



计算机专业入门小指南

The introduction to the Computer Science

主编: Sukuna

组织: 华中科技大学计算机科学与技术系

时间: Dec. 4, 2022

版本: Nahida v2.2.3

网站: www.sukunahust.com



注意: 本书仅对要从事计算机研究工作, 指保研、出国和考研的入门参考之用



作者简介

主编

Sukuna

华中科技大学计算机科学与技术专业学生，普通学生，主要编写第3、4、5、7章。

特别鸣谢

生产队的大萝卜

华中科技大学计算机学院副院长，锐评了我这本书，提供了非常多的指导意见，才有了现在的v2.x

主要参加者

Anti Mage

华中科技大学计算机科学与技术专业学生, 主要编写人工智能的内容. 已经保研到清华大学深圳研究院. 保研期间斩获 BUAA、ZJU、RMC 等高校的 offer.

Deiky

华中科技大学电子信息与通信专业学生, 有 SCI 一区一篇, 主要负责硬件部分的编写. 拿到了 THU、ZJU、SJTU、FDU 等高校的 offer, 现在放弃保研到 HK 读书了.

参加者

XiaoNanBei, 小南北, 华中科技大学计算机系学生, 回答了问题.

RHR, 华南理工大学软件工程系学生, 现在在华为深圳工作, 回答了问题.

Henry, 华中科技大学计算机系学生, 回答了问题.

Harvey, 华中科技大学计算机系学生, 回答了问题.

north, 华中科技大学计算机系学生, 帮忙修改了数学部分的内容.

shinnku, 华中科技大学信息安全专业学生, 帮忙修改了数学部分的内容.

Scott, 吉林大学数学系学生, 帮忙修改了数学部分的内容.

Alexander Snow, 哈尔滨工业大学学生, 帮忙修改了数学部分的内容.

charname, 浙江大学软件工程学院学生, 帮忙修改了算法部分的内容.

联系方式

欢迎对这本书提供意见, 也欢迎指出这本书的相关错误, 如果您有任何意见和建议, 可以通过下面的联系方式联系我.

QQ:1064687807.

博客:www.sukunahust.com





前言

简单介绍

作者有一天突发奇想，想写一个小册子，带着高中生或者是大一学生，甚至是不是计算机系的学生简略地了解一下计算机系应该或者是需要了解的知识。但是由于个人知识水平和精力有限，对于每一部分的知识作者只能简略地介绍基本内容和学习思路。读者想要对此了解更多，可以阅读作者推荐的阅读书目。

这个时候读者就会有疑问，作为计算机系的学生，我们要在大三前学习那么多专业的知识吗？其实不必，计算机科学有许许多多的方向，每个方向都需要一定的专业知识。每个方向可以面向计算机的不同层级。

那么计算机有哪些层级呢？拿程序来举例子吧，顶级的应用程序在计算机中首先用文本文件的形式存储，接着就会交付给编译器，编译器会把程序编译成汇编代码，汇编代码经过汇编程序和链接程序组合成可执行文件，可执行文件会被操作系统调度，开始执行，最后的执行就是在 CPU 上进行运算和传输。所以说要根据具体的层级进行学习。学习基础的知识，在属于你的层级或者说领域钻研就好了。

本书的使用方法

本书的内容比较多，东西比较多。现在在这里简单介绍一下本书的主要构成。首先会在前面用大纲的方式介绍一下计算机的基础知识和计算机学院开设的课

程,接下来的部分,我会用三章简单地介绍计算机三个方面的基础知识,分别是数学基础、编写代码的基础和简单的数据结构和算法,每一章都给出了简单的讲解,在每一章的最后我们给出了一些学习资料,大家可以配合着对学习资料的介绍自行探索。下一部分我会用三章分别介绍计算机研究的三个非常重要的方向,第一个就是系统,第二个是关于 AI 的研究,最后一个就是理论计算机的研究,在这一部分我们会提纲挈领地介绍学习路径,并且附上学习资料。这一部分主要是介绍学习路径,由于篇幅所限,我们不可能展开来详细地讲。

对于这本书,对于第 1 部分和第 2 部分的知识,大家一定要好好了解,在第 3 部分的知识中选择一个来进行了解。

最后本书给出了一些拓展知识,下面对于这些拓展知识做一点说明:

拓展的知识选读,您不读或者不读对于阅读前面的内容没有任何影响,如果您对计算机的数学基础有更加浓厚的兴趣,不妨看看附加数学的部分,如果您对计算机的硬件感兴趣,就像洛佳学长一样从事底层硬件的研究,不妨可以看看附加电路知识这一部分进行了解,这一部分在简单介绍电路知识的同时还添加了和之前一样的回顾与展望。

总之,学好课内的东西,在课外的时间阅读本书,并且根据自己的兴趣阅读课外的资料是极好的。

提问的艺术

作为一名计算机系的学生,我们几乎不可能靠自己学会知识,很多配置环境的问题也会遇到很多 bug 和困难,这个时候势必要去请教别人。这对我们跑通代码有积极的作用。身为计算机系的学生,遇到不懂的问题如果有方法能搞懂,那么可以说 99% 的问题都能解决。

搜索引擎

遇到问题第一时间不是求助别人,而是求助搜索引擎,搜索引擎并不会嫌弃你一直问个不停,所以说凡事第一点就是去搜索引擎去解决。这里推荐你使用科学上网的方式去谷歌搜索,相较于国内的百度,谷歌能给你更加专业的体验。

那么如何更加高效地使用搜索引擎呢?我在这里提几点。

- 可以把出错的语言标注,比如说 C++, 比如说 Java 等。
- 报错信息非常重要,报错信息提示了为什么你会错,这对解决问题非常重要。

著名教授辜希武曾在群里面说:“你的报错信息呢?你这样问问题,同学们很难帮到你。”可见报错信息之重要性。



- 添加上你的环境, 比如说 Ubuntu20.04、JDK13 等, 环境越详细越好.

神奇的网站

这里推荐一个神奇的网站, 这里有人会说 CSDN 了, 诚然, CSDN 可以解决很多问题. 这里我推荐一个网站叫做 stackoverflow. 在 stack overflow 上你可以与全世界的程序员交流, 在交流的时候顺便锻炼你的英语能力. 很多 BUG 你会在 stack overflow 上找到解答.

github

GitHub 是最受欢迎的开源软件开发平台之一。我们课程中提到的很多工具, 从 **vim** 到 **Hammerspoon**, 都托管在 GitHub 上。向你每天使用的开源工具作出贡献其实很简单, 下面是两种贡献者们经常使用的方法:

- 创建一个**议题 (issue)**。议题可以用来反映软件运行的问题或者请求新的功能。创建议题并不需要创建者阅读或者编写代码, 所以它是一个轻量化的贡献方式。高质量的问题报告对于开发者十分重要。在现有的议题发表评论也可以对项目的开发作出贡献。
- 使用**拉取请求 (pull request)** 提交代码更改。由于涉及到阅读和编写代码, 提交拉取请求总的来说比创建议题更加深入。拉取请求是请求别人把你自己的代码拉取 (且合并) 到他们的仓库里。很多开源项目仅允许认证的管理者管理项目代码, 所以一般需要**复刻 (fork)** 这些项目的上游仓库 (**upstream repository**), 在你的 GitHub 账号下创建一个内容完全相同但是由你控制的复刻仓库。这样你就可以在这个复刻仓库自由创建新的分支并推送修复问题或者实现新功能的代码。完成修改以后再回到开源项目的 GitHub 页面**创建一个拉取请求**。然后认证的管理者审批你的拉取请求, 审批通过了你就可以把你的修改上传到开源仓库了。





广告

欢迎各位计算机学院的同学参加华中科技大学华为智能基座社团月赛，这个月赛能为非 oier 提供一次比拼的机会。

- 1、比赛时间：每个月的中旬。
- 2、比赛内容：C 语言基础，数据结构和算法，其中 A 题考察 C 语言的基础语法，B 题和 C 题考察基本的数据结构，D 题考察基本的算法基础。
- 3、比赛用 PDF 将会于考试前 2 天上传，PDF 将参考 ICPC 的展现形式。PDF 的打开密码会在考试前 3 分钟公布。请大家比赛时认真阅读 PDF 上第一页和第二页的相关事宜。
- 4、赛制：IOI 赛制。赛程阶段可以实时看到分数。注意：参赛只可使用 C/C++，OJ 地址：<http://139.196.214.121>. 届时可以上 OJ 测试环境。
- 5、公平起见，本比赛仅限计算机学院 21 级和 22 级参加，其中 NOIP 或 CSP-J/S 二等奖及以上同学暂时不支持参加排名，即允许参加考试但是不计入总排名。
- 6、奖励和其他相关事宜可以加群:607334575.



目录

作者简介	i
前言	iii
广告	vi

1

第 1 部分 * 基本介绍

第 1 章 计算机的基础知识大纲	2
第 2 章 计算机学院专业必修课基本介绍	4

2

第 2 部分 * 计算机通用知识

第 3 章 数学基础	9
3.1 数学分析.....	9
3.2 线性代数.....	13
3.3 概率论.....	18
3.4 算法的复杂度标记.....	21
3.5 组合数学.....	23

3.6 图论.....	31
3.7 生成函数.....	40
3.8 差分方程的处理.....	41
3.9 回顾和展望.....	50
第 4 章 计算机开发基础知识	52
4.1 Linux 基本知识	52
4.2 数据整理技巧.....	57
4.3 Command Line 环境.....	60
4.4 代码调试和测试.....	64
4.5 语言以及代码编写平台.....	68
4.6 项目管理.....	69
4.7 操作系统的选择.....	70
4.8 数据结构和算法大纲.....	71
4.9 如何学习算法和数据结构?.....	74
第 5 章 常见工具介绍	76
5.1 Shell 脚本	76
5.2 Vim:Command Line 文本编译器	83
5.3 Git: 版本控制工具.....	86
5.4 Makefile & cmake: 构建系统	91
5.5 tmux: 终端多路复用	92
5.6 apt&homebrew: 依赖下载器.....	93
5.7 回顾与展望.....	93



第6章 AI:Artificial Intelligence	98
6.1 基础知识.....	98
6.2 深度学习.....	100
第7章 System	104
7.1 学习大纲.....	104
7.2 理论学习方法.....	105
7.3 实践学习方法.....	109

第A章问题回答	111
A.1 课程相关.....	111
A.2 科研、保研相关.....	112
A.3 就业相关.....	115
第B章基本算法的简要介绍	117
B.1 排序.....	117
B.2 查找.....	124
B.3 动态规划.....	128
B.4 贪心算法.....	130
B.5 搜索算法.....	131
第C章附加数学知识	139
C.1 图灵机和 λ 演算	139
C.2 凸优化.....	142
C.3 信息论.....	144



C.4 数值分析 145

C.5 数论 146

第 D 章附加电路知识 153

D.1 硬件设计基础 153

D.2 电路理论 154

D.3 模拟电子技术 158

D.4 数字电子技术 160

D.5 信号与线性系统 161

D.6 回顾与展望 165

第 E 章其他计算机相关知识 167

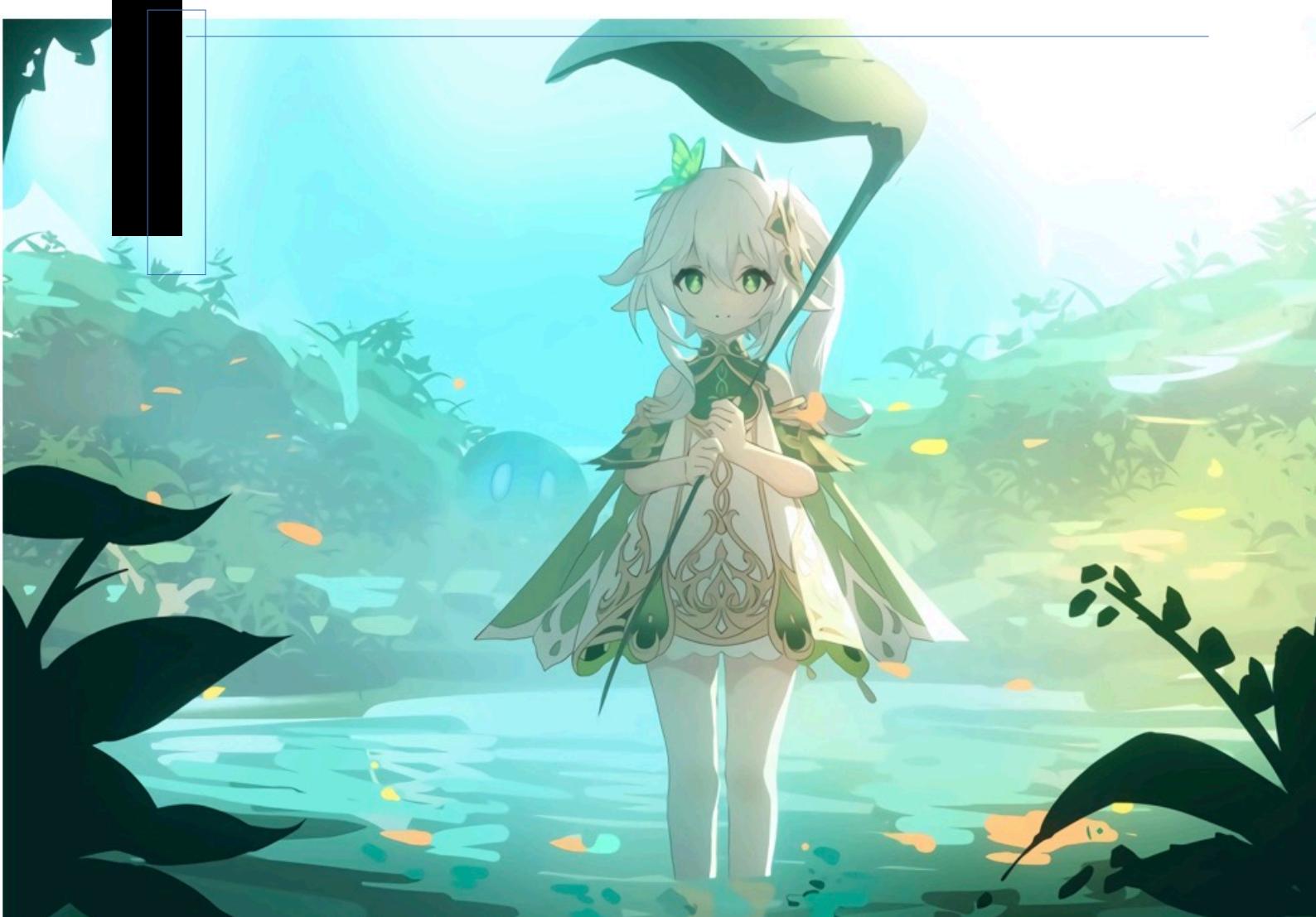
E.1 Linux 系统目录结构 167

E.2 Shell 语言运算符 170

第 F 章数学部分答案 173

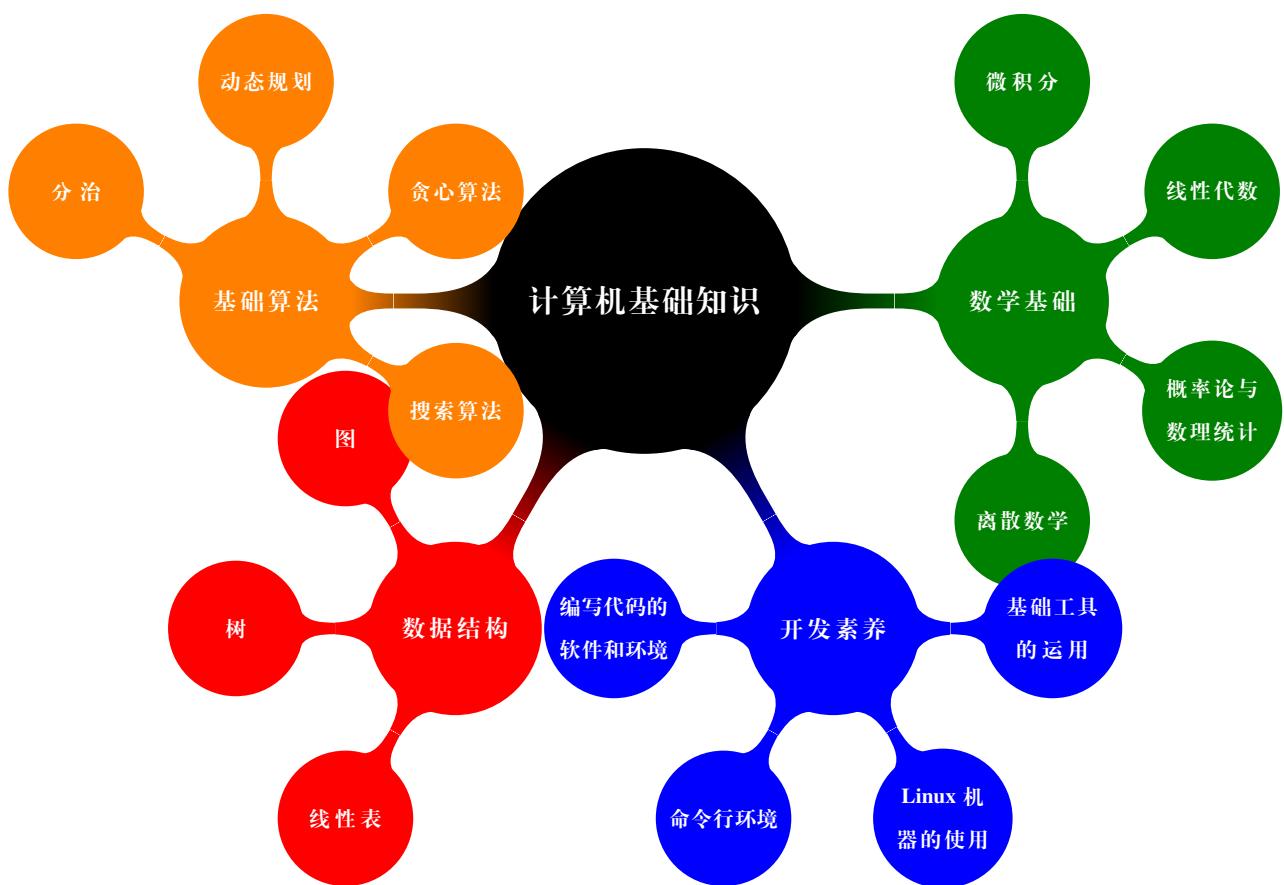


第一部分 * 基本介绍





第1章 ✕ 计算机的基础知识大纲



基础知识分成三个部分：下面配合着技能树具体了解一下：

第一部分是基础的数学基础，同学们要学习基础的分析理论、学习代数理论和统计学。对于数学基础，同学们可以阅读这一本书的第3章获取大概的了解。对于这一部分，除了跟着学校上课以外，可以自行阅读或者学习我们推荐的资料。

推荐的顺序: 必学的是微积分、线性代数和离散数学(本书组合数学和生成函数), 概率论, 这些尽量在大一结束前学完, 剩下的几部分根据个人喜好来学.

第二部分就是基础的开发素养, 这一部分的知识学校并不会开课去教, 关于这一部分的知识, 同学们可以阅读这一本书的第 4 章进行大概的了解. 对于这一部分, 同学们需要学习很多软件的使用, 比如说 git、vim、tmux 等. 了解一些命令行环境的知识, 了解一些可以用来写代码的软件. 这一部分必须学, 推荐在大一结束前完全了解.

第三部分就是数据结构和算法, 这一部分的知识可以通过学习学校的数据结构和算法课获得, 本书的第 5 章简单介绍了这一部分的知识. 同学们可以通过继续学习和大量的练习巩固这一部分的知识. 具体的资料和练习网站在本章后面. 这一部分必须学, 推荐在大一结束前完全了解.





第 2 章 ✕ 计算机学院专业必修课基本介绍

	数学课	电路相关电子学	软件相关	系统能力相关	课程设计
大一上	微积分 线性代数		C语言程序设计		
大一下	微积分 概率论		数据结构		
大二上	复变函数 离散数学	电路理论 数字逻辑	C++程序设计		程序课设
大二下		信号与系统	Java程序设计 算法设计	汇编语言 计算机组成原	
大三上	数值分析		软件工程	操作系统 计算机网络	组原课设
大三下				编译原理 数据库原理	操作系统课设
大四					系统能力培养

图 2.1: 计算机学院课程

计算机学院的课程有几条线, 我们分别对每一条线进行介绍.

数学课

微积分 微积分是大一阶段开设的数学基础课, 讲述了分析学的基础知识, 基本内容可以在第 3 章的 3.1 部分了解.

线性代数 线性代数是大一阶段开设的数学基础课, 讲述了大学代数学的基本知识, 了解较为高级的代数工具, 基本内容可以在 3.2 部分了解.

概率论与数理统计 概率论与数理统计也是大一阶段开设的数学基础课, 在高中我们讨论的概率一般是简单的离散概率, 现在我们讨论事件的函数分布, 并且对函数分布进行统计分析. 具体的部分可以在 3.3 部分了解.

复变函数与积分变换 复变函数是在大二开设的比较高阶的数学基础课, 在之前的学习我们一般会讨论实函数, 在这里我们把定义域放到复数, 并在复函数的基础上引入若干积分变换.

离散数学 离散数学本质上并不是一种数学, 而是对数学的很多种方向的一种大杂烩, 包括了基本的逻辑学, 图论, 关系理论, 数论和密码学, 涵盖了很多计算机理论所需要的数学知识. 这本书的配套网站还会提供离散数学定义和定理全集供同学们自测.

数值分析 数值分析提供了另外一种工程上处理数学问题的思路, 我们可以不获得解而去求取一个近似值. 对于积分、微分方程和方程组都可以用求近似解的方法来做, 这本书的配套网站还会提供数值分析定义和定理全集供同学们自测.

复变函数与积分变换 这一部分我们会把函数的定义域拓展到复数, 讨论复变函数的导数、积分、级数展开和线性变换, 并且引入拉普拉斯变换和傅立叶变换的概念.

推荐网课: 华中科技大学 MOOC: 复变函数与积分变换. 李红 (数学基础)

电路相关、电子学

电路理论 这个课程你需要学习最基本的电路知识, 了解多源的直流电路的分析, 交流电路的正弦稳态分析等知识.

数字逻辑 这一部分我们要用基本的逻辑门来设计数字电路, 在这里面电路量可以简化成 0、1, 在这里面你会设计简单的组合电路和时序电路, 并应用简单的集



成电路芯片来进行设计. 这本书的配套网站还会提供数字逻辑的基础知识供同学们自测.

推荐网课:UCB EE16A& B: Designing Information Devices and Systems I& II

信号与系统 这一部分你会了解与复变函数相关的内容, 对一个函数进行一定的积分变换, 通过组合积分变换来对函数进行频域上的处理.

推荐网课:MIT 6.007 Signals and Systems(奥本海默教的)

软件相关

C 语言程序设计 在这里我们学习到我们的第一个编程语言, 选择 C 语言作为第一个编程语言的原因有几点, 第一点就是它贴近于底层, 但是并不是最底层的, 第二点是命令性特征比较明显, 比较贴近人们日常生活中一件事一件事做的控制思想. 最后一点是了解一个语言的高级特性很难,C 语言的高级特性比较简易, 比较好入门. 在这门课中, 你将会学到普适很多语言的类型概念, 控制概念, 以及基本的数据结构概念.

数据结构 有人曾说过, 计算机的语言其实和人类语言一样, 都是有谓词和名词, 谓词我们用算法来形容, 名词我们就用数据结构来形容, 所以说如何用巧妙的数据结构来表示编程里面的名词(数据)很重要, 在这里面你会了解基本的数据结构, 包括数组, 栈, 队列, 树, 图等...

C++& Java 语言程序设计 在之前我们主要了解命令式的语言, 语句和数据的耦合度不高, 在 C++ 和 Java 中, 算法和数据结构将会耦合在一起, 这个叫做面向对象特性, 了解这个特性非常重要.

算法设计 我们在之前积累足够多的基础知识, 现在可以了解一下算法了(其实大一了解都可以), 在这门课中, 你将会了解 5 个基本的算法设计技巧, 至于哪 5 个需要你自行探索啦... 熟练地使用算法光靠上课可不行, 阅读第 5 章的内容, 并遵照 5.11 的提示去练习吧, 学长就是因为练晚了所以现在才那么菜的.

软件工程 在这门课你需要和你的同伴一同完成一款软件的开发.

系统能力相关

汇编语言 在计算机系统中, 上层的高级语言将会通过编译的手段编译成汇编语言, 汇编语言是贴合机器的语言, 因为执行汇编语句的时候, 汇编语句是不会被机



器打断的。了解汇编语言你可以了解机器是怎么运行你输入的代码的。或者更准确地说，代码是怎么被运行的。

计算机组成原理 在这里，你会了解计算机底层的一些构造，比如说 CPU 是如何进行计算的，计算机又是怎么存储你的数据的，你的数据是怎么被计算机提取出来的，在这门课中你都会找到答案。具体可以查看第 7 章的知识。

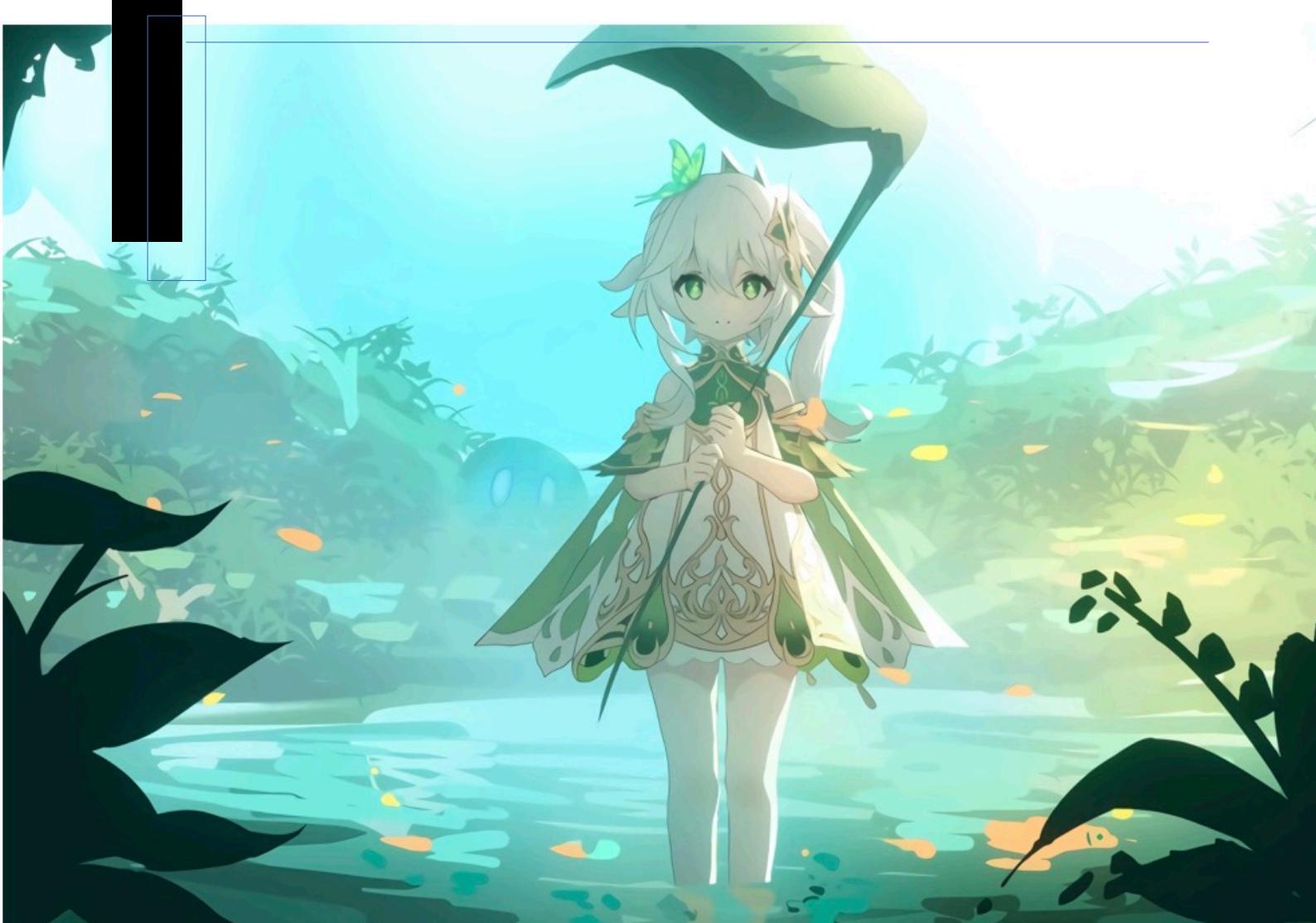
操作系统 在这里，你会了解硬件搭桥的系统软件，叫做操作系统，你会了解操作系统是怎样进行调度的，操作系统是怎么完美管理你的文件，操作系统又是怎么控制你的键盘和屏幕的。

计算机网络 在这里，你可以了解你和你的女朋友发的信息，是怎样曲曲折折地传播的。你的信息是怎样找到去往你女朋友的路的。

编译原理 在这里，你可以了解你之前编写的高级语言是怎么变成汇编语言的，你会发现计算机编程的语言也有语法，所有的语法都可以用形式语言来表达。



第二部分 * 计算机通用知识





第 3 章 ✕ 数学基础

对于计算机专业是一个传统且经典的理工科专业, 理工科专业必然是需要浓厚的数学基础, 我在这里不按课程, 按数学的方向来介绍学习计算机科学需要的数学能力. 但是我们对数学的要求肯定是不如数学专业的, 如果有更感兴趣的方向可以参考数学专业的教材.

3.1 数学分析

大学数学可以笼统地分成三个板块, 一个板块是分析学, 一个板块是代数学, 一个板块就是几何学.

分析, 本质上就是数学的一个分支学科, 以微积分方法为基本工具, 以函数(映射、关系等更丰富的内涵)为主要研究对象, 以极限为基本思想的众多数学经典分支及其现代拓展的统称, 简称分析. 具体来说, 就是微分学、积分学和级数理论.

多元函数

在微积分学的课程中, 我们把函数的定义进行一定的推广得到多元函数

定义 3.1

(多元函数的定义) 设 D 为一个非空的 n 元有序数组的集合, f 为某一确定的对应规则. 若对于每一个有序数组 $(x_1, x_2, \dots, x_n) \in D$, 通过对应规则 f , 都有

唯一确定的实数 y 与之对应, 则称对应规则 f 为定义在 D 上的 n 元函数. 记为 $y = f(x_1, x_2, \dots, x_n)$ 其中 $x_1, x_2, \dots, x_n \in D$. 变量 x_1, x_2, \dots, x_n 称为自变量, y 称为因变量.



极限

极限的本义就是数学中某一个函数中的某一个变量, 此变量在变大 (或者变小) 的永远变化的过程中, 逐渐向某一个确定的数值 A 不断地逼近而“永远不能够重合到 A ” (“永远不能够等于 A , 但是取等于 A ‘已经足够取得高精度计算结果’) 的过程中, 此变量的变化, 被人为规定为“永远靠近而不停止”、其有一个“不断地极为靠近 A 点的趋势”. 极限是一种“变化状态”的描述.

对于上面的通俗定义, 我们可以给出极限的严谨的定义:

定义 3.2

(极限的 Cauchy 定义) $\forall \epsilon > 0, \exists \delta > 0, s.t. |f(x) - A| < \epsilon$ 在 $0 < |x - a| < \delta$ 范围内成立.



按照这个极限的严谨定义可以推导出函数的左右极限以及连续性等关键性命题, 了解两个重要的极限, 了解求极限的两种方法--夹逼定则还有单调有界定则.

极限的定义为以后我们定义微分和积分提供了数学工具.

一元微分学

微分的定义就是当自变量的值从 a 比变化为 $a + dx$, 函数的值的变化值. 这个时候可以计作 $df(x)$, 具体用极限的定义可以这么写

$$\lim_{\Delta x \rightarrow 0} f(x + \Delta x) - f(x)$$

这个时候可以定义导数, 导数的定义就是



定义 3.3

$$(导数的定义) \frac{d(f(x))}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$



根据微分和导数的定义我们可以有很多工具来求导数, 包括你高中没学过的对数求导, 隐函数求导和参数方程求导, 我们可以根据导数和二次导数的值来认定函数的单调性和凹凸性.

多元微分学

多元微分学就是在一元微分学的基础上添加关于多元的讨论, 其中导数的定义会变成偏导数, 通俗地来说就是求导数的时候只把一个变量看成变量, 剩下的变量看成常量即可. 这个时候可以对导数的定义进行修整:

定义 3.4

(多元函数的导数的定义)

$$\lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x, y, \dots, z) - f(x, y, \dots, z)}{\Delta x}$$

**一元积分学**

在微积分中, 一个函数 f 的不定积分, 或原函数, 或反导数, 是一个导数等于 f 的函数 F , 即 $F' = f$.

定积分是积分的一种, 是函数 $f(x)$ 在区间 $[a, b]$ 上积分和的极限。这里应注意定积分与不定积分之间的关系: 若定积分存在, 则它是一个具体的数值, 而不定积分是一个函数表达式, 它们仅仅在数学上有一个计算关系, 这个关系叫做 Newton-Leibniz 关系. 当然一个函数, 可以存在不定积分, 而不存在定积分; 也可以存在定积分, 而不存在不定积分。一个连续函数, 一定存在定积分和不定积分; 若只有有限个间断点, 则定积分存在; 若有跳跃间断点, 则原函数一定不存在, 即不定积分一定不存在。

当然, 给出定积分的原始定义可以认为是若干个小矩形的和, 如图所示, 具体的数学定义过于复杂, 就不在这儿赘述了.



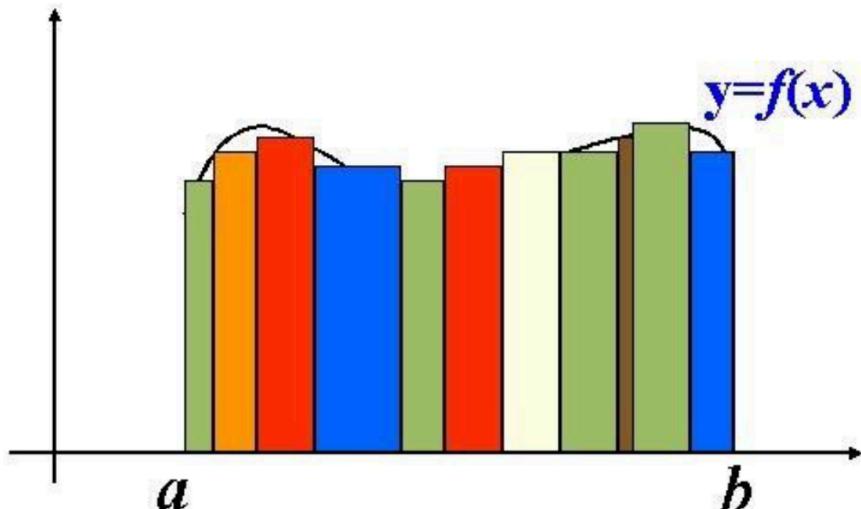


图 3.1: 定积分的定义

多元积分子

将积分的函数推广到多元函数就是多元积分子, 那么多元积分子的区间就会变成多种多样的几何图形. 比如说重积分, 第一类和第二类的线积分, 第一类和第二类的面积分. 我们知道定积分可以计算曲边梯形的面积, 重积分就可以计算体的体积, 对于曲线积分可以计算出曲线的重量, 第二类的曲线积分可以计算出做功的值, 曲面积分可以计算曲面的重量, 第二类曲面积分可以计算出函数的散度, 对于这几类积分, 做法就是降维, 转化成一元的定积分即可.

级数

级数是指将数列 a_n 的项 a_1, a_2, \dots, a_n 依次用加号连接起来的函数, 是数项级数的简称。如: $a_1 + a_2 + \dots + a_n$, 简写为 $\sum a_n$, a_n 称为级数的通项, 记 S_n 称之为级数的部分和。如果当 $n \rightarrow \infty$ 时, 数列 S_n 有极限, 极限为 S , 则说级数收敛, 否则就说级数发散。

当然, 数列的值可以是数也可以是函数, 这个都可以, 对于函数项的级数, 我们一般讨论幂级数, 对于幂级数又可以讨论收敛区域, 如何求幂级数的和等等。

我们可以把一个周期函数展开成幂级数, 也可以展开成傅里叶级数, 就是用三角函数表示一个函数。



微分中值定理

主要介绍了罗尔定理, 拉格朗日定理还有柯西中值定理. 罗尔定理主要就是, 当函数连续, 且有两点相等, 在两点之间一定有一点导数为 0. 这一部分还介绍泰勒展开, 假如说函数可以导无穷次, 那么这个函数可以表示成幂级数的形式.

3.2 线性代数

对于代数学的了解对于计算机系的学生也十分重要, 了解线性代数的知识有助于我们理解 system 的概念, 当然也有助于我们理解 AI 模型.

向量

在高中大家对向量已经有了初步的了解, 本节希望能带领大家对向量有更多直观的认识。在本节的前半部分, 向量均指二维向量。高中所学的向量通常会写成

$$\mathbf{a} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

这是一个横着排放的向量, 称为行向量, 与之对应, 列向量会被写作

$$\mathbf{b} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

大学阶段说的向量通常指的是列向量。怎么实现行列向量的转换呢, 很自然的想法是将行向量的行变成列, 就得到了列向量, 同理列向量列变行, 就得到行向量, 这种行列互换的操作称为转置。向量 \mathbf{a} 的转置记作 \mathbf{a}^\top 。

基底也是高中所学的概念, 任意两个不为 $\mathbf{0}$, 不共线的向量 \mathbf{a}, \mathbf{b} , 都可以作为一组基底, 而平面内任意向量 \mathbf{c} 均可以表示为 $\mathbf{c} = \alpha\mathbf{a} + \beta\mathbf{b} (\alpha \in \mathbb{R}, \beta \in \mathbb{R})$ 的形式, 我们定义 $\alpha\mathbf{a} + \beta\mathbf{b} (\alpha \in \mathbb{R}, \beta \in \mathbb{R})$ 为 \mathbf{a}, \mathbf{b} 的线性组合, 可以发现, 如果 \mathbf{a}, \mathbf{b} 可以作为一组基底, 那么他们的线性组合可以表示平面内的所有向量, 可以理解为这两个向量张成了二维空间。

将目光放到高维向量上, 考虑 n 个 d 维向量 $\mathbf{a}_1, \dots, \mathbf{a}_n$, 他们能张成几维空间呢?

首先考虑一个问题, 满足什么条件的 d 个向量可以作为 d 维空间的基底呢? 两个显然的条件是这 d 个向量互不共线, 且均不为 $\mathbf{0}$, 是否还需要满足其他条件呢? 一个重要的条件是任何向量不能是其他向量的线性组合, 也即该向量不能



存在于其他向量张成的空间中，否则如何指望它去开辟新的维度呢？事实上，这个条件已经包含之前两个条件了，我们称这样一组任何向量都不能表示为其他向量线性组合的向量组为线性无关向量组，这些向量线性无关，而线性无关向量组的向量个数反应了这组向量张成空间的维数。

接下来再考虑一个问题，这 n 个 d 维向量最多能张成几维空间呢？答案显然是 d ，即使 n 再大，你也不可能指望 n 个 d 维向量能张成 $d+1$ 维空间，换句话说，一组 d 维向量中的线性无关向量组大小最多为 d ，正如 14 亿生活在三维空间的人也无法亲眼窥得四维空间的真实面貌。

至此，我们可以回答最开始的问题， n 个 d 维向量 $\mathbf{a}_1, \dots, \mathbf{a}_n$ 张成的空间维数等于这组向量最大的线性无关向量组的个数。

矩阵

在具体谈及矩阵之前，先介绍线性变换，可以将变换看作一种函数，只是其输入是一个向量，输出也是一个向量，如果在变换后，原点位置没有改变，变换前的直线在变换后仍然是直线，这种变换就称为线性变换。关于线性变换，一个有趣的性质（也可能称不上一个性质）是，线性变换实质上只是变换了原空间的基底，进而，所有空间内的向量因为基底的改变也随之改变位置。

以二维空间内的线性变换为例，平面直角坐标系中的基底为 $\mathbf{x} = (1, 0), \mathbf{y} = (0, 1)$ ，假设在此空间内存在 $\mathbf{a} = (1, 2) = 1 * \mathbf{x} + 2 * \mathbf{y}$ ，通过线性变换后， $\mathbf{x}' = (1, 2), \mathbf{y}' = (3, 4)$ ，随之 $\mathbf{a}' = 1 * \mathbf{x}' + 2 * \mathbf{y}' = (7, 10)$ ，该线性变化对基底的改变为 $\mathbf{x}' = 1 * \mathbf{x} + 2 * \mathbf{y}, \mathbf{y}' = 3 * \mathbf{x} + 4 * \mathbf{y}$ 。

我们将这种线性变换记为

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

也即矩阵，并且有 $\mathbf{A} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} * x + \begin{bmatrix} 3 \\ 4 \end{bmatrix} * y = \begin{bmatrix} x + 3y \\ 2x + 4y \end{bmatrix}$ ，代表 \mathbf{a} 在经过线性变换后变成了 \mathbf{a}'

有时我们会对某一个二维空间进行连续的线性变换，若首先进行变换

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



即将两个基底位置互换，变换后 $\mathbf{x}' = (0, 1)$, $\mathbf{y}' = (1, 0)$ 之后进行变换

$$\mathbf{B} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

变换后 $\mathbf{x}'' = (1, 1)$, $\mathbf{y}'' = (0, 1)$, 也即 $\mathbf{Bx}' = \mathbf{x}''$ 由于 $\mathbf{Ax} = \mathbf{x}'$ 则 $\mathbf{Bx}' = \mathbf{BAx}$
又由于 $\mathbf{x}'' = 1 * \mathbf{x} + 1 * \mathbf{y}$, $\mathbf{y}'' = 1 * \mathbf{x} + 0 * \mathbf{y}$

$$\text{因此 } \mathbf{BA} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

显然 \mathbf{BA} 表示先进行线性变换 A 再进行线性变换 B , \mathbf{AB} 则恰好相反，最终的线性变换必然不同，因此矩阵乘法没有交换律，通过对基底两次变换的抽象分析，我们可以得到：

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a * e + b * g & a * f + b * h \\ c * e + d * g & c * f + d * h \end{bmatrix}$$

更抽象地，我们有， \mathbf{AB} 有合法结果的先决条件是 \mathbf{A} 的列数等于 \mathbf{B} 的行数，矩阵乘法的公式为：

$$\mathbf{A}_{mn} \mathbf{B}_{np} = (c_{ij})$$

其中 $c_{ij} = \sum_{r=1}^n a_{ir} b_{rj}$, 即乘积的第 i 行第 j 列元素等于 \mathbf{A} 的第 i 行与 \mathbf{B} 的第 j 列进行数乘的结果。

行列式

前面说到矩阵代表线性变换，尽管在线性变换前后直线仍然是直线，但其长度发生了变化，相对应的，两个向量在线性变换前后的面积也有变化，仍以二维矩阵为例，经过线性变换

$$\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

向量 $\mathbf{x} = (1, 0)$, $\mathbf{y} = (0, 1)$ 变为 $\mathbf{x}' = (2, 0)$, $\mathbf{y}' = (0, 3)$, 变换后面积是变换前的 6 倍，为了表示这种面积的变化，记

$$\begin{vmatrix} 2 & 0 \\ 0 & 3 \end{vmatrix} = 6$$



其中等式左侧代表行列式，右侧代表行列式的值，可以注意到，线性变换前后基底的相对位置没有发生改变，如果基底相对位置发生了改变呢？经过线性变换阵为例，经过线性变换

$$\mathbf{A} = \begin{bmatrix} 0 & 3 \\ 2 & 0 \end{bmatrix}$$

，向量 $\mathbf{x} = (1, 0), \mathbf{y} = (0, 1)$ 变为 $\mathbf{x}' = (0, 2), \mathbf{y}' = (3, 0)$ ，变换后面积是变换前的 6 倍，但此时基底相对位置已经发生了改变，记

$$\begin{vmatrix} 0 & 3 \\ 2 & 0 \end{vmatrix} = -6$$

即行列式值的正负代表线性变换有没有改变空间的方向，行列式值的绝对值代表线性变换前后面积改变值的大小。

而对于三维线性变换，方向是否变化可以用右手定则来确定，行列式值的绝对值自然代表了变换前后体积改变值的大小。

在此给出计算行列式值的一般公式，当然在此并不用熟记，通过做题大家自然不会忘记。

首先介绍代数余子式：对于行列式

$$A = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

其有 9 个代数余子式，分别为 $A_{ij} = (-1)^{i+j} * M_{ij}, i, j \in \{1, 2, 3\}$, M_{ij} 为余子式，代表行列式去掉第 i 行第 j 列后形成的行列式值。行列式值的计算方法为任取某一行/某一列，将对应元素与对应代数余子式相乘，并求和，公式为：

$$|A| = a_{i1}A_{i1} + \dots + a_{in}A_{in} = a_{1j}A_{1j} + \dots + a_{nj}A_{nj}, i, j \in \{1, 2, \dots, n\}$$

该方法称为拉普拉斯展开

逆矩阵与秩

对于线性变换 \mathbf{A} ，若向量 \mathbf{x} 在变换后变为 \mathbf{x}' ，可写作 $\mathbf{Ax} = \mathbf{x}'$ 自然地，我们会思考一个问题，若已知变换后矩阵 \mathbf{x}' 和线性变换 \mathbf{A} ，可否求出原向量 \mathbf{x} 呢？或者，在何种情况下可以求出原向量，何种情况下不可求呢？



首先思考一个问题，若线性变化的矩阵对应的行列式值为0，这个线性变换究竟做了什么呢？对于二维变换来说，行列式值为0代表变换后面积变成了0，也就意味着变换将平面压缩成了直线，显而易见，此时的变换损失了精度，对于同一个目标向量，有若干个原向量在变换前后与之对应。

与之相反，若行列式值不为0，则说明线性变换没有造成空间的压缩，也必然可以找到一种逆变换方式，在对 \mathbf{x}' 施加这种逆变换后，可以得到初始向量 \mathbf{x} ，记这种逆变换为 \mathbf{A}^{-1} ，显然，对一个空间连续施加变换 \mathbf{A} 与逆变换 \mathbf{A}^{-1} 后，最终这个空间应该毫无变化，即基底没有变化，记这种变化为单位矩阵

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

并且有 $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$

如果三维变换对应的行列式值为0呢？

该变换可能将空间压缩成了平面或直线，并且压缩方式取决于变换的维数，或者，该变换列向量张成的空间的维数，或者，该变换行向量张成的空间的维数，或者，行向量组的秩，或者，列向量组的秩。没错，他们就是秩，当你对秩感到不解之时，可以回过头来看看这些文字。

当然最后，给出逆矩阵的求解方法（如果它有逆矩阵的话）：定义矩阵的关联矩阵：

$$A^* = \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \vdots & \vdots & \vdots & \vdots \\ A_{1n} & A_{2n} & \cdots & A_{nn} \end{bmatrix}^T$$

矩阵的逆为：

$$A^{-1} = \frac{A^*}{|A|}$$

只有方阵可以求逆。

特征值与特征向量

在线性变换后，大多数向量都偏离了原来的方向，但也有少数例外，对于某一个线性变换，它对一些向量可能只起到了拉伸的作用，我们将这些向量称为



特征向量 \mathbf{x} , 对应的拉伸的倍率称为对应的特征值 λ , 记作:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

对等式移项得: $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$

则需 $(\mathbf{A} - \lambda\mathbf{I})$ 行列式值为 0.

一个有趣的事是, 对于一个除对角线外所有元素均为 0 的 $n \times n$ 矩阵, 其特征值恰好是对角线上的 n 个元素, 若将矩阵 \mathbf{B} 表示为 $\mathbf{A} = \mathbf{P}^{-1}\mathbf{AP}$ 其中 A 是对角阵的形式, 称为相似对角化。

相似对角化一大好处是计算矩阵的 n 次幂变得容易了, 即 $\mathbf{B}^n = \mathbf{P}^{-1}\mathbf{A}^n\mathbf{P}$

3.3 概率论

对于计算机系的学生, 对于概率的了解也比较重要, 在人工智能领域的基础算法很多情况下就是基本的概率算法.

条件概率与事件的分解

首先, 概率不是频率这个要搞清楚。怎么说呢, 概率是个理论上算出来的东西, 频率是靠试验和统计得出来的个实际数, 和理论有偏差这很正常。另外, 事件分必然事件可能事件和不可能事件, 事件与事件之间也是有关系的。比如说明天太阳打西边出来, 这就是个不可能事件。事件与事件之间可能并不独立, 按照马克思的说法, 他们是联系的 (哲学带师上线)。比如说我掷出一个骰子, 它是偶数和它的点数能被 3 整除就并不独立。因为如果是个 6 点这两个条件是同时满足的。事件与事件有互相独立的, 有互相排斥的, 有包含的, 有对等的, 等等等等。

对于条件概率, 就是在 B 成立的情况下 A 也成立的概率 $P(A|B)$, 有一个贝叶斯公式:

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

而对于独立事件 A , B , 它们的联合概率 $P(AB) = P(A)P(B)$, 当有更多的时候就成为了一个连乘。

这里需要提醒一下, 贝叶斯公式、联合概率公式和全概率公式是不同的, 对于不一定独立的事件而言, 有很多条件 $\{A_n\}$, 我们的全概率展开:

$$P(B) = \sum_{i=1}^n P(B|A_i)P(A_i)$$



概率分布

这下子我们的事件是与一个随机变量的取值有关系了。还是拿骰子打比方，一颗骰子的点数为 6 是一个事件，那我可以把骰子的点数作为一个变量，取值为 1, 2, 3, 4, 5, 6。对于每一个随机变量的取值都可以算一个概率出来。像骰子点数这种取值情况相当有限的变量我们称其为离散型随机变量。但是，如果我这个随机变量是类似于一根绳子剪一刀能剪成两条多长的绳子，这不好说。因为它是一个连续型随机变量，3, 3.1, 3.14, 3.1415926535……都有可能。取值的可能性有无数个，那每个值概率趋近于 0 就没办法算了。这个时候我们的概率通常采用 $P(X \leq x)$ 这种形式，把概率写成一个关于 x 的函数叫做概率函数。但很多概率写成函数很麻烦，所以我们更多的会用函数的导数叫做概率密度来描述分布情况。

下表列出了常用分布的概率表达式或概率密度表达式：

表 3.1: 常用分布的概率表达式

分布	概率或者概率密度
0-1 分布 $B(1, p)$	$P(X = 1) = p$
二项分布 $B(n, p)$	$P(X = k) = C_n^k p^k (1 - p)^{1-k}$
超几何分布 $H(M, N, a)$	$P(X = k) = \frac{C_M^k C_N^{a-k}}{C_{M+N}^a}$
几何分布	$P(X = k) = (1 - p)^k p$
泊松分布 $P(\lambda)$	$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$
连续分布 $U[a, b]$	$f(x) = \frac{1}{b-a}$
正态分布 $N(\mu, \sigma)$	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
指数分布 $E(\theta)$	$f(x) = \theta e^{-\frac{x}{\theta}}$

另外，有条件概率，就会有条件分布。类比一下从事件到分布列，想一想条件分布是个什么情况。

期望与方差、协方差

这几个是最常见的统计量了可以说。期望描述了一个概率分布的平均水平，方差表示集中程度，协方差比较特殊是多个随机变量之间的，它描述的是一种关联关系。具体的计算方法：



期望：

$$E(X) = \sum_{i=1}^n x_i P(X = x_i)$$

方差：

$$D(X) = \sum_{i=1}^n (x_i - E(X))^2 P(X = x_i)$$

其实如果稍作展开计算一下我们就可以得到：

$$D(X) = E(X^2) - E^2(X)$$

协方差与协方差矩阵也是很重要的概念。和上面不同，在概率论和统计中，协方差是对两个随机变量联合分布线性相关程度的一种度量。两个随机变量越线性相关，协方差越大，两个变量完全线性无关，协方差为零。定义如下：

$$Cov(X, Y) = E((X - E(X))(Y - E(Y)))$$

和上面黑塞矩阵的导出类似，如果我有很多个随机变量 $\{X_n\}$ ，我们对其两两比较并把协方差写到对应的位置这样我们形成了一个协方差矩阵。

需要注意的是，如果我们是对两列实际值进行协方差计算，情况是不一样的。实际上，如果是对观测值求平均数、方差，它们也会和理论上的期望方差有区别，因为概率层面上的数值是个理论值，实际观测与理论会有偏差。所以，如果是两列观测数值求解我们的公式是：

$$Cov(X, Y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

至于为啥是 $n-1$ 就不需要管了，这是样本估计总体的时候得到的。

统计分布

这不同于上面的概率分布，这里的分布都是基于试验的观测结果算出来的。但无论是 t 分布，F 分布还是卡方分布，都逃不开正态。

卡方分布：若对 n 个随机变量 (X_1, X_2, \dots, X_n) ， $X_i \sim N(0, 1)$ ，那么定义一个新的随机变量 $\chi^2 = X_1^2 + X_2^2 + \dots + X_n^2$ 叫做卡方分布，自由度为 n ，记作 $\chi^2 \sim \chi^2(n)$

t 分布：若随机变量 $X \sim N(0, 1)$, $Y \sim \chi^2(n)$ 而且相互独立，定义一个新的随机变量 $t = \frac{X}{\sqrt{Y/n}}$ ，它服从 t 分布自由度为 n ，记作 $t \sim t(n)$

F 分布：若随机变量 $X_1 \sim \chi^2(n_1)$, $X_2 \sim \chi^2(n_2)$ 而且相互独立，新的随机变



量 $F = \frac{X_1/n_1}{X_2/n_2}$, 它服从 F 分布, 自由度 (n_1, n_2) , 记作 $F \sim F(n_1, n_2)$

另外我们还定义一个分位点的概念。拿卡方分布做例子, 如果 $P(\chi^2 > chi^2_\alpha(n)) = \alpha$ 那么我们称这一点叫做上 α 分位点。这个得查表, 如果硬算这太难了。

最大似然法

最大似然方法用于参数值的估计。这个具体步骤是这样的: 首先, 在我们的概率或者概率密度当中存在未知的参数要估值。那怎么办呢, 我们把每一个观察值对应的概率(密度)乘起来然后取对数就是我们的似然函数:

$$L(\theta) = \ln \prod_{i=1}^n f(x_i; \theta)$$

当然了, 待估计的参数可能有几个, 我们干脆把它们统一看成一个向量 θ , 然后解方程 $\frac{\partial L(\theta)}{\partial \theta} = 0$ 我们就解出来了。

3.4 算法的复杂度标记

限界函数的定义

算法时间复杂度的限界函数常用的有三个对应的渐进记号. O (上界函数), Ω (下界函数), Θ (渐进紧确界函数).

定义 3.5

如果函数 $f(n)$, 存在一个 c 和 n_0 , 使得对于任意的 $n \geq n_0$, 都有 $0 \leq f(n) \leq cg(n)$, 那么我们可以说 $f(n) = O(g(n))$

含义: 当 n 足够大的时候, $f(n)$ 总小于 $|g(n)|$ 的一个常数倍.,



O 记号给出的是渐进上界, 称为上界函数 (upper bound)

上界函数代表了算法最坏情况下的时间复杂度.,

在确定上界函数时, 应试图找阶最小的 $g(n)$ 作为 $f(n)$ 的上界函数——紧确上界 (tight upper bound).

如: 若: $3n + 2 = O(n^2)$ 则是松散的界限;

而: $3n + 2 = O(n)$ 就是紧确的界限.

这样说来, 紧确上界就是阶数最小的上界函数.

同样的, 也有其他集合符号的定义.



定义 3.6

如果函数 $f(n)$, 存在一个 c 和 n_0 , 使得对于任意的 $n \geq n_0$, 都有 $f(n) \geq cg(n)$,
那么我们可以说 $f(n) = \Omega(g(n))$

含义: 当 n 足够大的时候, $f(n)$ 总不小于 $|g(n)|$ 的一个常数倍.



同样的, Ω 记号给出一个渐进下界, 称为下界函数 (lower bound). 在确定下界函数时, 应试图找出数量级最大的 $g(n)$ 作为 $f(n)$ 的下界函数——精确下界.

定义 3.7

如果函数 $f(n)$, 存在一个 c_1, c_2 和 n_0 使得对于任意的 $n \geq n_0$, 都有 $c_1g(n) \leq f(n) \leq c_2g(n)$, 那么我们可以说 $f(n) = \Theta(g(n))$

含义: 当 n 足够大的时候, $f(n)$ 总不小于 $|g(n)|$ 的一个常数倍. 总不大于 $|g(n)|$ 的一个常数倍.



但是 Θ 给出的就是一个渐进的精确界.

下面这张图片可以反映各个符号所代表的关系

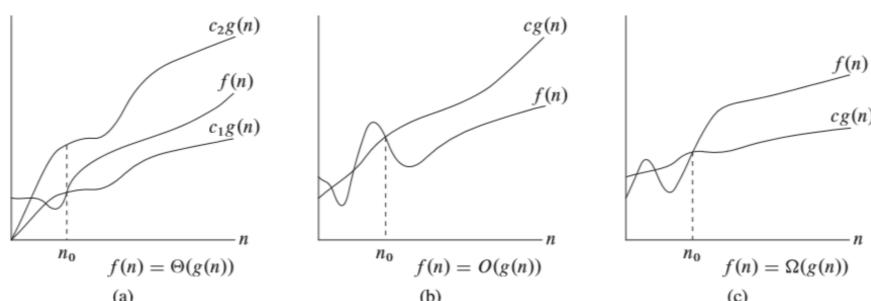


图 3.2: 渐进函数

一般来说, 可能会让你证明某个函数是怎样怎样的, 比如说

$$\frac{n^2}{2} - 3n = \Omega(n^2)$$

这个时候我们只需要求出三个数, 满足渐进符号的定义即可.

解 根据渐进符号的定义, 可以写出使得对于任意的 $n \geq n_0$, 都有 $c_1n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2n^2$

两边同除 n^2 , 有:

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$



这里只要取 $c_1 = \frac{1}{14}, c_2 = \frac{1}{2}, n_0 = 7$, 即可得证. ■

注意: $f(n)=O(g(n))$ 不能写成 $g(n)=O(f(n))$.

下面继续介绍渐进符号: 这里引入 o, ω 记号专门用来表示一种非渐进紧确的上界或下界.

定义 3.8

如果函数 $f(n)$, 对于任意正常数 c 存在常数 n_0 , 使得对于任意的 $n \geq n_0$, 都有 $f(n) \leq cg(n)$, 那么我们可以说 $f(n) = o(g(n))$



定义 3.9

如果函数 $f(n)$, 对于任意正常数 c 存在常数 n_0 , 使得对于任意的 $n \geq n_0$, 都有 $f(n) \geq cg(n)$, 那么我们可以说 $f(n) = \omega(g(n))$



限界函数的性质

所有的渐进符号都有传递性, 假设渐进符号为 X , 有

$$f(n) = X(g(n)), g(n) = X(h(n)) \rightarrow f(n) = X(h(n))$$

只有三个大写的渐进符号有自反性, 假设渐进符号为 X , 有

$$f(n) = X(f(n))$$

只有一个渐进符号有对称性.

$$f(n) = \Theta(g(n)) \rightarrow g(n) = \Theta(f(n))$$

还有两个符号有转置对称性.

$$f(n) = O(g(n)) \rightarrow g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \rightarrow g(n) = \omega(f(n))$$

最后, 如果说算法的时间复杂度为 $O(n)$, 也就是说算法的执行时间与问题的规模 n 成正比.

3.5 组合数学

由于组合数学和计算机非常贴近, 我们在这里着重介绍一下:



定义 3.10

(排列) 从 n 个不同元素中取出 $m (\leq n)$ 个元素, 按照一定的顺序排成一列, 叫做从 n 个元素中取出 m 个元素的一个排列, 排列的种数叫做排列数, 记为 $P(n, m)$. 排列数的计算公式为 $P(n, m) = \frac{n!}{(n - m)!}$.

**定义 3.11**

(组合) 从 n 个不同的元素中, 任取 $m (\leq n)$ 个元素为一组, 叫作从 n 个不同元素中取出 m 个元素的一个组合, 组合的种数叫组合数, 记为 $C(n, m)$. 组合数的计算公式 $C(n, m) = \frac{n!}{m!(n - m)!}$.



例题 3.1 设今有 4 个 e , 3 个 f , 2 个 g , 1 个 h , 这些字母能组合成几种有序字母串?

首先优先考虑重复的元素, 先把 4 个 e 放到 10 个位置中的 4 个, 有 $C(10, 4)$ 种方法; 再把 3 个 f 放在剩下 6 个位置中的 3 个, 有 $C(6, 3)$ 种方法; 接下来 2 个 g , 有 $C(3, 2)$ 种方法; 最后一个 h 只能放在最后一个剩余的位置上.

所以种类数为 $C(10, 4)C(6, 3)C(3, 2) = 12600$ 种.

另一种思考的方式是除序.

假设其他 6 个字母不变, 4 个 e 分别记为 1, 2, 3, 4 那么对于 1234 的排列或者是 4321 的排列都是一样的, 这个时候可以除以 $4!$ 来表示元素是没有区别的. 所以上述问题的答案就是 $\frac{10!}{1! \times 2! \times 3! \times 4!} = 12600$.

问题 3.1 灵梦和魔理沙打则, 灵梦有 10 张 Spell Card, 想组一套卡组: 5 个 A , 3 个 B , 2 个 C , 那么灵梦在最终打则的时候会遇到几种情况? (考虑 Spell Card 刷新情况)

接下来考虑有重复元素的组合, 一般采用隔板法. 对于 n 个元素放进 m 个盒子里面的情况, 我们可以构造 $m - 1$ 个隔板, 这样就可以用隔板间夹住的部分表示盒子了, 即 $C(n + m - 1, m - 1)$.

问题 3.2 夜雀烧烤摊一共有五种烤串, 分别是小碎骨串、八目鳗串、白泽肉串、团子串和毛玉串.

一个盲盒里面有 10 串烤串:

- (i) 可以有多少种不同的盲盒?
- (ii) 若每个味道的团子都要出现一遍, 可以有多少种不同的盲盒?



定义 3.12

(环排列) 从 n 个不同的元素中, 取出 m 个元素将其围成圆环 (经过旋转可以重合的视为同一种) 称为从 n 个不同元素中取出 m 个元素的一个环排列. 环排列的种数称为环排列数.

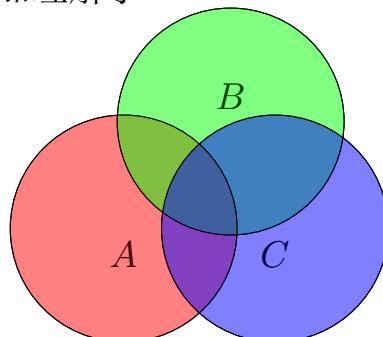


为了求环排列数, 我们不妨先考虑从 n 个元素中已经选好 m 个元素, 求这 m 个元素组成环的种数: 如果固定一个长度为 m 的圆, 在不将可旋转重合的情形视为一种时, 种数就是全排列数 $m!$, 而可视为一种的那些情况反过来都可以当成由取定环排列旋转得来的, 每个环排列经过旋转 $1 \sim m$ 个单位, 产生 m 个不同的情况, 故 m 倍的环排列数就等于排列数. 故从 n 个元素取出 m 个元素的环排列数为: $C(n, m) \frac{m!}{m} = C(n, m)(m - 1)!$ 种.

容斥定理

$$p(A + B) = p(A) + p(B) - p(AB)$$

概率论衡量的是本质是对事件的计数, 这一公式的本质就是容斥定理, 看看这个 Venn 图, 大概能够更加理解了.



推广后即

定理 3.1

(容斥定理)

$$\left| \bigcup_{i=1}^n S_i \right| = \sum_{m=1}^n (-1)^{m-1} \sum_{1 < a_1 < \dots < a_m < n} \left| \bigcap_{i=1}^m S_{a_i} \right|$$



最后一项可能看着很复杂, 其实就是可以表为 m 个集合的交的子集的元素个个数, 再让 m 从 1 跑到 n . 下面给出简要的证明:



对于元素 x , 假设它出现在 $T_1, T_2, T_3, \dots, T_m$ 的集合中, 那么它的出现次数为 1.

对于 m 个分类, 这个元素 x 都在分类里面, 我们可以写出:

$$\begin{aligned} \text{Cnt} &= |\{T_i\}| - |\{T_i \cap T_j | i < j\}| + \cdots + (-1)^{k-1} \left| \left\{ \bigcap_{i=1}^k T_{a_i} \middle| a_i < a_{i+1} \right\} \right| \\ &\quad + \cdots + (-1)^{m-1} |\{T_1 \cap \cdots \cap T_m\}| \\ &= C(m, 1) - C(m, 2) + \cdots + (-1)^{m-1} C(m, m) \\ &= C(m, 0) - \sum_{i=0}^m (-1)^i C(m, i) \\ &= 1 - (1 - 1)^m = 1 \end{aligned}$$

于是每个元素出现的次数为 1, 合并起来就是并集. \square

问题 3.3 寺子屋的运动会结束了, 琪露诺、大妖精、露米娅和三月精都参加了运动会. 慧音老师说有的人报了 100m; 有的人报了跳远; 有的人报了 1000m; 有的人报了跳高……

(i) 构建一个函数模型, 它是否是满射? 是否是单射?

(ii) 这几个人的成绩一共有多少种可能的结果?

现在我们来引出一个叫做错位排序的概念.

定义 3.13

(错位排列) 对于 $1 \sim n$ 的排列 P 如果满足对任意 $i = 1, 2, \dots, n$, 都有

$P_i \neq i$, 则称 P 是 n 的错位排列.



求错位排序的个数需要转换一下思路: 我们可以把全排序 $|U| = n!$ 求出来, 减去不满足错位排序要求的, 错位排序就是没有一个元素是匹配的, 那么我们设 S_i 代表有 $P_i = i$ 满足

这个 $\left| \bigcup_{i=1}^n S_i \right|$ 就可以用容斥定理求出来这个数就是满足一个的部分, 减去满足两个的部分, 再加上满足三个的部分, 再减去满足四个的部分……依此类推,

有

$$\left| \bigcup_{i=1}^n S_i \right| = \sum_{k=1}^n (-1)^{k-1} C(n, k)(n - k)!$$



上面的式子的含义是：对于有 k 个 i , 满足 $P_i = i$ 的情况的计数结果，就是先从 n 个数里面选 k 个，然后剩下的 $n - k$ 个元素进行全排序.

于是错位排序的计算式为

$$D_n = n! \left(1 - \sum_{k=1}^n \frac{(-1)^{k-1}}{k!} \right) = n! \sum_{k=0}^n \frac{(-1)^k}{k!}$$

我们注意到这个结果后面的部分很像 Taylor 展开式，其实它正是 e^{-x} 在 $x = 0$ 处展开，保留至 n 次项，令 $x = -1$ ，其与 e^{-1} 相差 Lagrange 余项为 $\frac{(-1)^{n+1}}{(n+1)!e^\theta}$, $\theta \in (0, 1)$. 于是

$$D_n = \frac{n!}{e} - \frac{(-1)^{n+1}}{(n+1)e^\theta}$$

由于余项的绝对值很小，不影响整数部分，而结果必然是与前一项接近的整数，只是由于符号问题，奇数项向下取整，偶数项向上取整. 我们可通过一些手段使其形式更简单，比如加一个适当小的数，使得无论奇偶项的波动都比正确结果大但都不超过 1. 注意到 $\left| \frac{(-1)^{n+1}}{(n+1)} \right| \in \left(0, \frac{1}{3} \right]$, $\frac{1}{e^\theta} \in \left(\frac{1}{e}, 1 \right)$ ，其乘积小于 $\frac{1}{3}$. 我们

可以选择这个适当大小的数大于 $\frac{1}{3}$ 以致偶数项时在加上此数后进 1，同时小

于 $\frac{2}{3}$ 以致加上此数时奇数项不会进 1，我们选为 $\frac{1}{e}$.

最后结果可以漂亮地简记为 $D_n = \left\lfloor \frac{n! + 1}{e} \right\rfloor$

问题 3.4 恋、魔理沙、紫、永琳在剧院看戏，然后妹红和辉夜打架不小心把剧院给烧了. 慌乱之中，她们四个随便戴上了一个帽子夺路而逃，却没有注意她们各自戴的帽子究竟是不是自己的. 问她们四个戴上的帽子不是自己的的概率是多少？

问题 3.5 设 $1 \leq x, y \leq N$, $f(k)$ 表示最大公约数为 k 的有序数对 (x, y) 的个数，求 $f(1)$ 到 $f(N)$ 的值. (提示：由容斥原理可以得知，先找到所有以 k 为公约数的数对，再从中剔除所有以的倍数为 k 公约数的数对，余下的数对就是以 k 为最大公约数的数对. 即 $f(k) = \text{以 } k \text{ 为公约数的数对个数} - \text{以 } k \text{ 的倍数为公约数的数对个数.}$)

本节的最后，给出与容斥定理相关的二项式反演公式，它也被称为广义反演公式.



定理 3.2

(二项式反演公式)

$$F_n = \sum_{i=0}^n C(n, i) G_i$$

$$G_n = \sum_{i=0}^n (-1)^{n-i} C(n, i) F_i$$



下面给出证明：

将二项式反演公式中的第一式代入第二式，有

$$\begin{aligned} G_n &= \sum_{i=0}^n (-1)^{n-i} C(n, i) F_i \\ &= \sum_{i=0}^n (-1)^{n-i} C(n, i) \sum_{k=0}^i C(i, k) G_k \\ &= \sum_{i=0}^n \left(\sum_{k=0}^i (-1)^{n-i} C(n, i) C(i, k) \right) G_i \end{aligned}$$

根据二项式定理，可以得到最后一行就是 G_n ，于是二项式反演公式成立。□

也可以用生成函数来进行证明。先给出生成函数相乘的数学依据：

$$\begin{aligned} \hat{F}(x)\hat{G}(x) &= \sum_{i \geq 0} a_i \frac{x^i}{i!} \sum_{j \geq 0} b_j \frac{x^j}{j!} \\ &= \sum_{n \geq 0} x^n \sum_{i=0}^n a_i b_{n-i} \frac{1}{i!(n-i)!} \\ &= \sum_{n \geq 0} \frac{x^n}{n!} \sum_{i=0}^n C(n, i) a_i b_{n-i} \end{aligned}$$

由于

$$F(x) = \sum_{i=0}^n F_i \frac{x^i}{i!}$$

并且数列 $a_n = 1$ 对应的生成函数就是 e^x ，所以有

$$F(x) = \sum_{i=0}^n \left(\sum_{k=0}^i C(i, k) G_k \right) \frac{x^i}{i!} = G(x)e^x$$

可得 $G(x) = F(x)e^{-x}$ 。

Stirling 数

定义 3.14

(第一类 Stirling 数) 第一类 Stirling 数 (或称 Stirling 轮换数) 的符号为 $s(n, k)$, 表示将 n 个不同的元素, 划分为 k 个环排列的方案数.



根据定义, 我们有下面这三条推论:

- (i) $s(n, 1) = (n - 1)!$
- (ii) $s(n, n) = 1$
- (iii) $s(n, n - 1) = C(n, 2)$

由定义我们还可以得到递推式

$$s(n, m) = s(n - 1, m - 1) + (n - 1)s(n - 1, m)$$

其组合意义是: 对 $n - 1$ 个元素的情况分类, 前一项表示在已经做好 $n - 1$ 个元素的安排基础上令第 n 个元素自成一个单元素的环排列, 后一项表示已将 $n - 1$ 个元素组成 m 个环排列, 再将第 n 个元素依次插入环排列中 $1 \sim n - 1$ 这些元素的前面形成新的 m 个环排列的情况.

定义 3.15

(第二类 Stirling 数) 第二类 Stirling 数的符号为 $S(n, m)$, 表示将不同的 n 个元素划分为 m 个集合的方法数.



类似地, 有第二类 Stirling 数的递推式: $S(n, m) = S(n - 1, m - 1) + mS(n - 1, m)$.

前一项表示将一个新元素单独放在一个子集之中, 有 $S(n - 1, k - 1)$ 个方案, 后一项表示将新元素放入一个现有的非空子集, 有 $kS(n - 1, k)$ 个方案.

下面我们来求第二类 Stirling 数的通项公式:

假设有 n 个两两不同的元素, 划分到 i 个两两不同的集合的方案数是数列 G_i , 如果还要求这个集合非空, 那就是数列 F_i , 这个时候我们可以写下 $G_i = i^n$.

我们现在知道, 可以假设有 $0 \sim i$ 个非空集合, 这个时候可以写出 F_i 和 G_i 的联系.

我们知道对于一个 i , G_i 里面包含 i 种情况, 就是有 $1, 2, 3, 4, \dots, i$ 个集合里面有元素, 这个时候可以根据加法原则得到 G_i 就是等于这几种情况相加得来的结果, 对于 j 个集合里面有元素的情况, 就是 $F_i C(i, j)^{[1]}$

^[1] i 个集合里面选 j 个集合有元素.



$$G_i = \sum_{j=0}^i C(i, j) F_j$$

根据二项式反演公式, $F_i = \sum_{j=0}^i (-1)^{i-j} C(i, j) G_j$, 代入 $G_i = i^n$. 第二类 Stirling 数要求集合之间互不区分, 因此 F_i 正好就是 $S(n, i)$ 的 $i!$ 倍.

Catalan 数

定义 3.16

(Catalan 数) Catalan 数可以表示成从 $(0, 0)$ 到 (n, n) 的除端点外不穿越直

线 $y = x$ 的非降路径数^[2], 记为 H_n .

^b非降路径是指只能向上或向右走的路径.



对于第一步和最后一步, 我们知道这两步一定是从 $(0, 0)$ 到 $(1, 0)$ 和从 $(n, n-1)$ 到 (n, n) 来的. 这时所有的非降路径就就是 $C(2n-2, n-1)$ 条, 随遇任意一条穿越了 $y = x$ 的路径, 我们可以把最后离开这条线的点到 $(1, 0)$ 的部分关于 $y = x$ 进行对称变换, 就可以得到一条从 $(0, 1)$ 到 $(n, n-1)$ 的非降路径, 同理反之亦成立.

上面那一部分就是说明: 总的有 $C(2n-2, n-1)$ 条路径, 其中不符合要求的就是从 $(1, 0)$ 到 $(n-1, n)$ 的路径, 就是 $C(2n-2, n)$.

根据对称性有 $2(C(2n-2, n-1) - C(2n-2, n))$, 这个时候我们可以求出

$$H_n = C(2n, n) - C(2n, n-1) = \frac{C(n, 2n)}{n+1}.$$

我们借助龙珠的情形来说, 第一次进行的操作必然是放龙珠, 我们假设这颗龙珠在第 i 次取出时被取走, 那么在其被取走之前就是一次 $i-1$ 颗龙珠小型但完整的进出龙珠的过程, 而且之后的 $n-i$ 次取出也构成了 $n-i$ 颗龙珠的进出过程. 这里的 i 会取遍 1 到 n , 把他们加起来就是所有的情况了. 写作:

$$H_n = \sum_{i=0}^{n-1} H_i H_{n-1-i}$$

¹当然只能在筒内有龙珠的情况下取最上面的一个.



设 $C(x) = \sum_{n=0}^{\infty} H_n x^n$, 计算

$$(C(x))^2 = \sum_{n=0}^{\infty} \left(\sum_{i=0}^n H_i H_{n-i} x^n \right) = \sum_{n=0}^{\infty} H_{n+1} x^n$$

这样我们就可以注意到 $(C(x))^2$ 的每项几乎是 $C(x)$ 的对应系数向后移位得到的, 那么我们凑角标作差即可发现 $x(C(x))^2 - C(x) = -H_0 = -1$.

对此二次方程解得

$$C(x) = \frac{1 - \sqrt{1 - 4x}}{2x}$$

这里显然要舍去负根, 我们就得到了 Catalan 数的生成函数.

问题 3.6 夜雀食堂, 小碎骨 5 元一串. 现在有 $2n$ 位幻想乡居民, 其中 n 个有 10 元钞票, 剩下 n 个有 5 元钞票, 请问有多少种情况可以使得给十块钱的居民有钱找? (提示: 可以转化成 $2n$ 个元素入栈顺序问题)

3.6 图论

图论是应用数学的一个分支, 它将实际问题抽象成“图”这一模型加以研究.

图论在很多领域中都有应用, 在外界的式神——计算机涉及到的如数据库、超文本、操作系统、计算机网络等关系型中都有很强的应用.

在学习图论的相关知识之前, 先要了解“图”是什么.

定义 3.17

(图) 设 G 是一个有序二元组 (V, E) . 若 E 中元素按照某种规则映射到 $V \times V^{[2]}$ 上, 则 G 就称为图, 并称 V 为顶点集, E 为边集. 其中, V 中的元素被称为顶点或节点, E 中的元素被称为边.

^b 此处 $V \times V$ 的含义是: $V \times V = \{(v, v) | v \in V\}$, 称这种集合运算为直积.



直观用琪露诺也能听懂的话来说“图就是点和线的集合, 表示一堆点和一堆连接点的线”.

图的种类有很多, 一个常用的分类方法是分为有向图和无向图.

定义 3.18

(有向图·无向图) 对于图 $G = (V, E)$, 如果任意连接两个顶点 D_1, D_2 的边 E 是没有方向的, 那么这个图就是无向图. 如果任意连接两个顶点 D_1, D_2



的边 E 是有方向的，那么这个图就是有向图.



图模型的建模需要说明点集和边集的意义以及图的类型.

例题 3.2 如果将幻想乡内朋友关系表示成一个图模型 G ，顶点集 V 就代表幻想乡内的每一个人，边集 E 代表的含义如下：

若 $(D_1, D_2) \in E$ ，代表 D_1 和 D_2 是朋友关系.

成为朋友肯定是双向的，那么这个图模型就是无向的，所以若 $(D_1, D_2) \in E$ ，则 $(D_2, D_1) \in E$.

例如：(灵梦, 魔理沙) $\in E$ ，则有 (魔理沙, 灵梦) $\in E$. [1]

在计算机科学领域，我们研究得比较多的是最简单的简单无向图.

定义 3.19

(简单无向图) 称连接顶点自身的边为自环，称连接同一对顶点的多条边为平行边.

没有自环和平行边的无向图称为简单无向图.



例 6.1 中的描述朋友关系的图就是一个简单无向图的实例，因为两个人之间不存在 2 个及以上的朋友关系.

问题 3.7 计算机程序中，语句对先前语句的依赖性可以用有向图表示. 每个语句用一个顶点表示，如果在第一个顶点表示的语句执行完毕之前，第二个顶点表示的语句不能执行，则从一个顶点到第二个顶点存在一条边.

根据上述规则，用图表示算式 $(a + 3) \times 4 + 5$.

度是图中很重要的一个概念，讨论度可以解决最短通路问题和握手问题.

定义 3.20

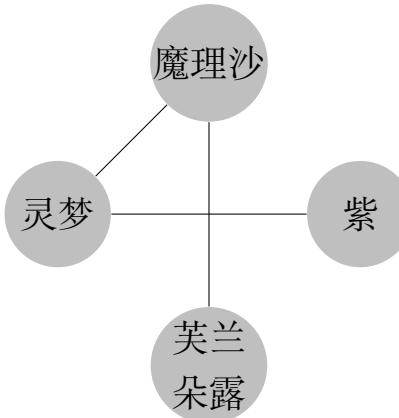
(度) 图中与某顶点 D 相连的边数称为该点的度，记为 $\deg(D)$.



例题 3.3 考虑如下的图：

¹作者是杂食党，cp 关系仅供参考.





这个时候 $\deg(\text{灵梦}) = 2$.

定理 3.3

(握手定理) 对于无向图 $G = (V, E)$, 有:

$$\sum_{D_i \in V} \deg(D_i) = 2|E|$$

其中 $|E|$ 表示边数.



问题 3.8 琪露诺、三妖精和大妖精她们在一块聊天，琪露诺说：“我能不能只和你们 4 个人的其中 3 个人做好朋友呢?”

琪露诺的假设能成立吗？

下面介绍一些特殊的图

二部图（也称二分图或偶图）是一种特殊的图，在之后的二部图博弈论中还会用到它。因此这里简要介绍一下二部图和匹配的概念。

定义 3.21

(二部图) 若一个简单图 $G = (V, E)$ 的节点集 V 可以分成两个子集 V_1, V_2 ，它们满足 $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, 且边集 E 中的任意一条边连接的两个节点都来自于不同的两节点子集 V_1, V_2 ，则称图 G 为二部图。



直观来说，二部图里面的每一个顶点都可以用给定两个颜色中的一个进行上色，并且每条边的连接的顶点颜色不同。

定义 3.22

(完全二部图) 设二部图 $G = (V, E)$ 的节点集 V 可分为两个子集 V_1, V_2 ，且对任意 $D_1 \in V_1, D_2 \in V_2$, 都有 $(D_1, D_2) \in E$ ，则称 G 为完全二部图，记为



$K_{m,n}$.**定义 3.23**

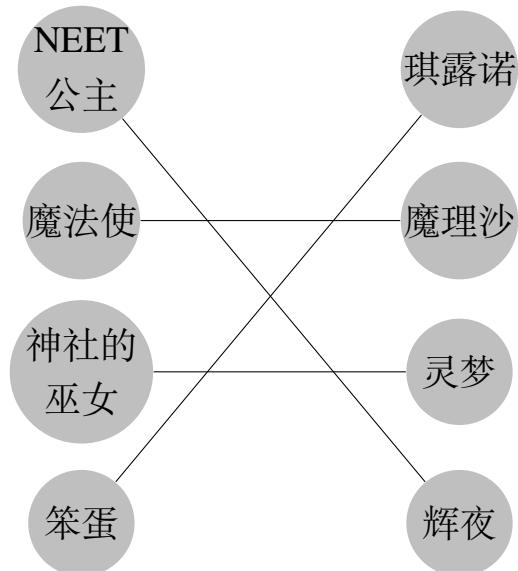
(匹配) 设 M 是图 $G = (V, E)$ 中边集 E 的子集, 且满足任取 M 中两边都没有公共顶点, 则称 M 是一个匹配.

换句话说, 若 (D_s, D_t) 和 (D_u, D_v) 是匹配 M 中的两条边, 则 s, t, u, v 是不同的顶点.

称边数最多的匹配为最大匹配, 记为 M_{\max} .

**例题 3.4**

在下面的二部图 $G = (V, E)$ 中, 最大匹配 $M_{\max} = E$.

**定义 3.24**

(完全匹配) 设二部图 $G = (V, E)$, $V = V_1 \cup V_2$. 如果一个匹配 M 由 $|V_1|^{[1]}$ 条从 V_1 引向 V_2 的边组成, 则称 M 是 V_1 对 V_2 的完全匹配.

^a $|V_1|$ 指顶点集 V_1 中元素的个数. 后边出现的类似的符号都有相似的含义, 即表示符号 || 内事物的数量.



在限制了 V_1 和 V_2 的内容的情况下, 一个完全匹配是边的个数最大的匹配.



定义 3.25

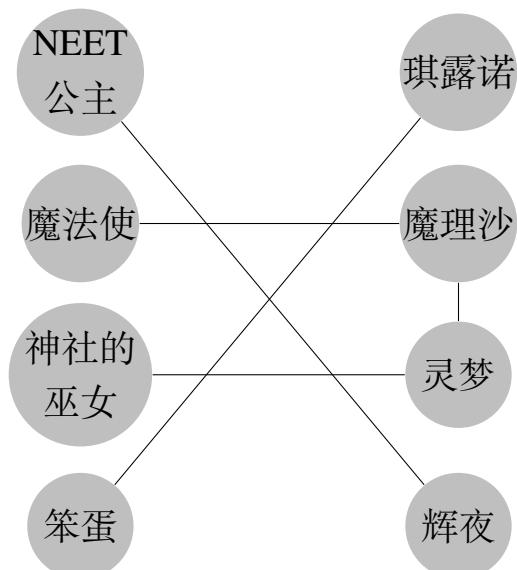
(点独立集) 对于一个图 $G = (V, E)$, 一个点集 $S \subseteq V$ 被称为点独立集当且仅当 S 中没有任意两个点相邻, 即 $\forall x, y \in S, (x, y) \notin E$, 称点集 S 为点独立集.

**定义 3.26**

(点覆盖集) 对于一个图 $G = (V, E)$, 一个点集 $S \subseteq V$ 被称为点覆盖集当且仅当对于图中的任意一条边, 这条边所连接的两个点至少有一个属于 S , 即 $\forall (x, y) \in E, x \in S$ 或 $y \in S$, 称点集 S 为点覆盖集.



例题 3.5 对于下面的匹配图, 有:



这个时候有一种点独立集: {蓬莱山辉夜, 笨蛋}.

点覆盖和点独立集里面点的个数最大或者最小有特殊的含义.

定义 3.27

(极小点覆盖 · 最小点覆盖) 若点覆盖 S 的任何真子集都不是点覆盖, 则称 S 是极小点覆盖. 顶点个数最少的点覆盖称为最小点覆盖.

**定义 3.28**

(极大点独立集 · 最大点独立集) 若在点独立集 S 中加入任何顶点都不再是独立集, 则称 S 为极大点独立集. 顶点数最多的点独立集称为最大点独立集.



这里的极大与最大的差异同上.

定理 3.4

设无向图 $G(V, E)$ 中无孤立顶点, 顶点集合 $S \subseteq V$, 则 S 是 G 的点覆盖, 当且仅当 $V - S$ 是 G 的点独立集.

设 G 是 n 阶 (n 个顶点) 无孤立点的图, 则 S 是 G 的极小 (最小) 点覆盖集, 当且仅当 $V - S$ 是 G 的极大 (最大) 点独立集, 从而有: $\alpha_0 + \beta_0 = n$ (n 为顶点个数).

其中 α_0 是极小点覆盖集的顶点数, β_0 是极大点独立集的顶点数.

设最小点覆盖集为 V , 假如有两个没在 V 中的点之间有一条边, 那么这条边就不会被 V 中的点所覆盖, 那么 V 就不是最小点覆盖集, 又因为 V 是最小点覆盖集, 所以刚才假设的两个点时不存在的, 除过 V 之外的点都是两两相互独立的.



问题 3.9 请证明对于任意一个无向图, 不同的点独立集的数量等于不同的点覆盖集的数量.

问题 3.10 永琳想设立若干个仓库, 她想节约成本, 有几种药不能放在一起, 其他的可以放在一起, 试建立图模型, 找出仓库最少要建几个?

定义 3.29

(边独立集) 一个无向图 $G(V, E)$, 一个边集 $S \subseteq E$ 被称为边独立集当且仅当 S 中任意两个边不可能与同一节点相邻, 即 $\forall x, y, z \in V, (x, y)(y, z) \notin S$, 称边集 S 为边独立集 (匹配).



定义 3.30

(边覆盖集) 一个无向图 $G(V, E)$, 一个边集 $S \subseteq E$ 被称为边覆盖集当且仅当任意取 V 中任何一个节点, 都能找到 S 中的一条边, 这个节点通过这条边与其他节点连接. 即 $\forall x \in V, \exists y \in V, (x, y) \in S$, 称边集 S 为边覆盖集.



类似的, 我们可以定义极小边覆盖、最小边覆盖、极大匹配、极小匹配:

- 极小边覆盖: 若边覆盖 S 的任何真子集都不是边覆盖, 则称 S 是极小边覆盖.



- 最小边覆盖：边数最少的边覆盖集称为最小边覆盖.
- 极大匹配：若在 S 中加入任意一条边所得到的集合都不是匹配，则称 S 为极大匹配.
- 最大匹配：边数最多的匹配称为最大匹配.

问题 3.11 (i) 模仿之前的定义，给出边覆盖数和边独立数的定义.

(ii) 证明：边独立数 + 边匹配数 = 顶点的个数.

最后介绍几个有趣的定理，证明从略.

定理 3.5

最大点独立集与最小顶点覆盖的和等于图的点数.



定理 3.6

在二分图中，最大匹配等于最小顶点覆盖.



补充信息

比如最大匹配是 M . 为了求最少的点让每条边都至少和图中一个点关联.

(1) M 个点是必需的. 匹配的 M 条边，由于他们两两无公共点，就是说至少有 M 个点才能把他们覆盖.

最小点覆盖一定 \geqslant 最大匹配，因为假设最大匹配为 n ，那么我们就得到了 n 条互不相邻的边，光覆盖这些边就要用到 n 个点.

(2) M 个点是足够的. 就是说他们覆盖最大匹配的那 M 条边后，假设有某边 e 没被覆盖，那么把 e 加入后会得到一个更大的匹配，出现矛盾.

任何一种 n 个点的最小点集覆盖，一定可以转化成一个 n 的最大匹配. 因为最小点集覆盖中的每点都能找到一条边，这条边只有这个点在点集中. (在点集中就代表被覆盖)

如果找不到则说明该点链接所有的边的另外一个端点都被覆盖，所以该点则没必要被覆盖，和它在最小点集覆盖中相矛盾.

只要每个端点都选择一个这样的边，就必然能转化为一个匹配数与点集覆盖的点数相等的匹配方案. 所以最大匹配至少为最小点集覆盖数，即



最小点集覆盖一定 \leq 最大匹配.

定理 3.7

Hungary 算法可以在 $O(|V||E|)$ 求出一组二分图的最大匹配. [1]



匈牙利算法的本质就是找到增广路^[1]，有增广路径的存在就可以扩充匹配的个数，找到最大匹配.

证明

我们从左侧一个点 A 出发，枚举所有和他有边连接的点 B 并试图与之进行匹配，即如果这个点 B 失败便再对下一个 C 尝试。如果 B 没有和任何左侧点匹配自然最好，如果没有 B 和左侧某个点 C 已经匹配了，那么我们连接线 AB ，拆掉 BC 匹配，重新试图给 C 找一个点和它匹配。

这样的处理方式是递归式的，为了让这个过程能够停止，我们不允许处理过程重复寻找点的匹配，比如找 A ，为此去找 C ，又为此去找 A 的匹配。这样一来，我们最多递归 n 次这样的寻找，有了停止步数的上界。

这样一条 $A \cdots B - C \cdots D - E$ 的路径，我们也称之为增广路。

这样的方法虽然思想简单粗暴，但却行之有效。（补充一个证明这样能得到最大匹配的证明）

我们再计算这个算法的速度。之前已经提到每次寻找增广路，我们会进行 n 层递归，而每一层递归当中我们最多会对另一侧所有的点尝试一遍，总共 n 次。而我们又只会从每一个点开始一次，寻找增广路。所以最后我们最多运行 n 三次方次的搜索语句，这个算法是 $O(N^3)$ 的。

下面介绍一个更高效的算法：

¹ $O(\cdot)$ 的含义可见 §0.1 的注脚。

¹若 P 是图 G 中一条连通两个未匹配顶点的路径，并且属于即已匹配和待匹配的边在 P 上交替出现，则称 P 为相对于 M 的一条增广路径。



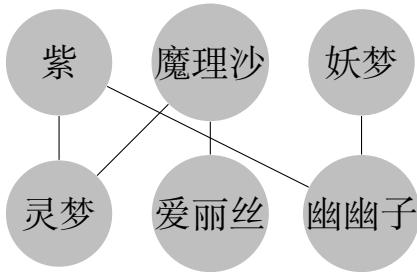
定理 3.8

Dinic 算法^[2]可以在 $O(\sqrt{|V||E|})$ 求出一组二分图的最大匹配.

^bDinic 算法 (又称 Dinitz 算法) 是一个在网络流中计算最大流的强多项式复杂度的算法, 设想由以色列 (苏联) 的计算机科学家 Yefim (Chaim) A. Dinitz 在 1970 年提出.



问题 3.12 用 Hungary 算法法求出下面 CP 组合的最大匹配:



问题 3.13 给定一个无向图, 请基于 Dinic 算法设计一个算法求出至少需要选出多少条互不相交也不自交的图上路径, 使得每个点都被覆盖. (提示: 这里可以假设给你一个 Dinic 算法黑盒, 扔进去一个二分图它就吐出来一个最大匹配. 根据 G 构造一个新的图, 使得新图的最大匹配等于原图的最小不相交路径覆盖, 然后在新图上使用 Dinic 算法)

定义 3.31

(完美匹配) 一个无向图 $G(V, E)$, 一个边独立集 S 是完美匹配当且仅当所有节点都与中的边相邻, 即 $2|S| = |V|$.



问题 3.14 一个有向图 G 中, 一个边集被称为洋流, 当且仅当这个边集由若干个 (可能为 0 个) 不相交的简单环组成, 构造一个无向图 H , 使得中完美匹配的数量和 G 中“洋流”的数量相等. (提示: 数量相等一般可以考虑构造双射; 要注意完美匹配选出来的边数量是一定的, 但一个洋流可能一条边都没有, 这时如何将空集双射到完美匹配; 太一般的情况难以求解, 可以以上一题为启发考虑 H 为二分图的情况)

Last Spell Card: 上面的中, 如果 G 是无向图如何构造?



3.7 生成函数

定义 3.32

(生成函数) 对于一个数列 a_n , 定义它的(普通)生成函数为 $F(x) = \sum_{i=1}^{\infty} a_i x^i$, 指数型生成函数为 $G(x) = \sum_{i=1}^{\infty} a_i \frac{x^i}{i!}$.



定义 3.33

(生成函数的封闭形式) 将函数项级数化简表示为一些函数的有限次四则运算与开方等简单运算, 称为其封闭形式. 封闭形式进行 Taylor 级数展开就是展开形式.



接下来补充一个关于计算生成函数的手段.

定义 3.34

(离散卷积) 给定数列 a_n 和 b_n , 定义 $\sum_{i=0}^n a_i b_{n-i}$ 为 a_n 和 b_n 的离散卷积, 记为 $a_n * b_n$.



根据离散卷积的定义我们可以介绍生成函数的基本运算.

定理 3.9

定义函数 $f(x) = \sum_{i=0}^n a_i x^i$, $g(x) = \sum_{i=0}^n b_i x^i$, 则二者之和为 $f(x) + g(x) = \sum_{i=0}^n (a_i + b_i)x^i$, 二者之积为 $f(x)g(x) = \sum_{i=0}^{2n} (a_i * b_i)x^i$.



下面来简单介绍生成函数在组合数学中的应用.

生成函数可以用乘幂来表示计数的相加, 这样可以便于计算出计数的结果. 考虑如下问题:

例题 3.6 我们要从小玉、大玉、札弹和光玉中拿一些弹幕出来, 要求小玉组成偶数弹, 大玉的个数要是 5 的倍数, 札弹最多拿 4 个, 光玉要么不拿, 要么只能拿一个. 问按这样的要求拿 n 个弹幕的方案数.

¹这个展开一般是 Maclaurin 展开, 也就是函数在 $x = 0$ 处的 Taylor 展开.



解：

$$\begin{aligned}
 g(x) &= (1 + x^2 + x^4 + \dots)(1 + x^5 + x^{10} + \dots)(1 + x + x^2 + x^3 + x^4)(1 + x) \\
 &= \frac{1}{1 - x^2} * \frac{1}{1 - x^5} * \frac{1 - x^5}{1 - x} * (1 + x) \\
 &= \frac{1}{(1 - x)^2} \\
 &= C(1, 0) + C(2, 1)x + C(3, 2)x^2 + C(4, 3)x^3 \dots \\
 &= 1 + 2x + 3x^2 + 4x^3 + 5x^4 + \dots
 \end{aligned}$$

解答的每一项 $a_n x^n$ 中， a_n 表示组弹幕的方法， x 的次数表示这一项所拿弹幕的数量。

3.8 差分方程的处理

3.8.1 差分

我们首先来回忆一下微分的定义：是当自变量 x 变化了一点点 (dx) 而导致了函数 ($f(x)$) 变化了多少。在微分里面，自变量变化的量 $dx \rightarrow 0$

差分的定义可以和微分紧密相连：可以定义成 $f(x_{k+1}) - f(x_k)$ 。当 $x_{k+1} - x_k \rightarrow 0$ 的时候，这个式子就变成了微分，当然可以定义向前差分和向后差分，这个没有太大的区别。

3.8.2 解法一：通解 + 特解

回忆 n 阶常微分方程的解法：对于 n 阶常微分方程：写出 $\sum_{i=0}^n a_i f^{(i)}(x) = g(x)$ ，我们可以得到解的形式为通解 + 特解。对于通解我们令 $f^{(n)}(x) = \lambda^n$ ，然后组合成一个 n 次方程，求出 n 个根，然后根据 $g(x)$ 的形式使用待定系数法求特解。这些读者都已经十分熟悉，我们就不在这里继续赘述。

对于离散形式的微分方程——差分方程 $\sum_{i=0}^n a_i y(t+i) = b(t)$ ，也是一样的解法：首先是通解，我们找到差分方程里面最小的标号，假设为 k_1 ，对于 $f(k_1 + k)$ 我们记为 p^k 。那么求通解的式子可以由 $\sum_{i=0}^n a_i y(t+i) = 0$ 改为 $\sum_{i=0}^n a_i p^i = 0$ ，可以转



化成一个 n 次的方程, 解得其解为 $p_1, p_2, p_3 \dots p_n$, 那么通解的形式就是:

$$\sum_{i=1}^n C_i p_i^t$$

如果上述方程的解出现了重根, 假设 p_i 出现了 k_i 重重根, 那我们就需要把解改成:(其中 p^i 含有 k_i 重实根, 满足 $\sum_{i=0}^{i_{\max}} k_i = n, i_{\max}$ 是解的种类数)

$$\sum_{i=1}^{i_{\max}} \left(\sum_{j=0}^{k_i} C_{ij} t^j \right) p_i^t$$

如果 p_i 不是实数, 我们假设 $p_1 = \alpha + \beta i = \rho e^{j\Omega_0}, p_2 = \alpha - \beta i = \rho e^{-j\Omega_0}$, 由 Euler 公式, 我们可以写出通解形式 $y(t) = C_1 \rho^t \cos(\Omega_0 t) + C_2 \rho^t \sin(\Omega_0 t)$

接下来就是特解的求法, 特解的求法根据 $b(t)$ 来定, 对于不同种类的 $b(t)$ 我们可以有不同的特解形式, 然后可以利用待定系数法就出假设的特解形式中的系数. 下表列出了特解形式于 $b(t)$ 的关系

$b(t)$	特解
t^k	$\sum_{i=0}^k D_i t^i$
a^t (n 重根)	$\left(\sum_{i=0}^n D_i t^i \right) a^t$
a^t (不是根)	$D a^t$

对于 $b(t)$ 有多项的情况, 我们可以用下面这个定理来进行处理

定理 3.10

对于差分方程 $\sum_{i=0}^n a_i f^{(i)}(x) = g_1(x) + g_2(x)$, 其特解形式为 $f, \sum_{i=0}^n a_i f^{(i)}(x) = g_1(x)$ 特解形式为 $f_1, \sum_{i=0}^n a_i f^{(i)}(x) = g_2(x)$ 特解形式 f_2 , 则有:

$$f = f_1 + f_2$$



对于这个题目:

$$y(t) - 2y(t-1) - 3y(t-2) = -2^t + 100t$$

先求通解, $p^2 - 2p - 3 = (p-3)(p+1) = 0$, 则有通解 $y_{\text{通}} = C_1(-1)^t + C_2 3^t$



在求特解,1、待定系数为 $D2^t$, 则有 $D2^t - D2\frac{1}{2}2^t - 3D\frac{1}{4} = -2^t$, 解得 $D = \frac{4}{3}2^t$
 2、待定系数 $at + b$, 则有 $at + b - 2a(t-1) - 2b - 3a(t-2) - 3b = 100t$, 则根据

多项式系数对应关系有 $\begin{cases} a - 2a - 3a = 100 \\ b + 2a - 2b + 6a - 3b = 0 \end{cases}$, 解得 $a = -25, b = -50$.

3.8.3 解法二: 频域上的 Z 变换解法

首先给出单边 Laplace 变换: ($s = \sigma + j\omega$)

$$F(s) = \int_0^\infty f(t)e^{-st}dt$$

现在我们考虑 $x(t)$ 在经过理想抽样后的信号单边拉普拉斯变换:

$$x_s(t) = \sum_{n=0}^{\infty} x(nT)\delta(t - nT)$$

$$\int_0^\infty x_s(t)e^{-st}dt = \int_0^\infty \left[\sum_{n=0}^{\infty} x(nT)\delta(t - nT) \right] e^{-st} dt$$

$$\int_0^\infty x_s(t)e^{-st}dt = \sum_{n=0}^{\infty} \left[\int_0^\infty x(nT)\delta(t - nT) \right] e^{-st} dt$$

$$\int_0^\infty x_s(t)e^{-st}dt = \sum_{n=0}^{\infty} x(nT)e^{-snT}$$

我们令 $z = e^{st}$, 令取样间隔 $T = 1$ 这个时候式子就可以变成 $\sum_{n=0}^{\infty} x(n)z^n$

这个就是 z 变换的式子. 我们暂时不考虑 z 变换的收敛域: 默认所有式子都是定义在 x 的正半部分上, 所以说收敛区域是包括零点的那一部分.

下面这个表给了一部分函数的 Z 变换结果. 结果的计算读者可以自己验证.



$f(k)$	$F(z)$
$\delta(k)$	1
$\delta'(k)$	z
$v^k u(k)$	$\frac{z}{z - v}$

由于差分方程的式子往往还有 $f(k + n)$, 所以说我们给出时移公式:

$$\mathcal{Z}(x(n - n_0)) \longleftrightarrow z^{-n_0} X(z)$$

对于向左移动的公式, 我们假设函数向左移动了 n 个单位, 首先整体的 z 变换要乘 z^n , 但是我们得考虑到有 n 个点移到了 x 的负半轴的位置, 那么我们把这些点给减掉表现在 Z 变换的式子里面就是这一部分 $\sum_{i=0}^n f(i)z^{n-i}$ 会因为被移出零点之外而被减去. 所以说向左移动的公式可以这么写:

$$\mathcal{Z}(x(n + n_0)) \longleftrightarrow z^{n_0} X(z) - \sum_{i=0}^{n_0} x(i)z^{n-i}$$

现在我们捋顺了一些基础知识, 现在我们总结一下如何利用 Z 变换来解差分方程:

- 1、把输入 $e(k)$ 做 Z 变换求出 $E(z)$.
- 2、利用时移形式将等式两边变成:

$$F(z) = H(z)E(z)$$

- 3、解出 $F(z)$.

- 4、把 $F(z)$ 分解成若干个简单多项式之和, 然后做反 Z 变换, 一般来说, 都是凑成我们熟悉的函数, 然后反过来做反 Z 变换.

3.8.4 解法三: 获得数字系统的响应.

我们可以把这个方程看成一个电路系统, 这个电路系统是输出离散信号的数字信号系统, 电路系统在输入端获得**激励**, 经过系统之后, 在输出端获得**响应**, 响应分成零输入响应和零状态响应.



注意: 我们这里讨论的是因果系统, 信号也是因果信号(也就是说系统的响应和激励都要乘阶跃函数 $u(t)$)

定义 3.35

零状态响应是电路的储能元器件(电容、电感类元件)无初始储能, 仅由外部激励作用而产生的响应

零输入响应是在没有外加激励时, 仅由 $t = 0$ 时刻的非零初始状态引起的响应。



根据电路的知识: 全响应 = 零输入响应 + 零状态响应. 我们要求出全响应, 只需要分别求出零输入响应和零状态响应.

首先是零输入响应, 零输入响应其实非常简单, 和上一部分的通解形式是一样的.

接着是零状态响应, 这个比较麻烦, 我们先约定一个新的运算, 叫做离散卷积:

定义 3.36

(系统函数) 给定一个因果线性系统, 对于激励 $\delta(t)$, 对应的响应 $h(t)$ 计为系统的系统函数



现在我们捋顺了一些基础知识, 现在我们把零状态响应的求法再总结一遍:(假设 $y(k)$ 是响应, $e(k)$ 是输入)

$$y(k+n) + a_{n-1}y(k+n-1) + \dots + a_0y(k) = b_m e(k+m) + \dots + b_0 e(k)$$

这个时候我们定义转移算子 $H(S)$:

$$\frac{b_m S^m + b_{m-1} S^{m-1} + \dots + b_0}{S^n + a_{n-1} S^{n-1} + \dots + a_0}$$

我们可以做这么一件事, 就是把 $H(S)$ 拆成若干个多项式的和, 每一个多项式可以转化成系统函数 $h(t)$ 的一部分. 可以查表把 $H(S)$ 转化成 $h(t)$.



$H(S)$	$h(t)$
$\frac{1}{s-v}$	$v^{k-1}u(k-1)$
$\frac{v}{s-v}$	$v^k u(k)$
$\frac{v}{(s-v)^n}$	$\frac{1}{(n-1)!} k(k-1)\dots(k-n+2)v^{k-n+1}u(k)$
s^n	$\delta(k)^{(n-1)}$

最后计算出零状态响应 $h(k) * e(k)$, $e(k)$ 就是传入系统的输入.

3.8.5 解法四: 生成函数求数列通项

在生成函数中, 其实生成函数中的 x^n 并非是主角, 反而仅仅是作为一个占位符, 用来表示这里是数列第 n 项的. 如果你已经了解了一些知识, 可能会不由得开始思考这个多项式函数在给定的 x 下是否收敛的问题. 例如 $1+x+x^2+\dots=\frac{1}{1-x}$ 这个式子, 他在 $0 < x < 1$ 时能够得到正确的值, 但在 $x > 1$ 时, 式子左边是正数之和, 右侧却是一个负数, 显然是“荒谬”的. 但在这里, 我们一般并不会考虑这个问题, 毕竟如之前所说, 这个多项式函数的系数才是真正的主角, 我们大可以假定 x 的取值范围就是能够使得这个和式绝对收敛的范围.

那么我们为什么要如此大费周章地构造一个的多项式函数呢? 因为这样的情况下我们可以使用分析中一些对多项式函数进行变换的技巧, 来帮助我们了解这个系数数列更多的知识.

例题 3.7 已知一个数列 $\{a_n\}$ 满足

$$a_0 = 1, a_1 = 4, a_n = 3a_{n-1} - 2a_{n-2}$$

我们构造其对应的生成函数 $F(x)$, 不难发现

$$xF(x) = \sum_{n=0}^{\infty} a_{n-1}x^n, x^2F(x) = \sum_{n=0}^{\infty} a_{n-2}x^n$$

再根据数列的递推式, 我们可以说

$$F(x) = 3xF(x) - 2x^2F(x)$$

在不考虑边界条件下成立.

考虑边界条件后更准确的说法是

$$(1 - 3x + 2x^2)F(x) = 1 + x$$



可解得

$$F(x) = \frac{1+x}{(2x-1)(x-1)} = \frac{2}{x-1} - \frac{3}{2x-1}$$

现在我们可以说，这个简单的函数

$$F(x) = \frac{2}{x-1} - \frac{3}{2x-1}$$

就包含了数列 $\{a_n\}$ 的所有信息，现在我们只需要把它展开即可.

$$\begin{aligned}\frac{2}{x-1} &= -2 \sum_{n=0}^{\infty} x^n \\ \frac{3}{1-2x} &= 3 \sum_{n=0}^{\infty} (2x)^n = \sum_{n=0}^{\infty} 3 \cdot 2^n \cdot x^n\end{aligned}$$

所以 $F(x) = \sum_{n=0}^{\infty} (3 \cdot 2^n - 2)x^n$, 由此 $a_n = 3 \cdot 2^n - 2$.

操作多项式函数的方法除了相加、相乘，还有求导、求积等.

定理 3.11

(组合数拓展定理) $C(n, m) = \frac{n(n-1)\cdots(n-m+1)}{m!}$. 这里，我们扩展组

合数的定义范围，允许其中的 $n \in \mathbb{R}$.



对于 $n, m \in \mathbb{N}^*$, 组合数拓展的一个推论是 $C(-n, m) = (-1)^m C(n+m-1, m)$.

除了利用定理 7.2 本身，另一种得出这个结果的方式是，对于 $\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$,

逐步进行求导.

问题 3.15 (Fibonacci 数列通项) 迷途之家中，假设有一片区域的小猫总是雌雄成对出生，且不会死亡，不与其他区域的猫流通。一对小猫幼崽一年后长成一对成年猫，猫在成年后每年会产生一对幼崽。求第 n 年在这一区域有多少对小猫。(答案: $a_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \right)$)

¹同类元素视为相同物体，只是有很多个。

²当你想扩展某个东西的定义范围时，请确保新计算方式和以前的不会产生冲突。



3.8.6 微分方程的差分解—Euler 方法

最后我想再提一个用法例子，就是 Euler 法求解微分方程的近似解。对于微分方程，在实际生产与科研中，除少数简单情况能获得初值问题的初等解（用初等函数表示的解）外，绝大多数情况下是求不出初等解的。有些初值问题即便有初等解，也往往由于形式过于复杂而不便处理。

实用的方法是在计算机上进行数值求解：即不直接求 $y(x)$ 的显式解，而是在解所存在的区间上，求得 y 在一系列点 x_n ($n = 1, 2, \dots$) 上的近似值。这一系列点应该是等间距的。间距假设为 h

约定：

$y(x_n)$ ：待求函数 $y(x)$ 在 x_n 处的精确函数值。

y_n ：待求函数 $y(x)$ 在 x_n 处的近似函数值。

Euler 法的意思就是给定微分方程

$$y' = f(x, y)$$

$$y(0) = C$$

可以求出一个带有初解的递推式：

$$y_{n+1} = \phi(x_n, y_n, x_{n+1})$$

$$y(0) = C$$

这个时候我们看一下怎么做。下面有三个对于积分的近似公式：

$$\int_{x_n}^{x^{n+1}} y' dx = y_{n+1} - y_n = h f(x^n, y^n)$$

$$\int_{x_n}^{x^{n+1}} y' dx = y_{n+1} - y_n = h f(x^{n+1}, y^{n+1})$$

$$\int_{x_n}^{x^{n+1}} y' dx = y_{n+1} - y_n = \frac{h}{2} [f(x^{n+1}, y^{n+1}) + f(x^n, y^n)]$$

这三个公式都是近似正确的，因为我们可以控制步长 h ，所以说我们可以控制误差。

下面给一个例题：



例题 3.8

$$y' = -y + x + 1$$

$$y(0) = C$$

$$h = 0.1$$

对于第一条公式我们可以写出:

$$y_{n+1} = y_n + h(-y_n + x_n + 1)$$

$$y_{n+1} = 0.9y_n + 0.1x_n + 0.1$$

对于第二条公式我们可以写出:

$$y_{n+1} = y_n + h(-y_{n+1} + x_{n+1} + 1)$$

$$y_{n+1} = \frac{y_n + 0.1x_n + 0.11}{1.1}$$

对于第三条公式我们可以写出:

$$y_{n+1} = y_n + \frac{h}{2}[(-y_{n+1} + x_{n+1} + 1) + (-y_n + x_n + 1)]$$

$$y_{n+1} = \frac{1.9y_n + 0.2x_n + 0.21}{2.1}$$

你有没有发现, 最后求得的差分解本质上竟然是一个递推关系式, 也就是这是一个差分方程! 那我们是不是可以用差分方程的思想来解呢?

最后我们挑一个方法来试着分析一下这个式子造成的误差是多少:

$$y_{n+1} = y_n + hf(y_{n+1}, x_{n+1})$$

由 Taylor 公式, 我们可以写出:

$$y(x_{n+1}) = y(x_n + h) = y(x_n) + hy'(x_n) + O(h^2)$$

这个时候由原式子可以写出来: $y'(x_n) = f(x_n, y_n)$

在工程上, 我们可以假设 $y_n = y(x_n), y(x_{n+1}) = y_{n+1}$.

这个时候我们可以写出:

$$y(x_{n+1}) = y_{n+1} = y(x_n) + hy'(x_n) - y_n - hf(y_{n+1}, x_{n+1})$$

$$y(x_{n+1}) = y_{n+1} = hy'(x_n) - hy'(x_{n+1})$$

$$y(x_{n+1}) = y_{n+1} = hy'(x_n) - hy'(x_n + h)$$



$$y(x_{n+1}) = y_{n+1} = hy'(x_n) - h(y'(x_n) + O(h))$$

得到误差 $T_n = O(h^2)$

3.8.7 拓展--卷积神经网络

未来会有

3.9 回顾和展望

本部分对应着计算机的 5 门数学专业课(微积分、线性代数、概率论与数理统计、离散数学、数值分析)进行简短的介绍,计算机的基础就是数学理论,学习数学理论是十分重要的,这对我们从事科研和工程都起着十分重要的作用.

本部分只是基本地展示数学学习的基本内容,如果您想从事算法、AI 甚至是理论计算机的研究,则需要学习更深入的数学,下面我们推荐一点数学相关的资料.有时间感兴趣可以观看:

《数学分析讲义》中国科技大学出版社常庚哲版本对于分析学的拓展是很重要的,了解一定的数学分析可以帮助你了解更多的数学知识.

Introduction to The Linear Algebra 5th edition Strang 著计算机对于线性代数的要求更高,我们学校的线性代数课有点上得云里雾里,不知道线代有什么用,这本书的第 10 章给我们展示了许多线性代数的应用.

随机过程:耶鲁大学讲义 随机过程是概率论的升级版,假如说您对 AI 感兴趣,请务必更加努力学习概率统计,因为本质上很多机器学习是统计学系模型,如果您对概率统计有更深入的了解,您会更加得心应手地学习人工智能相关的.

抽象代数:Thomas W. Hungerford 的《Algebra》和 David S. Dummit 的《Abstract Algebra》 如果您对计算机的纯理论感兴趣,您可以试试阅读一下抽象代数.

离散数学全版本第三本 Rosen 著 这个是我们离散数学的教材,但是如果您有时间可以阅读一下全版本,您可以尝试了解数论和密码相关的资料.

当然你也可以观看通过观看网课来进行学习:

微积分 (推荐: 大一):MIT18.01/18.02: Calculus

线性代数 (推荐: 大一):MIT18.06: Linear Algebra

信息论 (推荐: 大一):MIT6.050J: Information theory and Entropy

离散数学 (推荐: 大一):UCB CS70: discrete Math and probability theory

概率论 (推荐: 大一):UCB CS126: probability theory



离散数学和概率论 (推荐: 大一) MIT 6.042J: Mathematics for Computer Science

数值分析 (推荐: 大二) MIT18.330: Introduction to numerical analysis

凸优化理论 (推荐: 大二) Standford EE364A: Convex Optimization

信息论 (推荐: 大二) The Information Theory, Pattern Recognition, and Neural Networks





第 4 章 ➤ 计算机开发基础知识

4.1 Linux 基本知识

用户权限

Linux 系统是一种典型的多用户系统，不同的用户处于不同的地位，拥有不同的权限。

为了保护系统的安全性，Linux 系统对不同的用户访问同一文件（包括目录文件）的权限做了不同的规定。

在 Linux 中我们通常使用以下两个命令来修改文件或目录的所属用户与权限：

- **chown (change owner)**：修改所属用户与组。`chown [-R] 属主名 文件名`
- **chmod (change mode)**：修改用户的权限。`chmod [-R] xyz 文件或目录`, 其中 xyz 表示三个 0~7 之内的数字。
- **xyz**：就是刚刚提到的数字类型的权限属性，为 **rwx** 属性数值的相加。
- **-R**：进行递归 (recursive) 的持续变更，以及连同次目录下的所有文件都会变更。

其中我们可以使用 `ls -l` 指令来查看文件所属的用户和组。

其中这九个权限是三个三个一组的！其中，我们可以使用数字来代表各个权限，各权限的分数对照表如下：如果权限的数字对应是 6, 那么就代表可读可写。其中文件的权限由三组数字组成，分别代表本人，本人所属组和其他人的权限。其

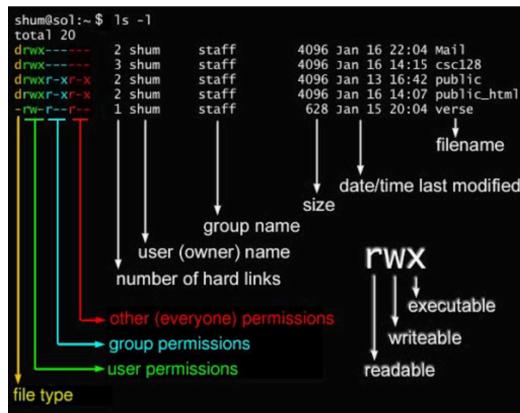


图 4.1

中这数字可以转换.Linux 用这几种数字来进行文件权限的标记.

- r:4
- w:2
- x:1

每种身份 (owner/group/others) 各自的三个权限 (r/w/x) 分数是需要累加的，例如当权限为：**-rwxrwx---** 分数则是：

- owner = rwx = 4+2+1 = 7
- group = rwx = 4+2+1 = 7
- others= --- = 0+0+0 = 0

所以等一下我们设定权限的变更时，该文件的权限数字就是 **770**。变更权限的指令 chmod 的语法是这样的：

`chmod 770 xxx`

常用命令

ls (列出目录) 在 Linux 系统当中，ls 命令可能是最常被运行的。

语法：

```
[Sukuna@sukuna]# ls [-aAdfFilnrRSt] 目录名称
[Sukuna@sukuna]# ls [--color={never,auto,always}] 目录名称
[Sukuna@sukuna]# ls [--full-time] 目录名称
```

选项与参数：

- **-a**：全部的文件，连同隐藏文件 (开头为. 的文件)一起列出来 (常用)
- **-d**：仅列出目录本身，而不是列出目录内的文件数据 (常用)
- **-l**：长数据串列出，包含文件的属性与权限等等数据；(常用)



cd (切换目录) cd 是 Change Directory 的缩写，这是用来变换工作目录的命令。因为 Linux 寻找文件是在工作目录为基准然后进行路径查找

语法：

cd [相对路径或绝对路径]

pwd (显示目前所在的目录)

pwd 是 Print Working Directory 的缩写，也就是显示目前所在目录的命令。

```
[Sukuna@Sukuna]# pwd [-P]
```

选项与参数：

- **-P**：显示出确实的路径，而非使用连结(link)路径。

mkdir (创建新目录) 如果想要创建新的目录的话，那么就使用 mkdir (make directory) 吧。

语法：

```
mkdir [-mp] 目录名称
```

选项与参数：

- **-m**：配置文件的权限喔！直接配置，不需要看默认权限(umask)的脸色～
`mkdir -m 711 test2`
- **-p**：帮助你直接将所需要的目录(包含上一级目录)递归创建起来！这可以帮助你一次性生成多级的目录

rmdir (删除空的目录) 语法：

```
rmdir [-p] 目录名称
```

选项与参数：

- **-p**：从该目录起，一次删除多级空目录。和上面的 mkdir 类似，可以直接一次性删除多级的目录。

```
[Sukuna@Sukuna tmp]# ls -l    <==看看有多少目录存在？
drwxr-xr-x  3 root  root 4096 Jul 18 12:50 test
drwxr-xr-x  3 root  root 4096 Jul 18 12:53 test1
drwx--x--x  2 root  root 4096 Jul 18 12:54 test2
[Sukuna@Sukuna tmp]# rmdir test    <==可直接删除掉，没问题
[Sukuna@Sukuna tmp]# rmdir test1   <==因为尚有内容，所以无法删除！
```



```
rmdir: `test1': Directory not empty
[Sukuna@Sukuna tmp]# rmdir -p test1/test2/test3/test4 <=可以一次性删除子目录
[Sukuna@Sukuna tmp]# ls -l           <==您看看，底下的输出中test与test1不见了
drwx--x--x  2 root  root 4096 Jul 18 12:54 test2
```

cp (复制文件或目录) cp 即拷贝文件和目录。

语法:

```
[Sukuna@Sukuna]# cp [-dfilprs] 来源档(source) 目标档(destination)
//把若干个文件放到directory位置/
[Sukuna@Sukuna]# cp [options] source1 source2 source3 .... directory
```

选项与参数:

- **-d**: 若来源档为链接文件的属性(link file), 则复制连结档属性而非文件本身;
- **-f**: 为强制(force)的意思, 若目标文件已经存在且无法开启, 则移除后再尝试一次;
- **-i**: 若目标档(destination)已经存在时, 在覆盖时会先询问动作的进行(常用)
- **-l**: 进行硬式连结(hard link)的连结档创建, 而非复制文件本身;
- **-p**: 连同文件的属性一起复制过去, 而非使用默认属性(备份常用);
- **-r**: 递归持续复制, 用於目录的复制行为;(常用)
- **-s**: 复制成为符号连结档(symbolic link), 亦即『捷径』文件;
- **-u**: 若 destination 比 source 旧才升级 destination !

用 root 身份, 将 root 目录下的.bashrc 复制到 /tmp 下, 并命名为 bashrc.

```
[Sukuna@Sukuna]# cp ~/.bashrc /tmp/bashrc
```

rm (移除文件或目录) 语法:

`rm [-fir]` 文件或目录

选项与参数:

- **-f**: 就是 force 的意思, 忽略不存在的文件, 不会出现警告信息;
- **-i**: 互动模式, 在删除前会询问使用者是否动作
- **-r**: 递归删除.(不建议使用)

mv (移动文件与目录, 或修改名称) 语法:

```
[Sukuna@Sukuna]# mv [-fiv] source destination
[Sukuna@Sukuna]# mv [options] source1 source2 source3 .... directory
```



选项与参数：

- -f：force 强制的意思，如果目标文件已经存在，不会询问而直接覆盖；
- -i：若目标文件(destination) 已经存在时，就会询问是否覆盖！
- -u：若目标文件已经存在，且 source 比较新，才会升级(update)

cat 由第一行开始显示文件内容

语法：

```
cat [-AbEnTv]
```

选项与参数：

- -A：可列出一些特殊字符而不是空白而已；
- -b：列出行号，仅针对非空自行做行号显示，空自行不标行号！
- -E：将结尾的断行字节 \$ 显示出来；
- -n：列印出行号，连同空自行也会有行号，与 -b 的选项不同；
- -T：将 [tab] 按键以 \t 显示出来；
- -v：列出一些看不出来的特殊字符

添加用户

```
useradd 选项 用户名
```

参数说明：

- 选项：
 - -c comment 指定一段注释性描述。
 - -d 目录指定用户主目录，如果此目录不存在，则同时使用-m 选项，可以创建主目录。
 - -g 用户组指定用户所属的用户组。
 - -G 用户组，用户组指定用户所属的附加组。
 - -s Shell 文件指定用户的登录 Shell。
 - -u 用户号指定用户的用户号，如果同时有-o 选项，则可以重复使用其他用户的标识号。
- 用户名：
 指定新账号的登录名。

删除帐号 如果一个用户的账号不再使用，可以从系统中删除。删除用户账号就是要将/etc/passwd 等系统文件中的该用户记录删除，必要时还删除用户的主目录。



删除一个已有的用户账号使用 `userdel` 命令，其格式如下：

`userdel 选项 用户名`

常用的选项是 `-r`，它的作用是把用户的主目录一起删除。

4.2 数据整理技巧

grep 指令

可以使用 `grep -i xxx` 找到包含 `xxx` 的输出，只要包含即可。

这样一条命令 `journalctl | grep -i intel`，它会找到所有包含 `intel`(不区分大小写)的系统日志。

比如说下面这个程序：注意，这里我们使用管道将一个远程服务器上的文件传递给本机的 `grep` 程序！

```
ssh myserver 'journalctl | grep sshd | grep "Disconnected from"' | less
```

多出来的引号是什么作用呢？这么说吧，我们的日志是一个非常大的文件，把这么大的文件流直接传输到我们本地的电脑上再进行过滤是对流量的一种浪费。因此我们采取另外一种方式，我们先在远端机器上过滤文本内容，然后再将结果传输到本机。`less` 为我们创建来一个文件分页器，使我们可以通过翻页的方式浏览较长的文本。为了进一步节省流量，我们甚至可以将当前过滤出的日志保存到文件中，这样后续就不需要再次通过网络访问该文件了：当然，当你需要多个 `grep` 过滤器的时候，这个时候需要用引号引起来，然后使用 `>` 符号把输出重定向到 `ssh.log` 文件中。

```
$ ssh myserver 'journalctl | grep sshd | grep "Disconnected from"' > ssh.log
$ less ssh.log
```

正则表达式

让我们从这一句正则表达式开始学习：`/.*Disconnected from /`。正则表达式通常以（尽管并不总是）`/`开始和结束。如果遇到了`/`，很有可能前面的是一个正则表达式。其中正则表达式会用下面这些符号来进行代替。

- 除换行符之外的”任意单个字符”
- * 匹配前面字符零次或多次，比如说 `a*` 代表匹配 `a` 这个字符 0~n 次，这个可以代指 0~n 个和前面字符一样的字符。



- $+$ 匹配前面字符一次或多次比如说 a^+ 代表匹配 a 这个字符 1~n 次, 这个可以代指 1~n 个和前面字符一样的字符.
- $[abc]$ 匹配 a, b 和 c 中的任意一个
- $(RX1|RX2)?$ 任何能够匹配 RX1 或 RX2 的结果
- $^$ 行首
- $$$ 行尾

回过头我们再看 `.*Disconnected from /`, 我们会发现这个正则表达式可以匹配任何以若干任意字符开头, 并接着包含”Disconnected from “的字符串。这也正式我们所希望的。

当然, 正则表达式会如何匹配? $*$ 和 $+$ 在默认情况下是贪婪模式, 也就是说, 它们会尽可能多的匹配文本。对于某些正则表达式的实现来说, 您可以给 $*$ 或 $+$ 增加一个? 后缀使其变成非贪婪模式。

这里我们需要做的是匹配一整行:

```
| sed -E 's/.*Disconnected from (invalid |authenticating )?user .* [^ ]+ [^ ]*/
```

让我们借助正则表达式在线调试工具 **regex debugger** 来理解这段表达式。OK, 开始的部分和以前是一样的, 随后, 我们匹配两种类型的“user”(在日志中基于两种前缀区分)。再然后我们匹配属于用户名的所有字符。接着, 再匹配任意一个单词 ($[^]^+$ 会匹配任意非空且不包含空格的序列)。紧接着后面匹配单“port”和它后面的一串数字, 以及可能存在的后缀 `[preauth]`, 最后再匹配行尾。

当然我们可以使用“捕获组 (capture groups)”来完成记录之。被圆括号内的正则表达式匹配到的文本, 都会被存入一系列以编号区分的捕获组中。捕获组的内容可以在替换字符串时使用(有些正则表达式的引擎甚至支持替换表达式本身), 例如`\1`、`\2`、`\3`等等, 因此可以使用如下命令: 命令使用的`\2`代表使用第 2 个参数

```
| sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^ ]+ [^ ]*/
```

sed 编辑器

`sed` 是一个基于文本编辑器 `ed` 构建的”流编辑器”。在 `sed` 中, 您基本上是利用一些简短的命令来修改文件, 而不是直接操作文件的内容(尽管您也可以选择这样做)。相关的命令行非常多, 但是最常用的是 `s`, 即替换命令。

`s` 命令的语法如下: `s/REGEX/SUBSTITUTION/`, 其中 REGEX 部分是我们需要使用的正则表达式, 而 SUBSTITUTION 是用于替换匹配结果的文本. 就是把 REGEX



对应的内容匹配, 把整个 REGEX 的内容替换成 SUBSTITUTION 对应的内容.

多重管道

现在我们可以通过:(| 是一个管道,| 左边的指令执行完之后会把输出传递给 | 右边的指令, 右边的指令会对左边指令传递的数据进行处理得到一个输出), 比如说 ssh myserver journalctl 这个指令会把输出交给 grep sshd 指令处理,grep sshd 指令执行完之后会有一个输出.

```
ssh myserver journalctl
| grep sshd
| grep "Disconnected from"
| sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^']*/\1/g'
```

`sort` 会对其输入数据进行排序。`uniq -c` 会把连续出现的行折叠为一行并使用出现次数作为前缀。`sort -n` 会按照数字顺序对输入进行排序（默认情况下是按照字典序排序-`k1,1` 则表示“仅基于以空格分割的第一列进行排序”）.

awk

它是一个比较贴近于脚本的语言, 主要由这几个部分组成

```
BEGIN { 一开始做什么 }
```

对于每一行的数据, 我们又做什么.

```
END { 最后做什么 }
```

比如说:

```
BEGIN { rows = 0 }
$1 == 1 && $2 ~ /[^c[^ ]]*e$/ { rows += $1 }
END { print rows }
```

对于每一行的数据的处理我们可以分成两个部分:

条件 { 条件成立了该怎么做 }

其中每一行可以被 awk 解释称用空格分成的几个部分,\$1 表示从左到右第一个部分.

```
2 this is a test
```

这个时候 \$1=2,\$2=this



4.3 Command Line 环境

程序的退出

程序的退出有三种方法, 分别是传递三种信号.

第一个是 SIGINT, 就是 Ctrl-C.

第二个就是 SIGQUIT, 就是 Ctrl-\.

第三个就是使用 kill 命令来传递 SIGTERM, 这个 kill 命令语法就是 `kill -TERM <PID>`, 其中 PID 就是进程号. 当然还可以使用 `kill -9 <PID>` 强制杀死. 当然 kill 指令中-后面的信号其实是可以指定的. 制定-6 也可以-STOP 也可以. 当然不指定信号类型 (`kill <PID>`) 直接使用 kill 就是退出,

程序的暂停

使用 SIGSTOP 信号, 就是 Ctrl-Z

当然, 还可以使用 `fg %num` 和 `bg %num` 来恢复暂停的工作, 它们分别表示在前台继续或在后台继续。

`jobs` 命令会列出当前终端会话中尚未完成的全部任务。您可以使用 pid 引用这些任务 (也可以用 `pgrep` 找出 pid)。更加符合直觉的操作是您可以使用百分号 + 任务编号 (`jobs` 会打印任务编号) 来选取该任务。如果要选择最近的一个任务, 可以使用 `$!` 这一特殊参数。

```
$ jobs
[1] + suspended sleep 1000
[2] - running    nohup sleep 2000
```

其中 1 和 2 就任务编号.

```
$ bg %1
[1] - 18653 continued sleep 1000
```

这一条语句就是让任务序号 1 继续执行. 不过就是在后台悄悄地运行.

让已经在运行的进程转到后台运行, 您可以键入 Ctrl-Z, 然后紧接着再输入 `bg`。注意, 后台的进程仍然是您的终端进程的子进程, 一旦您关闭终端 (会发送另外一个信号 SIGHUP), 这些后台的进程也会终止。为了防止这种情况发生, 您可以使用 `nohup` (一个用来忽略 SIGHUP 的封装) 来运行程序。



```
$ sleep 1000  
^Z
```

[1] + 18653 suspended sleep 1000

先进入暂停状态

```
$ nohup sleep 2000 &  
[2] 18745
```

Appending output to nohup.out

新造一个新的任务

```
$ jobs  
[1] + suspended sleep 1000  
[2] - running nohup sleep 2000
```

显示

```
$ bg %1  
[1] - 18653 continued sleep 1000
```

让一个程序停止暂停放入后台运行

```
$ jobs  
[1] - running sleep 1000  
[2] + running nohup sleep 2000
```

```
$ kill -STOP %1  
[1] + 18653 suspended (signal) sleep 1000
```

使用kill指令强行传递STOP信号给第一个任务编号对应的任务.

```
$ jobs  
[1] + suspended (signal) sleep 1000  
[2] - running nohup sleep 2000
```

```
$ kill -SIGHUP %1  
[1] + 18653 hangup sleep 1000
```

第一个任务被挂起了

```
$ jobs  
[2] + running nohup sleep 2000
```

第二个任务屏蔽挂起

```
$ kill -SIGHUP %2
```

```
$ jobs  
[2] + running    nohup sleep 2000  
  
$ kill %2  
[2] + 18745 terminated nohup sleep 2000  
强制杀死任务编号为2的任务。  
$ jobs
```

别名

输入一长串包含许多选项的命令会非常麻烦。因此，大多数 shell 都支持设置别名。shell 的别名相当于一个长命令的缩写，shell 会自动将其替换成原本的命令。例如，`bash` 中的别名语法如下：

```
alias alias_name="command_to_alias arg1 arg2"
```

注意，= 两边是没有空格的，因为 `alias` 是一个 shell 命令，它只接受一个参数。

配置文件

配置文件是一种特殊的文件，一般来说是一种文本文件，很多程序的配置都是通过纯文本格式的被称作点文件的配置文件来完成的（之所以称为点文件，是因为它们的文件名以 . 开头，例如 `~/.vimrc`。也正因为此，它们默认是隐藏文件，`ls` 并不会显示它们）。

对于 `bash` 来说，在大多数系统下，您可以通过编辑 `.bashrc` 或 `.bash_profile` 来进行配置。在文件中您可以添加需要在启动时执行的命令，例如上文我们讲到过的别名，或者是您的环境变量。

还有一些其他的工具也可以通过点文件进行配置：

- `bash` - `~/.bashrc`, `~/.bash_profile`
- `git` - `~/.gitconfig`
- `vim` - `~/.vimrc` 和 `~/.vim` 目录
- `ssh` - `~/.ssh/config`
- `tmux` - `~/.tmux.conf`

我们应该该如何管理这些配置文件呢，它们应该在它们的文件夹下，并使用版本控制系统进行管理，然后通过脚本将其 符号链接 到需要的地方。这么做有如下好处：



SSH 连接

通过如下命令，您可以使用 ssh 连接到其他服务器：

```
ssh foo@bar.mit.edu
```

这里我们尝试以用户名 `foo` 登录服务器 `bar.mit.edu`。服务器可以通过 URL 指定（例如 `bar.mit.edu`），也可以使用 IP 指定（例如 `foobar@192.168.1.42`）。

ssh 的一个经常被忽视的特性是它可以直接远程执行命令。`ssh foobar@server ls` 可以直接在用 `foobar` 的命令下执行 `ls` 命令。

当然我们也可以使用 scp 指令把一些文件复制到服务器中，也可以把某些文件从服务器扒下来。

从本地复制到远程 命令格式：

```
scp local_file remote_username@remote_ip:remote_folder
```

或者

```
scp local_file remote_username@remote_ip:remote_file
```

或者

```
scp local_file remote_ip:remote_folder
```

或者

```
scp local_file remote_ip:remote_file
```

- 第 1,2 个指定了用户名，命令执行后需要再输入密码，第 1 个仅指定了远程的目录，文件名字不变，第 2 个指定了文件名；这个文件名是要具体到某个目录中的，不能只是一个文件名，是文件目录和文件名的组合。
- 第 3,4 个没有指定用户名，命令执行后需要输入用户名和密码，第 3 个仅指定了远程的目录，文件名字不变，第 4 个指定了文件名；

应用实例：

```
scp /home/space/music/1.mp3 root@www.runoob.com:/home/root/others/music  
scp /home/space/music/1.mp3 root@www.runoob.com:/home/root/others/music/001.mp3  
scp /home/space/music/1.mp3 www.runoob.com:/home/root/others/music  
scp /home/space/music/1.mp3 www.runoob.com:/home/root/others/music/001.mp3
```

复制目录命令格式：

```
scp -r local_folder remote_username@remote_ip:remote_folder
```

或者

```
scp -r local_folder remote_ip:remote_folder
```



- 第1个指定了用户名，命令执行后需要再输入密码；
- 第2个没有指定用户名，命令执行后需要输入用户名和密码；

应用实例：

```
scp -r /home/space/music/ root@www.runoob.com:/home/root/others/  
scp -r /home/space/music/ www.runoob.com:/home/root/others/
```

上面命令将本地 music 目录复制到远程 others 目录下。

2、从远程复制到本地 从远程复制到本地，只要将从本地复制到远程的命令的后2个参数调换顺序即可，如下实例

应用实例：

```
scp root@www.runoob.com:/home/root/others/music /home/space/music/1.mp3  
scp -r www.runoob.com:/home/root/others/ /home/space/music/
```

4.4 代码调试和测试

代码调试

打印调试法

在合适的位置上使用打印语句进行调试。

断言调试法

在 debug 模式下使用 assert 关键字对程序的某些状态进行断言（即逻辑表达式），若断言为假则产生中断。可用来检测程序状态，并快速定位异常位置，处理不应该发生的非法情况。

日志

日志较普通的打印语句有如下的一些优势：

- 您可以将日志写入文件、socket 或者甚至是发送到远端服务器而不仅仅是标准输出；
- 日志可以支持严重等级（例如 INFO, DEBUG, WARN, ERROR 等），这使您可以根据需要过滤日志；



- 对于新发现的问题，很可能您的日志中已经包含了可以帮助您定位问题的足够的信息。

例如，执行 `echo -e "\e[38;2;255;0;0mThis is red\e[0m"` 会打印红色的字符串：This is red。其中前面的数据是 38、2 和 RGB 值，这几个数字用分号间隔开来。

第三方日志系统

我们在使用大型软件系统，往往要用到一些依赖，有些依赖会作为程序运行，如 Web 服务器、数据库、消息代理或者一些库或者系统包都是此类常见的第三方依赖。这些依赖也会产生日志。对于 UNIX 系统来说，程序的日志通常存放在 `/var/log`。例如，**NGINX** web 服务器就将其日志存放于 `/var/log/nginx`。

目前，各种操作系统开始使用 `system log` 来保存日志。大多数（但不是全部的）Linux 系统都会使用 `systemd`，这是一个系统守护进程，它会控制您系统中的很多东西，例如哪些服务应该启动并运行。`systemd` 会将日志以某种特殊格式存放在 `/var/log/journal`，您可以使用 `journalctl` 命令显示这些消息。也就是说可以从 `journalctl` 这个指令去读 `systemd` 这个守护进程的日志。对于 MacOS，对应地使用 `log show`。

对于大多数的 UNIX 系统，您也可以使用 `dmesg` 命令来读取内核的日志。

我们还可以使用 `logger` 指令把数据放到系统日志中。

调试器:debugger

调试器是一种可以允许我们和正在执行的程序进行交互的程序，它可以做到：

- 当到达某一行时将程序暂停；
- 一次一条指令地逐步执行程序；
- 程序崩溃后查看变量的值；
- 满足特定条件时暂停程序；
- 其他高级功能。

下面介绍一下 `gdb` 调试器：

可以不带任何参数或选项执行 `gdb` 命令，但是最常用的启动 `gdb` 的方式是带一个或者两个参数，指定一个可执行文件来作为参数：注意一定是可执行文件

`gdb program(gdb+可执行文件名称)`

`break [file:]function or b function(or symbol):`



设置一个断点在函数中(在文件中)

`bt Backtrace:`

显示堆栈

`print expr`

显示表达式的值

`c`

继续执行你的程序(程序停住后,例如:在断点处停止)

`next or n`

执行程序的下一行代码(程序停止以后);跨国任何当前行的函数调用。

`ni`

下一条汇编指令.

`edit [file:]function`

查看当前程序停在哪。

`list [file:]function`

显示程序当前停住的代码行附近的代码

`step` 单步调试

执行程序的下一行(程序停住后),进入当前行的函数调用的内部

`help [name]`

显示gdb命令的相关信息。

`quit`

退出gdb

二进制测试法

使用strace调用可以获得二进制程序执行了什么系统调用.(注意filename为可执行文件,一定要加上.\)例如`strace .\a.out`

`strace file_name`

`strace command`

执行名称为command的命令或程序并跟踪系统调用

`strace -p procid`

跟踪ID为的procid的进程系统调用情况

`strace -c -p procid`

统计ID为的procid的进程系统调用次数与用时,按CTRL+C结束统计,执行结果如下:



网络调试

有些情况下，我们需要查看网络数据包才能定位问题。像 **tcpdump** 和 **Wireshark** 这样的网络数据包分析工具可以帮助您获取网络数据包的内容并基于不同的条件进行过滤。

比如说 **tcpdump**:

```
tcpdump [ -DnqvX ] [ -c count ] [ -F file ] [ -i interface ] [ -r file ]
         [ -s snaplen ] [ -w file ] [ expression ]
```

抓包选项:

-c: 指定要抓取的包数量。

-i interface: 指定 **tcpdump** 需要监听的接口。默认会抓取第一个网络接口

-n: 对地址以数字方式显式，否则显式为主机名，也就是说-n 选项不做主机名解析。

-nn: 除了-n 的作用外，还把端口显示为数值，否则显示端口服务名。

-P: 指定要抓取的包是流入还是流出的包。可以给定的值为"in"、"out" 和"inout"，默认为"inout"。

-s len: 设置 **tcpdump** 的数据包抓取长度为 len，如果不设置默认将会是 65535 字节。

输出选项:

-e: 输出的每行中都将包括数据链路层头部信息，例如源 MAC 和目标 MAC。

-q: 快速打印输出。即打印很少的协议相关信息，从而输出行都比较简短。

-X: 输出包的头部数据，会以 16 进制和 ASCII 两种方式同时输出。

-XX: 输出包的头部数据，会以 16 进制和 ASCII 两种方式同时输出，更详细。

-v: 当分析和打印的时候，产生详细的输出。

-vv: 产生比-v 更详细的输出。

-vvv: 产生比-vv 更详细的输出。

还可以添加 expression 保证获取你需要的输出。

对于表达式语法，参考 [pcap-filter【pcap-filter - packet filter syntax】](#)

- 类型 type

- host, net, port, portrange

例如: host 192.168.201.128 , net 128.3, port 20, portrange 6000-6008'

- 目标 dir

- src, dst, src or dst, src and dst

- 协议 proto



tcp, udp, icmp, 若未给定协议类型, 则匹配所有可能的类型

不同的表达式还可以使用 and 和 or 连接在一起: 定义了表达式我们可以只输出符合这个表达式的信息.

7、静态分析, 可以使用静态分析工具帮助你分析代码, 对于代码的风格和质量, 也有很多工具帮助你写出来的代码和别人差不多.

性能分析

我们不能单凭打两个计时点的方式来计算时间, 对于工具来说, 需要区分真实时间、用户时间和系统时间。通常来说, 用户时间 + 系统时间代表了您的进程所消耗的实际 CPU. 时间分成三部分: 等待 + 执行用户态代码 + 执行系统态代码:

- 真实时间 - 从程序开始到结束流失掉的真实时间, 包括其他进程的执行时间以及阻塞消耗的时间 (例如等待 I/O 或网络);
- User - CPU 执行用户代码所花费的时间;
- Sys - CPU 执行系统内核代码所花费的时间。

当然还可以了解 CPU、内存和 I/O 的使用频率来评估性能.

模型检测

使用形式化的方法对软件系统的状态进行搜索, 自动验证并发系统正确性和可靠性。

4.5 语言以及代码编写平台

基础语言的代码编写平台

C/C++:VS2019、codeblocks、Clion(推荐)

Java:IDEA.

Python:Pycharm.

所有的代码都可以使用 VScode 编写.

具体的语言知识

具体的语言知识可以上菜鸟教程 (<https://www.runoob.com>) 处学习. 然后下面贴出了一些推荐的网课:



Python UCB CS61A: Structure and Interpretation of Computer Programs

C++ Stanford CS106L: Standard C++ Programming

Rust Stanford CS110L: Safety in Systems Programming

更高效的代码 MIT 6.031: Software Construction

顺序

大一新生建议立刻学习 C++, Java, Python 等语言, 大二学生可以了解 Haskell、Rust、Go 等语言, 语言的教程在菜鸟教程上都有。具体的语言需求由于篇幅所限暂时不写, 对于不同方向的语言要求将会在第 6、7、8 章具体介绍。届时对于不同的方向有不同的要求。

4.6 项目管理

4.6.1 意义

为何要使用 cmake 和 autotools 之类的项目构建工具? 我想, 这恐怕是刚刚接触软件项目的人最应该问的问题之一了。

“Hello, world!”这个最经典的程序相信我们每个人都写过。无论在什么平台上, 编译和运行这个程序都仅需要非常简单的操作。但事实上, hello,world 最多只能算是一个实例程序, 根本算不上一个真正的软件项目。

任何一个软件项目, 除了写代码之外, 还有一个更为重要的任务, 就是如何组织和管理这些代码, 使项目代码层次结构清晰易读, 这对以后的维护工作大有裨益。试想一下, 如果把一个像 xv6 操作系统那么大的项目像 hello world 那样, 把全部代码都放到一个 main.cpp 文件中, 那将会是多么恐怖的一件事情。别说 xv6 操作系统, 就是我们随便一个几千行代码的小项目, 也不会有人干这种蠢事。

决定代码的组织方式及其编译方式, 也是程序设计的一部分。因此, 我们需要 cmake 和 autotools 这样的工具来帮助我们构建并维护项目代码。这样子我们可以对项目有一个简单且清晰的认识。

4.6.2 依赖与版本控制

就您的项目来说, 它的依赖可能本身也是其他的项目。您也许会依赖某些程序(例如 python)、系统包(例如 openssl)或相关编程语言的库(例如 matplotlib)。现在, 大多数的依赖可以通过某些软件仓库来获取, 这些仓库会在一个地方托



管大量的依赖，我们则可以通过一套非常简单的机制来安装依赖。例如 Ubuntu 系统下面有 Ubuntu 软件包仓库，您可以通过 `apt` 这个工具来访问，RubyGems 则包含了 Ruby 的相关库，PyPi 包含了 Python 库。每个软件都是会更新版本的，假如我的库要发布一个新版本，在这个版本里面我重命名了某个函数。如果有人在我的库升级版本后，仍希望基于它构建新的软件，那么很可能构建会失败，因为它希望调用的函数已经不复存在了。所以说软件的版本会构造一个版本号。

这种版本号具有不同的语义，它的格式是这样的：主版本号. 次版本号. 补丁号。相关规则有：

- 如果新的版本没有改变 API，请将补丁号递增；
- 如果您添加了 API 并且该改动是向后兼容的，请将次版本号递增；
- 如果您修改了 API 但是它并不向后兼容，请将主版本号递增。

这可以保证那么只要最新版本的主版本号只要没变就是安全的，次版本号不低于之前我们使用的版本即可。可以防止使用库的用户因为代码版本更新了而不能正常使用。

4.7 操作系统的选择

关于 Linux 发行版，尽管有相当多的版本，但大部分发行版在大多数使用情况下的表现是相同的。可以使用任何发行版去学习 Linux 与 UNIX 的特性和其内部工作原理。发行版之间的根本区别是发行版如何处理软件包更新。某些版本，例如 Arch Linux 采用滚动更新策略，用了最前沿的软件包（bleeding-edge），但软件可能并不稳定。另外一些发行版（如 Debian, CentOS 或 Ubuntu LTS）其更新策略要保守得多，因此更新的内容会更稳定，但会牺牲一些新功能。我们建议你使用 Debian 或 Ubuntu 来获得简单稳定的台式机和服务器体验。

Mac OS 是介于 Windows 和 Linux 之间的一个操作系统，它有很漂亮的界面。但是，Mac OS 是基于 BSD 而不是 Linux，因此系统的某些部分和命令是不同的。另一种值得体验的是 FreeBSD。虽然某些程序不能在 FreeBSD 上运行，但与 Linux 相比，BSD 生态系统的碎片化程度要低得多，并且说明文档更加友好。除了开发 Windows 应用程序或需要使用某些 Windows 系统更好支持的功能（例如对游戏的驱动程序支持）外，我们不建议使用 Windows。

对于双系统，我们认为最有效的是 macOS 的 bootcamp，长期来看，任何其他组合都可能会出现问题，尤其是当你结合了其他功能比如磁盘加密。



4.8 数据结构和算法大纲

4.8.1 数据结构

线性表

场景：《弹幕天邪鬼》第二日

在逻辑上，除了最后一个元素外，每一个元素有且只有一个后继；除了第一个元素外，每一个元素有且只有一个前驱。

顺序表

逻辑关系上相邻的两个元素在物理位置上也相邻。

适用场合：表中元素变动不大，但需要快速存取元素；或存储空间需求预先知道。

单链表

逻辑关系上相邻的两个元素在物理位置上不一定相邻，元素之间的关系由指针（引用）指示。

适用场合：表中元素频繁插入和删除；或存储空间需求不定的应用。

栈

一种特殊的线性表，只允许在一端进行插入或删除的线性表（后进先出）。

应用：函数调用、表达式计算、括号匹配。

单调栈 维护一个永远单调的栈，用于解决 next-greater-element 问题，比如队伍中每个人向右看到的第一个比自己高的人。

队列

一种特殊的线性表，只允许在表的一端进行插入，而在另一端进行删除的线性表（先进先出）。

应用：滑动窗口、LRU(最近最少使用)。

思考：如何用两个栈实现队列？



树与森林

场景：《东方幕华祭·春雪篇》西行庭千本樱图

树与森林是层次结构的逻辑表示。

树

除了根结点外，每个结点有且只有一个前驱结点（一般称为父结点）；每个结点有零个（叶结点）或多个后继结点（非叶结点）。

信息的搜索或问题的求解过程可以用树表示，树的分叉点可以理解为某种操作或判定，子结点可以理解为搜索空间的划分，通过不断划分搜索空间，理论上可以达到对数的搜索速度，因此树是一种最常见的索引。

二叉树

每个结点最多只有2个子结点的树，是最常见的树。

思考：如何用二叉树表示一般的树？

二叉树有许多变种，如满二叉树、完全二叉树、二叉排序树、二叉平衡树，是各种算法的基础。

二叉树的遍历一般分为前序、中序、后序和层序，前三种通过根结点的访问次序区分，一般利用栈辅助遍历，而层序顾名思义按照结点的层次顺序进行遍历，一般利用队列辅助遍历。

森林

二木成林，三木成森。

森林是多棵树的集合，具有多个根结点，每个根节点递归产生各自的树。

在计算机中森林的表示不方便，一般转换成二叉树的表示形式（如何表示？）。

在人工智能中，有一种监督学习算法称为随机森林，通过若干判定树的联合进行预测。

图

场景：《东方永夜抄》苏活「生命游戏·Life Game·」

图是网状结构的逻辑表示，每个结点都与零个或多个结点建立连接。
图也是最通用的结构，一切事物关系都可以建模为图。



图的遍历有深度优先和广度优先。

当你点开了一个东方沙雕 MMD，看完之后在下方推荐中点开下一个，然后再在推荐中点开下一个……当找不到你想看的了，返回上一页继续点开下一个……这叫深度优先。

当你点开了一个东方沙雕 MMD，看完之后把推荐中所有想看的放进稍后在看，然后从稍后在看尾部的视频重复这个过程……这叫广度优先。

当考虑到从起点经过的代价，每次从代价最小的路径选择称为代价树搜索。

当考虑到距终点的预期代价，每次从预期最小的路径选择称为启发式搜索。

当同时考虑当下的代价和未来的预期就称为 A 算法。

你的下一步该如何走？

图也是离散数学的重要研究对象，针对图提出了一系列问题和算法，如最短路径、关键路径、最小生成树、最大团。

4.8.2 算法

分治

场景《东方红魔乡》QED 「495 年的波纹」

分治就是分而治之，在躲避复杂的交叉弹时，一般采用分解弹幕成多层，每次只躲避一层。

分治法所能解决的问题一般具有以下几个特征：

- 该问题的规模缩小到一定的程度就可以容易地解决。
- 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。
- 利用该问题分解出的子问题的解可以合并为该问题的解。
- 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。

典型问题有棋盘覆盖、二分查找、归并排序、数组中的逆序对、快速排序、线性时间选择、最接近点对、循环赛日程表。

动态规划

想不出来有什么场景惹

动态规划与分治法类似，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。



动态规划与分治法不同的是，经分解得到的子问题往往不是相互独立的，有大量子问题会重复出现。

为了避免重复计算，动态规划是用一个表来存放计算过的子问题。

典型问题有斐波那契数、最长公共子序列、最长回文子串、最长递增子序列、编辑距离、0-1 背包。

贪心

场景《东方神灵庙》欲灵「贪分欲吞噬者」

贪心就是每次选择当前最优解，最后达到全局最优解。

局部最优解不一定能达到全局最优解，贪心能解决的问题需要满足贪心选择性质和最优子结构性质，即最优解可以通过一系列局部最优的选择达到，而问题的最优解包含其子问题的最优解。

贪心选择性质的证明往往非常困难，实际中如果一个问题在直觉上能够使用贪心那大胆地尝试吧！搏一搏，白莲骑摩托

典型问题有哈夫曼编码、单源最短路径、最小生成树、会场安排、最低加油次数。

回溯

场景《东方绀珠传》完美无缺模式

SL 大法好

回溯就像玩 RPG 游戏，为了最终的 HE 而不断 SL。

回溯实际上是一个类似枚举的搜索尝试过程，主要是在搜索尝试过程中寻找问题的解，当发现已不满足求解条件时，就“回溯”返回，尝试别的路径。

回溯可以用约束函数在扩展节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。

典型问题有 0-1 背包、旅行售货员、图的着色、N 皇后、最大团。

4.9 如何学习算法和数据结构？

学习算法和数据结构的推荐材料

《算法导论》第三版，机械工业出版社。

oiwiki



洛谷 大话数据结构

刷题

学习算法需要多去洛谷和力扣刷题.

洛谷刷题建议去官方题单刷题, 如果感觉自己水平比较差可以从头刷起, 感觉自己可以的可以直接从算法 1-1 开始刷, 力扣刷题可以刷剑指 offer 或者按照专题有针对性地刷.

洛谷新人建议刷橙色或者黄色的题目, 觉得自己可以了可以开始刷绿题.

推荐的网课

UCB CS61B: Data Structures and Algorithms

Coursera: Algorithms I & II

Stanford CS106B/X

UCB CS170: Efficient Algorithms and Intractable Problems.





第 5 章 ➤ 常见工具介绍

5.1 Shell 脚本

我们关注的是 Bash，也就是 Bourne Again Shell，由于易用和免费，Bash 在日常工作中被广泛使用。同时，Bash 也是大多数 Linux 系统默认的 Shell。

运行 Shell 脚本有两种方法：

1、作为可执行程序

将上面的代码保存为 test.sh，并 cd 到相应目录：

```
chmod +x ./test.sh #使脚本具有执行权限  
./test.sh #执行脚本
```

注意，一定要写成 **./test.sh**，而不是 **test.sh**，运行其它二进制的程序也一样，直接写 test.sh，linux 系统会去 PATH 里寻找有没有叫 test.sh 的，而只有 /bin, /sbin, /usr/bin, /usr/sbin 等在 PATH 里，你的当前目录通常不在 PATH 里，所以写成 test.sh 是会找不到命令的，要用 ./test.sh 告诉系统说，就在当前目录找。Linux 维护了一个环境变量 PATH, Linux 的程序寻找数据和可执行的程序会默认在 PATH 这个路径下面找。

2、作为解释器参数

这种运行方式是，直接运行解释器，其参数就是 shell 脚本的文件名，如：

```
/bin/sh test.sh  
/bin/php test.php
```

变量

定义变量 :`Sukuna="qwq"`

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

- 命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。
- 中间不能有空格，可以使用下划线 `_`。
- 不能使用标点符号。
- 不能使用 bash 里的关键字（可用 `help` 命令查看保留关键字）。

使用变量，只需要在变量名前面加美元符号即可，为了方便机器识别变量，我们可以加上一个花括号。

```
your_name="Sukuna"  
echo $your_name  
echo ${your_name}
```

删除变量：使用 `unset` 命令即可 `unset Sukuna`.

只读变量：在声明下一行添加一句:`readonly Sukuna`

字符串

推荐使用双引号引用起来。双引号的字符串允许有转义符号，还允许添加变量。比如说:`str="Hello, I know you are \"\$your_name\"! \\n"`，转义符号的用法同 C 语言，添加变量的用法和使用变量一样。

当然，用双引号引起来也是可以的：`str="Hello, I know you are \"\$your_name\"! \\n"`，当然，`str="Hello, I know you are \$\{your_name\}! \\n"` 也是可以的。

获取字符串的长度 :`#{#string}`

提取子字符串 :`#{string:n:m}` 提取第 `n+1` 到第 `m` 个字符。

echo 命令 Shell 的 `echo` 指令与 PHP 的 `echo` 指令类似，都是用于字符串的输出。命令格式：

```
echo string
```



下面举例说明 echo 的特殊用法:

1. 显示普通字符串:

```
echo "It is a test"
```

这里的双引号完全可以省略, 以下命令与上面实例效果一致:

```
echo It is a test
```

2. 显示转义字符

```
echo "\"It is a test\""
```

结果将是:

```
"It is a test"
```

同样, 双引号也可以省略

3. 显示变量

read 命令从标准输入中读取一行, 并把输入行的每个字段的值指定给 shell 变量

```
#!/bin/sh
read name
echo "$name It is a test"
```

以上代码保存为 test.sh, name 接收标准输入的变量, 结果将是:

```
[root@www ~]# sh test.sh
OK                      #标准输入
OK It is a test        #输出
```

4. 显示换行

```
echo -e "OK! \n" # -e 开启转义
echo "It is a test"
```

输出结果:

```
OK!
```

```
It is a test
```



5. 显示不换行

```
#!/bin/sh  
echo -e "OK! \c" # -e 开启转义 \c 不换行  
echo "It is a test"
```

输出结果：

```
OK! It is a test
```

6. 显示结果定向至文件

```
echo "It is a test" > myfile
```

7. 原样输出字符串，不进行转义或取变量(用单引号)

```
echo '$name\' '
```

输出结果：

```
$name\'
```

8. 显示命令执行结果

```
echo `date`
```

注意：这里使用的是反引号 `，而不是单引号 '。

结果将显示当前日期

```
Thu Jul 24 10:08:46 CST 2014
```

当然我们还可以了解 printf 命令, 其 printf 的命令除了调用不需要加 () 后面的参数不需要加逗号外和 C 语言几乎一样.

printf 命令的语法：

```
printf format-string [arguments...]
```

注释 用 # 开头的行就是注释.

传递参数.

```
$ chmod +x test.sh  
$ ./test.sh 1 2 3
```



可以给脚本类似于 Shell 的形式传递参数。就像上面的例子一样，程序执行 ./test.sh，然后传递三个参数，一个是 1，一个是 2，一个是 3。在 Shell 脚本中分别代表 \$1,\$2,\$3。其中脚本的文件名存储在 \$0 这个变量名中。

```
echo "file name $0"
echo "first parameter: $1"
```

另外，还有几个特殊字符用来处理参数：

参数处理 说明

\$#	传递到脚本的参数个数
\$*	以一个单字符串显示所有向脚本传递的参数。如 "*""1 2n" 的形式输出所有参数
\$\$	脚本运行的当前进程 ID 号
\$!	后台运行的最后一个进程的 ID 号
\$@	与 \$* 相同，但是使用时加引号，并在引号中返回每个参数。如 "\$@" 用 「」

*@ 区别：

- 相同点：都是引用所有参数。
- 不同点：只有在双引号中体现出来。假设在脚本运行时写了三个参数 1、2、3，，则 " * " 等价于 "1 2 3"（传递了一个参数），而 "@" 等价于 "1" "2" "3"（传递了三个参数）。

数组

定义数组 :array_name=(value0 value1 value2 value3)，当然一个一个地声明也是允许的：array_name[0]=value0。

读取数组 :valuen=\${array_name[n]}。也可以使用 @ 符号来使用数组中的所有元素 \${array_name[@]}。

获取数组的长度 :length=\${#array_name[@]}。

流程控制

if if 语句语法格式：



```
if condition  
then  
    command1  
    command2  
    ...  
    commandN  
fi
```

写成一行（适用于终端命令提示符）：

```
if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi
```

末尾的 fi 就是 if 倒过来拼写，后面还会遇到类似的。

if else if else 语法格式：

```
if condition  
then  
    command1  
    command2  
    ...  
    commandN  
else  
    command  
fi
```

for 循环 for 循环一般格式为：

```
for var in item1 item2 ... itemN  
do  
    command1  
    command2  
    ...  
    commandN  
done
```

in 后面可以是迭代器，换句话说可以是字符串、数组等。



while 循环 while 循环用于不断执行一系列命令，也用于从输入文件中读取数据。其语法格式为：当 condition 成立的时候就做下面的 command

```
while condition  
do  
    command  
done
```

until 循环 until 循环执行一系列命令直至条件为 true 时停止。

until 循环与 while 循环在处理方式上刚好相反。

一般 while 循环优于 until 循环，但在某些时候---也只是极少数情况下，until 循环更加有用。

until 语法格式：

```
until condition  
do  
    command  
done
```

函数

shell 中函数的定义格式如下：

```
[ function ] funname [()]
```

```
{
```

```
    action;
```

```
    [return int;]
```

```
}
```

说明：

- 1、可以带 function fun() 定义，也可以直接 fun() 定义，不带任何参数。
- 2、参数返回，可以显示加： return 返回，如果不加，将以最后一条命令运行结果，作为返回值。



函数也可以接受参数, 参数的个数和类型不需要在定义的时候声明. 在函数体内调用函数即可.

```
funWithParam(){
    echo "第一个参数为 $1 !"
    echo "第二个参数为 $2 !"
    echo "第十个参数为 $10 !"
    echo "第十个参数为 ${10} !"
    echo "第十一个参数为 ${11} !"
    echo "参数总数有 $# 个!"
    echo "作为一个字符串输出所有参数 $* !"
}
```

Shell 的读取参数.

使用下面这条语句即可

```
read parameter_name
```

文件重定向

重定向命令列表如下:

命令	说明
command > file	将输出重定向到 file。
command < file	将输入重定向到 file。

5.2 Vim:Command Line 文本编译器

编辑模式

Vim 的设计以大多数时间都花在阅读、浏览和进行少量编辑改动为基础, 因此它具有多种操作模式:

- 正常模式: 在文件中四处移动光标进行修改
- 插入模式: 插入文本
- 替换模式: 替换文本
- 可视化 (一般, 行, 块) 模式: 选中文本块



- 命令模式: 用于执行命令

在不同的操作模式下，键盘敲击的含义也不同。比如，`x` 在插入模式会插入字母 `x`，但是在正常模式会删除当前光标所在的字母，在可视模式下则会删除选中文块。

在默认设置下，Vim 会在左下角显示当前的模式。Vim 启动时的默认模式是正常模式。通常你会把大部分时间花在正常模式和插入模式。

你可以按下 `<ESC>`（退出键）从任何其他模式返回正常模式。在正常模式，键入 `i` 进入插入模式，`R` 进入替换模式，`v` 进入可视（一般）模式，`V` 进入可视（行）模式，`<C-v>`（`Ctrl-V`, 有时也写作 `^V`）进入可视（块）模式，`:` 进入命令模式。

插入文本

在正常模式，键入 `i` 进入插入模式。现在 Vim 跟很多其他的编辑器一样，直到你键入 `<ESC>` 返回正常模式。你只需要掌握这一点和上面介绍的所有基础知识就可以使用 Vim 来编辑文件了（虽然如果你一直停留在插入模式内不一定高效）。

缓存，标签页，窗口

Vim 会维护一系列打开的文件，称为“缓存”。一个 Vim 会话包含一系列标签页，每个标签页包含一系列窗口（分隔面板）。每个窗口显示一个缓存。跟网页浏览器等其他你熟悉的程序不一样的是，缓存和窗口不是一一对应的关系；窗口只是视角。一个缓存可以在多个窗口打开，甚至在同一个标签页内的多个窗口打开。这个功能其实很好用，比如在查看同一个文件的不同部分的时候。

Vim 默认打开一个标签页，这个标签也包含一个窗口。

命令行

在正常模式下键入`:`进入命令行模式。在键入`:`后，你的光标会立即跳到屏幕下方的命令行。这个模式有很多功能，包括打开，保存，关闭文件，以及 **退出 Vim**。

- `:q` 退出（关闭窗口）
- `:w` 保存（写）
- `:wq` 保存然后退出
- `:e {文件名}` 打开要编辑的文件



- :ls 显示打开的缓存
- :help {标题} 打开帮助文档
 - :help :w 打开 :w 命令的帮助文档
 - :help w 打开 w 移动的帮助文档

移动

多数时候你会在正常模式下，使用移动命令在缓存中导航。在 Vim 里面移动也被称为“名词”，因为它们指向文字块。

- 基本移动: h j k l (左, 下, 上, 右)
- 词: w (下一个词), b (词初), e (词尾)
- 行: 0 (行初), ^ (第一个非空格字符), \$ (行尾)
- 屏幕: H (屏幕首行), M (屏幕中间), L (屏幕底部)
- 翻页: Ctrl-u (上翻), Ctrl-d (下翻)
- 文件: gg (文件头), G (文件尾)
- 行数: :{行数}<CR> 或者 {行数}G ({行数} 为行数)
- 杂项: % (找到配对，比如括号或者 /* */ 之类的注释对)
- 查找: f{字符}, t{字符}, F{字符}, T{字符}
 - 查找/到向前/向后在本行的 {字符}
 - , / ; 用于导航匹配
- 搜索: /{正则表达式}, n / N 用于导航匹配

编辑

所有你需要用鼠标做的事，你现在都可以用键盘：采用编辑命令和移动命令的组合来完成。这就是 Vim 的界面开始看起来像一个程序语言的时候。Vim 的编辑命令也被称为“动词”，因为动词可以施动于名词。

- i 进入插入模式
 - 但是对于操纵/编辑文本，不单想用退格键完成
- o / O 在之上/之下插入行
- d{移动命令} 删除 {移动命令}
 - 例如， dw 删除词, d\$ 删除到行尾, d0 删除到行头。
- c{移动命令} 改变 {移动命令}
 - 例如， cw 改变词
 - 比如 d{移动命令} 再 i



- **x** 删除字符 (等同于 `d1`)
- **s** 替换字符 (等同于 `xi`)
- 可视化模式 + 操作
 - 选中文字, **d** 删除或者 **c** 改变
- **u** 撤销, <C-r> 重做
- **y** 复制 / “yank” (其他一些命令比如 `d` 也会复制)
- **p** 粘贴
- 更多值得学习的: 比如 ~ 改变字符的大小写

计数

你可以用一个计数来结合“名词”和“动词”，这会执行指定操作若干次。

- `3w` 向前移动三个词
- `5j` 向下移动 5 行
- `7dw` 删除 7 个词

修饰语

你可以用修饰语改变“名词”的意义。修饰语有 `i`, 表示“内部”或者“在内”，和 `a`, 表示“周围”。

- `ci(` 改变当前括号内的内容
- `ci[` 改变当前方括号内的内容
- `da'` 删除一个单引号字符串，包括周围的单引号

5.3 Git: 版本控制工具

为什么说版本控制系统非常有用？即使您只是一个人进行编程工作，它也可以帮您创建项目的快照，记录每个改动的目的、基于多分支并行开发等等。和别人协作开发时，它更是一个无价之宝，您可以看到别人对代码进行的修改，同时解决由于并行开发引起的冲突。

基础

- `git help <command>`: 获取 git 命令的帮助信息
- `git init`: 创建一个新的 git 仓库，其数据会存放在一个名为 `.git` 的目录下
- `git status`: 显示当前的仓库状态



- `git add <filename>`: 添加文件到暂存区
- `git commit`: 创建一个新的提交
- `git log`: 显示历史日志
- `git log --all --graph --decorate`: 可视化历史记录 (有向无环图)
- `git diff <filename>`: 显示与暂存区文件的差异
- `git diff <revision> <filename>`: 显示某个文件两个版本之间的差异
- `git checkout <revision>`: 更新 HEAD 和目前的分支

分支和合并

- `git branch`: 显示分支
- `git branch <name>`: 创建分支
- `git checkout -b <name>`
: 创建分支并切换到该分支
 - 相当于 `git branch <name>; git checkout <name>`
- `git merge <revision>`: 合并到当前分支
- `git mergetool`: 使用工具来处理合并冲突
- `git rebase`: 将一系列补丁变基 (rebase) 为新的基线

远端操作

- `git remote`: 列出远端
- `git remote add <name> <url>`: 添加一个远端
- `git push <remote> <local branch>:<remote branch>`: 将对象传送至远端并更新远端引用
- `git branch --set-upstream-to=<remote>/<remote branch>`: 创建本地和远端分支的关联关系
- `git fetch`: 从远端获取对象/索引
- `git pull`: 相当于 `git fetch; git merge`
- `git clone`: 从远端下载仓库

撤销

- `git commit --amend`: 编辑提交的内容或信息
- `git reset HEAD <file>`: 恢复暂存的文件
- `git checkout -- <file>`: 丢弃修改



Git 高级操作

- `git config`: Git 是一个高度可定制的工具
- `git clone --depth=1`: 浅克隆 (shallow clone), 不包括完整的版本历史信息
- `git add -p`: 交互式暂存
- `git rebase -i`: 交互式变基
- `git blame`: 查看最后修改某行的人
- `git stash`: 暂时移除工作目录下的修改内容
- `git bisect`: 通过二分查找搜索历史记录
- `.gitignore`: 指定故意不追踪的文件

快照

Git 将顶级目录中的文件和文件夹作为集合，并通过一系列快照来管理其历史记录。在 Git 的术语里，文件被称作 Blob 对象（数据对象），也就是一组数据。目录则被称之为“树”，存储了这个目录下所有 Blob 和目录的信息。快照则是被追踪的最顶层的树，也就是最顶层的目录。换句话说，这个目录的所有文件的在某一个时间的状态就被称为一个快照。

例如，一个树看起来可能是这样的：

```
<root> (tree)
|
+- foo (tree)
|   |
|   + bar.txt (blob, contents = "hello world")
|
+- baz.txt (blob, contents = "git is wonderful")
```

Git 靠维护一系列的快照来追溯历史的版本，具体来说，快照是用有向无环图来进行快照的追踪的，这代表 Git 中的每个快照都有一系列的“父辈”，理解就是父快照中的文件经过了修改再进行提交就是子快照。在 Git 中，这些快照又可以称为被称为“commit”(提交)，每一次提交 git 并不会保存所有文件的拷贝，git 每次存储会存储每次提交改动的内容。所以说每个快照都要保存就是父辈的相关信息，快照的时间和提交信息等，还有当前目录下所有文件的状态。

o <-- o <-- o <-- o



```
^  
 \  
--- o <-- o
```

Git 中的提交是不可改变的。但这并不代表错误不能被修改，只不过这种“修改”实际上是创建了一个全新的提交记录。而引用则被更新为指向这些新的提交。

数据模型及其伪代码表示

以伪代码的形式来学习 Git 的数据模型，可能更加清晰：

```
// 文件就是一组数据  
type blob = array<byte>  
  
// 一个包含文件和目录的目录  
type tree = map<string, tree | blob>  
  
// 每个提交都包含一个父辈，元数据和一个树，这个树一般是代表git仓库目录的顶层  
type commit = struct {  
    parent: commit  
    author: string  
    message: string  
    snapshot: tree  
}
```

所以在这里面可以看到,git 中一个 commit 其实就是一个快照, 这个 commit 保存了当前所有目录所有文件的信息, 实际的 git 中维护的不是文件本身而是改动.

所以说 git 的数据模型你可以简单理解, 就是由很多个快照串联在一起的结构, 快照是由普通文件和目录文件组成的. 快照之间靠类似于链表之间的关系来维护的.

内存模型

Git 中的对象可以是 blob、树或提交：我们了解了 git 中抽象数据的结构, 现在我们了解这些抽象数据是怎么存储在存储系统中的, 总的来说 git 是靠一个索引来维护这些对象的.



```
type object = blob | tree | commit
```

Git 在储存数据时，所有的对象都会基于它们的 SHA1 哈希进行寻址。Blobs、树和提交都一样，它们都是对象。当它们引用其他对象时，它们并没有真正的在硬盘上保存这些对象，而是仅仅保存了它们的哈希值作为引用。

```
100644 blob 4448adb7ecd394f42ae135bbeed9676e894af85baz.txt  
040000 tree c68d233a33c5c06e0340e4c224f0afca87c8ce87foo
```

当然我们还可以自定义指针来引用哈希值, Git 可以使用诸如“master”这样人类可读的名称来表示历史记录中某个特定的提交，而不需要在使用一长串十六进制字符了。

git 的区域

Git 中还包括一个和数据模型完全不相关的概念，但它确是创建提交的接口的一部分。

就上面介绍的快照系统来说，您也许会期望它的实现里包括一个“创建快照”的命令，该命令能够基于当前工作目录的当前状态创建一个全新的快照。有些版本控制系统确实是这样工作的，但 Git 不是。我们希望简洁的快照，而且每次从当前状态创建快照可能效果并不理想。例如，考虑如下场景，您开发了两个独立的特性，然后您希望创建两个独立的提交，其中第一个提交仅包含第一个特性，而第二个提交仅包含第二个特性。或者，假设您在调试代码时添加了很多打印语句，然后您仅仅希望提交和修复 bug 相关的代码而丢弃所有的打印语句。

Git 处理这些场景的方法是使用一种叫做“暂存区（staging area）”的机制，它允许您指定下次快照中要包括那些改动。

首先我们要把 git 文件先交付到暂存区，然后在 commit 到 git 的仓库中。

总结

- git 是维护快照的软件，也就是维护目录下所有文件在某一个时间的状态.
- 在 git 中每一次 commit 都会保存一个快照.
- git 维护快照并不会把所有文件都拷贝一遍,git 维护主要是维护文件的改变信息.
- git 的所有提交、目录和普通文件都有一个 SHA1 索引表示, 当然我们还可以给 SHA1 索引起别名.



- git 可以有很多个分支, 每个分支都有一个名字, 这个名字相对应的其实是一个提交对象.
- git 的空间由用户的世界空间和 git 维护的暂存区和提交区的空间组成.

5.4 Makefile & cmake: 构建系统

对于大多数系统(或者是软件、亦或是程序)来说, 不论其是否包含代码, 都会包含一个“构建过程”。有时, 您需要执行一系列操作。通常, 这一过程包含了很多步骤, 很多分支。执行一些命令来生成图表, 然后执行另外的一些命令生成结果, 然后再执行其他的命令来生成最终的论文。有很多事情需要我们完成, 您并不是第一个因此感到苦恼的人, 幸运的是, 有很多工具可以帮助我们完成这些操作。这些工具通常被称为“构建系统”。

一般我们使用 make 作为我们的构建系统, 当我们执行 make 时, 它会去参考当前目录下名为 Makefile 的文件。所有构建目标、相关依赖和规则都需要在该文件中定义, 它看上去是这样的:

```
paper.pdf: paper.tex plot-data.png  
pdflatex paper.tex  
  
plot-%.png: %.dat plot.py  
./plot.py -i $*.dat -o $@
```

这个文件中的指令, 即如何使用右侧文件构建左侧文件的规则。或者, 换句话说, 冒号左侧的是构建目标, 冒号右侧的是构建它所需的依赖。缩进的部分是从依赖构建目标时需要用到的一段程序。在 make 中, 第一条指令还指明了构建的目的, 如果您使用不带参数的 make, 这便是我们最终的构建结果。或者, 您可以使用这样的命令来构建其他目标: make plot-data.png。

举个例子, 在上面的 MAKEFILE 中, 生成 paper.pdf 需要 paper.tex 和 plot-data.png, 生成这个文件需要执行 pdflatex paper.tex, 所以说 MAKEFILE 提供了一些信息, 我能生成什么文件, 生成这个文件需要什么, 怎么样生成这个文件.

还有一个更加方便的 Makefile 生成器, 名字叫做 Cmake, 可以非常方便地管理我们写的 Cpp 项目, 但由于比较复杂, 本书暂且不作介绍, 下面给一个学习地址大家可以阅读:

https://blog.csdn.net/kai_zone/article/details/82656964

还有一个简明教程, 可以在写代码的时候方便你查阅:

<https://zhuanlan.zhihu.com/p/492932151>



看完这两个文章再去实操一个项目, 你可以更好地掌握 Cmake 的用法.

5.5 tmux: 终端多路复用

终端多路复用器提供了一个帮助我们用一个 cmd 窗口执行两个任务的思路.

我们一般使用 tmux 来进行终端的多路复用

tmux 的快捷键需要我们掌握, 它们都是类似 <C-b> x 这样的组合, 即需要先按下 Ctrl+b, 松开后再按下 x。tmux 中对象的继承结构如下:

- 会话-每个会话都是一个独立的工作区, 其中包含一个或多个窗口, 每一个会话包括多个窗口.
 - tmux 开始一个新的会话
 - tmux new -s NAME 以指定名称开始一个新的会话
 - tmux ls 列出当前所有会话
 - 在 tmux 中输入 <C-b> d, 将当前会话分离
 - tmux a 重新连接最后一个会话。您也可以通过 -t 来指定具体的会话
- 窗口-相当于编辑器或是浏览器中的标签页, 从视觉上将一个会话分割为多个部分
 - <C-b> c 创建一个新的窗口, 使用 <C-d> 关闭
 - <C-b> N 跳转到第 N 个窗口, 注意每个窗口都是有编号的
 - <C-b> p 切换到前一个窗口
 - <C-b> n 切换到下一个窗口
 - <C-b> , 重命名当前窗口
 - <C-b> w 列出当前所有窗口
- 面板-像 vim 中的分屏一样, 面板使我们可以在一个屏幕里显示多个 shell
 - <C-b> " 水平分割
 - <C-b> % 垂直分割
 - <C-b> < 方向 > 切换到指定方向的面板, < 方向 > 指的是键盘上的方向键
 - <C-b> z 切换当前面板的缩放
 - <C-b> [开始往回滚动屏幕。您可以按下空格键来开始选择, 回车键复制选中的部分
 - <C-b> < 空格 > 在不同的面板排布间切换

会话包含若干个窗口-> 窗口包含若干个面板. 这样我们可以灵活地处理 Command Line 环境了.



5.6 apt&homebrew: 依赖下载器

apt 和 homebrew 是很好的软件依赖下载器, 使用的方法非常简单. 在 Linux/Ubuntu 的环境下, 我们可以使用下面这条语句来进行安装.

```
sudo apt install xxx
```

在 MacOS 的环境下, 我们可以使用 brew 来进行安装.

```
brew install xxx
```

Ubuntu apt 常用命令

- 列出所有可更新的软件清单命令: **sudo apt update**
- 升级软件包: **sudo apt upgrade**
列出可更新的软件包及版本信息: **apt list --upgradeable**
升级软件包, 升级前先删除需要更新软件包: **sudo apt full-upgrade**
- 安装指定的软件命令: **sudo apt install <package_name>**
安装多个软件包: **sudo apt install <package1> <package2> <package_3>**
- 更新指定的软件命令: **sudo apt update <package_name>**
- 显示软件包具体信息, 例如: 版本号, 安装大小, 依赖关系等等: **sudo apt show <package_name>**
- 删除软件包命令: **sudo apt remove <package_name>**
- 清理不再使用的依赖和库文件: **sudo apt autoremove**
- 移除软件包及配置文件: **sudo apt purge <package_name>**
- 查找软件包命令: **sudo apt search**
- 列出所有已安装的包: **apt list --installed**
- 列出所有已安装的包的版本信息: **apt list --all-versions**

5.7 回顾与展望

生产力工具

- **Markdown 基本语法**: 你必然会用到的简易排版语言
- **Typora**: 一款好用的 Markdown 编辑器
- **Notion**: 一款结合文档、知识库以及任务管理功能的全能协作工具 (自己用来做笔记极好, 支持 Markdown 排版)

基本素养学习



- 认识 Linux

- 命令行学习

- Git 学习

- Vim 学习

- man 学习

更多技能学习

- Vim 配置

- 受不了 Vim 基础配置过于简易的功能，快来让你的 Vim 更适合自己。

- A GDB Tutorial with Examples

- 你早晚要尝试脱离 IDE 进行编程，快来看看如何在纯命令行模式下进行代码的调试。这是一份非常简短的入门教程，如果你想深入学习，可以尝试搜索一下“GDB 完全中文手册”或者直接看更多的英文文档。

- Makefile 基本使用

- Makefile 是让你脱离 IDE 的项目构建工具，来瞧瞧它的基本使用方法。

- 正则表达式 30 分钟入门教程

- 正则表达式在程序设计中是强有力的工具，它可以帮助你轻易地描述不同的字符串模式，还不快来试试？

- Tmux 使用教程

- 如何在一个命令行窗口中开启多个子窗口？如何让程序不受限于窗口一直运行下去？快来试试 Tmux。

进阶学习

- LaTeX 入门

- LaTeX 是论文排版语言，你目前可以只学习其中数学公式排版的部分，帮助你在电脑中排出好看的数学公式。链接是一本很好的中文快速入门书。

- Linux 命令行与 shell 脚本编程大全

- 极丰富的 Linux 命令行与脚本学习，可作手册供查阅。

- Linux C 编程一站式学习

- 学习完 C 语言了，也许你还不知道它能用来做什么。这份教程可以帮助你在 C 基本语法的基础上向高阶的系统编程进军，欢迎进入 Linux 的世界！

- 跟我一起写 Makefile

- 更完全的 Makefile 使用方法。

- Docker 入门教程



- 目前最流行的 Linux 容器解决方案。

基本素养试题

Linux 与命令行

1. Linux 下 `root` 用户主目录的绝对路径是?
2. 一般来说, `zilize` 用户主目录的绝对路径是?
3. Linux 系统中存放进程信息的目录是?
4. 用什么命令来查看当前所在路径?
5. 当前目录下有个目录 `hello world`, 如何切换到该目录中?
6. 当前的路径为 `~/zilize/hi`, 想切换到 `~/zilize/hello`, 需要使用的单条命令是?
7. 列出当前路径下的文件和目录(包括隐藏项)的命令是?
8. 递归地列出当前路径下所有文件和目录的详细信息的命令是?
9. 创建目录 `hello` 的命令是?
10. 创建新文件 `main.c` 的命令是?
11. 有个配置文件叫做 `hustport.com.conf`, 想查看其内容需要的命令是?
12. (接上题) 如果文件太大, 想一点点地看, 用什么命令好呢?
13. (接上题) 看着看着不想看了, 按什么键退出呢?
14. 有个目录 `hustport.com` 存放了网站的所有内容, 用什么命令删除它比较好?
15. 有个复杂的目录 `hustport.com`, 你想复制一份到当前路径中, 新目录名字叫 `hustport2.com`, 用什么命令呢?
16. 想把目录 `hello` 重命名为 `hi`, 需要用什么命令?
17. 你想下载压缩包 `https://hustport.com/file.zip`, 用什么命令呢?
18. 根据下面的权限描述符 `drwxr-xr-x`, 系统中的其他用户能否进行写操作?
19. (接上题) 用三位八进制数描述上述权限为?
20. 将目录 `hello` 的权限改为 `drwxr--r--` 的命令是?
21. 实时监测系统进程需要的命令是?
22. 杀掉进程号为 23333 的进程的命令是?
23. 看看下面的 PATH 环境变量, 猜猜 Apache 的安装路径是什么?
`/usr/local/php/bin:/usr/local/apache/bin:/usr/local/mysql/bin:/usr/lo`
24. 你想用 `python` 运行 `run.py`, 为避免关闭 SSH 连接后进程中断, 你需要使用的命令是?
25. 下面命令的功能是?

```
scp -P5102 zilize@233.233.233.23:/home/zilize/main.c ~/zilize/main.c
```



Git 使用方法

1. 你修改了文件 `main.c` 后，通过什么命令将其加入暂存区？
2. 将暂存区修改提交到分支并备注"FIX: `main.`" 的命令为？
3. 你刚编辑保存了 `main.c` 就发现不应该修改，用什么命令撤销修改呢？
4. 创建开发分支 `dev` 并切换到该分支的命令是？
5. 删除开发分支 `dev` 的命令是？
6. 回退到 ID 为 `e475afc93c209a690c39c13a46716e8fa000c366` 的提交的命令为？
7. 希望将当前的 `test` 分支推送到远程主机 `origin` 的 `dev` 分支的命令是？
8. 想要将分支 `dev` 合并到当前分支中，并不进行自动提交，采用的命令是？
9. 假设 GitHub 有一个开源项目叫做 `HUSTPORT`，你发现了其中的 BUG，你想对其加以修复，并向该开源项目贡献自己的代码，用自己的话说说该完成哪些步骤？

Vim 使用方法

1. Vim 共有哪三种模式？请用英文作答。
2. 向后翻页（行号值由小变大）的快捷键是？
3. 快速定位到 1037 行的命令是？
4. 在命令模式下保存并强制退出的命令是？
5. 命令 `cc` 和 `dd` 的区别是？
6. 向下跳转 100 行的命令是？
7. 在命令模式下表示撤销的是那个按键？
8. 跳到最后一行的快捷键是？
9. 命令模式下，想在文档中定位到关键字 `return` 处的命令是？
10. (接上题) 点击什么按键定位到下一个匹配处？

Shell 脚本

1. 尝试写一个脚本 `asimpleshell.sh`, 通过调用 `asimpleshell.sh` 计算出若干输入的加法.
2. 尝试写一个脚本 `asimpleshell.sh`, 通过调用 `asimpleshell.sh`. 脚本的参数是一个文件路径, 你需要把这个文件目录的元素输出出来.

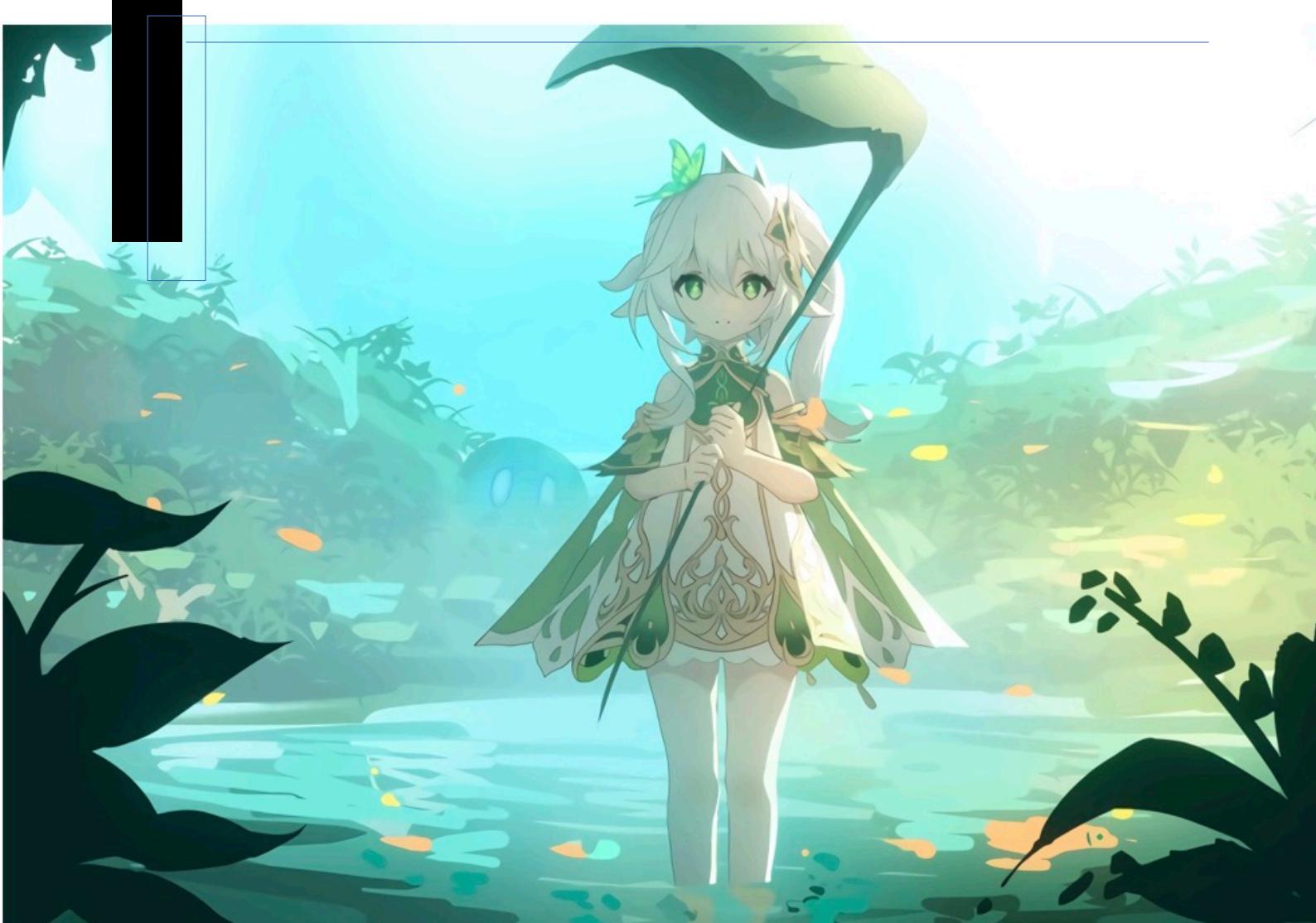
正则表达式

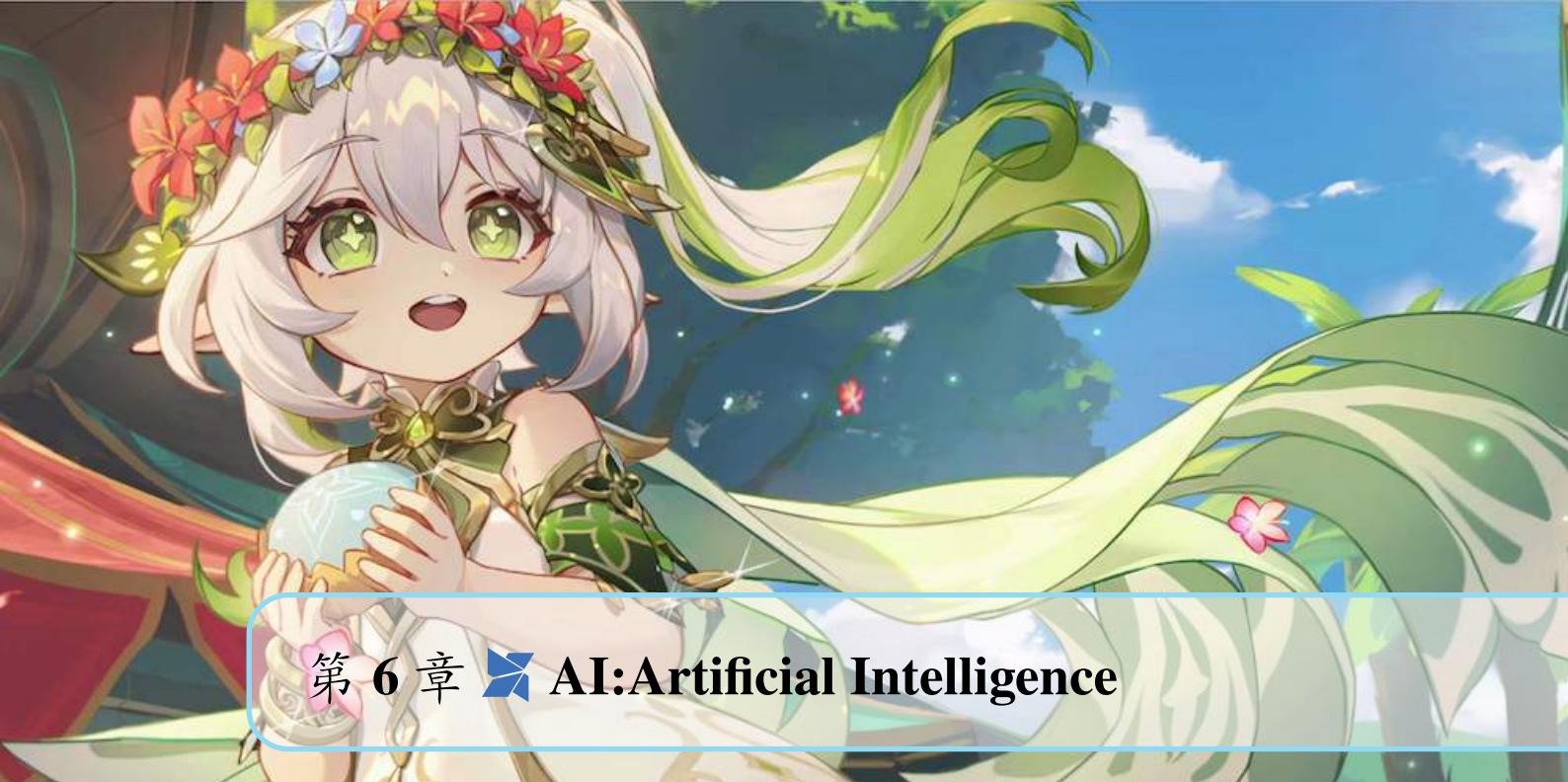
1. 写一个正则表达式, 表达一个 C 语言的标识符.
2. 写一个 awk 命令, 给定 n 行指令, 统计出 cd 指令的数量.

原文地址:<http://hustport.com/d/228>



第三部分 * 计算机研究领域





第 6 章 ➤ AI:Artificial Intelligence

6.1 基础知识

6.1.1 数学

数学是人工智能的基石，这里主要聚焦与人工智能高度相关的数学分支以供参考。

- 微积分：每一位理工科学生的大学必修课，在微积分中，需要注重基础概念的学习，因为它们会是深度学习中很多模型和方法的理论基础，我举两个例子：大家如果学习神经网络，一定避不开反向传播算法的推导，这里就需要应用函数求导的链式法则；随机梯度下降（Stochastic Gradient Descent）等优化算法，需要你对梯度的概念有深刻的认识。
- 线性代数：线性代数亦是人工智能数学里的重中之重，如果你深入学习机器学习和深度学习，你会发现，这个领域随处可见的向量空间、矩阵运算以及各种矩阵特征在不同任务中的独特用处。
- 概率统计：一想到概率统计，再想到人工智能，我的第一反应是三个字，“贝叶斯”，贝叶斯定理是一个经典而永恒的公式，它告诉我们在评价现象与其原因的过程中，不仅要考虑这个原因的可能性即先验概率，还要考虑该原因产生该现象 + 的概率，也就是似然概率，此外学习概率统计知识中的参数估计以及假设验证也大有裨益。
- 最优化方法：机器学习和深度学习参与的任务往往都有严格的效果评价指

标，单单从任务本身来讲，一个模型在评价指标中获得的分数越高，可以认为它更加优秀，你可以简单理解为，这个指标向更好的方向转变的过程，就是最优化的过程。而机器学习和深度学习的问题大多可以建模成一种最优化模型的求解过程。

书籍我只推荐一个《Mathematics for Machine Learning》，这本书并没有讲很高深的数学，但是吃透了这本书，可以说做 AI 的数学底子是有了。

6.1.2 机器学习

首先我想讨论一下我眼中的机器学习，机器学习绝对不是简简单单调个库，使用 Python 完成一个回归或者分类任务就够了，这些过程仅仅只能说明你进行了机器学习的应用，并不能说明你是在真正的学习机器学习。大家都说机器学习和深度学习是黑箱，我认为，了解黑箱的原理、探究黑箱的优化路径、制作出更优秀的黑箱，不失为 deep learning 的一个美妙的学习方式。有点扯远了，这里我们先谈机器学习。自己总结不如借用一句精辟的话，机器学习是“使用算法解析数据，从中学习，然后对世界上的某件事情做出决定或预测”，第一句话就框定了我们机器学习的核心：“数据”，可能你已经听说过“数据和特征决定机器学习的上限，模型和算法只是在逼近这个上限”，没错，数据的清洗和特征工程是机器学习的一个重要且不可或缺的步骤，试想，一堆杂乱的数据，没有经过特征加工，再厉害的模型和算法，或许也难逃出过拟合的魔爪。

6.1.2.1 数据清洗，特征工程

因此，你可能需要先了解数据清洗和特征工程的流程；

内容：了解数据清洗的常见场景，理解特征工程的意义，掌握特征工程的常用方法，并付诸实践。

推荐学习方式：

了解：推荐博客 <https://www.mygreatlearning.com/blog/feature-extraction-in-image-processing/>

学习：学习机器学习一定不要错过 Kaggle，上面有大量各种场景的机器学习/深度学习应用任务，你可以先查看高分选手的代码，掌握数据处理等 Pretreatment 的方法，然后考虑在自己的任务中应该怎么做并总结经验。此外，并不推荐大家阅读《Python 数据分析基础》等书籍，最好的学习方式是上手和总结。由于这是引导性的文章，所以具体方法就不做赘述，下文亦然。



6.1.2.2 模型选择

第二部分需要分两个部分来讲，一个是认识并学习各种模型，一个是如何选择模型；

Part1: 机器学习模型学习：推荐周志华老师的《机器学习》，江湖人称“西瓜书”，配合《机器学习理论导引》，了解模型原理，动笔完成推导；

实践：同样，Kaggle，阿里天池，此外，推荐参加数学建模竞赛，数学建模竞赛可以用上很多机器学习方法以及优化算法，此外，更重要的是数模竞赛中会有各种背景的问题，可以锻炼针对不同处境下问题的解决能力，既能增长自己的技能树，还能拿奖状，何乐不为。

Part2: 如何选择模型：推荐两个博客，一个是<https://www.projectpro.io/article/common-machine-learning-algorithms-for-beginners/202>，这篇博客分析了主流模型，并且给出了适用背景，适合初学了解；

一个是<https://towardsdatascience.com/considerations-when-choosing-a-machine-learning-model-aa31f52c27f3>，这篇博客从多种角度介绍了模型需要考虑的要点。

此外，最重要的还是多写代码，根据问题类型、模型特点以及实际表现来进行选择。

6.1.2.3 参数调整

如何在选择好的模型上达到更佳的效果，这大概就是调参的意义，调参其实并不好讲，因为要说常用方法，也并不多，深入一点，需要你对模型的参数有非常深刻的理解，知道某参数的变动大概会影响模型的哪些表现，我觉得调参达到最优效果只有两个因素，经验和时间。

常见的最佳参数搜索方法可以参考该链接：<https://programs.wiki/wiki/summary-of-four-parameter-adjustment-methods-in-machine-learning.html>

给出了代码示例，可以针对自己的任务进行实践。

6.2 深度学习

深度学习的目的是为了能够学习数据的更多本来的规律，最终能让机器拥有类似于人的思考和解决问题的能力。而深度学习里最具有表征意义的就是神经网络，倘若没有神经网络的相关学习经验，你可以这么考虑神经网络的功能并理解它的流程，我们将神经网络简化为一个分类器或者回归器，对于输入数据，经过神经网络内部的传播通路，得到一个输出，在前馈神经网络中，各神经元分层排列，接受前一层的输入，通过数学运算得到输出结果并传递给下一层，最



终得到输出结果，以分类任务为例，他可能得到一个表示归属不同类别的概率向量 \vec{l} ，然而在训练集上这个结果会与真实结果产生差距，不同任务不同背景下会用不同的方式来表征这个“差距”，称为损失函数，然后计算这个损失函数相对于神经网络中各种参数的梯度来进行反向传播，找到使这个损失函数降低的参数修改方法，从而逐步降低损失。

有关深度学习的学习资料众多，这里推荐几个比较好的社区：

<https://towardsdatascience.com/>

<https://machinelearningmastery.com/>

<https://medium.com/>

问答网站：

<https://stackoverflow.com/>

<https://jp.quora.com/>

其他学习资料分方向推荐，课程我私心是推荐 Stanford University AI 课程全套的，我自己的使用方法是观看课程，完成纸质作业和 Colab 编程作业，如果对某方向非常感兴趣了，就从该领域的经典论文开始看，如果论文有开源代码，就配合代码进行学习，然后从难度和重要程度等方面精心选择 1~2 篇进行复现，如果你打算进一步挖掘该领域的奥秘，你可以联系本校或者保研意向学校相关方向的老师，请老师让你加入相关课题组，一般学长学姐会让你先看相关小方向的文献，如何能在课题组中贡献自己的代码并总结一些跑实验的经验就更好了，最佳的结果是你跟着课题组有了 Paper 产出，即使没有 Paper 产出或者因为事务繁忙放弃了，这也能够成为你的科研经历，写入你的简历。

下面对较为火热和我比较了解的方向进行简要介绍和资料分享。

6.2.1 计算机视觉

计算机视觉是研究让机器如何“看”的科学，它的目的是实现实代替人眼进行目标的检测、识别等的机器视觉。主要有模式识别，图像分类，图像分割，图像增强，图像生成，风格迁移（风格化）等方向。

推荐课程：Stanford University CS231n, CAP5415 – Computer Vision

经典论文收录：[Classic computer vision papers](#)

CV 方向相关国际学术会议和期刊：

会议：CVPR, ICCV, ECCV, AAAI, ICML, NeurIPS, ICPR

期刊：IJCV, TPAMI, TIP, PR



6.2.2 自然语言处理

自然语言处理是融合计算机、数学和语言学于一身的科学，它训练机器具有人一样处理和表达语言的能力，主要包括机器翻译、文本分类、阅读理解、问答系统、舆情监测、语音识别、信息提取和因果推断等方向。

推荐课程：Stanford University CS224n

推荐书籍：自动化所宗成庆老师的《统计自然语言处理》

经典论文收录：<https://github.com/mhagiwara/100-nlp-papers>

NLP 方向相关国际学术会议和期刊：

会议：ACL, EMNLP, NACAL, AAAI, ICLR, COLING,

期刊：Computational Linguistics (CL), TACL

6.2.3 图机器学习

图学习是近几年一个比较热门的研究方向，图机器学习以由结点和连接边构成的网络作为数据，通过学习图结构数据来挖掘图上特征和内在模式，在图上，你可以执行 node、edge 和 graph 级别的分类任务，或者是 missing edge 的预测任务等等，进一步地，图机器学习在知识图谱、推荐系统和社交网络等领域都有广泛的应用。

推荐课程：Stanford University CS224W

经典论文收录：<https://github.com/thunlp/GNNPapers>

会议：ICML, NeurIPS, ECCV, CVPR, ICLR, IJCAI, AAAI

如果你学习图机器学习，并且有了 CS224W 的理论基础，那么可以访问 Stanford 大学的 **Open Graph Benchmark** 网站，自己动手用 GCN、GraphSAGE、GAT 等模型实现一些图上任务，算是一个非常不错的入门 GNN 的方式。

6.2.4 对抗样本

相较于前面介绍的领域，对抗样本（Adversarial Example）是贯穿深度学习各种领域的研究方向，对抗样本的思想在于为数据集添加轻微的扰动，从而使得表现优秀的模型输出产生较大的误差，从而挖掘模型尚未学习到的特征，进而通过对抗训练来增进模型的鲁棒性。

对抗样本生成任务自 Goodfellow 在视觉领域中提出以来就受到了很大的关注，它使得人们对于深度学习模型的安全性产生了诸多思考，同时通过对模型的攻击也让人们对于模型的理解更加深入。而在视觉领域中，由于图像数据的



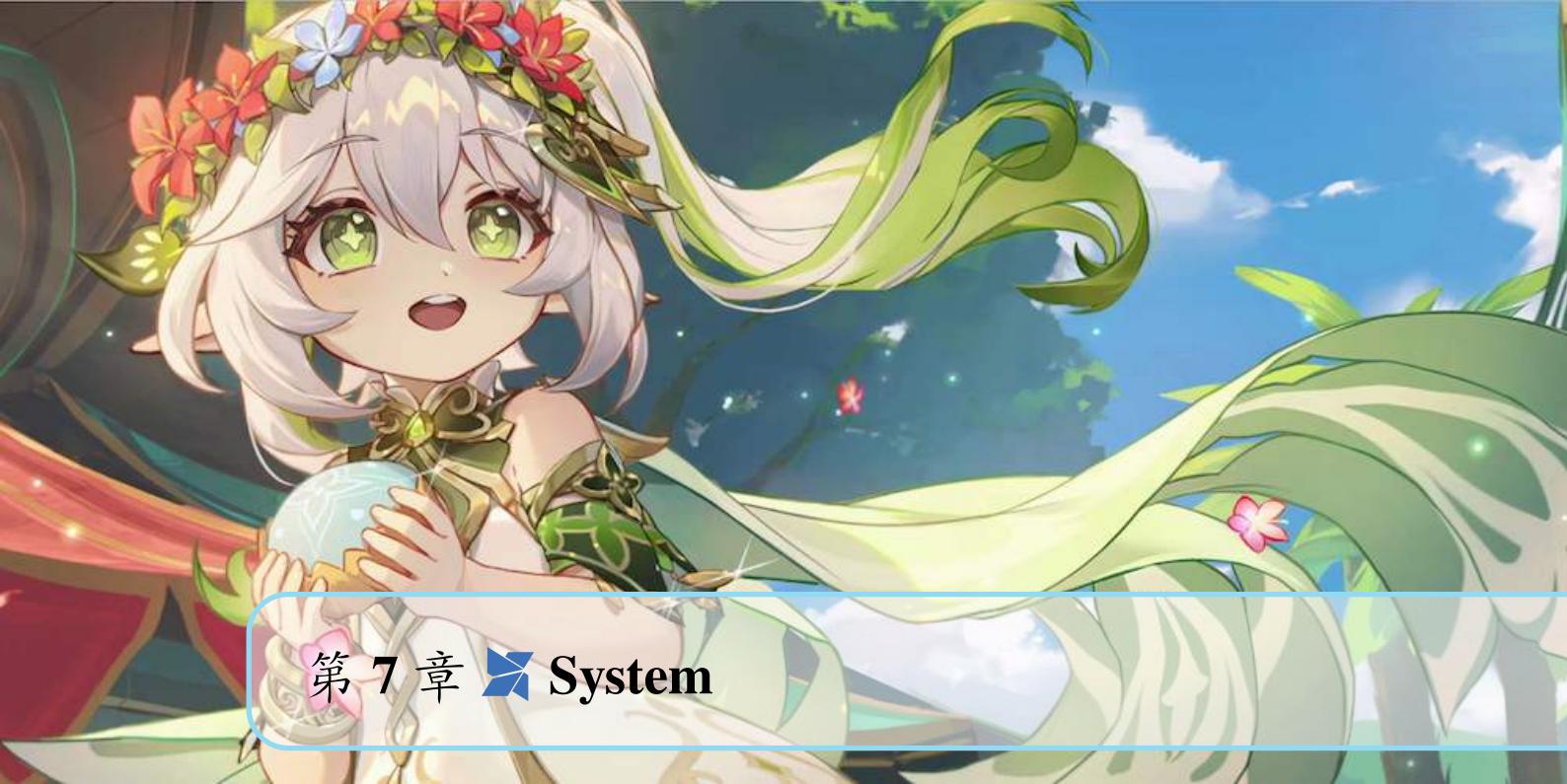
连续性特点，使得其扰动大小能够被精确度量，因此视觉领域的对抗样本生成技术发展较快，反观文本领域，由于文本数据的离散化特性以及语言本身的高度复杂性，使得扰动大小难以衡量，且文本模型本身鲁棒性要高于视觉相关模型，这些都增加了文本对抗本身的难度。

目前，对抗攻击的方式根据掌握信息的程度分为黑盒攻击和白盒攻击，黑盒攻击方法对于模型的参数一无所知，仅仅依靠修改输入并得到反馈输出来进行侦测，而白盒攻击则主要依赖于所期望指向的错误输出对于模型内部参数的梯度来进行攻击，是了解模型内部结构的攻击方式，对于对抗攻击，清华大学thunlp组总结了各个领域的对抗攻击的有代表性的论文，可以针对自己的情况挑选阅读。链接如下：

[thunlp/NLP-THU: NLP Course Material & QA \(github.com\)](#)

由于本人知识广度和深度限制，对于不曾了解和学习的领域不敢擅自定义和指教，如果真的对人工智能感兴趣，那么在按这个路线学习的过程中，你一定会自发地发现自己所感兴趣的领域，并投入其中，收获颇丰。





第 7 章 System

下面我们来介绍 System 这一个计算机领域。作为计算机科班出身的我们，我们更需要了解的就是计算机底层的系统构造。当我们对整个计算机系统有着深刻的理解，我们的看待计算机的视野就更加宽广。

7.1 学习大纲

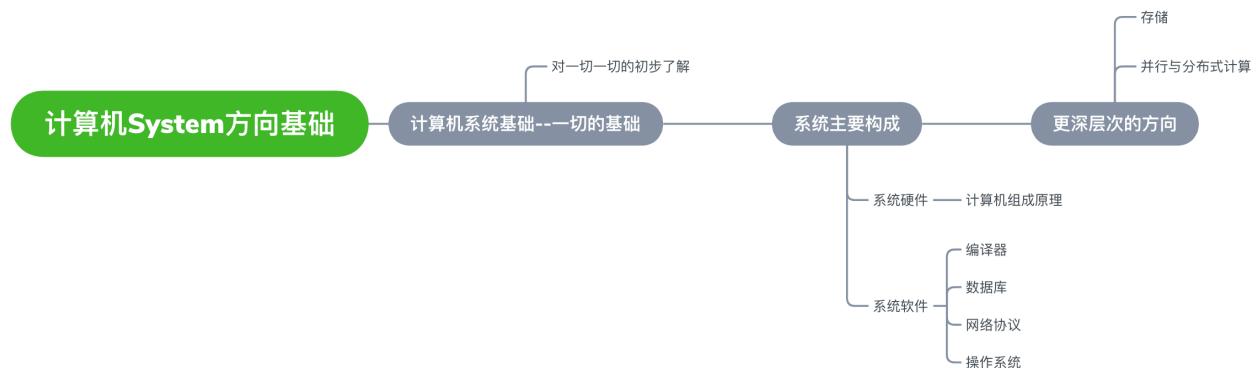


图 7.1: System 领域的框架

由于 System 领域的知识很多都和科班课程内容有很大的联系，所以说我在这里给出大概的学习路径和学习方法，由于本人也仅仅是刚刚入门，所以说只能提供一部分入门的思路。

7.2 理论学习方法

计算机系统基础

汇编语言

目的: 了解系统的基础就是汇编语言, 因为了解系统就是了解更加底层的东西, 在面向更加底层的东西我们就需要更加底层的语言来去实现.

学习内容: 基本的汇编语句, 我们不需要了解太多, 只需要了解汇编语句的作用就好了.

推荐学习时间: 大二时间段.

校内课程: 大二下会开设汇编语言、但我觉得偏晚.

推荐书籍: 曹爽《汇编语言》

推荐网课: 无

计算机系统基础

目的: 这一部分的学习会提纲挈领地介绍计算机系统的结构, 对系统结构的每一个部分都有一个简单的介绍. 通过这个部分的学习, 你可以大概了解计算机由哪些层次组成, 每个层次都有哪些东西组成, 每个层次都有什么作用. 这帮助你大概了解系统的成分及其作用, 对于之后的学习有打基础的帮助.

学习内容: 覆盖了汇编语言、体系结构、操作系统、编译链接、并行、网络等基础知识, 这些知识比较难, 所以说需要花很大的力气去了解.

推荐学习时间: 大一下、大二上时间段.

校内课程: 大二下会开设计算机系统基础、但我觉得偏晚.

推荐书籍: CSAPP:<Computer System:a programmer's perspective>

《程序员的自我修养》

推荐网课: CMU 15-213: CSAPP

有精力: UCB CS61C: Great Ideas in Computer Architecture(上手造 CPU, 用汇编写神经网络, 非常牛)

补充: 函数式编程

学习并法度高的函数式编程也有必要

推荐学习时间: 大而上、大二下时间段.

校内课程: 大二上会开设函数式编程的选修课.



推荐书籍: ML for the Working Programmer, PAULSON, LAWRENCE C. (Univ. of Cambridge, Cambridge, UK)

推荐网课: CMU 15-150 Functional Programming

系统主要构成

下面用数字 I、II... 来代表学习的顺序. 如果有两个数字是一样的, 就代表可以并发学习.

I、计算机组成原理

目的: 这一部分的学习会直接学习冯诺伊曼结构的各个硬件的功能和设计思路, 这是直接学习硬件本身, 了解硬件特性更加有助于你了解贴近硬件的软件.

学习内容: 了解冯诺伊曼体系结构的控制器、存储器、运算器、I/O 器件的硬件设计思路, 体会了软件和硬件协同工作的思路. 推荐同学们自己用 logism 或者 Verilog 来自己搭建控制器、存储器、运算器和 I/O 器件. 并用适当的数据通路连接在一起.

推荐学习时间: 大二下时间段

校内课程: 大二下开设计算机组成原理

推荐书籍: 谭志虎: 《计算机组成原理》

推荐网课: 华中科技大学 MOOC: 《计算机组成原理》以及《自己动手造 CPU》

I、计算机网络

目的: 这一部分学习计算机网络是如何工作的. 网络协议是计算机重要的系统软件, 这帮助我们了解计算机如何与外界沟通

学习内容: 了解网络的成员和协议栈, 简单了解应用层、运输层、网络层、链路层的工作原理和工作任务. 推荐同学们自己用 C++ 实现一套 TCP 协议, 或者用 cisco 的 Packet Tracer 的软件自己搭建一个网络.

推荐学习时间: 大二下时间段

校内课程: 大三上开设计算机网络课程

推荐书籍: 《计算机网络: 自顶而下方法》

推荐网课: Stanford CS144: Computer Network(这门课程直接教你手把手写 TCP/IP 协议, 然后把 Linux 内核中的 TCP/IP 协议给替换了, 这个真的很帅, 实验也很循序渐进.)



I、数据库系统原理

目的: 这一部分学习计算机的一个系统软件数据库, 在这里面我们更好地了解计算机存储相关的知识.

学习内容: 学习关系型数据库的各种操作、设计思路以及事务处理功能. 也为后面学习大数据相关的非结构的数据库打下基础.

推荐学习时间: 大二下甚至是大二上时间段.

校内课程: 大三下开设数据库课程

推荐书籍: 数据库系统全书《Database System: The Complete Book》

推荐网课: CMU 15-445: Database Systems 和 UCB CS186: Introduction to Database System.

II、操作系统

目的: 这一部分学习计算机顶层应用和底层硬件的中间商, 如果您能了解操作系统的知识, 你可以更加深入地了解计算机是如何对硬件资源进行分配的, 又是如何驱使硬件正常工作的.

学习内容: 了解操作系统的基本功能、了解操作系统的进程调度策略、文件管理策略、设备驱动策略、内存分配策略等.

推荐学习时间: 大三上时间段

校内课程: 大三上开设操作系统课程(但是我们学校的操作系统讲的比较拉)

推荐书籍: Operating Systems: Principles and Practice (2nd Edition) & 《现代操作系统》.

推荐网课: MIT 6.S081: Operating System Engineering.(推荐配合华中科技大学 PKE 的操作系统实验使用, 因为个人感觉 xv6 的实验比较偏应用态, 而不是内核态. 如果能配合 UCB CS162: Operating System 的 Pintos 操作系统实验可以更好)

II、编译原理

目的: 编译器是一个贴近于应用程序和硬件的软件, 通过编译器的实现, 你会了解我们的应用程序怎么变成机器代码, 并且能够理解编译器如何优化我们的代码

学习内容: 理论部分覆盖了词法分析、语法分析、语义分析、运行时环境、寄存器分配、代码优化与生成等内容.

推荐学习时间: 大三上时间段



校内课程: 大三下开设编译原理课

推荐书籍: 龙书 (不是龙叔啊喂)

推荐网课: Stanford CS143: Compilers.(为 COOL 语言设计一个支持 MIPS 指令集的编译器, 的确很 COOL)

更深层次的话题...

接下来你应该在主要构成中选择一个方向进行更加深层次的研究. 但是你也可以对下面的课题进行一些初步的了解. 当然前提是第二部分的知识您已经全部搞定了

安全

目的: 研究系统的构造是一个思路, 但是用户不仅需要计算机系统够快, 还对计算机系统的安全性提出了更高的要求. 所以说研究系统的安全更加重要

校内课程: 无

推荐网课: MIT 6.858: Computer System Security.(最后实现一个远端的文件系统, 面对一个完全不可信的系统, 提供基本的安全和完整性要求).

分布式系统

目的: 研究集群十分重要, 因为众人拾柴火焰高, 研究多个计算机一块工作也是当下的热点话题

校内课程: 大数据相关课程

推荐网课: MIT 6.824: Distributed System(和 6.S081 是一块的 PDOS 实验室, 由于分布式系统非常复杂, 你需要花费大量的精力处理随机性带来的同步问题, 祝好运).

并行编程

目的: 由于现在指令级的并行山穷水尽, 对于线程级的并行研究十分广泛, 多核硬件的普及对并行编程提供了土壤. 研究程序的并行处理也很重要.

校内课程: 大三下: 并行编程与实验

推荐网课: CMU 15-418/Stanford CS149: Parallel Computing.



7.3 实践学习方法

研究系统需要动手去实现很多系统的功能或者软件,下面给出了一些实践的学习思路,同学们可以根据自己的兴趣来进行学习.(其实我推荐的网课也配套了实践课程了)

cache 的设计设计一个 cache 很有意思,这里的 cache 不仅仅指硬件的 cache,你可以设计一个软件 cache,比如说微信小程序的 cache 的实现等等,当然最基础的就是存储系统的 cache 元件,当你设计完之后你会对存储系统的分级管理更加了解

CPU 的设计设计一个简单的 CPU,从最简单的开始做起吧,假设你的 CPU 只支持 4 条指令,尝试设计输出信号控制器和时序元件,然后把 CPU 和其他部件连在一起.你会发现设计一个简单的 CPU 并不难.

流水线你可以把 CPU 或者浮点数处理分成几个部分来做,这个时候就可以构成流水线,每个部分执行每个部分的功能.

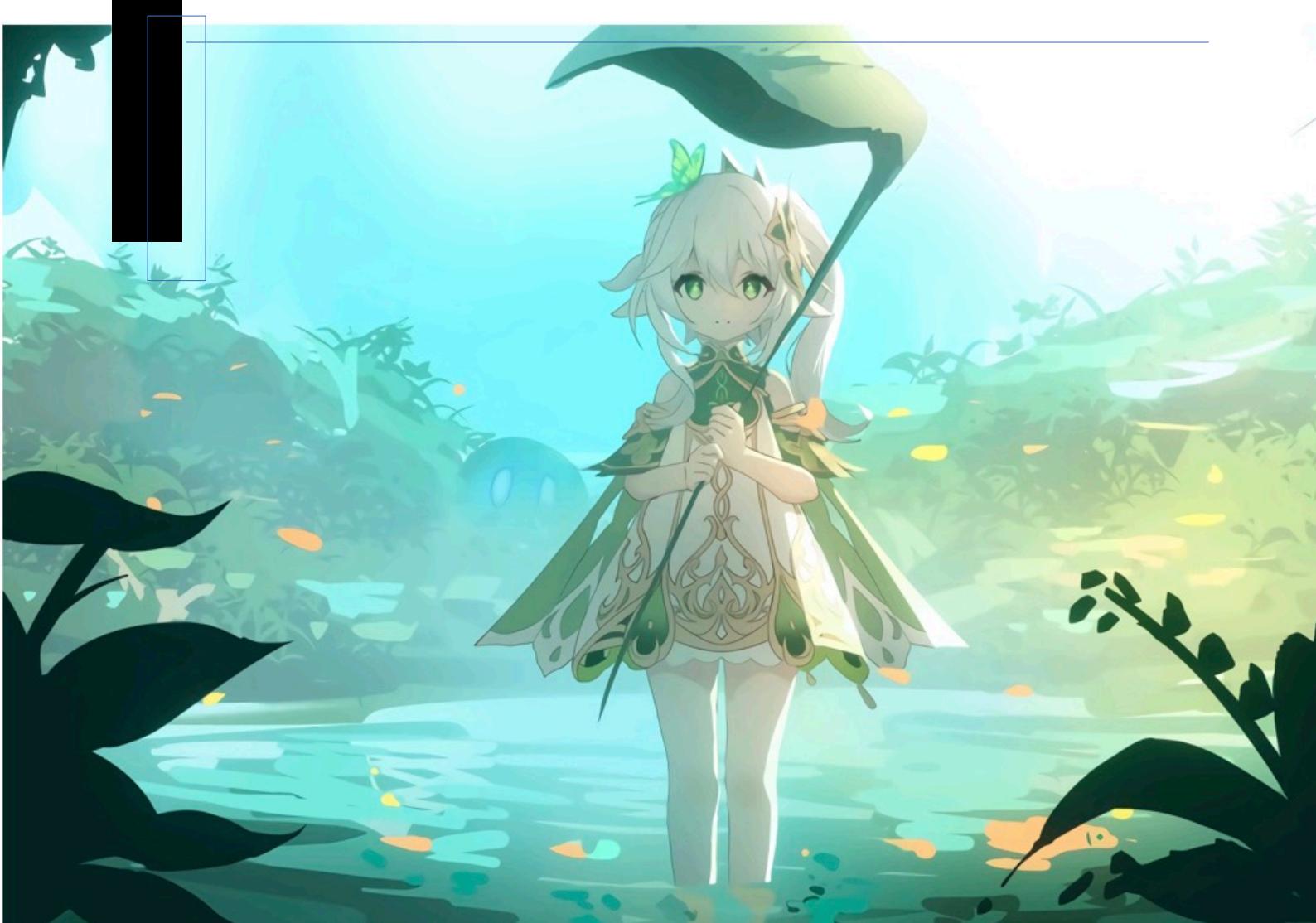
操作系统设计设计一个操作系统太难了,不如从魔改一个操作系统开始,找到一个简单的、人畜无害的操作系统(比如说 PKE)、然后魔改,实现自己希望的功能.

嗯造 TCP实现一个 TCP 协议吧,对于网络层和应用层的数据可以先不管,我们就把它当成一个管道吧.

设计编译器我们可以从一个非常简单的语言做起,完成词法分析和句法分析.



第四部分 * 附录





第 A 章 ✨ 问题回答

A.1 课程相关

Q: 学电路理论这样的专业课程是不是有点和时代需求脱节了?

A: 电路理论是工科基础课程, 学科交叉也很重要的, 你无法预测将来会进入什么领域, 并且很多学校有这个课程.

(Answered by 生产队的大萝卜)

Q: 学校是不是过于重视理论课程了, 缺少实践?

A: 我院课程实践比例其实很大, 只是学分比例没那么大而已.

(Answered by 生产队的大萝卜)

Q: 为什么大一课那么多?

A: 新课改把很多课往前压了, 目的是让同学们在大三有更多的时间做事.

Q: 英语有必要学吗?

A: 第一, 看论文需要看英文的论文, 第二, 到了研究生写论文肯定也需要写英文的论文. 第三, 就算就业了, 想进入程序员的圈子, 比如说 github 和 stackoverflow 也需要英文交流. 第四, 看技术文档是有 80% 可能性遇到英文的文档的.

Q: 电路理论等电路课能否设为专选课? 因为喜欢“软”的学生学起来吃力不

讨好，反而浪费了大量时间精力，不能探索自己擅长喜欢的领域。如果只是为了“培养科学思维”，似乎大可不必。

A: 谈判中。虽然浪费时间，但是总会有那么一捏捏用。

A.2 科研、保研相关

Q: 保外的流程是怎样的？

A: 可以参考这个帖子：[click](#), 非常详细。

一般来说保研的时间线是这样子的：

3 4 月：套瓷，获得导师的认可。

5 7 月：各高校发布暑期夏令营的相关通知，夏令营开始报名。

7 月：夏令营开始。

8 月：各高校发布九月预推免的通知。

9 月：各高校开始九月预推免的考核。

9 月末：填写教育部推免系统，恭喜成功上岸。

保外的必要的三个条件：

需要获取到本校本科生院的推免资格，就是大萝卜给的综合排名达到保研的线。

获得目的学校学院的认可，比如说你要保到清华大学，就需要获得清华大学计算机系的认可，这个认可就是你在七月份的夏令营或者九月份的预推免考核中通过考核。

获得目的学校的一个导师的认可，这个每个导师有每个导师的考核方法，但确保你可以获得这个导师的认可。

一般考核的形式：

机试：上机写代码。

笔试：开卷或者闭卷考试：南大，考试内容涵盖大学所有的专业课，包括组成原理、网络、操作系统、面向对象的程序设计、Linux 工作流等。

面试：什么都会问，唠家常，问专业课问题，让你介绍项目，看一篇英文论文发表感想，都会有。甚至可能会让你用英文回答问题，需要综合素养的准备！

Q: 怎么样刷 GPA？

A: 第一点，就是保持平常心，佛说“可遇不可求”，如果你保持着很浓的对加权的渴望的话，往往是不能得到的，对于分数，由于是一次考试决定六成到八成，



随机性比较大,有什么分数都会有可能,保持平常心是第一点.

第二点就是做好每一件事,天才不多有,有人随随便便考高分,在你不确定自己能不能随随便便考高分的时候,就上好每一堂课,做好每一次作业,温习好每一个知识点就好了.这么做完,你的分数不会低.

更重要的是怎么看待 GPA,GPA 仅仅是陪伴你大学四年的一个符号,仅仅是 4 年而已,无论你是高分还是低分,最重要的是培养自己的能力,而不是在各种 GPA 课程和算加分的竞赛中迷失.GPA 高自然是好事,这个时候就要了解更多的知识.GPA 低未必是一件坏事,因为研究生这种东西,不一定适合你,本科尽快地准备就业也是一个很棒的出路,如果你觉得 GPA 太低焦虑,不如找一点事情或者找一点东西去学,做一点小项目不是也很棒么.(Answered by Sukuna)

Q: 科研是怎样的?

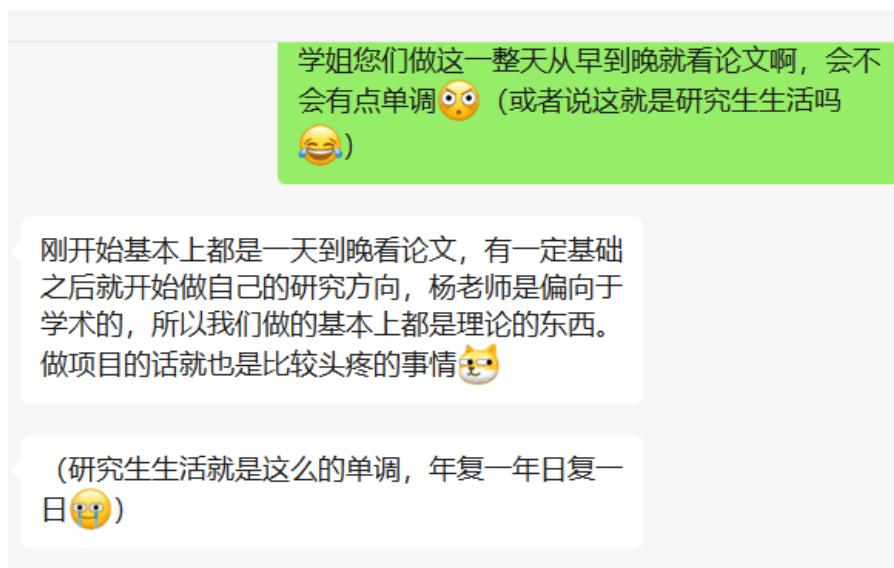


图 A.1: 学姐的回答

Q: 加权能够到保研线,有必要再打水赛拉高加分吗?

A: 有时间就打呗,不水的比赛像我这种普通学生又拿不了奖,就我来看大三还是挺空的,打水赛能拿分的话其实名次提升比卷甲权要来的高效,打的信息差嘛!

Q:(跟着上一问) 如果已经确定保研了,还需要打水赛么?

A: 如果是提升自己方面那肯定做项目好呀,水赛其实就是无意义的内卷.
(Answer by Henry)

Q: 怎么样寻找适合自己的科研方向?



A: 多试错.

Q: 19 级保研情况?

A: 普通班: 一共 11 个班大约 330 人, 保研到 98 名, 其中 98 名强制保送到核聚变研究中心.

计卓: 30 个人保 11 个.

物联网: 30 个人保 5 个.

Q: 考研对计算机行业重要么?

A: 研究生仅仅是个学历, 最主要的还是能力. 一些比较卷的岗位还是看研究生学历的.

Q: 新入学该如何面对各式各样的比赛呢, 想学很多东西但是太多了, 全是零基础真的不知道怎么安排而且不知道怎么下手. 况且听说后面的课会越来越多现在的课业已经感觉到有点压力了.

A: 多看看这本书, 找到自己感兴趣的东西, 抽空看看教程多学学就好.

Q: 对于高中没有基础的, 怎样高效达一定水准? 包括参与科研, 参加比赛, 保研等?

A: 多找点赛道卷, 多收集信息, 信息差是竞争的关键, 一定能找到适合自己的赛道. 反正多尝试新的事物, 不能只学课内的东西, 多向大牛请教学习方法或者多看看相关的经验帖子.

Q: 保研去哪些大学分别要什么水平?

A: 这个只能说看你夏令营和九推的发挥, 但是我可以大概跟你说说进入夏令营和九推的条件. 就是清北大概需要前 15, 复旦交大可能前 20 到 25, 浙大可能前 30. 南大中科大可能前 40 就可以. 但是这只是进入夏令营而已, 具体的表现还要看你自己的发挥了.

Q: 大一新生争取保研更好还是争取校招更好或者两者一起努力?

A: 我只能跟你说, 看你自己, 我无法告诉你究竟哪个好.

Q: 请问一下要做好哪些准备才能保研呢?

A: 1、必要大前提, 就是综排靠前, 获得了保送研究生的资格, 这是最基本最基



本的.

2、GPA 不能低,GPA 是进入夏令营的必要条件,如果你的加权成绩不高,你很难拿到好学校的资格.但是进入夏令营,加权就和你无关了!

3、四六级,部分学校会卡四六级成绩报,保研时如果你不是特别强,请注意你的六级不要低于 500 分!

4、科研与项目,找自己喜欢的去搞.

5、准备好每个学校的夏令营和九推考试,考试内容请看上面.

6、两篇副教授及以上级别的推荐信.

Q: 加权 86、87 还有希望吗?

A: 希望比较少,趁早换赛道最好,早准备有优势.

Q: 什么推荐的比赛?

A: 谁知道你擅长什么呢?我建议是什么比赛先别怕,都去试试.

Q: 请问一下,大学期间计算机的科研项目是自己一个人做还是要找渠道进组,而且一直不懂计算机的科研项目有些什么东西,就是不知道要研究什么?

A: 对的,找认识的教授进组,一般来说本科生任务不会特别大,具体来说就是教授会给你一个课题,你去阅读该课题相关的论文并把这个课题解决掉,如果你比较牛,解决解决着有属于自己的心得,就可以写论文了.但是科研过程中,包括调研,包括做实验,包括写论文,需要你投入非常大量的时间,这个看你自己.

A.3 就业相关

Q: 想本科就业,该怎么准备?

A1: 想找工作就学点技术,刷题,背背面试项目,打好基础,根据想要的工作岗位类型做项目.

(Answer by XiaoNanBei,50w offer)

A2: 准备好简历,企业会看中你的比赛和项目经历,多打比赛多搞项目,刷力扣,最好刷到能稳过 hard 的程度.企业的笔试题大多都是从力扣里面出.寒暑假可以找实习,在公司实习可以避免走弯路,毕竟最接近生产.

(Answer by RHR. 华为深圳)

Q: 项目该怎么准备呢?



A: 你看看你想做什么岗位, 一般是前端和后端, 前端就是做网页, 后端就是做一切相关的后台, 可以做系统架构师, 数据库, 网络安全相关, 还有单纯写码, 具体要看岗位要求, 你可以让他直接上大厂的招聘网站, 各个岗位要求的技能都写的很清楚. 除了前端和后端以外还可以搞测试, 测试门槛低一点, 还有客户端开发, 客户端开发是指手机端 app 开发, 有时候也会要你搞小程序开发. 还有比较偏门的就是搞算法, 一般而言本科生卷不过研究生博士生, 算法一般是指机器学习算法而不是 acm 那种算法.

(Answer by RHR. 华为深圳)

Q: 就业岗位的相关资讯和信息有哪些种类? 一般怎么获得?

A: 直接关注个大厂的招聘网站和公众号, 然后可以联系学长学姐, 有时候学长学姐组内缺人, 你联系的上的话可以直接内推实习生, 免去一些流程.

(Answer by RHR. 华为深圳)

Q: 面试会问什么?

A: 一般而言如果你简历够丰富, 会问你项目经历, 如果不丰富就会问八股文, 一般看面试官态度, 有些面试官就很鄙视八股, 就不会问你, 有的很重视八股, 认为这是基础.

(Answer by RHR. 华为深圳)

Q: 找工作有什么温馨提示吗??

A: 一定都有在面试官面前打代码的环节, 笔试是第一步, 最后成绩是看综合排名, 但是如果你在面试过程中有一步得分低也会直接刷, 多联系 hr, 多催催进度, 最后就是不要作弊, 会上黑名单的, 有时候面试会要你讲讲之前笔试的思路, 心理素质要强, 被面试官怼到哑口无言是很正常的.

(Answer by RHR. 华为深圳)





第 B 章 基本算法的简要介绍

B.1 排序

排序的概念

定义 B.1

排序的稳定性: 存在两个具有相同关键字的记录 $R(i)$ 与 $R(j)$, 其中 $R(i)$ 位于 $R(j)$ 之前。在用某种排序法排序之后, $R(i)$ 仍位于 $R(j)$ 之前, 则称这种排序方法是稳定的; 否则, 称这种排序方法是不稳定的。



下面介绍一下几种排序.

直接插入排序

直接插入算法、比较关键字的次数、移动记录次数, 稳定性

设待排序的文件为: $(r[1], r[2], \dots, r[n])$

关键字为: $(r[1].key, r[2].key, \dots, r[n].key)$

首先, 将初始文件中的记录 $r[1]$ 看作有序子文件;

第 1 遍: 将 $r[2]$ 插入有序子文件中, 若: $r[2].key < r[1].key$, 则 $r[2]$ 插在 $r[1]$ 之前; 否则, 插在 $r[1]$ 的后面。

第 2 遍: 将记录 $r[3]$ 插入前面已有 2 个记录的有序子文件中, 得到 3 个记录

的有序子文件。

以此类推，依次插入 $r[4], \dots, r[n]$ ，最后得到 n 个记录的递增有序文件。

直接插入排序

```
void InsertSort(RecType r[], int n)
{
    // 对数组 r[1..n] 中的 n 个记录作插入排序

    { int i,j;

        for (i=2; i<=n; i++) {
            r[0]=r[i];           //待插记录 r[i] 存入监视哨中
            j=i-1;               //以排序的范围 1 - i-1
            //从 r[i-1] 开始向左扫描

            while(r[0].key<r[j].key)
            { r[j+1]=r[j];       //记录后移
                j--;             //继续向左扫描
            }

            r[j+1]=r[0];         //插入记录 r[0]，即原 r[i]
        }
    }
}
```

选择排序

简单排序算法：一个一个选出最小的，插进去

简单排序

```
void SelectSort(RecType r[], int n)
{
    int i,j,min;
    RecType x;           //交换记录的中间变量
```



```

for (i=1; i<n; i++)           //共 n-1 趟 (遍)

{ min=i;                      //r[i] 为最小记录 r[min]

  for (j=i+1; j<=n; j++)

    if (r[j].key<r[min].key)

      min=j;                  //修改 min

    if (min!=i)              //若 r[min] 不是 r[i]

    { x=r[min];               //交换 r[min] 和 r[i]

      r[min]=r[i];

      r[i]=x;

    }

  }

}

```

当然,还可以用堆(一种用数组模拟的完全二叉树)来进行排序,我们称之为堆排序.下面给出堆排序的代码.

堆排序

```

void HeapAdjust(HeapType &H, int s, int m)

{ rc=H.r[s];                  //保存需调整的数据元素,空出 s 的记录位置

  for(j=2*s; j<=m; j*=2) { //j<=m 表示 s 有左孩子序号 j=2*s

    if (j<m&&H.r[j].key<H.r[j+1].key) //j<m 表示 s 有右孩子 j+1

      j++;                     //计算 s 的具有较大关键字的孩子的序号 j

    if (rc.key> H.r[j].key)//rc 在 s 的结点时满足结点定义, 调整完毕

      break;

    H.r[s]=H.r[j]; s=j;        //s 的较大孩子上移, 修改 s 下移
}

```



```
    } //正常结束时， s 的结点无孩子结点。  
    H.r[s]=rc;  
}
```

调整方法：找孩子中最大的，交换，一直到比孩子大或者越界
怎么初始化一个堆？从 $n/2$ 开始调用堆调整的代码

2-路归并排序

本质上是一种“分而治之”的手法，首先把这一部分分成两半，对于左边一半和右边一半都进行一次排序，然后把左边排序的结果和右边排序的结果归并在一起即可。

2-路归并排序

```
void merge(r,y,low,mid,high)  
RecType r[],y[]; int low,mid,high;  
{ int k=i=low,j=mid+1;  
  while (i<=mid && j<=high)  
  { if (r[i].key<=r[j].key)  
    { y[k]=r[i]; //归并前一个子文件的记录  
      i++; }  
  else  
    { y[k]=r[j]; //归并后一个子文件的记录  
      j++; }  
  k++; }  
  while (j<=high) //归并后一个子文件余下的记录
```



```

{ y[k]=r[j];
  j++; k++;
}

while (i<=mid) //归并前一个子文件余下的记录
{
  y[k]=r[i];
  i++; k++;
}

} // merge

```

但是两个两个归并, 剩下的元素数量不是整数倍怎么办?

正常情况下, 剩下的元素的数量大于 $2s$, 那就归并

如果大于 s 小于 $2s$, 剩下一个正常的和残疾人归并
小于 s , 之间往下 copy 即可.

//一趟归并

```

void mergepass(RecType r[], RecType y[], int s)
{
  int i=1;
  while(i+2*s-1<=n) //两两归并长度均为 s 的子文件
  {
    merge(r,y,i,i+s-1,i+2*s-1);
    i=i+2*s;
  }
  if (i+s-1<n) //最后两个子长度为 s 和长度不足 s 的文件
    merge(r,y,i,i+s-1,n);
  else
    while(i<=n) //复制最后一个子文件, 长度 < s
    {
      y[i]=r[i];
      i++;
    }
}

```

//对文件 $r[1..n]$ 归并排序的算法 (调用算法 $mergepass$)

```
void mergesort(RecType r[], int n)
```



```

{
    RecType y[n+1];
    int s=1;           //子文件初始长度为 1
    while (s<n)
    {
        mergepass(r,y,s); //将 r[1..n] 归并到 y[1..n]
        s=2*s;           //修改子文件长度
        mergepass(y,r,s); //将 y[1..n] 归并到 r[1..n]
        s=2*s;           //修改子文件长度
    }
}

```

交换排序

冒泡排序是我们第一个学到的基本的交换排序.

冒泡排序

```

void bubble1(int a[],int n)

{ int i,j,temp;

    for(i=0; i<n-1; i++)      //作 n-1 趟排序

        for(j=0; j<n-1-i; j++)

            if (a[j]>a[j+1])

            { temp=a[j];           //交换记录

                a[j]=a[j+1];

                a[j+1]=temp;

            }

    for(i=0; i<n; i++)

        printf("%d",a[i]); //输出排序后的元素

}

```

//可以加一个 swap 来判断是不是完全有序了



比冒泡排序更高级的还有快速排序：

先选择第一个元素放到最前，保存到临时变量 x，然后往前遍历，比 x 低的放到前面，比 x 大的放到后面，通过移动 i 和 j 指针进行处理

快速排序

```
void quksort(RecType r[], int low, int high)
{
    RecType x; int i, j;

    if (low < high)                                //有两个以上记录
    {
        i = low; j = high; x = r[i];                //保存记录到变量 x 中
        do {
            while (i < j && r[j].key >= x.key)      //此时 i 指示位置可用
                j--;
            if (i < j)                                //i, j 未相遇
            {
                r[i] = r[j]; i++;                      //此时 j 指示位置可用
                while (i < j && r[i].key <= x.key)
                    i++;
            if (i < j)
            {
                r[j] = r[i]; j--;
            }
        }
    } while (i != j);                            //i, j 未相遇
}

//划分结束，i 经过的是 key 不大于 x.key 的元素；
//j 经过的是 key 不小于 x.key 的元素。
//i, j 至少有一个指示的位置可用
r[i] = x;
```



```
    quksort(r, low, i-1);           //递归处理左子文件  
    quksort(r, i+1, high);         //递归处理右子文件  
}  
}
```

对文件 $r[1..n]$ 快速排序：

```
void quicksort(RecType r[], int n)  
{  
    quksort(r, 1, n);  
}
```

基数排序

(1) 首先根据最高有效位进行排序，然后根据次高有效位进行排序，直到根据最低有效位进行排序，产生一个有序序列。

(2) 首先根据最低有效位进行排序，然后根据次低有效位进行排序，直到根据最高有效位进行排序，产生一个有序序列。

对于 288,371,260,531,287,235,56,299,18,23:

第一趟排序后，结果为： 260,371,531,23,235,56,287,288,18,299

第二趟排序后，结果为： 18,23,531,235,56,260,371,287,288,299

第三趟排序后，结果为： 18,23,56,235,260,287,288,299,371,531

两趟排序之间不改变元素的之间的相对位置，上一轮排序落后的，下一轮排序数值一样还是在后面

B.2 查找



查找表的定义

定义 B.2

关键字: 可以标识一个记录的数据项

主关键字: 可以唯一地标识一个记录的数据项

次关键字: 可以识别若干记录的数据项

查找----根据给定的某个关键字值, 在查找表中确定一个其关键字等于给定值的记录或数据元素。

设 k 为给定的一个关键字值, $R[1..n]$ 为 n 个记录的表, 若存在 $R[i].key = k, 1 \leq i \leq n$, 称查找成功; 否则称查找失败。

静态查找: 查询某个特定的元素, 检查某个特定的数据元素的属性, 不插入新元素或删除元素(记录)。

动态查找: 在查找过程中, 同时插入查找表中不存在的数据元素(记录)。

平均查找长度----查找一个记录时比较关键字次数的平均值。



二分查找

二分查找: 就是在左边看看有没有, 在右边看看有没有(注意: 查找需要保持有序才可以进行查找)

二分查找

```

int binsrch(SSTable ST, keytype k)
{
    int low, mid, hig;
    low=1; hig=ST.length;
    while (low<=high)
    {
        mid=(low+high)/2;
        if (k<ST.elem [mid].key) hig=mid-1; //查左子表
    }
}
```



```

    else if (k==ST.elem [mid].key)

        return mid;           //查找成功, 返回 mid

    else low=mid+1;         //查右子表

}

return 0 ;                //查找失败, 返回 0

}

```

动态查找表

如果二叉树的任一结点大于其非空左子树的所有结点，而小于其非空右子树的所有结点，则这棵二叉树称为二叉排序树。对一棵二叉排序树进行中序遍历，所得的结点序列一定是递增有序的。

生成二叉树: 做 n 次二叉树排序

存储结构

二叉搜索树

```

struct node

{ struct

{ int key ;                  //关键字

....                         //其它数据项

} data ;

struct node *lchild,*rchild ; //左右子树的指针

} *root,*t;

```

插入: 大的话就到右边, 小的话到左边, 遇到空就插入

查找: 找到了就算成功, 找不到就返回 NULL, 跟插入一个样子

删除: 删除叶结点, 只需将其双亲结点指向它的指针置空, 再释放它即可。被删结点缺左子树 (或右子树), 可以用被删节点的右子树 (或左子树) 顶替它的位置, 再释放它。



被删结点左、右子树都存在，可以在它的右子树中寻找中序下的第一个结点（关键值最小，或者说是最左边的元素），用它的值填补到被删结点中，再来处理这个结点的删除问题。（往往这个结点没有左孩子，就可以降级到第二个问题了）

大概查找的次数：

平均情况： $O(\log_2 n)$

最坏情况： $\frac{n+1}{2}$

哈希表

Hash 函数实现的是将一组关键字映象到一个有限的地址区间上，理想的情况是不同的关键字得到不同的地址。

设 K_1 和 K_2 为关键字，若 $K_1 \neq K_2, H(K_1) = H(K_2)$ ，则称 K_1, K_2 为同义词， K_2 与 K_1 发生了冲突

构造哈希函数的方法

1. 直接定址法

2. 除留余数法：设哈希表 $HT[0..m-1]$ 的表长为 m ，哈希地址为 key 除以 p 所得的余数：

3. 平方取中法----取关键字平方后的中间某几位为哈希地址，即： $H(k) = \text{取 } k^2 \text{ 的中间某几位数字}$

4. 折叠法

将关键字分割成位数相同的几部分，然后取这几部分的叠加和作为哈希地址。

5. 数字分析法

设哈希表中可能出现的关键字都是事先知道的，则可取关键字的若干分布均匀的位组成哈希地址。

解决冲突的方法

1. 开放地址法（开式寻址法）

线性探测再散列：这个不行就找下一个地址，直到找到了元素的地址，轮了一圈没找着那就不行，报错

二次探测再散列：假定记录 R_i 和 R_j 的关键字 K_i 和 K_j 为同义词，散列地址为 q ， R_i 已存入 $HT[0..m-1]$ 中的 $HT[q]$ 中。若依次在地址 $q + 1^2, q - 1^2, q + 2^2, q - 2^2, \dots, q + i^2, q - i^2, \dots$ 中寻找一个空位，叫做二次探测再散列。

2. 链地址法

将关键字为同义词的所有记录存入同一链表中。（表尾插入）

3. 建立公共溢出区。



B.3 动态规划

最优化问题：这一类问题的可行解可能有很多个. 每个解都有一个值, 我们希望寻找有最优值的解(最小值或最大值).

注：这里, 我们称这个解为问题的一个最优解 (an optimal solution), 而不是 the optimal solution, 因为最优解也可能有很多个. 这种找最优解的问题通常称为最优化问题.

对于最优化问题的求解有很多思路, 根据描述约束条件和目标函数的数学模型的特性和问题的求解方法的不同, 可分为: 线性规划、整数规划、非线性规划、动态规划等问题. 而研究解决这些问题的科学一般就总称之为最优化理论和方法.

动态规划算法的步骤:

1. 刻画一个最优解的结构特征;
2. 递归地定义最优解的值;
3. 计算最优解的值;
4. 利用计算出的信息, 构造一个最优解.

注：1) 前三步是动态规划算法求解问题的基础. 如果仅需要一个最优解的值, 而非解本身, 可以忽略步骤 4.

2) 如果确实要做步骤 4, 则有时需要在执行步骤 3 的过程中维护一些额外的信息, 以便用来构造一个最优解.

- 3) 第三步通常采用自底向上的方法计算最优解;

下面用一个例子来描述一下动态规划.

Serling 公司购买长钢条, 将其切割为短钢条出售. 不同的切割方案, 收益是不同的, 怎么切割才能有最大的收益呢? 注意: 假设, 切割工序本身没有成本支出. 出售一段长度为 i 英寸的钢条的价格为 p_i ($i = 1, 2, \dots$), 下面是一个价格表 P.

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

图 B.1: 长度为 i 英寸的钢条可以为公司带来 p_i 美元的收益

那么问题就可以这么描述给定一段长度为 n 英寸的钢条和一个价格表 P, 求切割钢条方案, 使得销售收益 r_n 最大.

其实我们可以知道切割一个钢条的收益等于切割钢条的两部分的收益之和, 所以我们知道我们需要找到一个最佳的切割点 i 来获得结果. 使得两部分的和 $r_i + r_{n-i}$ 最大, 这种性质就是最优子结构性质.

最优子结构指的是, 问题的最优解包含子问题的最优解. 反过来说就是, 我们



可以通过子问题的最优解,推导出问题的最优解.换句话说,原问题分成若干个子问题,对于若干个子问题的最优解结合在一起就是原问题的最优解.

$$\max_i \quad r_i + r_{n-i}$$

即,首次切割后,将两段钢条看成两个独立的钢条切割问题实例.若分别获得两段钢条的最优切割收益 r_i 和 r_{n-i} ,则原问题的解就可以通过组合这两个相关子问题的最优子解获得.

那么问题就转化成了切割切在哪里的问题,就是遍历所有可能的切割点,计算出结果,选择值最大的那个就可以了.

将子问题按规模排序:最小子问题、较小子问题、…、较大子问题、原问题.按由小到大的顺序顺次求解:当求解某个子问题时,它所依赖的更小子问题都已求解完毕,结果已经保存,故可以直接引用并组合出它自身的解.

对于这个问题,处理方法就是先计算 r_0 ,再计算 r_1 ,再计算 r_2 ,一次类推,直到计算出 r_n ,那么 r_n 怎么求出来的呢?就是寻找一个最优的切割,将这个钢条切成两份,使得这两份的价值最高即可,也就是寻找到一个 i (切割点),使得 $r_i + r_{n-i}$ 最大.:

$$r_n = \max_i \quad r_i + r_{n-i} (0 < i \leq n)$$

EXTENDED-BOTTOM-UP-CUT-ROD(p, n)

```

1  let  $r[0..n]$  and  $s[0..n]$  be new arrays
2   $r[0] = 0$ 
3  for  $j = 1$  to  $n$ 
4       $q = -\infty$ 
5      for  $i = 1$  to  $j$ 
6          if  $q < p[i] + r[j-i]$ 
7               $q = p[i] + r[j-i]$ 
8               $s[j] = i$ 
9       $r[j] = q$ 
10 return  $r$  and  $s$ 
```

图 B.2: 拓展的切割钢条动态规划的伪代码

那么怎样决定子问题的处理顺序呢?这个时候我们可以使用子问题图来进行处理.

子问题图:当思考一个动态规划问题时,应该弄清楚所涉及的子问题与子问题之间的依赖关系,可用子问题图描述,子问题图用于描述子问题与子问题之间的依赖关系.

子问题图是一个有向图,每个顶点唯一地对应一个子问题.



若求子问题 x 的最优解时需要直接用到子问题 y 的最优解, 则在子问题图中就会有一条从子问题 x 的顶点到子问题 y 的顶点的有向边.

自底向上的动态规划方法处理子问题图中顶点的顺序为: 对一个给定的子问题 x , 在求解它之前先求解邻接至它的子问题. 即, 对于任何子问题, 仅当它依赖的所有子问题都求解完成了, 才会求解它. 这告诉我们, 要先解决出度为 0 的子问题.

CUT-ROD 算法给出了最优收益, 但怎么切割的、切割点在哪里呢? 通过扩展上述动态规划算法, 在求出最优收益之后, 即可求出切割方案. 具体做法就是切割钢条之后设置一个数组 s . 数组 s 用于记录对规模为 j 的钢条切割出的第一段钢条的长度 $s[j]$.

B.4 贪心算法

贪心算法是这样一种方法: 分步骤实施, 它在每一步仅作出当时看起来最佳的选择, 即局部最优的选择, 并寄希望这样的选择最终能导致全局最优解.

贪心求解的一般步骤:

- 1) 确定问题的最优子结构;
- 2) 将最优化问题转化为这样的形式: 每次对其作出选择后, 只剩下一个子问题需要求解;
- 3) 证明作出贪心选择后, 剩余的子问题满足: 其最优子解与前面的贪心选择组合即可得到原问题的最优解 (具有最优子结构).

贪心选择性质和最优子结构性是两个关键要素.

如果能够证明问题具有这两个性质, 则基本上就可以实施贪心策略.

一般就是证明: 问题可以转化成一次选择 + 一个子问题 (而不是之前的一次选择 + 两个子问题, 所以说这个选择都是选择靠近边缘的)

定义 B.3

贪心选择性质: 可以通过做出局部最优 (贪心) 选择来构造全局最优解的性质.

贪心选择性使得我们进行选择时, 只需做出当前看起来最优的选择, 而不用考虑子问题的解.



假定有一个活动的集合 S 含有 n 个活动 $\langle a_1, a_2, \dots, a_n \rangle$, 每个活动 a_i 都有一个开始时间 s_i 和结束时间 f_i . 同时, 这些活动都要使用同一资源 (如演讲会场), 而



这个资源在任何时刻只能供一个活动使用.(在结束时间 f_i 的时候活动不占用会场, 只在 $[s_i, f_i)$ 的时候占用).

不失一般性, 设活动已经按照结束时间单调递增排序:

最优子结构: 令 S_{ij} 表示在 a_i 结束之后开始且在 a_j 开始之前结束的那些活动的集合. 问题和子问题的形式定义如下: 设 A_{ij} 是 S_{ij} 的一个最大兼容活动集, 并设 A_{ij} 包含活动 a_k , 则有: A_{ik} 表示 A_{ij} 中 a_k 开始之前的活动子集, A_{kj} 表示 A_{ij} 中 a_k 结束之后的活动子集.

并得到两个子问题: 寻找 S_{ik} 的最大兼容活动集合和寻找 S_{kj} 的最大兼容活动集合.

活动选择问题具有最优子结构性, 即: 必有: A_{ik} 是 S_{ik} 一个最大兼容活动子集, A_{kj} 是 S_{kj} 一个最大兼容活动子集.

$$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$$

总的来说就是给定一组活动的最优解, 最优解里面选择一个活动, 求出活动前和活动后的那个区间的最优解.

问题的本质上就是对一个区间求最优解, 最优子结构就是找到一个中点, 然后根据中点前后划分出两个区间, 求出这两个小区间的最优解.

活动选择问题的贪心选择: 每次总选择具有最早结束时间的兼容活动加入到集合 A 中.(假如说两个活动的活动举办时间是不重叠的, 我们就称这两个活动是兼容活动. 一个活动的开始时间晚于一个活动结束时间就是兼容的.)

```

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]      // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
    
```

图 B.3: 活动选择问题的伪代码

B.5 搜索算法

BFS

从结点 v 开始, 首先访问结点 v, 给 v 标上已访问标记.

访问邻接于 v 且目前尚未被访问的所有结点, 此时结点 v 被检测, 而 v 的这些邻接结点是新的未被检测的结点. 将这些结点依次放置到一个称为未检测结点



表的队列 Q 中。(换句话说, 就是把所有与 v 相连但是没有被放入队列的结点入队, 是否入队可以用一个数组来表示)

若未检测结点表为空, 则算法终止; 否则

取未检测结点表的表头结点作为下一个待检测结点, 重复上述过程. 直到 Q 为空, 算法终止.(队列队首出队给 u)

```

procedure BFS(v)
    //宽度优先检索, 它从结点v开始。所有已访问结点被标记为VISITED(i)=1。//
    VISITED(v)←1 //VISITED(1:n)是一个标志数组, 初始值为VISITED(i)=0, 1≤i≤n //
    u←v
    将Q初始化为空      //Q是未检测结点的队列//
    loop
        for 邻接于u的所有结点w do
            if VISITED(w)=0 then //w与u邻接未被访问//
                call ADDQ(w,Q) //ADDQ将w加入到队列Q的末端//
                VISITED(w)←1 //同时标示w已被访问//
            endif
        repeat
        if Q 为空 then break;
        else call DELETEQ(u,Q) //DELETEQ取出队列Q的表头, 并赋给变量u//
        repeat
    end BFS

```

图 B.4: BFS 的伪代码

如果使用邻接表, 算法时间为 $O(n + e)$, 如果使用邻接矩阵, 则算法时间为 $O(n^2)$.

宽度优先周游就是对图的每个连通分支调用一次 BFS, 就是看看调用一次 BFS 之后还有没有没有被访问到的元素, 如果有的话就是继续对那个没有被访问到的元素调用一次 BFS.

图周游算法有什么应用 判定图 G 的连通性: 若调用 BFS 的次数多于 1 次, 则 G 为非连通的.

生成图 G 的连通分图: 一次调用 BFS 中所访问到的所有结点及连接这些结点的边构成一个连通分图.

构造无向图自反传递闭包矩阵 A^* : 若两个结点 i,j 在同一个连通分图中, 则在自反传递闭包矩阵中 $A^*[i,j]=1$.

构造生成树: 修改算法 BFS, 添加一个变量, 名字叫做 T(生成树), 每次遍历一次没有被访问到的结点, 就加一条边 (u,w), 边就是两个端点, 其中 u 就是现在讨论的结点,w 就是新加入至队列的结点..

从结点 v 开始, 首先访问 v, 给 v 标上已访问标记; 然后中止对 v 的检测, 并从邻接于 v 且尚未被访问的结点的中找出一个结点 w 开始新的检测. 在 w 被检测



后(检测后就是 w 的所有的邻接结点的邻接节点的邻接... 全部检测完. 有点像树的前序遍历), 再恢复对 v 的检测. 当所有可到达的结点全部被检测完毕后, 算法终止.

深度优先周游算法 DFT: 反复调用 DFS, 直到所有结点均被检测到.

对于 DFS 的生成树, 还是一个思路.

图的深度优先检索算法

```
procedure DFS(v)
//已知n结点的图G = (V,E)以及初值置零的数据组VISITED(1:n). //
VISITED(v)←1
for 邻接于v的每个结点w do
    if VISITED(w) = 0 then
        call DFS(w)
    endif
repeat
END DFS
```

图 B.5: DFS

当然我们可以修改 BFS 算法来完成 DFS, 就是在 BFS 中, 使用队列来保存未遍历到的结点. 在 DFS 中我们可以选择栈, 每次遍历就把新元素放进堆栈栈顶, 每次取元素也是从栈栈顶取元素.

回溯法

回溯法是算法设计的基本方法之一. 用于求解问题的一组特定性质的解或满足某些约束条件的最优解.

什么样的问题适合用回溯法求解呢?

1) 问题的解可用一个 n 元组 $(x_1, , x_n)$ 的向量来表示; 其中的 x_i 取自于某个有穷集 S_i .

2) 问题的求解目标是求取一个使某一规范函数 $P(x_1, , x_n)$ 取极值或满足该规范函数条件的向量.

回溯和分支限界能给我们带来系统化的搜索方法. 逐步扩充解向量的大小, 对于不满足约束条件的执行舍弃, 不往下进行扩充, 称之为剪枝.

定义 B.4

约束条件: 问题的解需要满足的条件: 可以分为显式约束条件和隐式约束条件.



显式约束条件：一般用来规定每个 x_i 的取值范围.

解空间：实例 I 的满足显式约束条件的所有元组，构成 I 的解空间，即所有 x_i 合法取值的元组的集合——可行解.

隐式约束条件：用来规定解空间中那些满足规范函数的元组，隐式约束条件描述了 x_i 之间的关系和应满足的条件.



回溯法将通过系统地检索给定问题的解空间来求解，从而需要有效地组织问题的解空间. 采用何种形式组织问题的解空间？可以用树结构组织解空间，形成状态空间树. 这里的状态指的就是解向量的状态. 比如说解向量 (1) 是一个状态，解向量为 (1, 2) 为一个状态. 所以说，搜索本质上就是对状态进行讨论. 对解进行选择，对的话继续做，不对的话就回退.

状态空间树的状态就是对解向量而言的，解向量不同状态就不同.

如果存在边的关系，代表这个状态可以进行拓展，拓展到另外一种状态. 比如说做了一个选择，解向量会进行拓展移动到别的状态.

这个时候怎么规定解向量十分重要.

状态空间：由根结点到其他结点的所有路径确定了这个问题的状态空间 (state space).

解状态：是这样一些问题状态 S，对于这些问题状态，由根到 S 的那条路径确定了这个问题解空间中的一个元组 (solution states).

答案状态：是这样的一些解状态 S，对于这些解状态而言，由根到 S 的这条路径确定了问题的一个解 (满足隐式约束条件的解)(answer states) .

在状态空间树生成的过程中，结点根据被检测情况分为三类：

活结点：自己已经生成，但其儿子结点还没有全部生成并且有待生成的结点.

E-结点 (expansion node)：当前正在生成其儿子结点的活结点.

死结点：不需要再进一步扩展或者其儿子结点已全部生成的结点.

构造状态空间树的策略：

1. 深度优先策略：当 E-结点 R 一旦生成一个新的儿子 C 时，C 就变成一个新的 E-结点，当完全检测了子树 C 之后，R 结点再次成为 E-结点.

2. 宽度优先策略：一个 E-结点一直保持到变成死结点为止.

根据深度优先策略还是宽度优先策略，可以分成回溯法和分支-限界法.

回溯法：使用限界函数的深度优先状态结点生成方法称为回溯法 (backtracking).



分支-限界方法：使用限界函数的 E 结点一直保持到死为止的状态结点生成方法称为分支-限界方法.

下面描述回溯算法:

设 (x_1, x_2, x_{i-1}) 是由根到结点 x_{i-1} 的路径.

$T(x_1, x_2, x_{i-1})$ 是下述所有结点 x_i 的集合, 它使得对于每一个 x_i , (x_1, x_2, x_{i-1}, x_i) 是由根到结点 x_i 的路径.(总得来说,T 集合就是 x_{i-1} 的儿子集合)

限界函数 B_i : 如果路径 (x_1, x_2, x_i) 不可能延伸到一个答案结点, 则 $B_i(x_1, x_2, x_i)$ 取假值, 否则取真值.

解向量 $X(1 : n)$ 中的每个 x_i 即是选自集合 $T(x_1, x_2, x_{i-1})$ 且使 B_i 为真的 x_i .

```

procedure BACKTRACK(n)
    integer k, n; local X(1:n)
    k←1
    while k>0 do
        if 还剩有没检验过的X(k)使得
            X(k) ∈ T(X(1),...X(k-1)) and B(X(1),...X(k))=true
        then
            if(X(1),...,X(k)) 是一条已抵达一答案结点的路径
                then print(X(1),...,X(k)) endif
            k ←k+1 //考虑下一个集合//
        else
            k ←k-1 //回溯到先前的集合//
        endif
    repeat
end BACKTRACK

```

图 B.6: 回溯法的一般框架

这个伪代码看上去比较乱, 现在我来解释一下:

从现在 E 结点的儿子中选择一个满足条件的当 E 结点, 如果找不到的话就回退一步, 如果找到的话就进一步. 如果刚好是答案节点, 就可以输出答案. 在这个地方, 解向量 X 是一个数组,k 表示当前解向量长度为 k, 数组的前 k 项是解. 在这里, 不同的解向量代表不同的取值,k+1 的话就代表往深处走一格,k-1 的话就表示回溯.

下面对部分问题进行一个求解:

问题 1: 已知 n 个正数的集合 $W = w_1, w_2, \dots, w_n$ 和正数 M . 找出 W 中的和数等于 M 的所有子集.

我们对解向量进行定义: n 元组 $(x_1, x_2, \dots, x_n), x_i = 1$ 或 0 . 对于 $(1, 0, 1)$ 和 $(1, 0)$,



是不同的状态.

结点: 对于 i 级上的一个结点, 其左儿子对应于 $x_i = 1$, 右儿子对应于 $x_i = 0$.

限界函数就是, 1. 当前的选择没有超过 M , 2. 剩下的数之和 + 当前的选择能超过 M 即可., 每次都往两边走, 选和不选, 只要能满足两个限界函数就可以进入.

分支-限界法

分支 - 限界法: 采用宽度优先策略, 在生成当前 E-结点全部儿子之后再生成其它活结点的儿子, 且用限界函数帮助避免生成不包含答案结点子树的状态空间的检索方法. 所有符合要求的儿子都进入队列. 然后从队列队首选择一个作为 E 结点.

LC 搜索

对下一个 E-结点的选择规则过于死板. 对于有可能快速检索到一个答案结点的结点没有给出任何优先权. 如何解决?

寻找一种“有智力”的排序函数 $C()$, 用 $C()$ 来选取下一个 E 结点. 这个函数叫做节点成本函数. 这个函数某种程度上能够该节点到答案节点的成本.

- 1) 如果 X 是答案, 那么 $C(X) =$ 由状态空间树的根结点到 X 的成本.
- 2) 如果 X 不是答案, 而且 X 的儿子到不了答案, 那么 $C(X) = \infty$.
- 3) 如果 X 不是答案, 但是 X 的儿子能找到答案, 那么 $C(X)$ 就是从根节点到子树中答案结点的成本.

但是往往答案是不可知的, 所以说我们只能估计一下 $c(X)$, 用函数 $\hat{c}(X)$ 表示. 其中 $\hat{c}(X)$ 包括两部分. $\hat{g}(X)$ 和 $h(x)$.

其中 $\hat{g}(X)$ 表示 X 到达一个答案节点所需要的成本估计函数. $h(x)$ 表示根结点到 X 的成本.

$$\hat{c}(X) = f(h(X)) + \hat{g}(X)$$

下面给出几个问题:

问题 2:15-迷问题:

问题描述: 在一个分成 16 格的方形棋盘上放有 15 块编了号的牌. 对于这些牌给定的一种初始排列, 要求通过一系列的合法移动将初始排列转换成目标排列.

在这问题中

对于当前 E 节点的所有可行的儿子入队, 然后从队列中挑选一个 C 值最小的来当 E 结点.



```

procedure LC(T,  $\hat{C}$ )
    //为找答案结点检索T,  $\hat{C}$  为结点成本估计函数,从T结点开始搜索//
    if T是答案结点 then 输出T; return endif //T为答案结点, 输出T//
    E ← T //E- 结点//
    将活结点表初始化为空
    loop
        for E的每个儿子X do
            if X是答案结点 then 输出从X到T的路径; return endif
            call ADD(X) //X是新的活结点, ADD将X加入活结点表中//
            PARENT(X) ← E //指示到根的路径//
        repeat
        if 不再有活结点 then print("no answer code"); stop endif
        call LEAST(E) //从活结点表中找  $\hat{C}$  最小的活结点, 赋给X, 并从活结点表中删除//
        repeat
    end LC

```

找到答案结点,
输出到根的路径

图 B.7: LC 搜索的伪代码

相对应的, 如果 $f(h(X))$ 为 0, 就代表回溯法, 如果 $\hat{g}(X)$ 为 0 就代表分支-限界法.

如何确保检索出来的答案是最小成本的呢?

- 1) 对于 $c(X) > c(Y), c(X) > c(Y), \hat{c}(X) > \hat{c}(Y)$. (C 是实际的结点成本函数, \hat{c} 就是估计的成本函数)
- 2) 对每一个节点有, $\hat{c}(X) \leq c(X)$.

对于限界函数而言, 我们假设: $LESS(i)$ 是这样牌 j 的数目:

$$ji, \text{but } POSITION(j) > POSITION(i)$$

即: 编号小于 i 但初始位置在 i 之后的牌的数目.

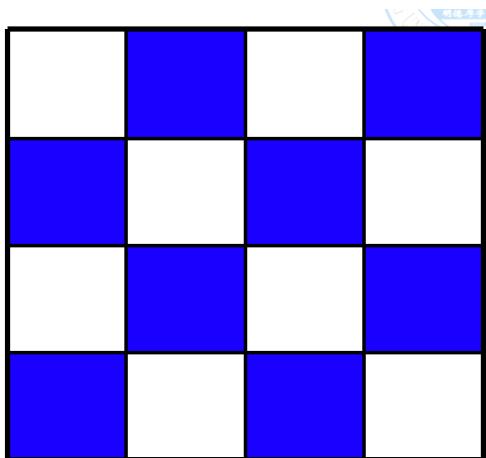


图 B.8: pic

如图所示, 初始状态时, 若空格落在蓝色方格上, 则 $X = 1$:

若空格落在白色方格上, 则 $X = 0$. 如果 $\sum_{i=1}^{16} LESS(i) + X$ 为偶数, 原命题



有解.

对于启发函数.

$$\hat{c}(X) = \hat{g}(X) + f(x)$$

其中: $h(x)$ 由根到结点 X 的路径长度.

$\hat{g}(X)$ = 不在其目标位置的非空白牌数目.

最小成本核算 $\hat{c}(X)$ 具有上界, 对于 $\hat{c}(X) > U$ 的节点, 就可以被杀死.

最小成本上界 U 的求取:

1) 初始值: 利用启发性方法赋初值, 或置为 ∞

2) 每找到一个新的答案结点后修正 U , U 取当前最小成本值.

问题 3: 作业调度, 每个作业都有一个期限值, 完成该作业所需要的时间还有完成不了的惩罚. 对于状态: 状态空间是可以变化的:

1) 元组大小可变的表示: 表示选了哪些作业, 用作业编号表示.(边对应第几个选什么)

2) 元组大小固定的表示: 表示是否选中作业, 每个作业对应一个 0/1 值.(边对应选不选第几个作业)

这个问题中, $\hat{c}(X)$ 代表罚款值. 同样可以给定一个 U 来记录之.





第 C 章 ➤ 附加数学知识

C.1 图灵机和 λ 演算

1900 年, 希尔伯特提出了一个问题, 就是对于一个整系数不定方程, 也就是整系数丢番图方程, 有没有一个可行的方法, 在有限次运算内, 可以判定这个方程有没有整数解. 希尔伯特的第 10 问揭开了关于计算理论的思考.

1928 年在意大利波伦亚举行的数学家大会上, Hilbert 和 Wilhelm Ackermann (阿克曼) 发表了"Principles of Mathematical Logic", 其中 Hilbert 10th 问题给出了更为形式化的描述, 细化成三个子问题:

First, was mathematics complete ... Second, was mathematics consistent
... And thirdly, was mathematics decidable?" (Hodges p. 91, Hawking p.
1121)数理的完整性、一致性和可判定性

在这一年之后, 1929 年, 哥德尔在他的博士论文中证明了在一阶谓词演算中所有逻辑上有效的公式都是可以证明的.

在命题逻辑里, “苏格拉底是哲学家”、“帕雷托是哲学家”只能简单标记为 p 及 q .

但是在一阶逻辑中: 我们可以把 $F(x)$, 解释成 x 是哲学家., 比如说 $F(y)$ 就是 y 是哲学家. 当然一阶逻辑的变量不仅仅是一个参数可以表示的. 其中 F 称为谓词, 在这种逻辑中, 我们可以通过量化的形式来解释一个句子: $\forall a, a \in S$, 对于任何 a, a 都属于 S , 但很可惜, 一阶逻辑只允许在论域内进行量化

二阶逻辑允许量化谓词，因此它可以做到比一阶逻辑更强的结论，或者更加一般地说，二阶逻辑允许量化任意多的元素。比如说任何集合都有上确界。

$$\forall A \left[(\exists w(w \in A) \wedge \exists z \forall w(w \in A \rightarrow w \leq z)) \rightarrow \exists x \forall y ([\forall w(w \in A \rightarrow w \leq y)] \leftrightarrow x \leq y) \right]$$

比如定义外延相等：

$$(x = y) \stackrel{\text{def}}{=} \forall P.P(x) \leftrightarrow P(y)$$

回到完备性定理的介绍：上述词语“可证明的”意味着有着这个公式的形式演绎。这种形式演绎是步骤的有限列表，其中每个步骤要么涉及公理要么通过基本推理规则从前面的步骤获得。然后如果一个公式在这个公式的语言的所有模型中都为真，它就被称为“逻辑上有效”的。为了形式的陈述哥德尔完备性定理，你必须定义这个上下文中词语“模型”的意义。这是模型论的基本定义。

1930 年 9 月 8 日，在德国柯尼斯堡召开了一次三个科学社区的联合会议。已经 68 岁、被哥廷根 (Göttingen) 大学强制退休的 Hilbert 做开幕致辞，也是他的退休演讲。这次会议第三天的一个圆桌讨论上 Kurt Gödel 阐述了他的不完备性定理，解决了前两点，并于次年发表了题为“On Formally Undecidable Propositions in Principia Mathematica and Related Systems I”的论文，正如名字所示，他当时想着要写个 II 的，不过从来没真正动手写。

他的不完备性定理分为两部分。

任何自治的形式系统，只要蕴涵皮亚诺算术公理，就可以在其中构造在体系中不能被证明的真命题，因此通过推理演绎不能得到所有真命题（即体系是不完备的）。

还有一条不完备性定理：

任何逻辑自治的形式系统，只要蕴涵皮亚诺算术公理，它就不能用于证明其本身的自治性。

在这部长篇大论里，Gödel 首先证明了一个形式系统中的所有公式都可以表示为自然数，并可以从自然数反过来得出相应的公式。这对于今天的程序员都来说，数字编码、程序即数据计算机原理，这些最核心、最基本的常识，在那个时代却是脑洞大开的创见。运用这个编码系统，Gödel 证明在系统 T 内可以形式化定义命题 P (但在系统 T 内不可被证明)，从而给确定性问题给出了否定回答。

剩下第三个问题，分别由图灵和 Church 解决，这两个人分别在 1936 年给出了问题的否定答案。



图灵认为配合一个有限状态机、一个无穷长的纸带和一个探针可以解决任何一个数学问题, 无穷长的纸带上存储了一些控制信息, 有限状态机通过探针读取纸带上的控制信息然后根据控制信息执行相关的操作并移动探针. 图灵的这个构想未日后来存储程序型的计算机提供了思路.

而 Church 在 1933 年就搞出来一套以纯 λ 演算为基础的逻辑, 以期对数学进行形式化描述, 其中 λ 演算是从数学逻辑中发展, 以变量绑定和替换的规则, 来研究函数如何抽象化定义、函数如何被应用以及递归的形式系统。 λ 演算作为一种广泛用途的计算模型, 可以清晰地定义什么是一个可计算函数, 而任何可计算函数都能以这种形式表达和求值, 它能模拟单一磁带图灵机的计算过程。但最初的 演算系统被证明是逻辑上不自洽的——在 1935 年 Stephen Kleene 和 J. B. Rosser 举出了 Kleene-Rosser 悖论。随后, 在 1936 年邱奇把那个版本的关于计算的部分抽出独立发表——现在这被称为无类型演算。在 1940 年, 他创立了一个计算能力更弱但是逻辑上自洽的系统, 这被称为简单类型演算。最后在 1936 年, 它通过论证“关于两个 λ 表达式是否等价的命题, 无法由一个“通用的算法”判断”来证明第三种问题是错误的

在 λ 演算中, 每个表达式都代表一个函数, 这个函数有一个参数, 并且会返回一个值。不论是参数和返回值, 也都是一个单参的函数。可以说, λ 演算中只有一种“类型”, 那就是这种单参函数。函数是通过表达式匿名地定义的, 这个表达式说明了此函数将对其参数进行什么操作。对于基础的 λ 演算就是利用 λ 项进行规约和演算的。

Name	Syntax	Example	Explanation
Variable	$<\text{name}>$	x	a variable named “x”
Function	$\lambda <\text{parameters}>. <\text{body}>$	$\lambda x. x$	a function with parameter “x”
Application	$<\text{function}> <\text{variable or function}>$	$(\lambda x. x) a$	calling the function “x.x” with argument “a”

接着就是三种规约: 第一种是 α 等价, 也就是把字母长什么样给替换了也无所谓.

$$\lambda x. P =_{\alpha} \lambda y. P[y/x] \quad (y \notin FV(P))$$

接下来是了 β 规约:

$$((\lambda V. E) E) = E[V := E]$$



其实就是用实参替换函数体中的形参.

根据 β 规约提出替换的概念: 形式: $E[V := R]$, 意为将表达式 E 中的所有“自由变量” V 替换为表达式 R

最后就是 η 规约:

$$x.Mx = M$$

其中 M 与 x 无关.

下面我们来对 λ 演算进行一些基本的介绍:

首先我们了解变量, 在函数 $\lambda x.x$ 是约束变量, 被称为约束变量, 因为它既在函数体中又是形参. 但是在函数 $\lambda x.a$ 是自由变量. 根据 β 规约可以进行化简. 尽管 lambda 演算传统上仅支持单个参数的函数, 但我们可以通过一种叫作柯里化(Currying) 的技巧创建多个参数的函数.

$$(\lambda x.\lambda y.\lambda z.xyz) \Rightarrow (\lambda x.y.z)=xyz$$

尽管 lambda 演算中没有数字, 我们还可以用邱奇编码(Church numerals) 将数字嵌入到 lambda 演算中. 对于任何数字: $n : n = \lambda f.f^n$ 来表示.

比如说 3:

$$3 = \lambda f.\lambda x.f(f(x))$$

同样用这种类型的函数来表示计算也是一种可行的方法. 所以说截止 1936 年, 一共有两种可以自动计算并解决问题的方法: 图灵机和 λ 表达式.

C.2 凸优化

如果您对 AI 感兴趣, 你可以了解一下凸优化, 这个对于构建有限状态机很有帮助.

凸优化问题的基本形式

对于一个凸优化问题, 限制条件和优化目标绝对是逃不掉的重中之重. 而限制条件又可以分成等量关系的限制和不等关系的限制, 所以, 我们规定一个凸



优化问题的标准形式如下：

$$\begin{aligned} \min_x F(x) \\ \left\{ \begin{array}{l} f_i(x) \leq 0 \\ Ax + b = 0 \end{array} \right. \end{aligned}$$

拉格朗日乘子法

即使你没学过凸优化，应该也知道如何用拉格朗日乘子法。如果你学过微积分下或者多元数学分析。我们通过引入一个所谓的拉格朗日乘子 λ 把限制的条件带进目标，然后像求普通函数的极值一样求函数的极值就可以了。

就是说，求 $f(x, y)$ 在 $g(x, y) = 0$ 的条件下的极值，可以化成一个函数 $F = f(x, y) + \lambda g(x, y)$ 无差别地求三元极值就行。极值求法大家应该还记得吧，就是对各个元素求偏导等于 0 得到一个方程组解方程组就行。

KKT 条件和对偶问题

KKT 实际上是对上面拉格朗日乘子的泛化。前面我们说到拉格朗日乘子的时候，我们只考虑了等量关系，但是不等关系怎么说？我们能不能一样引进一种乘子来求解呢？唉，其实是可以的，只需要我们的乘子能够满足：

$$\begin{cases} g(x) < 0, \lambda = 0; \\ g(x) = 0, \lambda > 0; \end{cases} \Rightarrow \lambda \geq 0, \lambda g(x) = 0$$

这次我们引入的就是 KKT 乘子。通过引入 KKT 乘子，我们的优化问题就可以无差别为一个函数：

$$L(\alpha, \beta, x) = f(x) + \sum_{i=0}^m \alpha_i g_i(x) + \sum_{i=0}^n \beta_i h_i(x)$$



其中， $\begin{cases} h(x) = 0; \\ g(x) \leq 0; \\ \alpha \geq 0; \\ \alpha_i g_i(x) = 0 \end{cases}$ 。这就是不等式约束优化问题的 KKT 条件 (Karush-Kuhn-Tucker Condition)，KKT 条件是拉格朗日乘子法在不等式约束优化问题上的泛化。

KKT 条件是极小点的必要条件，即满足 KKT 条件不一定是极小点，但极小点必满足 KKT 条件。

将原始问题转化为对偶问题是求解带约束优化问题的一种方法，当然这不是唯一的方法，只不过转化为对偶问题后往往更容易求解，因而被广为应用。我们把函数 L 对 x 求出极小值以后，剩下了拉格朗日乘子和 KKT 乘子我们还暂未考虑过。现在我们把结果看作 $G(\alpha, \beta)$ ，求它的极大值就可以得到对偶问题的最优解。Lagrange 对偶问题是一个凸优化问题，与原问题的凸性无关。拉格朗日对偶的结果必然不会超过最优解，这称为对偶性。

C.3 信息论

对于信息论的知识，大家可以简单了解一下。

信息熵：如果对于样本集合 S ，一共可以划分为 k 个类，每个类概率是 p_k ，那么信息熵定义为：

$$E(S) = - \sum_{i=1}^k p_k \log p_k$$

信息增益与信息增益率：如果对数据集 S 按照某一个属性 A 对 S 进行划分，将它划分成 v 个子集，定义属性 A 的信息熵为：

$$E(S, A) = \sum_{i=1}^v \frac{|A_i|}{|S|} E(A_i)$$

数据集本身的 k 个类是按照最后的返回标签排的，所以数据集本身的信息熵与属性的信息熵并不是一个东西。信息增益就是这个差值，看看按照这个属性划分我的信息到底增长了多少。

$$Gain(A) = E(S) - E(S, A)$$



增益率是增益与信息熵的比值。

$$GainRatio(A) = \frac{Gain(A)}{E(S, A)}$$

基尼指数：假设数据集有 n 个类，第 k 类的概率是 p_k ，定义基尼指数：

$$GINI(S) = 1 - \sum_{i=1}^n p_k^2$$

C.4 数值分析

数值分析应该说是数学与计算机学科的专业课。不过我们提到的算法可能没有那么多，我们可能比较多的就用两个算法吧，梯度下降和牛顿法。我们了解数值分析主要是为了帮助我们更好地去理解计算机相关的知识

梯度下降法

梯度下降是机器学习最核心的数值分析方法没有之一！这个方法适合于求各种优化目标函数的数值解。什么是梯度呢？对于一个多元函数而言，梯度的定义：

$$grad(F) = \left\{ \frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \dots, \frac{\partial F}{\partial x_n} \right\}$$

那它有啥几何意义吗？有的。它是沿着法线方向的，指向变化最快的一端。把寻优过程看作爬山，沿着梯度指的方向，一步慢慢移动一点距离，最终我们可以抵达山底也就是函数的最低点。

如果我们以一个向量形式的偏导数 $\frac{\partial F}{\partial \mathbf{x}}$ 代替梯度，爬山的每一步步长我们称为学习率 α ，我们得到梯度下降的表达式：

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \frac{\partial F}{\partial \mathbf{x}}$$

梯度下降一个比较典型的问题就是局部最小。因为是个数值算法，所以当它沿着梯度方向走进了一个盆地以后就收敛不动了，然而最小值可能并不在这里，在山的另一个区域可能有一片更大更低的盆地，我们如果始终从这一个位置沿着一条既定的路线走可能会走进死胡同得不出正确答案。所以，选择合适的初始点，合适的步长进行比较才能得出更加准确的数值解。

梯度下降家族有三种算法，分为在线、离线和折衷三部分。这三种算法叫做**批量梯度下降，随机梯度下降，小批量梯度下降**。感兴趣的同学可以查一下资料，这三种都是梯度下降的方式。



牛顿法

牛顿法还有个名字叫做切线法。具体的我拿一个一元函数举个例子吧。首先牛顿法可以用来解方程，譬如说我们解 $x^2 - 3 = 0$ ，我们取一个初值 2，函数在 2 这一点的切线是 $y = 4x - 4$ ，切线的零点是 1，然后我们再来找函数在 1 处的切线零点，不断迭代直到次数达标或者两个解的相差满足了精度。把它写成方程是这样的：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

如果牛顿法用来求最优化问题，本质上也是解一个方程。导数方程嘛，取极值的条件嘛，对吧。我们的变化也不大，只需要：

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

就可以了。

C.5 数论

C.5.1 基本理论知识

1. 整除

整数集定义: $\mathbb{Z} = \{\dots - 2, -1, 0, 1, 2, \dots\}$

自然数集定义: $\mathbb{N} = \{0, 1, 2, 3, \dots\}$

整除定义：设 $a, b \in \mathbb{Z}, a \neq 0$, 如果存在 $k \in \mathbb{Z}$ 使得 $b = aq$, 就说 b 被 a 整除，记作 $a | b$, 称 b 为 a 的倍数, a 是 b 的约数。若 b 不能被 a 整除就记作 $a \nmid b$ 。

2. 素数（不可约数）如果 $p \in \mathbb{Z}, p \neq 0, \pm 1$, 且它的约数只有 $\pm 1, \pm p$ 。则称 p 为素数。

素数可以是负的，但没什么用，所以以后说素数是总是指正的素数。注：素数有无穷多个。是因为若只有有限个，假设为 p_1, p_2, \dots, p_k , 那么令 $p_{k+1} = p_1 p_2 \cdots p_k + 1$, p_{k+1} 也是素数，矛盾。

除了 1 和素数之外的数称为合数。1 既不是质数也不是合数。

3. 最大公约数与最小公倍数

我们称 n 个自然数 x_1, x_2, \dots, x_n 共有的约数叫做公约数，公约数最大的那个叫做最大公约数，记作 (x_1, x_2, \dots, x_n) , 我们用 $\mathcal{D}(a_1, \dots, a_n)$ 来表示所有正公约数的全体。

我们称 n 个自然数 x_1, x_2, \dots, x_n 共有的倍数叫做公倍数，公约数最小的那个



叫做最小公倍数, 记作 $[x_1, x_2, \dots, x_n]$, 我们用 $\mathcal{L}(a_1, \dots, a_n)$ 来表示所有正公倍数的全体。

我们有 $(a, b) = (a - kb, b)$ 和 $(a, b)[a, b] = ab$ 两个重要性质。当 $(a, b) = 1$ 时, 我们称 a, b 互素。

4. 裴蜀定理 (Bezout)

若 $a, b \in \mathbb{Z}$, 则存在整数 u, v , 使得 $ua + vb = (a, b)$ 。

注: 这个定理可以推广到 n 个元素。当然, 它在多项式域 $\mathbb{K}[X]$ 内也是成立的。甚至在更广的一些代数理论下仍然是成立的!

若 $a, b \in \mathbb{Z}, a \neq 0$, 那么一定存在惟一的一对整数 q 与 r , 满足

$$b = qa + r, \quad 0 \leq r < |a|$$

并且, $a | b \Leftrightarrow r = 0$ 。我们称 q 为商, r 为余数。

5. 整数分类 设 $a \geq 2$ 是给定正整数, $r = 0, 1, \dots, a - 1$, 则除 a 余 r 所构成的集合为:

$$\{ka + r, \quad k \in \mathbb{Z}\}$$

我们记为 $r \bmod a$, 叫作剩余类。则 $\mathbb{Z} = \bigcup_{r=0}^{a-1} (r \bmod a)$ 。

若 $x, y \in (r \bmod a)$, 则可重新表示为: $x \equiv y \equiv r \pmod{a}$ 。

完全剩余系: 设 K_1, K_2, \dots, K_a 为 $\bmod a$ 的所有剩余类, 从每个剩余类 K_i 当中选出代表元素 r_i , 则 $\{r_1, r_2, \dots, r_a\}$ 称为 $\bmod a$ 的一个完全剩余系, 简称完系。

我们称 $\{0, 1, \dots, a - 1\}$ 为 $\bmod a$ 的最小非负完系。

下面介绍数论中最重要的一个定理!

唯一分解定理 (算术基本定理) :

任何一个自然数 n 皆可以唯一的表示成素数之积: $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}$, 其中 $p_1 < p_2 < \cdots < p_k$ 为素数, $\alpha_1, \alpha_2 \cdots \alpha_k$ 为自然数。

我们先看看这个定理有多重要:

定义 Euler 乘积为: $\zeta(s) = \prod_p \left(1 - \frac{1}{p^s}\right)^{-1}$ 。则 $\ln \zeta(x) = \sum_p \ln \left(1 - \frac{1}{p^x}\right)^{-1}$ 因而我们有:

$$\sum_p \frac{1}{p^s} < \sum_p \ln \left(1 - \frac{1}{p^s}\right)^{-1} < \sum_p \frac{1}{p^s - 1} < 2 \sum_p \frac{1}{p^s} < 2 \sum_{n=1}^{\infty} \frac{1}{n^s}, \quad s > 1$$

所以 $\zeta(x) = \prod_p \left(1 - \frac{1}{p^x}\right)^{-1}$ 在当 $x > 1$ 时是收敛的。

设 $c(n)$ 为 n 表示为 $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ 的方法个数, 约定 $c(1) = 1$, 则算术基本定理等价于对任意 $n, c(n) = 1$ 。



由于对任意的 $N \in \mathbb{N}$, 存在 $k \in \mathbb{N}, 2^{k-1} \leq N < 2^k$, 所以:

$$\begin{aligned}\sum_{n=1}^N \frac{c(n)}{n^s} &\leq \prod_{p \leq N} \left(1 + \frac{1}{p^s} + \frac{1}{p^{2s}} + \cdots + \frac{1}{p^{ks}}\right) \\ &\leq \prod_{p \leq N} \left(1 - \frac{1}{p^s}\right)^{-1} \\ &\leq \prod_p \left(1 - \frac{1}{p^s}\right)^{-1}, \quad s > 1\end{aligned}$$

所以我们有 $\sum_{n=1}^{\infty} \frac{c(n)}{n^0} \leq \prod_p \left(1 - \frac{1}{p^2}\right)^{-1}$ 。

而当我们对任意的 M 和 h , 取 $N_1 = \prod_{p \leq M} p^h$, 由 $c(n)$ 的定义容易知道:

$$\prod_{p \leq M} \left(1 + \frac{1}{p^s} + \cdots + \frac{1}{p^{ks}}\right) \leq \sum_{n=1}^{N_1} \frac{c(n)}{n^s} < \sum_{n=1}^{\infty} \frac{c(n)}{n^s}, \quad s > 1$$

$$\text{今 } h \rightarrow \infty \text{ 和 } M \rightarrow \infty \text{ 得 } \sum_{n=1}^{\infty} \frac{c(n)}{n^s} \geq \prod_p \left(1 - \frac{1}{p^s}\right)^{-1}.$$

所以我们有结论:

算术基本定理等价于欧拉恒等式: $\sum_{n=1}^{\infty} \frac{1}{n^d} = \prod_p \left(1 - \frac{1}{p^d}\right)^{-1}$ 。算术基本定理证明:

将 a 的分解由小到大排列 $a = p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_\Delta$, 其中, $p_1 \leq p_2 \leq \cdots \leq p_r$, $q_1 \leq q_2 \leq \cdots \leq q_s$ 下面证明 $r = s, p_i = q_i (i \in \{1, 2, \dots, s\})$ 。

由于 $q_1 | p_1 p_2 \cdots p_n$, 必然 $\exists i$, 使得 $q_1 = p_i$, 同理有 $p_1 = q_j$ 。再由 $p_i \geq p_1$, 和 $q_i \geq q_1$ 得到 $p_1 = q_1$ 。从而我们有 $p_2 p_3 \cdots p_r = q_2 q_3 \cdots q_b$, 依次下去, 分别可以证明 $p_2 = q_2, p_3 = q_3 \cdots$ 。因此知道原命题成立。

推论:

约数个数定理: 记 $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}$, 用 $d(n)$ 表示所有正约数的个数 (当然也可以用 $\tau(n)$, 则 $d(n) = (\alpha_1 + 1)(\alpha_2 + 1) \cdots (\alpha_k + 1)$)

约数和定理: 记 $n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_k^{\alpha_k}$, 用 $\sigma(n)$ 表示所有正约数的和。则:

$$\sigma(n) = \frac{p_1^{\alpha_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{\alpha_2+1} - 1}{p_2 - 1} \cdots \frac{p_k^{\alpha_k+1} - 1}{p_k - 1}$$

注: 依据定义, 我们能够知道 $d(n) = \sum_{d|n} 1$ 和 $\sigma(n) = \sum_{d|n} d$ 。

我们还能得到一般结论: $\sum_{d|n} f(d) = \sum_{i_1=0}^{\alpha_1} \cdots \sum_{i_k=0}^{\alpha_k} f(p_1^{i_1} \cdot p_2^{i_2} \cdots p_k^{i_k})$ 和 $\prod_{d|n} f(d) = \prod_{i_1=0}^{\alpha_1} \cdots \prod_{i_k=0}^{\alpha_k} f(p_1^{i_1} \cdot p_2^{i_2} \cdots p_k^{i_k})$ 成立。



完全平方数：对于正整数 a , 若存在 $b^2 = a \quad b \in \mathbb{N}$, 则称 a 为完全平方数。若存在 $n \in \mathbb{N}$ 使得 $n^2 < a < (n+1)^2$, 则 a 不是完全平方数。

C.5.2 常用定理证明 *

在有了数论的基本概念比如整除, 同余, 约数之类的概念还有算术基本定理之后, 我们就可以细细推导数论当中的公式了。

我们先引入多项式同余的概念:

定义: 设 $f(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_0, g(x) = b_nx^n + b_{n-1}x^{n-1} + \cdots + b_0$ 为两个整系数多项式, 若对于 $\forall x \in \{0, 1, \dots, n\}$ 都有 $a_i \equiv b_i \pmod{m}$ 称 $f(x) \equiv g(x) \pmod{m}$ 。

拉格朗日定理 (Lagrange's theorem)

内容: 设 p 为素数, 考察 \pmod{p} 意义下的整系数多项式,

$$f(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_0 \quad (p \nmid a_n)$$

则同余方程 $f(x) \equiv 0 \pmod{p}$ 在 \pmod{p} 意义下之多 n 个不同的解。

证明: 对 n 用第二数学归纳法, 当 $n = 0$ 时, 由于 $a_n \neq 0$, 所以无解。现在假设次数小于 n 的多项式均成立, 则用现在利用反证法, 假设 $f(x) \equiv 0 \pmod{p}$ 当 $x = x_0, x_1, \dots, x_n$ 时都是方程的解, 则当 $i \in \{1, 2, \dots, n-1\}$:

$$(x_i - x_0)g(x_i) \equiv f(x_i) - f(x_0) \equiv 0 \pmod{p}$$

而由于 x_0 与 x_i 不同余, 所以 $g(x_i) \not\equiv 0 \pmod{p}$, 所以 $g(x) \equiv 0 \pmod{p}$ 至少 n 个解, 矛盾!

卢卡斯定理 (Lucas's theorem)

内容: 设 p 为素数, $a, b \in \mathbb{N}^*$, 并且:

$$a = a_kp^k + a_{k-1}p^{k-1} + \cdots + a_1p + a_0$$

$$b = b_kp^k + b_{k-1}p^{k-1} + \cdots + b_1p + b_0$$

这里 $0 \leq a_i, b_i < p$ 为整数, 则我们有:

$$\begin{pmatrix} a \\ b \end{pmatrix} \equiv \begin{pmatrix} a_k \\ b_k \end{pmatrix} \begin{pmatrix} a_{k-1} \\ b_{k-1} \end{pmatrix} \cdots \begin{pmatrix} a_0 \\ b_0 \end{pmatrix} \pmod{p}$$



证明: 由于 p 为素数, 所以对于 $i \in \{1, 2, \dots, p-1\}$ 时, 我们有:

$$\binom{p}{i} = \frac{p}{j} \binom{p-1}{j-1} \pmod{p}$$

所以由 p 进制数的性质, 我们知道: $\sum_{b=0}^a \binom{a}{b} x^b = (1+x)^a = \prod_{i=0}^k ((1+x)^{p^i})^{a_i}$

$$\equiv \prod_{i=0}^k (1+x^{p^i})^{a_i} = \prod_{i=0}^k \left(\sum_{b_i=0}^{a_i} \binom{a_i}{b_i} x^{b_i p^i} \right) = \prod_{i=0}^k \left(\sum_{b_i=0}^{p-1} \binom{a_i}{b_i} x^{b_i p^i} \right) =$$

$$\sum_{b=0}^a \left(\prod_{i=0}^k \binom{a_i}{b_i} \right) x^b \pmod{p}, \text{ 证毕。}$$

Fermat-Euler 定理

内容: 设 $(a, m) = 1$, 则有:

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$

特别当 p 为素数时, 对任意 a 均有

$$a^{p-1} \equiv 1 \pmod{p}$$

我们通常把 $a^{\varphi(m)} \equiv 1 \pmod{m}$ 称为 Euler 定理, 而把 $a^p \equiv a \pmod{p}$ 称作 Fermat 小定理。

证明: 我们把 $r_1, r_2, \dots, r_{\varphi(m)}$ 作为 m 的简缩剩余系。则 $ar_1, ar_2, \dots, ar_{\varphi(m)}$ 也是简缩剩余系, 那么有:

$$\prod_{i=1}^{\varphi(m)} r_i = \prod_{i=1}^{\varphi(m)} ar_i = a^{\varphi(m)} \prod_{i=1}^{\varphi(m)} r_i$$

由于 $(r_i, m) = 1$, 所以我们有 $a^{\varphi(m)} \equiv 1 \pmod{m}$ 成立。

威尔逊定理 (Wilson's theorem)

内容: 设 p 为素数, 则 $(p-1)! \equiv -1 \pmod{p}$ 。利用拉格朗日定理可以很快速的导出此定理。

记多项式 $f(x) = (x-1)(x-2)\cdots(x-p+1) - (x^{p-1} - 1)$ 。则 $f(x)$ 在 \pmod{p} 意义下有 $p-1$ 个

解: $1, 2, \dots, p-1$, 而 $f(x)$ 的次数比 $p-1$ 小, 所以

$$f(x) \equiv 0 \pmod{p}$$



所以当 p 为奇数时，常数项为 $-(p-1)! + 1$ ，即威尔逊定理成立。

中国剩余定理（孙子定理）

内容：设 m_1, m_2, \dots, m_n 两两互素，对于任意整数 a_1, a_2, \dots, a_n ，给出了以下的一元线性同余方程组：

$$\left\{ \begin{array}{l} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_n \pmod{m_n} \end{array} \right.$$

则同余方程组在 $\text{mod } m_1 m_2 \cdots m_n$ 意义下有且仅有一个解 x_0 。

注：设 $M = m_1 \times m_2 \times \cdots \times m_n = \prod_{i=1}^n m_i$ 和 $M_i = \frac{M}{m_i}, \forall i \in \{1, 2, \dots, n\}$ ，则我们有：

$$x_0 = a_1 M_1^{-1} M_1 + a_2 M_2^{-1} M_2 + \cdots + a_n M_n^{-1} M_n = \sum_{i=1}^n a_i M_i^{-1} M_i$$

这个定理的证明是容易的，令 $x_1 = a_1$ ，则 x_1 满足第一个方程且只有这 1 个解，考察下面 m_2 个数：

$x_1, x_1 + m_1, x_1 + 2m_1, \dots, x_1 + (m_2 - 1)m_1$ ，构成完全剩余系，有且仅有 1 个满足第二个方程。这样不断迭代下去就完成本题的证明了。

注：孙子定理解决了所有的 1 元 1 次不定方程。我们现在来解决所有的 n 元 1 次不定方程。

引入矩阵同余， $A \equiv B \pmod{m}$ ，是指 $a_{ij} \equiv b_{ij} \pmod{m}$ ($\forall i \in \{1, 2, \dots, n_1\}, j \in \{1, 2, \dots, n_2\}$)。我们利用孙子定理，总是可以将 n 元 1 次不定方程转为如下方程组：

$$A \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \equiv \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \pmod{m}$$



而此方程在 $\text{mod } m$ 意义下有唯一解

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \equiv \det(A)^{-1} \cdot \text{adj}(A) \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$





第 D 章 附加电路知识

D.1 硬件设计基础

注：这部分导论主要查阅了 CSDN 博主“队长-Leader”的文章。

硬件设计，可以说是包罗万象，它涉及到非常庞大的知识量，一个电路错一点小地方，都有可能导致整个系统不能工作。本文主要只介绍初级理论篇（电路理论，模拟电路，数字电路）和中级理论篇（信号与系统）。当然在此基础上，高等数学、线性代数、复变函数和大学物理等这些基础课要过关，根基扎实。

就电路理论而言，其实电路基础的理论并不难，但是有些抽象的东西，是暂时不能很好地理解，比如说受控源，所以学完模电还要再回过头来再看一遍。模拟电子技术是电子专业的核心基础课，同时还可以配合 Multisim 仿真软件加深理解。如果说电路基础、高数当中的答案都是明确、唯一的，那么模电的答案将是不明确、多样化的，需要在实践中权衡取舍，一定要把以前的思维转变过来，不然后面没法学。这门课全部都是重点，但是学完它，除了抄书上的电路，你仍然什么都做不了，因为还需要其它方面的知识一起用才可以。数字电子技术相对于模电来说，要简单很多很多。它把三级管搭成各种门电路、触发器，以便于直接把数学知识运用起来，同时它也是 FPGA 的先修课，是硬件工程师向算法工程师（跟计算机的算法有很大区别）转变的基础。这门课全部都是重点，但是要真正掌握它，还是得学 FPGA 才可以。

中级理论篇首先需要依赖更高阶的数学基础（复变函数）。这门课跟高数的微积分一样，是一种数学工具。复数信号是物理不可实现的，但是为什么需要

复数？诚然，正弦波（包括余弦，下同）有振幅、频率和相位三要素，如何在一个图上面表示振幅与频率的关系或者相位与频率的关系（方便观察分析才需要这样弄）？这就需要用到复数了，其中 j （因为电流的符号是 i ，所以才换成 j ，以防混淆）表示的就是方向，对应着极坐标的向量。我们可以把复数转成模和辐角的形式，想象一下，模就是时钟的秒针，而辐角就是秒针转动的角度，秒针转一圈就是个圆，而把这个圆的各点按照出现的时间先后，重新描绘在直角坐标系中，就是一个正弦波。这就意味着，用复数可以表示一个正弦波的三要素，振幅就是模（秒针的长短），相位就是秒针转动的角度，频率就是秒针转动的快慢。想一下，如果用实数来表示正弦波的三要素，是不是很麻烦？信号与系统主要介绍如何利用数学建模去描述电路，就是这门课要研究的内容。什么是信号？LED 灯的亮灭、喇叭发出的声音、天线感应的电磁波等，有实际用途的信息载体（包括声、光、电、热等）都是信号。什么是系统？就是处理信息载体的东西（包括放大器、传动装置等）。系统是一种更为抽象的概念，可大可小，小到一个三极管，大到一个无线收发装置，这些都要根据实际需求来确定，不能一概而论。这门课都是重点。当然中级理论还包括高频电子线路、单片机、自动控制原理等等，就不一一介绍了。

D.2 电路理论

电路理论课程以分析电路中的电磁现象，研究电路的基本规律及电路的分析方法为主要内容，全面系统地介绍了电路分析的基本原理和基本分析方法。电路理论是当代电工科学技术的重要理论基础之一，它有两个部分组成，电路分析和电路综合。电路分析研究的直接对象是电路模型，是由实际电路抽象出来的。核心内容有电路模型及其两类约束、等效和替代、线性叠加、选择合适电路变量建立电路方程等。电阻电路是指除电源之外，仅包含有电阻元件的电路，而动态电路则是含有电容、电感等储能动态元件的电路，由于动态元件的特性方程当中含有微分与积分形式，因而动态电路主要采用微分方程进行描述，其中方程的阶数通常取决于动态元件个数。电路分析当中的动态又称为暂态，指电路从一个稳态变化至另一个稳态的中间过渡过程。而稳态也被称为静态，指电路当中的电压、电流等参数达到一个稳定状态，如果其它参数不发生变化，就会一直以该状态运行。无论是分析简单电路，还是分析复杂电路，通常都会遵循如下一系列步骤：1. 确定分析目标 2. 建立电路模型 3. 计算目标变量 4. 选择分析方法 5. 校验所得结果。



D.2.1 Part 1

一些概念

- 独立源用于提供电能，只是其端电流/电压独立于其它电路变量受控电源用于表征变量之间的耦合关系，为此需要定义电压控制电流源 VCCS、电流控制电流源 CCCS、电压控制电压源 VCVS、电流控制电压源 CCVS 一共 4 种受控电源。
- 电压源与电阻的串联称为戴维南支路，而电流源与电阻的并联称为诺顿支路。

基本定律

- 基尔霍夫电流定律 (KCL)：集中参数电路当中，任意时刻流入结点的支路电流的代数和为零：

$$\sum_{k=1}^b i_k = 0$$

- 基尔霍夫电压定律 (KVL)：集中参数电路当中，任意时刻回路当中所有支路电压的代数和为零：

$$\sum_{k=1}^b u_k = 0$$

- 拥有 n 个结点 b 条支路的电路，可以列写出 $n - 1$ 个独立的 KCL 方程，以及 $b - n + 1$ 个独立的 KVL 方程

下面电路拥有 4 个结点 6 条支路，支路形式包括了电阻支路、戴维南支路、诺顿支路，如何确定所有支路电流与电压？

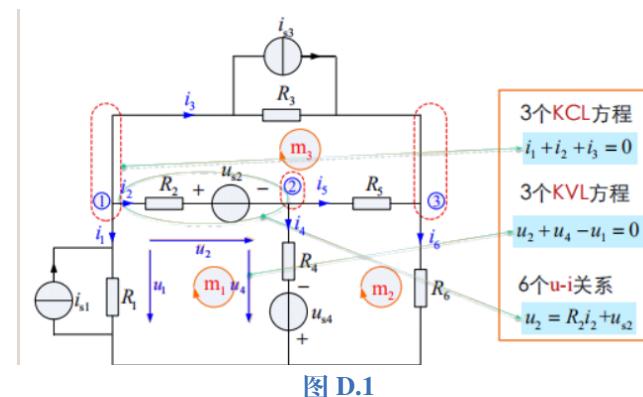


图 D.1

如果以支路电流与支路电压为变量列写方程，那么就需要列写出如下 12 个电路基本方程：1. 3 个结点的 KCL 方程；2. 3 个网孔的 KVL 方程；3. 6 条支路的电压电流关系；上面这 12 个方程是电路的基本方程，虽然列写容易，但是联

立求解过程较为复杂。为了减少联立求解方程数量，电路理论进一步提出了结点方程与网孔方程。采用结点方程来分析电路称为结点分析法，它是由独立结点的 KCL 方程来确定结点电位，再由结点电位确定支路电压与电流；当电路中不存在电压源支路的时候，结点方程有 $n - 1$ 个（其中 n 为结点数量）。采用网孔方程来分析电路称为网孔分析法，它是由网孔的 KVL 方程来确定网孔电流，再由网孔电流确定支路电流与电压；当电路中不存在电流源支路时，网孔方程有 $b - n + 1$ 个（其中 b 为支路数量）。

D.2.2 Part 2

电阻电路等效变换

想要确定下面电路当中的电流，可以将 3 个电阻等效为 1 个 5Ω 的电阻，由此可知电流 i 等于 1 A。

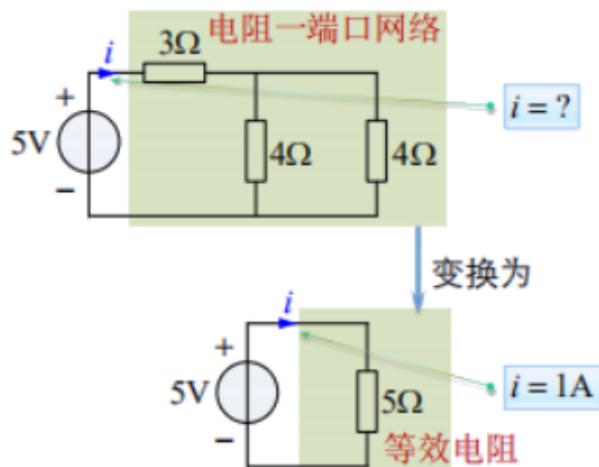


图 D.2

这种分析电路的方法就称为等效变换，而串联和并联是最基本的等效变换方法。

小结

- 端口电压电流关系相同的网络称为等效网络；
- 由线性电阻构成的一端口网络可以等效为一个线性电阻，称为等效电阻；
- 电压源并联任何元件或者一端口网络，均等效为电压源；
- 电流源串联任何元件或者一端口网络，均等效为电流源；
- 重新绘制电路并且去掉短路线是化简混联电路的有效方法；



D.2.3 Part 3

电路定理

- 叠加定理：关注线性电路如何去体现齐次性与可加性。叠加定理一般表述为线性电路当中，多个独立电源共同作用下的响应（即电路当中的任意电压与电流）等于各个独立电源单独或者分组作用下的响应之和。
- 替代定理：替代定理应用的前提是原电路和替代之后的电路均具有唯一解（线性电路通常具有唯一解）；替代是维持工作点不变，改变电路的参数，就意味着改变了被替代支路的工作点，因而替代该支路的电压电流源也会发生变化；等效是维持电压电流关系不变，即使改变了电路的参数，只要被等效部分的参数不变，等效电路就不会变化。
- 戴维南定理和诺顿定理：戴维南定理和诺顿定理，是确定含源一端口网络等效电路的另外一种方法。戴维南定理与诺顿定理，是获得含源一端口网络等效电路的首选方法，其更加优于等效变换法；应用戴维南定理与诺顿定理时，需要计算开路电压、短路电流、等效电阻当中比较容易计算获得的两个；分析最大功率传输问题时，将含源一端口网络等效为戴维南电路，再应用最大功率传输定理；切记可以变化的是负载参数，一端口网络的等效参数不变。

D.2.4 Part 4

正弦稳态电路的功率

有功功率与无功功率是非常重要的概念，必须深刻理解其物理含义：无源网络吸收的瞬时功率 = 恒为正的消耗功率 + 均值为零的交换功率。电阻吸收的瞬时功率恒大于零，属于消耗功率；电感、电容吸收的瞬时功率依照正弦变化，属于交换功率；有功功率等于吸收功率的平均值；无功功率是交换功率的幅值；为了体现电感、电容的互补性，通常约定电感吸收无功功率，电容发出无功功率。

在无功功率、有功功率概念的基础之上，扩展出了视在功率、功率因数、复功率以及复功率守恒的概念。但是，有功/无功功率才是正弦稳态概念的核心，需要在深刻理解两者概念的基础之上，熟练掌握复功率、视在功率、功率因数与有功/无功功率的关系，并且理解各种功率守恒的意义。



正弦稳态电路的功率	
复功率	$\bar{S} = P + jQ = UI^* = UI \angle(\phi_u - \phi_i)$
视在功率	$S = UI = \sqrt{P^2 + Q^2}$
有功功率	$P = UI \cos(\phi_u - \phi_i)$
无功功率	$Q = UI \sin(\phi_u - \phi_i)$
功率因数	$\lambda = \frac{P}{S} = \cos(\phi_u - \phi_i)$
功率守恒	$\sum \bar{S}_k = 0 \quad \sum P_k = 0 \quad \sum Q_k = 0$

图 D.3

D.3 模拟电子技术

D.3.1 学习导论

模拟电子技术(简称模电)是电子、电气、自动化、计算机等相关专业的专业课程, 模电和数电是做电路设计相关工作必须掌握的专业知识, 模电相对来说确实比较难, 比数电要难一些。模电是一门研究半导体二极管、半导体三极管和场效应管为关键电子器件, 包括功率放大电路、运算放大电路、反馈放大电路、信号运算与处理电路、信号产生电路、电源稳压电路等的学科。

模电学习上启电路(电路分析更复杂), 下承通信电子线路, 要懂得一些管子的知识, 像常见的三级管了: NPN, PNP, CMOS 管等, 了解他们的伏安特性。理解放大原理, 常用的放大倍数计算的公式, 会画电路的微变电路, 弄清楚输入电阻和输出电阻。再一部分就是基本的放大电路: 射级放大, 集电极放大, 基级放大。对电路的分析, 知道放大是放大的什么。

很多学生学习模电时感觉很难, 模电之所以难, 是因为模拟电路形式多种多样, 千变万化, 而且很多参数计算分析复杂。当然, 难和易是相对的, 只要自己努力、用心去学, 我相信都可以学得好。模电入门阶段一定要弄清楚 PN 结的结构原理, 以及电流形成过程, 三极管的电流走向与分配关系等, 入门理顺了, 后面的学习相对会轻松一些。

后面章节的集成运放、比较器也是必须要掌握的, 运放和比较器在电路设计中很常用, 一定要熟悉最基本的几种运放电路模型(反相比例放大、同相比例放大、加法器、减法器、差分放大等), 会应用运放“虚断”与“虚短”两个重要特性分析运放电路。

学习模电要多看、多思考, 课后最好到图书馆结合基本参考书认真复习。课余时间最好多动手实践, 多参加一些电子项目设计, 比如电子设计竞赛, 那是



非常锻炼人的竞赛项目，参加电子设计竞赛，特别锻炼人，可以从中学到很多东西。

D.3.2 基本知识

需要掌握的基本知识（非全部）：

1. 直流通路三原则：

- 交流电压源短路，交流电流源开路；
- 大电容视为开路；
- 大电感视为短路。

2. MOSFET：截止区、饱和区、可变电阻区的判断；

3. BJT：截止区、放大区、饱和区的判断；

4. 转移特性曲线、传输特性曲线的理解；

5. 直流负载线的理解，判断截止失真和饱和失真；

6. 静态工作点分析、小信号模型分析

7. 各种放大组态的分析以及多级放大电路分析。多级放大电路的分析思路：

- 前一级的输出电阻是后一级的电源内阻；
- 后一级的输入电阻是前一级的负载；
- 前一级的输出电压后一级的输入电压；
- 整体的增益是各级增益的乘积。

8. 差分放大电路主要概念：

- 差模输入（单出、双出、输入电阻）；
- 共模输入（单出、双出、输入电阻）；
- 不对称输入；
- 带有源负载的差分放大器。

9. 反馈技巧：

- 判断组态；
- 计算增益、反馈系数（注意是谁比谁）、闭环电压增益；
- 负反馈电路的设计、连接；
- 误差分析（失调电压、失调电流）。



D.4 数字电子技术

D.4.1 学习导论

数字电子技术（简称数电），是一项与电子计算机相伴相生的科学技术，它是指借助一定的设备将各种信息，包括：图、文、声、像等，转化为电子计算机能识别的二进制数字“0”和“1”后进行运算、加工、存储、传送、传播、还原的技术。主要内容包括数制与逻辑基础、门电路、组合逻辑电路、触发器、时序逻辑电路、脉冲的产生与整形、存储器和可编程逻辑器件、模拟量和数字量的转换等。

模拟电路主要处理模拟信号，不随时间变化。时间域和值域上均是连续的信号，如语音信号。而数字信号则相反，是变化的。数字信号的处理包括信号的采样，信号的量化，信号的编码。我们来举个例子，如果说想从远方传过来一段由小变大的声音，用调幅，模拟信号进行传输（相应的应采用模拟电路），那么在传输过程中的信号的幅度就会越来越大，因为它是在用电信号的幅度特性来模拟声音的强弱特性。但如果采用数字信号传输，就要采用一种编码，每一级声音大小对应一种编码，在声音输入端，每采一次样，就将对应的编码传输出去。与模电相比，数电电路抗干扰能力强，数字信号易存储。联系在于：模拟电路是为数字电路供给电源而又完成执行机构的执行；在模拟电路和数字电路中，信号的表达方式不同。对模拟信号能够执行的操作，例如放大，滤波，限幅等，都可以对数字信号进行操作。由此可以看出，所有的数字电路从根本上来说都是模拟电路，其基本电学原理，都与模拟电路相同。

D.4.2 基本知识

1. 逻辑代数与硬件描述语言基础
2. 组合逻辑电路

逻辑图的分析方法和步骤：

- 根据逻辑电路图逐级标出表达式，最后写出总的函数表达式
- 根据函数表达式列出真值表
- 根据真值表进行功能判断（加减乘除、挑选、奇偶校验等）

根据功能设计逻辑图：

- 根据要求确定输入、输出变量，然后列出符合要求的真值表
- 根据真值表得到相对简单的逻辑表达式
- 根据逻辑表达式画出逻辑电路图



3. 时序逻辑电路

同步电路的分析与设计

- 分析：1. 根据电路列出各逻辑方程组列出转换表或状态表 2. 画出状态图和时序图 3. 得到电路的逻辑功能
- 设计：1. 根据逻辑功能的需求 2. 推导原始状态图或原始状态表 3. 确定激励方程组和输出方程组 画出逻辑图
- 检查电路自启动以及输出信号是否正确

设计异步电路的波形图：1. 列出各逻辑方程组 2. 列出转换表、画出状态图
3. 画出波形图

4. 其他知识点（非全部）

- 逻辑门电路：CMOS 集成电路——超大、甚大规模集成电路；TTL 集成电路——中大规模集成电路（已淘汰）
- 锁存器和触发器：状态图、特性表、特性方程；各种触发器的功能和作用

D.5 信号与线性系统

信号与系统是很多基础课的基础课，诸如通信原理，数字信号处理，语音信号处理，数字图像处理，自适应滤波器理论，这些课想学会都必须以信号与系统学好为前提。信号与系统的应用不止这些，和香农的信息理论挂钩，它还可以用于信息处理（声音，图像），模式识别，智能控制等领域。如果说，计算机专业的课程是数据表达的逻辑模型，那么信号与系统建立的就是更底层的，代表了某种物理意义的数学模型。信号与系统更是作为广大院校 IT 类专业考研的必考科目。因此本科目知识体量非常庞大，难以速成。笔者囿于水平，在基本章节介绍的基础上又以问答方式提炼收集了一些重要概念，希冀起到抛砖引玉的作用，能激发大家自己去探索学习的欲望。

注：章节顺序安排以奥本海姆的书的顺序排布（强烈推荐此书）。

信号与系统简介

消息是信号传递的内容，信号是消息的物理载体（电，光，热等），数学上的体现就是函数，信息是消息中所包含的不确定性。所谓信号处理就是对表示物理量的函数进行处理，调制解调，变频，滤波等对函数的操作都算是信号处理。信息获取、变换、存储、传输、交换都离不开以信号为载体。信息处理主要位于系统的顶层，比如目标检测，人体跟踪都是在进行信息处理，就是对信号中的信息利用的过程。



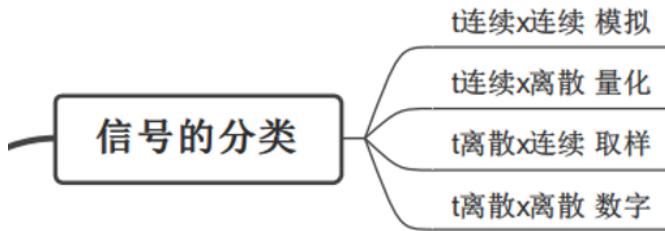


图 D.4: 信号分类

1. 信号的能量与功率

能量计算式:

$$E = \int_{-\infty}^{+\infty} |x(t)|^2 dt$$

$$E = \sum_{n=-\infty}^{+\infty} |x[n]|^2$$

当上述积分式和无穷级数不绝对收敛时, 我们认为信号的能量是无穷的, 有限能量的信号通常称为能量信号。

功率计算式:

$$P = \left(\frac{1}{2T} \int_{-T}^{T} |x(t)|^2 dt \right)$$

$$P = \left(\frac{1}{2N+1} \sum_{n=-N}^{N} |x[n]|^2 \right)$$

可以看到, 能量信号在无穷区间上的平均功率必然为零, 有限功率的信号记做功率信号。

在信号与系统涉及的范围内, 我们认为周期信号均是功率信号, 而非周期信号按幅度和持续时间是否无限可分为三类, 持续时间有限的都是能量信号, 持续时间无限而幅度有限的信号是功率信号。

在电路理论已经初步接触过冲激响应。冲激响应怎么理解? 冲激响应是系统在单位冲激信号的作用下而引起的零状态响应。一个信号可以分解为不同时延的冲激信号的叠加, 这样一个信号在经过一个系统处理之后就可以看做是很多不同时延的冲击信号经过了这个系统, 这样可以大大简化我们对系统响应的计算! 冲激信号实际上是一个理论模型, 并不能实际存在, 表示在瞬间激发出无穷大的脉冲。

因果系统就是输出不会超前如输入, 稳定系统就是输入有限输出也有限的系统, 线性系统当输入为输入的线性组合时, 输出为对应输出的线性组合。时不变系统指的是输出与输入作用时刻无关的系统。



2. LTI 系统

整个 LTI 系统实际上就是在讲卷积。卷积法的原理是将信号分解为冲击信号之和，然后借助系统的冲击响应求解系统对任意激励信号的零状态响应。什么是零状态响应、零输入响应和完全响应？在研究微分方程和差分方程的系统时，分清此二者非常重要。

零状态响应：是指我们的系统初始状态为零，仅有外部激励作用而产生的响应。从电路上看就是电路的储能元器件没有初始储能，仅有外部输入而产生的输出。零输入响应：是指系统没有输入仅有初始状态而产生的响应。完全响应就是上面两者的综合，系统有外部激励同时也有初始状态而产生响应。线性时不变系统的特性？

(Linear Time-Invariant) 叠加性和均匀性，时不变特性，微分特性。

3. 周期信号的傅里叶级数

课本是用了两章分别讲连续和离散的傅里叶，事实上它们几乎相同，反而是傅里叶级数和傅里叶变换之间的区别需要重点掌握，因此本文还是将连续和离散放在一起说，把级数和变换拆开。这一章要背的并不多，因为下一章的傅里叶变换会包括本章的内容，这里就重点理解为什么要引入傅里叶变换和感受这个过程即可。傅里叶变换与傅里叶级数关系在于：傅里叶变换是在傅里叶级数正交函数展开的基础上发展而产生的，这一方面的问题也被称为傅里叶分析。在分析周期信号时，可以用傅里叶级数也可以用傅里叶变换，傅里叶级数就相当于傅里叶变换的一种特殊表现形式。

吉伯斯现象了解一下即可。

4. 非周期信号的傅里叶变换

上节提到的傅里叶级数仅仅只能对周期信号进行分析，事实上很多信号都是非周期的。随着信号周期的增大，离散的谱线会越来越密集。我们就想到一种可能，非周期信号可以理解为周期无穷大的信号，而 T 增大到无穷时谱线会密集到成为连续函数，事实上这确实是正确的，也就是说，我们通过增大周期来逼近非周期信号，最后取极限得到最终非周期信号的结果。

周期非周期，连续离散，时域频域的关系：

- 时域周期必定频域离散（周期信号的傅里叶级数）；
- 时域非周期必定频域连续（非周期信号的傅里叶变换）；
- 时域离散必定频域周期（离散信号的变换有 2 周期）；
- 时域连续必定频域非周期（连续信号的变换都是无限区间无周期的）；

课本上的相关、能量谱、功率谱是随机信号的重点，小波变换是高级 DSP 的重点，DFT 和圆周卷积是 DSP 的重点，这些可以都先不看。



抽样定理——为了从抽样信号中回复原连续信号，需要什么条件？

首先模拟信号（时间幅值都连续）、抽样信号（时间离散、幅值连续）、数字信号（时间幅值均离散）。最低抽样频率为 $2f_m$ 。但为什么正弦波无法利用频率的 2 倍进行采样？答：角度一：在载频处发生了频谱混叠。角度二：相当于用正弦波对冲激串进行调制，只改变了冲激串的幅度，所以无法反推出原信号。

5. 频率响应

全通函数和最小相移函数：全通函数的极点零点左右对称，成镜像。全通网络可以保证不改变信号的幅度频域特性，只改变相位频域特性；最小相移函数零点仅位于左半平面。

影响频率分辨率的因素有哪些？答：两种定义：1. 区分两个谱峰的能力，用来评判不同算法性能；2. 频谱最小的频率间隔。影响因素：1. 信号有效长度 N 。 N 越大，频率分辨率越高。2. 所加的窗函数。不同的窗函数有不同的主瓣宽度。（简而言之：和信号截断的时刻以及方式有关）频率分辨率的理解：不改变频谱形状的变化都不能改变物理频率分辨率。（补零，有效时间 T 不变，所以不改变频率分辨率）

6. 拉普拉斯变换

当复指数不再限于纯虚数时，即 $s = \sigma + j\omega$ 时，就由傅里叶变换变成了拉普拉斯变换，也就是说傅里叶变换是拉普拉斯变换的一个特殊情况。

那么为什么要引申到拉普拉斯变换呢，这是因为狄利克雷条件的绝对可积几乎只有衰减信号才能满足，而引入拉普拉斯变换之后，复数 s 的实部 σ 乘在信号上可能会使原本不绝对可积的信号满足条件从而可以变换，扩大了变换的适用范围。

拉氏变换的优点及傅氏变换的区别：拉氏变换将微分与积分运算转换成了乘法和除法运算，把微分方程转换成了代数方程，计算起来方便；拉氏变换对于处理指数函数，超越函数以及由不连续点的函数有很大便利。拉氏变换与傅氏变换的基本区别：傅氏变换是将时域信号转换到频域或者进行相反的变换，时域变量或者频域变量都是实数；拉氏变换是将时域信号转换到复频域或者进行相反变换，时域变量是实数，但是复频域变量是复数。简单来说，傅氏变换建立的是时域和频域的关系，而拉氏变换建立的是时域和复频域的关系。

双边拉氏变换的优点：信号不必局限在 $t > 0$ 的范围内，双边拉氏变换与傅里叶变换的联系更紧密，准确来说，双边拉氏变换是广义的傅里叶变换。当然，由于实际中的信号都是在 $t > 0$ 时才出现，因此下面的单边拉普拉斯变



换式更加常用。

注意：拉氏变换的收敛域非常重要，同一个拉普拉斯变换 $X(s)$ ，收敛域不同，其对应的时域函数 $x(t)$ 就不同。收敛域满足如下几条规则

- 收敛域必然是平行于虚轴的带状区域
- 若 $X(s)$ 有理则收敛域不包括任何极点
- 若 $x(t)$ 有限时长或绝对可积，则在整个 s 平面上收敛
- 右边信号 (t 从某一值延伸到正无穷) 的收敛域为最右极点的右侧；左边信号 (t 从某一值延伸到负无穷) 的收敛域为最左极点的左侧；双边信号收敛域不延伸到无穷，而是被某两个极点限制。

7. Z 变换

与连续傅里叶变换扩展为拉普拉斯变换类似，离散的也可以扩展为 Z 变换，其意义和拉普拉斯变换差不多。与离散时间傅里叶变换的关系。显然当 $z = e^{j\omega}$ 时 Z 变换成为傅里叶变换，也就是 z 平面上半径为 1 的单位圆，也就是说，相比于拉普拉斯变换是在纯虚数的基础上添加了实部，Z 变换是在单位模值复数的基础上乘了倍。注意：收敛域同样非常重要，也会导致时域函数不同。

什么是特征函数？（联系线代学的知识：什么是特征值，特征向量）答：特征函数其实有两种定义。一般指的是在《信号与系统》中所学的那个定义，即特征函数通过一个线性系统，输出为某一常数乘以该特征函数，该常数称作特征值。对于傅里叶变换，拉式变换，Z 变换而言，它们的特征函数就是变换的基。

D.6 回顾与展望

看完上面的章节，你对基础的硬件知识有了一定的了解，然后硬件时需要集成的，下面给出了推荐的学习路径和学习知识。

1) 按顺序，了解上面四节的知识，如果您学有余力，可以考虑在大二上学期结束的时候全部了解，当然，大二下学习再了解也不迟。2) 学习基本的软件知识，对于上层操控硬件的软件，了解 C 语言和汇编语言的语法就足够了。

3) 在大三的时候选择一块单片机尝试去进行开发，对单片机进行控制可以帮助你更好地组合之前的电路知识，并了解集成电路的特性和结构。

在了解了基础的知识之后，你可以尝试阅读相关的书籍增进了解。

相关书籍：

1. 基础理论知识：



- 电路分析基础（李瀚荪），这本书是教材，对于底子薄的同学，还是建议先看看的，主要是一些电路分析的定理、转换方式等。
- 像微机原理、线性电子电路、非线性电子电路、高频电子线路这些教材都建议看看。

2. 模拟电路设计：

- 教材推荐《模拟电子技术基础》（华成英）。
- 模拟电路相比较数字电路，有点晦涩难懂，除了国内的教材外，也推荐读一读国外的一些书籍，如《晶体管电路设计》（铃木雅臣），《你好，放大器》（杨建国）。

3. 数字电路设计：

- 《数字电子技术基础》（第六版，阎石），数字电路基础是电子专业必修课，高校教材内容也都是大差不差的，也可以看华成英的第五版。
- 如果是走 FPGA/CPLD 路线，需要学 Verilog 语言，可以看《Verilog 数字系统设计教程》（夏宇闻）。

4. 电源设计：《精通开关电源设计》（Sanjaya Maniktala 著，王健强等译）。





第 E 章 ➤ 其他计算机相关知识

E.1 Linux 系统目录结构

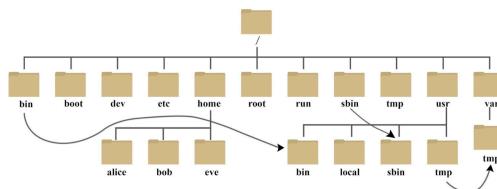


图 E.1

一般来说就是由这几个部分组成的。当出现系统上的问题的时候，可以参考进行 debug

- **/bin:**

bin 是 Binaries (二进制文件) 的缩写，这个目录存放着最经常使用的命令。比如说很多个 Shell 的程序都会放在/bin 文件夹里面。

- **/boot:**

这里存放的是启动 Linux 时使用的一些核心文件，包括一些连接文件以及镜像文件。

- **/dev :**

dev 是 Device(设备) 的缩写，该目录下存放的是 Linux 的外部设备，在 Linux 中访问设备的方式和访问文件的方式是相同的。

- **/etc:**

etc 是 Etcetera(等等) 的缩写，这个目录用来存放所有的系统管理所需要的配

置文件和子目录。

- **/home:**

用户的主目录，在 Linux 中，每个用户都有一个自己的目录，一般该目录名是以用户的账号命名的，如上图中的 alice、bob 和 eve。

- **/lib:**

lib 是 Library(库) 的缩写这个目录里存放着系统最基本的动态连接共享库，其作用类似于 Windows 里的 DLL 文件。几乎所有的应用程序都需要用到这些共享库。

- **/lost+found:**

这个目录一般情况下是空的，当系统非法关机后，这里就存放了一些文件。

- **/media:**

linux 系统会自动识别一些设备，例如 U 盘、光驱等等，当识别后，Linux 会把识别的设备挂载到这个目录下。

- **/mnt:**

系统提供该目录是为了让用户临时挂载别的文件系统的，我们可以将光驱挂载在 /mnt/ 上，然后进入该目录就可以查看光驱里的内容了。

- **/opt:**

opt 是 optional(可选) 的缩写，这是给主机额外安装软件所摆放的目录。比如你安装一个 ORACLE 数据库则就可以放到这个目录下。默认是空的。

- **/proc:**

proc 是 Processes(进程) 的缩写，/proc 是一种伪文件系统（也即虚拟文件系统），存储的是当前内核运行状态的一系列特殊文件，这个目录是一个虚拟的目录，它是系统内存的映射，我们可以通过直接访问这个目录来获取系统信息。

- **/root:**

该目录为系统管理员，也称作超级权限者的用户主目录。

- **/sbin:**

s 就是 Super User 的意思，是 Superuser Binaries (超级用户的二进制文件) 的缩写，这里存放的是系统管理员使用的系统管理程序。

- **/selinux:**

这个目录是 Redhat/CentOS 所特有的目录，Selinux 是一个安全机制，类似于 windows 的防火墙，但是这套机制比较复杂，这个目录就是存放 selinux 相关的文件的。

- **/srv:**



该目录存放一些服务启动之后需要提取的数据。

- **/sys:**

这是 Linux2.6 内核的一个很大的变化。该目录下安装了 2.6 内核中新出现的一个文件系统 sysfs。

sysfs 文件系统集成了下面 3 种文件系统的信息：针对进程信息的 proc 文件系统、针对设备的 devfs 文件系统以及针对伪终端的 devpts 文件系统。

该文件系统是内核设备树的一个直观反映。

当一个内核对象被创建的时候，对应的文件和目录也在内核对象子系统中被创建。

- **/tmp:**

tmp 是 temporary(临时) 的缩写这个目录是用来存放一些临时文件的。

- **/usr:**

usr 是 unix shared resources(共享资源) 的缩写，这是一个非常重要的目录，用户的很多应用程序和文件都放在这个目录下，类似于 windows 下的 program files 目录。

- **/usr/bin:**

系统用户使用的应用程序。

- **/usr/sbin:**

超级用户使用的比较高级的管理程序和系统守护程序。

- **/usr/src:**

内核源代码默认的放置目录。

- **/var:**

var 是 variable(变量) 的缩写，这个目录中存放着在不断扩充着的东西，我们习惯将那些经常被修改的目录放在这个目录下。包括各种日志文件。

- **/run:**

是一个临时文件系统，存储系统启动以来的信息。当系统重启时，这个目录下的文件应该被删掉或清除。如果你的系统上有 /var/run 目录，应该让它指向 run。

很多个文件夹都不能随意更改:/etc,/bin,/sbin,/usr/bin, 这里面存放着配置文件和一些预设的执行文件的目录。



E.2 Shell 语言运算符

E.2.1 算术运算符

下表列出了常用的算术运算符，假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
+	加法	expr \$a + \$b 结果为 30。
-	减法	expr \$a - \$b 结果为 -10。
*	乘法	expr \$a * \$b 结果为 200。
/	除法	expr \$b / \$a 结果为 2。
%	取余	expr \$b % \$a 结果为 0。
=	赋值	a=\$b 把变量 b 的值赋给 a。
==	相等。用于比较两个数字，相同则返回 true。	[a ==b] 返回 false。
!=	不相等。用于比较两个数字，不相同则返回 true。	[a!=b] 返回 true。

注意：条件表达式要放在方括号之间，并且要有空格，例如：[\$a==\$b] 是错误的，必须写成 [\$a == \$b]。

E.2.2 关系运算符

下表列出了常用的关系运算符，假定变量 a 为 10，变量 b 为 20：

(T = true, F = false)

运算符	说明	举例
-eq	检测两个数是否相等，相等返回 T。	[\$a -eq \$b] 返回 F。
-ne	检测两个数是否不相等，不相等返回 T。	[\$a -ne \$b] 返回 T。
-gt	检测左边的数是否大于右边的，如果是，则返回 T。	[\$a -gt \$b] 返回 F。
-lt	检测左边的数是否小于右边的，如果是，则返回 T。	[\$a -lt \$b] 返回 T。



运算符	说明	举例
-ge	检测左边的数是否大于等于右边的，如果是，则返回 T。 [\$a -ge \$b] 返回 F。	
-le	检测左边数是否小于等于右边的，如果是，则返回 T。 [\$a -le \$b] 返回 T。	

E.2.3 Boolean 运算符

下表列出了常用的布尔运算符，假定变量 a 为 10，变量 b 为 20：
(T = true, F = false)

运算符	说明	举例
!	非运算，表达式为 T 则返回 F，否则返回 T。 [! F] 返回 T。	
-o	或运算，有一个表达式为 T 则返回 T。 [a -lt 20 -o b -gt 100] 返回 T。	
-a	与运算，两个表达式都为 T 才返回 T。 [a -lt 20 -a b -gt 100] 返回 F。	

E.2.4 字符串运算符

下表列出了常用的字符串运算符，假定变量 a 为"abc"，变量 b 为"efg"：

运算符	说明	举例
=	检测两个字符串是否相等，相等返回 true。 [a =b] 返回 false。	
!=	检测两个字符串是否不相等，不相等返回 true。 [a! =b] 返回 true。	
-z	检测字符串长度是否为 0，为 0 返回 true。 [-z \$a] 返回 false。	
-n	检测字符串长度是否不为 0，不为 0 返回 true。 [-n "\$a"] 返回 true。	
\$	检测字符串是否为空，不为空返回 true。 [\$a] 返回 true。	

E.2.5 文件测试运算符

属性检测描述如下：其中 file 参数是一个字符串，但是存储了文件的路径



操作符 说明

-
- b file 检测文件是否是块设备文件，如果是，则返回 true。
 - c file 检测文件是否是字符设备文件，如果是，则返回 true。
 - d file 检测文件是否是目录，如果是，则返回 true。
 - f file 检测文件是否是普通文件(既不是目录，也不是设备文件)，如果是，则返回 true。
 - g file 检测文件是否设置了 SGID 位，如果是，则返回 true。
 - k file 检测文件是否设置了粘着位 (Sticky Bit)，如果是，则返回 true。
 - p file 检测文件是否是有名管道，如果是，则返回 true。
 - u file 检测文件是否设置了 SUID 位，如果是，则返回 true。
 - r file 检测文件是否可读，如果是，则返回 true。
 - w file 检测文件是否可写，如果是，则返回 true。
 - x file 检测文件是否可执行，如果是，则返回 true。
 - s file 检测文件是否为空（文件大小是否大于 0），不为空返回 true。
 - e file 检测文件（包括目录）是否存在，如果是，则返回 true。
-





第 F 章 ➤ 数学部分答案

问题 F.1 $\frac{10!}{5! \times 3! \times 2!} = 2520.$

问题 F.2 (i) $C(10 + 5 - 1, 5 - 1) = 1001.$

(ii) $C(10 + 5 - 1 - 5, 5 - 1) = 126.$

问题 F.3 (i) 我们设函数的定义域是那 6 位妖精，值域是四种项目，函数的映射关系是 A 的成绩是 B ，由于四种项目都会有人，所以说这个函数是满射，自然不可能是单射。

(ii) 我们利用容斥定理进行分析，首先 6 位妖精随机分配成绩，可能性有 4^6 种，接着是仅有三种项目有人，有一种项目一个人都没有，情况有 $3^6 \times 4$ 种，接着就是仅有项目有人，有两种项目一个人都没有，情况有 $2^6 \times 6$ 种，接着就是仅有项目有人，有三种项目一个人都没有，有 4 种情况，则根据容斥定理有 1560 种可能。

问题 F.4 15.

问题 F.5 $f(k) = \left\lfloor \frac{N}{k} \right\rfloor^2 - \sum_{i=2}^{i*k \leq N} f(i*k).$

问题 F.6 $C(2n, n) - C(2n, n - 1).$

问题 F.7 略。

问题 F.8 不可能成立：

构建图模型 $G(V, E)$ 其中边代表朋友关系，点代表人，每个人只和 3 个人做好朋友代表每个点的度为 3，一共有 5 个人，度的总和为 15，违反握手定则。

问题 F.9 因为任何一个点覆盖集的补集都是点独立集，一个点覆盖集可以推出

一个点独立集所以说不同的点独立集的数量等于不同的点覆盖集的数量.

问题 F.10 构建图模型 $G(V, E)$ 其中边代表药品不能放在一块，点代表药品，极大独立点集的是可以放在一起的.

问题 F.11 (i) 边覆盖数：最小的边覆盖所含的边数称为边覆盖数.

边独立数（也称匹配数）：最大匹配的边数称为边独立数或匹配数.

(ii) 设最大匹配数为 m , 总顶点数为 n 为了使边数最少, 又因为一条边最多能干掉两个点, 所以尽量用边干掉两个点. 也就是取有匹配的那些边, 当然这些边是越多越好, 那就是最大匹配了, 所以先用最大匹配数目的边干掉大多数点. 剩下的解决没有被匹配的点, 就只能一条边干掉一个点了, 设这些数目为 a , 显然, $2m + a = n$, 而最小边覆盖 = $m + a$, 所以最小边覆盖 = $(2m + a) - m = n - m$.

问题 F.12 略.

问题 F.13 构造一个 $2n$ 个点的二分图, 设左边的点集是 $[1, n]$, 右边点集 $[n+1, 2n]$, 对原图中的每一条边 (x, y) 在新图中加入 $(x, n+y)$.

问题 F.14 构造一个 $2n$ 个点的二分图, 设左边的点集是 $[1, n]$, 右边点集 $[n+1, 2n]$, 对原图中的每一条边 (x, y) 在新图中加入 $(x, n+y)$. 并对所有的 $i \in [1, n]$, 加入边 $(i, i+n)$.

LSC: 略.

问题 F.15 现在写出式子 $a_n = a_{n-1} + a_{n-2}$, 并且有 $a_0 = 0, a_1 = 1$.

则有 $F(x) = xF(x) + x^2F(x) - a_0x + a_1x + a_0$, 解得:

$$F(x) = \frac{x}{1-x-x^2}$$

这个时候可以拆成: 拆的方法就是令分母 = 0, 求出来两个解 x_1, x_2 , 然后表示成 $\frac{A}{1-x_1} + \frac{B}{1-x_2}$, 利用待定系数法求出 A 和 B . 因为 Taylor 展开, 我们可以把这个封闭形式转化成一般形式.

$$\frac{x}{1-x-x^2} = \sum_{n \geq 0} x^n \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

$$\text{这个时候可以知道 } a_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right).$$

