数据结构和算法

Day1数据结构

程序设计=算法+数据结构

数据结构：数据元素之间存在的一种或多种特定关系的集合。

数据：描述客观事物的符号，是计算机中可以被操作的对象。是能被计算机识别，并输入给计算机处理的符号集合。数据不仅仅包括整型、实型等数值类型，还包括字符、声音、图像、视频等非数值类型。

数据元素：组成数据的，有一定意义的基本单位，又称记录。

数据项：一个数据项可以由若干个数据项组成。是数据不可分割的最小单位。

数据对象：是数据相同的数据元素的集合，是数据的子集，即具有相同数量和类型的数据项。

传统上，数据结构分为逻辑结构和物理结构。

逻辑结构：数据对象中，数据元素之间的相互关系。

物理结构：数据中的逻辑结构在计算机中的存储形式。

四大逻辑结构：集合结构、线性结构、树形结构、图形元素。

集合结构：数据元素同属于一个集合，没有任何关系。

线性结构：数据元素存在一对一的关系。

树形结构：数据元素存在一对多的层次关系。

图形结构：数据元素存在多对多的关系。

物理结构：如何把数据元素存储在计算机的存储器中。

存储器：主要是针对内存而言的，比如硬盘、软盘、光盘等外部存储器的数据组织通常用文件结构来描述。

数据元素的存储形式分为：顺序存储和链式存储。

顺序存储结构：是把数据元素存放在地址连续的存储单元里，其数据间的逻辑关系和物理关系是一致的。例如：数组。

链式存储结构：把数据元素存放在任意的存储单元里，这组存储单元可以是连续的，也可以是不连续的。类似排号系统。

链式存储结构的数据元素存储关系并不能反映其逻辑关系，因此需要用指针来存放数据元素的地址。即每个数据单元不仅存放自身数据，还存放指向下一个数据单元地址的指针。

数据类型：是指一组性质相同的值的集合及定义在此集合上的操作的总称。

Day2算法

算法：解决特定问题求解步骤的描述，在计算机中表现为指令的有限序列，并且每条指令表示一个或多个操作。

算法的五个特性：输入、输出、有穷性、确定性、可行性。

确定性：每个步骤有确定的含义，不会出现二义性。

算法设计的要求：正确性、可读性、健壮性、时间效率高和存储量低。

时间复杂度：在进行算法分析时，语句总的执行次数T(n)是关于问题规模的函数，进而分析T(n)随n的变化情况并确定T(n)的数量级。算法的时间复杂度，也就是算法的时间量度，记作：T(n)=O(f(n))。它表示随问题规模n的增加，算法执行时间的增长率和f(n)的增长率相同，称作算法的渐进时间复杂度。其中f(n)是问题规模n的某个函数。

这样用大写O来体现算法时间复杂度的记法，称之为大O记法。

推导大O阶的方法：

1.用常数1取代运行时间中所有的加法常数。

2.在修改后的运行次数函数中，只保留最高阶次。

3.如果最高阶存在且不是1，则去除与这个项相乘的常数。

得到的最后结果为大O阶，一般为1，n，n的平方，n的3次方…

常数阶：与问题的大小无关(n的多少)，执行时间恒定的算法，我们称之为O(1)的时间复杂度，又叫常数阶。（Ps：不管这个常数为多少，都是O(1)）

对数阶：无论是以哪个数为底的对数阶，其时间复杂度均为O(logN)，证明参考高数求极限。

算法的空间复杂度通过计算算法所需的存储空间实现，算法空间复杂度的计算公式记作：s(n)=O(f(n)),其中，n为问题的规模，f(n)为语句关于n所占存储空间的函数。

使用“时间复杂度”来指运行时间的需求，使用“空间复杂度”指空间需求，直接说“复杂度”，通常是指时间复杂度。

Day3线性表

线性表：0或多个数据元素的有限序列。

线性表的数据元素有限，除第一个和最后一个数据元素之外，每个数据元素都有一个直接前驱元素和一个直接后继元素。线性表中元素的个数为线性表的长度，当个数为0时，称为空表。

数据元素在线性表中的位序：即线性表中的位置，如a1的位于为1。

在较复杂的线性表中，一个数据元素可以由若干个数据项组成。

有序数组：查询快（使用二分查找法O(logN)），增删慢（O(N)）

无序数组：查询慢（O(N)），删除慢（需要数组项向前移动以填补空缺，有序数组同理O(N)），增加时间恒定（直接在数组末尾插入数据项O(1)）

二分查找：又称折半查找，时间复杂度为logN，适用于有序数组

**int** start = 0, end = n - 1, middle = 0;

**while**(start < end){

middle = (start + end) / 2;

**if**(a[middle] >= x){

end = middle - 1;

}**else**{

start = middle + 1;

}

}

**if**(a[end] == x){

**return** end;

}

**return** -1;

冒泡排序：时间复杂度为n^2，比较次数为1/2 \* (n - 1) \* n，最坏交换次数为1/2 \* (n - 1) \* n

**for**(**int** i = 0; i < a.length; i++){

**for**(**int** j = i + 1; j < a.length; j++){

**if**(a[i] > a[j]){

t = a[i];

a[i] = a[j];

a[j] = t;

}

}

}

选择排序：时间复杂度为n^2，比较次数为(n - 1) \* n / 2，最坏交换次数为n - 1

**for** (**int** i = 0; i < a.length; i++) {

min = i;

**for** (**int** j = i + 1; j < a.length; j++) {

**if**(a[j] < a[min]){

min = j;

}

}

t = a[i];

a[i] = a[min];

a[min] = t;

}

插入排序：时间复杂度为n^2比较次数为(n - 1) \* n / 2，最坏交换次数为n - 1

**for** (i = 1; i < a.length; i++) {

**int** t = a[i];

int j = i;

**while**(j > 0 && a[j - 1] >= t){

a[j] = a[j - 1];

j--;

}

a[j] = t;

}

插入排序比冒泡排序快一倍，比选择排序略快，对于基本有序或已经有序的数组来说，while循环的条件为false，因此不会执行内层循环，所以，它变成了外层循环的一个简单语句。

简单排序算法中的两个基本操作是：比较并交换（或复制）数据项

选择排序中，最小的关键字被重复的发现。

选择排序中的不变性指的是：下标小于标记的项部分有序。（或大于，具体看程序先排哪边）

向左向右进行移动一组数据需要重复的进行复制。

插入排序中的不变性是指下标小于标记的项部分有序。（或小于，具体看程序先排哪边）

栈、队列、优先级队列主要通过接口进行定义，这些接口表明它们可以完成的操作，而它们的实现过程对用户来说是不可见的。

栈：基于数组（也可以用链表实现），由于访问受限制，不能利用下标访问各数据项。后进先出（LIFO）

只允许访问一个数据项，即最后插入的数据项。移除这个数据项之后才能访问倒数第二个数据项，以此类推。

栈限制线性表中元素的插入和删除只能在线性表的同以端进行的一种特殊的线性表，允许插入和删除的一端，为变化的一端，称为栈顶，另一端为固定的一端，称为栈底。

最先放入栈中元素在栈底，最后放入的元素在栈顶，删除元素相反，最先删除最后放入的元素，最后删除最先放入的元素。

数据项入栈和出栈的时间复杂度都为常数O(1)，即栈操作所耗的时间不依赖于栈中数据项的个数。

使用栈完成表达式的计算思路：

1.通过一个index值，来遍历表达式；

2.如果发现是一个数字，就直接push入数栈；

3.如果发现是一个符号，分如下情况：

3.1 如果当前的符号栈为空，就直接push入符号栈；

3.2 如果符号栈有操作符，就进行比较，如果当前操作符的优先级小于或等于栈中的操作符 ，就需要从数栈中pop出两个数据，进行运算（后pop出的数据 操作符 先pop出的数据），将得到的结果push入数栈，然后将当前操作符push入数栈，如果当前操作符的优先级大于栈中的操作符，就push入符号栈；

4.当表达式遍历完毕，就顺序的从数栈和符号栈中pop出对应的数和符号，并进行运算；

5.最后在数栈中只有一个数字，就是表达式的结果。

前缀表达式的计算机求值过程：

从右至左扫描表达式，遇到数字时，将数字压入堆栈，遇到运算符时，弹出栈顶的两个数，用运算符对它们做相应的计算（栈顶元素和次顶元素），并将结果入栈，重复上述过程直到表达式最左侧，最后得出的值即为表达式的结果。

中缀表达式转后缀表达式的步骤：



逆波兰表达式的计算思路：

1.从左到右依次扫描，将3和4压入堆栈；

2.遇到+运算符，弹出4和3（4为栈顶元素，3为次顶元素），计算出3和4的值，得7，再将7入栈；

3.将5入栈；

4.接下来是\*运算符，因此弹出5和7，计算出7\*5=35，将35入栈；

5.将6入栈；

6.最后是-运算符，计算出35-6的值，即29，由此得出最终结果

队列：基于数组实现，先进先出（FIFO）

队列中元素的插入和移除的时间复杂度都为O(1)

优先级队列：在队列中将关键字按重要程度排序（最大值/最小值）

优点：可以快速访问到最小/最大关键字的数据项。

通常使用堆来实现，也可以使用数组实现，但插入速度慢，适用于数据量小且不是特别注重插入速度的情况。

优先级队列的时间复杂度：插入O(n)，删除O(1)

使用数组进行数据结构实现的问题：

在无序数组中，搜索很慢，插入很快；

在有序数组中，插入很慢，搜索比较快；

不论在哪一种数组中，删除都很慢；

数组在创建之后，大小就不能被修改。

单向链表头指针指向新插入的数据，next指针指向上一个插入的数据。

链表是以节点的方式来存储；

每个节点都包含data域，next域指向下一个节点；

链表的各个节点不一定是连续存储；

链表分带头节点和没有头的节点。

双向链表：

遍历：可以向前遍历，也可以向后遍历

添加：默认添加到双向链表的最后

删除：可以实现 自我删除，直接找到节点

temp.pre.next = temp.next;

temp.next.pre = temp.pre;

稀疏数组：当一个二维数组中存在很多为0或者数值一致的值，可以使用稀疏数组来表示，即将存在不同值的元素行和列以及元素值存在一个较小数组中。

二维数组 转 稀疏数组的思路：

1.遍历原始的二维数组，得到有效数据的个数sum；

2.根据sum就可以创建稀疏数组spaseArr int[sum + 1][3]；

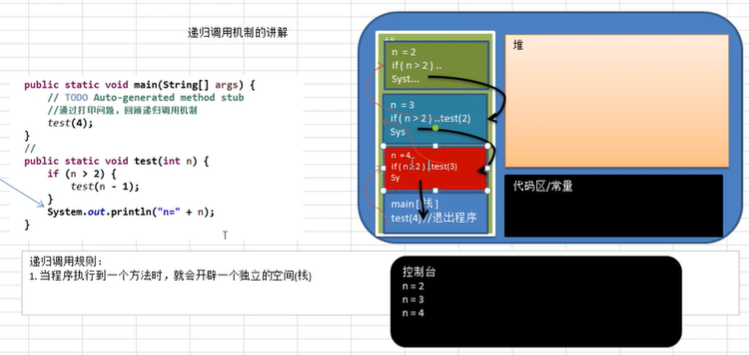
3.将二维数组的有效数据存入到稀疏数组中。

稀疏数组 转 原始的二维数组的思路：

1.先读取稀疏数组的第一行，根据第一行的数组，创建原始的二维数组；

2.再读取稀疏数组后几行的数组，并赋给原始的二维数组即可。

递归：



方法被创建在堆栈（即栈）中，调用方法时，将调用者先放入栈中，之后依次压入栈，因此当方法出错时，依次pop出方法出错的位置。

递归需要遵循的重要规则：

1.执行一个方法时，就创建一个新的受保护的独立空间（栈空间）；

2.方法的局部变量是独立 的，不会相互影响；

3.如果方法使用的是引用类型变量（比如数组），就会共享该引用类型的数据；

4.递归必须向退出递归的条件逼近，否则就是无限递归，出现StackOverflowError；

5.当一个方法执行完毕，或者遇到return，就会返回，遵守谁调用，就将结果返回给谁，同时当方法执行完毕或者返回时，该方法也执行完毕。

排序算法的介绍：

排序是将一组数据，依指定的顺序进行排列的过程。

排序的分类：

1.内部排序：指将需要处理的所有数据都加载到内部存储器中进行排序。

常用的有：插入排序（直接插入排序和希尔排序）、选择排序（简单选择排序和堆排序）、交换排序（冒泡排序和快速排序）、归并排序、基数排序

2.外部排序：数据量过大，无法全部加载到内存中，需要借助外部存储进行排序。



散列表：又称哈希表，是根据关键码值而直接进行访问的数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，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。

用途：可以在操作数据库时，将常用数据存放在哈希表中，以免频繁操作数据库，还可以在轻量级（不能使用缓存时），替代缓存

散列表分为两种：数据+链表实现、数据+二叉树实现。

数据+链表实现：实际是将链表头节点存放在数组中。

为什么需要树？

数组存储方式的分析：

优点：通过下标方式访问数组，查询快，对于有序数组，还可使用二分查找来提高检索速度

缺点：如果要检索具体某个值，或者插入（按一定顺序）会整体移动，效率较低

链式存储方式的分析：

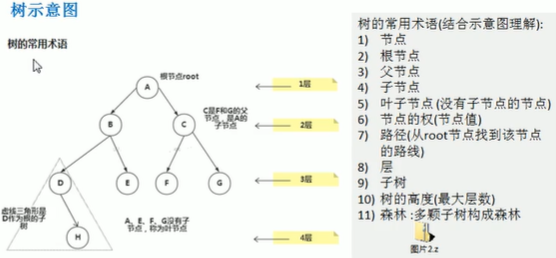
优点：增删节点的速度快

缺点：查询节点的速度慢

树存储方式的分析：

能提高数据存储、读取的效率，比如利用二叉排序树，既可以保证数据的检索速度，同时也可以保证数据的插入、删除、修改的速度

数组扩容：每次在底层都需要创建新数组，要将原来的数据拷贝到数组，并插入到新的数组



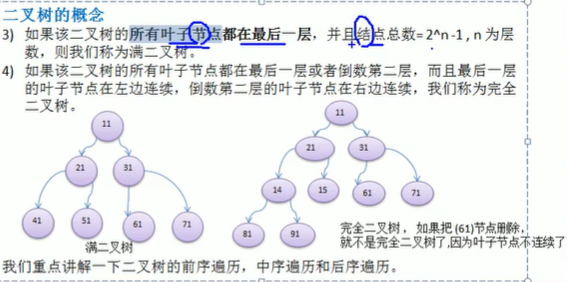
二叉树：

树有很多种，每个节点只能有2个子节点的一种形式称为二叉树；

二叉树的子节点分为左节点和右节点；

如果该二叉树的所有叶子节点都在最后一层，并且节点总数= 2 ^ n - 1，n为层数，则我们称为满二叉树（如三层的数共有7个节点）；

如果该二叉树的所有叶子节点都在最后一层或者倒数第二层，而且最后一层的叶子节点在左边连续，倒数第二层完全填充，称之为完全二叉树。



前序遍历：先输出父节点，再遍历左子树和右子树；

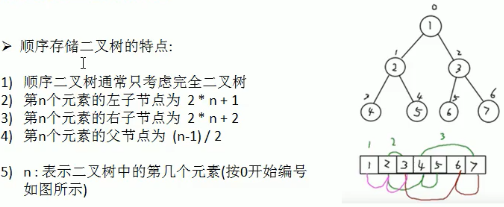
中序遍历：先遍历左子树，再输出父节点，再遍历右子树；

后序遍历：先遍历左子树，再遍历右子树，最后输出父节点；

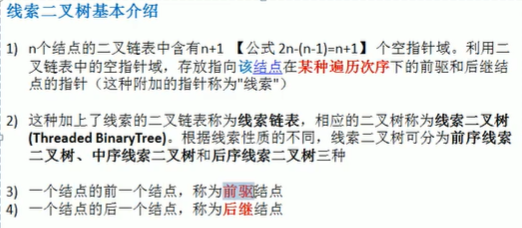
看输出父节点的顺序，就确定是前序、中序还是后序。

顺序存储二叉树的概念：

基本说明：从数据存储来看，数组存储方式和树的存储方式可以互相转换，即数组可以转换成树，树也可以转换成数组。



线索二叉树的介绍



公式推导过程：一共有n个节点，因此有2n个指针域，n – 1条边，因此：所有指针 – 已经使用的指针 = 空指针 => 2n – (n - 1) = n + 1

堆排序基本介绍：

堆排序是利用堆而设计出的排序算法，是一种选择排序。它的最好、最坏、平均时间复杂度均为O(nlogn)，它是不稳定排序。

堆是具有以下性质的完全二叉树：

大顶堆：每个节点的值都大于或等于其左右孩子节点。（不要求节点的左右孩子之间有值的关系）

特点：array[i]>=array[2\*i+1] && array[i]>=array[2\*i+2]

小顶堆：每个节点的值都小于或等于其左右孩子节点的值

特点：array[i]<=array[2\*i+1] && array[i]<=array[2\*i+2]

一般升序用大顶堆，降序用小顶堆。

堆排序思路：

1.将无序序列构建成一个堆，根据升降序需求选择大顶堆或小顶堆；

2.将堆顶元素与末尾元素交换，将最大元素“沉”到数组末端；

3.重新调整结构，使其满足堆定义，然后继续交换堆顶元素与当前末尾元素，反复执行调整+交换步骤，直到整个序列有序。

赫夫曼树：

基本介绍：给定n个权值作为n个叶子节点，构建一颗二叉树，若该树的带权路径长度（wpl）达到最小，称这样的二叉树为最优二叉树，也称为哈夫曼树（霍夫曼树）。

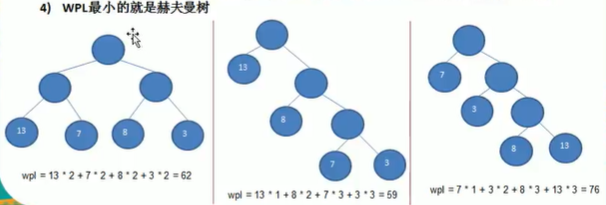
赫夫曼树是带权路径长度最短的树，权值较大的节点离根较近。

重要概念：

路径和路径长度：在一棵树中，从一个节点往下可以达到的孩子或者孙子节点之间的通路，称为路径。通路中分支的数目称为路径长度。若规定根节点的层数为1，则从根节点到第L层节点的路径长度为L – 1。

节点的权及带权路径长度：若将书中节点赋给一个有着某种含义的数值，则这个数值称为该节点的权。节点的带权路径长度为：从根节点到该节点之间的路径长度与该节点的权的乘积。

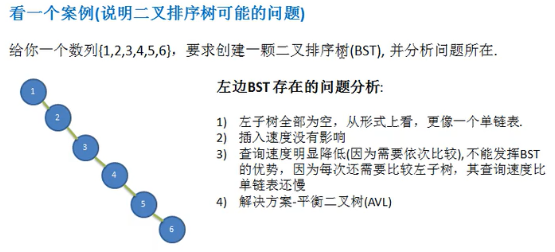
树的带权路径长度：树的带权路径长度规定为所有叶子节点的带权路径长度之和，记为wpl，权值越大的节点离根节点越近的二叉树才是最优二叉树。



二叉排序树：BST（Binary Sort Tree）对于二叉排序树的任何一个非叶子节点，要求左子节点的值比当前节点的值小，右子节点的值比当前节点的值大

Ps：如果有相同的值，可以将该节点放在左子节点或者右子节点。

BST存在的问题：如果给定一个有序数组，则创建出来的二叉排序树其左子树全部为空，更像是一个单链表。在对数据插入时，不受影响，但查询速度是明显降低的。因为需要依次比较，甚至每次还需要比较左子树，其查询速度比单链表还慢。



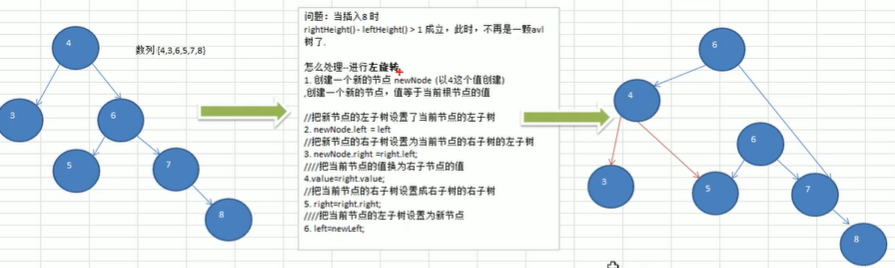
解决方法：平衡二叉树。

平衡二叉树：也叫平衡二叉搜索树，又被称为AVL树，可以保证查询效率较高。

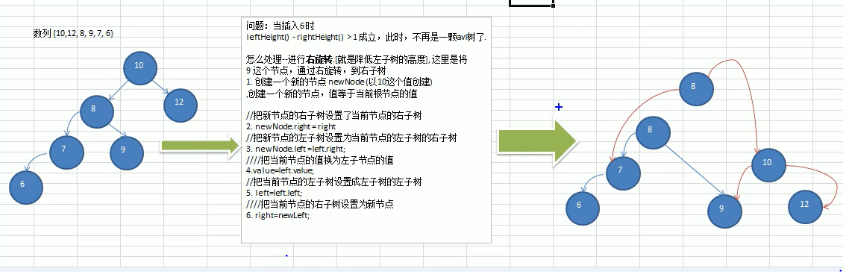
特点：它是 一颗空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是平衡二叉树，平衡二叉树的常用实现方法有红黑树、AVL、替罪羊树、Treap、伸展树等。



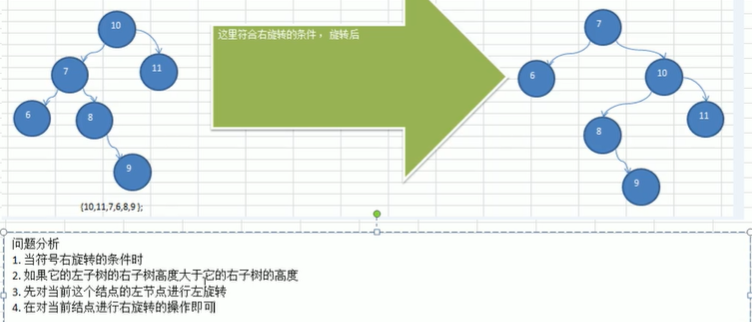
左旋转：



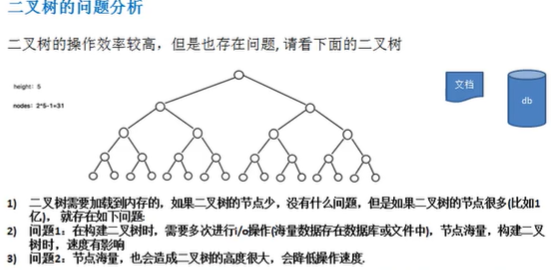
右旋转：



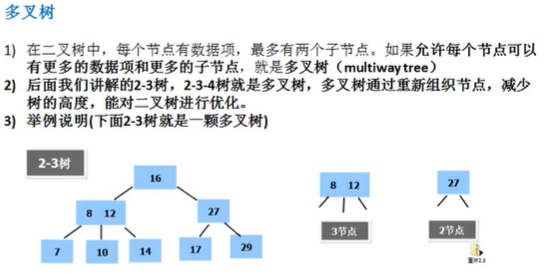
双旋转：



二叉树存在的问题：



多叉树：



2-3树基本介绍：

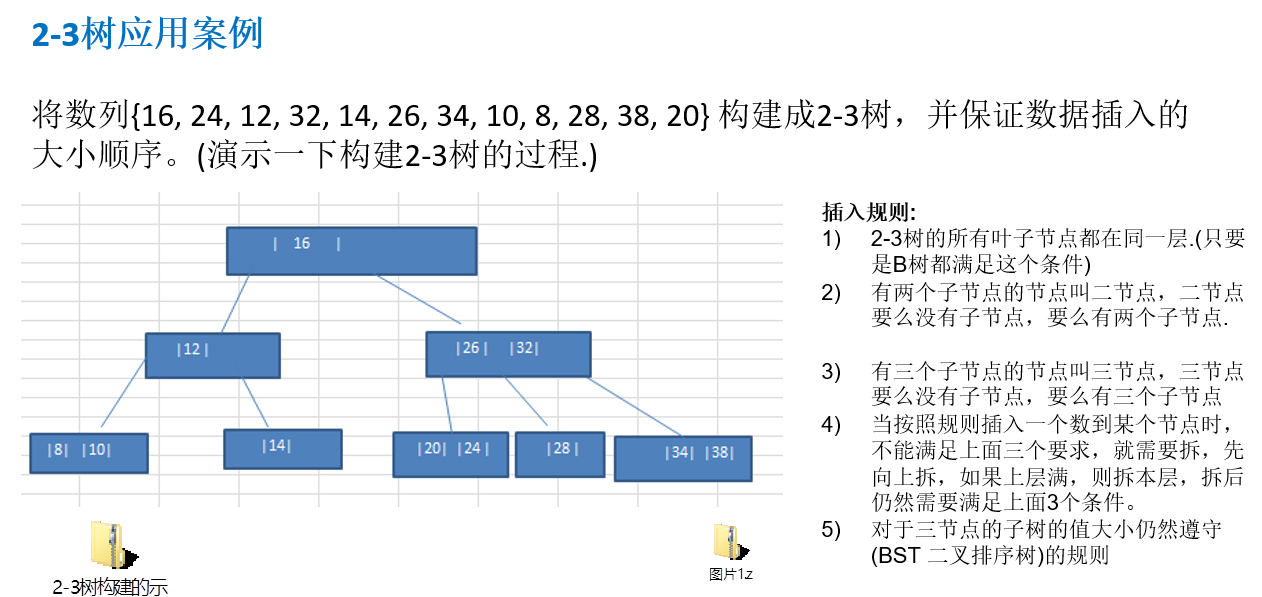
2-3树是最简单的B树结构，具有如下特点：

（1）2-3树的所有叶子节点都在同一层（只要是B树就满足该条件）；

（2）有两个子节点的节点为二节点。二节点要么没有子节点，要么有两个子节点；

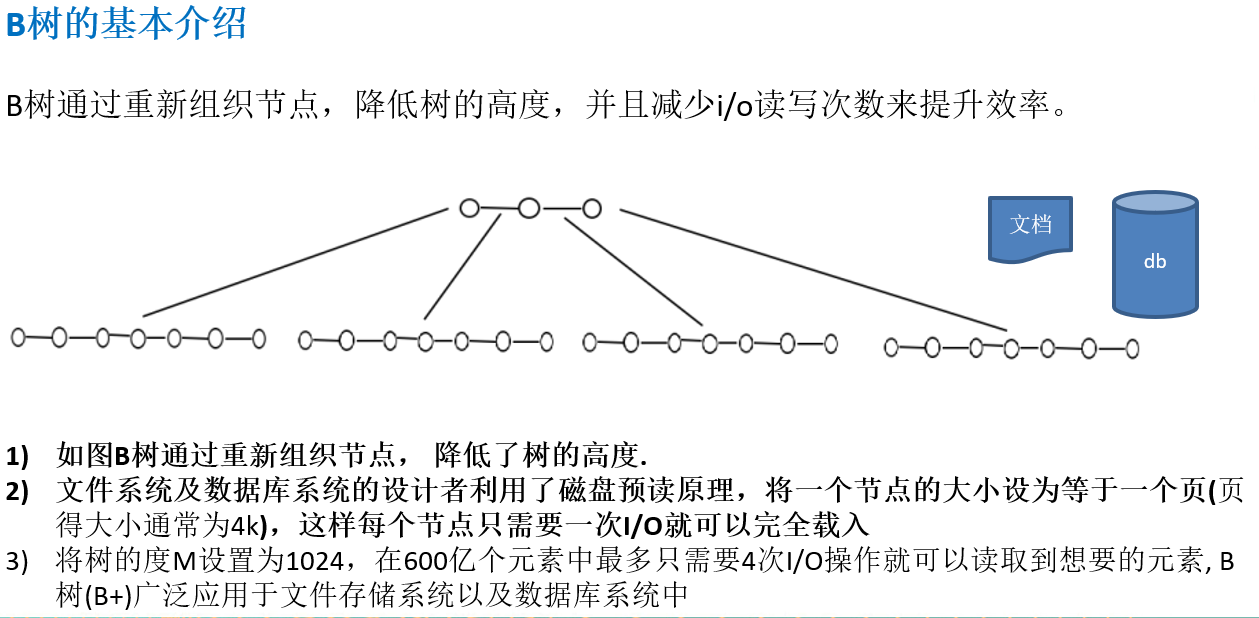
（3）有三个子节点的节点为三节点，三节点要么没有子节点，要么有三个子节点；

（4）2-3树是由二节点和三节点构成的树。

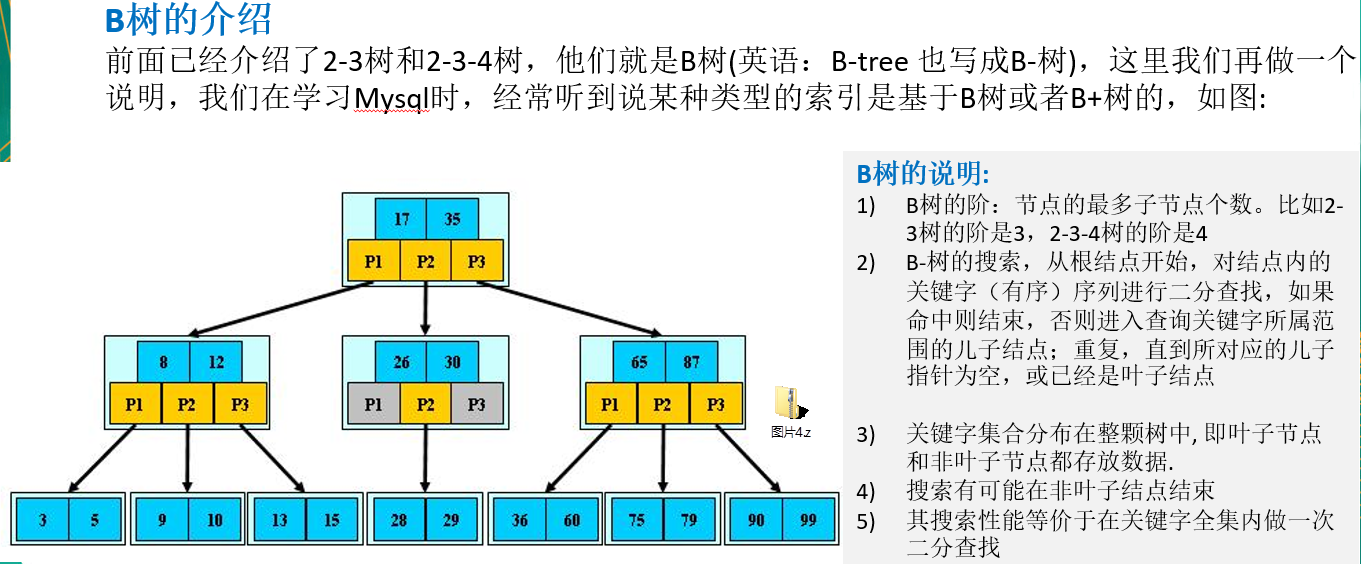


B树的简单介绍：

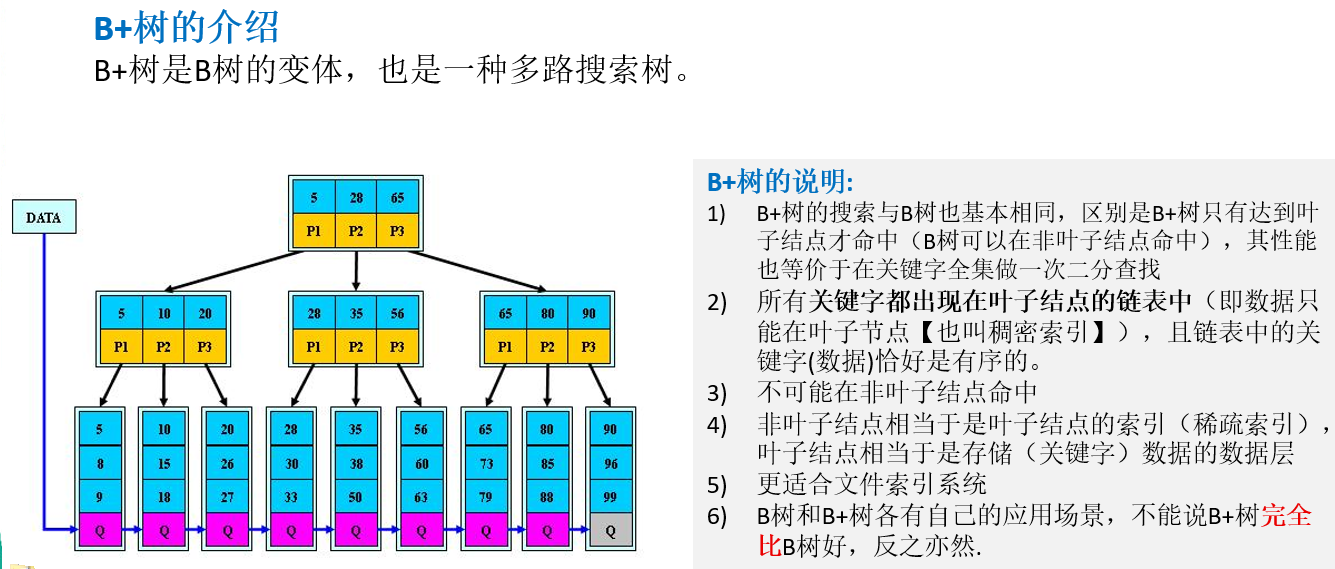
B树，又名B-树，即B-Tree



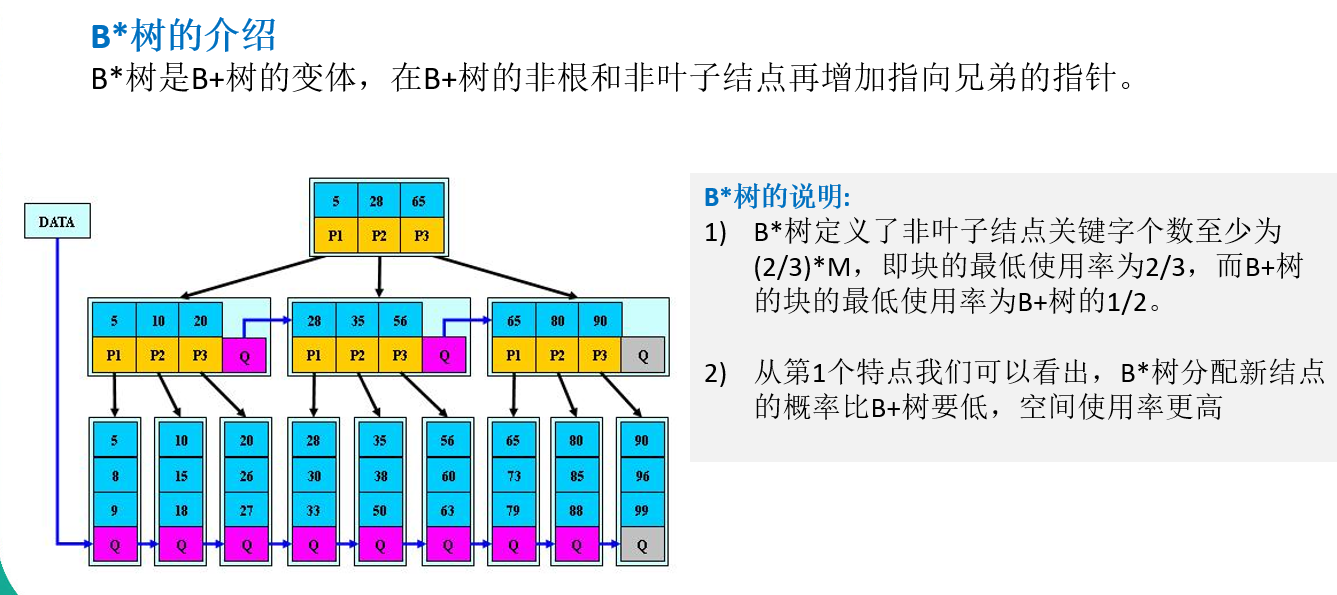
B树的说明：



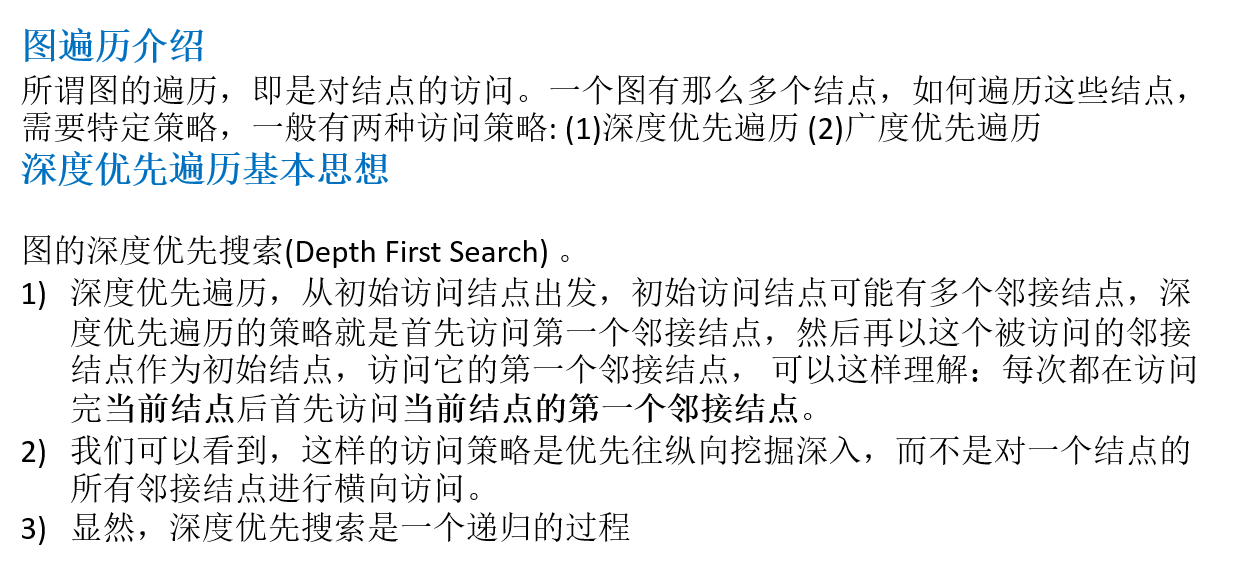
B+树：

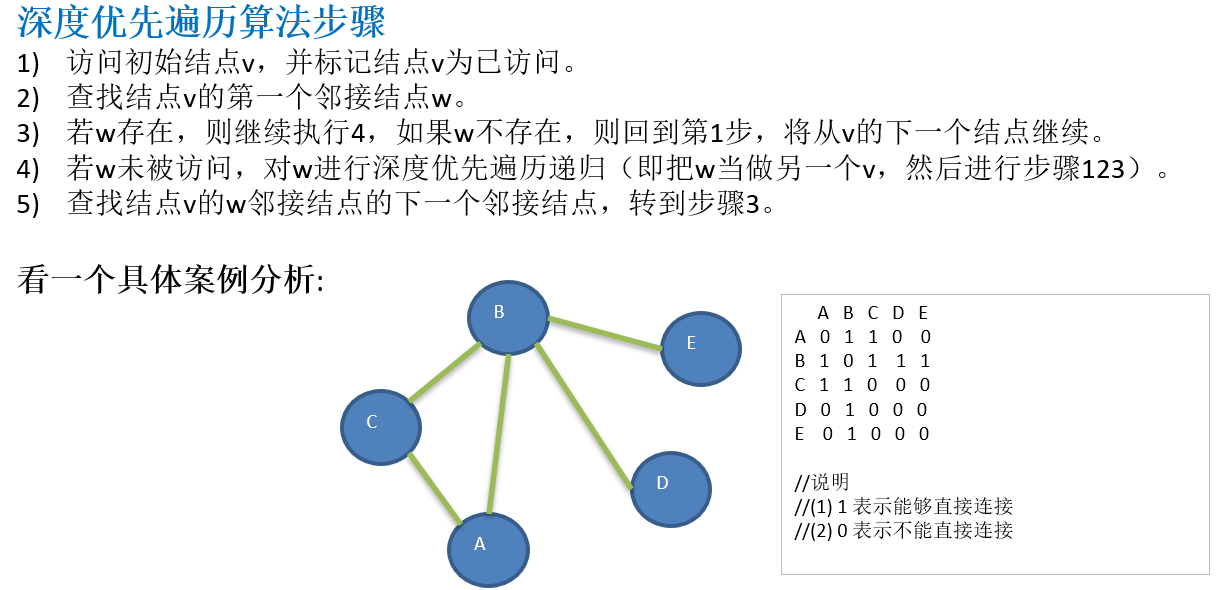


B\*树：



深度优先遍历：





广度优先遍历：

