# 1 课程基本信息

① 课程目的：

掌握算法设计、分析的基本技术；掌握计算复杂性理论的基本概念；学习应用算法理论求解实际计算问题

# Programs = Algorithms + Data Structures

② 成绩评定：

|  |  |
| --- | --- |
| 平时成绩 |  |
| 实验成绩 | 上机 |
| 课程报告 | 读一篇2022年发表的计算问题求解的论文(注意**在线发表**和正式发表的时间)，并撰写报告，报告至少包含**问题描述、求解方法，可能的改进**三部分。  # 在线发表为准 |

③ 课程内容

|  |  |
| --- | --- |
| 算法分析的基本方法和理论 | 算法设计的基本技术 |
| 算法复杂性的估计、问题复杂性的上下界、计算复杂性的概念、Cook定理、NP完全性理论及其应用 | 分治、回溯法、动态规划、线性规划、整数规划、贪心法、启发式算法、元启发算法 |
| # pintia — 刷题插件；上机时间2：30 ~ 5：30 | |

④ 参考书

1、introduction to Algorithms (Third Edition), Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, The MIT Press 2009.

# 老师建议备一本当作字典用

2、The Art of Computer Programming, Donald Knuth, Pearson Education, 1997.

# 2 引言

## 2.1 概述

① 算法(algorithm)的定义

求解问题的方法。在数学和计算机科学中，算法是一系列指令，通常用来解决一类问题。

In mathematics and computer science, an algorithm is a sequence of instructions, typically to solve a class of problems.

# algorithm词源？

自于9世纪波斯数学家**花拉子米**[al-Khwarizmi]，拉丁语转写为Algoritmi，英语转写为algorithm

**花拉子米**撰写的《印度数字算术》被翻译成拉丁语“Algoritmi denumero Indorum，在欧洲广泛传播。它的现代含义是19世纪引入的。

② 世界上第一个算法

欧几里得算法。又称**辗转相除法**，用以求最大公约数的算法，首次出现于公元前300年的欧几里得的《几何原本》，在中国可以追溯至东汉出现的《九章算术》。

# 假如需要求 1997 和 615 两个正整数的最大公约数,用欧几里得算法，是这样进行的：

1997 / 615 = 3 (余 152)

615 / 152 = 4(余7)

152 / 7 = 21(余5)

7 / 5 = 1 (余2)

5 / 2 = 2 (余1)

2 / 1 = 2 (余0)

至此，最大公约数为1

以除数和余数反复做除法运算，当余数为 0 时，取当前算式除数为最大公约数，所以就得出了 1997 和 615 的最大公约数 1。

③ 算法研究的两个理论

|  |  |
| --- | --- |
| 可计算性理论[2.2] | 计算复杂性理论[3] |

④ 常见的研究问题

|  |  |
| --- | --- |
| 投资问题 | 有限的资源下使收益最大化 |
| Hanoi塔问题 | 指定规则下解决一个问题的方法 |
| 搜索问题 | 指定数组下搜索目标数值的位置 |
| 排序问题 | 按递增顺序排列n个数 |
| 选择问题 | 指定数组下选择某个需求的数 |

⑤ 算法研究的重要性

好的算法，可以提高求解问题的效率，可以节省存储空间。衍生出了如下的几个课题：

|  |  |
| --- | --- |
| **背景** | **课题** |
| 问题 → 寻找求解算法 | 算法设计技术 |
| 算法 → 算法的评价 | 算法分析技术 |
| 算法类 → 问题复杂度的评价 | 问题复杂性分析 |
| 问题类 → 能够求解的边界 | 计算复杂性理论[2.3] |

## 2.2 可计算性理论

# 可计算性是不依赖于计算模型的客观性质

① 研究目标

确定什么问题是可计算的，即存在求解算法

② 合理的计算模型

归函数、Turing机、入演算、Post系统等

条件：(1) 计算一个函数只要有限条指令；(2) 每条指令可以由模型中的有限个计算步骤完成；(3)指令执行的过程是确定的

# 简单来说，邱奇-图灵论题认为“任何在算法上可计算的问题同样可由图灵机计算”。

③ 三种概念

理论上可计算：至少存在指数时间的算法

现实上可计算：具有多项式时间的算法——多项式时间可解的

理论上不可计算：图灵机不可计算函数，图灵机根据设定好的规则运行，并不一定能够最终自动停下来

# 如忙碌的海狸问题

## 2.3 计算复杂性理论

① 核心内容

算法复杂度——算法所使用的时间、空间的估计

问题复杂度——估计问题的难度

② 术语和重要概念

1、问题

需要回答的一般性提问，通常含有若干参数问题描述所包含的内容，如对问题参数的一般性描述，解满足的条件。

2、算法

(1) 非形式化定义

有限条指令的集合；指令集确定了解决某个问题的运算或操作的序列；输入个数大于等于0；输出个数大于0

(2) 基于图灵机的形式化定义

对所有的有效输入停机的图灵机

(3) 算法的描述

伪代码或者具体的编程语言

(4) 算法A解问题Р

把问题P的任何实例作为算法A的输入，A能够在有限步停机，并输出该实例的正确的解

3、算法的时间复杂度

针对问题选择基本运算，将基本运算次数表示为输入规模的函数

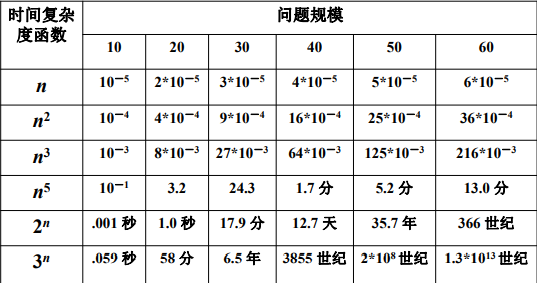
最坏情况下的时间复杂度：算法求解输入规模为n的实例所需要的最长时间W(n)

平均情况下的时间复杂度：算法求解输入规模为n的实例所需要的平均时间A(n)

4、多项式时间的算法与指数时间的算法

至少指数级：

多项式级：

 对数多项式级：

# 3 计算复杂性理论

## 3.1 问题

① 问题的主要分类

就难解性而言，问题的主要分类有如下三种：

1、易解问题

# 优化问题

一个称为多项式时间的算法必须符合：算法于最差情况(Worst-case)的时间复杂度**以多项式函数**为限。

若一个问题存在多项式时间的求解算法，则称该问题是易解问题。

# 有n100时间复杂度的算法，那问题还算易解吗? 是！这样的算法有进步空间，只要它不是指数，它可以变得更高效。

2、没被证明的难解问题

在计算机领域中，若无法在最差情况下，以多项式时间的算法来解决某个问题，该问题就被称为**难解问题**。即，一个难解问题，必须没有任何多项式时间的算法可以解它。

但是，如果有一个问题在最差情况下，目前还找不到一个多项式时间的算法解它，但是也**无法保证未来**就找不到多项式时间的算法来解这个问题，则无法证明该问题是Intractable。

# 如，旅行商问题、装箱问题、背包问题、...

3、已被证明的难解问题

时间复杂度被证明为指数复杂度(以上)的问题。

# 如:汉诺塔问题

不存在有解决问题之算法的问题。

# 如:程序停止问题(Halting Problem)，是否有算法能够判定某个图灵机在某个输入下是否停机

② 决策问题

# 优化问题均可以找出一个与之对应的决策问题(判定问题)

除了过去所讨论过的各种优化问题(如最短路径､最小生成树､旅行商问题)以外，尚有另外一种形态的问题：**决策问题**。此类问题输出的答案非常简单，就是“yes”或“no”两者之一。

(1) The Satisfiability Problem (布尔可满足性问题，SAT)：给一个布尔函数E，我们对存在于此函数E中的一些变量分别赋值True或False，使这个函数结果为True。

(2) The partition problem(分割问题):给予一组正整数集合S={a1, a2，…，an,}，是否可将其分割成两个子集合S1与S2，而S1的总和与S2的总和相等。

# 例：给定一个图和一个常数。求最小生成树[优化问题]，最小生成树是否小于常数[决策问题]

如果要解某个优化问题的决策版本已经很困难了，则该问题的优化版本一定更难解决。以下课程内容所讨论到的问题，若无特别说明，皆以“决策版本”的问题为主。优化问题和决策问题的本质是一样的。

## 3.2 图灵机

① 分类

有穷状态自动机 → 下推自动机 → 线性有界非确定图灵机 → 图灵机

# 从低级到高级的变化

② 确定性图灵机(deterministic Turing Machine)

1、背景知识

用机器来模拟人们用纸笔进行数学运算的过程，图灵把人们的运算过程看作下列两种简单的动作：

(1) 在纸上写上或擦除某个符号;

(2) 把注意力从纸的一个位置移动到另一个位置。

2、初级定义

为了模拟人的这种运算过程，图灵构造出一台假想的机器，即图灵机，该机器由以下几个部分组成:

(1) 一条纸带。纸带被划成格子，每个格子上包含一个符号。

(2) 一个读写头。读写头可以在纸带上左右移动，它能读出当前所指的格子上的符号，并能改写。

(3) 一个状态寄存器。保存图灵机当前所处的状态。

(4) 一套控制规则。根据当前状态以及当前符号来确定读写头下一步的动作，并改变状态和符号。

3、正式定义

图灵机的正式定义为七元组(*Q*, : ; ，, , )，其中，

Q表示有穷状态集合, : ; ，分别表示起始、接受、拒绝状

态

为非空有穷输入字母表

是非空有穷带字母表，含特殊的空白符

是状态转移函数，表示根据当前状态以及当前符号来改变状态和符号，并确定读写头左移L或右移R。

# https://math.hws.edu/eck/js/turing-machine/TM.html

4、确定性图灵机的定义

在运行过程中每一个步骤的运算都需要被唯一定义，因此产生的结果也是唯一的。

由于在此类计算器器运作的算法在处理问题时，每一步只有一件事，因此，只要有一个处理器即可，故容易实现。

# 现代计算机就是一种确定性图灵机。

③ 通用图灵机

# 邱奇图灵论题：任何在算法上可计算的问题同样可由图灵机计算。

对于任意一个图灵机，因为它的描述是有限的，因此总可以用某种方式将其编码为字符串，我们用<M>表示图灵机M的编码。

可以构造出一个特殊的图灵机，接受任意一个图灵机M的编码<M>，然后模拟M的运作，这样的图灵机称为**通用图灵机**。

# 现代电子计算机就是这样一种通用图灵机

④ 程序停止问题(Halting Problem)

是否有算法能够判定某个图灵机在某个输入下是否停机？没有。

⑤ 非确定性图灵机(Non-deterministic Turing Machine)

# 仅是理论模型，实际上并不存在此种机器。

1、定义

和确定型的不同之处在于，在计算的每一时刻，根据当前状态*q*和读写头所读符号*x*，机器存在多种状态转移方案。

例如，(*q*, *x*) = {(*q*1, *x*1, *d*1), (*q*2 ,*x*2 ,*d*2), ..., (*q*n, *x*n, *d*n)}，则将产生*n*个分支进行下一步计算，直到某个分支上停机为止。

只要有任意一个分支进入接受状态，则称接受；反之则称拒绝。非确定性图灵机必须是无矛盾的，即不能有某个分支接受而同时另一个分支拒绝。

2、定理

对于任意一个非确定型图灵机M，存在一个确定型图灵机M’，使得它们的计算能力相等。

# 计算能力相等指计算结果相同；速度不一定相同。

3、执行步骤

(1) 猜测阶段(Guess)：由于没有一个既定的程序来从事此阶段的猜测工作，因此本阶段是非确定性的。

# 对于本阶段，我们只知道: 如果一个问题有正确解的话，此阶段一定可以将这个正确解给猜出来; 反之，若该问题没有正确解的话，则此阶段就会随便给解答。

(2) 验证阶段(Verification)

将上一阶段所猜出来的结果加以验证是否为真(True)

# 类别理解：无穷多个CPU在并行运行

4、应用

以一个具有n个变量之布尔函数E之满足问题(SAT)为例：

如果此问题是使用确定性图灵机，则时间复杂度为O(2")

如果此问题是使用非确定性图灵机，则时间复杂度为O(n)

## 3.3 多项式时间归约(polynomial-time reduction)

① 定义

图示, 示意图

描述已自动生成若有两个问题Q1和Q2，其解集合分别为L1和L2:如果存在一个多项式时间可计算的函数*f*(x)，使得对所有x而言， Q1可以多项式时间归约成Q2。如图所示：

Q1 可以归约为Q2，那么表示Q1问题可以用Q2问题的算法求解。

Q1问题可以由处理Q2问题的算法所解决。若Q2问题有一个有效率的算法，可以在多项式时间内将Q2问题给解掉，表示Q1问题也一定可以在多项式时间内被解掉。

② 意义

Q2问题比Q1问题难

若Q2有解，则Q1就有解

归约具有传递性

想要证明Q1和Q2一样难，则它们必须能够相互归约

③ 案例

一元二次方程一元三次方程

二元函数三元函数

哈密顿环问题TSP问题

# 哈密顿环指能将图中所有顶点各拜访一次并回到原点的路径。哈密顿环问题:给定图G=<V,E>，G是否包含一个哈密顿环?

# TSP判定问题：给定常数B，是否存在一种巡回路径长度≤B?

## 3.4 四类问题

# 重点：前面三个小章节就是为了引出这个小章节

① P类问题

可由确定性图灵机在多项式时间内求解的决策问题。

② NP类问题(Non-deterministic Polynomial)

可由确定性图灵机在多项式时间内**验证**的决策问题。

可由非确定性图灵机在多项式时间内**求解**的决策问题。

# 这两个定义是等价的。至于为什么等价？不重要！

由于确定性图灵机为非确定性图灵机的一个特例，且容易找到答案也会容易验证答案。因此，P可视为NP的一个特例：

# 至于P = NP，还需要验证

③ NPC类问题(Non-deterministic Polynomial Complete)

1、定义

问题Q∈NPC，当且仅当Q是一个NP问题(决策问题)，且任何NP问题Q'都可以在多项式时间内归约到Q(Q'Q)。

NPC是NP中最难的一类问题，如果有一个NPC问题找到了多项式时间复杂度的确定性求解算法，则所有NP问题都有了多项式时间的确定性求解算法，即P=NP。

# 此类问题至今仍未找到一个多项式复杂度的确定性算法，且一般相信没有多项式复杂度的确定性算法存在，即P≠NP。

2、证明某个问题为NPC问题的意义

此类问题至今仍未找到一个多项式复杂度的确定性算法，且一般相信没有多项式复杂度的确定性算法存在。

3、**库克定理**(Cook’s Theorem)

全世界第一个被证明出来的NPC问题。

# 具体的过程请看看B站视频或者维基百科。因为太难理解了。GOD DAMN IT！

4、3-SAT问题

可以被证明为是NPC问题。

④ NP-hard类问题

问题Q∈NP-hard ，当且仅当存在一个NPC问题可以在多项式时间内归约到Q。

# NPC ∈ NP-hard

NP-hard问题并不局限于决策性问题，也适用于优化问题、搜索问题。

图示, 维恩图

描述已自动生成 一般而言，人们相信上述复杂性类之间的关系如下：

⑤ 卡普二十一个NPC问题

# 可以自己了解一下

# 3 递归、分治、回溯

本章介绍三种算法的基本思想。

## 3.1 递归(Recursion)

① 定义

直接或间接调用自身的算法称为递归算法。

② 要素

边界条件 and 递归方程

# 常见的递归：阶乘、斐波拉契数列、汉诺塔问题、整数划分问题

③ 特点

1、优点：结构清晰，可读性强，可用数学归纳法来证明算法的正确性，因此它为设计算法带来很大方便。

2、缺点：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。

3、解决方案：消除递归调用，使其转化为非递归算法。

(1) 用递推来实现递归函数。这种方法在时空复杂度上均有较大改善，但其适用范围有限。

(2) 采用一个用户定义的栈来模拟系统的递归调用工作栈。该方法通用性强，但本质上还是递归，相当于人工做了本来由编译器做的事情。

# 用递推来实现递归函数。这种方法在时空复杂度上均有较大改善，但其适用范围有限。

## 3.2 分治(Divide and Conquer)

① 定义

将要求解的较人规模的问题分割成k个更小规模的子问题。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。

② 分治法的适用条件

该问题的规模缩小到一定的程度就可以容易地解决

# 因为问题的计算复杂性一般是随着问题规模的增加而增加，因此大部分问题满足这个特征。

该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。

# 这条特征是应用分治法的前提，它同样是大部分问题满足的特征。

利用该问题分解出的子问题的解可以合并为该问题的解;

# 能否利用分治法完全取决于问题是否具有这条特征，如果具备了前两条特征，而不具备第三条特征，则可以考虑贪心算法或动态规划。

该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

# 这条特征涉及到分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然也可用分治法，但一般用动态规划较好。

③ 分治法的基本步骤

文本

描述已自动生成包括三个基本步骤：divide､conquer､merge。算法框架通常为：

在用分治法时，最好使子问题的规模大致相同，即将一个问题分成大小相等的k个子问题是行之有效的。

由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。

# 分治法是很多并行模型的基础：分治法是MapReduce、Hadoop、openMP、Map-Reduce等大数据并行框架的核心思想。

④ 复杂度

分治法将规模为n的问题分成a个规模为n/ b的子问题去解

设分解阀值n=1，且conquer规模为l的问题耗费1个单位时间。

再设将问题divide为i个子问题以及用merge将a个子问题的解合并为原问题的解需用个单位时间。

用T(n)表示该分治法解规模为P=n的问题所需的计算时间，则有以下递推关系式：

⑤ 求解

1、代入法：猜测解的形式，并用数学归纳法求出解中的常数

2、递归树方法：画出递归树，并确定整棵树的代价

3、Master定理：

# 本章只介绍Master定理

我们重点关注的时间复杂度。

case 1：只能求出上界。

此时：

case 2：能求出上下界

此时：

case 3：只能求出下界

且对于某个常数和所有充分大的*n*有

此时：

⑤ 经典案例

归并排序、快速排序、最大子数组问题、strassen矩阵乘法、大整数的乘法

下面介绍最大子数组问题：在一个数组中找出最大的非空连续子数组

|  |
| --- |
| 比如你获得了一个投资某个股票的机会，并且你已经准确知道了将来几天这一只股票的相对于前一天的差值，比如为[ 13,-3,-25,20,-3,-16,-23,18,20,-7,12,-5,-22,15,-4,7]。  1、暴力解法：两层循环遍历，时间复杂度为O(n2)  2、分治法求解：如果把一个数组从中间点分为左右两个子数组，那么最大子数组可能存在的位置可能存在的情况有三种:  (1) 最大子数组在左边的数组中  (2) 最大子数组在右边数组中  (3) 最大子数组跨越了左右两个子数组 |

下面介绍strassen矩阵乘法：

|  |
| --- |
| 传统两个矩阵相乘的时间复杂度显然为n3  分治算法：将矩阵A，B和C中每一矩阵都分块成4个大小相等的子矩阵。  为了降低时间复杂度，必须减少乘法的次数;依次引入如下7个辅助矩阵: |

下面介绍大整数乘法问题：请设计一个有效的算法，可以进行两个n位大整数的乘法运算

|  |
| --- |
| 传统方法：O(n^2)  Karatsuba算法：O(n1.59) |

## 3.3 回溯(backtrack)

① 定义

在分步解决过程中，当它发现现有的部分解不能得到有效的完整解时，将回溯取消上一步或多步的计算，通过其它可能的部分解再次尝试寻找问题的解。

回溯法是种暴力搜索算法，最坏情况下会导致指数时间复杂度。

回溯法可以找出所有解，适用于求解约束满足问题和组合优化问题。

② 实例

4后问题：如何在4×4的棋盘上无冲突的摆放4个呈后棋子。在国际象棋中，皇后的移动方式为横竖交叉的，因此在任意一个皇后所在位置的水平、竖直、以及45度斜线上都不能出现皇后的棋子。每一行只能放一个皇后。求出符合要求的情况的个数。

解表示成一个4维向量<x1, x2, x3, x4>，其中x,表示第i个皇后放置在i行、x;列。搜索空间为4叉树。

# 5 贪心、近似、启发式算法

## 5.1 贪心算法

① 概念

贪心算法是指，在对问题求解时总是做出在当前看来是最好的选择(即，不从整体最优上加以考虑)，所做出的仅是在某种意义上的局部最优解。

过程：逐步开展，以当前情况为基础根据某个度量标准作最优选择。

② 特点

优点：省去了为找最优解要穷尽所有可能而必须耗费的大量时间，每做一次贪心选择仅为求解原问题简化后的一个规模更小的子问题。

难点：每步采用局部最优解，但产生的最终解不一定全局最优。

③ 设计步骤

如何寻找能产生问题最优解的最优度量标准?

(1) 将优化问题转化为如下形式:对其做出一次选择后，只剩下一个了问题需求解

(2) 证明贪心选择性质:可通过做出局部最优选择来构造全局最优解。

正确性证明：归纳法，反证法

# 等价:原问题县有拟阵属性。

(3) 证明最优子结构性质:问题的最优解所包含的子问题的解也是最优的

能获得最优解的贪心算法VS动态规划：

动态规划的两个要素：最优子结构、子问题重叠

贪心算法的两个要素：最优子结构、贪心选择性质

贪心法的时间复杂度和空间复杂度低。每个贪心算法之下，总有一个更繁琐的DP算法可以完成同样的任务。

## 5.2 近似算法

退而求其次，在多项式时间复杂度内，找寻一个能保证近似率的近似解,而非最佳解。

## 5.3 启发式算法

是一类不能保证最优或完美，但仍可找到一个“足够好”的解的的问题求解方法。

没有理论上的性能保证，仅通过实验进行性能分析。

部分启发式算法可以理解为:引入启法信息，问题求解的每一步骤，相当于对启法信息的贪心选择。