2024计算机保研真题与面试资料整理（自己整理）

原创 Better Rose 已于 2024-08-16 07:56:34 修改 阅读量2.5k 收藏 78 点赞数 34

分类专栏： 保研经验分享 文章标签： 面试 算法 职场和发展

保研经验分享 专栏收录该内容 7 篇文章 订阅专栏

目录

1 数据结构

1.1 考察范围

- 堆、栈、队列、链表等数据结构
- 树：红黑树、二叉树的各类分支等
- 图：欧拉图：哈密顿图
- 查找算法、哈希算法等

1.2 常见问题

> Q:数据结构的定义
> 逻辑结构 （集合、线性结构、树形结构、网状结构）、存储结构、数据的运算

> Q:数组与链表的区别
> 数组的存储空间是栈上分配的，存储密度大，当要求存储的大小变化不大时，且可以事先确定大小，宜采用数组存储数据；链表的存储空间是堆上动态申请的，当要求存储的长度变化较大时，且事先无法估量数据规模，宜采用链表存储；
> （1）数组在内存中连续； (2)使用数组之前，必须事先固定数组长度，不支持动态改变数组大小；(3) 数组元素增加时，有可能会数组越界；(4) 数组元素减少时，会造成内存浪费； （5）数组增删时需要移动其它元素；
> （1）链表采用动态内存分配的方式，在内存中不连续 (2)支持动态增加或者删除元素 (3)需要时可以使用malloc或者new来申请内存，不用时使用free或者delete来释放内存

> Q: 二叉搜索树的优缺点（查找可，但是维护不可）
> 优点：查找效率高，插入删除操作算法简单，时间复杂度与查找差不多
> 缺点：根据插入顺序的不同，二叉搜索树最坏情况下可能会退化为单支树，也就是所有的节点在同一侧,这时查找的时间复杂度就是O（N）
> 平衡二叉树：是严格平衡的BST（平衡因子不超过1）。那么查找过程与BST一样，只是AVL不会出现最差情况的BST(单支树)。因此查找效率最好，最坏情况都是O(logN)数量级的;
> 二叉平衡树的严格平衡策略以牺牲建立查找结构(插入，删除操作)的代价，换来了稳定的O(logN) 的查找时间复杂度
> 红黑树：排序二叉树有不平衡的问题，可能左子树很长但是右子树很短，造成查询时性能不佳（logn退化成n），完全平衡的二叉树能保证层数平均，从而查询效率高，但是维护又很麻烦，每次插入和删除有很大的可能要大幅调整树结构。红黑树就是介于完全不平衡和完全平衡之间的一种二叉树，通过每个节点有红黑两种颜色、从节点到任意叶子节点会经过相同数量的黑色节点等一系列规则，实现了【树的层数最大也只会有两倍的差距】，这样既能提高插入和删除的效率，又能让树相对平衡从而有还不错的查询效率。

> Q: 红黑树的概念与性质介绍
> 自平衡的二叉查找树，1.结点是红色或黑色。2.根结点是黑色。3.每个叶子结点都是黑色的空结点（NIL结点）。4 每个红色结点的两个子结点都是黑色。(从每个叶子到根的所有路径上不能有两相个连续的红色结点)5.从任一结点到其每个叶子的所有路径都包含相同数目的黑色结点 左右深度差一倍以内，查找时间复杂度O（logN）

Q：在一定量数据中查找数据的方法：
有序数组。这种方式的存储结构，优点是支持数据的随机访问，并且可以采用二分查找算法降低查找操作的复杂度。缺点同样很明显，插入和删除数据时，为了保持元素的有序性，需要进行大量的移动数据的操作。
二叉查找树。如果需要一个既支持高效的二分查找算法，又能快速的进行插入和删除操作的数据结构，那首先就是二叉查找树莫属了。缺点是在某些极端情况下，二叉查找树有可能变成一个线性链表。
平衡二叉树。二叉树表示不服，于是基于二叉查找树的优点，对其缺点进行改进，引入了平衡的概念。根据平衡算法的不同，具体实现有AVL树 /
跳跃表。跳跃表让已排序的数据分布在多层次的链表结构中，同样支持对数据进行高效的查找，插入和删除数据操作也比较简单，最重要的就是实现比较平衡二叉树真是轻量几个数量级。缺点就是存在一定数据冗余。

Q:并查集
并查集是一种树型的数据结构，用于处理一些不相交集合（disjoint sets）的合并及查询问题。
查找(Find)：查询两个元素是否在同一个集合中
合并(Union)：把两个不相交的集合合并为一个集合
路径压缩：用于提高并查集的效率

Q:哈夫曼树、前缀树（字典树）
是一类带权路径最短的树,用于通讯及数据传送中构造传输效率最高的二进制编码（哈夫曼编码），用于编程中构造平均执行时间最短的最佳判断过程；
前缀匹配，词频统计（trie树来压缩下空间，因为公共前缀都是用一个节点保存的）

Q:图的表示：邻接矩阵和邻接表
图的表示：邻接矩阵和邻接表，

Q:DFS：每一次都是沿着路径到不能再前进时才退回到最近的岔道口，并前往访问那些未访问分支顶点，直到遍历完整个图；
连通分量：在无向图中，如果两个顶点之间可以互相到达（可以是通过路径间接到达），那么称这两个顶点连通；
如果图G(V,E)的任意两个顶点都连通，则称G为连通图；否则，称G为非连通图，且称其中的极大连通子图为连通分量。
强连通分量：如果两个顶点可以各自通过一条有向路径到达另一个顶点，就称这两个顶点强连通。如果图G(V,E)的任意两个顶点都强连通，则称图G为强连通图；否则，称图G为非强连通图，且称其中的极大强连通子图为强连通分量。

Q:如何判断有向图是否有环?
拓扑排序：在一个有向图中，对所有的节点进行排序，要求没有一个节点指向它前面的节点。
先统计所有节点的入度，对于入度为0的节点就可以分离出来，然后把这个节点指向的节点的入度减一。一直做改操作，直到所有的节点都被分离出来。如果最后不存在入度为0的节点，那就说明有环，不存在拓扑排序
深度优先搜索：对一个图进行DFS， 则DFS的路径可以生成一个棵树；对于DFS树上的结点，如果存在指向祖先的边或指向自己边，则图存在回路；该方法的时间复杂度与DFS遍历的时间复杂度相同，为O （（V + E） O（V + E）O（V+E）

Q:欧拉图
通过图中所有边恰好一次且行遍所有顶点的回路称为欧拉回路，具有欧拉回路的无向图称为欧拉图；通过图中所有边恰好一次且行遍所有顶点的通路称为欧拉通路。具有欧拉通路但不具有欧拉回路的无向图称为半欧拉图。
1.对于无向图 G GG , G GG 是欧拉图当且仅当 G GG 是连通的且没有奇度顶点。
2.对于无向图 G GG , G GG 是半欧拉图当且仅当 G GG 是连通的且 G GG 中恰有 0 00 个或 2 22 个奇度顶点。
3.对于有向图 G GG , 是欧拉图当且仅当 G GG 的所有顶点属于同一个强连通分量且每个顶点的入度和出度相同。

Q:关键路径
AOE网，边表示活动的网络。AOE网中仅有一个入度为0的顶点，称为源点，仅有一个出度为0的顶点，称为汇点。从源点到汇点的所有路径中，具有最大路径长度的路径称为关键路径，而关键路径上的活动称为关键活动。完成整个工程的最短时间就是关键路径的长度，若关键活动不能按时完成，则整个工程的完成时间就会延长。只要找到了关键路径，就可以得出最短完成时间。

1.3 遇到的问答*

Q: 红黑树的应用，能是否实现过红黑树?
一般用于内存的排序，时间复杂度较低，为O（logn），STL中map和set内部使用的就是红黑树，Linux　进程调度 CFS
```
typedef int KEY_TYPE;
typedef struct _rbtree_node {
unsigned char color;
struct _rbtree_node *right;
struct _rbtree_node *left;
struct _rbtree_node *parent;
KEY_TYPE key;
void *value;
} rbtree_node;
typedef struct _rbtree {
rbtree_node *root;
rbtree_node *nil;//定义一个通用叶子节点，因为叶子节点是黑，并且左右子树都为空， 直接定义一个通用的叶子节点，如果节点是这个公共的叶子节点，那么就判断当前节点是叶子节点
} rbtree;
```

Q:哈希表，冲突怎么处理
是根据关键字和值（Key-Value）直接进行访问的数据结构。也就是说，它通过关键字 key 和一个映射函数 Hash(key) 计算出对应的值 value，然后把键值对映射到表中一个位置来访问记录，以加快查找的速度
1、开放地址法。基于该重复地址，对地址进行增量，线性探测再散列，二次探测再散列，伪随机探测再散列。

> 2、链地址法。将所有按给定的哈希函数求得的哈希地址相同的关键字存储在同一线性链表中，且使链表按关键字有序，链地址法适用于经常进行插入和删除的情况。
> 3、再哈希法，同时构造多个不同的哈希函数

> Q: 数据结构基本 算法？算法？快排和归并的思想？堆排序，树的介绍？比较快的排序算法？动态规划问题？
> 快速排序（不稳定）：基本思路为：在序列中任意选择一个元素作为中心，比它大的元素一律向后移动，比它小的元素一律向前移动，形成左右两个子序列，再把子序列按上述操作进行调整，直到所有的子序列中都只有一个元素时序列即为有序。
> 归并排序（稳定）：基本思想为：把两个或者两个以上的有序表合并成一个新的有序表。
> 堆排序（不稳定）：基本思想为：让此序列排列成完全二叉树，该树具有以下特点，该树中任意节点均大于或小于其左右孩子，此树的根节点为最大值或者最小值。
> 初始相对有序：直接插入，冒泡排序
> 最差情况：堆排序，归并排序
> 平均情况：堆排序，归并排序，快速排序大于希尔排序
> 小，不稳定的：直接排序；稳定的：冒泡排序

> Q：数据库的视图，数据库的物理与逻辑结构
> Q：讲一讲你最熟悉的算法？
> Q：数组与链表的区别？

#2 算法

2.1 考察范围

- 不会超过动态规划
- 算法复杂度的计算方法
- 具体的排序算法、数据结构算法如最小生成树、最短路径等、深搜与广搜、模式匹配等
- 网站刷题

2.2 常见问题

> Q:算法定义
> 算法是解题的方法和步骤，衡量算法的标准，有穷性、正确性、可行性、零个或多个的输入
> 一个或多个的输出；时间复杂度：程序大概要执行的次数，而非执行的时间（时间和机器有关）；空间复杂度：程序执行过程中大概所占用的最大内存空间；正确性、可读性、稳健性、高效率低存储

Q:各个排序算法的时间复杂度；快排和哈希排序，具体算法流程与步骤，算法的特点
直接插入排序（稳定）：基本思想为：将序列分为有序部分和无序部分，从无序部分依次选择元素与有序部分比较找到合适的位置，将原来的元素往后移，将元素插入到相应位置上。
折半插入排序（稳定）：基本思想为：对序列中的每个元素做以上处理，找到合适位置将其他元素后移进行插入。优点是：比较次数大大减少。
希尔排序（不稳定）：基本思想为：先将序列分为若干个子序列，对各子序列进行直接插入排序，等到序列基本有序时再对整个序列进行一次直接插入排序。
简单选择排序（不稳定）：基本思想为：将序列分为2部分，每经过一趟就在无序部分找到一个最小值然后与无序部分的第一个元素交换位置。
堆排序（不稳定）：基本思想为：让此序列排列成完全二叉树，该树具有以下特点，该树中任意节点均大于或小于其左右孩子，此树的根节点为最大值或者最小值。
冒泡排序（稳定）：基本思路为：每一趟都将元素进行两两比较，并且按照“前小后大”的规则进行交换。
快速排序（不稳定）：基本思路为：在序列中任意选择一个元素作为中心，比它大的元素一律向后移动，比它小的元素一律向前移动，形成左右两个子序列，再把子序列按上述操作进行调整，直到所有的子序列中都只有一个元素时序列即为有序。
归并排序（稳定）：基本思想为：把两个或者两个以上的有序表合并成一个新的有序表。
计数排序（稳定）；遍历三次O(n)

Q:迪杰斯特拉算法与弗洛伊德算法
带权图的单源最短路径问题，是一个贪心算法，只是Dijkstra算法更适合求图中给定两点的最短距离和路径，求每对顶点之间的距离计算量比较大；
Floyd算法是经典的动态规划算法，1) 设置顶点 vi 到顶点 vk 的最短路径已知为 Lik，顶点 vk 到 vj 的最短路径已知为 Lkj，顶点 vi 到 vj 的路径为 Lij， 则 vi 到 vj 的最短路径为：min((Lik+Lkj),Lij)，vk 的取值为图中所有顶点，则可获得 vi 到 vj 的最短路径
2)至于 vi 到 vk 的最短路径 Lik 或者 vk 到 vj 的最短路径 Lkj，是以同样的方式获得
时间复杂度：O(n²)，佛洛依德算法的时间复杂度为O(n^3)
空间复杂度：O (n)，O (n 2)

Q:最小生成树,prim和kusral
Prim:时间复杂度为O(n²)，首先我们确定一根节点，然后从此节点出发，利用贪心算法寻找权值最小的边。逐步扩大所含顶点的数目，直到遍及连通图的所有顶点。
Kusral，时间复杂度为O（eloge）（e为网中的边数），所以，适合于求边稀疏的网的最小生成树，令最小生成树的初始状态为只有n个顶点而无边的非连通图T=（V，{}），概述图中每个顶点自成一个连通分量。在E中选择代价最小的边，若该边依附的顶点分别在T中不同的连通分量上，则将此边加入到T中；否则，舍去此边而选择下一条代价最小的边。依此类推，直至T中所有顶点构成一个连通分量为止

Q:中序遍历二叉树，而且允许用递归来做；
运用栈来实现；

```
void INOrderTraverse(BiTNode *T){
if(T){
INOrderTraverse(T->lchild); // 遍历左孩子
displayElem(T); // 调用操作结点数据的函数方法
INOrderTraverse(T->rchild); // 遍历右孩子
}
```

```
// 如果结点为空，返回上一层
return;
}
```

2.3 遇到的问答*

Q: 在算法比赛中承担的角色（团队协作能力），编程能力如何，写过多少行代码，对于编程是否感兴趣?
1.作为比赛的队长，组织成员共同学习往年比赛算法题目进行训练，在正式比赛过程中，首先对于十一道题目进行题目的分配，因为比赛的题目并不是按照由易到难，在最开始我个人部分，完成一道模式匹配算法寻找字串在主串当中的位置，当然可以使用暴力求解，当时使用KMP算法可以进一步优化时间效率，另外，一到最短路径问题求1点到n点的最短路径，使用迪杰斯特拉算法，另外还有，比赛也涉及到背包问题的变形求解，团队最后完成七道题拿到银奖。
2.如果分为优秀，良好，一般，我会给自己良好，首先我熟练掌握 C,C++,java，python编程语言，并且可以运用这些语言去开发一个项目，比如，之前学习java课程结合数据库的内容完成跨组织人才管理系统的开发，完成3000+行代码并申请软著，在暑期企业实训的过程中运用C++开发了一个订餐管理系统，当然，我认为代码能力还是熟能生巧，比如之前学习的前端开发的基本知识可能长期没有训练就会十分生疏。另外，我对于编程还是十分感兴趣的，比如在算法比赛以及刷题过程中AC，比如在深度学习训练一个模型　并且不断调参优化得到更好的效果，都会感到有成就感与喜悦。

Q; 基本编程算法问题，讲述你最喜欢的算法?
动态规划(Dynamic Programming)算法的核心思想是：将大问题划分为小问题进行解决，从而一步步获取最优解的处理算法
2、动态规划算法与分治算法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。
3、与分治法不同的是，适合于用动态规划求解的问题，经分解得到子问题往往不是互相独立的。(即下一个子阶段的求解是建立在上一个子阶段的解的基础上，进行进一步的求解)
4、动态规划可以通过填表的方式来逐步推进，得到最优解。

Q:分治思想和动态规划思想中的联系与区别
先是字面解释了下，感觉解释的不是很好，然后具体举例例子，分而治之的思想就可以用归并排序来解释，动态规划01背包进行解释两者区别。

Q:你熟悉的算法？应用场景，改进方向
动态规划，

Q:NP问题和动态规划的联系与区别

3 计算机网络

3.1 考察范围

计算机网络：七层模型、各类通信协议、各层传输过程等

3.2 常见问题

> Q:计算机网络，OSI七层模型的分层思想是不是也是分而治之的思想
> 分层主要是分工进行，让功能区块化，但是分而治之每一小部分是没有联系的，而七层模型中，每一层的服务都是基于下一层的
> Q：各层及各层的协议
> 传输层主要是为进程提供通用数据传输服务，包含可靠不可靠传输、差错控制、流量控制、复用分用等；
> 网络层主要是为主机提供数据传输服务，包含路由选择、差错控制、流量控制、拥塞控制等；数据链路层主要是为同一链路的主机提供数据传输服务，包含成帧、差错控制、流量控制、访问控制等；
> 物理层主要是怎样在传输媒体上传输数据比特流，而不是指具体的传输媒体，即尽可能屏蔽传输媒体和通信手段的差异。

> Q:电路交换，报文交换和分组交换的区别?
> 电路交换：整个报文的比特流从源点连续的直达终点，像在一个管道中传输，包括建立连接、传输数据和断开连接三个阶段，最典型的电路交换网络是传统电话网络；其优点是时延小、有序传输、没有冲突；但有缺点建立连接时间长、独占线路。报文交换：将整个报文转发到相邻节点，全部存储下来，查找转发表，转发到下一个节点，是存储-转发类型的网络；其优点是无需建立连接、动态分配路线、线路利用率高；但有缺点由于存储转发存在转发时延、对报文大小没有控制需要较大存储缓存空间。分组交换：将报文分组转发到相邻节点，查找转发表，转发到下一个节点，也是存储-转发类型的网络；其优点是加速传输、简化存储管理；但有缺点工作量大可能存在分组丢失、需要分组排序

> Q:计算机网络：三次握手
> 为了防止已失效的连接请求报文段突然又传送到了服务端，因而产生错误。已失效的连接请求报文段：client发出的第一个连接请求报文段并没有丢失，而是在某个网络结点长时间的滞留了，以致延误到连接释放以后的某个时间才到达server 。本来这是一个早已失效的报文段。但server收到此失效的连接请求报文段后，就误认为是client再次发出的一个新的连接请求。于是就向client发出确认报文段，同意建立连接。假设不采用“三次握手”，那么只要server发出确认，新的连接就建立了。由于现在client并没有发出建立连接的请求，因此不会理睬server的确认，也不会向server发送数据。但server却以为新的运输连接已经建立，并一直等待client发来数据。这样，server的很多资源就白白浪费掉了。采用“三次握手”的办法可以防止上述现象发生。

> Q:计网的四次挥手
> 因为服务器收到客户端断开连接的请求时，可能还有一些数据没有发完，这时先回复ACK，表示接收到了断开连接的请求。等到数据发完之后再发FIN，断开服务器到客户端的数据传送。

> Q:客户端TIME_WAIT状态的意义是什么?
> 第四次挥手时，客户端发送给服务器的ACK有可能丢失，TIME_WAIT状态就是用来重发可能丢失的ACK报文。如果Server没有收到ACK，就会重发FIN，如果Client在2*MSL的时间内收到了FIN，就会重新发送ACK并再次等待2MSL，防止Server没有收到ACK而不断重发FIN。
> MSL(Maximum Segment Lifetime)，指一个片段在网络中最大的存活时间，2MSL就是一个发送和一个回复所需的最大时间。如果直到2MSL，Client都没有再次收到FIN，那么Client推断ACK已经被成功接收，则结束TCP 连接。

Q: 什么是流量控制？流量控制的目的?
如果发送者发送数据过快，接收者来不及接收，那么就会有分组丢失。为了避免分组丢失，控制发送者的发送速度，使得接收者来得及接收，这就是流量控制。流量控制根本目的是防止分组丢失，它是构成TCP可靠性的一方面。

Q: 如何实现流量控制?
主要的方式就是接收方返回的 ACK 中会包含自己的接收窗口的大小，并且利用大小来控制发送方的数据发送。

Q: 流量控制引发的死锁？怎么避免死锁的发生?
当发送者收到了一个窗口为0的应答，发送者便停止发送，等待接收者的下一个应答。但是如果这个窗口不为0的应答在传输过程丢失，发送者一直等待下去，而接收者以为发送者已经收到该应答，等待接收新数据，这样双方就相互等待，从而产生死锁。
为了避免流量控制引发的死锁，TCP使用了持续计时器。每当发送者收到一个零窗口的应答后就启动该计时器。时间一到便主动发送报文询问接收者的窗口大小。若接收者仍然返回零窗口，则重置该计时器继续等待；若窗口不为0，则表示应答报文丢失了，此时重置发送窗口后开始发送，这样就避免了死锁的产生

Q:拥塞控制、窗口
① 慢开始算法② 拥塞避免算法③ 快重传算法④ 快恢复算法
快重传即收到三个连续的确认就立即重传，这样就不会出现超时，发送方就不会误认为出现了网络拥塞。（慢开始门限）

Q:拥塞控制和流量控制的区别
拥塞控制：拥塞控制是作用于网络的，它是防止过多的数据注入到网络中，避免出现网络负载过大的情况；常用的方法就是：（1）慢开始、拥塞避免（2）快重传、快恢复。
流量控制：流量控制是作用于接收者的，它是控制发送者的发送速度从而使接收者来得及接收，防止分组丢失的。

Q:为什么有了MAC地址，还需要IP地址
由于全世界存在着各式各样的网络，它们使用不同的硬件地址。要是这些异构网络能够互相通信就必须进行非常复杂的硬件地址转换工作，因此由用户或用户主机来完成这项工作几乎是不可能的事。但统一的IP地址把这个复杂问题解决了。连接到因特网的主机只需拥有统一的IP地址，它们之间的通信就像连接在同一个网络（虚拟互连网络或者简称IP网）上那么简单方便，因为调用ARP的复杂过程都是由计算机软件自动进行的，对用户来说是看不见这种调用过程的。

Q:计算机网络中核心思想
1.分布式思想–分组交换
2.分层思想：分层好处是降低耦合，上层不关心底层实现，只关心底层提供服务（接口）；
这样层与层之间通信就可以标准化。标准化意味着层与层之间独立性。，可以独自发展，这样设计带来很大灵活性和扩展性，比如传输层有TCP/UDP/DCCP/STCP等，网络层有IPv4/IPv6等, 数据链路层有以太网，VLAN，WIFI，无线3G，4G，5G协议等；分层模式拥有递归特性，该特性允许逻辑意义的任意封装和再封装，比

> 如overlay网络，VPN，各种tunnel等，使网络扩展性大大增强；
> 3.公平思想–传输控制：TCP的拥塞控制使网络更加公平和稳定，提高系统的容错率，让系统可以持续正常运转；

> Q:密码学
> AES、DES
> 哈希函数SHA256,SHA3,SM3：哈希（散列、杂凑）函数，是把任意长度的输入通过散列算法变换成固定长度的输出，该输出就是散列值。简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数，又被成为信息指纹。
> 哈希函数的特点:
> 压缩:输入为任意长度，输出是固定长度;>高效:计算速度快;
> 单向性:由哈希值得到输入是计算不可行的;
> 抗碰撞性:
> 弱碰撞性:给定x1，找到x2≠x使得H(x1)
> =H(X2);
> 强碰撞性:找到一对x1和xz使得H(1)= H(x;
> 灵敏性:明文的一比特变化对散列值由明显影响
> 数字签名:是签名者利用其私钥和消息产生的别人无法伪造的一段数字串，这段数字串的有效性可以利用签名者的公钥和消息来验证。
> 椭圆曲线数字签名算法(ECDSA)SM2是使用椭圆曲线密码(ECC)对数字签名算法(DSA)的模拟。
> 区块链是一种安全共享的去中心化的数据账本。区块链技术支持一组特定的参与方共享数据。它可以收集和共享多个来源的事务数据能够将数据细分为以哈希形式的唯一标识符链接在一起的共享区块，并通过单一信息源确保数据完整性，消除数据重复，提高数据安全性。相关技术:哈希函数（SHA-256）、数字签名算法（ECDSA）等

3.3 遇到的问答*

> Q: RSA加密算法
> 其公钥和私钥是一对大素数（100到200位十进制数或更大）的函数。从一个公钥和密文恢复出明文的难度，等价于分解两个大素数之积（这是公认的数学难题）
> 非对称的加密算法，单向函数正向求解简单，逆向求解复杂，信息接受方生成公钥与私钥，将公钥公布，发送方将原文与接收方的公钥运算得到密文，接受方通过密文与私钥进行解密

> Q: 数据链路层与传输层的流量控制的关联与区别?
> 1.数据链路层的流量控制是点对点的，而传输层的流量控制是端到端的。
> 2.数据链路层流量控制的手段是接收方收不下就不回复确认帧。传输层的流量控制手段是接收端通过滑动窗口告诉发送方。

4 计算机语言

4.1 考察范围

C++：面向对象，继承、封装、多态，虚函数，纯虚函数
C语言：指针、内部存储、实现机制等
Python语言
Java语言

4.2 常见问题

Q:指针在底层是如何实现的?
Q:python和C++在底层上的区别
Q:指针和引用的区别，
指针：是保存另一个变量内存地址的变量，指针通过 * 访问保存的内存地址所指向的值；
引用：是另一个变量的别名，一旦被初始化就不能被改变，引用可以认为是一个具有自动间接性的常量指针，相当于编译器帮助实现了自动间接性取值，即：编译器帮助加上了 * 。引用内部实现为指针。
1.指针可以先定义，再初始化，可以重复赋值。引用必须定义时初始化，一旦被初始化，就不能改变，类似于 const 定义常量
2.指针变量有地址，而引用的地址是引用变量的地址；
3.指针可以使用 NULL 赋值为空，引用不可以；
4.指针可以多级，引用只能一级；
5.指针需要 （*）来引用值，而引用可以直接取值。
6.指针支持算术运算，而引用不能。

> Q:什么情况下使用析构函数?
> 构造函数（方法）是对象创建完成后第一个被对象自动调用的方法。它存在于每个声明的类中，是一个特殊的成员方法。作用是执行一些初始化的任务。Php中使用__construct()声明构造方法，并且只能声明一个。
> 析构函数（方法）作用和构造方法正好相反，是对象被销毁之前最后一个被对象自动调用的方法。是PHP5中新添加的内容作用是用于实现在销毁一个对象之前执行一些特定的操作，诸如关闭文件和释放内存等
> 当我们在类中声明了一些指针变量时，我们一般就在析构函数中进行释放空间，因为系统并不会释放指针变量指向的空间，我们需要自己来delete，而一般这个delete就放在析构函数里面。

> Q:程序改错题，基类私有派生出一个子类，则基类中共有成员变量在派生类中变为私有变量
> A:错误的地方为:主函数中通过对象名直接访问了子类的这个变量

4.3 遇到的问答*

> Q:熟悉掌握的编程语言有哪些?

> Q:java和python区别，怎么理解面向对象，接口是什么
> Java是一种静态类型语言，Python是一种动态类型语言，Python的变量是动态的，而java的变量是静态的，需要事先声明；
> Python有好多程序用的是面向过程设计方法，class在Python中是后加入的，而java是为了实现没有指针的c++，主要采用面向对象的设计方法，很多概念是oop的概念。面向过程，相对简洁直观，面向对象，相对抽象优雅，但容易过度抽象；
> Python的库强大，无论gpu运行，神经网络，智能算法，数据分析，图像处理，科学计算，各式各样的库可以使用，而java没有Python那么多的开源库，很多库是商业公司内部使用，或发布出来只是一个jar包，看不到原始代码；
> java偏向于商业开发，Python适合于数据分析；
> 是一种编程思想，从面向过程过渡而来的，面向过程的思维方式，它更加注重这个事情的每一个步骤以及顺序，比较直接高效，需要做什么可以直接开始编程；面向对象的思维方式，它更加注重事情有哪些参与者，需求里面有哪些对象，这些对象各自需要做些什么事情。将其拆解成一个个模块和对象，这样会更易于维护和拓展。
> 把数据及对数据的操作方法放在一起，作为一个相互依存的整体–对象。对同类对象抽象其共性，形成类。类中的大多数数据，只能在本类的方法进行处理。类通过一个简单的外部接口与外界发生关系，对象与对象之间通过消息进行通信。程序流程由用户在使用中决定。简单讲就是说万物皆对象。
> 封装、继承、多态；

> Q: 熟悉的语言？Java的GC原理？多线程？C++和java的区别？引用是否可以传参数？C语言的malloc函数？C语言return返回多个值？给你5000行代码你会怎么办？栈溢出的原因
> 1.GC是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的。CPU与内存以及IO之间存在着巨大的性能差异。
> 2.Java 比C++更加可靠，语法更加简洁，完全面向对象，Java运行在JVM上可移植性更强，java不需要对于内存进行分配与回收，都是自动进行的；没有指针的概念而是直接使用数组；使用接口代替了C++的多继承；
> 3.在向参数传递引用类型的值时，会把这个值在内存中的地址复制给一个局部变量，因此这个局部变量的变化会反映在函数的外部。
> 相当于一个指针，引用类型传递时相当于指针的指向的传递。
> 4.在堆栈当中进行内存块的申请，返回的是void* 需要进行强制类型转换，使用free进行释放；
> 5.使用全局变量，从定义变量到程序结束；
> 数组地址法，返回值元素定义为一个数组，并使用数组的地址作为形参；
> 结构体地址法，结构体指针传递给形参结构体指针；
> 1.首先，编译代码，使得代码能够运行起来，顺利地搭建调试环境；
> 明确的目的，对于框架进行分析，主函数是如何去调用各个模块的，各个模块实现了哪些功能，这段代码主要完成的是什么任务
> 7.函数调用太深，例如递归；动态申请空间后没有释放；

> Q: 程序编译执行过程？

5 其他专业课

5.1 考察范围

- 操作系统：作业调度、进程调度、内存管理等

- 数据库：关系数据库模型、视图等
- 软件工程：软件开发模型

5.2 常见问题

> Q:操作系统：生产者消费者模型

> Q:调度算法
> Q:进程和线程，多线程多进程 ， 线程与进程的异同点

> Q：计组的衡量计算机的性能指标
> CPU的综合性能、存储容量、数据的输入\输出能力（即I\O吞吐率）、响应时间和功耗等。

> Q: 视图
> 是从一个或多个表导出的虚拟的表，其内容由查询定义。具有普通表的结构，但是不实现数据存储，对视图的修改：单表视图一般用于查询和修改，会改变基本表的数据，多表视图一般用于查询，不会改变基本表的数据。
> 1）简单：使用视图的用户完全不需要关心后面对应的表的结构、关联条件和筛选条件，对用户来说已经是过滤好的复合条件的结果集。
> 2）安全：使用视图的用户只能访问他们被允许查询的结果集，对表的权限管理并不能限制到某个行某个列，但是通过视图就可以简单的实现。
> 3）数据独立：一旦视图的结构确定了，可以屏蔽表结构变化对用户的影响，源表增加列对视图没有影响；源表修改列名，则可以通过修改视图来解决，不会造成对访问者的影响。
> 总而言之，使用视图的大部分情况是为了保障数据安全性，提高查询效率
> 存储过程：事先经过编译并存储在数据库中的一段SQL语句的集合，想要实现相应功能时，直接调用该存储过程即可。优点：select、update、select……，可能多次操作数据库，那么就会涉及到多次网络请求，所以采用存储过程，实现数据库SQL语言层面的代码封装和重用。
> 触发器：触发器是与表有关的数据库对象，指在insert、update、delete之前或者之后，触发并执行触发器中定义的SQL语句集合。作用：协助应用在数据库端确保数据的完整性、数据校验等操作。

> Q:三级封锁协议 两段锁协议?
> 答：三级封锁协议（数据一致性）、两段锁协议（并行调度可串行）。
> 一级封锁：事务在修改之前必须先对其加X锁，直到事务结束才释放。

> Q：软件开发模型
> 软件的生命周期分为6个阶段，即需求分析、计划、设计、编码、测试、运行维护。
> 1.瀑布模型：每个阶段都只执行一次，因此是线性顺序的软件开发模型。

2.螺旋模型：是渐进式开发模型的代表之一。
3.迭代模型：迭代模型是类似小型的瀑布式项目。每一个迭代都会产生一个可以发布的产品，这个产品是最终产品的一个子集。
4.增量模型：采用随时间进展而交错的线性序列。
5.敏捷模型：敏捷模型是一种轻量、高效、低风险、更强调团队协作和沟通的开发方式，适合于中小型开发团队，客户需求模糊或多变。

5.3 遇到的问答*

Q: 进程与线程的区别？OS实现什么？中断相关？互斥锁如何解决临界区的资源问题？
1.进程是操作系统资源分配的基本单位，而线程是处理器任务调度和执行的基本单位（也可以理解为进程当中的一条执行流程）；线程是进程的一部分；同一进程的线程共享本进程的地址空间和资源，而进程之间的地址空间和资源是相互独立的；一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个线程崩溃整个进程都死掉。所以多进程要比多线程健壮。
2.是计算机硬件上的第一层软件，对于硬件系统的首次扩充，是计算机系统资源的管理者（文件、设备、处理机、存储器），是用户与计算机之间的接口（命令接口，GUI图形用户接口），具有并发（并行，多CPU），共享、虚拟、异步（由于资源有限，进程的执行不是一贯到底，而是以不可预知二等速度向前推进）；
3.中断使得CPU由内核态变为用户态，夺回操作系统对于CPU的控制
中断是否与正在执行的指令相关，内中断，外中断（时钟中断，I\O设备中断）
每执行一条指令，检查是否由外部中断；中断类型->中断向量表；
应用程序可以通过系统调用来请求获得操作系统内核的服务，系统中的各种共享资源都由操作系统内核统一掌管，因此凡是与共享资源有关的操作(如存储分配、I/O操作、文件管理等)，都必须通过系统调用的方式向操作系统内核提出服务请求；
4.多个并发进程都需要访问临界区的资源时候，只能有一个进程可以进去临界区访问，其他进程不可以访问，其他进程需要等到进入临界区的进程访问结束后，才有资格访问临界区，这就是互斥；

显示推荐内容