1. **OSI和TCP/IP模型各个层之间的协议和功能★★★★★★（必考）**

<https://www.cnblogs.com/z-sm/p/7501549.html>

<https://blog.csdn.net/zejunwzj/article/details/82455124>

OSI网络七层模型:

7应用层：

规定通信协议：为用户的应用程序和各种网络服务之间提供相应的接口 （协议）

常见网络服务：www、Email、FTP,

6表示层：

把己方要发送的数据信息，翻译成对方表示层能够识别的中间信息，

5会话层：

协商双工，分割进程间数据（把接受到的数据分配给对应的进程）

a.允许同时发送和接收 全双工 （网卡）

b.不能同时发送和接收 单工 （光纤）

c.同一时间发送和接收不能同时进行 半双工 （ 对讲机）

4传输层：

建立双方进程间连接，

封装：打包

寻址方式：Port-端口，寻址目标：进程（1进程1端口）

封装单位：数据段

进程连接方式：面向连接（TCP） 保证质量

无连接（UDP ） 保证速度

3网络层：

广域网寻址，寻址对方主机所在的子网

实现跨子网的通信

寻址方式：ip，寻址目标：子网

封装单位：数据包

2数据链路层：

子网内定位目标主机

功能仅限于单个子网，跨子网失效

寻址方式：mac，寻址目标：主机

　　 寻址单位：数据帧

两个子层： （1）逻辑链路控制（LLC）

（2）介质访问控制（MAC）通过mac来定位主机

MAC：物理地址，48位（bit）的二进制

1物理层：

物理层：通过网线等介质传输,1二进制数据：规定电器特性

注：（4传输层,3网络层,2数据链路层，）较重要，有寻址，有封装。

各层协议总结：

1. 应用层：TELNET、FTP、TFTP、SMTP、SNMP、HTTP、BOOTP、DHCP、DNS（注：DNS使用的传输协议既可为TCP又可为UDP）

**http---超文本传输协议，web浏览，下载。 端口：80**

**https---http+ssl(安全套阶层协议）加密传输，CA颁发网站证书，端口：443**

**ftp---文件传输协议，下载，上传。 端口:20 （传输数据）: 21（建立连接）**

**smtp--简单邮件传输协议，邮件收发 端口:25**

**pop3--邮局协议，通过软件（outlook, foxmail）邮件的接收 端口:110**

**IMAP4--功能同pop3 端口：143**

**DHCP--动态主机控制协议，分配IP, 端口：UDP:67（S:服务器的端口）UDP:68（C:客户端的端口）**

**DNS---域名解析服务器， 端口UDP:53 TCP: 53**

**NFS---实现liunx或者aix之间数据的共享 端口:2049**

**共享--Windows之间共享文件，端口：139，445**

**Samba--Windows和Linux之间共享，端口：139,445**

**Telnet---字符界面的远程登录 明文传输 端口：23**

**ssh--- 字符界面的远程登录，加密传输 端口：22 （windows 不支持ssh，linux.aix.unix支持ssh）**

**TS---终端服务，图形界面的远程登录 端口：3389**

**VPN---虚拟局域网，异地联网**

**SNMP---简单网络管理协议，管理网络架构(例：电信监控用户路由连接多少电脑) 端口:UDP:161,162**

（2）表示层：

文本：ASCII，EBCDIC

图形：TIFF，JPEG，GIF，PICT

声音：MIDI，MPEG，QUICKTIME

（3）会话层：Socket、NFS、SQL、RPC 、X-WINDOWS、ASP（APPTALK会话协议）、SCP

（4）传输层：TCP、UDP、SPX

**端口： 16bit 1-65535 分为知名端口：1-1023 动态端口：1024-65535**

**UDP： 1--65535 TCP：1--65535**

**TCP---面向连接，保证通信质量**

**1、建立连接，三次握手（syn syn,ack ack）**

**2、数据传输，确认重传，流量控制（滑动窗口）,push标记位**

**3、断开连接，四步断开。（fin fin,ack fin fin,ack）**

**请求 标记位：syn 同意 标记位：ack**

**seq ：本数据段的段号 ctl：数据段的功能**

**UDP---,无连接 ,保证速度**

**数据传输又叫数据推送**

1. 网络层：IP、IPX、ICMP、RIP、OSPF(Open Shortest Path First开放式最短路径优先)

**IP 协议： 使用IP地址在广域网内进行寻址,寻找对方所在的子网。**

**ICMP:协议：网络控制信息协议，探测网络状况，内置了很多命令：例ping命令、tracert、netstat等。（有时服务器防火墙会关闭该协议，但客户机仍能上网访问，因为上网用的http协议）**

**ARP:协议：以广播的形式利用对方的IP地址，获取对方主机mac**

**同子网通信，发送方使用arp协议，获取对方主机mac**

**异子网通信，发送方使用arp协议，获取到网关的mac**

**Cmd命令： arp -a ---查看ARP缓存表，如下图：**

**RARP协议：反ARP协议，以广播的形式，通过自己的mac地址，获取自己需要的IP地址，DHCP协议专用**

**ddos攻击又叫拒绝服务攻击。**

（6）数据链路层：SDLC、HDLC、PPP、STP（Spanning Tree Protocol）、帧中继

（7）物理层：EIA/TIA RS-232、EIA/TIA RS-449、V.35、RJ-45

应用层：

**五层协议的原理体系结构各层所解决的问题如下：**

**物理层解决使用何种信号来传输比特的问题。**在物理媒体上传输原始的数据比特流。信道复用技术，宽带接入技术。

**数据链路层解决分组在一个网络（或一段链路）上传输的问题。**将数据分成一个个数据帧，以数据帧为单位传输。有应有答，遇错重发。封装成帧，透明传输，差错检测。

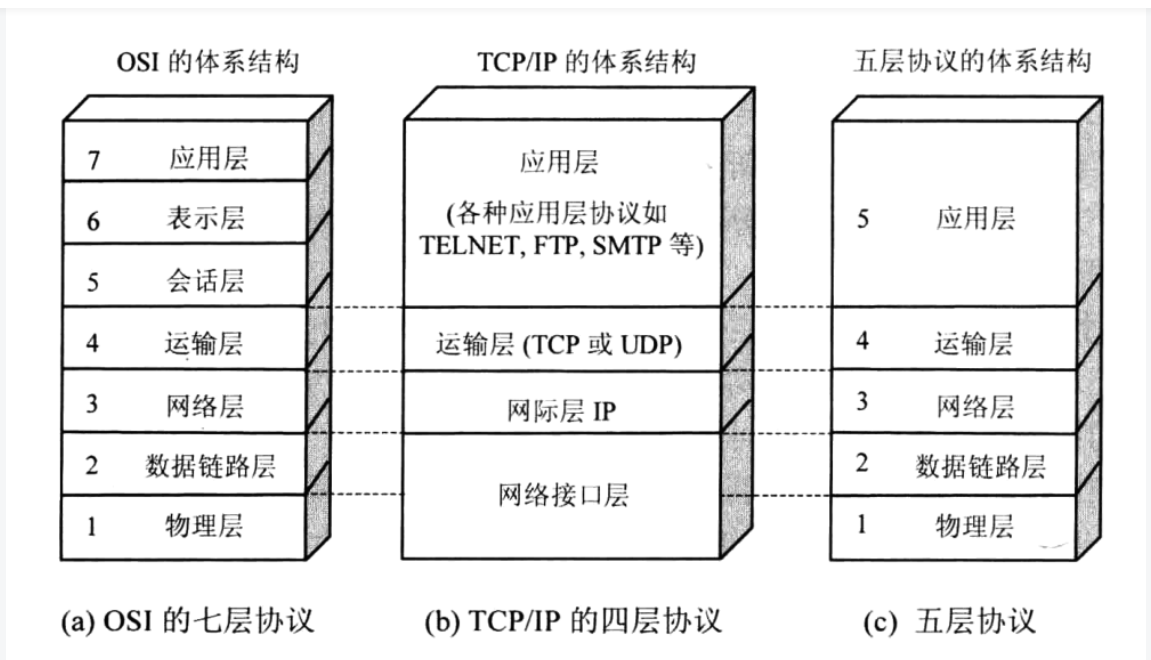
**网络层解决分组在多个网络间传输（路由）的问题。**将数据分成一定长度的分组，将分组穿过通信子网，从信源选择路径后传到信宿。路由选择

**运输层解决进程之间基于网络的通信问题。**提供不具体网络的高效、经济、透明的端到端数据传输服务。TCP的滑动窗口，流量控制，拥塞控制，连接管理。

**应用层解决应用进程间的交互来实现特定网络应用的问题。**

会话层：进程间的对话也称为会话，会话层管理不同主机上各进程间的对话。

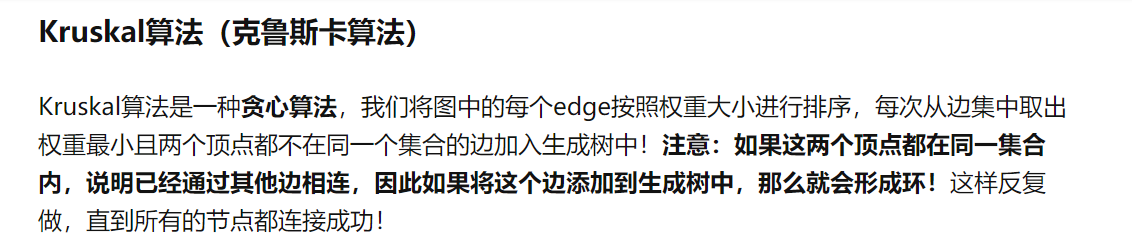
表示层： 为应用层进程提供格式化的表示和转换数据服务。

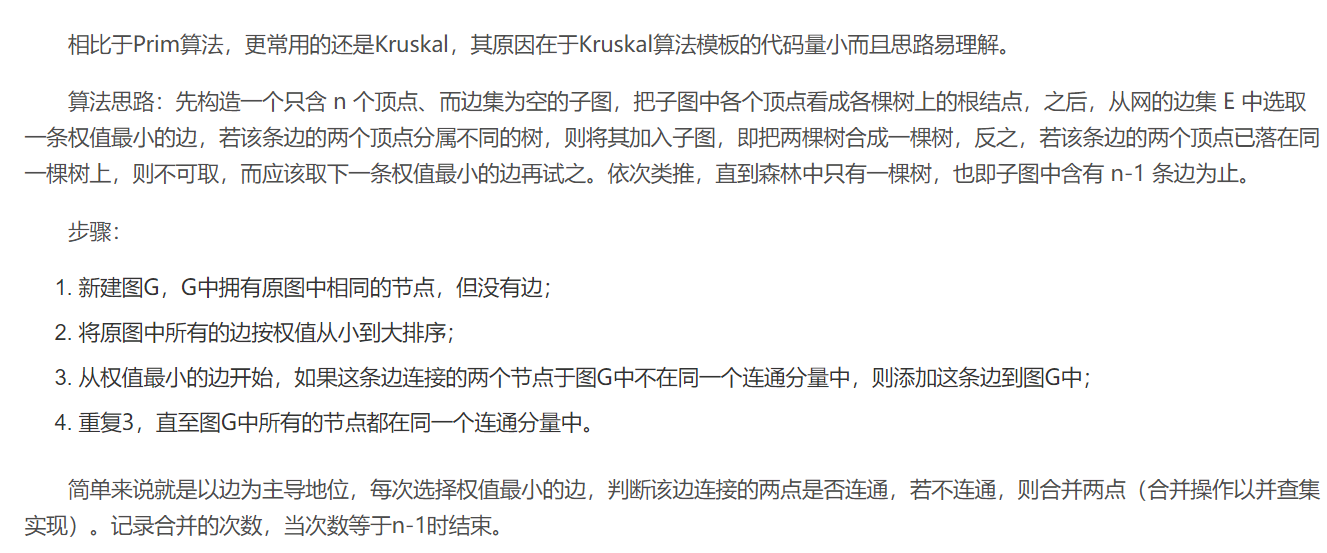


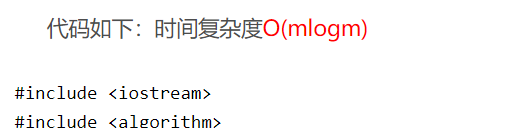
1. **最小生成树和最短路径用什么算法来实现？**

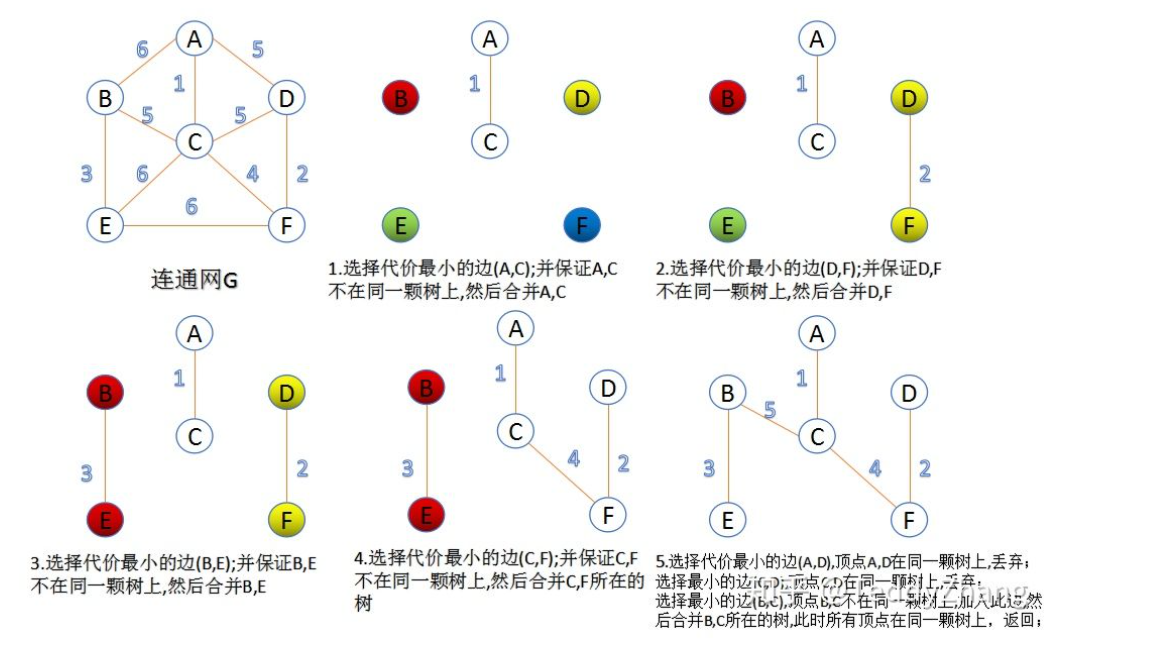
**最小生成树：**



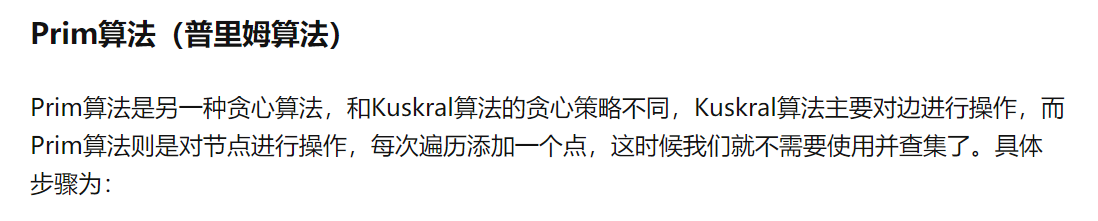


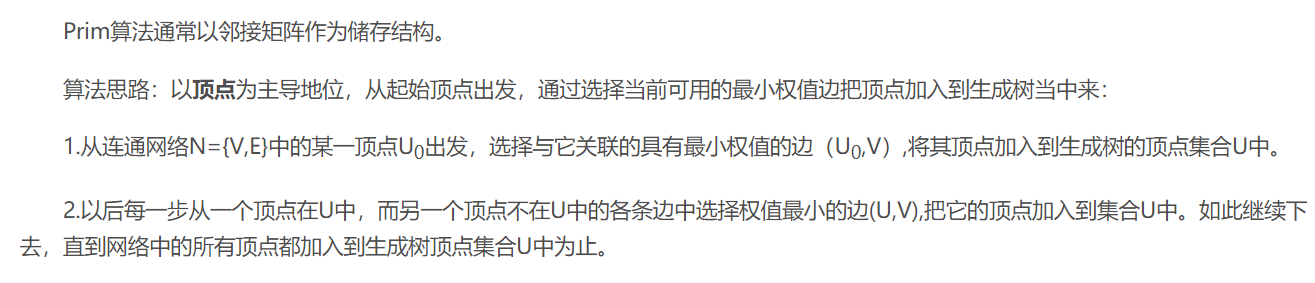


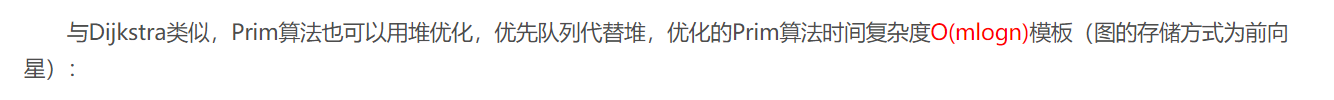


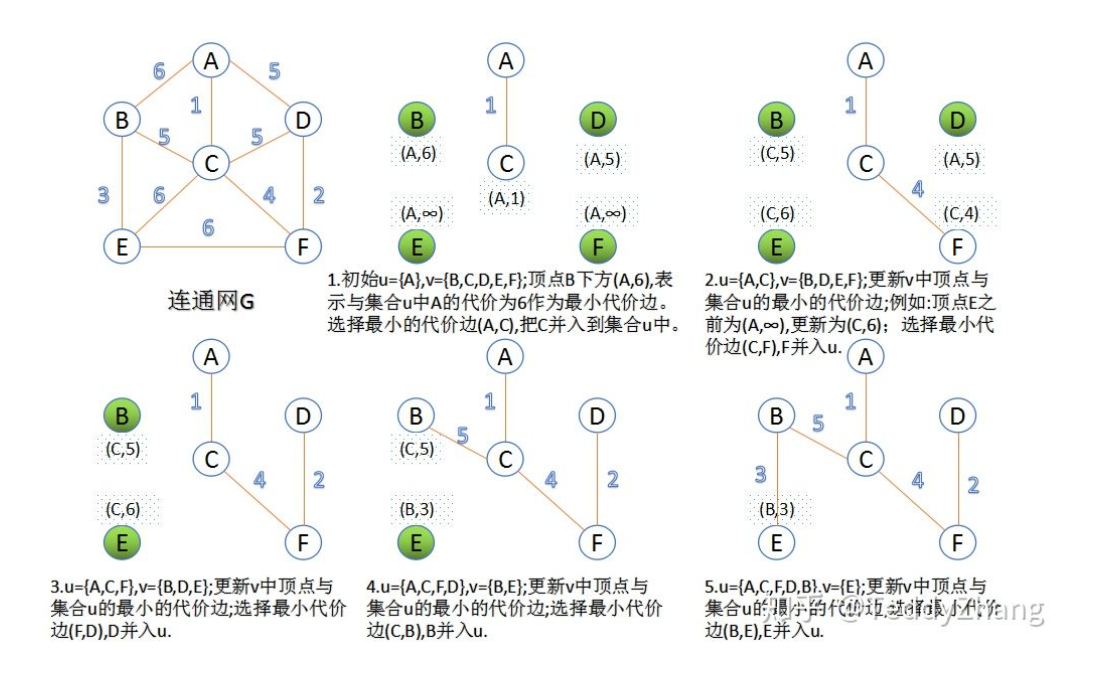


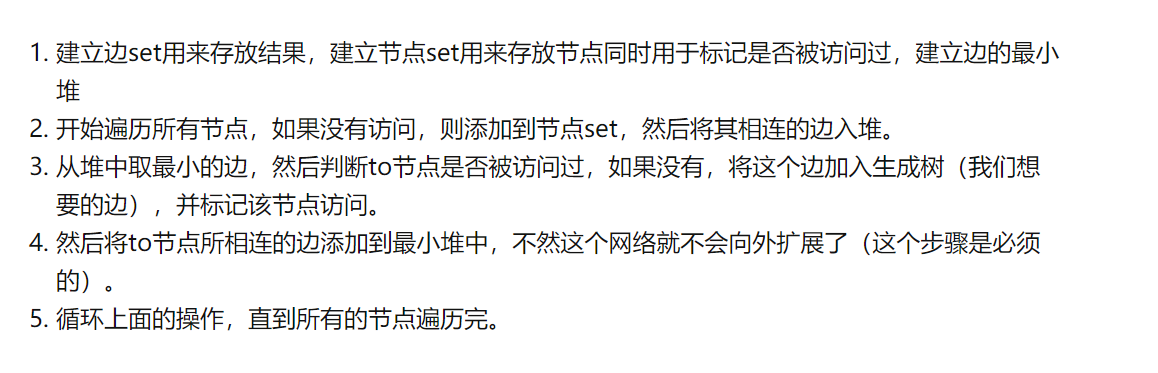


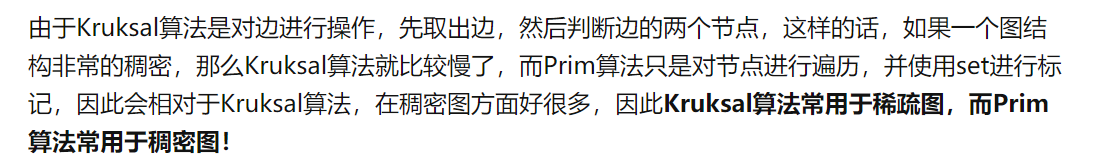




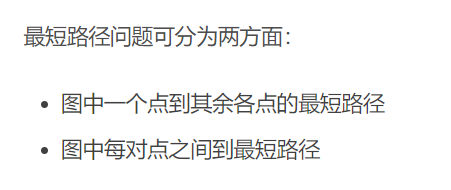


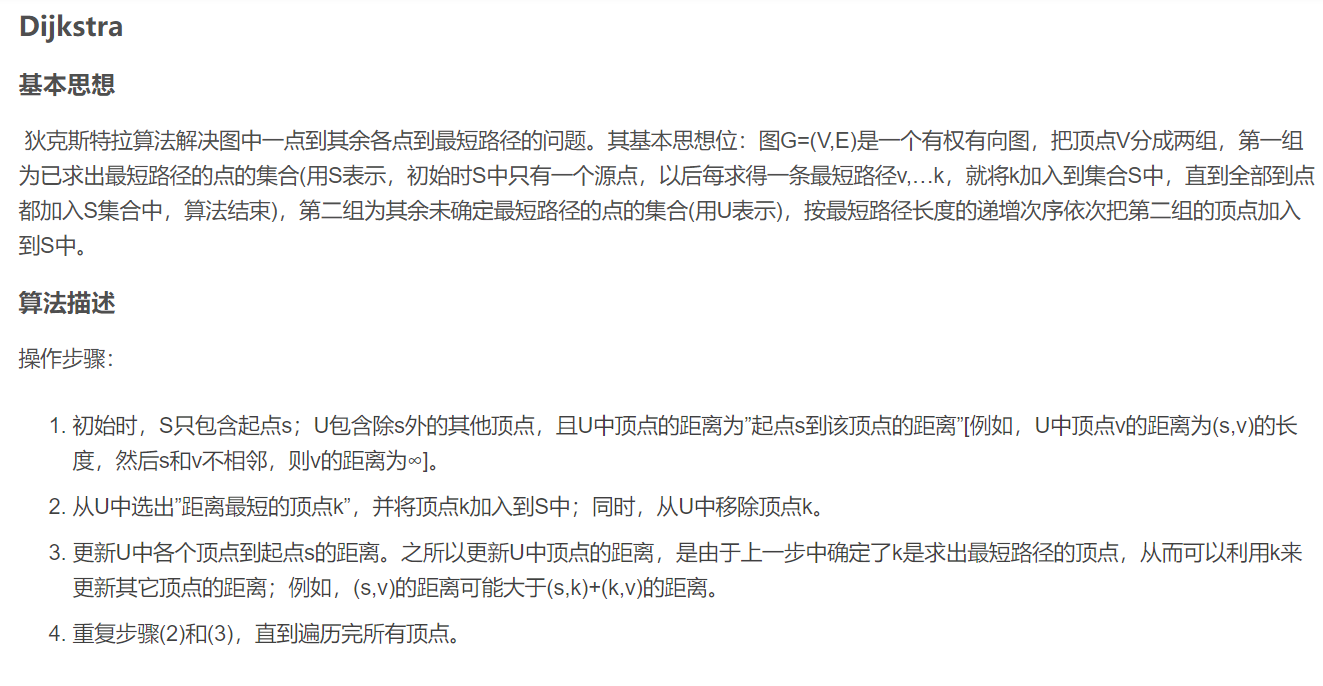




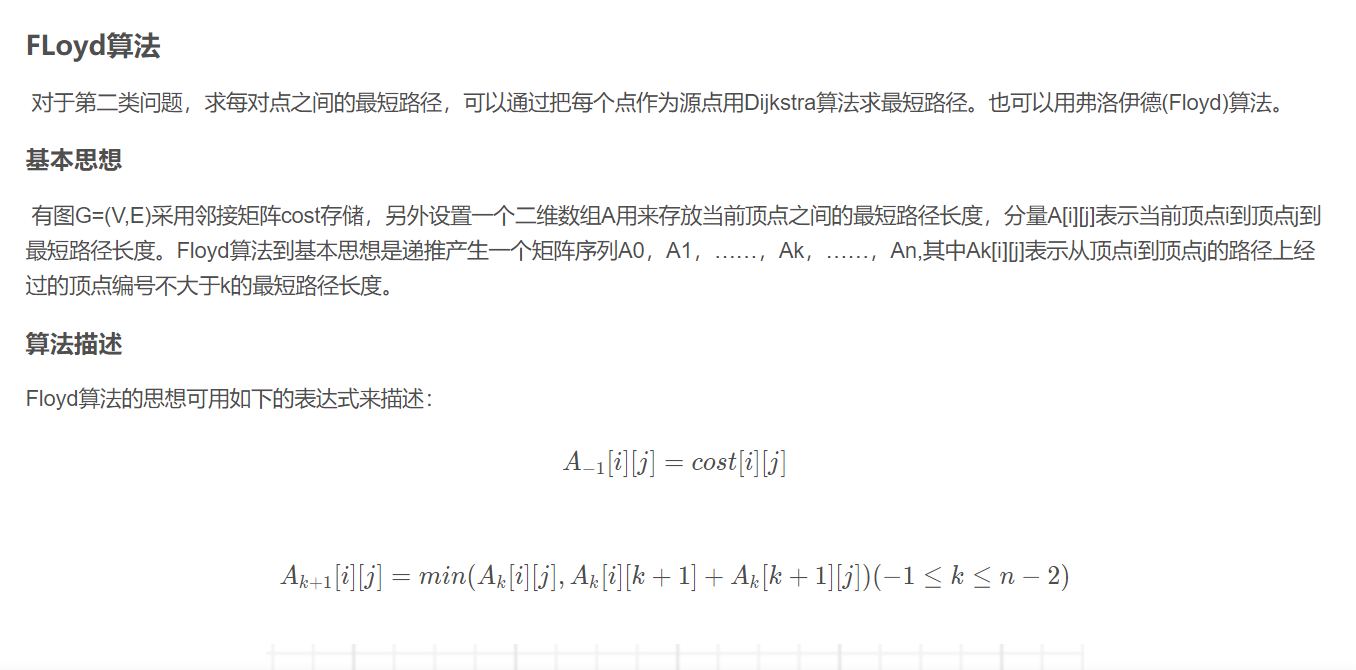


**最短路径算法：**





狄克斯特拉算法的时间复杂度O(n^2)。



弗洛伊德算法的时间复杂的O(n^3),n为顶点数。

1. **介绍一下深度优先搜索和广度优先搜索是如何实现的**

**广度优先遍历算法，从一个节点开始，优先打印其所有的下一节点，然后再打印其下一节点的下一节点，这时候就需要我们标记这个节点是否被打印过，避免重复打印！因此我们使用unordered\_set用来储存访问过的节点，并使用队列结构来储存将要打印的节点，接着在打印一个节点的同时要把其所有下一节点且未访问过的压入队列中！**

**深度优先遍历算法同样也需要一个容器用来标记是否被访问过，但是与BFS不同的是其使用的是栈结构，原因是对于DFS来说是从一个点一直遍历到最后节点，然后还要返回到上一节点判断，如果其nexts中的节点都标记访问过了，那么就再向上回溯，如果有没有访问过的节点，那么就访问，一直重复这个过程！而栈结构可以维护我们的访问节点顺序，便于回溯！**

**深度优先搜索：**

**主要思路是从图中一个未访问的顶点 V 开始，沿着一条路一直走到底，然后从这条路尽头的节点回退到上一个节点，再从另一条路开始走到底...，不断递归重复此过程，直到所有的顶点都遍历完成，它的特点是不撞南墙不回头，先走完一条路，再换一条路继续走。**

**那么深度优先遍历该怎么实现呢，有递归和非递归两种表现形式，接下来我们以二叉树为例来看下如何分别用递归和非递归来实现深度优先遍历。**

**递归：**

**递归实现比较简单，由于是前序遍历，所以我们依次遍历当前节点，左节点，右节点即可，对于左右节点来说，依次遍历它们的左右节点即可，依此不断递归下去，直到叶节点(递归终止条件)，代码如下：**

**非递归：栈**

**非递归实现**

**仔细观察深度优先遍历的特点，对二叉树来说，由于是先序遍历(先遍历当前节点，再遍历左节点，再遍历右节点)，所以我们有如下思路：**

**对于每个节点来说，先遍历当前节点，然后把右节点压栈，再压左节点(这样弹栈的时候会先拿到左节点遍历，符合深度优先遍历要求)。**

**弹栈，拿到栈顶的节点，如果节点不为空，重复步骤 1， 如果为空，结束遍历。**

**我们以以下二叉树为例来看下如何用栈来实现 DFS**

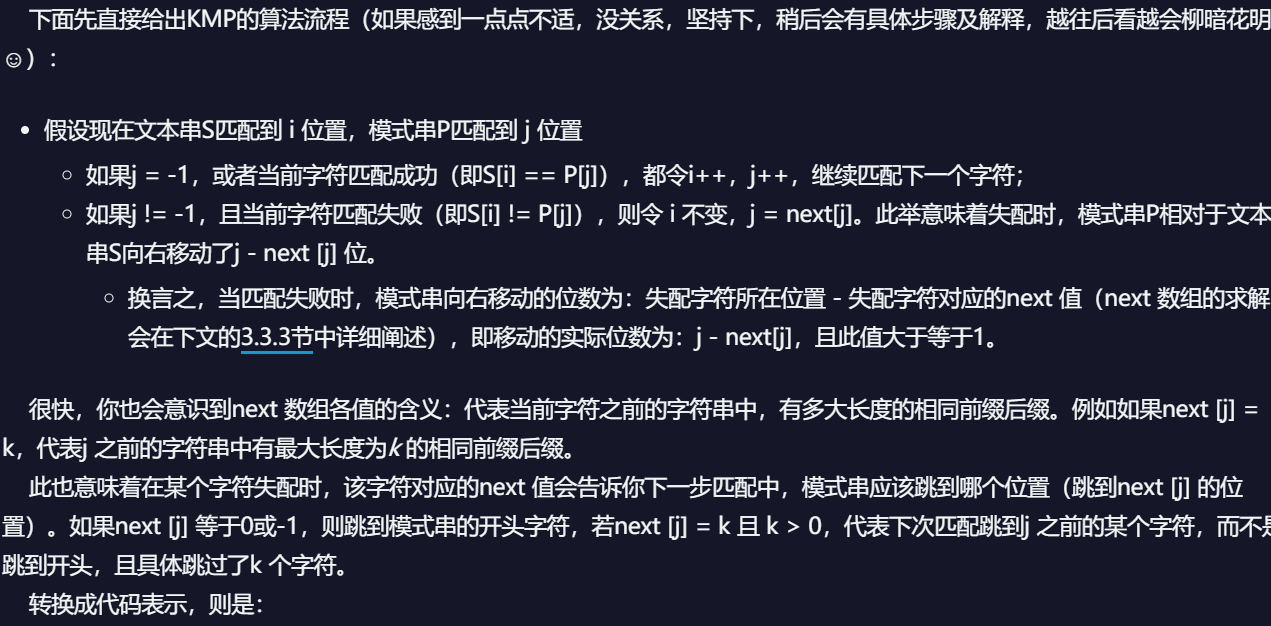
**广度优先搜索：**

**广度优先遍历，指的是从图的一个未遍历的节点出发，先遍历这个节点的相邻节点，再依次遍历每个相邻节点的相邻节点。**

1. **KMP算法**

**KMP算法是一种改进的字符串匹配算法,KMP算法的核心是利用匹配失败后的信息，尽量减模式串与主串的匹配次数以达到快速匹配的目的。具体实现就是通过一个next()函数实现，函数本身包含了模式串的局部匹配信息，KMP算法的时间复杂度O(m+n)，而使用暴力匹配的时间复杂度则是O(mn)。**

**所以kmp算法的核心即是计算字符串f每一个位置之前的字符串的前缀和后缀公共部分的最大长度（不包括字符串本身，否则最大长度始终是字符串本身）。获得f每一个位置的最大公共长度之后，就可以利用该最大公共长度快速和字符串O比较。当每次比较到两个字符串的字符不同时，我们就可以根据最大公共长度将字符串f向前移动(已匹配长度-最大公共长度)位，接着继续比较下一个位置。事实上，字符串f的前移只是概念上的前移，只要我们在比较的时候从最大公共长度之后比较f和O即可达到字符串f前移的目的。**



1. **判断链表是否有环（非常重要！）**

**方法一：穷举遍历**

**方法一：首先从头节点开始，依次遍历单链表的每一个节点。每遍历到一个新节点，就从头节点重新遍历新节点之前的所有节点，用新节点ID和此节点之前所有节点ID依次作比较。如果发现新节点之前的所有节点当中存在相同节点ID，则说明该节点被遍历过两次，链表有环；如果之前的所有节点当中不存在相同的节点，就继续遍历下一个新节点，继续重复刚才的操作。**

**例如这样的链表：A->B->C->D->B->C->D， 当遍历到节点D的时候，我们需要比较的是之前的节点A、**

**B、C，不存在相同节点。这时候要遍历的下一个新节点是B，B之前的节点A、B、C、D中恰好也存在**

**B，因此B出现了两次，判断出链表有环。**

**假设从链表头节点到入环点的距离是D，链表的环长是S。那么算法的时间复杂度是0+1+2+3+…+(D+S-1) = (D+S-1)\*(D+S)/2 ， 可以简单地理解成 O(N\*N)。而此算法没有创建额外存储空间，空间复杂度可以简单地理解成为O(1)。**

**方法二：哈希表缓存**

**\*\*\*\*首先创建一个以节点ID为键的HashSet集合，用来存储曾经遍历过的节点。然后同样是从头节点开始，依次遍历单链表的每一个节点。每遍历到一个新节点，就用新节点和HashSet集合当中存储的节点作比较，如果发现HashSet当中存在相同节点ID，则说明链表有环，如果HashSet当中不存在相同的节点ID，就把这个新节点ID存入HashSet，之后进入下一节点，继续重复刚才的操作。**

**这个方法在流程上和方法一类似，本质的区别是使用了HashSet作为额外的缓存。**

**假设从链表头节点到入环点的距离是D，链表的环长是S。而每一次HashSet查找元素的时间复杂度是O(1), 所以总体的时间复杂度是1\*(D+S)=D+S，可以简单理解为O(N)。而算法的空间复杂度还是D+S-1，可以简单地理解成O(N)。**

**方法三：快慢指针**

**首先创建两个指针1和2（在java里就是两个对象引用），同时指向这个链表的头节点。然后开始一个大循环，在循环体中，让指针1每次向下移动一个节点，让指针2每次向下移动两个节点，然后比较两个指针指向的节点是否相同。如果相同，则判断出链表有环，如果不同，则继续下一次循环。**

**例如链表A->B->C->D->B->C->D，两个指针最初都指向节点A，进入第一轮循环，指针1移动到了节**

**点B，指针2移动到了C。第二轮循环，指针1移动到了节点C，指针2移动到了节点B。第三轮循环，**

**指针1移动到了节点D，指针2移动到了节点D，此时两指针指向同一节点，判断出链表有环。**

**此方法也可以用一个更生动的例子来形容：在一个环形跑道上，两个运动员在同一地点起跑，一个运动员速度快，一个运动员速度慢。当两人跑了一段时间，速度快的运动员必然会从速度慢的运动员身后再次追上并超过，原因很简单，因为跑道是环形的。假设从链表头节点到入环点的距离是D，链表的环长是S。那么循环会进行S次（为什么是S次，有心的同学可以自己揣摩下），可以简单理解为时间复杂度O（N）。除了两个指针以外，没有使用任何额外存储空间，所**

**以空间复杂度是O（1）。**

1. **TCP和UDP的异同点★★★★★★（必考）**

# TCP

TCP（Transmission Control Protocol，传输控制协议）

# UDP

UDP（User Data Protocol，用户数据报协议)

## TCP和UDP的相同点

TCP和UDP都是在网络层，都是传输层协议，都是能保护网络层的传输，双方的通信都需要开放端口。

## TCP和UDP的不同点

传输，可靠，连接，负载，结构，功能，传输数据机制，  
  


1. **TCP的三次握手四次挥手过程？为什么会采用三次握手，若采用二次握手可以吗？★★★★★（必考）**

TCP三次握手和四次挥手的全过程

TCP是主机对主机层的传输控制协议，提供可靠的连接服务，采用三次握手确认建立一个连接:

位码即tcp标志位,有6种表示:

SYN(synchronous建立连接)

ACK(acknowledgement 表示响应、确认)

PSH(push表示有DATA数据传输)

FIN(finish关闭连接)

RST(reset表示连接重置)

URG(urgent紧急指针字段值有效)

三次握手：

第一次握手：客户端发送syn包(syn=x)到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（syn=y），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=y+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

握手过程中传送的包里**不包含数据**，三次握手完毕后，**客户端与服务器才正式开始传送数据**。理想状态下，**TCP连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都将被一直保持下去。**

**确认号：其数值等于发送方的发送序号+1(即接收方期望接收的下一个序列号)。**

**四次挥手**：

与建立连接的“三次握手”类似，断开一个TCP连接则需要“四次挥手”。

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可以接受数据。

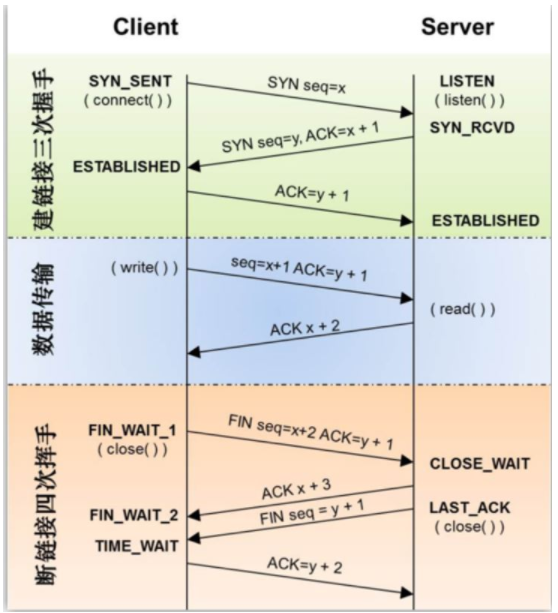
第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。

第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。

第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

TCP的四次挥手过程（简言之）：主动关闭方向被动关闭方发送不会再给你发数据了的信息；被动关闭方对收到的主动关闭方的报文段进行确认；被动关闭方向主动关闭方发送我也不会再给你发数据了的信息；主动关闭方再次对被动关闭方的确认进行确认。

TCP三次握手和四次挥手的全过程如下图：



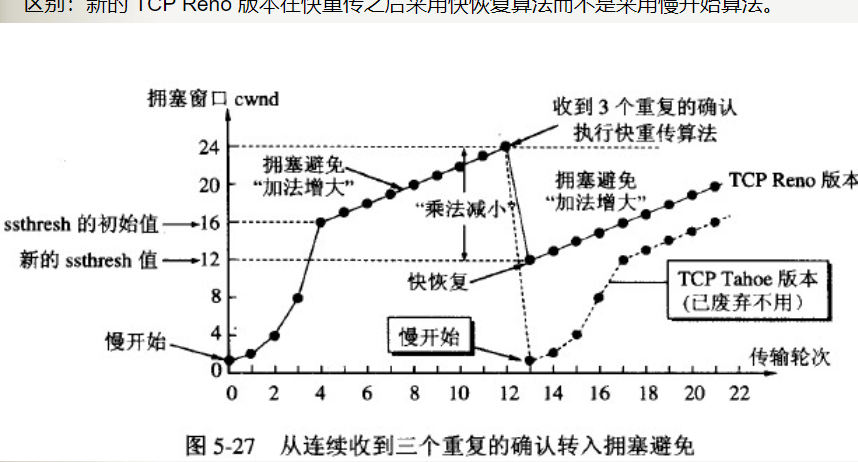
**TCP的三次握手过程？为什么会采用三次握手，若采用二次握手可以吗？**

答：建立连接的过程是利用客户服务器模式，假设主机A为客户端，主机B为服务器端。

（1）TCP的三次握手过程：主机A向B发送连接请求；主机B对收到的主机A的报文段进行确认；主机A再次对主机B的确认进行确认。

（2）采用**三次握手**是为了**防止失效的连接请求报文段突然又传送到主机B**，因而产生错误。**失效的连接请求报文段是指：**主机A发出的连接请求没有收到主机B的确认，于是经过一段时间后，主机A又重新向主机B发送连接请求，且建立成功，顺序完成数据传输。考虑这样一种特殊情况，主机A第一次发送的连接请求并没有丢失，而是因为**网络节点导致延迟达到主机B**，主机B以为是主机A又发起的新连接，于是主机B同意连接，并向主机A发回确认，但是此时主机A根本不会理会，主机B就一直在等待主机A发送数据，导致主机B的资源浪费。

1. **采用两次握手不行，原因就是上面说的失效的连接请求的特殊情况，因此采用三次握手刚刚好，两次可能出现失效，四次甚至更多次则没必要，反而复杂了。**
2. 在TCP拥塞控制中，什么是慢开始、拥塞避免、快重传和快恢复算法？★★★★



**慢开始**： 在主机刚刚开始发送报文段时可先将**拥塞窗口cwnd设置为一个最大报文段MSS的数值**。在每收到一个对新的报文段的确认后，将拥塞窗口增加至多一个MSS的数值。用这样的方法逐步增大发送端的拥塞窗口cwnd，可以分组注入到网络的速率更加合理。

**拥塞避免**：当**拥塞窗口值大于慢开始门限**时，停止使用慢开始算法而改用拥塞避免算法。拥塞避免算法使发送的拥塞窗口每经过一个往返时延RTT就增加一个MSS的大小。

**快重传算法规定：**发送端只要一连收到三个重复的ACK即可断定有分组丢失了，就应该**立即重传丢手的报文段**而不必继续等待为该报文段设置的重传计时器的超时。

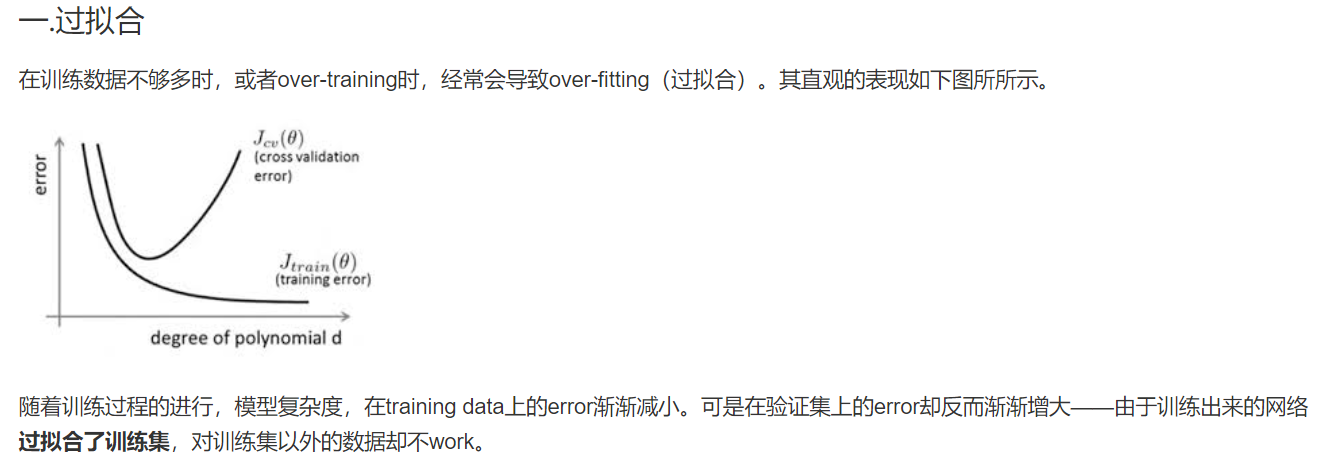
**快恢复算法**：当发送端收到连续三个重复的ACK时，就重新设置**慢开始门限ssthresh与慢开始不同之处是拥塞窗口cwnd不是设置为1，而是设置为ssthresh，若收到的重复的AVK为n个（n>3），则将cwnd设置为ssthresh，若发送窗口值还容许发送报文段，就按拥塞避免算法继续发送报文段。若收到了确认新的报文段的ACK，就将cwnd缩小**

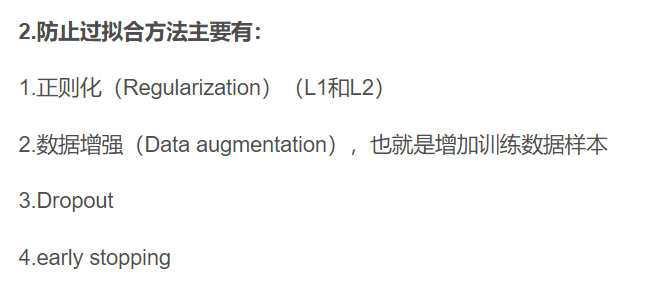
**到ssthresh**

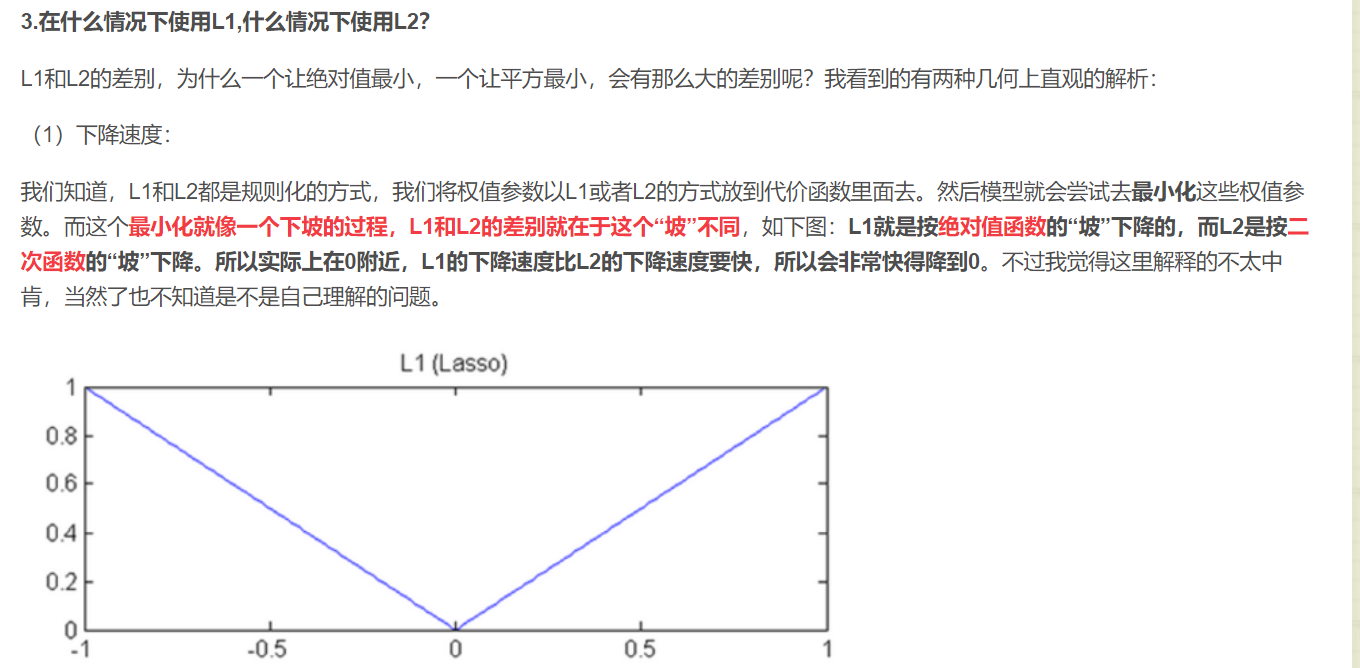
**乘法减小**：是指不论在慢开始阶段还是拥塞避免阶段，只要出现一次超时（即出现一次网络拥塞），就把慢开始门限值ssthresh设置为当前的拥塞窗口值乘以0.5。当网络频繁出现拥塞时，ssthresh值就下降得很快，以大大减少注入到网络中的分组数。

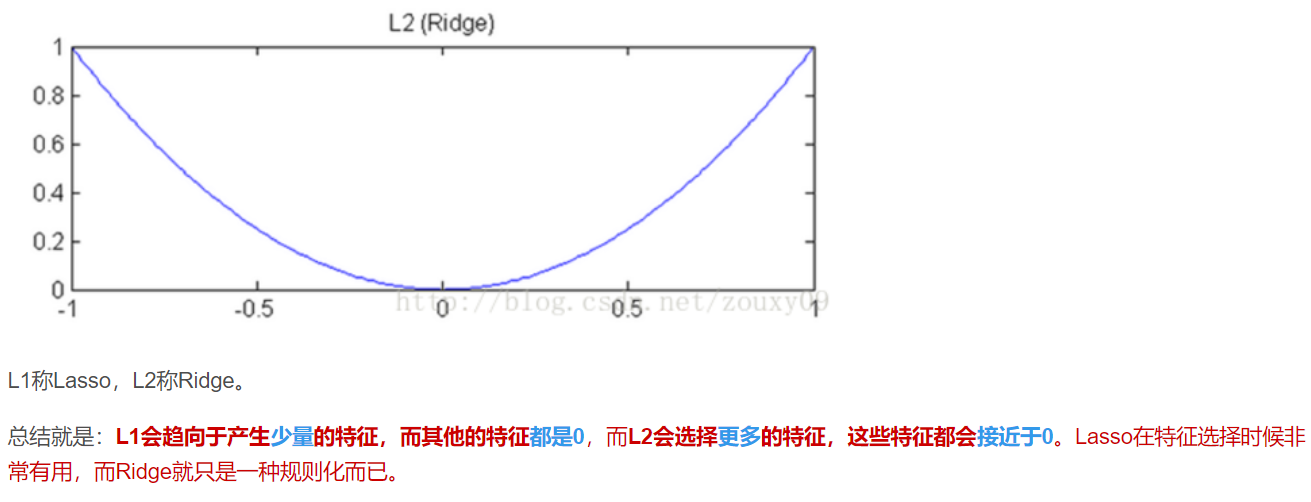
**加法增大**：是指执行拥塞避免算法后，在收到对所有报文段的确认后（即经过一个往返时间），就把拥塞窗口cwnd增加一个MSS大小，使拥塞窗口缓慢增大，以防止网络过早出现拥塞

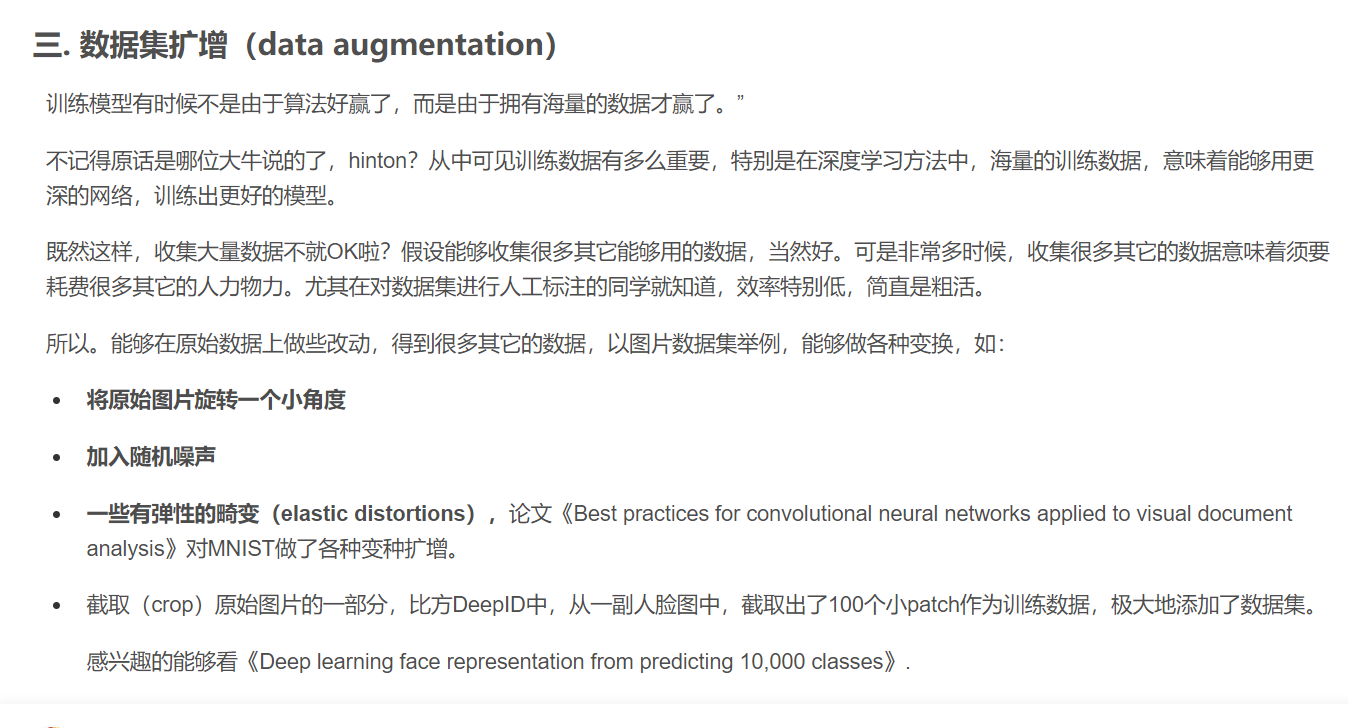
1. **如何解决过拟合和欠拟合**

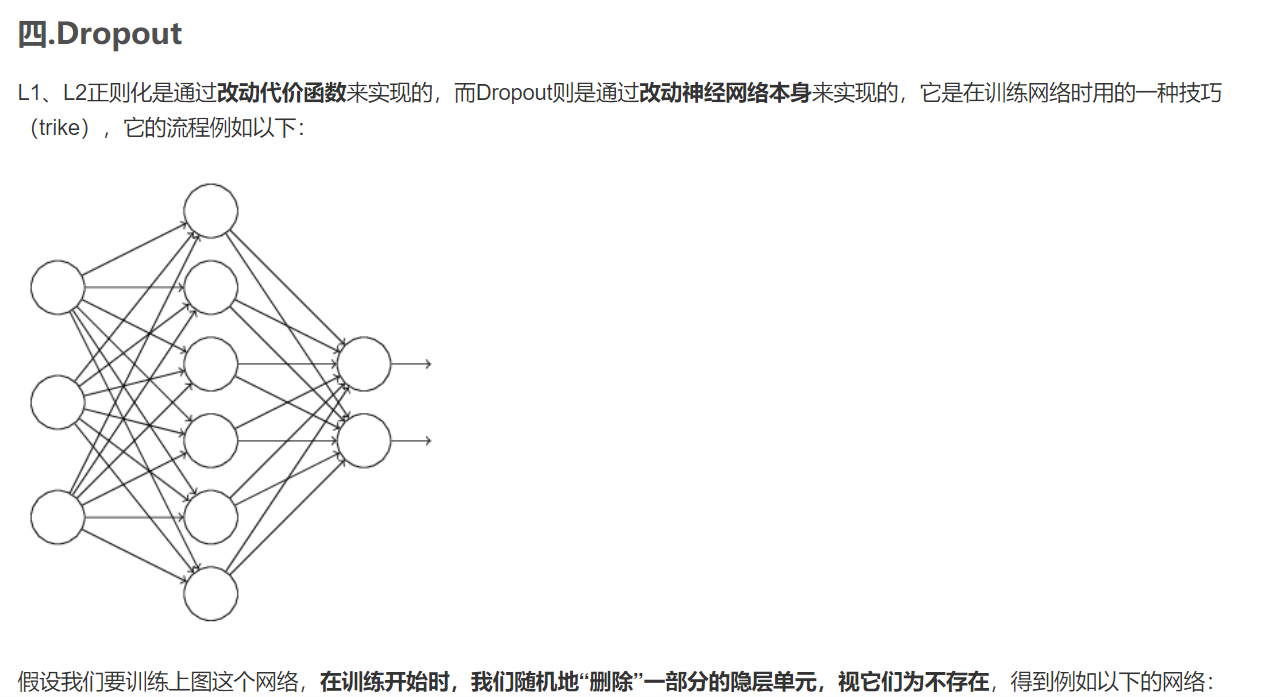


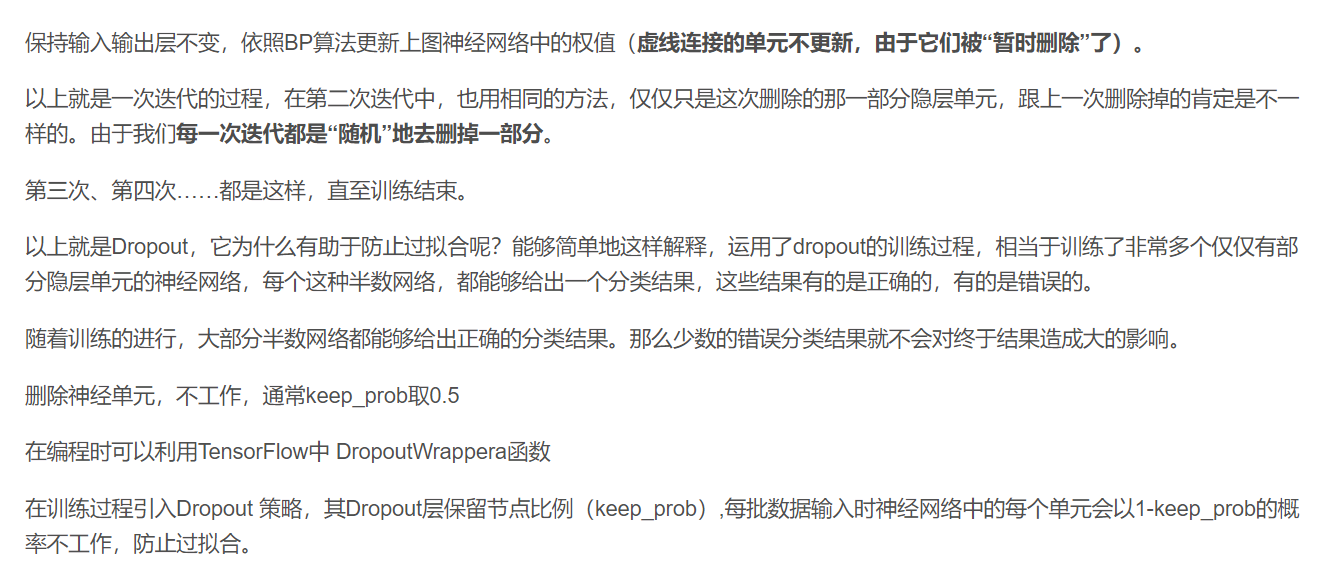


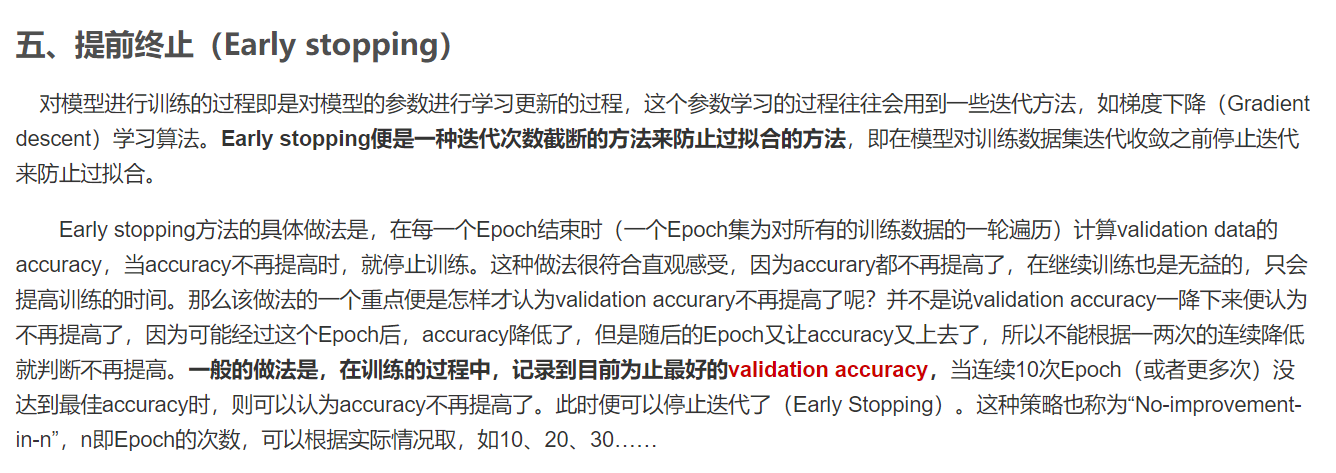


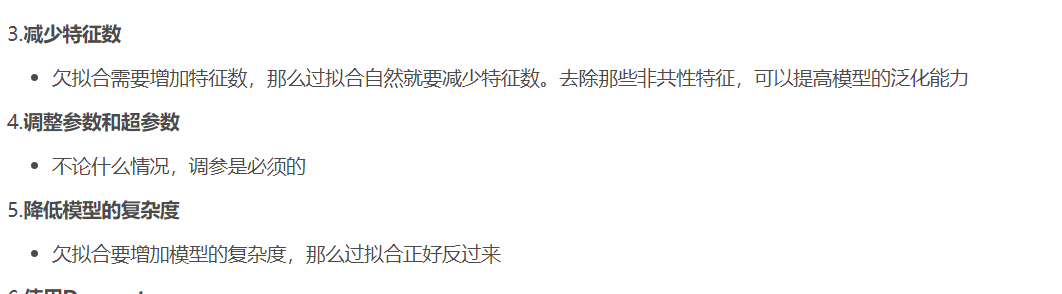














1. **如何解决梯度消失和梯度爆炸**

**一、为什么会产生梯度消失和梯度爆炸？**

目前优化神经网络的方法都是基于BP，即根据损失函数计算的误差通过梯度反向传播的方式，指导深度网络权值的更新优化。其中将误差从末层往前传递的过程需要**链式法则（Chain Rule）**的帮助，因此反向传播算法可以说是梯度下降在链式法则中的应用。

而链式法则是一个**连乘的形式**，所以当层数越深的时候，梯度将以指数形式传播。梯度消失问题和梯度爆炸问题一般随着网络层数的增加会变得越来越明显。在根据损失函数计算的误差通过梯度**反向传播**的方式对深度网络权值进行更新时，得到的**梯度值接近0**或**特别大**，也就是**梯度消失**或**爆炸**。梯度消失或梯度爆炸在本质原理上其实是一样的。

**二、分析产生梯度消失和梯度爆炸的原因**

**【梯度消失】**经常出现，产生的原因有：一是在**深层网络**中，二是采用了**不合适的损失函数**，比如sigmoid。当梯度消失发生时，接近于输出层的隐藏层由于其梯度相对正常，所以权值更新时也就相对正常，但是当越靠近输入层时，由于梯度消失现象，会导致靠近输入层的隐藏层权值更新缓慢或者更新停滞。这就导致在训练时，只等价于后面几层的浅层网络的学习。

**【梯度爆炸】**一般出现在**深层网络**和**权值初始化值太大**的情况下。在深层神经网络或循环神经网络中，**误差的梯度可在更新中累积相乘**。如果网络层之间的**梯度值大于 1.0**，那么**重复相乘会导致梯度呈指数级增长**，梯度变的非常大，然后导致网络权重的大幅更新，并因此使网络变得不稳定。

梯度爆炸会伴随一些细微的信号，如：①模型不稳定，导致更新过程中的损失出现显著变化；②训练过程中，在极端情况下，权重的值变得非常大，以至于溢出，导致模型损失变成 NaN等等。

**三、解决方法**

梯度消失和梯度爆炸问题都是因为网络太深，网络权值更新不稳定造成的，本质上是因为梯度反向传播中的连乘效应。解决梯度消失、爆炸主要有以下几种方法：

**（1） pre-training+fine-tunning**

此方法来自Hinton在2006年发表的一篇论文，Hinton为了解决梯度的问题，提出采取无监督逐层训练方法，其基本思想是每次训练一层隐节点，训练时将上一层隐节点的输出作为输入，而本层隐节点的输出作为下一层隐节点的输入，此过程就是逐层“预训练”（pre-training）；在预训练完成后，再对整个网络进行“微调”（fine-tunning）。此思想相当于是先寻找局部最优，然后整合起来寻找全局最优，此方法有一定的好处，但是目前应用的不是很多了。

**（2） 梯度剪切：对梯度设定阈值**

梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。

**（3） 权重正则化**

另外一种解决梯度爆炸的手段是采用权重正则化（weithts regularization），正则化主要是通过对网络权重做正则来限制过拟合。如果发生梯度爆炸，那么权值就会变的非常大，反过来，通过正则化项来限制权重的大小，也可以在一定程度上防止梯度爆炸的发生。比较常见的是 L1 正则和 L2 正则，在各个深度框架中都有相应的API可以使用正则化。

关于 L1 和 L2 正则化的详细内容可以参考我之前的文章——[欠拟合、过拟合及如何防止过拟合](https://zhuanlan.zhihu.com/p/72038532)

**（4） 选择relu等梯度大部分落在常数上的激活函数**

relu函数的导数在正数部分是恒等于1的，因此在深层网络中使用relu激活函数就不会导致梯度消失和爆炸的问题。

关于relu等激活函数的详细内容可以参考我之前的文章——[温故知新——激活函数及其各自的优缺点](https://zhuanlan.zhihu.com/p/71882757)

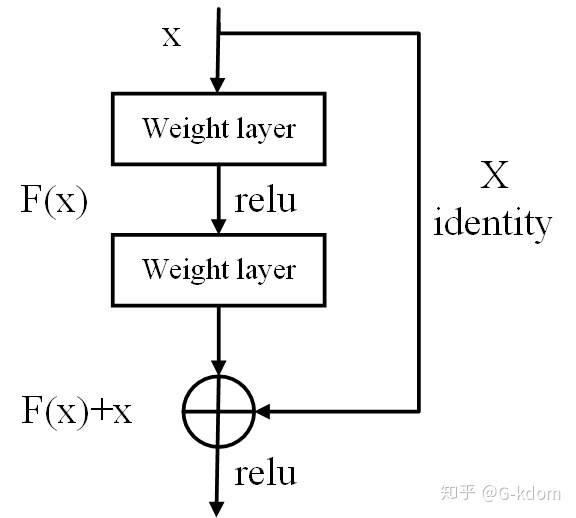
**（5） batch normalization**

BN就是通过对每一层的输出规范为均值和方差一致的方法，消除了权重参数放大缩小带来的影响，进而解决梯度消失和爆炸的问题，或者可以理解为BN将输出从饱和区拉倒了非饱和区。

关于Batch Normalization（BN）的详细内容可以参考我之前的文章——[常用的 Normalization 方法：BN、LN、IN、GN](https://zhuanlan.zhihu.com/p/72589565)

**（6） 残差网络的捷径（shortcut）**

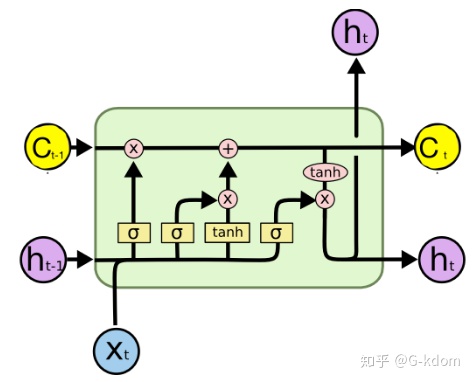
说起残差结构的话，不得不提这篇论文了：Deep Residual Learning for Image Recognition。论文链接：[http://openaccess.thecvf.com/content\_cvpr\_2016/papers/He\_Deep\_Residual\_Learning\_CVPR\_2016\_paper.pdf](https://link.zhihu.com/?target=http://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf)



相比较于以前直来直去的网络结构，残差中有很多这样（如上图所示）的跨层连接结构，这样的结构在反向传播中具有很大的好处，可以避免梯度消失。

**（7） LSTM的“门（gate）”结构**

**LSTM**全称是长短期记忆网络（long-short term memory networks），LSTM的结构设计可以改善RNN中的梯度消失的问题。主要原因在于LSTM内部复杂的“门”(gates)，如下图所示。



LSTM 通过它内部的“门”可以在接下来更新的时候“记住”前几次训练的”残留记忆“。

**计算机网络原理**

1、 TCP三次握手

2、 OSI 7层结构各自的作用

3、 说下TCP和UDP区别

4、 说下网络中的主机通信流程

5、 一个主机将两个端口接到网络上是否会提升吞吐量？为什么？

6、 udp和tcp哪个更快，哪个更可靠

7、点击网站链接背后发生的事

**1：你在项目里边主要做了哪些工作（每次面试必问的）**

**（这就需要你对第一个项目非常熟悉了，面试的时候是不能看简历的，所以建议写下来，给会背，形成文字版的）**

**2：你这个xxx项目主要是做什么的？你这个xx算法描述下？**

**3：xxx数据集哪来的？算法的识别效果、精度怎么样？损失函数？**

**4：xx实验是怎么做的？这里会问一些实验的细节部分。**

**5：算法还有哪些需要改进的地方？你们做了哪些改进？**

**（这里考察你是否有改模型的潜力，一般发论文都是要针对原有的模型进行改进的）**

**6：为什么使用xx，不使用xx算法？**

**（一般都是同系列的算法，这里考察你的思考能力，各种模型的优缺点要了解）**

**7：你在项目中遇到什么问题或者困难，是**

**如何克服的？**

**面试注意事项：**

**1:面试的时候老师是不会问代码的**

**2：面试的时候只需要能说出来就行，即使算法不理解或者公式看不懂，只需要能说出来就行**

**3：面试的时候老师问一个问题，自己就把准备的全部都说出来就行（自己要多说，老师才会少问）**

**4：面试的时候提前准备好几个点，面试的时候即使不问也把这些都说出来，深入的说**