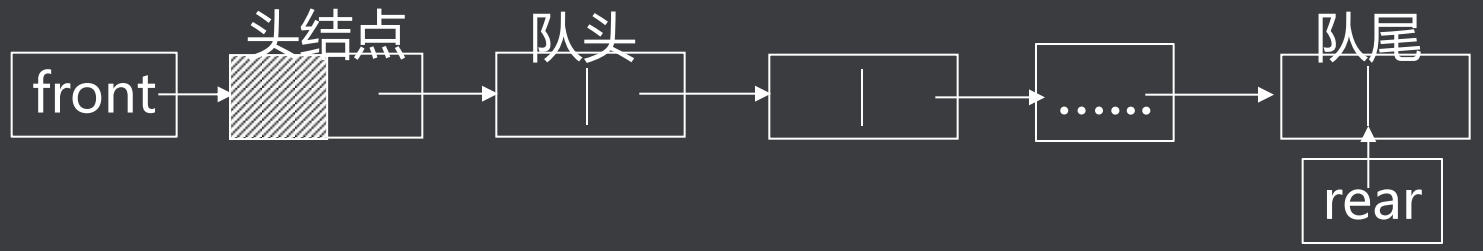
链式队列

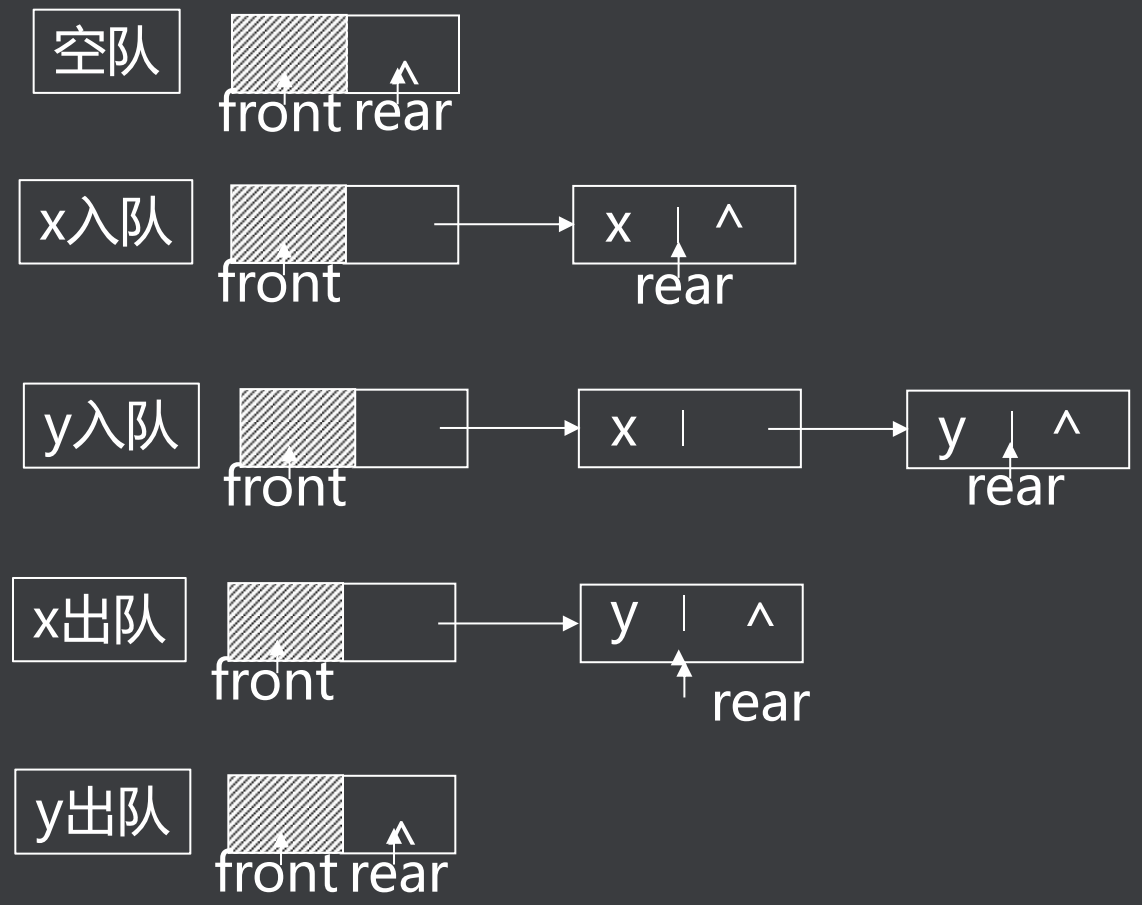
链式队列：

插入操作在队尾进行，删除操作在队头进行，由队头指针和队尾指针控制队列的操作。

```
typedef  int  data_t;     /*定义链队列中数据元素的数据类型*/
typedef  struct  node_t
{   data_t data ;                    /*数据域*/
          struct node_t   *next;          /*链接指针域*/
} linknode_t,  *linklist_t;              /*链表元素类型定义*/
typedef  struct
{  linklist_t  front,  rear;         /*链队列指针*/
} linkqueue_t;            /*链队列类型定义*/
```



队列


空队
front
rear
x入队
front
x
^
rear
y入队
front
x
y
^
rear
x出队
front
y
^
rear
y出队
front
rear

队列

创建空队列：

```
linkqueue_t  *CreateQueue()
{
      linkqueue_t  *lq  =  (linkqueue_t  *)malloc(sizeof(linkqueue_t));
      lq->front = lq->rear = (linklist_t)malloc(sizeof(linknode_t));
      lq->front->next = NULL ;          /*置空队*/
      return  lq;     /*返回队列指针*/
}
```

判断队列空：

```
int   EmptyQueue(linkqueue_t  *lq)  {
      return ( lq->front  =  =  lq->rear) ;
}
```

队列

入队：

```
void  EnQueue (linkqueue_t *lq, data_t x)
{
      lq->rear->next = (linklist_t)malloc(sizeof(linknode_t)) ;
      lq->rear = lq->rear->next;                    /*修改队尾指针*/
      lq->rear->data = x ;                  /*新数据存入新节点*/
      lq->rear->next = NULL ;                       /*新节点为队尾*/
      return;
}
```

出队

出队：

```
data_t  DeQueue(linkqueue_t  *lq)
{
        data_t   x;
        linklist_t   p;              /*定义一个指向队头结点的辅助指针*/
        p = lq->front->next ;    /*将它指向队头结点*/
        lq->front->next  =  p->next ;     /*删除原先队头结点
        x = p->data;
        free(p) ;              /*释放原队头结点*/
        if  (lq->front->next  ==  NULL)  lq->rear  =  lq->front;
       return  x;
}
```

3.3

栈和队列
应用举例

球钟问题

问题描述：球钟是一个利用球的移动来记录时间的简单装置。它有三个可以容纳若干个球的指示器：分钟指示器，五分钟指示器，小时指示器。若分钟指示器中有2个球，五分钟指示器中有6个球，小时指示器中有5个球，则时间为5:32

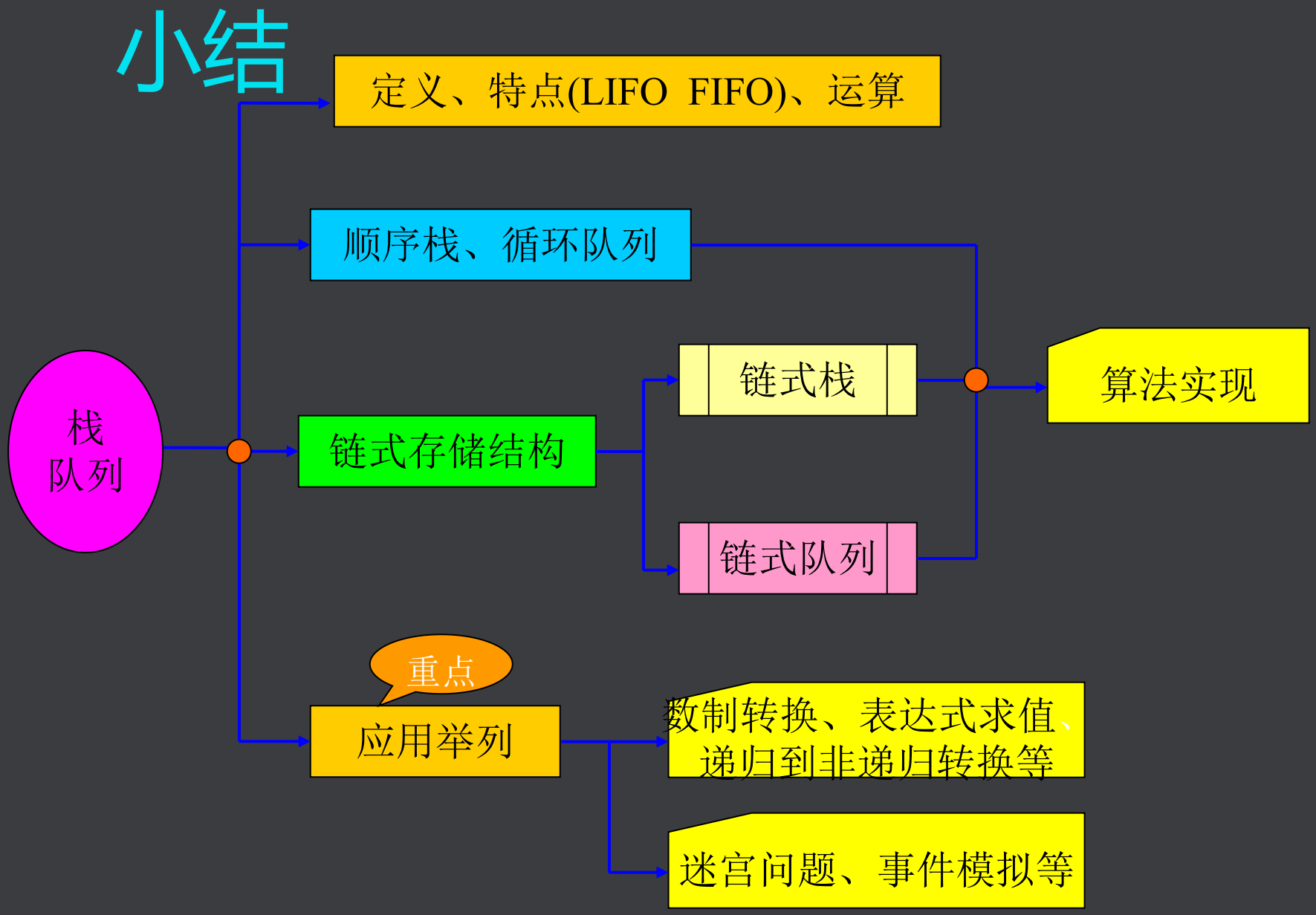
工作原理：每过一分钟，球钟就会从球队列的队首取出一个球放入分钟指示器，分钟指示器最多可容纳4个球。当放入第五个球时，在分钟指示器的4个球就会按照他们被放入时的相反顺序加入球队列的队尾。而第五个球就会进入五分钟指示器。按此类推，五分钟指示器最多可放11个球，小时指示器最多可放11个球。

球钟问题

当小时指示器放入第12个球时，原来的11个球按照他们被放入时的相反顺序加入球队列的队尾，然后第12个球也回到队尾。这时，三个指示器均为空，回到初始状态，从而形成一个循环。因此，该球钟表示时间的范围是从0：00到11：59。

现设初始时球队列的球数为27，球钟的三个指示器初态均为空。问，要经过多久，球队列才能回复到原来的顺序?

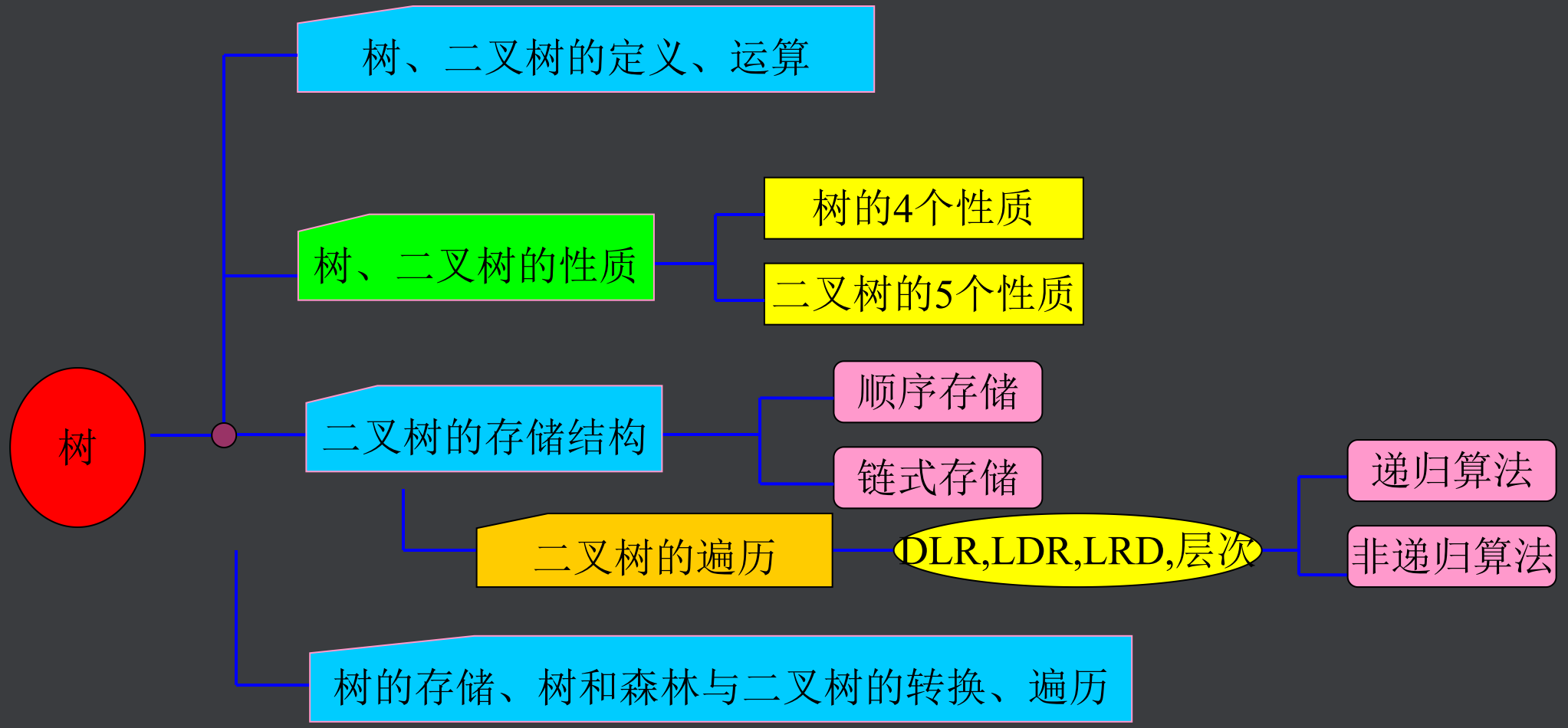
小结


定义、特点(LIFO FIFO)、运算
顺序栈、循环队列
栈
队列
链式存储结构
链式栈
链式队列
算法实现
重点
应用举列
数制转换、表达式求值、
递归到非递归转换等
迷宫问题、事件模拟等

4

树与二叉树

知识点


树
树、二叉树的定义、运算
树、二叉树的性质
树的4个性质
二叉树的5个性质
二叉树的存储结构
顺序存储
链式存储
二叉树的遍历
DLR,LDR,LRD,层次
递归算法
非递归算法
树的存储、树和森林与二叉树的转换、遍历

4.1

树的概念

树

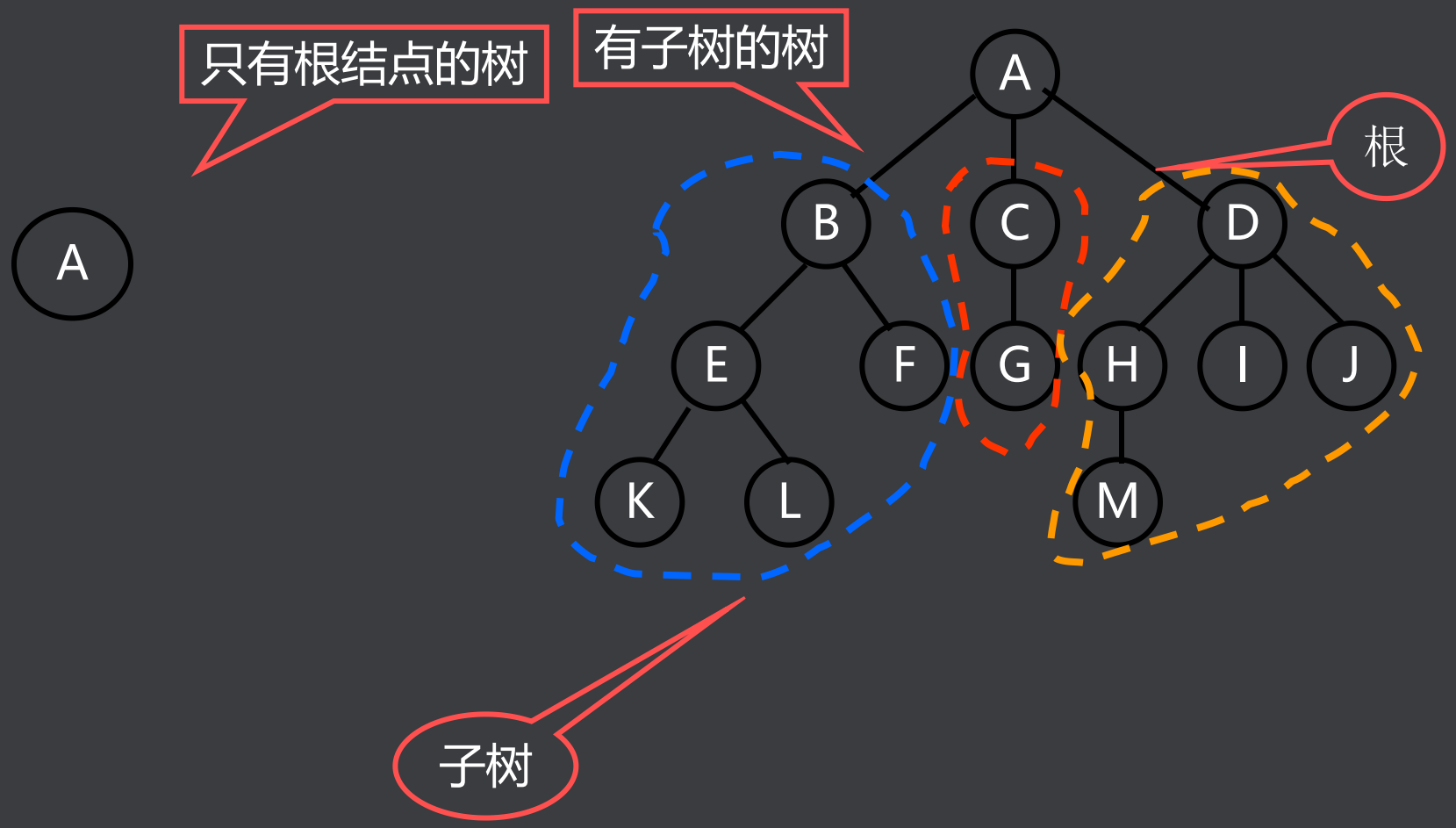
树的概念

树（Tree）是n（n≥0）个节点的有限集合T，它满足两个条件：

- 有且仅有一个特定的称为根（Root）的节点；
- 其余的节点可以分为m（m≥0）个互不相交的有限集合T1、T2、……、Tm，其中每一个集合又是一棵树，并称为其根的子树（Subtree）。

表示方法：**树形表示法、目录表示法。**

树



树

基本概念：

一个节点的子树的个数称为该节点的**度数**，一棵树的度数是指该树中节点的最大度数。

度数为零的节点称为**树叶**或**终端节点**，度数不为零的节点称为分支节点，除根节点外的分支节点称为内部节点。

一个节点的子树之根节点称为该节点的**子节点**，该节点称为它们的**父节点**，同一节点的各个子节点之间称为兄弟节点。一棵树的根节点没有父节点，叶节点没有子节点。

树

一个节点系列$k_1,k_2,\ldots\ldots,k_i,k_{i+1},\ldots\ldots,k_j$,并满足$k_i$是$k_{i+1}$的父节点，就称为一条从$k_1$到$k_j$的**路径**，路径的长度为j-1,即路径中的**边数**。路径中前面的节点是后面节点的祖先，后面节点是前面节点的子孙。节点的层数等于父节点的层数加一，根节点的层数定义为一。树中节点层数的最大值称为该树的**高度**或**深度**。

若树中每个节点的各个子树的排列为从左到右，不能交换，即兄弟之间是有序的，则该树称为**有序树**。一般的树是有序树。

m（m≥0）棵互不相交的树的集合称为**森林**。树去掉根节点就成为森林，森林加上一个新的根节点就成为树。

树

结点A的度：3
结点B的度：2
结点M的度：0

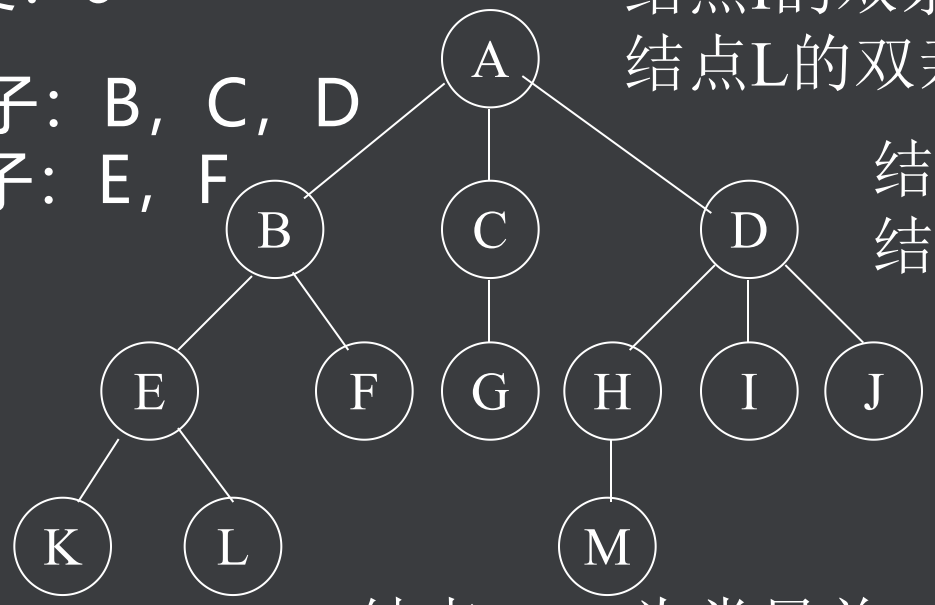
叶子：K，L，F，G，M，I，J

结点I的双亲：D
结点L的双亲：E

结点A的孩子：B，C，D
结点B的孩子：E，F

结点B，C，D为兄弟
结点K，L为兄弟

树的度：3



树的深度：4

结点F，G为堂兄弟
结点A是结点F，G的祖先

结点A的层次：1
结点M的层次：4

树

树的逻辑结构： 树中任何节点都可以有零个或多个直接后继节点（子节点），但至多只有一个直接前趋节点（父节点），根节点没有前趋节点，叶节点没有后继节点。

4.2

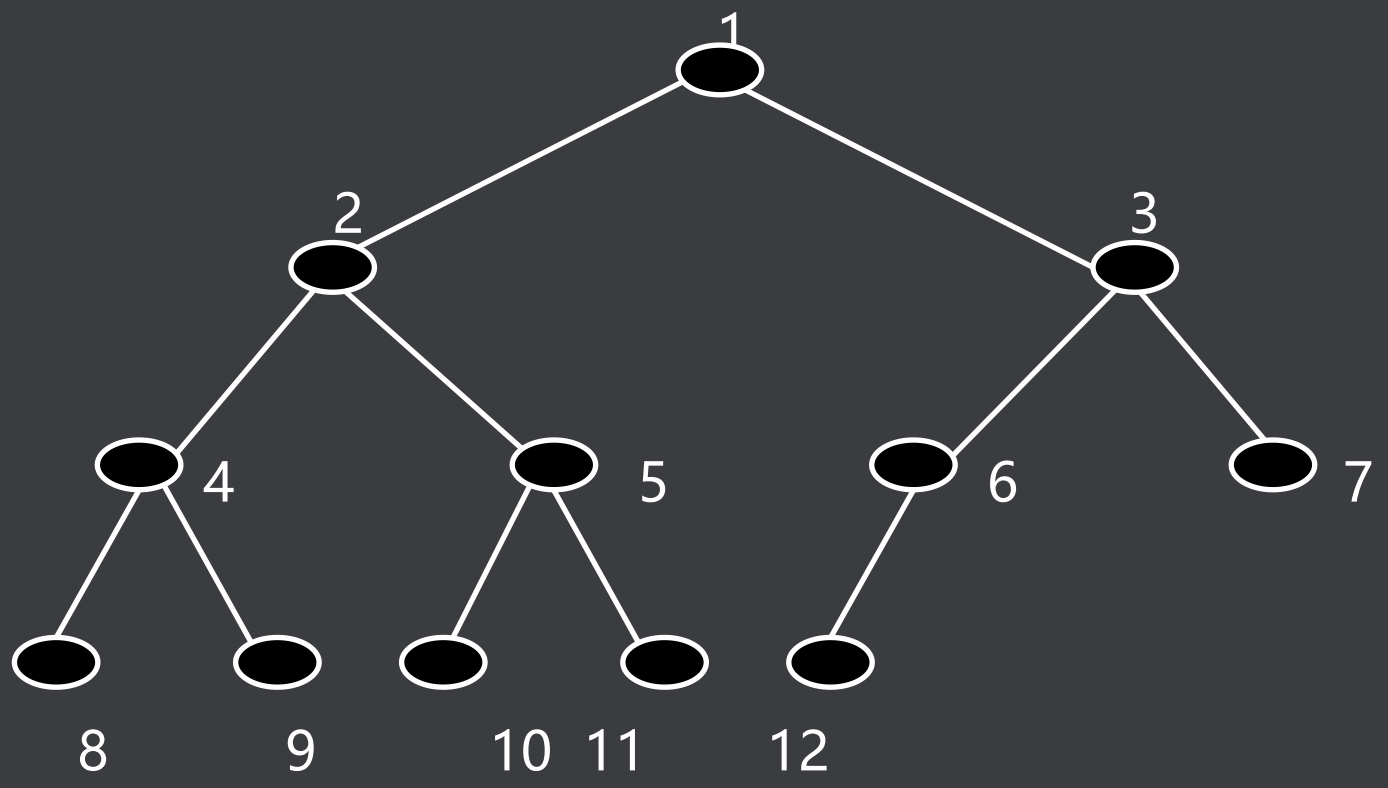
二叉树

二叉树

二叉树

二叉树的定义： **二叉树（Binary Tree）**是n（n≥0）个节点的有限集合，它或者是空集（n = 0），或者是由一个根节点以及两棵互不相交的、分别称为左子树和右子树的二叉树组成。二叉树与普通有序树不同，二叉树严格区分左孩子和右孩子，即使只有一个子节点也要区分左右。

二叉树



二叉树

二叉树的性质：

二叉树第i（i≥1）层上的节点最多为2^{i-1}个。

深度为k（k≥1）的二叉树最多有$2^k - 1$个节点。

在任意一棵二叉树中，树叶的数目比度数为2的节点的数目多一。

总节点数为各类节点之和：$n = n_0 + n_1 + n_2$

总节点数为所有子节点数加一：$n = n_1 + 2*n_2 + 1$

故得：$n_0 = n_2 + 1$；

满二叉树：深度为k（k≥1）时有$2^k - 1$个节点的二叉树。

完全二叉树：只有最下面两层有度数小于2的节点，且最下面一层的叶节点集中在最左边的若干位置上。

具有n个节点的完全二叉树的深度为

（log2n）+1或『log2(n+1)。

二叉树

二叉树的存储：

顺序存储结构：完全二叉树节点的编号方法是从上到下，从左到右，根节点为1号节点。设完全二叉树的节点数为n，某节点编号为i

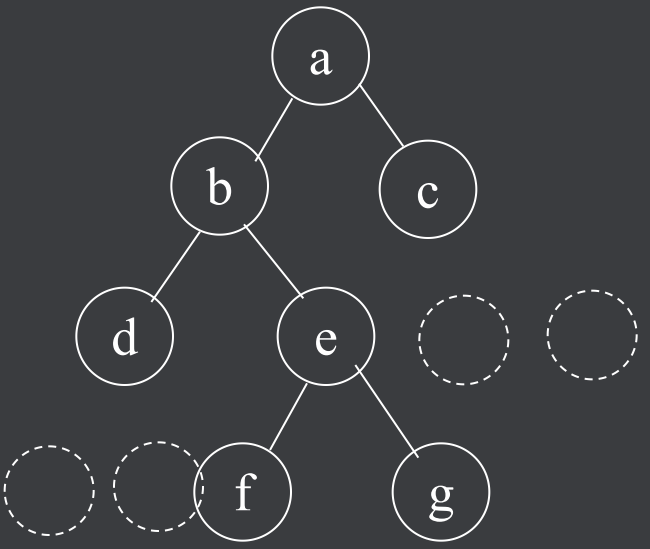
- 当i＞1（不是根节点）时，有父节点，其编号为i/2;
- 当2*i≤n时，有左孩子，其编号为2*i ,否则没有左孩子，本身是叶节点;
- 当2*i＋1≤n时，有右孩子，其编号为2*i+1 ,否则没有右孩子；
- 当i为奇数且不为1时，有左兄弟，其编号为i-1,否则没有左兄弟；
- 当i为偶数且小于n时，有右兄弟，其编号为i＋1,否则没有右兄弟；

二叉树

有n个节点的完全二叉树可以用有n+1个元素的数组进行顺序存储，节点号和数组下标一一对应，下标为零的元素不用。

利用以上特性，可以从下标获得节点的逻辑关系。不完全二叉树通过添加虚节点构成完全二叉树，然后用数组存储，这要浪费一些存储空间。

二叉树



1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	0	0	0	0	f	g

浪费空间

二叉树

链式存储结构：

```
typedef  int  data_t ;                    /*定义数据类型*/
typedef  struct  node_t;                  /*定义二叉树节点的内部结构*/
{
    data_t data ;                 /*数据域*/
    struct node_t *lchild ,*rchild ;          /*指向左孩子和右孩子的指针*/
} bitree_t ;                              /*二叉树节点类型*/
bitree_t *root ;                  /*定义指向二叉树的指针*/
```

二叉树由根节点指针决定。

二叉树