CS研究生复试面试问题整理

数据结构

1.O(n)的大O是什么意思？什么是时间复杂度? ★★★

- **O(n)** 表示算法的时间复杂度，其中的 **n** 是问题的规模（通常指数据的规模）。**O(n)** 意味着算法的运行时间与问题规模 **n** 呈线性关系。
- **时间复杂度** 表示算法运行所需要的时间与输入数据规模之间的关系。它描述的是算法的运行时间随着问题规模增长时的增长趋势，而不是精确的运行时间。通常使用大O表示法来表示时间复杂度。

2.线性存储结构和链式存储结构的优点★★★

- **线性存储结构**:
    - 优点:
        - 存取速度快，因为数据在内存中是连续存储的，可以通过下标直接访问元素。
        - 空间效率高，不需要额外的空间存储指针。
    - 缺点:
        - 需要预先确定存储空间大小，无法动态扩展。
        - 插入和删除操作需要移动大量元素，效率较低。
- **链式存储结构**:
    - 优点:
        - 动态分配内存，不需要预先确定存储空间大小，可以根据需要动态扩展。
        - 插入和删除操作简单高效，不需要移动大量元素。
    - 缺点:
        - 存取速度相对较慢，因为需要通过指针进行遍历访问。
        - 空间效率较低，需要额外的空间存储指针。

3.解释一下顺序存储与链式存储★★★

- **顺序存储**:
    - 使用一块连续的内存空间来存储数据结构中的元素，元素之间通过下标进行访问。
    - 适用于线性表、栈、队列等数据结构。
    - 优点是存取速度快，但缺点是不便于动态扩展和插入删除操作。
- **链式存储**:
    - 使用一系列通过指针连接的节点来存储数据结构中的元素，节点可以分布在内存的任意位置。
    - 适用于链表、树、图等数据结构。
    - 优点是动态分配内存，便于插入删除操作和动态扩展，但缺点是存取速度相对较慢。

4.头指针和头结点的区别？★★

- **头指针**:

    - 指向链表中第一个节点的指针，用于标识链表的起始位置。
    - 头指针是链表的必要组成部分，它可以为空指针（表示空链表），也可以指向链表的第一个节点。

- **头结点**：

    - 不存储数据的额外节点，位于链表的第一个节点之前。
    - 头结点的主要作用是简化链表的操作，例如在链表的第一个位置插入节点时，可以不用特殊处理。
    - 头结点可以使空链表和非空链表的处理方式统一。

5.栈和队列的区别和内存结构★★★

- 逻辑结构：FIFO，LIFO
- 存储结构：栈，数组，连续；队列，数组和链表，链队可以离散

6.有一个循环队列Q，里面的编号是0到n-1，头尾指针分别是f，p，现在求Q中元素的个数？★★

- 初始时头尾指向0，p>f, p-f ;
- p=f,空;
- p<f,n+p-f;

7.如何区分循环队列是队空还是队满？★★★

- 使用一个额外的变量;
- 牺牲一个存储单元.

8.堆、大顶堆、小顶堆实现及应用 ★★

- **概念**： 逻辑上特殊的树形结构。
- **性质**： 对于堆中的每个节点 i，父节点的值$<=$ 其子节点的值（小顶堆），$>=$（大顶堆）
- **应用**：
    - 优先队列;
    - 堆排序，选出最大/最小k个值（维护一个大小为k的堆）
    - 迪杰斯特拉算法单源最短路径;
    - 构造哈夫曼树时选择使用频率最低的字符

9. 哈希表的概念、构造方法、哈希有几种类型？哈希冲突的解决办法？★★★★

- **概念**：把key值映射到表中一个位置存放，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。
- **构造方法**（选取散列函数方法）：
    - 直接定址法
    - 除留余数法（除以一个素数p）
    - 平方取中法
- **冲突处理**：
    - 开放定址法
    - 拉链法
    - 线性探测/二次探测/伪随机序列再散列法

- **效率**：
    1. 散列函数选取
    2. 处理冲突的方法
    3. 散列表的装填因子。散列表的装填因子定义为：α = 填入表中的元素个数 / 散列表的长度

10. 判断链表是否有环（非常重要！）★★★★★★★

11. 平衡二叉树、二叉排序树、完全二叉树、二叉搜索树的区别及如何构造★★★

- **平衡二叉树**：左右子树高度差不超过1，平衡因子 = 左 - 右；
- **二叉排序树**：左 < 根 < 右；
- **二叉搜索树**：二叉排序树 + 失败节点（虚拟）
- **构造**：递归的插入新节点，保证左根右，平衡二叉树还要保证平衡因子 = -1/0/1，计算：设置一个变量求子树高度，相减即可；
- **完全二叉树和满二叉树**：对树中的结点按从上至下、从左到右的顺序进行编号，编号为i的结点在满二叉树中的位置相同。性质：除了最后一层外，其他层的节点都满，且最后一层的节点都尽可能地靠左排列。

12. 如何由遍历序列构造一颗二叉树？/已知先序序列和后序序列能否重现二叉树？（笔试经常考）★★★

- 先序+中序，中序+后序唯一确定；不行
- 先序+中序：1. 先序第一个是根节点；2. 找到根节点元素在中序序列中的位置，划分左右子树；3. 对左右子树递归执行该过程
- 中序+后序：1. 后序遍历序列最后的节点是根节点；2. 找到根节点元素在中序序列中的位置，划分左右子树；3. 对左右子树递归执行该过程

13. B树是什么?在数据库中有什么应用？（B数和B+树的区别）★★★

特征	B树	B+树
主要目的	一次查多个关键字，减少树的高度和磁盘I/O次数	同B树，但优化了范围查询和大数据量处理
根节点	至少有两个指针	至少有两个指针
节点存储	每个节点存储键和数据	只有叶子节点存储数据，内部节点只存储键
关键字排序	节点内关键字以升序排列，支持二分查找	内部节点和叶子节点的关键字以升序排列
节点关键字数	每个节点有M-1个key	有n棵子树的结点中含有n个关键字，仅用于索引
子节点指针	位于M-1和M key的子节点值位于对应Value之间	所有叶子结点包含全部关键字信息及记录指针
叶子节点特征	位于同一层	依关键字大小顺序链接，并增加链指针便于区间查找
链接	无特殊链接，各节点独立	叶子节点间通过指针链接形成链表

特征	B树	B+树
搜索效率	数据可在内部节点和叶子节点找到，路径长度可能不同	每次搜索路径长度相同，搜索性能稳定
空间利用率	由于存储数据，空间利用率较低	内部节点不存储数据，只存储键，空间利用率高
特性和用途	结构简单，适用于简单查找和插入	优于B树于处理范围查询和大数据量的数据库系统

。

14.红黑树★★★

- 左根右，根叶黑，不红红，黑路同

15.二分搜索和单纯的线性搜索的区别/时间复杂度★★★

- 二分：O(logn)，要求序列有序，待查元素和序列中间元素比较，每次排除一半
- 线性：从头到jio

16.插入排序、希尔排序、选择排序、冒泡排序、归并排序、快速排序（必考）、堆排序、基数排序等排序算法的基本思想是什么？时间复杂度？是否稳定？给一个例子，问冒泡和快速排序在最坏的情况下比较几次？（排序必考）★★★★★★

排序算法	描述	时间复杂度	稳定性
直接插入排序	从有序序列中查找待插入元素的位置，逐步将元素插入到已排序的序列中。	O(n^2)	稳定
希尔排序	改进的插入排序，将原列表分割成多个子序列，对每个子序列应用直接插入排序。	O(n)~O(n^2)	不稳定
选择排序	从未排序的序列中找到最小（大）元素，存放到排序序列的起始位置。	O(n^2)	不稳定
冒泡排序	重复访问序列，比较两个元素，如果顺序错误则交换它们。	O(n^2)	稳定
归并排序	将已有序的子序列合并，得到完全有序的序列。	O(nlogn)	稳定
快速排序	通过基准元素将数组分为两部分，递归地排序两部分。	O(nlogn)	不稳定

排序算法	描述	时间复杂度	稳定性
堆排序	利用堆数据结构进行的一种排序算法。	O(nlogn)	不稳定
基数排序	根据数字的有效位或基数进行排序，从最低位开始排序。	O(nk)	稳定

- 具体见 https://blog.csdn.net/m0_73900674/article/details/132418128

17.最小生成树和最短路径用什么算法来实现？（迪杰斯特拉、弗洛依德、普利姆、克鲁斯卡尔）算法的基本思想是什么？算法的时间复杂度？如何进行优化？（必考）★★★★★★★

- 最小生成树：
  - 普利姆(Prim)：从一个顶点开始，逐渐增加新的边和顶点，直到找到最小生成树。时间复杂度O(V^2)，可以通过优先队列优化到O(E log V)。
  - 克鲁斯卡尔(Kruskal)：按边的权重顺序选择边，保证不形成环，直到连接所有顶点。时间复杂度O(E log E)或O(E log V)。
- 最短路径：
  - 迪杰斯特拉(Dijkstra)：适用于没有负权边的图，通过逐步确定最短路径的长度。时间复杂度O(V^2)，可以通过优先队列优化到O(E + V log V)。
  - 弗洛依德(Floyd)：计算图中所有顶点对的最短路径。时间复杂度O(V^3)，适用于边稠密的图。

18.邻接表和邻接矩阵（如何存储大数据）★

- 邻接矩阵：适用于稠密图，使用二维数组存储图中的边信息。
- 邻接表：适用于稀疏图，使用链表数组存储图中的边信息。

19.介绍一下深度优先搜索和广度优先搜索是如何实现的？★★★

- DFS(node)
  1. 标记node为已访问
  2. 对于node的每个未访问的neighbor
     a. 调用DFS(neighbor)
- DFS_Stack(start)
  1. 将起始节点start入栈
  2. 当栈不为空时
     a. 弹出栈顶元素current
     b. 如果current未被访问
     i. 标记current为已访问
     ii. 将current的所有未访问neighbor入栈
- BFS(start)
  1. 创建一个队列Q
  2. 将起始节点start入队并标记为已访问
  3. 当队列Q不为空时
     a. 从Q中出队一个元素current

b. 对于current的每个未访问的邻居neighbor
i. 将neighbor入队并标记为已访问

20.介绍一下字符串匹配算法：朴素的匹配算法和KMP算法。（如何实现要会用语言描述）★★★

计算机网络

1. OSI和TCP/IP模型各个层之间的协议和功能★★★★★★（必考）

```
- OSI模型：
  - 应用层：HTTP，FTP，SMTP
  - 表示层：SSL，TLS
  - 会话层：NetBIOS
  - 传输层：TCP，UDP
  - 网络层：IP，ICMP，ARP
  - 数据链路层：Ethernet，PPP
  - 物理层：Ethernet，DSL
- TCP/IP模型：
  - 应用层：HTTP，FTP，SMTP
  - 传输层：TCP，UDP
  - 网际层：IP，ICMP，ARP
  - 网络接口层：Ethernet，PPP
```

2.计算机网络为什么要分层？优点？ ★★★

```
- 模块化设计：简化设计和实现。
- 可扩展性：方便添加新功能。
- 互操作性：实现设备和软件的互通。
- 故障隔离：便于定位和解决问题。
```

3.简述一下层次路由的原理（叙述一下与自治系统相关的内部网关协议和外部网关协议）单工、半双工、全双工通信？★★

层次路由的原理：
层次路由是一种路由策略，通过将网络划分为不同的层次结构来管理路由信息，从而减少路由表的规模和路由信息的传播范围，提高路由的效率和可扩展性。通常，层次路由采用分层次的路由协议，比如内部网关协议（IGP）用于自治系统内部的路由，外部网关协议（EGP）用于不同自治系统之间的路由。

内部网关协议（IGP）：
内部网关协议用于自治系统内部的路由信息交换，常见的IGP协议有OSPF（Open Shortest Path First）和IS-IS（Intermediate System to Intermediate System）。这些协议根据网络拓扑和链路状态计算最短路径，并维护路由表。

外部网关协议（EGP）：
外部网关协议用于不同自治系统之间的路由信息交换，常见的EGP协议有BGP（Border Gateway Protocol）。

BGP负责在不同自治系统之间传递路由信息，通过策略路由选择最佳的路径，以实现自治系统间的互联互通。

单工、半双工、全双工通信：

- **单工通信**：通信双方只能在一个方向上传输数据，不能同时进行双向通信。类似于广播电台只能发送信息而无法接收信息的情况。
- **半双工通信**：通信双方可以在不同的时间段内进行数据传输，但不能同时进行双向通信。类似于对讲机，同一时间内只有一方能发送消息，另一方只能接收消息。
- **全双工通信**：通信双方可以同时进行双向数据传输，既可以发送数据也可以接收数据。类似于电话通信，双方可以同时说话和听取对方的话语。

4.协议三要素？（语法、语义、时序）★★

协议的三要素指的是：

- **语法**（Syntax）：指定了数据的格式和结构，即数据在传输过程中的编码规则和格式要求。它决定了数据的组成和表示方法，确保通信双方能够正确理解和解析数据。
- **语义**（Semantics）：定义了数据的含义和解释规则，即数据的意义和传输的信息。它规定了数据的解释方式，确保通信双方能够理解数据所传达的意思。
- **时序**（Timing）：规定了数据的发送顺序和时机，即数据的传输时间和顺序要求。它决定了数据的发送和接收时机，确保通信双方能够在正确的时间接收和处理数据。

这三个要素共同构成了协议的基本规范和约定，保证了通信双方能够有效地进行数据交换和通信。

5.香农公式？信道容量含义？带宽增加，信道容量怎么变？香农公式的前提条件？

香农公式（Shannon's Formula）：
香农公式用于计算数字通信中的理论最大传输速率，其公式为：
$$ C = B \cdot \log_2(1 + S/N) $$
其中，（C）是信道的最大传输速率（信道容量），单位为比特每秒（bps）；（B）是信道的带宽，单位为赫兹（Hz）；（S/N）是信号与噪声的比值。

信道容量含义：
信道容量指的是在给定的信道带宽和信噪比条件下，信道所能传输的最大信息量或数据速率。它表示了在理想条件下，信道所能达到的最高传输速率。

带宽增加，信道容量怎么变？
如果信道的带宽增加，即（B）增大，那么根据香农公式，信道的最大传输速率（C）也会增加。这是因为带宽的增加意味着信道能够传输更多的频率成分，从而提高了信道的传输能力和传输速率。

香农公式的前提条件：

1. 噪声是加性高斯白噪声，且信号与噪声是独立的。
2. 信号在信道中传输的过程中不受到衰减、失真或干扰。
3. 信道的带宽足够宽，能够容纳所有频率成分。
4. 信号的功率较大，可以忽略噪声对信号的影响。

6.简述一下CSMA/CD协议★★★★

CSMA/CD（Carrier Sense Multiple Access with Collision Detection）是一种用于以太网中的多路访问协议，用于解决多个设备共享同一信道时可能发生的冲突问题。

工作原理：

1. **载波侦听（Carrier Sense）：** 网络中的设备在发送数据前会先监听信道，检测是否有其他设备正在传输数据。如果信道空闲，即没有其他设备在发送数据，那么设备可以开始发送数据；如果信道忙碌，则设备等待一段随机时间后再次进行侦听。
2. **多路访问（Multiple Access）：** 多个设备共享同一信道，任何设备都可以在信道空闲时发送数据。
3. **碰撞检测（Collision Detection）：** 如果两个设备同时发送数据，它们的数据包将在信道中发生碰撞。设备在发送数据的同时也在监听信道，一旦检测到碰撞，设备会立即停止发送数据，并发送一个特殊的信号（碰撞检测信号）通知其他设备发生了碰撞。之后，设备会等待一段随机时间后重新发送数据。

CSMA/CD协议能够有效地解决多个设备竞争信道资源时可能发生的冲突问题，提高了以太网的传输效率和可靠性。然而，随着以太网技术的发展，现代以太网通常采用了全双工通信和交换机技术，CSMA/CD协议已经不再被广泛应用。

7.TCP和UDP的异同点★★★★★★（必考）

- 相同点
  - 工作在传输层，基于网络层的ip协议
  - 端口号：它们使用端口号来区分不同的应用程序或进程，确保数据准确送达目标应用。
- 不同点
  - 连接方式：
    - TCP是面向连接的协议，它在数据传输前需要建立连接，数据传输结束后关闭连接。
    - UDP是无连接的协议，发送数据之前不需要建立连接。
  - 可靠性：
    - TCP提供可靠的数据传输服务，通过序号、确认应答、重传等机制保证数据完整性和顺序。
    - UDP提供尽最大努力的传输，不保证数据的可靠性，可能出现丢包、错误或数据顺序错乱。
  - 传输效率：
    - TCP由于需要进行连接管理、状态维护、流量控制和拥塞控制等操作，相比UDP有更多的开销，传输效率较低。
    - UDP由于不进行可靠性保证，开销小，传输效率高
  - 使用场景：
    - TCP适用于需要可靠数据传输的场景，如网页浏览、文件传输、电子邮件等。
    - UDP适用于对传输效率和实时性要求高的场景，如在线视频播放、实时游戏、VoIP等。

8. 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）

三次握手过程：

1. 客户端发送SYN报文段，进入SYN_SENT状态。
2. 服务器收到SYN报文段，回复SYN+ACK报文段，进入SYN_RCVD状态。
3. 客户端收到ACK报文段，发送确认报文段，完成三次握手，连接建立。

四次挥手过程：

1. 客户端发送FIN报文段，进入FIN_WAIT_1状态。
2. 服务器收到FIN报文段，回复ACK报文段，进入CLOSE_WAIT状态。
3. 服务器发送FIN报文段，进入LAST_ACK状态。
4. 客户端收到FIN报文段，回复ACK报文段，进入TIME_WAIT状态。

5. 服务器收到ACK报文段，完成四次挥手。

- 三次握手确保双方都能接收和发送数据，避免了已失效的连接请求报文段突然又传送到了服务端
- 采用两次握手存在的问题是，无法确认服务端是否接收到了客户端的连接请求，容易导致已失效的连接请求报文段被服务端误认为是新的连接请求

9. 介绍下TCP和UDP协议的特点、头部结构★★★★★

TCP特点：

- 面向连接：建立可靠的连接，保证数据传输的可靠性。
- 可靠性：通过重传机制、序号和确认应答实现数据可靠传输。
- 慢启动和拥塞控制：控制数据的发送速率，避免网络拥塞。
- 高效性：通过滑动窗口、拥塞控制、流量控制等机制实现高效的数据传输。

UDP特点：

- 无连接：不需要建立连接，直接发送数据。
- 不可靠性：无确认机制，不保证数据传输的可靠性。
- 简单快速：头部开销小，传输效率高。
- 适用于实时应用：如音频、视频等实时传输的应用场景。
  以下是TCP和UDP头部结构的比较：

字段	TCP	UDP
源端口	占2字节	占2字节
目的端口	占2字节	占2字节
序号	占4字节	无序号
确认号	占4字节	无确认号
数据偏移	占4位，指示TCP头部长度	无
保留位	占3位，保留未来使用	无
标志位	SYN、ACK、FIN、RST等	无
窗口大小	占2字节	无
校验和	占2字节	占2字节
紧急指针	占2字节	无
选项	可变长，最多占40字节	无

- TCP头部相对于UDP头部而言更为复杂，其中包含了序号、确认号、窗口大小、校验和、紧急指针等字段，以实现TCP的可靠传输和流量控制机制。
- TCP头部中的标志位用于指示TCP连接的状态和控制连接的建立、终止等行为，如SYN用于建立连接，ACK用于确认，FIN用于结束连接等。
- UDP头部相对较简单，只包含了源端口、目的端口、长度、校验和等字段，不包含序号和确认号等用于可靠传输的机制，因此UDP的传输速度比TCP更快，但不具备TCP的可靠性。
- TCP头部中的选项字段是可变长的，最多占用40个字节，而UDP头部则没有选项字段，更为简洁。

- TCP头部中的窗口大小字段用于流量控制，表示接收方还有多少空间可以接收数据，而UDP头部没有窗口大小字段。

10. 简述下TCP建立连接的过程,TCP如何保证可靠传输？★★★★★★

- TCP建立连接的过程：

    1. 客户端发送SYN报文段，请求建立连接。
    2. 服务器收到SYN报文段后，回复SYN+ACK报文段表示同意建立连接。
    3. 客户端收到SYN+ACK报文段后，发送ACK报文段，连接建立成功。

- TCP保证可靠传输的机制包括：

    - 序号与确认：每个数据包都有序号和确认号，接收方可以根据序号和确认号进行数据包的重组和确认。
    - 超时重传：发送方在一定时间内未收到确认，会重新发送数据包。
    - 滑动窗口：发送方和接收方维护滑动窗口，控制数据流量，避免网络拥塞。
    - 拥塞控制：通过慢启动、拥塞避免等机制控制发送速率，避免网络拥塞导致丢包。

11. 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（Slow Start）：发送方初始将拥塞窗口设为1，然后每收到一个ACK，拥塞窗口就加倍，直到达到慢开始阈值。
- 拥塞避免（Congestion Avoidance）：当拥塞窗口达到慢开始阈值后，每收到一个ACK，拥塞窗口增加1，实现线性增长。
- 快重传（Fast Retransmit）：当发送方连续收到三个重复的ACK时，认为出现了丢包，立即重传对应的数据包。
- 快恢复（Fast Recovery）：在快重传后，拥塞窗口大小减半，然后进入拥塞避免状态。

12. TCP的快速重传机制★★★★

- TCP的快速重传机制指的是在发送方连续收到三个重复的ACK时，认为出现了丢包，立即重传对应的数据包，而不必等待超时。
- 更快地恢复丢失的数据包，提高传输效率。

13. 流量控制和拥塞是什么关系？★★★★

- 流量控制：用于控制发送方发送数据的速率，确保接收方来得及处理数据，避免数据溢出。通过滑动窗口机制实现。
- 拥塞控制：用于控制网络中的数据流量，防止网络拥塞并保证网络的稳定性。通过慢开始、拥塞避免、快重传等机制实现。
- 都是为了避免数据丢失和网络拥塞
- 流量控制关注的是点对点通信的速率匹配，确保发送方不会以高于接收方处理速度的速率发送数据。
- 拥塞控制则关注整个网络的拥塞情况，通过调整发送速率以及对网络拥塞的响应来维持网络的稳定性和性能。

14.两个服务器之间网络已经联通，却收不到彼此的UDP报文原因★★★★

- 防火墙或网络设备屏蔽了UDP报文的传输。
- 服务器的UDP端口未正确打开或未正确配置。

- 网络拓扑问题，例如路由器配置错误导致UDP数据包无法正确路由到目标服务器。
- 网络质量问题，例如丢包严重或网络延迟过高导致UDP数据包丢失。

15.地址解析协议（ARP）和RARP协议★★

- ARP（Address Resolution Protocol）：用于将IP地址解析为MAC地址的协议。当主机知道目标IP地址但不知道其MAC地址时，会发送ARP请求广播来获取目标MAC地址。
- RARP（Reverse Address Resolution Protocol）：与ARP相反，用于将MAC地址解析为IP地址的协议。当主机只知道自己的MAC地址但不知道自己的IP地址时，会发送RARP请求来获取自己的IP地址。

19.简述下DNS域名解析的过程。★★★★★

1. **递归解析**：
    - 客户端发起DNS查询请求到本地DNS服务器，并设置递归查询标志。
    - 本地DNS服务器收到请求后，如果有缓存或已知的DNS记录，则直接返回结果给客户端；否则，本地DNS服务器会向根域名服务器发起DNS查询请求。
    - 根域名服务器返回对应顶级域名服务器的IP地址给本地DNS服务器。
    - 本地DNS服务器继续向顶级域名服务器发起DNS查询请求。
    - 顶级域名服务器返回对应的权威域名服务器的IP地址给本地DNS服务器。
    - 本地DNS服务器继续向权威域名服务器发起DNS查询请求。
    - 权威域名服务器返回对应域名的IP地址给本地DNS服务器，本地DNS服务器将结果返回给客户端。
2. **迭代解析**：
    - 客户端发起DNS查询请求到本地DNS服务器，但不设置递归查询标志。
    - 本地DNS服务器收到请求后，如果有缓存或已知的DNS记录，则直接返回结果给客户端；否则，本地DNS服务器会向根域名服务器发起DNS查询请求，但此时不设置递归查询标志。
    - 根域名服务器返回对应顶级域名服务器的IP地址给本地DNS服务器。
    - 本地DNS服务器再向顶级域名服务器发起DNS查询请求，但同样不设置递归查询标志。
    - 顶级域名服务器返回对应的权威域名服务器的IP地址给本地DNS服务器。
    - 本地DNS服务器再向权威域名服务器发起DNS查询请求，并最终将结果返回给客户端。

操作系统

1. 操作系统的特点？功能？★

- **特点：**
    - 控制计算机硬件资源
    - 提供用户与计算机硬件的接口
    - 管理和组织计算机系统的软硬件资源
    - 实现对计算机系统的抽象
- **功能：**
    - 进程管理
    - 存储管理
    - 文件管理
    - 设备管理
    - 用户接口

2. 中断和系统调用的区别★★★

- **中断：**
  - 是硬件发起的，可以打破CPU的正常执行流程
  - 可能由外部设备的请求、错误或软件指令触发
  - 会引起当前程序的暂停，执行相应的中断服务程序
- **系统调用：**
  - 是软件发起的，通过软中断或特殊的机器指令实现
  - 提供用户空间程序访问内核功能的接口
  - 通常用于请求操作系统提供服务，如文件操作、进程控制等

3. 进程、线程的概念以及区别？进程间的通信方式？★★★★★★

- **进程：**
  - 是程序的执行实例，有独立的地址空间和资源
  - 操作系统进行资源分配和调度的基本单位
- **线程：**
  - 是进程内的一个执行单元，共享相同的地址空间和资源
  - 通过调度器分配CPU时间，实现并发执行
- **区别：**
  - 进程拥有独立的地址空间，线程共享所属进程的地址空间
  - 进程间切换开销较大，线程间切换开销较小
- **进程间通信方式：**
  - 管道、信号、消息队列、共享内存、套接字等

4. 进程有哪几种状态，状态之间的转换、进程调度策略？★★★★

- **进程状态：**
  - 就绪态：已经具备运行条件，等待分配CPU
  - 运行态：正在CPU上执行
  - 阻塞态：由于等待某事件发生而暂停执行
- **状态转换：**
  - 就绪态 → 运行态：进程被调度执行
  - 运行态 → 阻塞态：等待某事件发生
  - 阻塞态 → 就绪态：事件发生，等待调度
- **进程调度策略：**
  - 先来先服务（FCFS）
  - 短作业优先（SJF）
  - 优先级调度
  - 时间片轮转

5. 读写者问题是用进程实现的还是线程实现的？简述读者写者问题。

- 读写者问题是可以用进程或线程来实现的。
- 问题描述：多个读者可以同时读取共享资源，但写者在写入时需要独占资源，且写者与写者、写者与读者之间都不能并发访问。
- 解决方案：使用信号量或互斥锁来实现对共享资源的访问控制，保证写操作的互斥性，读操作的并发性。

6.什么是死锁？死锁产生的四个必要条件？如何预防死锁？★★★★★★

- 死锁是指两个或更多的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力干涉那它们都将无法继续执行下去。
- 死锁产生的四个必要条件：
    - 互斥条件：一个资源每次只能被一个进程使用。
    - 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。
    - 不剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺。
    - 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。
- 预防死锁的基本方法：
    - 破坏互斥条件：这意味着允许多个进程同时访问某一资源，但这在很多情况下是不可能的。
    - 破坏请求与保持条件：一次性申请所有的资源，这样就不会在执行过程中因申请资源而阻塞。
    - 破坏不剥夺条件：允许剥夺进程占有的资源，使其释放后供其他进程使用。
    - 破坏循环等待条件：实施资源排序和顺序申请策略，避免循环等待。
    - 资源分配图：在资源分配图中，如果存在一个环路，则表明系统处于死锁状态。这种方法适用于单个资源的每类只有一个实例的情况。
    - 银行家算法：对于每类资源有多个实例的情况，可以使用银行家算法进行死锁检测。银行家算法通过模拟资源分配和回收，检查系统是否存在安全序列，如果不存在，则系统可能处于死锁状态。

7. 哲学家进餐有哪些实现方式？★★★★

哲学家进餐问题的实现方式主要包括：

- **互斥锁（Mutex）**：为每根筷子分配一个互斥锁，哲学家必须先锁定两边的筷子才能吃饭，用完后释放。
- **信号量（Semaphores）**：使用信号量控制同时拿起筷子的哲学家数量，避免死锁。
- **条件变量（Condition Variables）**：哲学家通过条件变量等待两边筷子都可用时，才开始吃饭。
- **资源分级**：给筷子分配优先级，哲学家必须按照一定的顺序拿起筷子。

8. 简述下银行家算法★★★★

银行家算法是一种避免死锁的算法，用于资源分配的安全性检查。它模拟银行家分配贷款的方式，确保分配后系统处于安全状态。算法过程包括：

1. **检查请求**：当一个进程请求资源时，系统检查分配后是否还有足够的资源满足其他进程的最大需求。
2. **尝试分配**：如果可以，系统尝试分配资源。
3. **回滚**：如果分配后系统不安全，则不分配资源并让进程等待。

9. 介绍下几种常见的进程调度算法及其流程

- **FCFS（先来先服务）**：按照请求的顺序进行服务。简单，但可能导致短任务等待时间长。
- **SJF（最短作业优先）**：优先调度最短的任务。减少了平均等待时间，但可能饿死长任务。
- **剩余短作业优先**：考虑了任务的剩余执行时间，动态调整。
- **优先级调度**：基于任务的优先级进行调度，高优先级的任务先执行。
- **轮转法（Round Robin）**：每个进程分配一定时间片，轮流执行，适合时间共享系统。
- **多级反馈队列**：结合了多种方法，有多个队列，根据任务的行为动态调整其所在的队列和优先级。

10. 分页的作用，好处？和分段有什么区别？★★★

- **分页**：将物理内存分成固定大小的块。每个程序分成相同大小的页，提高了内存的利用率，简化了内存管理。
- **分段**：将程序分成意义上的段，如代码段、数据段。分段更符合程序的逻辑结构，但实现相对复杂。
- **区别**：分页是物理上的划分，页大小固定，对程序员透明；分段是逻辑上的划分，段的长度不固定，每个段有其意义。

11. 内存分配有哪些机制？（JVM的内存管理及垃圾回收算法）★★★

内存分配机制包括静态分配、动态分配。JVM内存管理涉及堆、栈、方法区等。垃圾回收算法包括标记-清除、复制、标记-整理、分代收集等。

12. 什么是虚拟内存？什么是共享内存？★★★

- **虚拟内存：**
  - 是一种利用硬盘空间来扩展内存的技术
  - 允许程序访问超出物理内存容量的内存空间
  - 将内存空间划分成固定大小的页面，将不常用的页面存储到磁盘上，从而释放内存空间
- **共享内存：**
  - 是一种允许多个进程访问同一块物理内存的机制
  - 进程间可以通过共享内存直接进行数据交换，而无需通过内核
  - 通常用于提高进程间通信的效率

13. 有哪些页面置换算法？★★★★

- **页面置换算法：**
  1. 最佳（Optimal）置换算法
  2. 先进先出（FIFO）置换算法
  3. 最近最久未使用（LRU）置换算法
  4. 最不经常使用（LFU）置换算法
  5. 时钟（Clock）置换算法
  6. 先进先出改进（Enhanced Second Chance）置换算法

14. 说一说操作系统中缓冲区溢出怎么处理★★★

- **缓冲区溢出处理：**
  - **栈溢出和堆溢出：** 常见于程序运行时，可以通过编程技巧和代码审查来避免，如合理控制递归深度、动态分配内存时检查边界等。
  - **缓冲区溢出攻击：** 攻击者利用程序中存在的漏洞，通过向缓冲区输入超出其边界的数据，覆盖程序的关键数据或代码，以执行恶意代码。防范措施包括使用安全编程技术、输入验证、代码审查、使用编译器提供的安全选项等。

15. 磁盘调度算法以及磁盘空间存储管理？★★★★

- **磁盘调度算法：**
  - 先来先服务（FCFS）调度算法
  - 最短寻道时间优先（SSTF）调度算法
  - 扫描（SCAN）调度算法
  - 循环扫描（C-SCAN）调度算法
  - LOOK 调度算法

    - C-LOOK 调度算法
- **磁盘空间存储管理：**
    - 连续分配：将磁盘空间划分成固定大小的分区，每个文件占据连续的分区
    - 链式分配：将文件分成多个块，每个块可分配到磁盘上的任意空闲块中，通过链表进行管理
    - 索引分配：为每个文件建立索引表，将文件的数据块存储在磁盘上不连续的位置，并通过索引表进行管理
    - 位示图法

16. 文件系统中文件是如何组织的？★★

- **文件组织方式：**

    - 顺序文件组织：文件中的记录按顺序存放，适用于顺序访问。这种方式简单直观，但不适合频繁的插入和删除操作。

    - 索引文件组织：文件中的记录通过索引表进行管理，支持随机访问，适用于大文件和频繁随机访问的文件。这种方式可以快速定位到特定的记录，但需要维护索引表，增加了一定的复杂性。

    - 索引顺序文件组织：将索引文件组织和顺序文件组织结合，适用于较大的文件，并支持随机访问。这种方式既保留了顺序文件组织的顺序访问优势，又具有索引文件组织的随机访问能力。

    - 链式文件组织：文件中的记录通过链表进行管理，适用于插入和删除频繁的文件。这种方式可以方便地进行记录的插入和删除，但访问特定的记录可能需要遍历链表，效率较低。

计算机组成原理

17. 冯诺依曼机的体系结构★★★★★

冯·诺依曼机是一种计算机结构，其特点包括：

- 存储程序：指令和数据存储在同一存储器中。
- 指令和数据用二进制表示。
- 由运算器、控制器、存储器、输入设备和输出设备组成。
- 指令执行按顺序进行，且能够被中断和跳转。
- 具有自动执行指令的能力。

18. 衡量计算机性能指标★★★★

计算机性能指标包括：

- 时钟频率：CPU每秒钟执行的时钟周期数。
- 吞吐量：单位时间内处理的任务数量。
- 响应时间：从请求任务到完成任务所需的时间。
- CPU利用率：CPU在单位时间内的工作时间占总时间的比例。
- 内存带宽：单位时间内从内存中读取或写入数据的速率。

19. 原码、反码、补码★★★

- 原码：用最高位表示符号，0表示正数，1表示负数。
- 反码：正数的反码与原码相同，负数的反码是对其原码逐位取反。
- 补码：正数的补码与原码相同，负数的补码是其反码加1。

20. 奇偶校验、汉明码校验，循环冗余校验★★

- 奇偶校验：通过在数据位中添加一位校验位，使得数据中1的个数为奇数或偶数，用于检测单比特错误。
- 汉明码校验：通过添加冗余位来检测和纠正多比特错误，具有更强的纠错能力。
- 循环冗余校验：通过除法运算产生余数进行校验，常用于检测数据传输中的错误。

21. 存储器的分类（RAM、DAM的区别）★★

存储器可分为RAM（随机存取存储器）和DAM（直接访问存储器）：

- RAM：数据读写速度快，但断电后数据丢失。
- DAM：数据读写速度相对较慢，但数据持久保存。

22. 段页式虚拟内存★★★

段页式虚拟内存将物理内存分为若干页框，逻辑地址空间划分为若干段，通过段表和页表实现地址转换，提供了更高的内存利用率和更好的地址保护。

23. CPU一个指令周期的流程是什么？★★★★★

CPU执行指令的周期包括：

1. 取指周期：从内存中读取指令到指令寄存器。
2. 执行周期：根据指令操作码执行相应操作，可能包括算术逻辑运算、数据传输等。
3. 访存周期：若需要访问内存，则进行读取或写入操作。
4. 写回周期：将结果写回寄存器或内存。
   这些周期依次循环，直至指令执行完毕。

24. 总线通讯的四种方式★★

总线通讯的四种方式包括：

- 单工通信：数据只能单向传输。
- 半双工通信：数据可以双向传输，但不能同时进行。
- 全双工通信：数据可以双向传输，并且可以同时进行。
- 多主机通信：多个主机共享同一总线进行通信。

项目经历相关（LSTM）

1. **模型的代码有自己实现的部分吗？还是网上的代码？**

- 本项目的代码包含了原创的部分，包括数据预处理、模型训练、参数调优、以及使用2023年数据进行回测的过程。
- 可能有类似的在线资源，但具体的参数选择、数据处理和模型架构的调整都反映了个人理解和实验。

2. **Pytorch搭建网络的过程？参数传递与梯度下降的回传？**

- **搭建网络过程**：
  - 在PyTorch中，搭建网络首先涉及定义一个模型类，该类继承自`nn.Module`。
  - 在这个类中，定义了网络层及其连接方式。

- **参数传递**： 参数传递发生在`forward`方法中，它指定了数据通过网络的前向传播路径。
- **梯度下降的回传**：通过调用`loss.backward()`自动完成，利用了PyTorch的自动梯度计算功能来计算损失函数对每个参数的梯度，并使用优化器自动更新模型参数。

3. 模型/算法具体怎么使用在任务上的?

- 在本项目中，LSTM模型用于预测股票市场指数。
- 具体来说，模型接收过去一定时间窗口内的股票价格和交易量等特征作为输入，学习这些数据与未来某一时刻的股价之间的关系，用于预测该时刻的股价。

4. 模型/算法的原理是什么？描述一下模型/算法的过程?

- LSTM（长短期记忆网络）是一种特殊的RNN（循环神经网络），设计用来解决传统RNN在长序列数据处理上的短期记忆问题。
- LSTM通过引入三种门（输入门、遗忘门、输出门）来控制信息的流入、保留和流出，从而能够更好地捕捉长距离依赖关系。

5. 为什么使用A模型不使用B模型?

- LSTM被选择用于此项目是因为其优越的长序列数据处理能力，特别是在处理具有时间依赖性的股票市场数据时。
- 相比其他模型如简单的全连接网络或基本的RNN，LSTM能更有效地捕捉时间序列数据中的长期依赖关系。

6. A任务的损失函数是什么？为什么不用其它损失函数?

- 在本项目中，损失函数使用的是均方误差（MSE），它是预测值和真实值差异平方的平均值。
- MSE是回归任务中常用的损失函数，因为它能直观地反映预测误差的大小，并且对大误差赋予更大的惩罚，有助于模型更精确地拟合数据。

7. 针对A模型有创新改进的地方吗?

- 参数调优、特定于任务的数据预处理策略，或者是模型结构的微调。
- 在模型架构、训练过程、或数据处理等方面的特殊设计和优化。

8. 描述一下算法/模型的训练过程?

- 通过数据预处理和归一化来准备训练数据，
- 构造时间序列数据以适应LSTM模型的输入需求。
- 定义模型架构和参数，选择适当的损失函数和优化器。
- 训练阶段，模型在训练集上进行多次迭代学习，每次迭代中，模型预测输出，计算损失，并通过梯度下降更新权重。
- 包括使用验证集来调整超参数以防过拟合。
- 使用测试集或独立数据集评估模型性能。

9. 实验过程的数据集哪来的?

- 实验数据集来自于`yfinance`库，它提供了一个方便的API来下载雅虎财经上的股票市场数据。

- 项目中使用了2018年1月1日至2022年12月31日的股市数据进行模型训练，以及2023年的数据进行模型回测和评估。

10. 算法/模型在数据集上的效果怎么样?

- 根据项目描述，LSTM模型在道琼斯指数和上证综指的预测任务上表现良好。
- 道琼斯指数的预测准确性指标包括MAE、MSE、RMSE、R^2和MAPE分别为397.18、231303.40、480.94、0.836和0.0117；
  - 上证综指的对应指标为31.57、1521.68、39.01、0.890和0.01。

11. 知道A领域有哪些期刊吗?

- IEEE Transactions on Neural Networks and Learning Systems
- International Conference on Computer Vision
- Conference on Computer Vision and Pattern Recognition
- European Conference on Computer Vision
- International Journal of Computer Vision

简化版：

- 1. **代码实现**：项目结合个人编写与常规方法，特别体现在数据处理和模型训练上。
- 2. **PyTorch建模**：定义一个模型类，通过前向传播方法指定数据流动，利用自动梯度计算和优化器更新模型。
- 3. **模型应用**：LSTM模型用于预测股市指数，依据历史数据学习时间序列特征。
- 4. **原理**：LSTM通过特殊的门控制机制捕捉长期数据依赖，优于简单RNN。
- 5. **选择理由**：LSTM适合处理时间序列预测任务，因其能理解数据的长期依赖性。
- 6. **损失函数**：使用均方误差（MSE），适用于回归问题，能有效衡量预测误差。
- 7. **创新点**：可能在数据预处理和模型参数调优上有所创新。
- 8. **训练过程**：数据划分、归一化、模型训练（前向传播、损失计算、梯度下降、权重更新）。
- 9. **数据来源**：`yfinance`库下载的股市数据。
- 10. **效果**：模型在道琼斯和上证综指预测上表现良好，准确捕捉走势。
- 11. **期刊**：
  CVPR：
  ICCV：
  ECCV：
  TPAMI：
  IJCV：
- 12. **调了哪些参数**: 批大小，学习率和迭代次数

专利

```
attention: 主要是根据我个人的情况编写，仅供参考
```

1. 专利的主要特征：

- 特定电极的正极和负极分别与充电电路的信号输入端相连，其目的是当特定电路断开时，通过充电电路对延时电路的电容进行充电。
- 延时电路与控制电路相连，并且控制电路的输出端与水泵电机相连。
- 自动水泵控制器可以省去水塔或水箱的浮杆、浮球或探头等设备及安装与布线工作，避免因老化和故障维修带来的麻烦。
- 本发明的自动水泵控制器设计简洁，可控性高，成本低廉，操作方便，能有效降低维护成本。

2. 你在开发这个自动水泵控制器的过程中遇到了哪些技术挑战?

- 开发过程中，我遇到的一个主要技术挑战是如何准确地利用水的弱电解质特性来控制水泵。
- 需要确保电极能够稳定地检测到水的存在，并且在没有水时迅速断开电路。
- 为此，我进行了多次实验，优化了电极的设计，并测试了不同材质。此外，为了确保系统的可靠性和稳定性，我对充电电路和延时电路进行了详尽的调试，以实现精确的时间控制。

3. 水泵控制器的工作原理

- 水作为弱电解质的导电性特性。
    - 当水存在时，它能够闭合电路中的特定电极，从而启动充电电路。
    - 电极放置在一定高度，当水位下降至电极以下时，电阻急剧增大，触发延时电路开始对电容充电，进而激活控制电路，启动水泵电机。
    - 当水位上升至特定电极时，电路被断开，但由于延时电路的作用，水泵将继续工作直至设定时间结束。

论文

算法设计与实现

- 1. 算法设计原理：
    - 1. 由于是支架，所以需要保证规整性，实验用的是格子
    - 2. 非平面上，轴向等分并没有办法保证弧等分，因此对于圆柱体的部分来说，转换成极坐标计算，对于球体的小部分（人骨、关节）来说需要转换成球坐标。实验用的是圆柱体的部分。
- 2.算法实现：
    - 1. 坐标转换：用极坐标计算后转换成直角坐标。
    - 2. 迭代生成路径：根据计算出的坐标，迭代生成G代码（一种数控编程语言），控制打印头沿预定路径移动。这包括沿着圆柱体的长轴（Y轴）逐层打印，以及在每一层上循环打印每个预定的X-Z坐标。
    - 3. 打印层次控制：程序还考虑了多层打印，通过在每层打印完成后增加Z轴偏移（Z_up），以及在每层结束时通过特定的G代码指令移动打印头，确保多层结构的连续性和稳定性。