数据结构周末夜校

Weekend evening classes for data structures courses

主讲人：谢滨冰

时间：　2025.05.30

湖南大学
HUNAN UNIVERSITY

目录

湖南大学
HUNAN UNIVERSITY

第一部分

复杂度分析

标题一 时间复杂度

$T(n) = O(f(n))$	$T(n) = \Omega(f(n))$	$T(n) = \Theta(f(n))$	$T(n) = o(f(n))$
存在一个正数c和n_0，使得对所有$n \geqslant n_0$，满足 $T(n) \leqslant c \cdot f(n)$	存在一个正数c和n_0，使得对所有$n \geqslant n_0$，满足 $T(n) \geqslant c \cdot f(n)$	存在一组正数c_1、c_2和n_0，使得对所有$n \geqslant n_0$，满足 $c_1 \cdot f(n) \leqslant T(n) \leqslant c_2 \cdot f(n)$	对所有$c>0$，存在一个n_0的选择，满足对所有$n \geqslant n_0$，都有 $T(n) < c \cdot f(n)$

最好情况复杂度
最差情况复杂度
平均情况复杂度

考试一般写这种

求和定理：假设两个已知程序片段的时间复杂度分别为T1(n)=O(f(n))和T2(n)=O(g(n))，那么顺序组合两个程序片段得到的程序的时间复杂度为：
T1(n)+T2(n)= O(Max(f(n), g(n)))

求积定理：假设两个已知程序片段的时间复杂度分别为T1(n)=O(f(n))和T2(n)=O(g(n))，那么交叉乘法组合两个程序片段得到的程序时间复杂度为：
T1(n)·T2(n)= O(f(n)·g(n))

标题一 时间复杂度

1.（10 分）分析下列程序的时间复杂度，给出其最佳、最差情况下的时间代价 Θ 表达式，要求写出具体的分析过程。

```
void insertHeap(int *nums,int n,int value) {
    nums[n] = value;
    while (n > 0) {
        int pre = (n-1) / 2;
        if (nums[pre] <= nums[n]) {
            break;
        }
        else {
            swap(nums[pre], nums[n]);
        }
        n = (n-1) / 2;
    }
}
```

标题一 空间复杂度

算法的空间复杂性：度量算法所使用的**辅助空间大小**和数据规模n之间的关系。

```
算法1-4：将数组中元素反转存放ReverseArray2（array）
输入：一维数组array，数组长度n≥0，数组的元素类型记为ElemSet
输出：反转后的数组array
1.  n ← array.size
2.  b← new ElemSet[n] //创建一个同类型和大小的辅助数组
3.  for i←1 to n do
4.  | b[i] ← array[n-i+1]
5.  end
6.  for i←1 to n do
7.  | array[i] ← b[i]
8.  end
9.  delete b
```

算法**额外**使用了大小为n的辅助数组b
空间复杂度就是O(n)

湖南大学
HUNAN UNIVERSITY

第二部分

线性表、栈与队列

顺序表

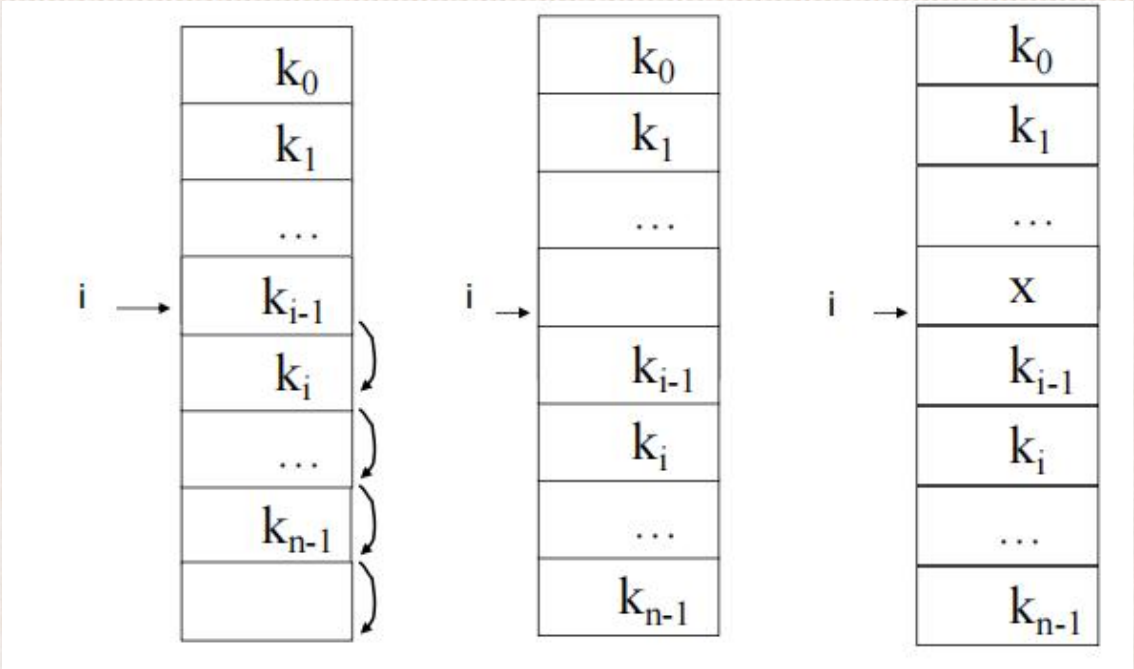
1.初始化: 顺序表的初始化即构造一个空表

2.查找：在线性表中查找与给定值x相等的数据元素
朴素查找O(n)、二分查找……（第11章的内容）

3.查询：在顺序表中找到下标为i的元素
T(n)=O(1)

4.插入：在表的第i个位序上插入一个值为x的新元素
$(a_1,a_2,...,a_{i-1},a_i,a_i+1,...,a_n)$ ->
$(a_1,a_2,...,a_{i-1},x,a_i,a_i+1,...,a_n)$
插入后的元素都要向后移一个单位
T(n)=O(n)

5.删除：将线性表中的第i个位序上的元素删除
$(a_1,a_2,...,a_{i-1},a_i,a_i+1,...,a_n)$ ->
$(a_1,a_2,...,a_{i-1},a_i+1,...,a_n)$
插入后的元素都要向前移一个单位
T(n)=O(n)



单向链表

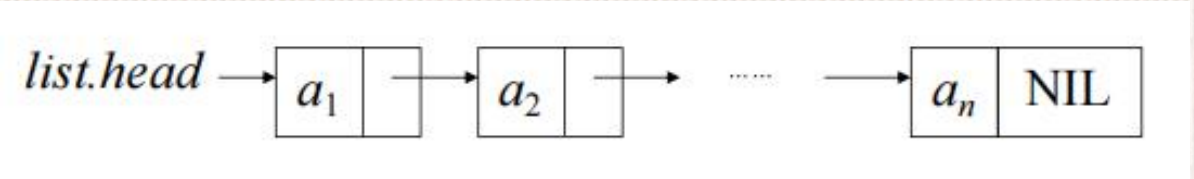
1. 求表长: 求单链表元素个数 T(n)=O(n)

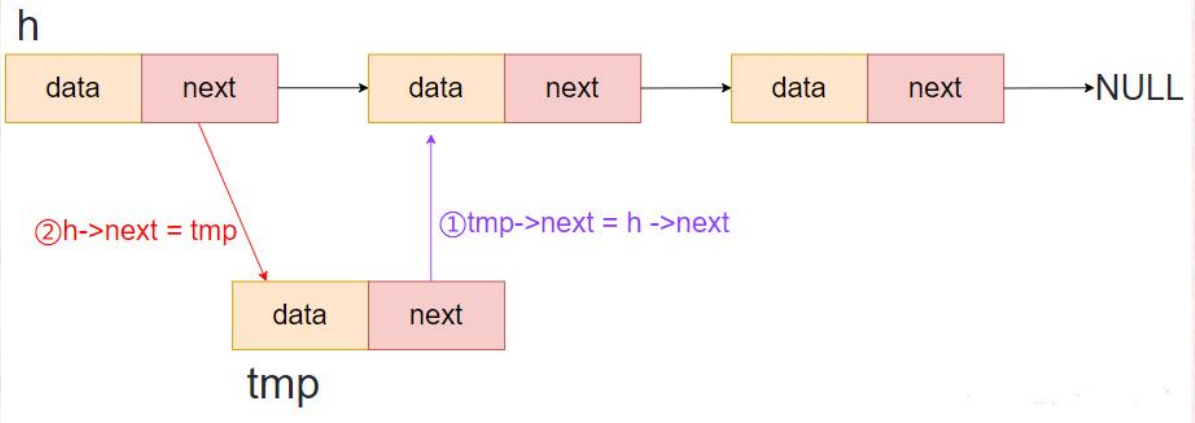
2. 查找：分按序号查找Get (list, i)、按值查找Search(list, x)　　T(n)=O(n)

3. 插入：在list的第i个位置上插入元素x
　　T(n)=O(n)
　　注意：要判断链表是否为空

4. 删除：在单链表中删除指定位序i的元素
　　T(n)=O(n)

双向链表、循环链表、块状链表





2.（10 分）在线性表的以下链式存储结构中，若未知链表头结点的地址，仅已知 p 指针指向的结点，能否从中删除该结点？请简述原因。

(1) 单链表。

(2) 双链表。

(3) 循环单链表。

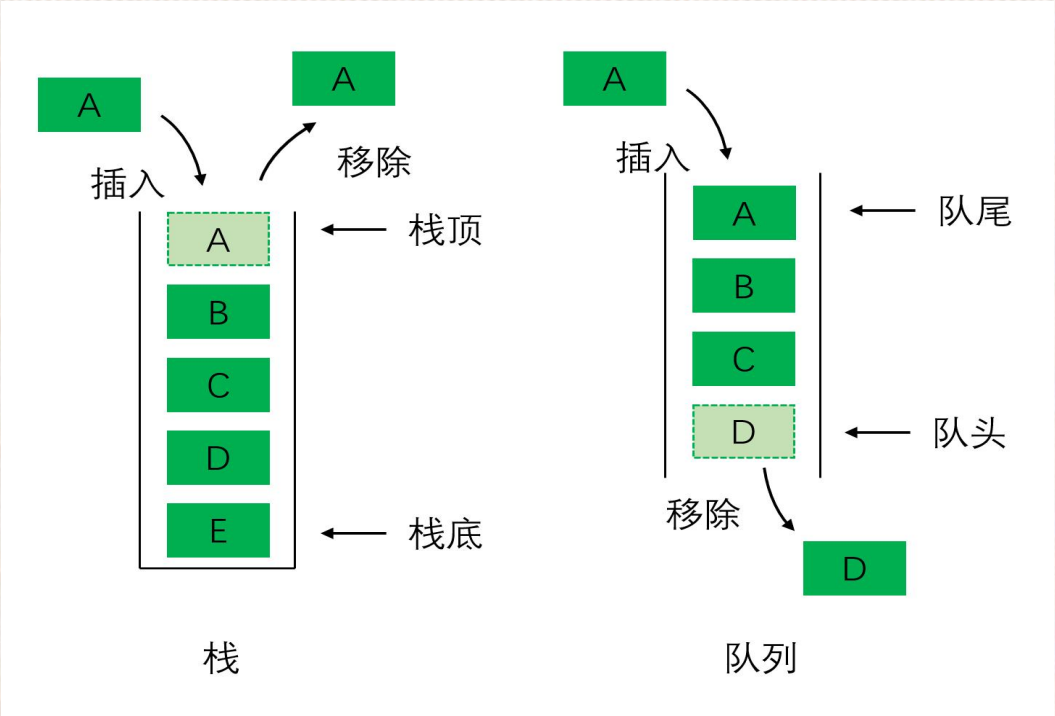
栈与队列都是受限线性表

栈（Stack）允许插入和删除的一端称为栈顶 (top)，另一端称为栈底 (bottom)

特点：后进先出 (LIFO)

队列（Queue）允许插入一端称为队尾(rear)，另一端称为队首(front)

特点：先进先出 (FIFO)



2.（5分）有一个火车站，其车厢调度过程可以用一个栈来模拟，若车厢的入栈的序列为0，1，2，3，4，则车厢可能的出栈序列是以下答案中的哪一个？简要说明其它三个答案错误的原因。

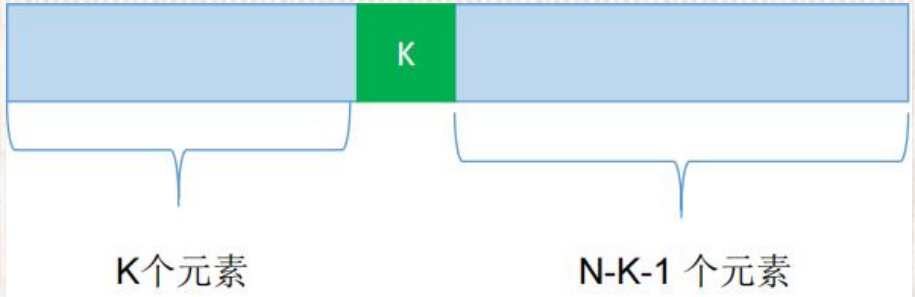
A. 03142　　B.41203　　C.12340　　D.40123

一、（10 分）假设某个用例程序会进行一系列入栈操作（I）和出栈操作（0）。入栈操作会将整数 0 到 9 按顺序压入栈；出栈操作会打印出返回值。请分别判断下面的序列是否可能产生。如果能产生，请说明操作序列（依次用 I 或 0 的序列表示）；如不能产生，请结合画图说明理由。

(1) 4 3 2 1 0 9 8 7 6 5

(2) 4 6 8 7 5 3 2 9 0 1

给定一个入栈序列，序列长度为N，请问有多少种出栈序列?



$$f(N)=\sum_{k=0}^{n-1} f(k)\times f(N-1-k)=\frac{1}{N+1}\times C_{2N}^{N}$$

卡兰特数

表达式求值21

二、（10 分）已知操作数栈 S_1 和操作符栈 S_2 如下图所示（见下页），其中 S_1 的栈顶元素为 2，S_2 的栈顶元素为+，栈底的$为结束标记。函数 EXP（）的作用如下：

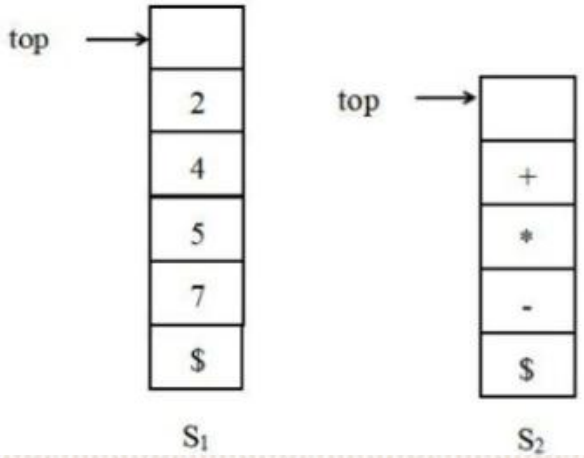
① 从栈 S_1 中依次弹栈 2 次分别存储在 $OPRD_2$ 和 $OPRD_1$（第一次弹栈存储在 $OPRD_2$，第二次弹栈存储在 $OPRD_1$）；

② 从栈 S_2 中弹栈一次存储到 OP；

③ 执行指令 $OPRD_1$ OP $OPRD_2$，把运算结果压入 S_1。

（1）请参照下图画出第一次调用 EXP（）后的两个栈 S_1 和 S_2 的示意图；

（2）反复调用 EXP（）直至栈 S_1 中只剩 1 个操作数和结束标记$，画出此时栈 S_1 的示意图。



表达式求值

1）后缀表达式求值

①遍历后缀表达式，如果遇到数字则直接入栈，如果遇到操作符，则弹出栈顶的两个元素，进行计算后将结果入栈。

②最终，栈内剩余的元素就是整个表达式的计算结果。

2）中缀表达式转后缀表达式

①遇到数字就直接输出到后缀表达式中，遇到操作符就判断其优先级，并将其压入栈中。

②如果栈顶元素的优先级大于等于当前操作符，则先将栈顶元素弹出并输出到后缀表达式中，再将当前操作符压入栈中。

③如果遇到了左括号，则直接将其压入栈中，如果遇到了右括号，则弹出栈中的元素，直到遇到了左括号为止，并将这些元素输出到后缀表达式中。

④最后，将栈中剩余的元素依次弹出，并输出到后缀表达式中。

湖南大学
HUNAN UNIVERSITY

第三部分

字符串

标题三 字 符 串

顺序存储结构

1.字符串插入

时间复杂度：最坏情况是将t插在s的头，时间复杂度是O(n+m)，其中n和m分别是两个字符串的长度。

2. 字符串删除

最坏情况是将s开头的len个字符删除，时间复杂度是O(n)，其中n是s的长度。

3. 字符串截取

时间复杂度：该操作仅与子串长度有关，为O(len)

4. 字符串连接

时间复杂度：该操作仅与t的长度有关，为O(m)

5. 字符串比较

仅与s和t的最小长度有关，为O(min(n, m))

链式存储结构

1.字符串插入

该操作时间复杂度是O(n+m)，其中n和m分别是两个字符串的长度。

2. 字符串删除

时间复杂度：这个操作的时间复杂度与s的长度有关，为O(n)

3. 字符串截取

最坏时间复杂度为O(n)

4. 字符串连接

这个操作的时间复杂度与字符串t的长度无关，仅与字符串s的长度成正比，为O(n)。

5. 字符串比较

时间复杂度：这个操作的时间复杂度也是仅与s和t的最小长度有关，为O(min(n, m))

标题三 字符串的模式匹配

在字符串s中找出与字符串t相等的子串的操作称为字符串的模式匹配，又称为子串的定位操作。其中字符串s称为主串或目标串，字符串t称为模式串。

模式匹配算法：朴素字符串匹配算法（BF算法）、KMP算法、BM算法、KR算法、Sunday算法等。

BF算法：

将模式串t的第0位字符与目标串s的第0位字符对齐，然后依次比对每个字符，若都相等，则匹配成功；若s和t对应位置上的字符不相等，则将t后移1位重新从模式串t的第0位开始依次比对。

最好情况下，仅需匹配m次，此种情况时间复杂度为O(m)

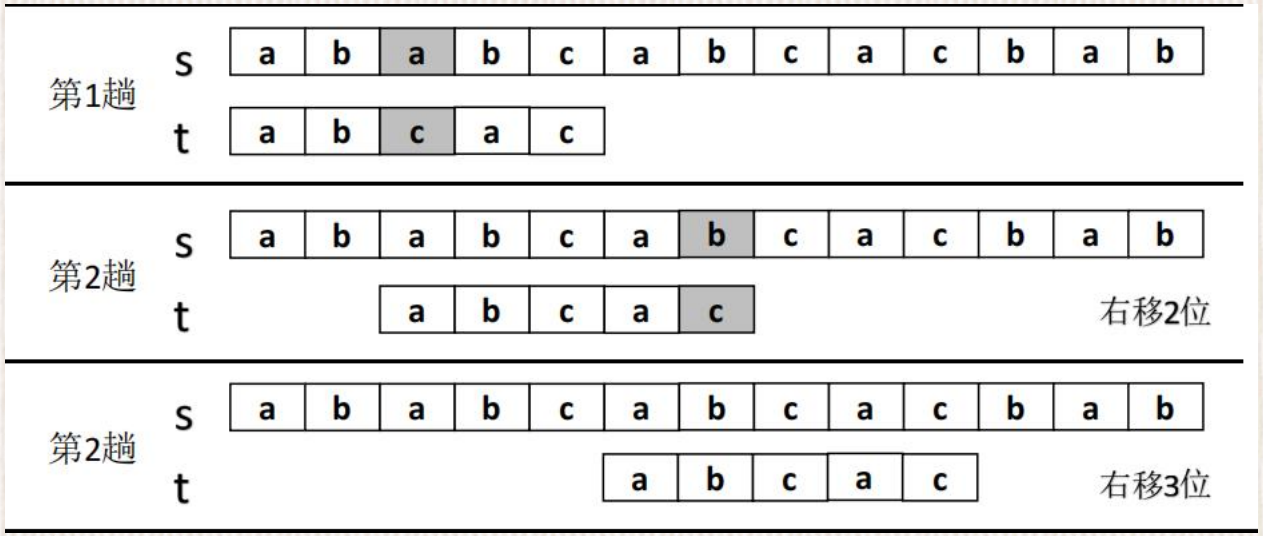
最坏情况下，需要移动n-m+1次，每次匹配m次，时间复杂度为O(nm-m2)，即O(nm)

标题三 KMP算法

KMP算法：

假设BF算法在失配时已经匹配到了字符串的第 j 位，则说明目标串与模式串的前 j-1 位是匹配成功的，利用已匹配的信息可对BF算法进行优化。

例：目标串s="ababcabcacbab"，模式串t="abcac“



标题三 KMP算法

字符串特征向量：长度为m的字符串t的特征向量是一个m维向量，通常记作next，且用数组形式进行存储，所以特征向量也可非正式地称为next数组。next[i]表示字符串特征向量的第i位分量（0≤i<m），其形式化定义为：

$$next[i]=\begin{cases}\text{满足}t[0...k]=t[i-k...i]\text{的最大}k(\text{存在}k，\text{且}k<i)\\ -1，\qquad\qquad\qquad\text{如果这样的}k\text{不存在}\end{cases}$$

next[i]表示的真实含义是字符串t的子串t[0...i]的最长公共前后缀中的前缀最末尾的字符的位置。由于字符串从0位开始，所以t[0...i]的最长公共前后缀长度为next[i]+1。如果t[0...i]的最长公共前后缀不存在，那么将next[i]置为-1。

例：字符串"abcac"的特征向量为[-1,-1,-1,0,-1]

i	子串 *t[0...i]*	前缀集合	后缀集合	最长公共前后缀	特征向量第*i*位分量
i=0	a	空	空	空	-1
i=1	ab	a	b	空	-1
i=2	abc	a,ab	c,bc	空	-1
i=3	abca	a,ab,abc	a,ca,bca	a	0
i=4	abcac	a,ab,abc,abca	c,ac,cac,bcac	空	-1

标题三 KMP算法

若朴素计算next数组，时间复杂度为O(m3)，不能满足进行快速模式匹配的要求

一种时间复杂度为 O(m)的字符串特征向量算法：

如果t[i]与t[next[i-1]+1]相等，则tb加上字符t[i]构成的子串与ta加上字符t[next[i-1]+1]构成的子串能够完全匹配，这两个子串即为t[0...i]的最长公共前后缀，故可推出next[i]=next[i-1]+1

如果当前t[i]与t[next[i-1]+1]不同，则说明tb与其后一个字符形成的子串t[0...i]的后缀，与tb与其后一个字符形成的子串t[0...i]的前缀并不匹配。所以，在求解子串t[0...i]的最长公共前缀时需要进一步缩小搜索范围，在子串t[0...next[i-1]]中去寻找可能的最长公共前后缀，即为t[0...i]的最长公共前后缀，而子串t[0...next[i-1]]的最长公共前后缀的前缀的末位下标即 为 n e x t [n e x t [i - 1] + 1] 。 在 此 种 情 况 下 ， 有next[i]=next[next[i-1]+1]成立。

例：abaabc

时间复杂度：算法执行过程中目标串只向右移动，比较的复杂度为O(n)。
计算next数组的复杂度为O(m)，其中m为模式串的长度，故
KMP算法的时间复杂度为O(n+m)

```
算法4-13 求解字符串t的next数组GetNext (t, next)

输入：字符串t
输出：字符串t的next数组
m ← t.length
next[0] ← -1
for i←1 to m-1 do    //求出next[1]~next[m-1]
  j ← next[i-1]
  while j ≥ 0 且 t.data[i] ≠ t.data[j+1] do
    j ← next[j]
  end
  if t.data[i]=t.data[j+1] then
    next[i] ← j+1
  else
    next[i] ← -1
  end
end
```

湖南大学
HUNAN UNIVERSITY

第四部分

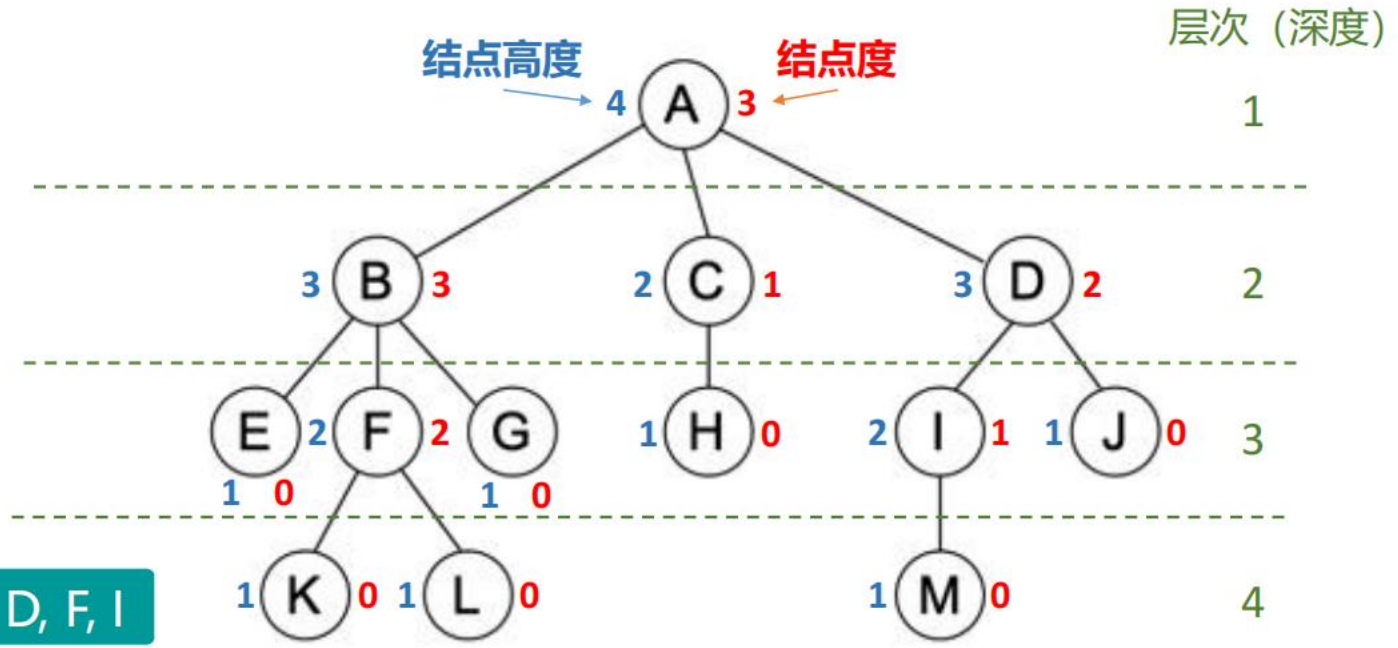
树与二叉树

标题四 树的基本术语

树的基本术语

树的高度：4

树的度：3



中间结点：A, B, C, D, F, I

叶结点：E, G, H, J, K, L, M

兄弟结点：{B, C, D} / {E, F, G} / {I, J} / {K, L}

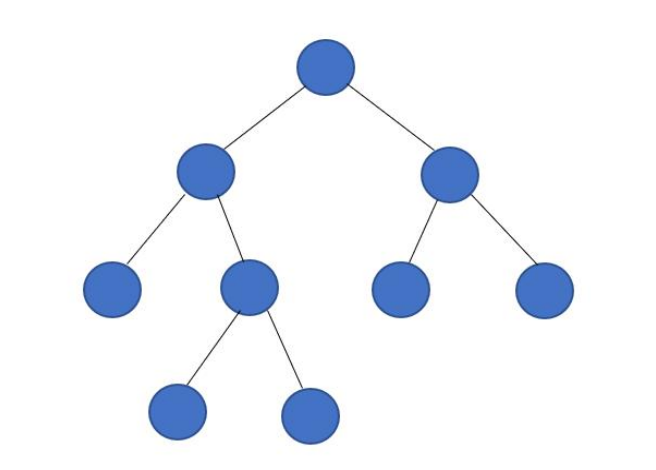
标题四

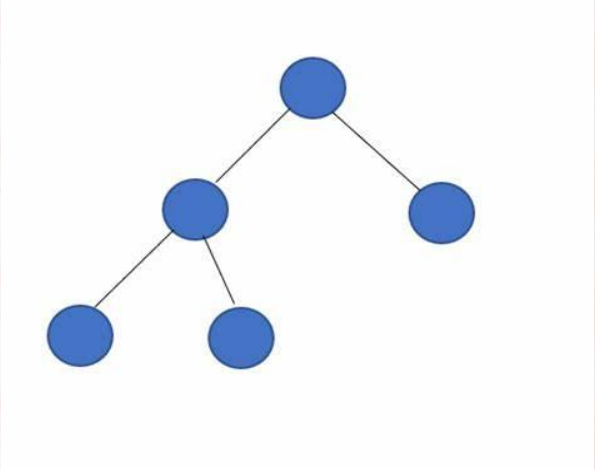
树的基本术语

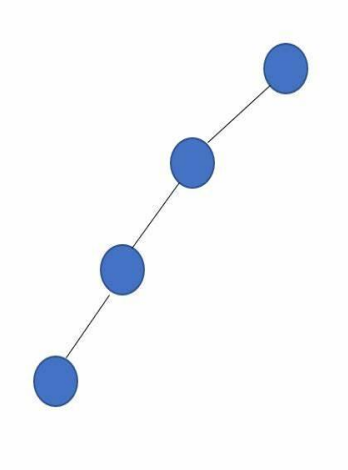
满二叉树：由度为0的叶结点和度为2的中间结点构成的二叉树，树中没有度为1的结点

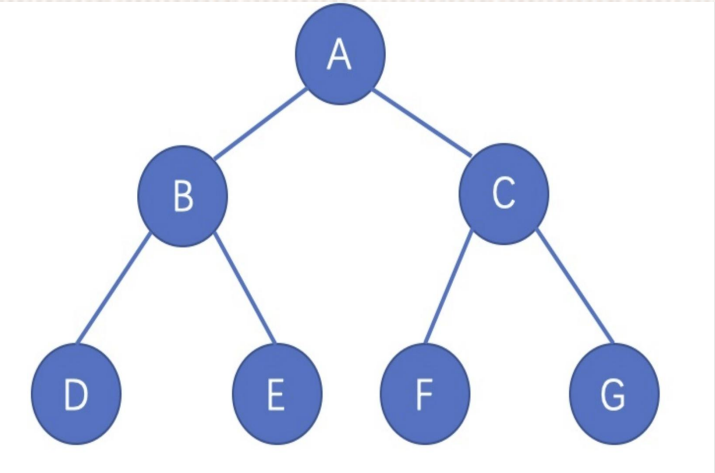
完全二叉树：从第1层到第d−2层全是度为2的中间结点;第d层的结点都是叶结点，度为0;在第d−1层，各结点的度从左向右单调非递增排列，同时度为1的结点要么没有，要么只有一个且该结点的左子树非空

完美二叉树：对于高度（深度）为d ≥ 1 的完全二叉树，如果第d-1层所有结点的度都是2，则该树是一个完美二叉树









标题四 树的基本性质

①设非空二叉树中度为i ∈[0 ,2]的结点数为n_i ，则$n_0 = n_2 + 1$ 。

②（满二叉树定理）非空满二叉树中叶结点数等于中间结点数加1。

③二叉树的第n 层最多有 2^{n-1} 个结点 (n ≥ 1) 。

④深度（高度）为d(≥ 1)的二叉树最多有 $2^d - 1$ 个结点。

⑤深度（高度）为d(≥ 1)的二叉树是完美二叉树的充分必要条件是树中有 $2^d - 1$ 个结点。

⑥完全二叉树有d个结点(d ≥ 1)，按层次从左向右连续编号。树中任一结点k (1 ≤ k ≤ d)满足以下性质

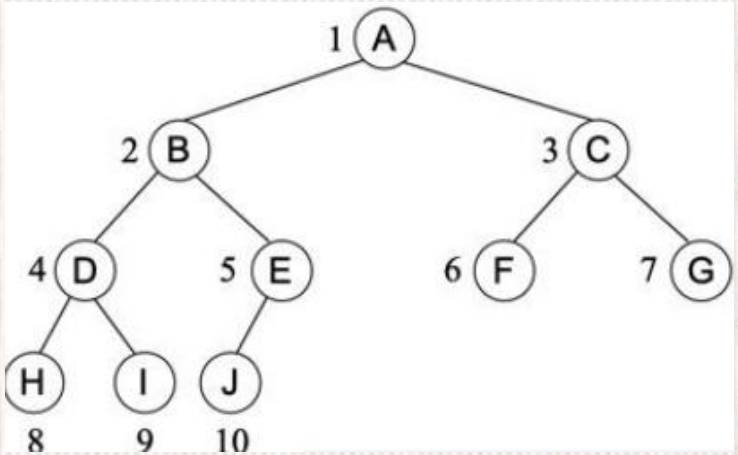
1) 如果2k ≤ n，则结点 k 的左子结点是 2k，否则没有左子结点；
2) 如果2k+1 ≤ n，则结点 k 的右子结点是 2k+1，否则没有右子结点；
3) 如果 k > 1，则结点 k 的父结点是 k/2（向下取整）。

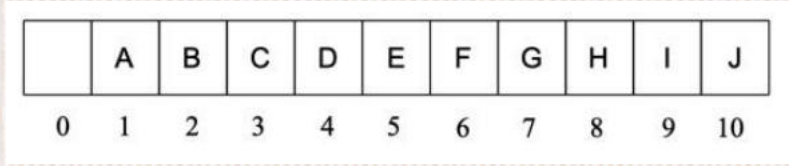
⑦有n个结点(n ≥ 1)的完全二叉树的深度$d = \log_2 (n + 1)$（向上取整） 。

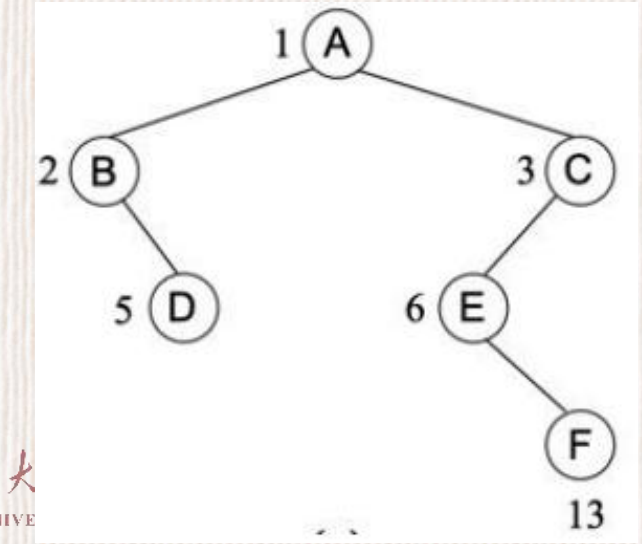
由⑥⑦得，$2^{d-1} - 1 < n \le 2^d - 1$

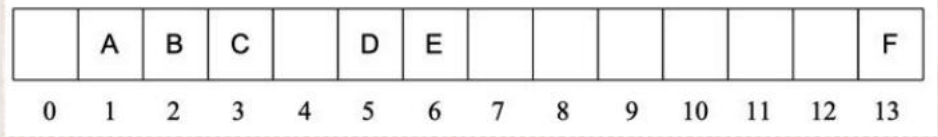
标题四 二叉树的存储方式

顺序存储



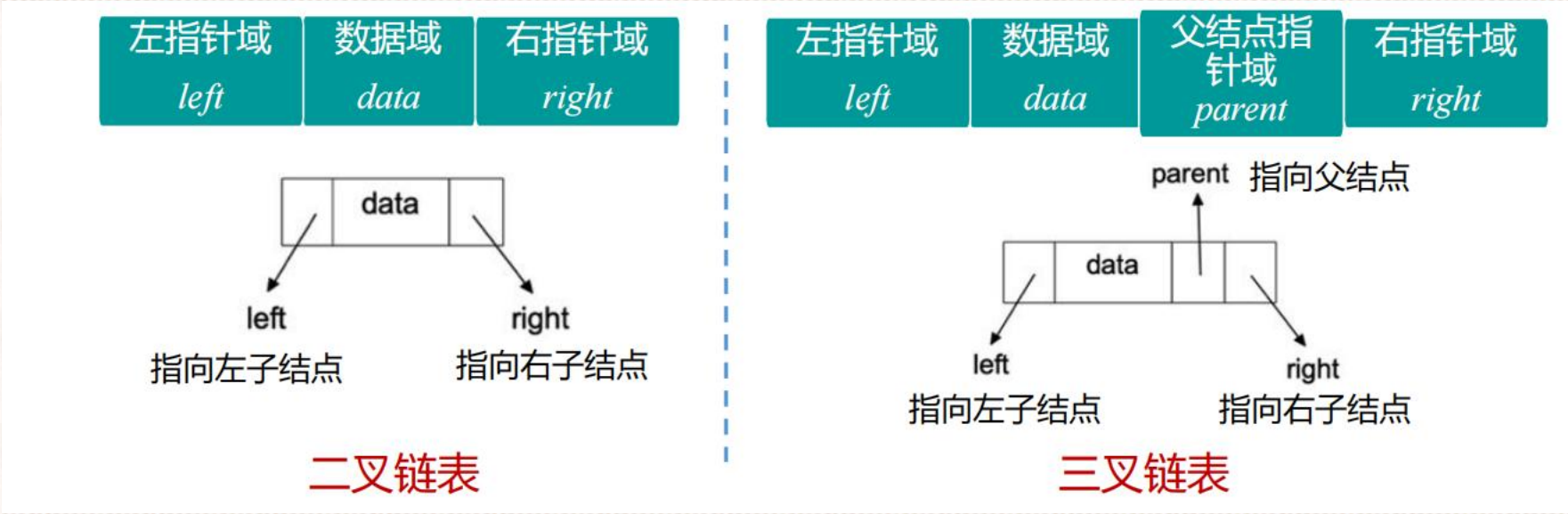






标题四 二叉树的存储方式

链式存储



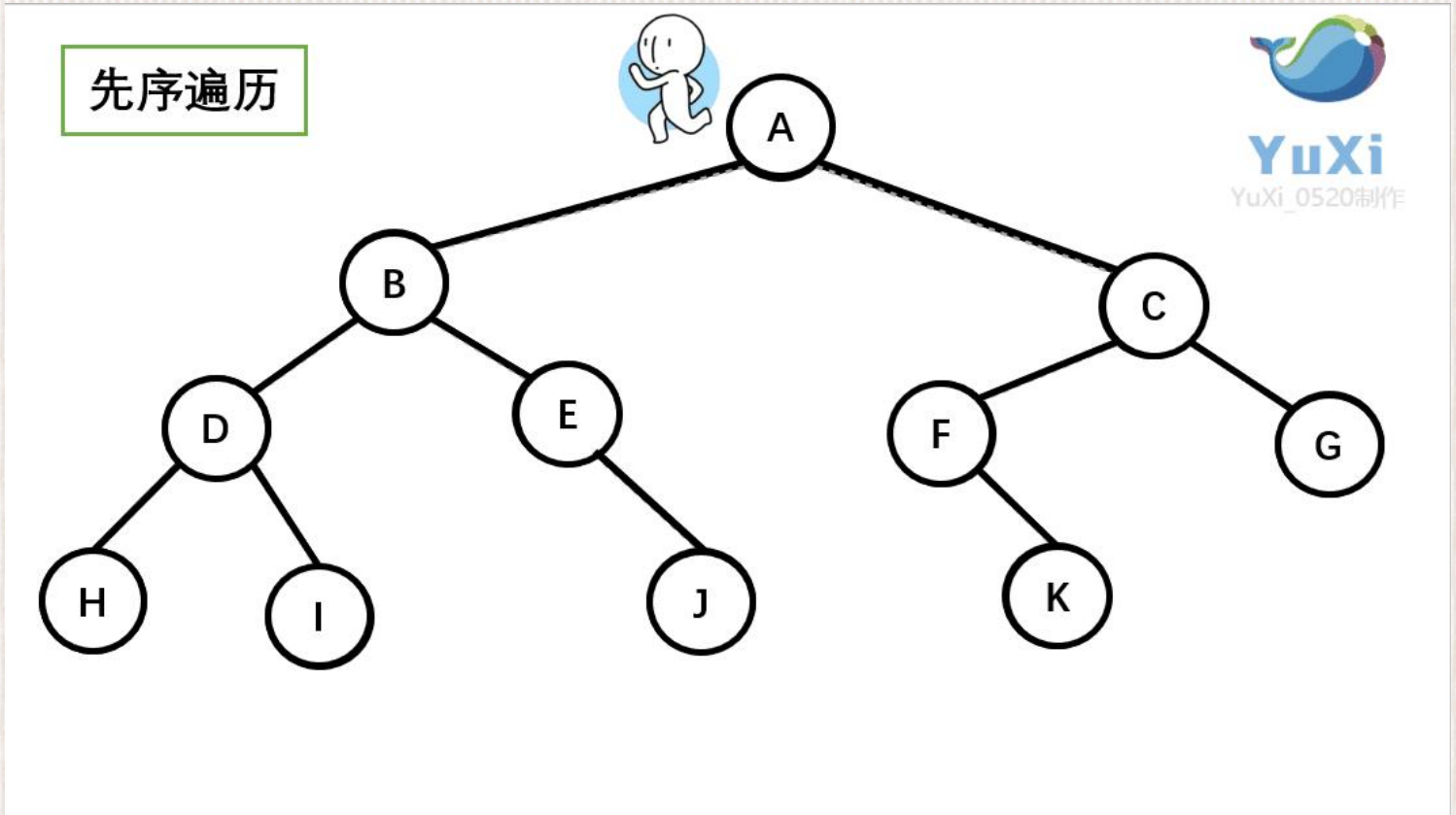

```
struct Treenode{
    int data;
    Treenode* left,right;
};
```

标题四 二叉树的遍历

前序遍历(DFS)：从根结点出发，先访问该结点，然后前序遍历该结点的左子树，再然后前序遍历该结点的右子树
中序遍历：从根结点出发，先中序遍历该结点的左子树，然后访问该结点，再然后中序遍历该结点的右子树
后序遍历：从根结点出发，先后序遍历该结点的左子树，然后后序遍历该结点的右子树，再然后访问该结点
层序遍历(BFS)：按照树的层级从上到下、从左到右访问所有节点

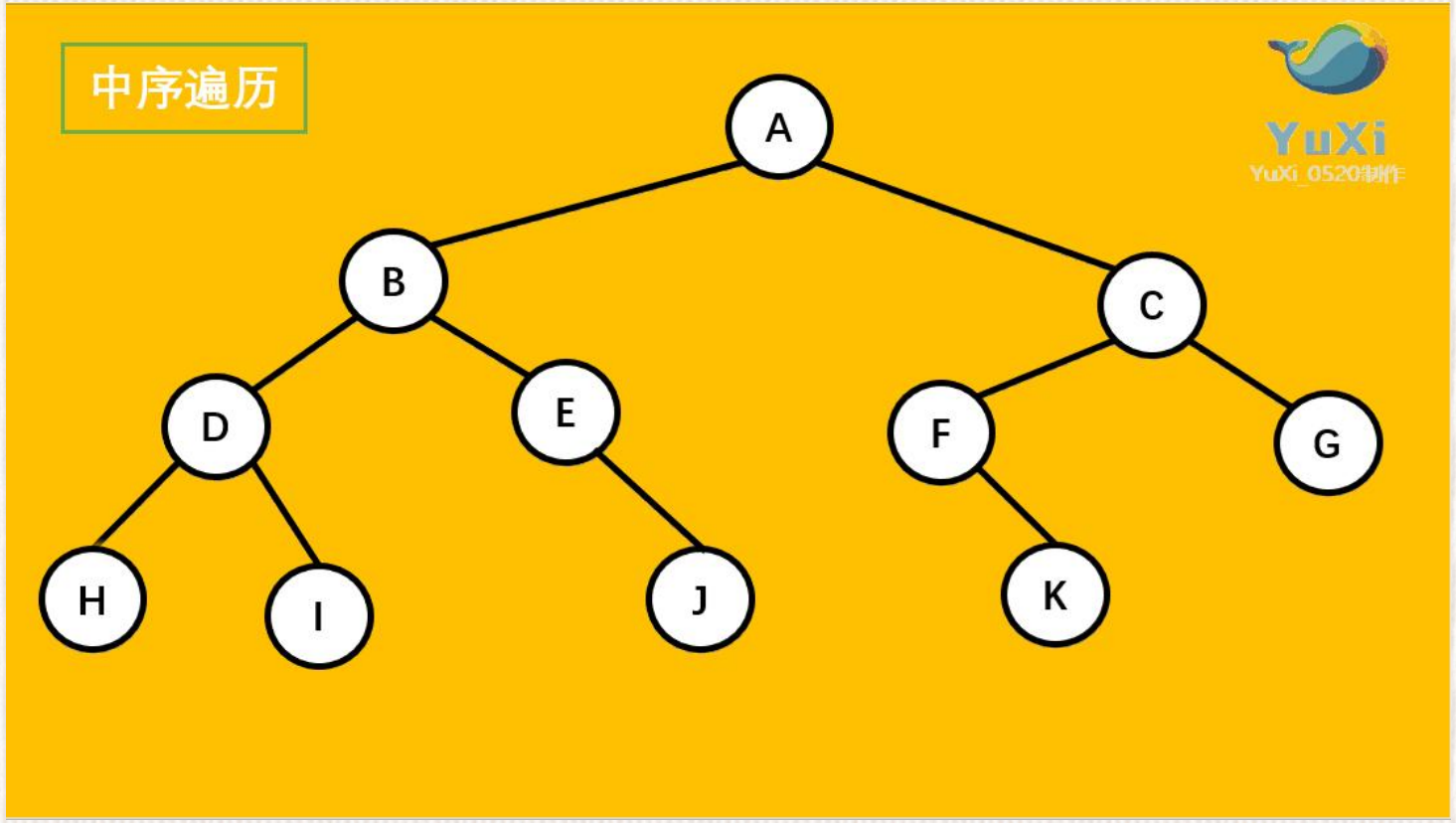
前序遍历可以想象成，小人从树根开始绕着整棵树的外围转一圈，经过结点的顺序就是先序遍历的顺序

可以利用栈实现
非递归的DFS



前序遍历结果：
ABDHIEJCFKG

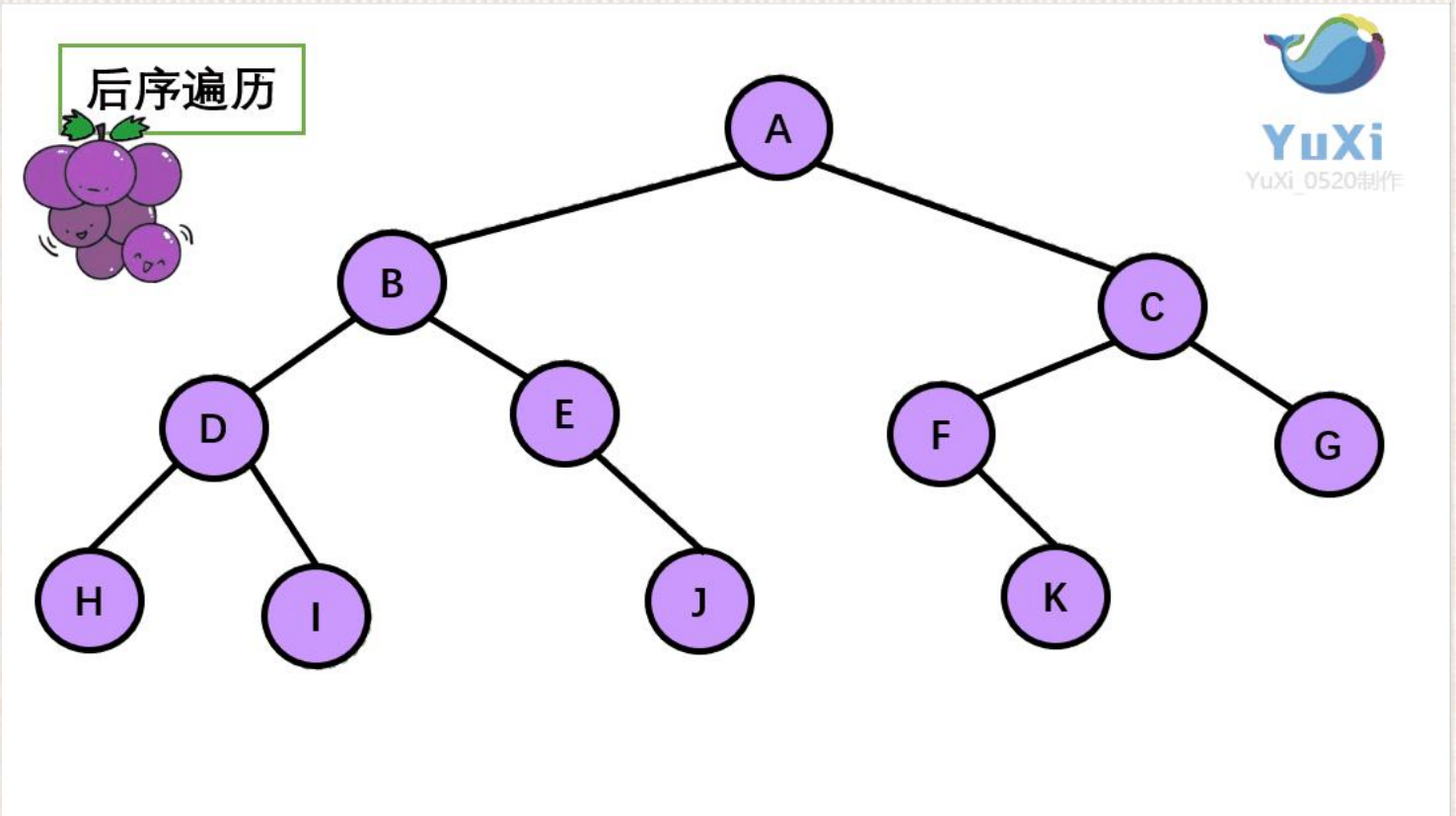
中序遍历可以想象成，按树画好的左右位置投影下来就可以了



中序遍历结果：HDIBEJAFKCG

标题四 二叉树的遍历

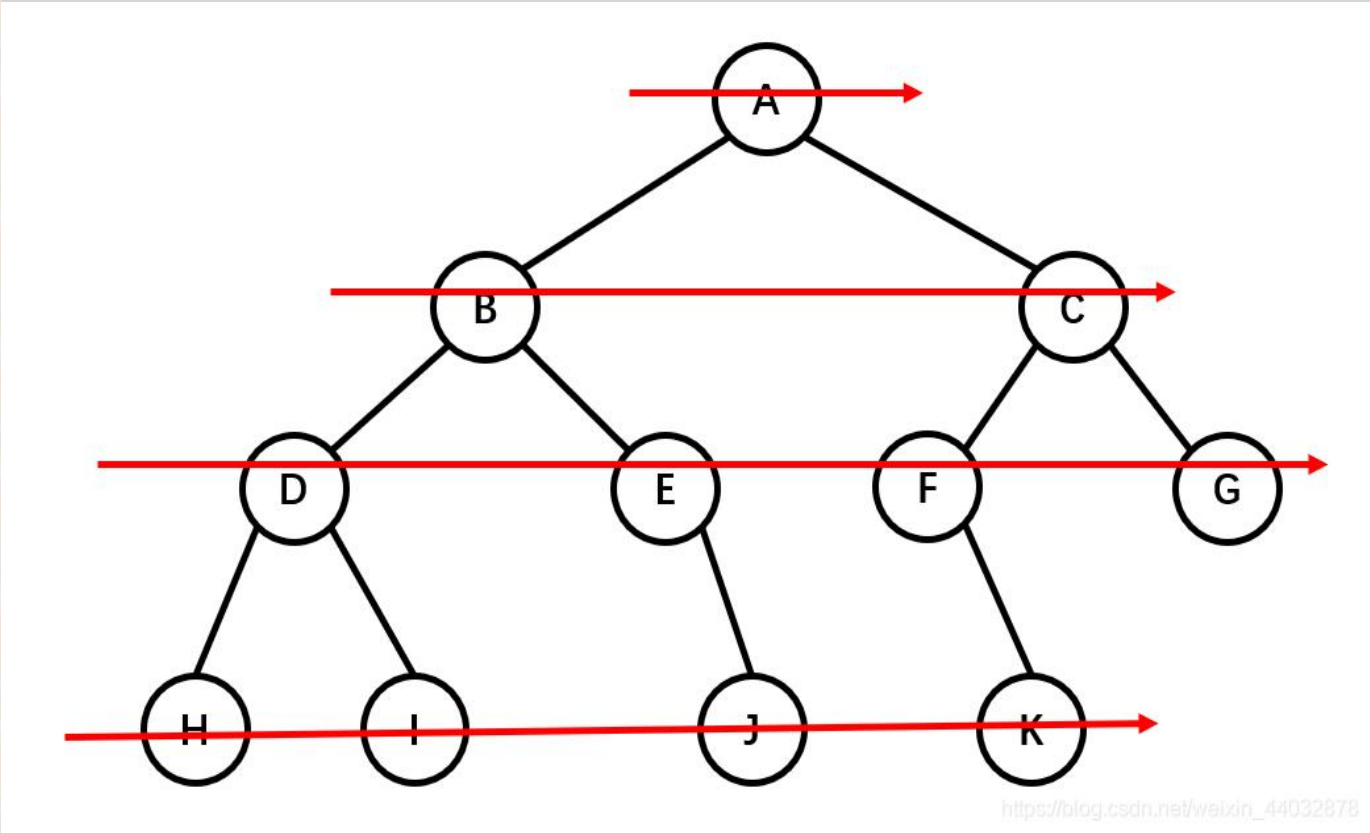
后序遍历就像是剪葡萄，我们要把一串葡萄剪成一颗一颗的。
就是围着树的外围绕一圈，如果发现一剪刀就能剪下的葡萄（必须是一颗葡萄），就把它剪下来，组成的就是后序遍历了。



后序遍历结果：HIDJEBKFGCA

标题四 二叉树的遍历

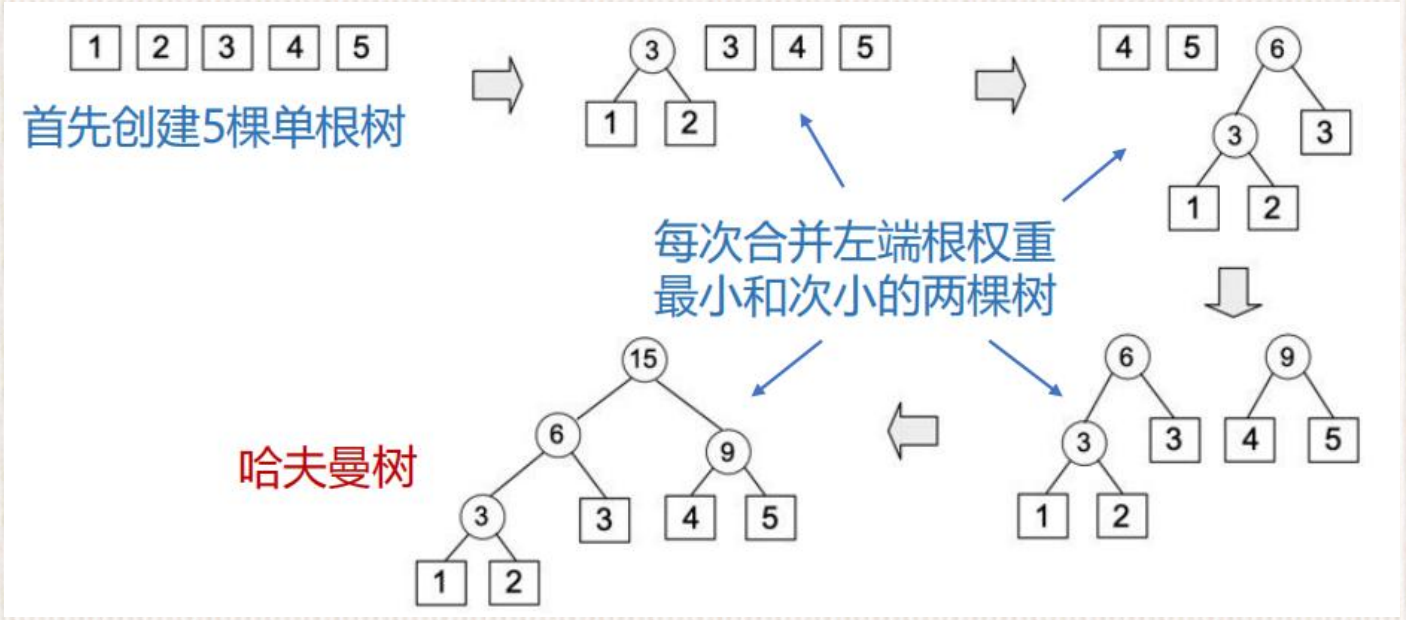
层序遍历按照一层一层的顺序，从左到右写下来就行了。（通常利用队列）

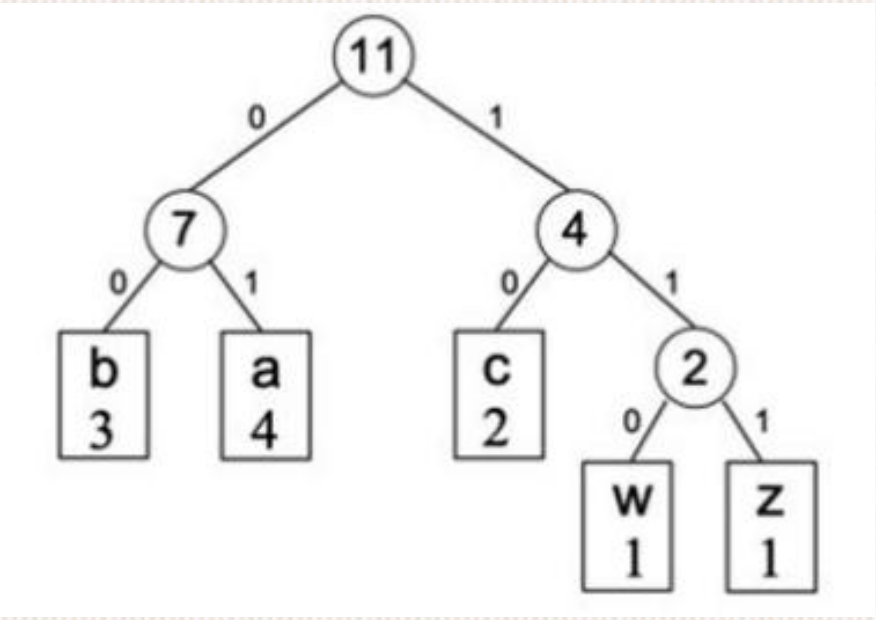


层序遍历结果：ABCDEFGHIJK

标题四 哈夫曼树

哈夫曼算法：一种至下而上构建最优二叉树的方法，通过不断合并两个带权二叉树，最终生成最优二叉树





a–01, b–00, c–10, w–110, z–111
前缀码树的带权路径长度WPL = 2*3 + 2*4 + 2*2 + 3*1 + 3*1 = 24

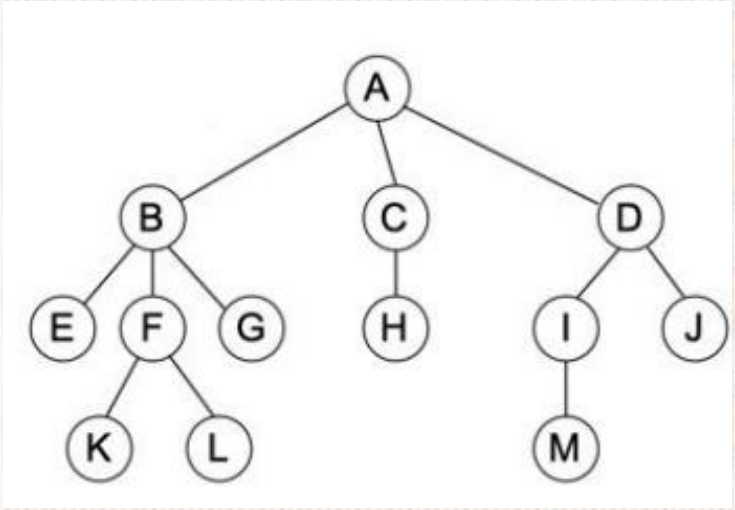
标题四 哈夫曼树

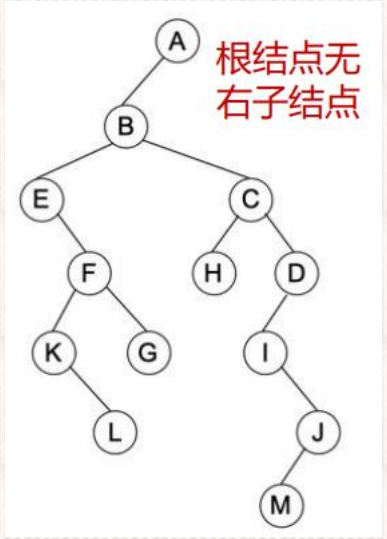
三、（10 分）

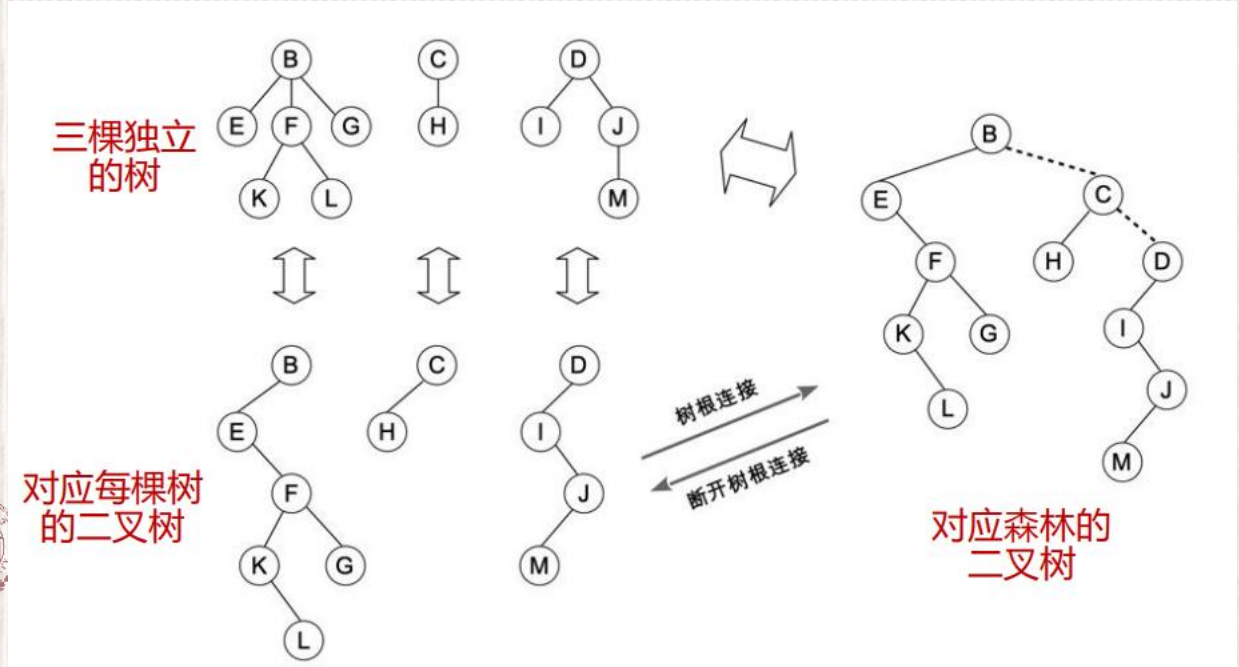
（1）设哈夫曼编码的长度不超过 4，若已经对两个字符编码为 0 和 10，则最多还可以对其它多少个字符编码，请结合画图说明理由。

（2）假设一段报文由字符集{A，B，C，D，E，F}上的字母构成，这 6 个字母在报文中出现的频率分别是 0.12，0.25，0.18，0.15，0.26，0.04，请为这几个字母设计哈夫曼编码，要求画出最终的哈夫曼树并给出每个字符的哈夫曼编码。

标题四 树、森林与二叉树的转换







树的前序遍历与对应的二叉树的前序遍历结果相同
树的后序遍历与对应的二叉树的中序遍历结果相同

森林的前序遍历与对应的二叉树的前序遍历结果相同
森林的后序遍历与对应的二叉树的中序遍历结果相同

湖南大学
HUNAN UNIVERSITY

第五部分

优先队列

标题五 二叉堆

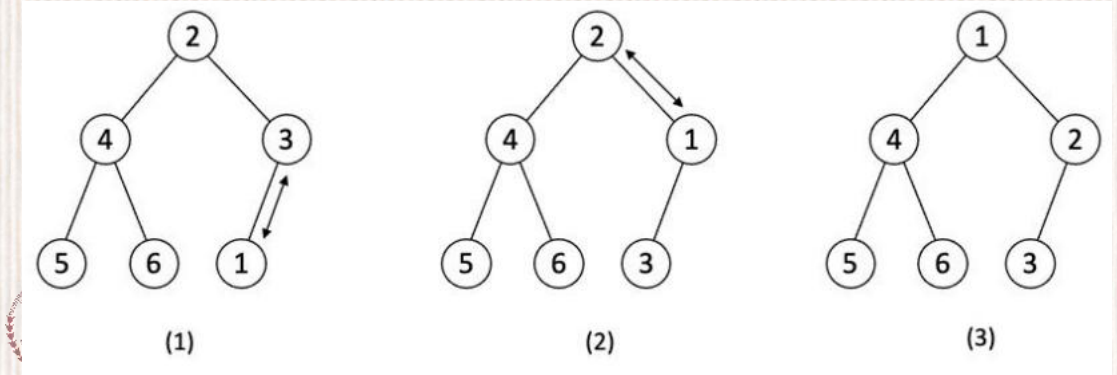
最小堆：如果完全二叉树 T 中的所有父子结点对都有父结点的元素不大于子结点的元素，则称 T 为最小堆。
最大堆：如果完全二叉树 T 中的所有父子结点对都有父结点的元素不小于子结点的元素，则称 T 为最大堆。

二叉堆是一棵**完全二叉树**，可以将其保存在一个数组中

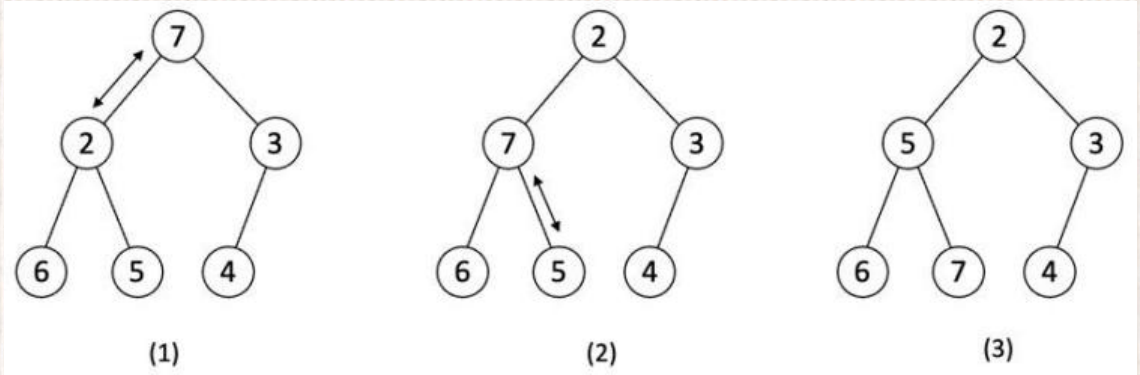
以最小堆为例：
上调：如果堆中某结点 i 小于其父结点 p，可交换结点 i 和结点 p 的元素，即把结点i沿着堆的这棵树往“上”调整。再比较新的父结点与它的大小，重复该过程，直到结点 i 被调到根结点位置或者和新的父结点大小关系满足条件。

下调：如果堆中某结点 i 大于其子结点，则要将其向“下”调整。调整时需先比较i的左右两个子结点的值，再将其中较小者（只有这样才能保证调整之后三者的关系能够满足堆的性质）与结点 i的值进行比较，如果i的值更大，则交换他们的位置。重复该过程，直到结点 i 被调到叶结点位置或者和新的子结点大小关系满足条件。

循环的次数也不会超过树的高度，因此时间复杂度为O(log n)。



上调



下调

标题五 二叉堆

①插入：向堆尾部追加待插入的元素，然后上调插入的元素将其调整到合适的位置来完成堆的调整。

②删除：可删除堆中的第一个元素（堆顶元素），然后把堆中的最后一个元素移到第一个位置，并通过下调新堆顶将这个元素调整到合适的位置

③建堆（朴素）：对于任意一组元素，可以通过逐个上调的方式把它们转化为一个堆，相当于依次插入堆中，堆中后一半的元素进行上调时都可能需要O(log n)的时间，因此总的时间复杂度是O(n log n)

④建堆（快速）：先将元素按照顺序存储为一棵完全二叉树，然后用从最后一个分支结点（大概是一半的位置）到根逐个下调每个分支节点，把它们转化为一个堆。时间复杂度是O(n)

多叉堆、最小最大堆、对顶堆

22

3.（10分）对给定元素序列：13，75，95，22，10，66，77，60，40。

（1）将上述元素依次按照层次遍历顺序存储为一棵完全二叉树，画出该完全二叉树；

（2）请将第（1）小题的完全二叉树调整为最大值堆。

三、（10 分）

(1) 画出对下列存储于数组中的值执行建堆（buildheap）后得到的最大堆。

21 16 23 13 12 11 18 17 20 15

(2) 画出从该最大堆中删除最大元素后得到的堆。

23

第六部分

图

标题六 图的术语

有向边
有向图
无向边
无向图
邻接：图的顶点间有边相连，称顶点间有邻接关系。(vi,vj)是一条无向边，称vi和vj邻接、vj和vi邻接、边(vi,vj)邻接于顶点vi和vj；<vi,vj>是条有向边，称vi邻接到vj 、或vj和vi邻接、边<vi,vj>邻接于顶点vi和vj。
出度：有向图中一个顶点的出度是指由该顶点射出的有向边的条数。
入度：有向图中一个顶点的入度是指射入该顶点的有向边的条数。
度： 无向图中一个顶点的度是指邻接于该顶点的边的总数。
无向完全图：当无向图中边的条数达到最大，为 n （ n - 1 ） / 2 时的图。
有向完全图：当有向图中边的条数达到最大，为 n （ n - 1 ）时的图。
加权有向图
加权无向图
网络：加权有向图和加权无向图，统称为网络。
简单路径：一条路径上除了第一个顶点和最后一个顶点可能相同之外，其余各顶点都不相同。
简单回路或简单环：简单路径上第一个顶点和最后一个顶点相同。

标题六 图的术语

连通图：在一个无向图中，如果任意两个顶点对之间都是连通的，称该无向图

极大连通子图：将该子图外的任意一个顶点增加进子图都会造成子图不连通，且该子图包含了其中顶点间所有的边，该子图称极大连通子图。

连通分量：无向图的极大连通子图称连通分量。

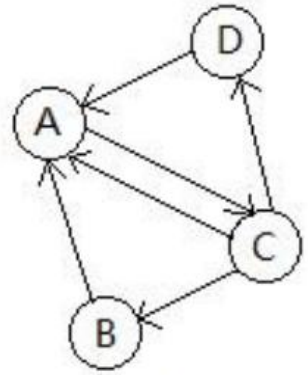
强连通图：在一个有向图G中，如果任意两个顶点对之间都是连通的，称有向图G是强连通图。
强连通分量：有向图的极大连通子图，称强连通分量。

生成树：连通图的极小连通子图，该子图包含连通图的所有n个顶点，但只含它的n-1条边。如果去掉一条边，这个子图将不连通；如果增加一条边，必存在回路。

标题六 图的存储

①邻接矩阵存储法

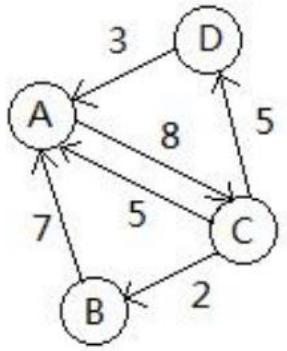
顶点用一维数组存，边用二维矩阵存



0	1	2	3
A	B	C	D

	0	1	2	3
0	0	0	1	0
1	1	0	0	0
2	1	1	0	1
3	1	0	0	0

G8　　G8的邻接矩阵



0	1	2	3
A	B	C	D

	0	1	2	3
0	0	∞	8	∞
1	7	0	∞	∞
2	5	2	0	5
3	3	∞	∞	0

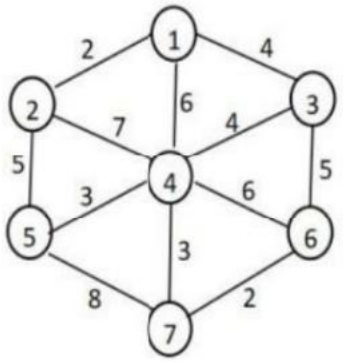
G10　　G10的邻接矩阵

23

四、（10 分）

（1）画出下图的邻接矩阵（带权值）。

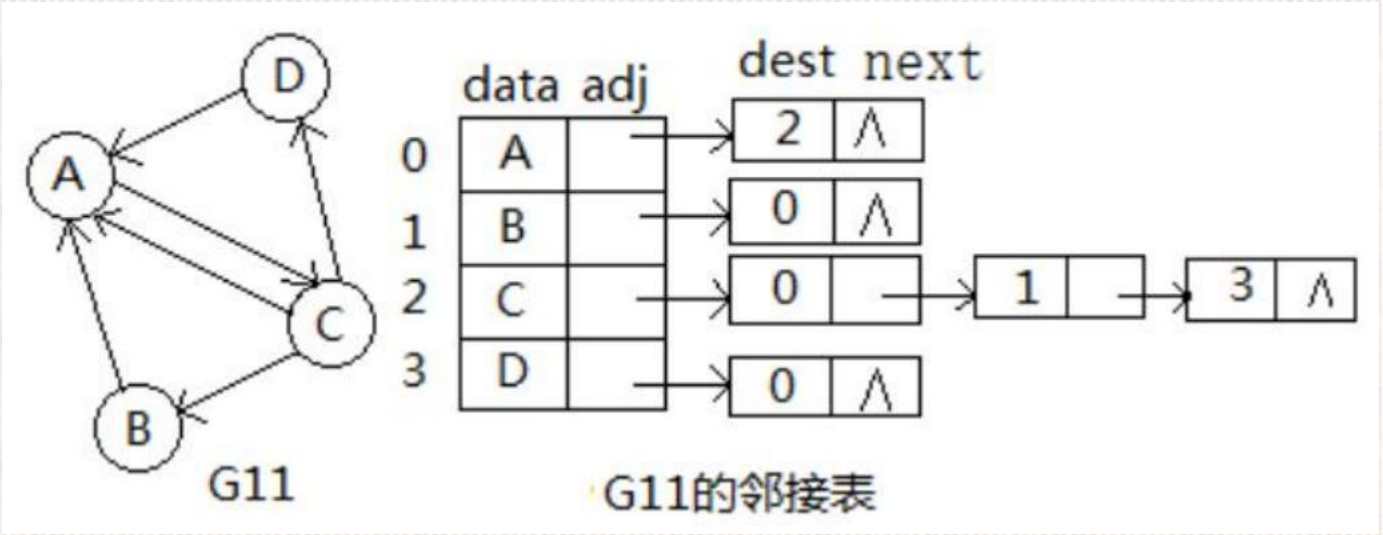
（2）给出从顶点 1 出发，使用 Dijkstra 最短路径算法产生的最短路径长度，要求以二维表格形式写出过程，每处理一个顶点时给出相应的 D 值。



	1	2	3	4	5	6	7
1	0	2	4	6	∞	∞	∞
2	2	0	∞	7	5	∞	∞
3	4	∞	0	4	∞	5	∞
4	6	7	4	0	3	6	3
5	∞	5	∞	3	0	∞	8
6	∞	∞	5	6	∞	0	2
7	∞	∞	∞	3	8	2	0

②邻接表存储法

顶点依然用一个一维数组来存储，而边的存储是将由同一个顶点出发的所有边组成一条单链表。
存储顶点的一维数组称顶点表，存储边信息的单链表称边表。一个图由顶点表和边表共同表示。
顶点表不仅保存各个顶点的信息，还保存由该顶点射出的边形成的单链表中
首结点的地址（首指针）

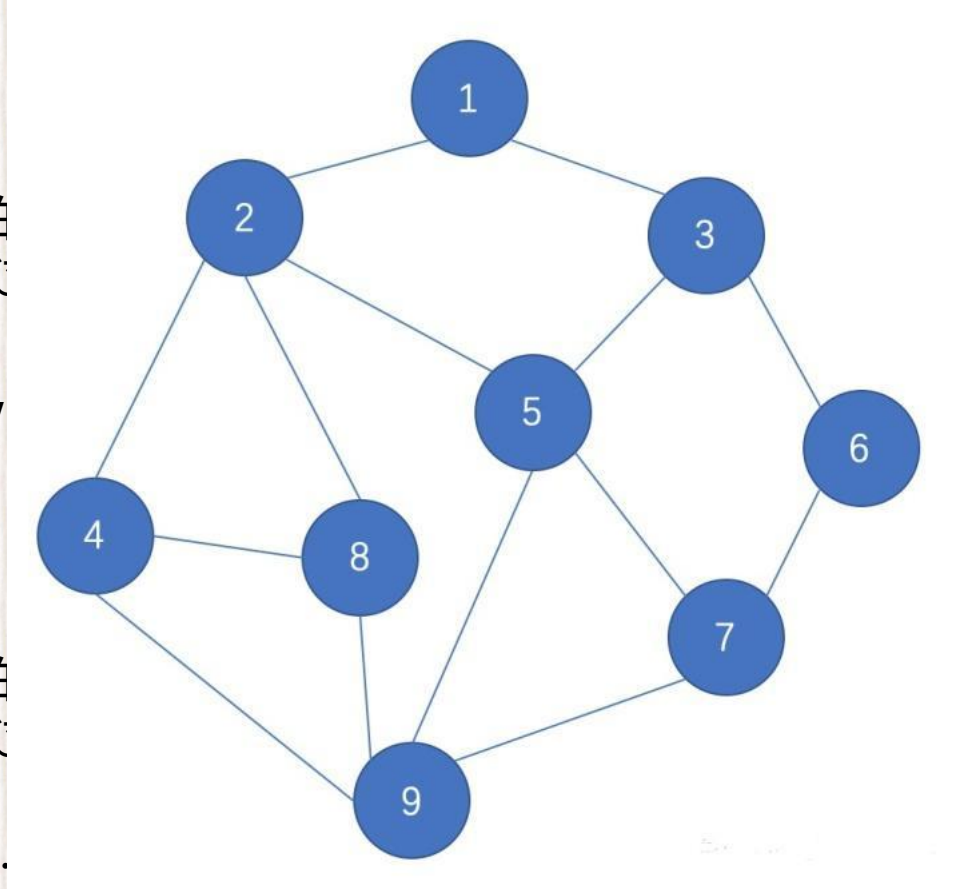


其他形式的邻接表存储：逆邻接表、顶点表为链表、邻接多重表、十字链表

标题六

图的遍历

深度优先遍历（DFS）：
访问方式如下：
1. 从选中的某一个未访问过的
2. 依次从该顶点的未被访问过
转向1。
3. 如果还有顶点未被访问过，
到，遍历结束。



依次进行深度优先遍历，即

。如果所有的顶点都被访问

广度优先遍历（BFS）：
访问方式如下：
1. 从选中的某一个未访问过的
2. 依次对该顶点的未被访问过
且加已访问标志。
3. 依次对顶点 v1 、v2、v3..

1 、v2、v3..... vk进行访问

4. 如果还有顶点未被访问过，选中其中一个顶点作为起始顶点，再次转向1。如果所有的顶点都被访问到，遍历结束。

标题六 图的遍历

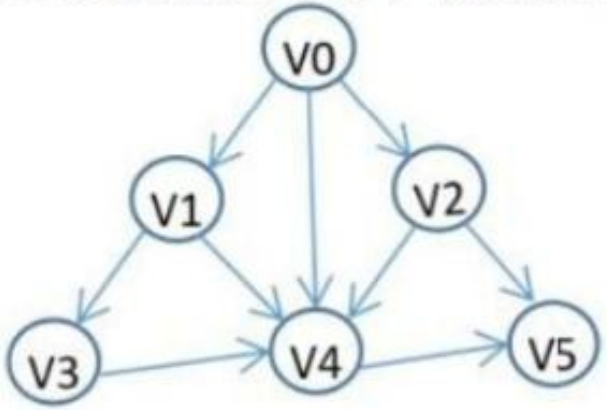
五、（15 分）小明班级在玩传信息游戏，游戏规则如下：共有 n 名同学，每个人的编号从v_0到v_{n-1}，其中小明的编号为v_0；每名同学都有固定的若干个可传递信息的其他玩家（也可能没有），传信息的关系是单向的（比如v_0可以向v_4传信息，但是v_4不能向v_0传信息）。请使用深度优先搜索，对给定的图，判定小明（v_0）是否能将信息传递给全部同学，能则返回 true，不能则返回 false。

要求：

(1) 给出算法的基本设计思想；

(2) 根据设计思想实现算法，并在关键处给出注释；

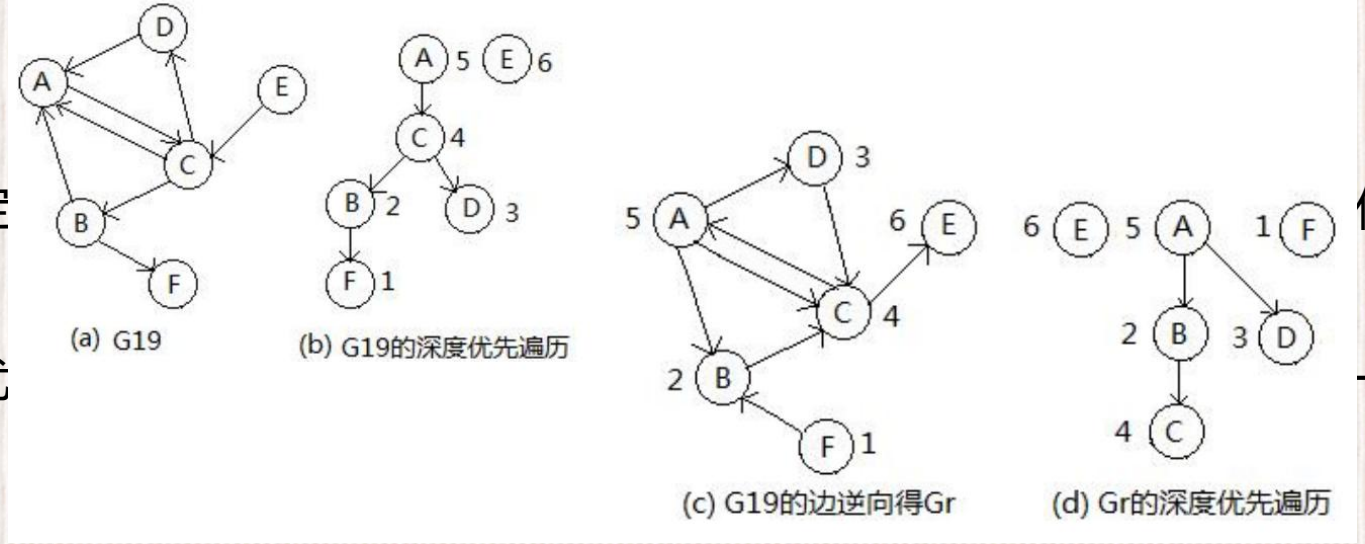
(3) 分析算法的时间复杂度。



标题六

图的连通性

无向图

如果无向图是连通的，那么选定 

他所有顶点。

只需在以上的深度优先、广度优一次，最后比较计数器和顶点树。

有向图

1、对有向图G进行深度优先遍历，按照遍历中回退顶点的次序给每个顶点进行编号。最先回退的顶点的编号为1，其它顶点的编号按回退先后逐次增大1。

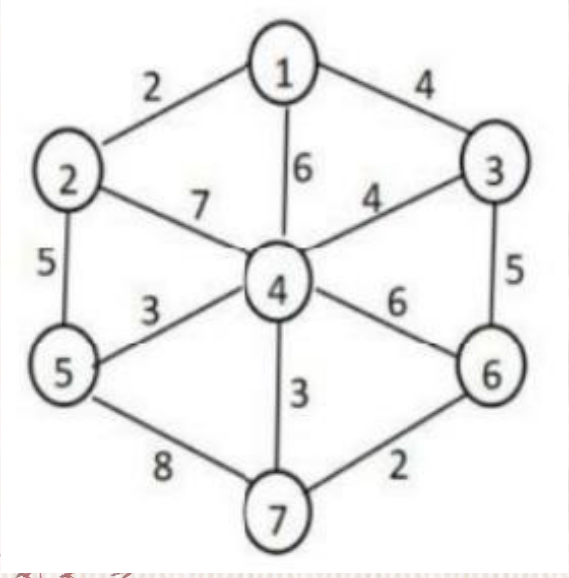
2、将有向图G的所有有向边反向，构造新的有向图Gr。

3、选取未访问顶点中编号最大的顶点，以该顶点为起始点在有向图Gr上进行深度优先遍历。如果没有访问到所有的顶点，再次返回3，反复如此，直至所有的顶点都被访问到。

标题六 Dijkstra算法

Dijkstra算法是从一个固定的源顶点出发寻找到图中所有其它顶点的最短路径，产生一个最短路径树。该算法每次取出未访问顶点中距离最小的，再用该顶点更新其他顶点的距离。（边权值为非负）

给出从顶点1出发，使用Dijkstra最短路径算法产生的最短路径长度，要求以二维表格形式写出过程，每处理一个顶点时给出相应的D值



设图中顶点数为|V|，则时间复杂度为$|V|^2$

从1到其余顶点的dist值和最短路径的求解过程

终点	i=1	i=2	i=3	i=4	i=5	i=6
2	2 <1,2>					
3	4 <1,3>					
4	6 <1,4>					
5		7 <1,2,5>				
6					17 <1,2,5,7,6>	
7				15 <1,2,5,7>		
u	2	3	4	5	7	6
S	{1,2}	{1,2,3}	{1,2,3,4}	{1,2,3,4,5}	{1,2,3,4,5,7}	{1,2,3,4,5,6,7}

dist[2]=2 dist[3]=4 dist[4]=6 dist[5]=7 dist[6]=17 dist[7]=15

标题六 图的遍历

九、（15 分）湘江游船俱乐部在湘江上设置了 n 个游船出租站 1，2，…，n。游客可在这些游船出租站租用游船，并在下游的任何一个游船出租站归还游船。游船出租站 i 到游船出租站 j 之间的租金为 r(i,j),1<=i<j<=n。请一个算法，计算出从游船出租站 1 到游船出租站 n 所需的最少租金。

输入样例及说明：

4　　（表示有 n 个游船出租站，且 n=4，接下来的 n-1 行数据表示租金 r(i,j)）

5 15 25　　（表示第 1 站到 2,3,4 站的租金，且 r(1,2)=5，r(1,3)=15，r(1,4)=25）

7 14　　（表示第 2 站到 3,4 站的租金，r(2,3)=7，r(2,4)=14）

8　　（表示第 3 站到 4 站的租金，r(3,4)=8）

输出样例及说明：

19　　（输出从游船出租站 1 到游船出租站 n 所需的最少租金）

要求：

(1) 给出算法的基本设计思想；

(2) 根据设计思想实现算法，并在关键处给出注释；

(3) 分析算法的时间复杂度。

标题六 图的遍历

21

六、（15 分）给定 n 个村庄之间的交通图（注：该图为无向图），其中村庄被抽象为图中的顶点，村庄 i 与村庄 j 之间的道路被抽象为图中的边，该边上的权值 w_{ij} 表示这条道路的长度。现打算在这 n 个村庄中选定一个村庄建一所小学。该小学应该建立在最中心的村庄里，这样使得距离小学最远的村庄到小学的路径长度 D 最短，(即如果选择小学建立在另外一个非最中心的村庄，则距离小学最远的村庄到小学的路径长度 D’ 将大于 D)。请设计一个算法求出该小学应建在哪个村庄才能使距离小学最远的村庄到小学的路径长度最短。要求：

（1）给出算法的基本设计思想；

（2）根据设计思想，采用伪码描述算法，关键之处给出注释；

（3）分析算法的时间复杂度。

标题六 带负权的最短路径算法

Bellman-Ford算法（单源最短路径边权值可以为负）

算法 8-2： 求单源最短路的 Bellman-Ford 算法 Bellman-Ford (*graph, s, dist*)

输入： 加权有向图 *graph*、起点 *s*

输出： 不存在权值为负值的环路时，返回从 *s* 到其余顶点 *v* 的最短路径长度数组 *dist*[*v*]，并且返回 **true**；否则返回 **false**

```
1  n ← graph.n_verts
2  for v←0 to n-1 do
3  |  dist[v] ← ∞    //初始化源顶点到各个顶点距离为无穷大
4  end
5  dist[s] ← 0;  // 从源顶点开始
6  for i←1 to n-1 do
7  |  for u←0 to n-1 do    //遍历所有的边
8  |  |  for v←0 to n-1 do
9  |  |  |  if graph.edge_matrix[u][v]≠graph.no_edge_value then
10 |  |  |  |  if dist[v]>dist[u]+graph.edge_matrix[u][v] then
11 |  |  |  |  |  dist[v] ← dist[u] + graph.edge_matrix[u][v] //对边松弛操作,更新到 v 的距离
12 |  |  |  |  end
13 |  |  |  end
14 |  |  end
15 |  end
16 end
```

算法 8-3： 求所有点对间最短路的 Floyd-Warshall 算法 Floyd-Warshall (*graph, path, dist*)

输入： 加权有向图 *graph*

输出： 图 *graph* 中任意两顶点 v_i 到 v_j 的最短路径途径顶点 *path*[*i*][*j*]及加权长度 *dist*[*i*][*j*]

```
1  n ← graph.n_verts
2  for i←0 to n-1 do   //初始化各对顶点之间的已知路径和距离
3  |  for j←0 to n-1 do
4  |  |  dist[i][j] ← graph.edge_matrix[i][j]
5  |  |  path[i][j] ← NIL
6  |  end
7  end
8  for k←0 to n-1 do
9  |  for i←0 to n-1 do
10 |  |  for j←0 to n-1 do
11 |  |  |  if dist[i][k] + dist[k][j] < dist[i][j] then // 从 vi 经过 vk 到 vj 的一条路径更短
12 |  |  |  |  dist[i][j] ← dist[i][k] + dist[k][j]
13 |  |  |  |  path[i][j] ← k
14 |  |  |  end
15 |  |  end
16 |  end
17 end
```

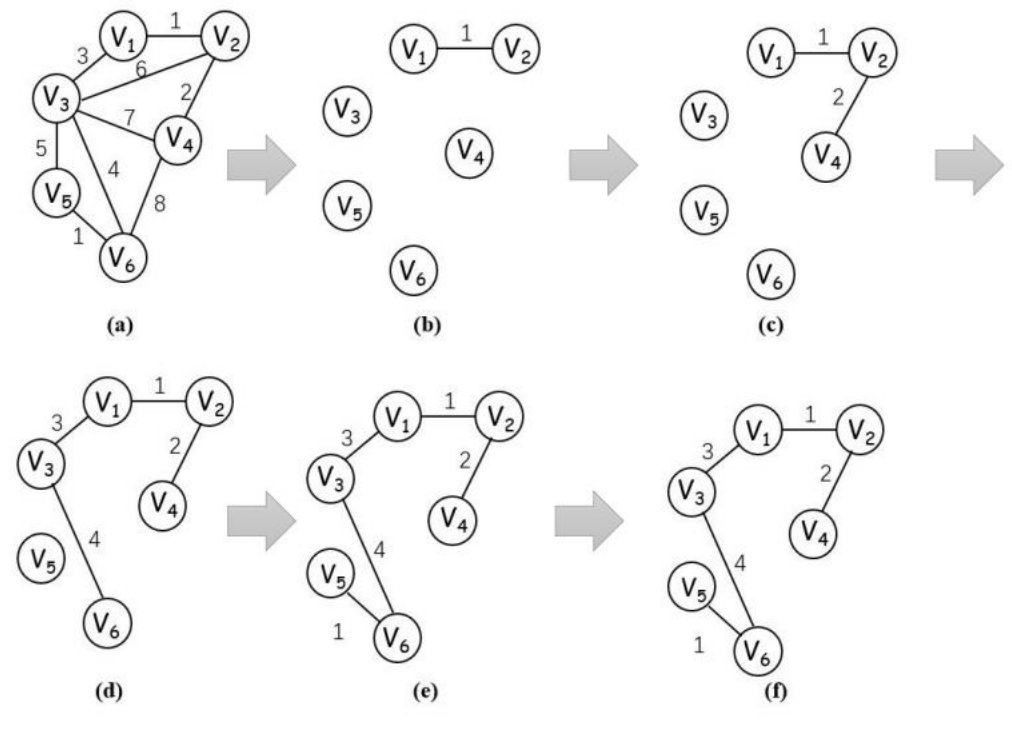
标题六

最小生成树

生成树：连通图的**极小连通子图**，该子图包含连通图的所有n个顶点，但只 含它的n-1条边。如果去掉一条边，这个子图将不连通；如果增加一条边， 必存在回路。

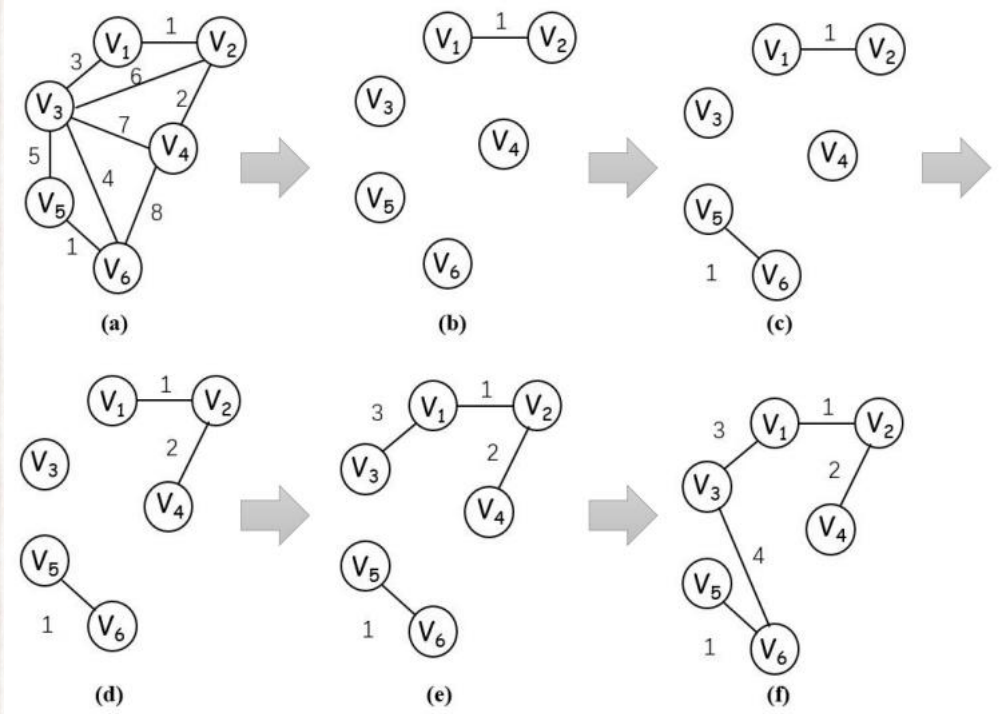
最小生成树：求解权值最小的生成树。

Prim算法：每次选择一个点



采用邻接矩阵存储，Prim算法时间复杂度为$|V|^2$
(适合稠密图)

Kruskal算法：每次选择一条边

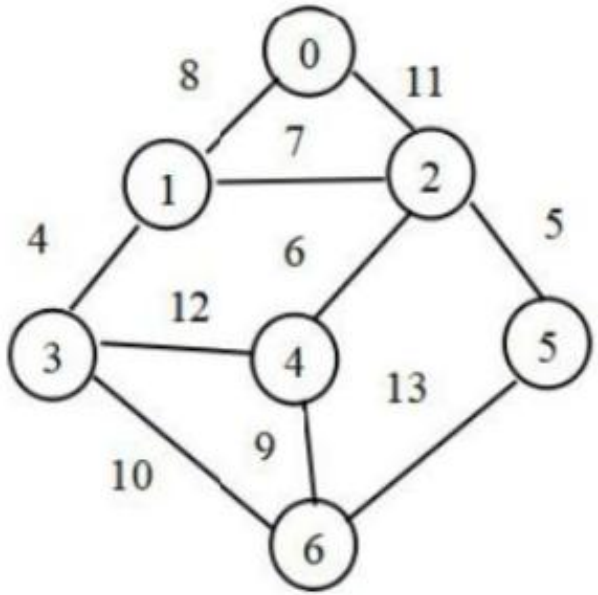


用堆来存储边，时间复杂度为|E|log|E|
(适合稀疏图)

标题六 最小生成树

21

四、（10 分）已知几个城市之间的交通代价图——带权无向图 G，如下所示。



（1）写出从顶点 0 出发调用 BFS 的广度优先搜索树；

（2）画出用 Kruskal 算法构造最小生成树将这几个城市连通的过程。

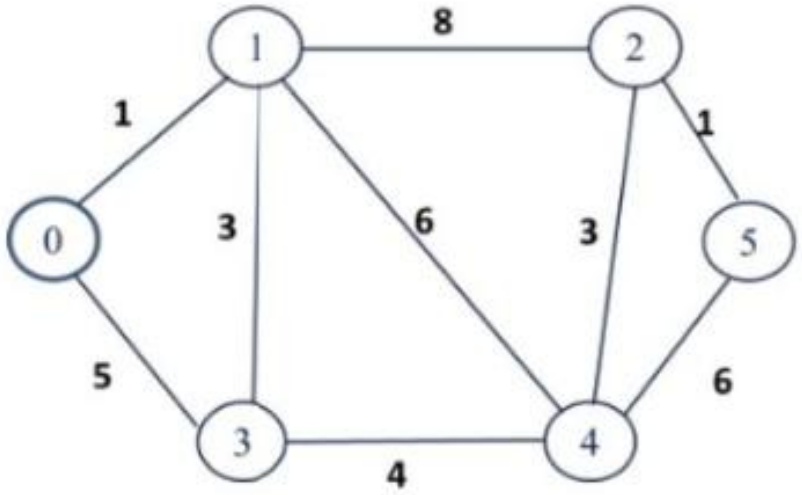
标题六 最小生成树

22

5.（10分）对于给定带权无向图，如下图所示：

（1）采用Dijkstra算法求从点0到到其它各个顶点的最短路径长度，给出求解过程。

（2）采用prim算法求出从顶点0出发的最小生成树，画出最终的最小生成树（不需要求解过程）。

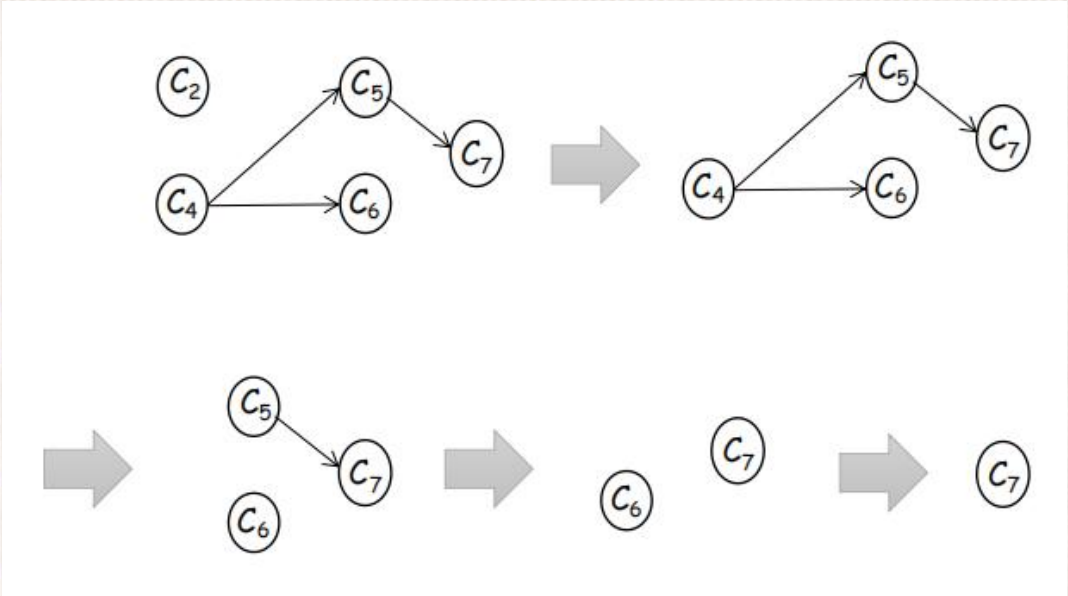


标题六 拓扑排序

拓扑排序过程：

① 初始化空序列S；
② 任选一个入度为0的顶点v，加入S；
③ 删除顶点v及其所有出边；
④ 重复步骤②和③，直到无顶点可选。

时间复杂度：O(|V| + |E|)（每个点和边都会被遍历一次然后被删除）



标题六 拓扑排序

6. 某工厂的产品生产线上，产品A由若干个零部件组成。零部件生产时，它们相互之间的先后关系存在如下两种可能的值：

◆ 零部件存在先后关系，即一个部件必须在完成后才能生产另一个部件；

◆ 零部件间无关系，即这两个部件可以同时生产，互不影响。

现在要生产一批产品A零件，假定需要的零部件和零部件之间存在关系如下表所示(例如，C1无先期要生产的部件，可以直接生产；C4和C2,C3之间存在先后关系，在生产C4之前，必须先生产C2,C3):

零部件代号	C1	C2	C3	C4	C5	C6	C7
零部件生产的先期部件	无	无	C2	C2,C3	C3,C6	C7	C1

运用所学的数据结构知识，请根据数据之间特征抽象出数据之间的逻辑结构。基于下面图的ADT，设计一个算法，输出一个能顺利生产出产品A的所有零件的产品生产序列。

(1)给出算法的基本设计思想。

(2)根据设计思想，采用伪代码算法，关键之处给出注释。

(3)分析算法的时间复杂度。

湖南大學
HUNAN UNIVERSITY

谢 谢 观 看

thank you for watching