中 n 是问题的规模（输入大小）。

算法的复杂性取决于

(1)求解问题的规模 N;

(2)具体的输入数据 I;

(3)算法本身的设计



算法复杂性（Algorithm Analysis）

= 算法所需要的计算机资源

=时间+空间

依赖于求解问题的规模、算法的输入和算法本身的函数。

C=F(N,I,A)，其中，C表示复杂性，N表示求解问题的规模，I表示算法的输入，A表示算法本身

算法分析的目的：

- 设计算法——设计出复杂性尽可能低的算法
- 选择算法——在多种算法中选择其中复杂性最低者

算法分析的意义：

- 对算法的设计和选用有着重要的指导意义和实用价值

任何可用计算机求解的问题所需的时间都与其 [填空1] 有关。

4、 算法的度量标准

度量标准分为五个:

① **时间复杂度**: 指算法运行效率高，即算法运行所消耗的时间短。

② **空间复杂度**: 指算法所需的存储空间少。

③ **难易程度**: 指算法遵循标识符命名规则，简洁，易懂，注释语句恰当、适量，方便自己和他人阅读，便于后期调试和修改。

④ **健壮性**: 指算法对非法数据及操作有较好的反应和处理。

⑤ **正确性**:指算法能够满足具体问题的需求，程序运行正常，无语法错误。

第二章 分治法

5、 递归的定义

直接或间接地调用自身的算法称为**递归算法**。

用函数自身给出定义的函数称为**递归函数**。

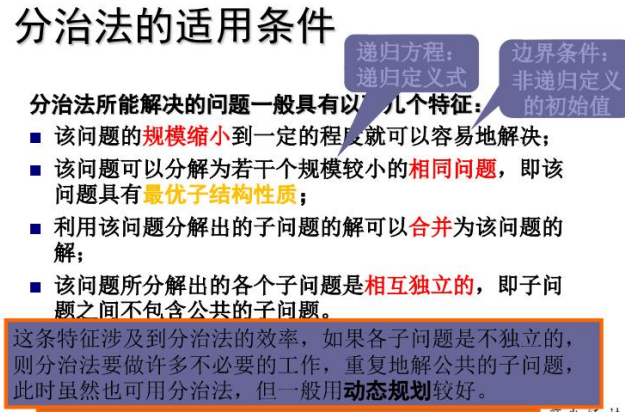
递归小结

优点：**结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来很大方便。**

缺点：**递归算法的运行效率较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。**

6、 分治的定义

就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。



7、 分治的基本思想和求解步骤

基本思想：将一个问题划分为 k 个子问题，对这 k 个子问题分别求解。如果子问题的规模仍然不够小，则再划分为 k 个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止（最优子结构性质）。将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。

求解步骤：

① **划分步**：把输入的问题划分为 k 个子问题，并尽量使这 k 个子问题的规模大致相同。平衡子问题的表现比规模不等的做法好

② **治理步**：当问题的规模大于某个预定的阈值 n0 时，治理步由 k 个递归调用组成。

③**组合步**：组合步把各个子问题的解组合起来，它对分治算法的实际性能至关重要，算法的有效性很大地依赖于组合步的实现。

8、 分治法的经典求解问题

排序算法（快速排序、归并排序）、傅立叶变换（快速傅立叶变换）

- 掌握设计有效算法的分治策略
- 掌握二分搜索技术及的递归分治实现算法
- 掌握大整数乘法的递归分治实现算法
- 掌握Strassen矩阵乘法的递归分治实现算法
- 掌握棋盘覆盖的递归分治实现算法
- 掌握棋盘覆盖的递归分治实现算法
- 掌握合并排序和快速排序的递归分治实现算法
- 掌握线性时间选择的递归分治实现算法
- 循环赛日程表

第三章 蛮力法和回溯法

9、 蛮力法和回溯法的定义

蛮力法：蛮力法是一种简单直接地解决问题的方法(暴力求解)，常常直接基于问题的描述和所涉及的概念定义。

回溯法：回溯法（探索与回溯法）是一种选优搜索法，又称为试探法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。

10、 蛮力法和回溯法的基本思想和求解步骤

蛮力法：蛮力法是指采用**遍历**（扫描）技术，即采用一定的策略将待求解问题的所有元素依次处理一次，从而找出问题的解。**依次处理**所有元素是蛮力法的关键，为了避免陷入重复试探，应保证处理过的元素不再被处理。蛮力法本质是先有策略地穷举，然后验证。简单来说，就是列举问题所有可能的解，然后去看看是否满足题目要求，是一种逆向解题方式。

回溯法：在回溯法中，每次扩大当前部分解时，都面临一个可选的状态集合，新的部分解就通过在该集合中选择构造而成。这样的状态集合，其结构是一棵**多叉树**，

每个树结点代表一个可能的部分解，它的儿子是在它的基础上生成的其他部分解。树根为初始状态，这样的状态集合称为**状态空间树**。回溯法对任一解的生成，一般都采用逐步扩大解的方式。每前进一步，都试图在当前部分解的基础上扩大该部分解。它在问题的状态空间树中，从开始结点（根结点）出发，以深度优先搜索整个状态空间。

求解方法：

回溯法将解空间看作树形结构(状态空间树)，进行深度优先遍历+跳跃式搜索

跳跃式：算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解，如果肯定不包含，则跳过该节点为根的子树的搜索，逐层向其祖先节点回溯；否则进入该子树，继续深度优先策略搜索

回溯法与暴力查找是一样的吗？

可以把回溯法和分支界限法看成是穷举法的一个改进。该方法至少可以对某些组合难题的较大实例求解。

不同点：

每次只构造候选解的一部分，然后评估这个部分的候选解，如果加上剩余的分量也不可能求得一个解，就绝对不会生成剩下的分量。

11、 蛮力法和回溯法的经典求解问题

蛮力法：选择排序、冒泡排序、顺序查找

回溯法：n 皇后问题、0-1 背包问题

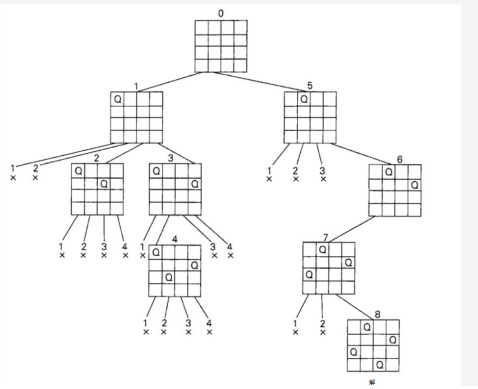
➢ 0-1背包问题

问题描述：给定n种物品和一个背包。物品i的重量是w(i)，其价值为v(i)，背包的容量为c。问应该如何选择装入背包的物品，使得装入背包中的物品的总价值最大？

注： 每个物品只能使用一次。

➢ 八皇后问题

问题描述：在一个N*N的棋盘上放置N个皇后，使其不能互相攻击。**(同一行、同一列、同一斜线上的皇后都会自动攻击)**那么问，有多少种摆法？



➢ 八皇后问题

【整体思路】
在第一行占据一个位置，接着在下一行占据一个位置，判断两个位置之间是否存在冲突。如果不存在冲突，则继续在下下行占据一个位置，判断与前面的两个位置之间是否存在冲突，如果不存在冲突则继续往下占据位置，当占据到最后一行，且占据的位置与前N-1行的位置均不冲突，那么便成功找到了一种放置皇后的方案。
如果在当前行占据的位置与前面的位置存在冲突，那么需要寻找当前行的其他位置，直到找到不冲突的位置并进入下一行寻找，或者当前行寻找完毕未发现不冲突的位置，那么则返回上上行，寻找上上行的其他位置，如果在上上行也没有找到，则继续向上寻找直到整个棋盘遍历完毕。

第四章 分支界限

12、 分支界限的定义

所谓分支就是采用**广度优先**的策略，一次搜索活结点的所有分支。为了有效的选择下一扩展节点，以加速搜索的进程。在每一处活节点处，计算一个函数值（限界函数），在当前的活结点中选择一个可行最优解作为下一拓展节点。

分支界限法按广度优先策略遍历问题的解空间树，在遍历过程中，对已经处理的每一个结点根据限界函数估算目标函数的可能取值，从中选取使目标函数取得极值的结点优先进行广度优先搜索，从而不断调整搜索方向，尽快找到问题的解。

13、 分支界限的基本思想和求解步骤

分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树。

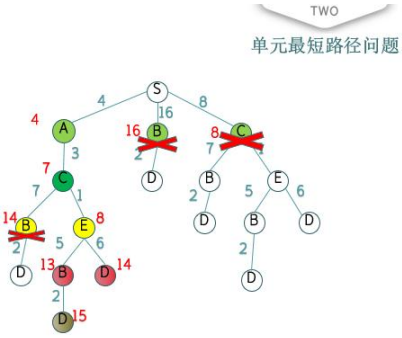
在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。

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。

分支界限法是一种搜索方法，用于求解最优化问题。其基础是回溯法，基本思路在于解节点的生成，扩展，剪枝。

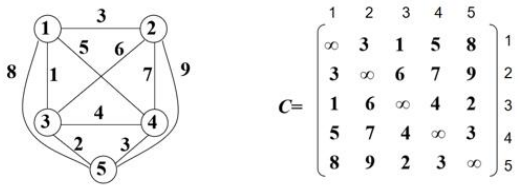
14、 分支界限的经典求解问题

0-1 背包问题、旅行商问题、单元最短路径问题。



旅行商问题

问题描述：某售货员要到若干城市去推销商品，已知各城市之间的路程（以耗费矩阵的形式表示）。从1出发求一条回路，该回路经过每个城市一次，且总的耗费（总距离）最小。



$$C=\begin{array}{c}\begin{matrix}1 & 2 & 3 & 4 & 5\end{matrix}\\ \begin{bmatrix}\infty & 3 & 1 & 5 & 8\\ 3 & \infty & 6 & 7 & 9\\ 1 & 6 & \infty & 4 & 2\\ 5 & 7 & 4 & \infty & 3\\ 8 & 9 & 2 & 3 & \infty\end{bmatrix}\end{array}\begin{matrix}1\\2\\3\\4\\5\end{matrix}$$

第五章 动态规划

15、 动态规划的定义

动态规划与分治方法类似，都是通过组合子问题的解来来求解原问题的。它是解决多阶段决策过程最优化的一种方法。把多阶段问题变换为一系列相互联系的的单阶段问题，然后逐个加以解决。

16、 动态规划的基本思想

核心思想是：将大问题划分为小问题进行解决，从而一步步获取最优解的处理算法。

动态规划算法与分治算法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。

17、 动态规划的求解步骤

1）找出最优解的性质，并刻划其结构特征。2)递归地定义最优值。3)以自底向上的方式计算出最优值。4）根据计算最优值时得到的信息，构造最优解。

18、 动态规划的经典求解问题

0-1 背包问题、最长公共子序列问题、矩阵连乘问题

第六章 贪心算法

19、 贪心算法的定义

贪心算法又称贪婪算法是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，算法得到的是在某种意义上的局部最优解。

20、 贪心算法的基本思想

贪心法的当前选择可能要依赖已经做出的所有选择，但不依赖于有待于做出的选择和子问题。因此贪心法自顶向下，一步一步地做出贪心选择。

贪心法总是做出在当前时刻看起来最优的决策，即希望通过局部最优决策导致问题的全局最优解。

21、 贪心算法的求解步骤

贪心算法一般按如下步骤进行①建立数学模型来描述问题②把求解的问题分成若干

个子问题③对每个子问题求解，得到子问题的局部最优解④把子问题的解局部最优解合成原来解问题的一个解

22、 贪心算法的经典求解问题

背包问题

第七章 随机算法与近似算法

23、 随机算法的定义

一个随机算法是一种算法，它采用了一定程度的随机性作为其逻辑的一部分。该算法通常使用均匀随机位作为辅助输入来指导自己的行为，超过随机位的所有可能的选择实现了“平均情况下的"良好业绩的希望。从形式上看，该算法的性能将会是一个随机变量，由随机位决定;因此无论是运行时间，或输出(或两者）是随机变量。

在常见的实践中，随机化算法是使用近似的伪随机数发生器代替随机比特的真实来源的;这样的实施可以从预期的理论行为偏离。

24、 近似算法的定义

近似算法是一种处理优化问题 NP 完全性的方式，它无法确保最优解。近似算法的目标是在多项式一时间内尽可能地接近最优值。

它虽然无法给出精确最优解，但可以将问题收敛到最终解的近似值。其目标满足以下三个关键特性:①能够在多项式时间内高效运行;②能够给出最优解;③对于每个问题实例均有效。

25、 启发式算法的定义

启发式算法是相对于最优化算法提出的。一个问题的最优算法求得该问题每个实例的最优解。启发式算法可以这样定义:一个基于直观或经验构造的算法，在可接受的花费(指计算时间和空间)下给出待解决组合优化问题每一个实例的一个可行解，该可行解与最优解的偏离程度一般不能被预计。

26、 随机算法、近似算法、启发式算法的应用场景和具体应用

随机算法：场景有单位年会抽奖、随机挑观众互动、学生上课随机挑选回答问题。

常用应用：传统的快排序算法、随机快排序算法、模式匹配

近似算法：目前大规模的 NPC 问题我们无法通过计算得到，因此我们需要通过损失一部分精度的做法来找到多项式的近似算法。常用应用：顶点覆盖问题的近似算法、旅行售货员问题近似算法、集合覆盖问题的近似算法、子集和问题的近似算法

启发式算法：一般用于解决 NPhard 问题。常用算法有：模拟退火算法(SA)、遗传算法（GA)、蚁群算法(ACO)、人工神经网络(ANN)

第八章 计算复杂性理论

27、 P、NP、NPC、NP hard 类问题的定义

P:能在多项式时间内解决的问题。

NP:不能在多项式时间内解决或不确定能不能在多项式时间内解决，但能在多项式时间验证的问题。

NPC:NP 完全问题，所有 NP 问题在多项式时间内都能约化(Reducibility)到它的 NP 问题，即解决了此 NPC 问题，所有 NP 问题也都得到解决。

NP hard:NP 难问题，所有 NP 问题在多项式时间内都能约化(Reducibility)到它的问题(不一定是 NP 问题)。

28、 P、NP 和 NPC 的关系

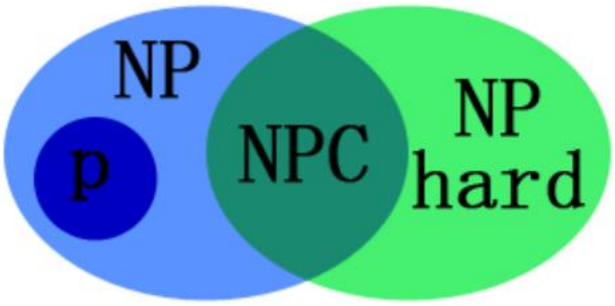


图1 P NP NPC NPhard关系的图形表示

说明:

1.P 问题属于 NP 问题，NPC 问题属于 NP 问题。

2.NPC 问题同时属于 NP hard 问题，是 NP 与 NPhard 的交集。

背包问题
旅行商问题
N 皇后问题