非技术面问题

自我介绍

面试官您好，我叫王佳文。

2017年毕业于山东科技大学大学地球科学与工程学院，大四下学期在青岛蓝泽信息技术有限公司实习4个月，任职为前端开发工程师，主要负责根据需求实现指定的功能页及交互功能。2018年研究生录取后在长治科利来科技有限公司实习四个月，任职为JAVA后端开发工程师，主要根据客户需求进行页面调整以及开发一个外接的物料管理系统。

毕业后2018年跨考入太原理工大学计算机技术专业，主要从事基于人工智能的医学图像处理方面的研究，第1年主要完成课程需求以及帮助实验室博士学姐实现一些科研任务，期间以第三作者协助发表SCI论文三篇；第2年开始自己开始负责基于直肠癌MRI影响预测KRAS基因突变的研究任务，并于今年3月份以第一作者成功发表SCI3区论文一篇。今年4月份成功申请山西省研究生科研创新项目一项，目前正处于研究阶段。此外，除了科研任务以外，我还负责实验室所有送审论文的分配及审稿工作，负责实验室GPU服务器管理工作。

我是在实习僧网站上看到的岗位推荐，感觉我的工作经验和工作能力和该岗位比较匹配，所以投递了求职简历。

另外，xx是个很棒的城市，xx年前我有幸去过一次，我非常希望到xx市工作。我也在网上查询了贵公司的评价，公司发展的很好，制度也很完善，我非常希望能够进入公司工作。

自身的三个标签

学习能力强 ： 我的研究生考试全部是通过自学完成的；我在本科期间自学并通过了计算机二级、三级的数据库等级考试；我在研究生期间独自负责一个新的研究方向，并成功发表了论文。

善于融入新环境 ： 我很少会因为无法融入新环境而遇到生活或工作上的问题，并且一般跟大家相处的都还不错。

有自己的处事原则： 一般情况下我会按照老师们的安排完成任务，但是当我有自己的想法时，我会主动跟他们提出并征求意见。

自身缺点

过于理性： 对于自己或者身边朋友们遇到的问题，我不太能做到感同身受，一般会相当客观的提出自己的建议，经常会被人吐槽。近年来已经尽可能去理解他们的想法了。

学习新内容的时候偏向于实践化： 我一般更喜欢直接上手尝试新事物，对于从理论开始仔细研读可能没有太大的耐心，近年来我正在逐渐养成记笔记的习惯，尝试把我接触到的理论知识系统化，更好的指导我的日常实践活动。

对该公司的了解程度

好的回答需要包含公司长远战略、生态构建、各模块业务分工等

希望的薪资

技术面问题

1. 数据结构及算法

红黑树

红黑树，Red-Black Tree 「RBT」是一个自平衡(不是绝对的平衡)的二叉查找树(BST)，树上的每个节点都遵循下面的规则:

1. 每个节点都有红色或黑色
2. 树的根始终是黑色的 (黑土地孕育黑树根，)
3. 没有两个相邻的红色节点（红色节点不能有红色父节点或红色子节点，并没有说不能出现连续的黑色节点)
4. 从节点（包括根）到其任何后代NULL节点(叶子结点下方挂的两个空节点，并且认为他5. 们是黑色的)的每条路径都具有相同数量的黑色节点

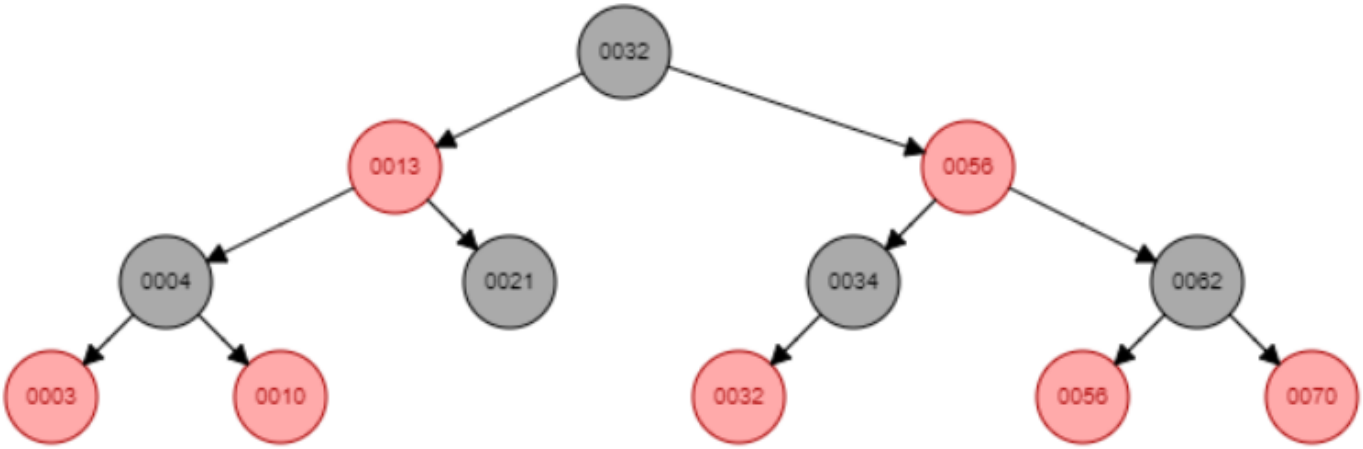

图 红黑树动画演示

红黑树的插入与查询

红黑树的查询首先遵守普通二叉排序树的插入规则，区别在于插入后需要进行适度的调整。

红黑树的插入大体上有如下几种情况：

- 空树，直接插入为根节点，标记为黑色
- 父节点和叔叔节点都为红色节点，此时将两节点标记为黑色，并将自己和祖父节点标记为红色，如果祖父节点为根节点则将祖父节点再标记为黑色。全过程不涉及数的结构的变化
- 父节点和叔叔节点为一黑一红两个节点，且红色节点端下接了一个红色节点，此时的调整方案与平衡二叉树貌似是一致的。共分为四种情况，两类，一类可以直接拎起来，把中间节点转化为父亲节点；另一类需要调整子节点顺序与父节点组成一条线再拎起来。

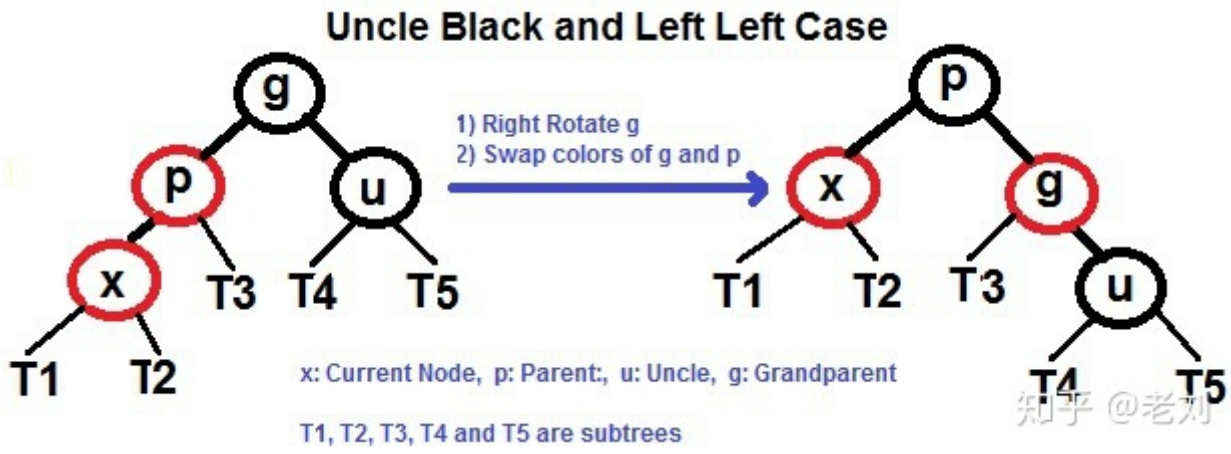

图 红黑树插入的第一类情况 直接拎

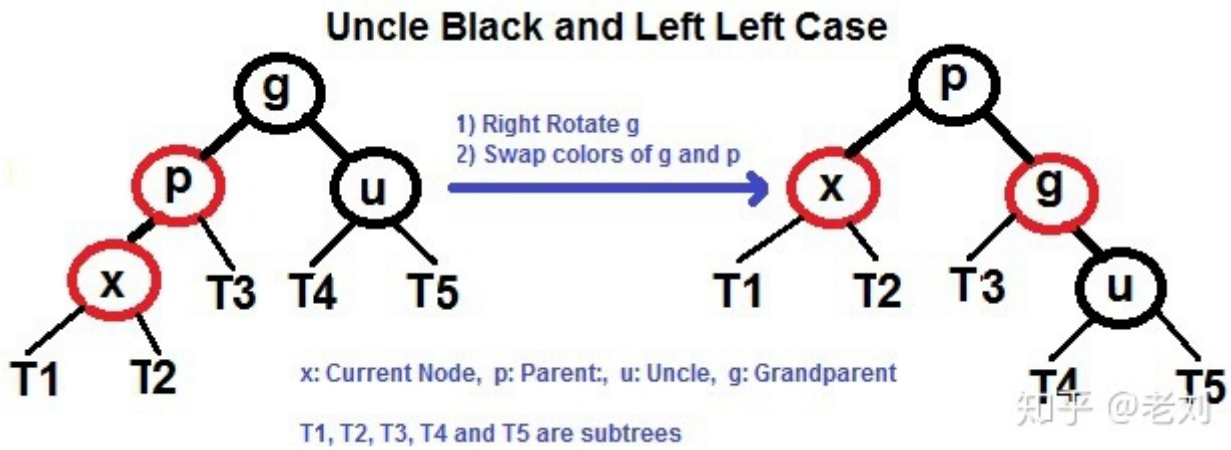

图 红黑树插入的第二类情况 先转化再拎

红黑树与平衡二叉树的区别

平衡二叉树保证了在最差的情况下，二叉树依然能够保持绝对的平衡，即左右两个子树的高度差的绝对值不超过1。但是这又会带来一个问题，那就是平衡二叉树的定义过于严格，导致每次插入或者删除一个元素之后，都要去维护二叉树整体的平衡，这样产生额外的代价又太大了。**二叉搜索树可能退化成链表，而平衡二叉树维护平衡的代价开销又太大了**，那怎么办呢？这就要谈到“中庸之道”的智慧了。说白了就是把平衡的定义适当放宽，不那么严格，这样二叉树既不会退化成链表，维护平衡的开销也可以接受。没错，这就是我们要谈的红黑树了。

红黑树不像平衡二叉树树一样追求绝对的平衡，他允许局部很少的不完全平衡，这样对于效率影响不大，但省去了很多没有必要的调平衡操作，平衡二叉树树调平衡有时候代价较大，所以效率不如红黑树，在现在很多地方都是底层都是红黑树的天下啦~

哈希表及常见的冲突解决方法

哈希表概念

哈希表，也称散列表，从根本上来说，一个哈希表包含一个数组，通过特殊的关键码(也就是key)来访问数组中的元素。哈希表的主要思想是通过一个哈希函数， 把关键码映射的位置去寻找存放值的地方，读取的时候也是直接通过关键码来找到位置并存进去。用公式来表达就是：$f(key)$，而这样的函数所建立的表就是哈希表。比起数组和链表查找元素时需要遍历整个集合的情况来说，哈希表明显方便和效率的多。

因此，对于哈希表，其性能主要取决于哈希函数的设计。常见的哈希算法(函数)有如下几种：

- 直接定址法 --- 取关键字或关键字的某个线性函数值为散列地址取关键字或关键字的某个线性函数值为散列地址，$f(key) = a * key + b$
- 除留余数法 --- 取关键字被某个不大于散列表长度 m 的数 p 求余，得到的作为散列地址，$f(key) = key\%p, p < m$

- 数字分析法 --- 当关键字的位数大于地址的位数，对关键字的各位分布进行分析，选出分布均匀的任意几位作为散列地址
- 平方取中法 --- 先计算出关键字值的平方，然后取平方值中间几位作为散列地址
- 随机数法 --- 选择一个随机函数，把关键字的随机函数值作为它的哈希值

哈希表冲突解决办法

一般来说，哈希冲突是无法避免的，如果要完全避免的话，那么就只能一个字典对应一个值的地址，也就是一个字就有一个索引 (安 和 按就是两个索引)，这样一来，空间就会增大，甚至内存溢出。常见的哈希冲突解决办法有两种，开放地址法和链地址法。

开放地址法的做法是，当冲突发生时，使用某种探测算法在散列表中寻找下一个空的散列地址，只要散列表足够大，空的散列地址总能找到。按照探测序列的方法，一般将开放地址法区分为线性探查法、二次探查法、双重散列法等。

- 线性探查法 --- 探查时从地址d开始，首先探查T[d]，然后依次探查T[d+1]，…，直到T[m-1]，此后又循环到T[0]，T[1]，…，直到探查到有空余的地址或者到T[d-1]为止。缺点是需要不断处理冲突，无论是存入还是查找效率都会大大降低。
- 二次探查法 --- 探查时从地址d开始，首先探查T[d]，然后依次探查T[d+di]，di为增量序列，直到探查到有空余地址或者到T[d-1]为止。
- 双哈希函数探测法 --- 双哈希函数探测法，先用第一个函数 f(key) 对关键码计算哈希地址，一旦产生地址冲突，再用第二个函数 g(key) 确定移动的步长因子，最后通过步长因子序列由探测函数寻找空的哈希地址。
- 链地址法 --- 如果节点12和节点0的位置冲突了，然后我们把该数组的每一个元素变成了一个链表头，冲突的元素放在了链表中，这样在找到对应的链表头之后会顺着链表找下去，至于为什么采用链表，是为了节省空间，链表在内存中并不是连续存储，所以我们可以更充分地使用内存。

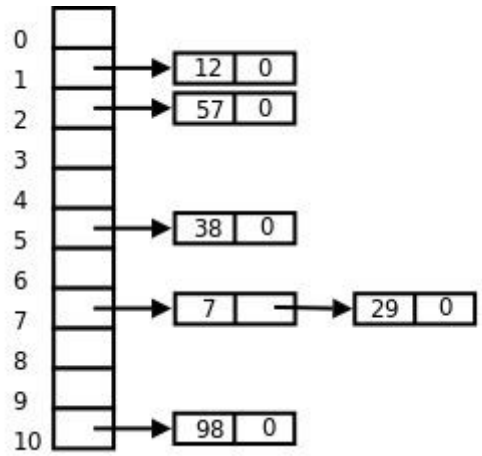

图 链地址法图解

HashMap与红黑树

HashMap中的值都是key，value。其实这里的存储与哈希表的很像，key会被映射成数据所在的地址，而value就在以这个地址为头的链表中，这种数据结构在获取的时候就很快。但这里存在的问题就是如果hash桶较小，数据量较大，就会导致链表非常的长。比如说上面的长为11的空间我要放1000个数，无论Hash函数如何精妙，后面跟的链表都会非常的长，这样Hash表的优势就不复存在了，反而倾向于线性检索。

在jdk1.8版本后，java对HashMap做了改进，在**链表长度大于8的时候**，将后面的数据存在红黑树中，以加快检索速度。

堆空间、栈空间

什么是堆栈

- 栈区（stack）：又编译器自动分配释放，**存放函数的参数值，局部变量的值**等，其操作方式类似于数据结构的栈。
- 堆区（heap）：一般是由程序员分配释放，若程序员不释放的话，程序结束时可能由OS回收，值得注意的是他与数据结构的堆是两回事，**分配方式类似于数据结构的链表**。

栈、堆的区别

- 分配方式不同。前者由系统分配；后者由程序员自己分配和定义。
- 申请方式不同。前者由系统自动分配；后者由程序员自己申请，C语言中使用malloc进行申请。
- 响应方式不同。**对于栈来讲**，只要栈的剩余内存足够，系统会自动为程序提供内存；**对于堆来讲**，首先应该知道操作系统有一个记录内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请的空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序。另外，对于大多数系统，**会在这块内存空间中的首地址处记录本次分配的大小**，这样代码中的delete或free语句就能够正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会将多余的那部分重新放入空闲链表中。
- 申请的大小和效率不同。在windows下，栈是向低地址扩展的数据结构，是一块连续的内存区域，栈顶的地址和栈的最大容量是系统预先规定好的，能**从栈获得的空间较小**；堆是向高地址扩展的数据结构，是不连续的内存区域，这是由于系统是由链表在存储空闲内存地址，自然**堆就是不连续的内存区域**，且链表的遍历也是从低地址向高地址遍历的，堆得大小受限于计算机系统的有效虚拟内存空间，由此空间，**堆获得的空间比较灵活，也比较大**。

2. 机器学习

决策树如何进行回归与分类

决策树的主要种类

ID3, C4.5, CART

其中ID3主要是通过信息熵和信息增益进行选择最优属性进行分裂， C4.5主要通过信息增益比进行分裂，CART主要通过基尼系数进行分裂。

- ID3算法。(a) ID3没有考虑连续值，如长度、密度等；(b) ID3没有对缺失值进行考虑；(c) 没有考虑过拟合问题；(d) 在相同条件下取值较多的特征，比取值较小的特征的信息增益大。

$$
\begin{aligned}
H(D) &= -\sum_{i=1}^{n} P_i \log P_i \\
g(D, A) &= H(D) - H(D|A) \\
&= H(D) - \sum_{i=1}^{n} P_i H(D|A_i)
\end{aligned}
$$

  其中，P_i为当前属性下不同取值的概率，$H(D|A_i)$为当前数据下下，当前属性值时，不同类别的信息熵。
- C4.5算法。解决了ID3的 (d) 问题，特征数越多的特征对应的信息熵越大，应作为分母以矫正信息增益的偏向的问题。

$$
I_R(D, A) = \frac{g(D, A)}{H(D)}
$$

为什么CART(Classification and Regression Tree)可以做回归，而ID3和C4.5不可以

回归决策树主要指CART算法，内部结点特征的取值为“是”和“否”， 为二叉树结构。所谓回归，就是根据特征向量来决定对应的输出值。回归树就是将特征空间划分成若干单元，每一个划分单元有一个特定的输出。因为每个结点都是“是”和“否”的判断，所以划分的边界是平行于坐标轴的。对于测试数据，我们只要按照特征将其归到某个单元，便得到对应的输出值。划分的过程也就是建立树的过程，每划分一次，随即确定划分单元对应的输出，也就多了一个结点。当根据停止条件划分终止的时候，最终每个单元的输出也就确定了，也就是叶结点。

CART可以做回归，而ID3和C4.5不可以回归是因为几个模型的损失函数及分裂方式都不同。

- 分裂方式。CART有两种评价标准：Variance和Gini系数。而ID3和C4.5的评价基础都是信息熵。信息熵和Gini系数是针对分类任务的指标，而**Variance是针对连续值的指标因此可以用来做回归。**
- 损失函数。**对于CART回归模型来讲**，其回归模型采用和方差的形式度量划分特征A的优劣。度量目标是对于划分特征A， 对应的划分点S两边的数据集D1,D2，求出使D1，D2各自集合的均方差最小，同时使D1和D2的均方差之和最小。

$$
L = \underbrace{min}_{A,S}[\underbrace{min}_{c1} \sum_{x_i \in D_1(A,S)} (y_i - c_1)^2 + \underbrace{min}_{c_2} \sum_{x_i \in D_2(A,S)} (y_i - c_2)^2]
$$

  对于分类模型来讲，采用基尼指数或信息熵等评价指标度量各个划分点的优劣。

决策树的优缺点

决策树的一些优点是：

- 易于理解和解释。决策树可以进行可视化。
- 需要很少的数据准备。其他技术通常需要数据规范化，需要创建伪变量并删除空白值。但是请注意，此模块不支持缺少的值。
- 使用树的成本（即预测数据）与用于训练树的数据点数量成对数。
- 能够处理数字和分类数据。其他技术通常专用于分析仅具有一种类型的变量的数据集。有关更多信息，请参见算法。
- 能够处理**多输出问题**。
- 使用白盒模型。如果模型中可以观察到给定的情况，则可以通过布尔逻辑轻松解释条件。相反，在黑匣子模型中（例如，在人工神经网络中），结果可能更难以解释。
- 可以使用统计测试来验证模型。这使得考虑模型的可靠性成为可能。
- **即使生成数据的真实模型在某种程度上违背了它的假设，也可以表现良好**。

决策树的缺点包括：

- **决策树学习者可能会创建过于复杂的树，从而无法很好地概括数据。这称为过度拟合**。为避免此问题，必须使用诸如剪枝、设置叶节点处所需的最小样本数或设置树的最大深度之类的机制。
- **决策树可能不稳定**，因为数据中的细微变化可能会导致生成完全不同的树。**通过使用集成学习中的决策树可以缓解此问题**。
- 在最优性的几个方面，甚至对于简单的概念，学习最优决策树的问题都被认为是NP完全的。因此，实用的决策树学习算法基于启发式算法（例如贪婪算法），其中在每个节点上做出局部最优决策。这样的算法不能保证返回全局最优决策树。可以通过在集成学习器中训练多棵树来缓解这种情况，在该学习器中，特征和样本将通过替换随机抽样。
- 有些概念很难学习，因为决策树无法轻松表达它们，例如XOR，奇偶校验或多路复用器问题。
- **类别不均衡的处理能力较差，如果某些类别占主导地位，则决策树学习者会创建有偏见的树**。因此，建议在与决策树拟合之前平衡数据集

决策树与随机森林

GBDT与xgboost

常见的集成学习算法并简述

GCForest及其改进

对傅里叶变换有什么了解，如何实现，概念讲解

如何理解最大似然估计

如何衡量两个分布的差异，KL散度与交叉熵有什么不同，关系是什么?

机器学习中如何衡量模型效果?如何理解AUC值?

L1, L2范数作为损失函数的效果及区别。huber函数跟他们的区别又是什么

3. 深度学习

残差网络及残差网络的发展

adam的优化思想

常见的目标检测模型

简述UNET

常见的注意力机制

怎么解决梯度消失问题

批量归一化的思想，还了解其他的归一化吗

深度学习的常见的trick

4. 场景题

如何给图片去水印

从一段长视频中截取或者拼凑10s-20s的短视频用于广告投放，吸引用户点击下载app，怎么得到目标短视频

References

- 面试常见的几个问题如何回答才能最得体?
- 字节跳动算法实习 一二三面面经分享（已offer）
- 为什么要有红黑树？什么是红黑树？画了20张图，看完这篇你就明白了
- 红黑树，超强动静图详解，简单易懂
- 红黑树动画在线演示
- 数据结构：哈希表以及哈希冲突的解决方案
- 堆空间与栈空间的区别
- 决策树之分类树与回归树
- 决策树算法原理(CART分类树)
- 完整决策树（回归树）推导加讲解
- 为什么CART能做回归而ID3和C4.5不可以?