英文：学过os? computer network? Data strcture?都是干什么的？

Big O和small O

Big O mean less than or equal to，small o mean less than。

Pcb: progress control block

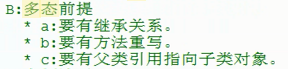
算法：algorithm [ˈælɡəˌrɪðəm]

PCB: PCB(进程控制块process control block): 一个数据结构，进程存在的唯一标志. 系统通过PCB来了解进程的状态信息,以便控制和管理

事务：指一个单元的工作，要么全做，要么全不做，保证一组数据的修改要么全部执行，要么全部不执行

事务的特点： ACID: 原子性（要么都做，要么都不做），一致性(运行中发生故障，必须回滚)，隔离性(一个事务不能被其他事务干扰)，持续性（事务一旦提交，对数据库的改变应该是永久性的）.原隔一尺（持）

多态: 接口的多种不同的实现方式. (允许将子类类型指针赋值给父类类型指针)



继承: 一个类继承另一个类的功能, 可以直接使用其属性和方法.并可以添加

Java单继承, c++可以多继承(多个爹)

封装: 隐藏对象的属性和细节, 仅对外提供公共访问方式.(控制属性的读和修改的访问级别)

虚函数：在某[基类](https://baike.baidu.com/item/%E5%9F%BA%E7%B1%BB)中声明为 virtual, 允许被一个或多个[派生类](https://baike.baidu.com/item/%E6%B4%BE%E7%94%9F%E7%B1%BB)中重写的[成员函数](https://baike.baidu.com/item/%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0)，以实现多态。

  Virtual 函数类型 函数名

纯虚函数：在基类中声明的虚函数，在基类中没有定义，但要求任何派生类都要定义自己的实现方法。在基类中实现纯虚数的方式是在函数原型后面加“=0”。

包含纯虚数的类是抽象类。

指令是内核态的多还是用户态的多？

内核态，CPU可以访问内存所有数据，用户态下是受限访问，只能访问自己空间中的内存。

**指针加一：对指针增加该**指针类型所占的内存的字节数。不同类型的指针加1后，增加的大小不同。

FAT: 文件配置表file allocation table: 分配给文件的所有盘块号都放在该表中，记录了文件所在位置。

文件系统：负责管理和存储文件信息的软件机构，由三部分组成：文件系统的接口，对对象操纵和管理的软件集合，对象及属性

文件系统类型： Windows中有FAT32，NTFS，Linus中有ext2,ext3,ext4

括号匹配：在很多字符串处理的场景中时常被用到，编译器编译时检查应该成对出现的括号是否符合要求等

栈应用 括号匹配

设置一个栈，每读入一个左括号，则将该括号压入栈内；如果读到右括号，则与栈顶的左括号进行匹配检验，若成功，则栈弹出一个元素，否则整个式子匹配失败，结束。

算法思想：设置一个栈，当读到左括号时，左括号进栈。当读到右括号时，则从栈中弹出一个元素，与读到的左括号进行匹配，若匹配成功，继续读入；否则匹配失败，返回FLASE。另外,在算法的开始和结束时,栈都应该是空的.所以匹配到最后还要判断栈是否为空,若非空,则说明匹配失败.

进程和线程(thread)的区别

线程是独立调度的基本单位, 本身没有资源, 进程是资源分配的基本单位, 同一进程中的线程共享进程的地址空间(放着各种资源: 进程代码段, 全局变量, 打开的文件), 但是, 一个线程的栈指针对其他线程透明(心里的客栈, 只对自己开放)

进程为什么是资源分配的基本单位？进程是一个程序对某个数据集的一次运行活动，创建进程的时候要分配CPU和内存等资源。

OSI/RM开放系统互连参考模型的全称  
Open System Interconnection Reference Model.

OSI七层

物理层physical layer，数据链路层data link layer，网络层network layer，传输层transport layer，会话层session layer，表示层presentation layer，应用层application layer

二叉排序树转换成排序的双向链表

用递归的方法。二叉排序树特点一个结点的左子树比它小，右子树比它大，中序遍历得到一个递增的序列。

在中序遍历时，让指向左子树的指针变为链表中指向前一个结点的指针，指向右子树的指针变为链表中指向后一个结点的指针。

内存交换(对换)/对换(交换)：多个进程之间

把处于等待状态的程序从内存中移出, 把准备好竞争CPU的程序移入内存

交换区。

提高并行：多道程序设计

中断：解决处理器速度和硬件速度不匹配，是多道程序设计的必要条件。每个中断都有自己的数字标识，当中断发生时，指令计数器PC和处理机状态字PSW中的内容自动压入处理器堆栈，同时新的PC和PSW的中断向量也装入各自的寄存器中。这时，PC中包含的是该中断的中断处理程序的入口地址，它控制程序转向相应的处理，当中断处理程序执行完毕，该程序的最后一条iret（中断返回），它控制着恢复调用程序的环境。

中断和系统调用的区别:

中断是由外设产生, 无意的, 被动的

系统调用是由应用程序请求操作系统提供服务产生, 有意的, 主动的。要从用户态通过中断进入内核态。（联系）

中断过程：中断请求 中断响应 断点保护 执行中断服务程序 断点恢复 中断返回

系统调用过程：应用程序在用户态执行时请求系统调用，中断，从用户态进入内核态，在内核态执行相应的内核代码。

中断之后保存什么？保存pc, psw, 通用寄存器。Pc程序计数器program count，存放下一条指令所在的单元的地址。Psw，program status word程序状态字，指处理器的状态。

二叉树：

先序二叉树：若二叉树为空，什么也不做，否则，访问根结点，然后先序遍历左子树，最后先序遍历右子树。

中序二叉树：若二叉树为空，什么也不做，否则，中序遍历左子树，然后访问根结点，最后中序遍历右子树。

后序二叉树: 若二叉树为空，什么也不做，否则，后序遍历左子树，后序遍历右子树，最后访问根结点。

先中后序，递归，用栈实现，因为每个结点都只访问一次，时间复杂度O(n)，递归工作栈的深度恰好是树的深度，最坏情况下，n结点，n层，时间复杂度O（层数）。

层次遍历：若二叉树为空，什么也不做，否则，访问根结点，然后逐层遍历。

用队列。

先+中，后+中，层+中序遍历都可以唯一确定一棵二叉树。

平衡二叉树(AVL树)：树上任一结点的左子树和右子树的深度之差不超过1。

栈stack [stæk] 堆heap [hip]

The character [ˈkærəktɚ]of stack is first in last out. we can insert [ɪnˈsɜrt] or delete [diˈlit] elements [ˈɛləmənts] in the same direction.

The character of heap is first in first out. We can insert elements in one direction, but delete delements in the other direction.

野指针，指向不可用内存区域的指针。指针变量未初始化（会乱指，可能存储关键信息，一下子就比修改了），指针释放后未置空，指针操作超越变量作用域造成。对野指针进行操作很容易造成程序错误**，甚至可能直接引起崩溃。**

野指针与空指针不同，野指针无法通过简单地判断是否为[NULL](http://baike.baidu.com/view/329484.htm)避免.

如何避免？指针变量初始化置null，指针释放后置null.

饥饿与死锁：

死锁:，这种多个进程因竞争资源而造成的僵局，两个或多个进程因无限等待一个事件, 而该事件只能由这些等待进程之一来产生若如外力，进程都无法向前推进。

饥饿: 进程等待永远不会分配给自己的资源.

相同点: 都是因为竞争资源引起的

不同点: 死锁至少涉及两个进程, 一定有循环等待, 且等待永远不会释放的资源（好几个在环里不释放）

饥饿可以只涉及一个进程, 不一定有循环等待, 等待会释放但不会给自己的资源.

给你一组N个数字(比如 vector<int> num), 然后给你一个常数(比如 int target) ，我们的goal是在这一堆数里面找到K个数字，使得这K个数字的和等于target。

2sum: 先对数组中的数进行排序，再设置两个指针，一个指向头，一个指向尾。判断两数和是否等于想要的数，如果是则在结果集添加这个数组；如果小了说明左边指针指向的数小了，因此左指针右移；反之如果大了则右指针左移。

2sum时间复杂度是O(nlogn)(排序)，因为头尾指针线性扫描，只需要O(n)就可以了

3sum: 同样先对数组排序，设置三个指针p,q,r，p指针指向第一个数x，则q,r要指向数组中剩余数中的两个，并且指向的两数和为-x，从而转化为两数和问题。对p指向第一个数的情况分析完毕后，不可能再有满足题意且包含x的情况，于是p右移。这样一直分析到p指向数组中倒数第三个数的情况。

3sum时间复杂度O（n^2），排序(nlogn)

最外层遍历一遍，等于选出一个数， 之后的数组中转化为找和为target-nums[i]的2SUM问题。

时间复杂度

(1)Θ（西塔）：紧确界。相当于"="

(2)O （大欧）：上界。相当于"<="

欧，牛，达到最大了

(3)o（小欧）：非紧的上界。相当于"<"

(4)Ω（大欧米伽）：下界。相当于">="

(5)ω（小欧米伽）：非紧的下界。 相当于">"

折半插入：O(lon2n)

折半查找： O（n^2）因为同一往后移要O（n）

打开open文件：系统调用open，指明文件名，通过索引找到指定文件，将文件的属性（包括在外存的物理地址）从外存拷贝到内存打开文件表中，并返回给用户一个文件描述符（open文件时产生的一个整数，用来标识文件，起到索引作用）。

读read文件：系统调用read，指明文件名称和要读入文件的内存地址，通过搜索目录找到文件位置，系统维护一个读位置的指针，每当发生读操作时，更新读指针。

（根据open返回的文件描述符，找到文件位置）

写write文件：系统调用，指明文件名和要写入文件的内容，通过搜索目录找到文件位置，系统维护一个写位置指针，每当发生写操作，更新该写指针。

HashMap和HashTable都是基于哈希表来实现键值映射的工具类，通过单链表9(拉链法)解决冲突问题，容量不足（超过了阀值）时，都会自动增长。

区别： Hashtable扩容时，将容量变为原来的2倍加1，而HashMap扩容时，将容量变为原来的2倍。

Hash 表最坏情况下相当于一个单链表。所有的关键字通过函数得到的结果都一样时。

哈希表也即是哈希数组，根据关键词直接进行访问的数据结构，建议了关键字和存储地址之间的一种直接映射的关系。与表中元素个数无关，时间复杂度O（1），最坏时为O（n）, 当所有的关键字通过函数得到的结果都一样时。

Hash函数的构建，取模法。

冲突：key即经过一个函数f(key)得到的结果的作为地址去存放当前的key value键值对(这个是hashmap的存值方式)，但是却发现算出来的地址上已经有人先来了。

解决冲突：

线性探测法，发生冲突时，顺序查看表中下一单元是否空闲，若不空闲，接着往下找，知道找到一个空闲单元。

拉链法：把所有通过函数得到结果一样的关键字都放在线性链表中，该线性链表有其hash地址唯一标识。

存储过程（数据库）： 一组为了完成特定功能的SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译

如何调用存储过程，通过指定存储过程的名字并给出参数来执行

操作系统、数据库、应用软件的关系？

OS是核心，数据库在os之上，

递归：函数直接或间接调用自身，把复杂问题转换成规模小但与原问题性质相同的问题，其大大减少了代码量。

汉诺塔模型

有三根柱A，B，C在柱A上有N块盘片，所有盘片都是大的在下面，小片能放在大片上面。现要将A上的N块片移到C柱上，每次只能移动一片，而且在同一根柱子上必须保持上面的盘片比下面的盘片小，请输出移动方法。

汉诺塔思想，用递归。每次都分为第n个，和前n-1个。

1、先把A柱上面的(N-1)个圆盘移到B柱（这一步使问题的规模减少了1）。

2、  再把A柱上剩下的那个最大的圆盘移到C柱。

3、  最后把B柱上的（N-1）圆盘移到C柱。

如果没有递归可以用栈实现。How？

栈有先进后出的特点，递归调用的实质也就是循环调用

结下一下用栈实现递归的算法  
1、进栈初始化：把一张TaskPaper放到办公桌面上。  
2、出栈，即从办公桌上取一张TaskPaper，如办公桌上没有任务纸（出栈失败），则到第4步。  
3、取出任务纸后，根据任务的信息，分两种情况进行处理：  
      A、如果任务不可再分，则执行这个任务（在上面这个程序中，体现为把搬运动作打印出来），返回到第2步。  
      B、否则，划分成若干个子任务，并把这些子任务，按照执行任务的相反顺序放进栈中，保证栈顶的任务永远是下一次出栈时最应优先处理的，返回到第2步。  
4、其它处理。

3时15分，时针和分针的角度为7.5°。

Tcp/udp区别

面向连接 每一条tcp连接只能由两个端点 提供可靠交付 全双工 面向字节流  
无连接 尽最大努力交付 面向报文, 首部开销小，不提供流量控制, 没有拥塞控制 (两个都是连接的事) 不保证顺序接受支持一对一，一对多，多对一，多对多的交互通信,

函数调用用栈，函数调用之间的逻辑本身就是符合栈的特征

系统调用，用寄存器传值。

快排：时间复杂度O（nlog2n）,空间复杂度O（log2n）

一趟排序，用枢纽值将待排记录分成两块，（较小记录，枢纽值，较大记录），分别对两块记录递归排序，最终，整个序列有序。

矩阵的秩：矩阵的最高阶非零子式，其阶数就是秩。

二叉搜索树中两个结点交换之后，怎么找出来？

中序遍历输出，异常的两个就是。

存储区：

静态存储区：永恒存在不消失，包括常量、、全局变量，常变量（const变量）、静态变量。

动态存储区：会变化的，堆和栈，栈存放函数的返回地址、参数和局部变量。堆存放new运算符和malloc函数分配得到的空间。

共享存储区：进程通信的一种方式。通信的两个进程之间存在一块共享空间，通过对其读写操作来进行信息交换。

进程通信：高级通信方式：消息传递，共享存储，管道通信

低级通信方式PV

我们通过信号量机制, 来解决进程同步和互斥的问题.而pv操作是可以访问信号量的两个原语. P[原语](https://baike.baidu.com/item/%E5%8E%9F%E8%AF%AD)操作的动作是：S减1；若S减1后仍大于或等于零，则进程继续执行；若S减1后小于零，则该进程被阻塞后进入等待队列。

V原语操作的动作是：S加1；若相加结果大于零，则进程继续执行；若相加结果小于或等于零，则从待队列中唤醒一等待进程.

PIPE

1. 其本质是一个伪文件（实为内核缓存区）

采用半双工通信（互斥，同共享存储）

1. 数据一旦被读走，便不在管道中存在，不可反复读取。

内存泄露

指程序中己动态分配的堆内存由于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程序运行速度减慢甚至系统崩溃等严重后果。

const与define区别

1. define定义的只是常数，不带类型。Const定义的是常数，带类型。
2. define在编译的预处理阶段起作用，const在编译、运行时起作用。
3. define只是简单的字符串替换，没有类型检查，可能会导致边界效应。Const有对应的数据类型，要进行判断，避免低级错误。

Const \* p指向常量的指针。\*const p：指针常量，指针指向的地址不变

Const对象。 类名 const 对象名；

1）const对象不能再被修改；

2）const对象不能调用非const类型的成员函数。在非const成员函数内部，有可能会对对象的数据成员进行修改

静态路由：

由用户或网络管理员手工配置的路由信息。当网络的拓补结构或链路的状态发生变化时，网络管理员需要手工去修改路由表中相关的静态路由信息。

动态路由：路由器中的路由表项由相连的路由器之间交换信息，会不断更新

RIP（UDP）：距离向量算法，路由器根据距离选择路由，允许最大站点数为15，RIP每隔30秒做一次路由信息广播。

OSPF（直接装在IP报文里）：链路状态路由算法，洪泛法，每个路由器向邻接路由器发送链路状态广播信息。迪杰斯特拉算法。

BGP（TCP）：外部网关协议

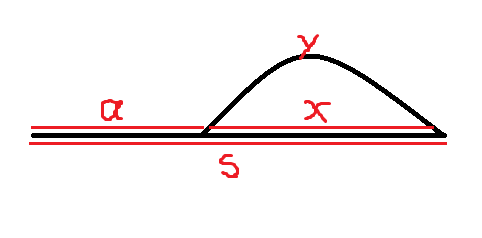
Rip应用层，ospf网络层。

隧道技术：

网络之间传递数据的方式，使用隧道技术的数据可以是不同协议的数据帧或包。隧道协议将其他协议的数据帧或包重新封装然后通过隧道发送。

链路怎么判断环,

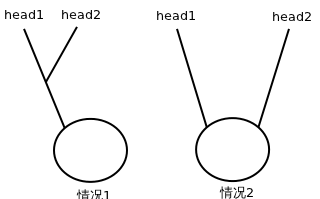
设置两个指针fast和slow，刚开始都指向链表头部，fast每次移动两个位置，slow每次移动一个位置，若两个指针相遇，说明有环。若fast为null（nʌl）, 则无环(**null只能是地址，不能是int型，表示地址为空，即不存在**)



**在链表里怎么找到环的入口**，

从链表头、与相遇点分别设一个指针，每次各走一步，两个指针再次相遇点就是环的入口。

判断可能有环可能没环的两个单链表是否相交：



找两个链表的入环结点，

先比较两个链表的入环节点是否相等，若相等，则相交，若不相等，则从某某个链表的入环节点开始循环一周，判断是否有节点等于另一个链表的入环节点，若相等，则相交，若不相等，则不相交。

若无环，直接比较两个链表的最后结点是否相等。

二叉排序树删除节点z（需先把z从二叉树中摘除，将断开的二叉树重新连起来）

1. 查找待删除节点 保存其父结点（使得二叉排序树不断）
2. 如果是叶节点（左右子节点都不存在），那么直接删除。
3. 如果其存在一颗左子树或右子树，直接将z的子树替换z即可。
4. 如果左右子树都存在，需要从左子树或右子树中选择后补节点。
   1. 如果左子树中选，选左子树中最右边的叶子节点。
   2. 如果右子树中选，选右子树中最左边的叶子节点。

lcs Longest Common Subsequence两个序列的最大公共子序列，时间复杂度O（两个序列长度的乘积）

设两个序列X,Y，分别有n个元素和m个元素。

X的最后一个元素与Y的最后一个元素相同，这说明该元素一定位于公共子序列中。因此，现在只需要找X的前n-1个元素和Y的前m-1个元素有无公共子序列。

X的最后一个元素与Y的最后一个元素不相同，则分别查找X的前n-1个元素和Y的前m个元素有无公共子序列。X的前n个元素和Y的前m-1个元素有无公共子序列。如此递归。

两个队列实现一个栈

栈是先入后出，队列是先入先出，即用两个先入先出实现先入后出。重点是，让每次插入的元素都处在队列的头部。

设两个队列q1, q2。如果有元素，则全在其中一个队列中。

插入时，将其插入到空队列中，并将另一队列的元素转移到该队列中。删除时直接从有元素的队列中其头部取出来一个。

友元：有友元函数，友元类。**提高程序的运行效率，但同时也破坏了类的封装性和数据的隐藏性，导致程序可维护性变差**

可以访问另一个类中的所有成员，包括私有成员和保护成员。

求特征向量和特征值

A为n阶矩阵，数入和n维非零列向量x，当Ax=入x时，入为A特征值，x为特征向量。

令（入E-A）的行列式为0，求得特征值。

深层含义：特征值实际上就是结构的固有振动频率，特征值对应的特征向量就是对固有频频率对应的模态

DNS递归三个步骤：如果主机所询问的本地域名服务器不知道被查询域名的ip地址，那么本地域名服务器就以DNS客户的身份，向根域名服务器继续发出查询请求报文（即替该主机查询），而不是让该主机自己进行下一步查询

C++回收：delete，或者编译时以确定生命周期。

Java动态存储管理。

Java垃圾回收机制：Java虚拟机提供的能力，用于在空闲时间以不定时的方式动态回收无任何引用的对象占据的内存空间。

1. **引用计数法。 2、 标记-清除算法**

**百度云网盘和FTP的区别**

**网盘侧重存储，FTP侧重上传。**

**FTP可以拷贝到其他计算机上，网盘不能拷贝。**

**集线器，物理层，不能隔绝冲突域和广播域**

**交换机，数据链路层，不能隔绝广播域，能隔绝冲突域**

**路由器，网络层，可以隔绝广播域和冲突域**

**数据库系统，操作系统，应用软件之间关系**

**核心是操作系统, 其之上式数据库系统, 再上是应用软件**

**最底层是硬件, 之上是操作系统(管理硬件资源),其他系统软件运行在操作系统的基础上, 再之上是应用程序**

**如何查询订单，用什么数据结构，索引**

建立索引，create unique index 索引名 on 表名

Select \* from 订单 where 订单号='XXXXX'，\* 表示所有信息。

无限插入数据：可以采用分库的方式，以解决容量和性能问题。同时采用流处理，不存储之后在处理，直接处理，同时在存储结束后再建立索引。

数据库：B+树，顺序访问，B树不行。

B+树，是B树的变形。其叶节点包含所有关键字。可以多路查找和顺序查找。

B树：所有结点的平衡因子等于0的多路查找/搜索/排序树，其叶节点未存储信息。

**使用索引查询一定能提高查询性能吗？**

**不一定。通常索饮查询比全表查询要快，但是索引需要空间来存储，定期要维护，而且当表中记录增减或更改时，索引本身也会更改，也会带来额外的I/O操作。**

**内存置换/页面置换：FLOC, 先进先出，最佳置换，最近最久未使用，时钟置换**

**内存分配：单一连续分配，固定分区分配，动态/可变分区分配（首次适应，最佳适应，最坏适应，邻近适应（同首次，只不过从上次查找结束位置开始查找）），一（分配），就有好有坏**

给定一颗二叉树，求出树中的任意两个节点之间的最大距离，即树的直径。（树是图的一种）

对于根节点，直径要么经过它，要么经过它的子树。对于前者（经过这个根节点），最远点对一定在两棵子树里面。那么我们求出**每棵子树中**距root最远的点是谁把这个距离扔进dis数组；然后从dis数组取出最大的点和第二大的点。

**于后者（经过它的子树），不用管，递归下去，交给子树处理。**

实现是dfs，**对于点x的每个子树**，记录下到x的最远点。取最远和第二远的点，这就是x所管辖的区间内的最远点对，这俩点记为u[x]和v[x]，距离记为dis[x]；然后把最远点回溯上去。

最后把所有的点都拎出来，看看谁的dis值最大

。

TLB; translation look aside buffer高速缓冲存储器, 又称联想存储器, 也称快表技术.存放进程正在访问的页表项,为了加快地址映射速度.

有并行查找能力: 多 CPU 让查询更快的查询

若页表全在内存中，读取一个数据至少要访问内存，一起找到物理地址，一次取数据。为了提高速度，引入快表。

N个结点的完全图有几个边。完全图：图中任意两个顶点之间存在边。

无向完全图有n(n-1)/2个边。N-1+。。。。+1

有向完全图有n(n-1)个边。

有n个叶子结点的平衡二叉树的深度为多少？

可以对比斐波那契数列（一个数是前两个数之和），n在哪两个之间，就是其深度。

贪心算法：通过一系列的局部最优解来达到整体的最优解。

分治：将大问题分解为若干个子问题，在子问题下继续往下分，直达不能再分通过基本问题的结果，一步步向上，最终解决大问题。

动态规划dynamic programming：通过拆分，使问题能以分治的方式姐姐。将待解决问题分解为若干个子问题，按顺序求解，前一问题的解为后面问题的求解提供了有用信息。在求任意子问题时，通过决策，从局部解中选取保留最优的局部解。依次解决各子问题，最后一个子问题的解就是初始问题的解。

现在计算机处理大数据有什么问题？

大数据的特点是：种类多，数量大，增长速度快，价值密度低，对计算机的存储和处理能力要求很高，可以采用分布式文件系统hadoop（核心mapreduce）。云计算平台，文件切分等进行处理

Java虚拟机：Java[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA)（JVM）是Java Virtual Machine的缩写，它是一个虚构出来的计算机，用来实现跨平台。一般的高级语言如果要在不同的平台上运行，至少需要编译成不同的[目标代码](https://baike.baidu.com/item/%E7%9B%AE%E6%A0%87%E4%BB%A3%E7%A0%81)。而引入Java语言[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA)后，Java语言在不同平台上运行时不需要重新编译。Java语言使用模式Java[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA)屏蔽了与具体平台相关的信息

Knn: k-nearest 算法的核心思想是如果一个样本在特征空间中的k个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。

**软件测试的目的**： 评估产品质量, 是否达到上线要求.

软件测试方法：黑盒black box测试，白盒white box测试, 灰盒gray box测试, 集成测试, 冒烟测试.

白盒测试：知道产品的内部工作过程，通过测试来验证内部动作是否按照规格说明书的规定正常进行.( 去研究里面的源代码和程序结构)  
黑盒测试：知道了产品具有的功能，通过测试来检验每个是否都能正常使用。(只关心输入和输出的结果)

灰盒测试: 黑白之间

集成测试Integration Testing：自顶向下集成测试, 自底向上集成测试等

自顶向下集成测试, 自底向上集成测试的区别:

自顶向下集成：从主控制模块开始(主程序, 根结点)，沿着程序的控制层次从上向下，逐渐将各模块组装起来。  
自底向上集成：从最底层模块(叶子结点)开始, 沿着程序的控制层次从下向上, 逐层将各模块组装起来.

冒烟测试: Smoke Testing

测试新版本是否存在致命性问题，基本功能是否正常，为后续的正式测试工作做准备. 节省测试时间,覆盖率低(细节问题不管了)

讲一下java反射机制的底层原理

反射让Java具有了动态的特性，   
 反射的核心是JVM在运行时才动态加载类或调用方法/访问属性，它不需要事先（写代码的时候或编译期）知道运行对象是谁。

迅雷快，因为用了P(to)P技术, P2P: 整个网络的传输内容不在中心服务器上，每个结点都可上传下载，网络更健壮，单个结点失效不影响其他结点。但占用过多内存，影响速度。

二叉树中找到两个结点的最近公共祖先。

若非排序二叉树

1. 找到从根到node1的路径，并存储在一个向量或数组中。  
   **2)**找到从根到node2的路径，并存储在一个向量或数组中。  
   **3)** 遍历这两条路径，直到遇到一个不同的节点，则前面的那个即为最低公共祖先.

若为排序/搜索/查找二叉树

从根节点开始，如当前节点的值比两个节点的值大，则最近公共祖先节点在该节点左子树中，下一步遍历当前节点的左子树；

如当前节点的值比两个节点的值都小，则最近公共祖先节点在该节点的右子树中，下一步遍历当前节点的右子树；这样直到找到第一个值是两个输入节点之间的值的节点，该节点就是两个节点的最近公共祖先节点

用栈输出序列的最小值

两个栈，一个栈存放数据，一个栈存放数据栈中的最小元素。

元素a入栈时，先入数据栈，然后将a最小栈的栈顶元素b比较，a<b，a入栈最小栈，a>b，b入栈最小栈。

数据栈中元素出栈时，最小栈也要出栈一个。

标记语言和c++/java等语言的区别

标记语言不用于向计算机发出指令，常用于格式化和链接，而java、c++可以向计算机发出指令。

**KNN(K Nearest Neighbours)**

看你周围的K个样本中哪个类别占的多，哪个多，那我就是多的那个。实现起来就是对每个训练样本都计算与其相似度，是Top-K个训练样本出来，看这K个样本中哪个类别的多些，谁多跟谁。

kNN 没有建立分类模型，它只是储存了一些分类好的训练数据。那么新的训练数据进入时，kNN 执行两个基本步骤：

1 首先，它观察最近的已经分类的训练数据点—也就是，k最临近点（k-nearest neighbors）

2 第二部，kNN使用新数据最近的邻近点的分类， 就对新数据分类得到了更好的结果了。

网络拓扑结构：

星形： 每个终端都以单独的线路与中央设备相连。特点便于集中控制和管理，成本高，中心结点对故障敏感。

总线形：单根传输线把计算机连接起来。建网容易，重负载通信效率不高，总线任一处对故障障碍。

环形：所有计算机接口设备连接成一个环。环中信号是单向传输。

网状形：每个结点至少有两条路径和其他结点相连。可靠性高，控制复杂，线路成本高

const void \*a这是**定义了一个指针, 可以指向任意类型的值，但它指向的值必须是常量。**

C语言中的static的作用.

让一个变量长期有效，而不管在什么地方被申明, 相当于一个全局变量(被调用后值会保存)

Static作用域，全局。

引用是一个变量的别名，对引用的操作等于对其指定的对象进行操作.

比特币：P2P形式的数字货币，虚拟货币。依据特定算法，通过大量的计算产生

C++传参：传值，传指针，传引用。

指针指向内存的地址。引用内存的别名。

插入一百和一万个数有什么区别？

当有索引时，插入数据量大，索引更新代价更高。

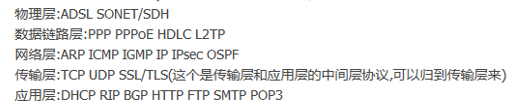
堆排的过程：利用堆顶记录最大或最小的关键字的性质，每次从无序序列中选择最大或最小记录。

将待排序列构成一个大顶堆，将堆顶元素与最后一个元素交换，将减少了一个关键字的无序序列重新进行构建大顶堆。直到有序区的元素个数为n-1.

一千个数找1百个，直接快排

一千万/百万找出一百个最小的：最大堆

首先读入前100个数来创建大小为100的最大堆，建堆的时间复杂度为O（mlogm）（m为数组的大小即为10000），然后遍历后续的数字，并于堆顶（最小）数字进行比较。如果比最小大的数大，则继续读取后续数字；如果比堆顶数字小，则替换堆顶元素并重新调整堆为最大堆。直至全部遍历完为止。然后按照中序遍历的方式输出当前堆中的所有10000个数字。该算法的时间复杂度为O（n）。

每一层的协议

电脑开始启动程序：BIOS启动，然后启动引导程序，在启动操作系统。

范式(NF Normal Form): 一张数据表中的表结构所符合的某种设计标准的级别.

**关系数据库: 采用的关系模型来组织数据的数据库. (**简单的说, 用二维表来保存数据**)**

“关系模式”和“关系”的区别，类似于面向对象程序设计中”类“与”对象“的区别。”关系“是”关系模式“的一个实例，你可以把”关系”理解为一张带数据的表，而“关系模式”是这张数据表的表结构

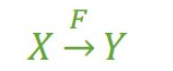
第一范式: **符合1NF的关系中的每个属性都不可再分(所有关系型数据库的最基本要求)**

**码：关系中的某个属性或者某几个属性的组合，**之外的所有属性都完全函数依赖于它,**用于区分每个元组**（可以把“元组”理解为一张表中的每条记录，也就是每一行）**。(**通常可以理解为：**假如当 K 确定的情况下，该表除 K 之外的所有属性的值也就随之确定，那么 K 就是码**一张表中可以有超过一个码。（实际应用中为了方便，通常选择其中的一个码作为**主码**）)

**非主属性** 包含在任何一个码中的属性成为主属性

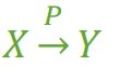
**第二范式: 关系中的每个属性都不可再分(1NF)属于第一范式，同时消除了非主属性对于码的部分函数依赖**

**完全函数依赖(full): Y函数依赖于x, 但是x的任一真子集(B是A的子集, A中至少一个元素不属于B, B为真子集)都不能唯一确定y的值.**

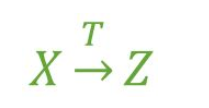


部分函数依赖(partly): Y函数依赖于x, 但是不完全依赖于x,





传递函数依赖(transfer): z函数依赖于Y, Y函数依赖于x, 则z函数依赖于x



第三范式: 关系中每个属性不可再分(1NF), 消除了非主属性对码的部分函数依赖(2NF), 同时消除了非主属性对码的传递函数依赖

巴斯-科德范式(**BC范式**): 关系中每个属性不可再分, 消除了**所有属性(非主属性和主属性)**对于码的部分与传递函数依赖。

流量控制：控制发送方的发送速度，是接收方有足够的缓冲空间来接受。停止-等待，滑动窗口。

拥塞控制：通过控制发送方的发送速度，放置过多的数据进入网络，造成通信时延的增加，是全局性。慢开始，快重传，快恢复。

最大子串和：穷尽O（n^2）

动态规划O（n）