编程技巧

陈元昊

2022 年 10 月 23 日

目录

1 Linux

1.1 指令技巧

1.1.1 man

1. 通过执行man man命令可以查看man的用法，常见的用法有当因用户命令和C库函数重名而无法看到C库函数的用法时，设名称为x可以通过man 2 x来查看名为x的库函数的用法

1.1.2 grep

1. 加-E选项后，可以在使用正则匹配时不用给括号转义

1.1.3 gcc/g++

1. 加-E选项后仅执行到预处理，文件后缀.i
2. 加-S选项后仅执行到编译，文件后缀.s
3. 加-c选项后仅执行到汇编，文件后缀.o

1.1.4 git

1. 当因为token的原因（一般存在于报错）无法clone时，可尝试设置一个在网站上设置一个具有权限的token，复制之，然后在Windows凭据管理器上新建/修改一个普通凭据，注意密码应为token

1.1.5 strace

1. strace用于跟踪并输出程序执行时进行的系统调用

1.1.6 time

1. 可使用time命令计算程序的用户态和内核态运行时间

1.1.7 nslookup

1. nslookup通过访问DNS服务器查询域名对应的IP地址

1.1.8 telnet

1. telnet命令用于向某一地址+端口请求连接并可以发送请求，可用于网络测试

1.1.9 重定向

1. 使用重定向输入输出时可能会遇到”
   r
   n”和”
   n”的问题，利用vscode右下角行尾序列，更改即可

1.2 基本概念

1.2.1 进程管理

1. 挂起，一般通过按ctrl+z实现，效果为暂停执行（前台或后台程序均可以），但可用fg或bg恢复执行
2. 后台运行，一般通过在命令行末尾加“&”符号实现，也可以通过挂起+后台恢复间接实现，效果为以不占用终端的方式运行
3. 由于后台运行不能让shell以阻塞方式等待，所以不能直接用waitpid的方式等待，而是要先使用信号通知shell某个子进程的结束，再进行waitpid
4. 单纯用户态和内核态之间的切换不一定涉及上下文切换，所做的工作可能只是将寄存器保存在内核栈中以及其他关于状态（用户态/内核态）、程序计数器、栈指针的调整
5. 可重入=线程安全+信号中断安全
6. 多级页表是一种链表+向量的数据结构，借此实现高速的访问和内存空间的节省

1.2.2 ECF

1. Exception分为Interrupt（async）、Trap（sync）、Fault（sync）、Abort（sync）
2. Signal是软件层级的ECF，位于软件层面，用于向进程发送通知
3. 信号处理程序中不能使用printf，原因是printf在更改缓冲区时会加锁，若主程序调用printf时，控制权离开并返回主程序，且返回主程序时发现有pending且非blocked的信号，进入信号处理程序时也调用printf，则信号程序中的printf由于主程序的printf的锁不得不等待，又由于主程序和信号处理程序处在同一个进程/线程中，因此主程序printf的锁总是无法解除，从而导致死锁
4. 每个线程有自己独享的信号处理

1.2.3 并发与并行

1. 在多线程中经常有加锁与解锁操作来保证二者之间操作的原子性，由于对于同一个锁，在任一时刻P操作数量多于V操作数量，不会有连续的两个P操作，由数学归纳法可知两种一定为形如“PVPVPVPV…”的序列
2. 由于多线程、cache机制与多核同时出现会造成cache不一致的问题，所以需要缓存一致性协议，但是要做到这一点较为复杂，这也是限制核心数量的原因之一

1.3 WSL2

1.3.1 网络

1. 宿主机可以用127.0.0.1访问WSL2，反之则不行
2. 当代理软件（Clash）位于Windows上时，Windows配置代理仅需要set http(s)_proxy=”127.0.0.1:7890”，而WSL2在使用export http(s)_proxy=”宿主机IP:7890”之前，要先用cat /etc/resolv.conf — grep nameserver — awk ’ print $2 ’获取宿主机IP（此外，用hostname -I — awk ’print $1’获取WSL2自身IP）

1.3.2 安全

1. 从宿主机复制而来的文件有时无法作为重定向输入输出文件被使用

2 C/C++

2.1 算法

2.1.1 栈

1. 栈混洗数为卡特兰数，通项公式为$C_{2n}^{n}-C_{2n}^{n-1}$，可由“不合法”的push+pop序列与push比pop数量多2的全排列一一对应这一事实得出（对应方法为“不合法”序列一定有第一次pop比push多1的位置，将这一位置之前push和pop翻转，则push比pop数量多2，而反之任一这样的全排列均可转换回去），递推公式为$SP(n)=\sum_{i=0}^{n-1}SP(i)SP(n-1-i)$，可考虑栈顶元素入栈的位次，由于栈顶元素入栈时中转栈必为空，之前入栈和之后入栈的元素次序彼此独立，可得递推公式

2.1.2 预处理

1. 二分前先使数组有序
2. 注意隐藏边界（长度为0,1等）
3. 先排序再计算往往可以简化计算过程
4. 有可能样例输入有序，测试点输入无序
5. 注意图的输入中的重边和自环，以及有向输入转化为无向图

2.1.3 算法执行

1. 二分区间的开闭由具体问题决定（一般一边开一边闭）
2. 递归算法需要数组记录答案时可以不用“触底”时全部修改，然后利用一个全局的bool变量连续退出，而是可以回溯时逐步修改，从而减小代码复杂度
3. 注意浮点数计算的上下浮动
4. 尝试将$n!$（排列）转化为2^n（组合）
5. 注意在寻找最高位非0数（常见于高精度或多项式等问题）时要考虑全为0导致循环变量等于-1这一corner case

2.1.4 算法评估

1. 计算递归算法复杂度可先计算递归实例的数量

2.2 实用函数

2.2.1 输入输出

1. 可以使用freopen函数进行输入输出重定向（但可能不如命令行加重定向符号好用）

2.2.2 排序

1. qsort在数据量较小时采用插入排序，数据量中等时采用归并排序，在数据量较大时采用快速排序，且快速排序采用三数取中法，时间上较为稳定

2.3 语法特性

2.3.1 面向过程

1. 尝试使用cassert头文件，用其中的assert宏进行运行时断言
2. 注意循环嵌套中，循环变量i、j、k等不要重复使用
3. 循环体中的变量地址不变
4. 使用getchar前注意去除cin等留下的回车等干扰字符
5. 注意数组下标越界有可能完全无异常（越在其他变量内部）
6. switch分支结构注意用break
7. 在C++中，不同的函数、结构体、类可以声明同名的静态变量，彼此独立
8. 只有第一次进入函数时静态变量会初始化，之后进入会忽略初始化语句，故静态函数调用计数可利用函数体中的局部静态变量
9. 在C中，结构体和联合不会引入新的变量作用域，不能声明静态成员变量
10. 可以使用位域直接操作内存中的位

2.3.2 面向对象

1. 对象内部局部变量需要初始化
2. 注意写public（默认为private）
3. 友元函数函数不是成员函数，不能加作用域符号
4. 引用本质只是别名，其创建时不会产生任何构造过程
5. 注意避免自身赋值
6. 当一个内部类或内部对象需要访问外部对象时，尽量通过外部对象成员变量的指针来访问，否则有可能出现构造顺序或访问权限的问题
7. 尽量不要创建野指针，如果不可避免要创建野指针，一定要初始化为nullptr
8. 移动构造、赋值前注意删除当前指针的内容，避免当前指针赋新值后内存泄漏
9. delete前对象最好指针最好不是nullptr，delete后对象指针最好置为nullptr
10. 在返回值和参数均可被析构时，先析构返回值，再析构参数（符合栈的顺序）
11. 静态成员变量要在main函数前进行初始化
12. 虚函数/常量函数不能为静态函数，因为其调用/参数中需要/含有this指针
13. 模板函数将成员函数作为形参时，成员函数应设为静态函数，非静态成员函数因为有this指针形参，参数数量不一致，可能导致错误（sort）
14. std::move()对常引用无效
15. 派生类新定义的非虚函数和新定义的变量会在函数形参为值/引用/指针（所有情况）时被切片
16. 重写函数调用时，与所有当前形式类中的函数同名且参数不同的函数会被隐藏，然后按虚函数机制调用

17. 在派生类没有直接写出新函数的情况下，派生类不会自动生成新的虚函数继承版本，而是在虚函数表中沿用旧版本（注意与重写隐藏的关系）

18. 基类指针指向派生类对象时，调用被基类声明、派生类继承的虚函数不需要dynamic_cast，调用派生类声明的函数需要dynamic_cast

19. 模板的声明与实现需要在同一文件中（模板实例化在编译期确定）

2.4 底层机制

2.4.1 多进程/多线程

1. volatile（易变）关键字影响编译器的编译，使得即使进行了优化，每次访问此变量时都会重新从内存取值，从而避免信号处理/其他线程在不经意间修改此变量造成的数据不一致的问题

2.5 Qt

2.5.1 绘图

1. Qt画圆的坐标原点为外界矩形的左上角点

3 Python

3.1 语言规范

3.1.1 程序结构

1. 引用原生库和手写库
2. 定义全局变量
3. 定义修饰器
4. 定义类（包括函数对象）
5. 定义函数（包括argparse）
6. 定义主函数

3.2 语言特点

3.2.1 易错点

1. 不要将python源码文件命名为包的名字，否则会因重复包含导致错误

3.2.2 运行特点

1. 在使用import时正确的路径是针对main.py而言的，而不是针对当前文件而言的
2. global关键字的使用是为了在局部作用域中引用并修改全局变量
3. 闭包函数若要修改上级作用域中的变量，需要用nonlocal关键字
4. Python中只有模块(module)，类(class)以及函数(def、lambda)才会引入新的作用域，其它的代码块(如if、try、for等)不会引入新的作用域，因此在代码块外部可以直接引用代码块内声明的变量
5. Python的变量是动态声明的，未考虑到这点可能出现bug，例如在if语句中声明了变量，若该if语句条件为假，则不仅其内部语句不执行，其内部变量也不会被定义。因此，很多时候有必要在if语句之前声明变量

3.3 具体应用

3.3.1 正则表达式

1. 在?、+、*以及{n,m}后加?表示进行懒惰匹配（与默认的贪婪匹配相反）
2. \b、$以及^匹配的是单词边界，而非字符（匹配的是“一条线”）

4 Javascript

4.1 语言特性

4.1.1 函数

1. 闭包中的作用于整个函数的变量为引用，而在某个循环内部的变量为拷贝

4.1.2 对象与原型

1. this永远指向最近的调用者

5 Machine Learning

5.1 Linear Regression

5.1.1 技巧

1. 特征归一化可以使不同特征的范围较为接近，从而加快收敛速度

5.2 logistic regression

5.2.1 作用

1. logistic函数即$f(x)=\frac{1}{1+e^{-x}}$，可以将自变量映射到(0,1)区间内，常用于聚类算法

5.2.2 损失函数

1. 一般使用交叉熵函数
2. 以$f(\vec{x})=\frac{1}{1+e^{-\vec{\theta}^T\vec{x}}}$作为模型时，交叉熵函数相比平方平均函数作为损失函数的最外层有两点优势，其一为不含logistic函数的导数项，可以保证较快的收敛速度，其二为可以保证损失函数永远处处为凸（保证Hessian矩阵一定处处半正定）