操作系统

1. 进程和线程的区别。

线程和进程的主要区别是:他们是操作系统不同的管理系统资源的方式。进程拥有独立的地址空间，一个进程崩溃后，在保护模式下不会对其他的进程造成任何的影响，而线程只是进程不同的执行路径而已。线程有自己的堆栈和局部变量，但是线程没有独立的地址空间。通常可以将线程看成是一个轻量级的进程。

在 cpu 的调度方面，线程是调度的基本单位。并且线程的调度是比较轻量级的，提高了系统的并发性能。同一进程中的线程调度不会引起进程的调度，但是不同进程之间的线程切换还是会引起进程的切换的。

在执行过程来看，进程拥有独立的内存单元，同一进程下的线程可以共享改内存区域，提高了运行效率。

从逻辑角度来看：（重要区别）

多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但是，操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理及资源分配。

2. 死锁的必要条件，怎么处理死锁。

四个必要的条件：

1 互斥条件：一个资源每次只能被一个进程使用

2 请求与保持条件：一个进程因请求资源而阻塞，对已获得的资源保持不放。

3 不剥夺条件：进程获得的资源，在未使用完之前，不可以被强行剥夺

4 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系

处理死锁：

1 死锁的预防：破坏四个死锁的条件 但是都没一定的局限性 相应系统的执行效率

2 死锁的避免：银行家算法

3 死锁的解决：

（1）最简单，最常用的方法就是进行系统的重新启动，不过这种方法代价很大，它意味着在这之前所有的进程已经完成的计算工作都将付之东流，包括参与死锁的那些进程，以及未参与死锁的进程。

（2）撤消进程，剥夺资源。终止参与死锁的进程，收回它们占有的资源，从而解除死锁。这时又分两种情况：一次性撤消参与死锁的全部进程，剥夺全部资源；或者逐步撤消参与死锁的进程，逐步收回死锁进程占有的资源。一般来说，选择逐步撤消的进程时要按照一定的原则进行，目的是撤消那些代价最小的进程，比如按进程的优先级确定进程的代价；考虑进程运行时的代价和与此进程相关的外部作业的代价等因素。

3. Window 内存管理方式：段存储，页存储，段页存储。

分页存储管理基本思想：

用户程序的地址空间被划分成若干固定大小的区域，称为“页”，相应地，内存空间分成若干个物理块，页和块的大小相等。可将用户程序的任一页放在内存的任一块中，实现了离散分配。

页式管理的优点是没有外碎片，每个内碎片不超过页的大小。缺点是,程序全部装入内存，要求有相应的硬件支持。
分段存储管理基本思想：
将用户程序地址空间分成若干个大小不等的段，每段可以定义一组相对完整的逻辑信息。存储分配时，以段为单位，段与段在内存中可以不相邻接，也实现了离散分配。
段式管理优点是可以分别编写和编译，可以针对不同类型的段采用不同的保护，可以按段为单位来进行共享，包括通过动态链接进行代码共享。缺点是会产生碎片。
段页式存储管理基本思想：
分页系统能有效地提高内存的利用率，而分段系统能反映程序的逻辑结构，便于段的共享与保护，将分页与分段两种存储方式结合起来，就形成了段页式存储管理方式。
在段页式存储管理系统中，作业的地址空间首先被分成若干个逻辑分段，每段都有自己的段号，然后再将每段分成若干个大小相等的页。对于主存空间也分成大小相等的页，主存的分配以页为单位。
段页式系统中，作业的地址结构包含三部分的内容：段号 页号 页内位移量
程序员按照分段系统的地址结构将地址分为段号与段内位移量，地址变换机构将段内位移量分解为页号和页内位移量。
为实现段页式存储管理，系统应为每个进程设置一个段表，包括每段的段号，该段的页表始址和页表长度。每个段有自己的页表，记录段中的每一页的页号和存放在主存中的物理块号。段页式管理是段式管理与页式管理方案结合而成的所以具有他们两者的优点。但反过来说，由于管理软件的增加，复杂性和开销也就随之增加了。

4. 进程的几种状态。
创建，就绪，运行，阻塞，退出

1）就绪——执行：对就绪状态的进程，当进程调度程序按一种选定的策略从中选中一个就绪进程，为之分配了处理机后，该进程便由就绪状态变为执行状态；
2）执行——阻塞：正在执行的进程因发生某等待事件而无法执行，则进程由执行状态变为阻塞状态，如进程提出输入/输出请求而变成等待外部设备传输信息的状态，进程申请资源（主存空间或外部设备）得不到满足时变成等待资源状态，进程运行中出现了故障（程序出错或主存储器读写错等）变成等待干预状态等等；
3）阻塞——就绪：处于阻塞状态的进程，在其等待的事件已经发生，如输入/输出完成，资源得到满足或错误处理完毕时，处于等待状态的进程并不马上转入执行状态，而是先转入就绪状态，然后再由系统进程调度程序在适当的时候将该进程转为执行状态；
4）执行——就绪：正在执行的进程，因时间片用完而被暂停执行，或在采用抢先式优先级调度算法的系统中,当有更高优先级的进程要运行而被迫让出处理机时，该进程便由执行状态转变为就绪状态。
5. IPC 几种通信方式。

通信方法	无法介于内核态与用户态的原因
管道（不包括命名管道）	局限于父子进程间的通信。
消息队列	在硬、软中断中无法无阻塞地接收数据。
信号量	无法介于内核态和用户态使用。
内存共享	需要信号量辅助，而信号量又无法使用。
套接字	在硬、软中断中无法无阻塞地接收数据。

共享文件

信号量:
信号机制是 UNIX 为进程中断处理而设置的。它只是一组预定义的值，因此不能用于信息交换，仅用于进程中断控制。例如在发生浮点错、非法内存访问、执行无 效指令、某些按键（如 ctrl-c、del 等）等都会产生一个信号，操作系统就会调用有关的系统调用或用户定义的处理过程来处理。
管道:
无名管道实际上是内存中的一个临时存储区，它由系统安全控制，并且独立于创建它的进程的内存区。管道对数据采用先进先出方式管理，并严格按顺序操作，例如不能对管道进行搜索，管道中的信息只能读一次。无名管道只能用于两个相互协作的进程之间的通信，并且访问无名管道的进程必须有共同的祖先。
有名管道的操作和无名管道类似,不同的地方在于使用有名管道的进程不需要具有共同的祖先，其它进程，只要知道该管道的名字，就可以访问它。管道非常适合进程之间快速交换信息。
共享储存段:
共享存储段是主存的一部分，它由一个或多个独立的进程共享。各进程的数据段与共享存储段相关联，对每个进程来说，共享存储段有不同的虚拟地址。
信号灯:
信号灯是一组进程共享的数据结构，当几个进程竞争同一资源时（文件、共享内存或消息队列等），它们的操作便由信号灯来同步，以防止互相干扰。
信号灯保证了某一时刻只有一个进程访问某一临界资源,所有请求该资源的其它进程都将被挂起，一旦该资源得到释放，系统才允许其它进程访问该资源。信号灯通常配对使用，以便实现资源的加锁和解锁。
进程间通信的实现技术的特点是:操作系统提供实现机制和编程接口,由用户在程序中实现，保证进程间可以进行快速的信息交换和大量数据的共享。但是，上述方式主要适合在同一台计算机系统内部的进程之间的通信。
6. 什么是存虚拟内。
虚拟内存是计算机系统**内存**管理的一种技术。 它使得应用程序认为它拥有连续的可用的**内存**（一个连续完整的地址空间），而实际上，它通常**是**被分隔成多个物理**内存**碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。
7. 虚拟地址、逻辑地址、线性地址、物理地址的区别。
逻辑地址（Logical Address） 是指由程序产生的与段相关的偏移地址部分。例如，你在进行

C 语言指针编程中，可以读取指针变量本身值(&操作)，实际上这个值就是逻辑地址，它是相对于你当前进程数据段的地址，不和绝对物理地址相干。只有在 Intel 实模式下，逻辑地址才和物理地址相等（因为实模式没有分段或分页机制,Cpu 不进行自动地址转换）；逻辑也就是在 Intel 保护模式下程序执行代码段限长内的偏移地址（假定代码段、数据段如果完全一样）。应用程序员仅需与逻辑地址打交道，而分段和分页机制对您来说是完全透明的，仅由系统编程人员涉及。应用程序员虽然自己可以直接操作内存，那也只能在操作系统给你分配的内存段操作。

线性地址（Linear Address） 是逻辑地址到物理地址变换之间的中间层。程序代码会产生逻辑地址，或者说是段中的偏移地址，加上相应段的基地址就生成了一个线性地址。如果启用了分页机制，那么线性地址可以再经变换以产生一个物理地址。若没有启用分页机制，那么线性地址直接就是物理地址。Intel 80386 的线性地址空间容量为 4G（2 的 32 次方即 32 根地址总线寻址）。

物理地址（Physical Address） 是指出现在 CPU 外部地址总线上的寻址物理内存的地址信号，是地址变换的最终结果地址。如果启用了分页机制，那么线性地址会使用页目录和页表中的项变换成物理地址。如果没有启用分页机制，那么线性地址就直接成为物理地址了。

虚拟内存（Virtual Memory） 是指计算机呈现出要比实际拥有的内存大得多的内存量。因此它允许程序员编制并运行比实际系统拥有的内存大得多的程序。这使得许多大型项目也能够在具有有限内存资源的系统上实现。一个很恰当的比喻是：你不需要很长的轨道就可以让一列火车从上海开到北京。你只需要足够长的铁轨（比如说 3 公里）就可以完成这个任务。采取的方法是把后面的铁轨立刻铺到火车的前面，只要你的操作足够快并能满足要求，列车就能象在一条完整的轨道上运行。这也就是虚拟内存管理需要完成的任务。在 Linux 0.11 内核中，给每个程序（进程）都划分了总容量为 64MB 的虚拟内存空间。因此程序的逻辑地址范围是 0x0000000 到 0x4000000。
因为是做 android 的这一块问得比较少一点，还有可能上我简历上没有写操作系统的原因。
推荐书籍：《深入理解现代操作系统》

TCP/IP

1. OSI 与 TCP/IP 各层的结构与功能，都有哪些协议。

OSI:物理层,数据链路层,网络层,运输层,会话层,表示层,应用层
五层模型:物理层.数据链路层,网络层,运输层,应用层
TCP/IP：网络接口层，网际层，运输层，应用层
应用层：通过应用进程间的交互来完成特定的网络应用，应用层协议有：http 协议，SMTP 协议(电子邮件)，FTP（文件传输）
运输层：为两个进程之间的通信提供通用的数据传输服务
传输控制协议（TCP）：提供面向连接的，可靠的数据传输服务，传输的单位是报文段，但是是面向字节流传输的，保证每一个字节都是正确传输的
用户数据报协议（UDP）：提供无连接的，尽最大可能交付的数据传输协议，传输单位是用户的数据报
网络层：网络层负责为分组交换网上的不同主机提供通信服务，协议有 IP 协议，传输单位

是 IP 数据报
数据链数层：将 IP 数据报封装成帧，在相邻结点间的链路上传送帧
物理层：
2. TCP 与 UDP 的区别。

	TCP	UDP
是否连接	面向连接	面向非连接
传输可靠性	可靠的	不可靠的
应用场合	传输大量的数据	少量数据
速度	慢	快

2. TCP 报文结构。

UDP:

源端口	目的端口	长度	检验和

4. TCP 的三次握手与四次挥手过程，各个状态名称与含义，TIMEWAIT 的作用。

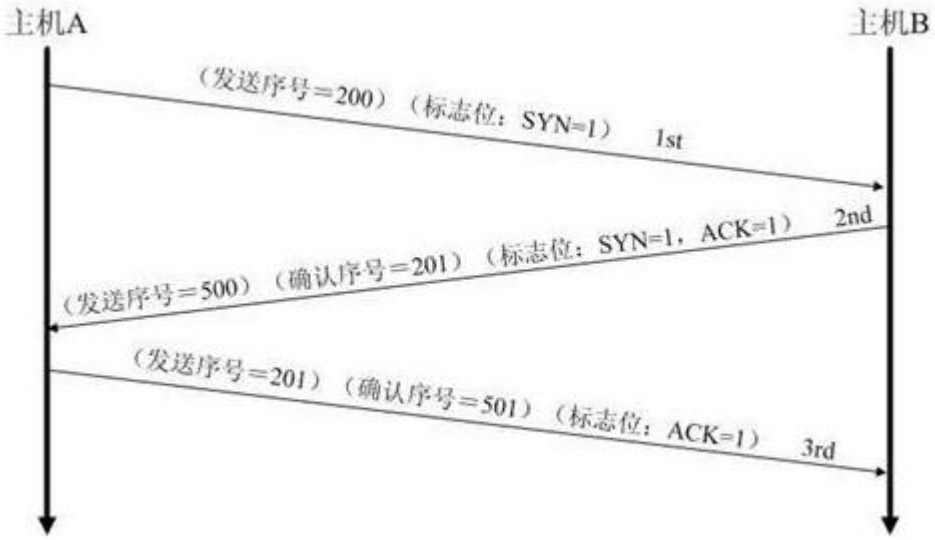
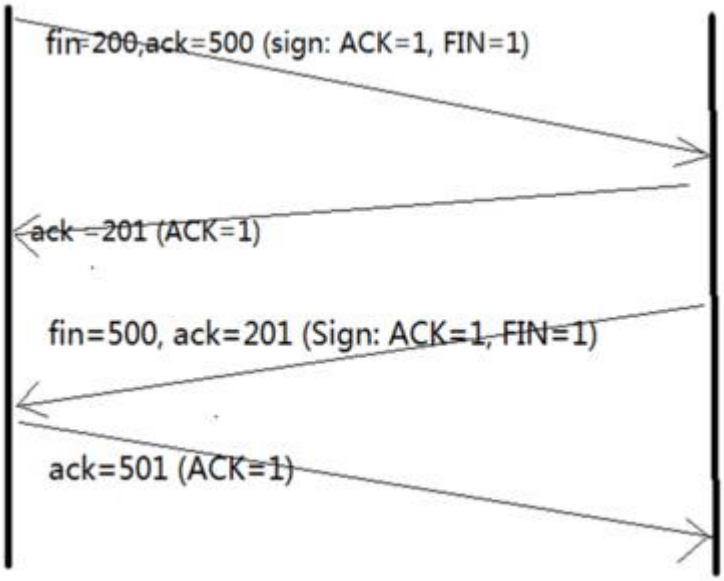


图1 TCP三次握手建立连接



为什么三次握手却四次挥手?

那可能有人会有疑问，在 tcp 连接握手时为何 ACK 是和 SYN 一起发送，这里 ACK 却没有和 FIN 一起发送呢。原因是因为 tcp 是**全双工模式，接收到 FIN 时意味将没有数据再发来，但是还是可以继续发送数据。**

TIME-WAIT

1 保证 A 最后发送的最后一个 ACK 报文段可以到达 B

2 防止出现”已失效的连接请求报文”

5. TCP 拥塞控制。

1 慢开始 2^0, 2^1,2^2 指数级增长

2 拥塞避免算法

3 快开始

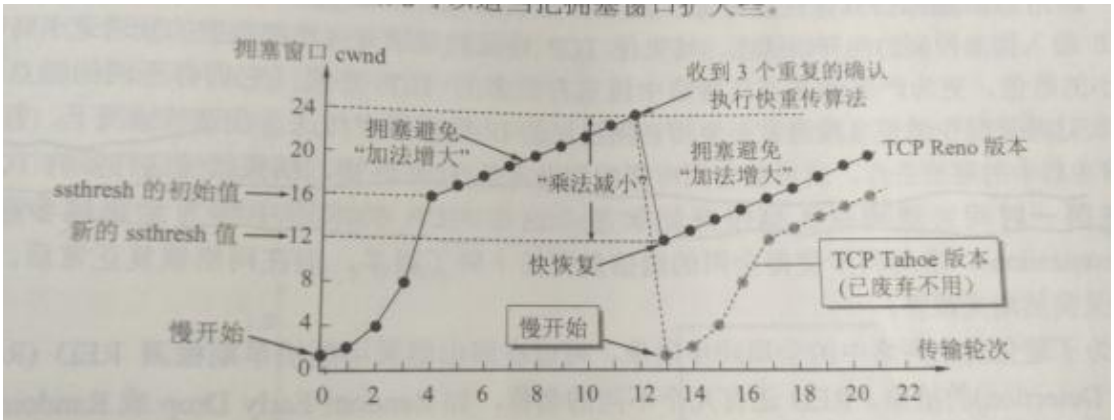
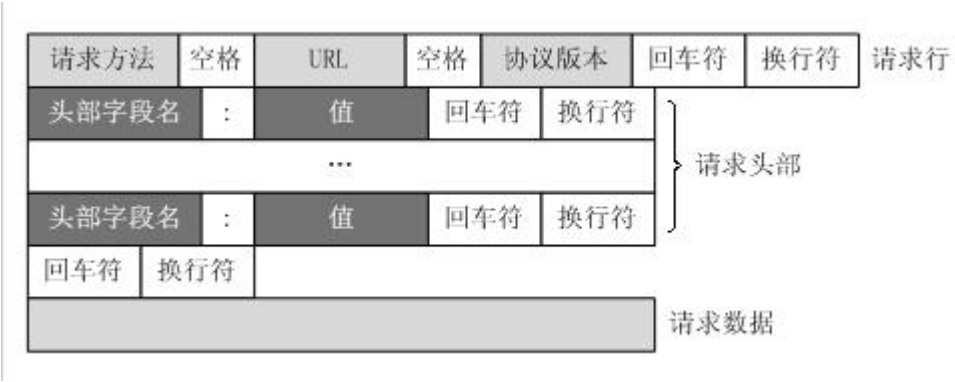


图 5-27 从连续收到三个重复的确认转入拥塞避免

6. TCP 滑动窗口与回退 N 针协议。

7. Http 的报文结构。
请求报文:
请求行,请求头部,请求数据



响应报文:状态行,消息报头,响应正文
状态行: 版本 状态码 短语
首部字段: 值
.
.
.
首部字段: 值

实体主体
8. Http 的状态码含义。
1xx:信息响应类，表示接收到请求并且继续处理
2xx:处理成功响应类，表示动作被成功接收、理解和接受
3xx:重定向响应类，为了完成指定的动作，必须接受进一步处理
4xx:客户端错误，客户请求包含语法错误或者是不能正确执行
5xx:服务端错误，服务器不能正确执行一个正确的请求
9. Http request 的几种类型。
HTTP GET: 获取资源

HTTP PUT/POST: 创建/添加资源
HTTP PUT/POST: 修改/增补资源
HTTP DELETE: 删除资源
10. Http1.1 和 Http1.0 的区别
1 长连接
HTTP 1.1 支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理，在一个 TCP 连接上可以传送多个 HTTP 请求和响应，减少了建立和关闭连接的消耗和延迟。例如：一个包含有许多图像的网页文件的多个请求和应答可以在一个连接中传输，但每个单独的网页文件的请求和应答仍然需要使用各自的连接。
2 缓存
在 HTTP/1.0 中，使用 Expire 头域来判断资源的 fresh 或 stale，并使用条件请求（conditional request）来判断资源是否仍有效。例如，cache 服务器通过 If-Modified-Since 头域向服务器验证资源的 Last-Modefied 头域是否有更新，源服务器可能返回 304（Not Modified），则表明该对象仍有效；也可能返回 200（OK）替换请求的 Cache 对象。
此外，HTTP/1.0 中还定义了 Pragma:no-cache 头域，客户端使用该头域说明请求资源不能从 cache 中获取，而必须回源获取。
11. Http 怎么处理长连接。
在 HTTP 1.0 中
没有官方的 keepalive 的操作。通常是在现有协议上添加一个指数。如果浏览器支持 keep-alive，它会在请求的包头中添加：
Connection: Keep-Alive
然后当服务器收到请求，作出回应的时候，它也添加一个头在响应中：
Connection: Keep-Alive
这样做，连接就不会中断，而是保持连接。当客户端发送另一个请求时，它会使用同一个连接。这一直继续到客户端或服务器端认为会话已经结束，其中一方中断连接。
在 HTTP 1.1 中
所有的连接默认都是持续连接，除非特殊声明不支持。

所以现在各个浏览器的高版本基本都是支持长连接。

12. Cookie 与 Session 的作用于原理。
Session 用于保存每个用户的专用信息. 每个客户端用户访问时，服务器都为每个用户分配一个唯一的会话 ID（Session ID）. 她的生存期是用户持续请求时间再加上一段时间(一般是 20 分钟左右).Session 中的信息保存在 Web 服务器内容中,保存的数据量可大可小.当 Session 超时或被关闭时将自动释放保存的数据信息.由于用户停止使用应用程序后它仍然在内存中保持一段时间,因此使用 Session 对象使保存用户数据的方法效率很低.对于小量的数据,使用 Session 对象保存还是一个不错的选择
Cookie 用于保存客户浏览器请求服务器页面的请求信息,程序员也可以用它存放非敏感性的用户信息，信息保存的时间可以根据需要设置.如果没有设置 Cookie 失效日期,它们仅保存到关闭浏览器程序为止.如果将 Cookie 对象的 Expires 属性设置为 Minvalue,则表示 Cookie 永远不会过期.Cookie 存储的数据量很受限制,大多数浏览器支持最大容量为 4K,因此不要用来保存数据集及其他大量数据.由于并非所有的浏览器都支持 Cookie,并且数据信息是以明文文本的形式保存在客户端的计算机中,因此最好不要保存敏感的,未加密的数据,否则会影响网站的安全性

13. 电脑上访问一个网页，整个过程是怎么样的：DNS、HTTP、TCP、OSPF、IP、ARP。

域名解析 --> 发起 TCP 的 3 次握手 --> 建立 TCP 连接后发起 http 请求 --> 服务器响应 http 请求，浏览器得到 html 代码 --> 浏览器解析 html 代码，并请求 html 代码中的资源（如 js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

14. Ping 的整个过程。ICMP 报文是什么。
ICMP 协议:网际控制报文协议
15. C/S 模式下使用 socket 通信，几个关键函数。

16. IP 地址分类。

17. 路由器与交换机区别。
交换机
用于同一网络内部数据的快速传输
转发决策通过查看二层头部完成
转发不需要修改数据帧
工作在 TCP/IP 协议的二层 —— 数据链路层
工作简单，直接使用硬件处理
路由器
用于不同网络间数据的跨网络传输
转发决策通过查看三层头部完成
转发需要修改 TTL ，IP 头部校验和需要重新计算，数据帧需要重新封装
工作在 TCP/IP 协议的三层 —— 网络层
工作复杂，使用软件处理
网络其实大体分为两块，一个 TCP 协议，一个 HTTP 协议，只要把这两块以及相关协议搞清楚，一般问题不大。
推荐书籍：《TCP/IP 协议族》
数据结构与算法
1. 链表与数组。
2. 队列和栈，出栈与入栈。
3. 链表的删除、插入、反向。
4. 字符串操作。

5. Hash 表的 hash 函数，冲突解决方法有哪些。
a)开放地址法
开放地执法有一个公式:Hi=(H(key)+di) MOD m i=1,2,...,k(k<=m-1)
其中，m 为哈希表的表长。di 是产生冲突的时候的增量序列。如果 di 值可能为 1,2,3,...m-1，称线性探测再散列。
如果 di 取 1，则每次冲突之后，向后移动 1 个位置.如果 di 取值可能为 1,-1,2,-2,4,-4,9,-9,16,-16,...k*k,-k*k(k<=m/2)
称二次探测再散列。如果 di 取值可能为伪随机数列。称伪随机探测再散列。仍然以学生排号作为例子，现有两名同学，李四，吴用。李四与吴用事先已排好序，现新来一名同学，名字叫王五，对它进行编制

10..	....	22	..	..	25
李四..	....	吴用	..	..	25

赵刚未来之前

10..	..	22	23	25
李四..		吴用	王五	

(a)线性探测再散列对赵刚进行编址，且di=1

10...	20	22	..	25
李四..	王五	吴用		

(b)二次探测再散列，且di=-2

1...	10...	22	..	25
王五..	李四..	吴用		

(c)伪随机探测再散列,伪随机序列为:5,3,2

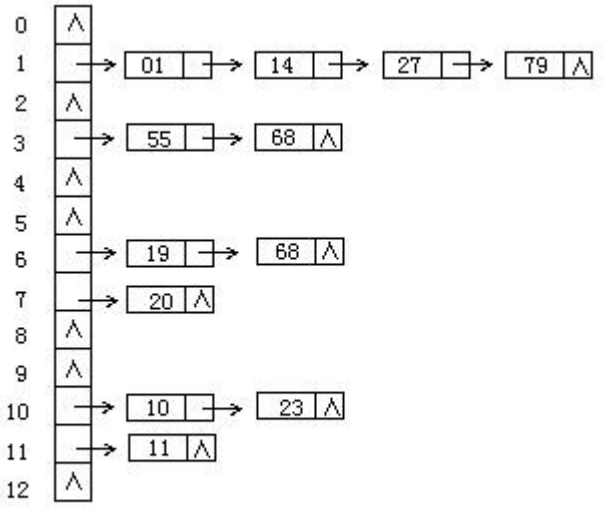
b)再哈希法

当发生冲突时，使用第二个、第三个、哈希函数计算地址，直到无冲突时。缺点：计算时间增加。

比如上面第一次按照姓首字母进行哈希，如果产生冲突可以按照姓字母首字母第二位进行哈希，再冲突，第三位，直到不冲突为止

c)链地址法

将所有关键字为同义词的记录存储在同一线性链表中。如下：



链地址法处理冲突时的哈希表
（同一链表中关键字有序）

因此这种方法，可以近似的认为是筒子里面套筒子

d.建立一个公共溢出区

假设哈希函数的值域为[0,m-1],则设向量 HashTable[0..m-1]为基本表，另外设立存储空间向量 OverTable[0..v]用以存储发生冲突的记录。

6. 各种排序：冒泡、选择、插入、希尔、归并、快排、堆排、桶排、基数的原理、平均时间复杂度、最坏时间复杂度、空间复杂度、是否稳定。

7. 快排的 partition 函数与归并的 Merge 函数。
8. 对冒泡与快排的改进。
9. 二分查找，与变种二分查找。

复杂度分析 [编辑]

时间复杂度

折半搜索每次把搜索区域减少一半，时间复杂度为$O(\log n)$。（n代表集合中元素的个数）

空间复杂度

$O(1)$。虽以递归形式定义，但是尾递归，可改写为循环。

应用 [编辑]

除直接在一个数组中查找元素外，可用在插入排序中。

示例代码 [编辑]

```
// 递归版本
int binary_search(const int arr[], int start, int end, int key) {
    if (start > end)
        return -1;

    int mid = start + (end - start) / 2; //直接平均可能會溢位，所以用此算法
    if (arr[mid] > key)
        return binary_search(arr, start, mid - 1, key);
    if (arr[mid] < key)
        return binary_search(arr, mid + 1, end, key);
    return mid; //最後檢測相等是因為多數搜尋狀況不是大於要不就小於
}
```

```
// while循环
int binary_search(const int arr[], int start, int end, int key) {
    int mid;
    while (start <= end) {
        mid = start + (end - start) / 2; //直接平均可能會溢位，所以用此算法
        if (arr[mid] < key)
            start = mid + 1;
        else if (arr[mid] > key)
            end = mid - 1;
        else
            return mid; //最後檢測相等是因為多數搜尋狀況不是大於要不就小於
    }
    return -1;
}
```

10. 二叉树、B+树、AVL 树、红黑树、哈夫曼树。

二叉树:最多两个孩子节点 除叶子节点外

满二叉树:深度为 k 结点数为 2^k-1

完全二叉树: 深度为 k,有 n 个结点 当且仅当每一个结点都与深度为 k 的 1-n 结点满二叉树一一对应时候

11. 二叉树的前中后续遍历：递归与非递归写法，层序遍历算法。

递归:

```
35. //前序遍历的算法程序
36. void PreOrder(BiTNode *root)
37. {
38.     if(root==NULL)
39.         return ;
40.     printf("%c ", root->data); //输出数据
41.     PreOrder(root->lchild); //递归调用，前序遍历左子树
42.     PreOrder(root->rchild); //递归调用，前序遍历右子树
43. }
44.
45. //中序遍历的算法程序
46. void InOrder(BiTNode *root)
47. {
48.     if(root==NULL)
49.         return ;
50.     InOrder(root->lchild); //递归调用，前序遍历左子树
51.     printf("%c ", root->data); //输出数据
52.     InOrder(root->rchild); //递归调用，前序遍历右子树
53. }
54.
55. //后序遍历的算法程序
56. void PostOrder(BiTNode *root)
57. {
58.     if(root==NULL)
59.         return ;
60.     PostOrder(root->lchild);       //递归调用，前序遍历左子树
61.     PostOrder(root->rchild);       //递归调用，前序遍历右子树
62.     printf("%c ", root->data);     //输出数据
63. }
```

非递归:

```
65. /*
66. 二叉树的非递归前序遍历，前序遍历思想：先让根进栈，只要栈不为空，就可以做弹出操作，
67. 每次弹出一个结点，记得把它的左右结点都进栈，记得右子树先进栈，这样可以保证右子树在栈中总处于左子树的下
    面。
68. */
69. void PreOrder_Nonrecursive(BiTree T)     //先序遍历的非递归
70. {
71.     if(!T)
72.         return ;
73.
74.     stack<BiTree> s;
75.     s.push(T);
76.
77.     while(!s.empty())
78.     {
79.         BiTree temp = s.top();
80.         cout<<temp->data<<" ";
81.         s.pop();
82.         if(temp->rchild)
83.             s.push(temp->rchild);
84.         if(temp->lchild)
85.             s.push(temp->lchild);
86.     }
87. }
```

```
138. void InOrderTraverse1(BiTree T)   // 中序遍历的非递归
139. {
140.     if(!T)
141.         return ;
142.     BiTree curr = T;    // 指向当前要检查的节点
143.     stack<BiTree> s;
144.     while(curr != NULL || !s.empty())
145.     {
146.         while(curr != NULL)
147.         {
148.             s.push(curr);
149.             curr = curr->lchild;
150.         }//while
151.         if(!s.empty())
152.         {
153.             curr = s.top();
154.             s.pop();
155.             cout<<curr->data<<"  ";
156.             curr = curr->rchild;
157.         }
158.     }
159. }
```

```
208. void PostOrder_Nonrecursive(BiTree T)  // 后序遍历的非递归     双栈法
209. {
210.     stack<BiTree> s1 , s2;
211.     BiTree curr ;           // 指向当前要检查的节点
212.     s1.push(T);
213.     while(!s1.empty())  // 栈空时结束
214.     {
215.         curr = s1.top();
216.         s1.pop();
217.         s2.push(curr);
218.         if(curr->lchild)
219.             s1.push(curr->lchild);
220.         if(curr->rchild)
221.             s1.push(curr->rchild);
222.     }
223.     while(!s2.empty())
224.     {
225.         printf("%c ", s2.top()->data);
226.         s2.pop();
227.     }
228. }
```

12. 图的 BFS 与 DFS 算法，最小生成树 prim 算法与最短路径 Dijkstra 算法。

DFS: Depth First Search

BFS: Breadth First Search

最小生成树:普里姆算法:从最小代价联通开始 一次寻找最小联通值

科卢卡尔算法:选择代价最小并没还没有联通的联通

13. KMP 算法。

14. 排列组合问题。
15. 动态规划、贪心算法、分治算法。（一般不会问到）
16. 大数据处理：类似 10 亿条数据找出最大的 1000 个数.....等等
算法的话其实是个重点，因为最后都是要你写代码，所以算法还是需要花不少时间准备，这里有太多算法题，写不全，我的建议是没事多在 OJ 上刷刷题（牛客网、leetcode 等），剑指 offer 上的算法要能理解并自己写出来，编程之美也推荐看一看。
推荐书籍：《大话数据结构》《剑指 offer》《编程之美》