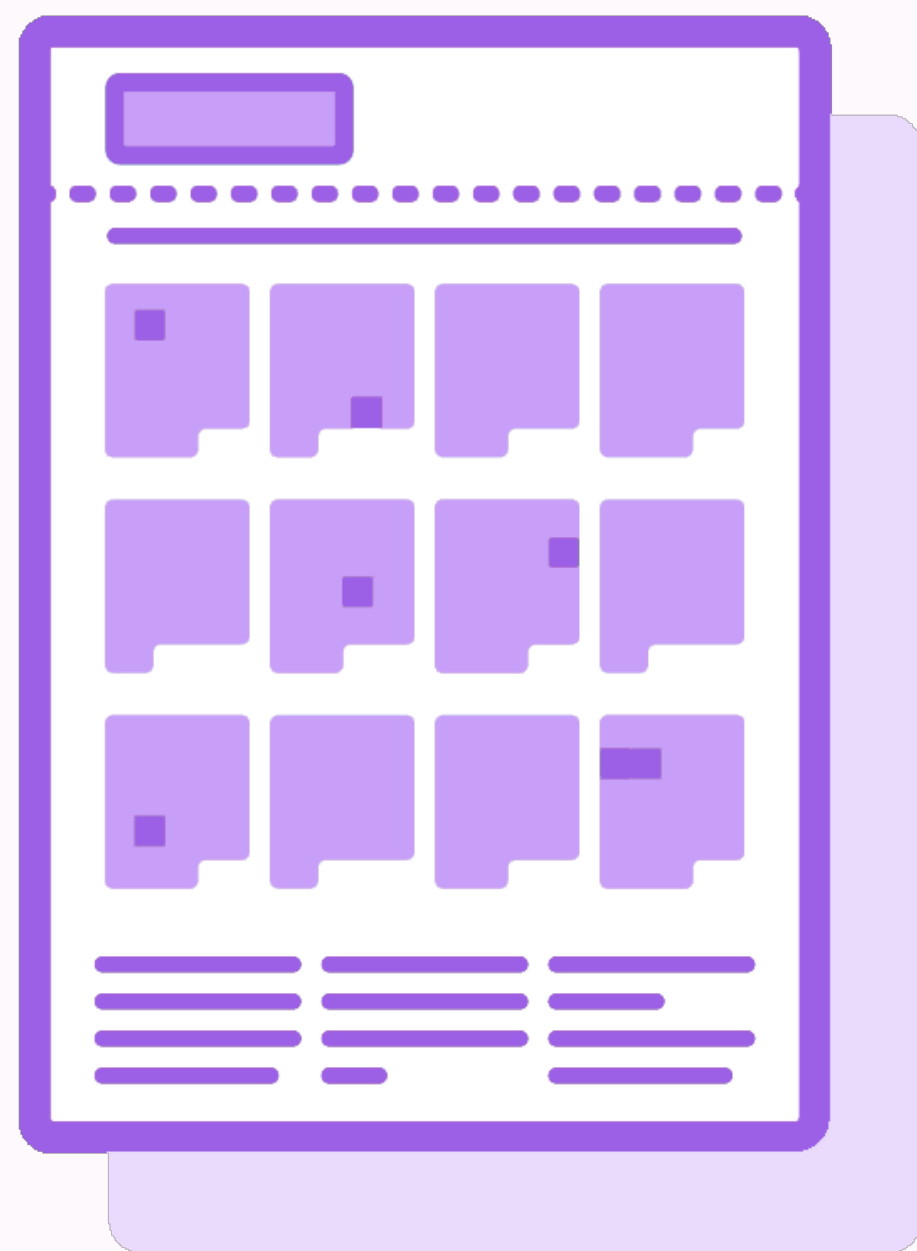


树与二叉树

斐多课堂  数据结构  第五讲
Phaedo Classes



3大模块



9道题目



树与二叉树

模块1 / 树的基本概念

模块2 / 二叉树

模块3 / 树、森林

树的基本概念

小节1 / 定义及特点

小节2 / 基本术语

小节3 / 性质

树的基本概念

小节1 / 定义及特点

小节2 / 基本术语

小节3 / 性质

树的定义

树是 $N(N \geq 0)$ 个结点的有限集合。

$N=0$ 时，称为空树。

树的定义

任意一棵非空树中应满足：

(1) 有且仅有一个特定的称为根^根的结点。

(2) 当 $N > 1$ 时，其余结点可分为 $m(m > 0)$ 个互不相交的有限集合 T_1, T_2, \dots, T_m ,

其中每个集合本身又是一棵树，称为根结点的子^{子树}树。

树的特点

树是一种递归的数据结构：

(1) 除根结点外的所有结点有且仅有一个前驱结点。

(2) 树中的所有结点可以有零个或多个后继结点。

树的基本概念

小节1 / 定义及特点

小节2 / 基本术语

小节3 / 性质

基本术语

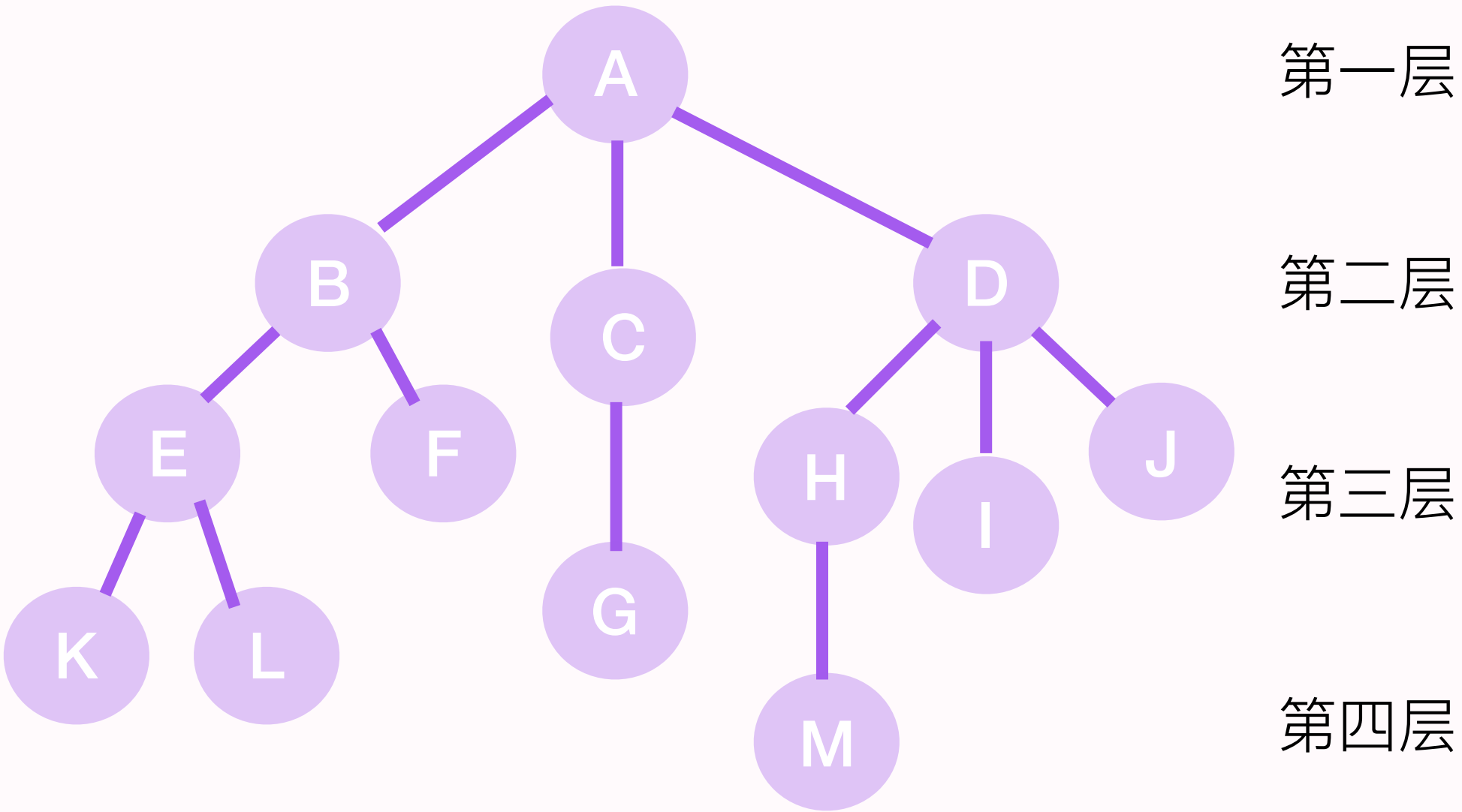
祖先结点与子孙结点：根A到结点K的唯一路径上的任意结点，称为结点K的祖先结点。结点B是结点K的祖先结点，结点K是结点B的子孙结点。

双亲结点与孩子结点：路径上最接近结点K的结点E称为K的双亲结点，K为结点E的孩子结点。

根A是树中唯一没有双亲的结点。

兄弟结点：有相同双亲的结点称为兄弟结点，如结点K和结点L。

高度为4



树的树形表示

基本术语

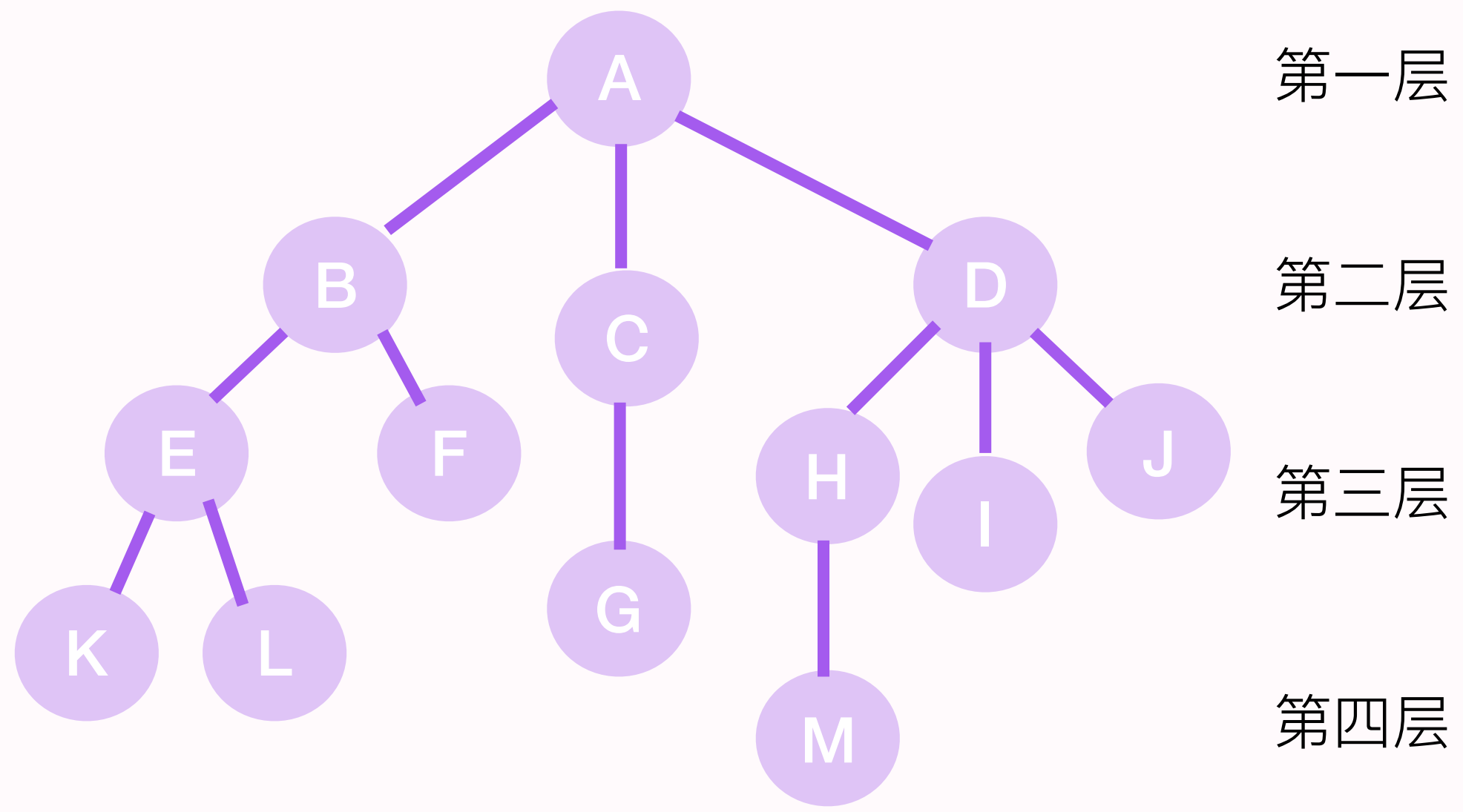
结点的度： 树中的一个结点的自结点个数称为该结点的度。

树的度： 树中结点的最大度称为树的度。

分支结点： 度大于0的结点称为分支结点。

叶子结点： 度为0的结点称为叶子结点。

高度为4



树的树形表示

基本术语

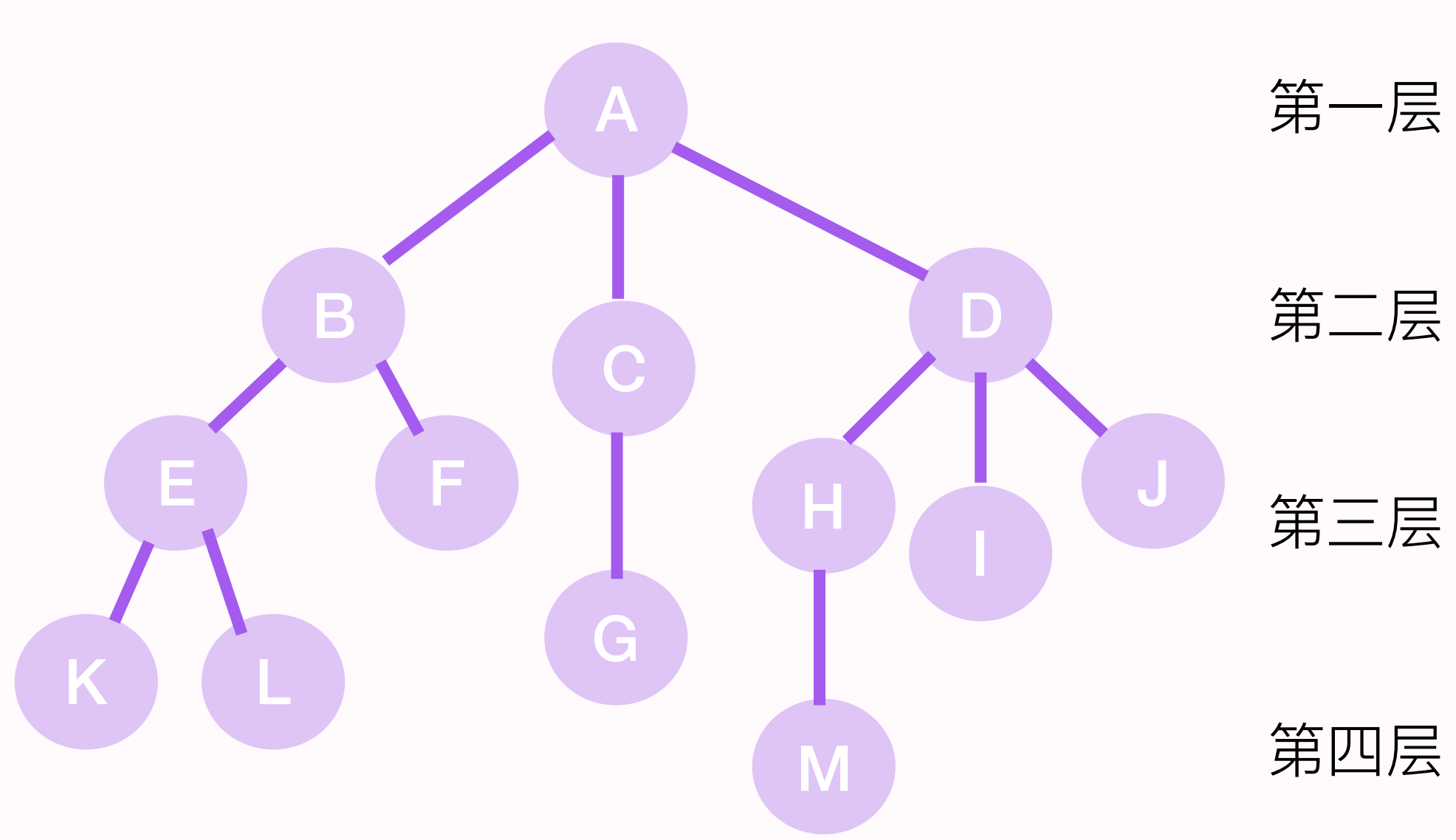
结点的层次：从根开始定义，根结点为第一层，它的子结点为第二层，依此类推。

结点的深度：从根结点开始自顶向下逐层累加。

结点的高度：从叶结点开始自底向上逐层累加。

树的高度：树中结点的最大层数。

高度为4



树的树形表示

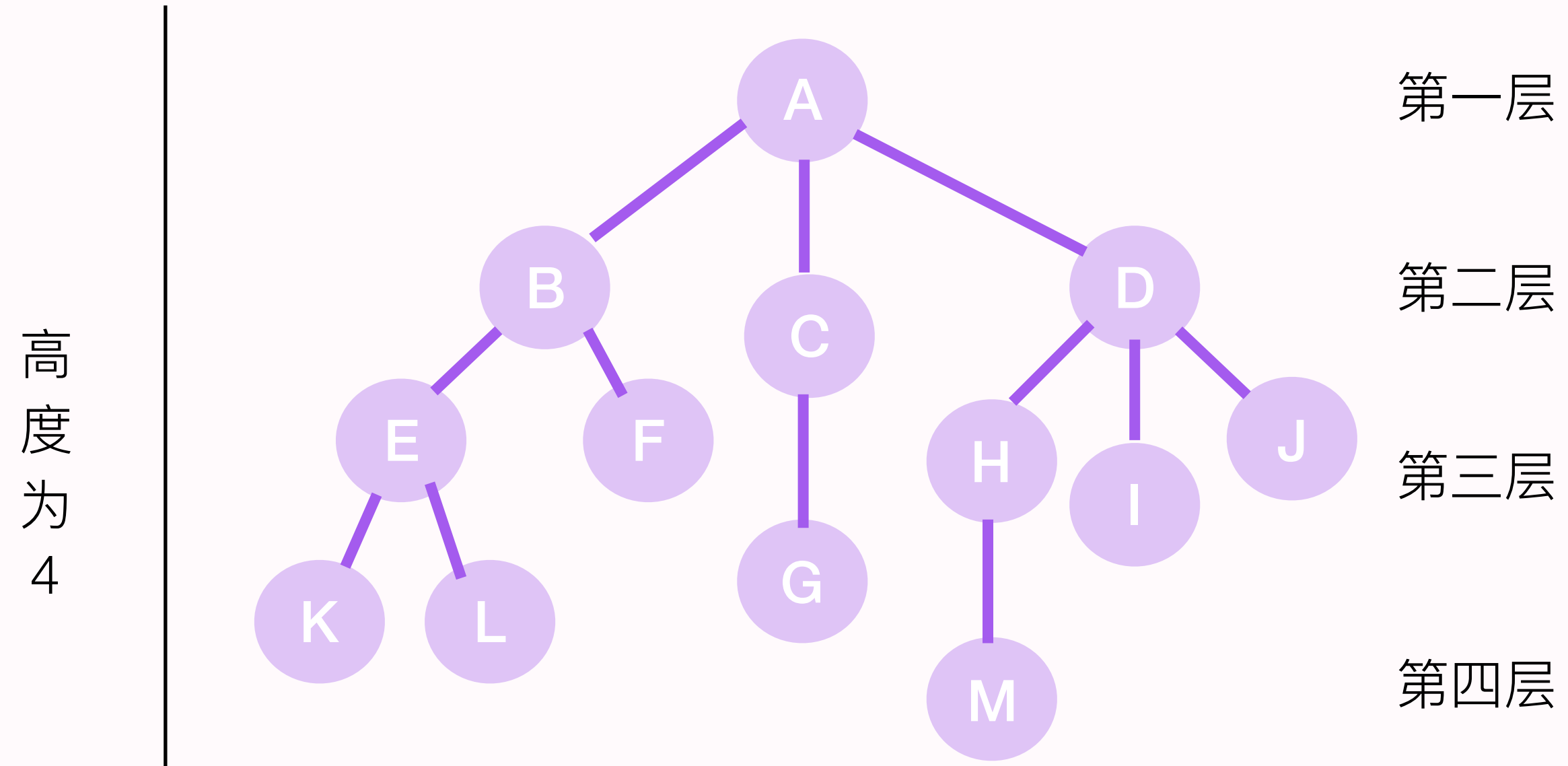
基本术语

有序树：树中结点的子树从左到右是有次序的，不能交换。

无序树：树中结点的子树从左到右没有次序，可以交换。

路径：树中两个结点之间的路径是由这两个结点之间所经过的结点序列构成的。

路径长度：路径上所经过的边的个数。



树的树形表示

例题5-1

树最适合用来表示（ ）的数据。

- A. 有序
- B. 无序
- C. 任意元素之间具有多种联系。
- D. 元素之间具有分支层次关系

解析5-1

D

树是一种分层结构，它特别适合组织那些具有分支层次关系的数据。

树的基本概念

小节1 / 定义及特点

小节2 / 基本术语

小节3 / 性质

树的性质

1. 树中的结点数等于所有结点的度数加1。
2. 度为m的树中第i层上至多有 m^{i-1} 个结点($i \geq 1$)。
3. 高度为h的m叉树至多有 $(m^h - 1) / (m - 1)$ 个结点。
4. 高具有n个结点的m叉树的最小高度为 $\log_m(n(m-1)+1)$ 向上取整。

例题5-2

一棵有 n 个结点的树的所有结点的度数之和为（ ）。

- A. $n-1$
- B. n
- C. $n+1$
- D. $2n$

解析5-2

A

除根结点外，其他每个结点都是某个结点的孩子结点，因此树中所有结点的度数加1等于结点数，所以所有结点的度数之和就等于总结点数减1。

例题5-3

树的路径长度是从树根到每个结点的路径长度到（ ）。

- A. 总和
- B. 最小值
- C. 最大值
- D. 平均值

解析5-3

A

树的路径长度是所有路径长度的总和，树根到每个结点的路径的最大值应是树的高度-1。

二叉树

小节1 / 基本概念

小节2 / 实体操作

小节3 / 具体应用

二叉树

小节1 / 基本概念

小节2 / 实体操作

小节3 / 具体应用

二叉树的定义

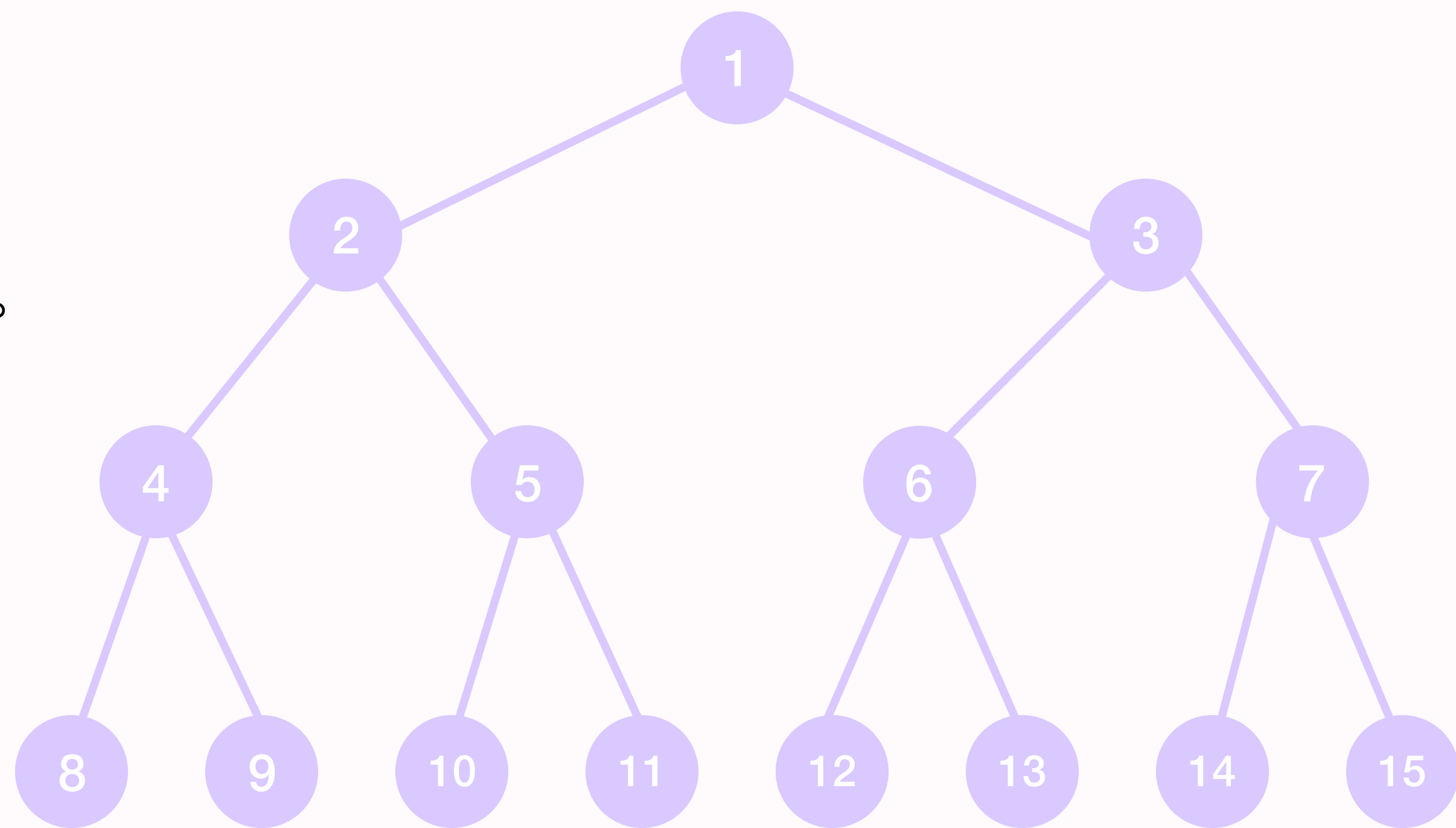
两个限制条件：

1. 每个结点至多有两棵树。
2. 子树有左右顺序之分，不能颠倒。

特殊的二叉树

满二叉树：

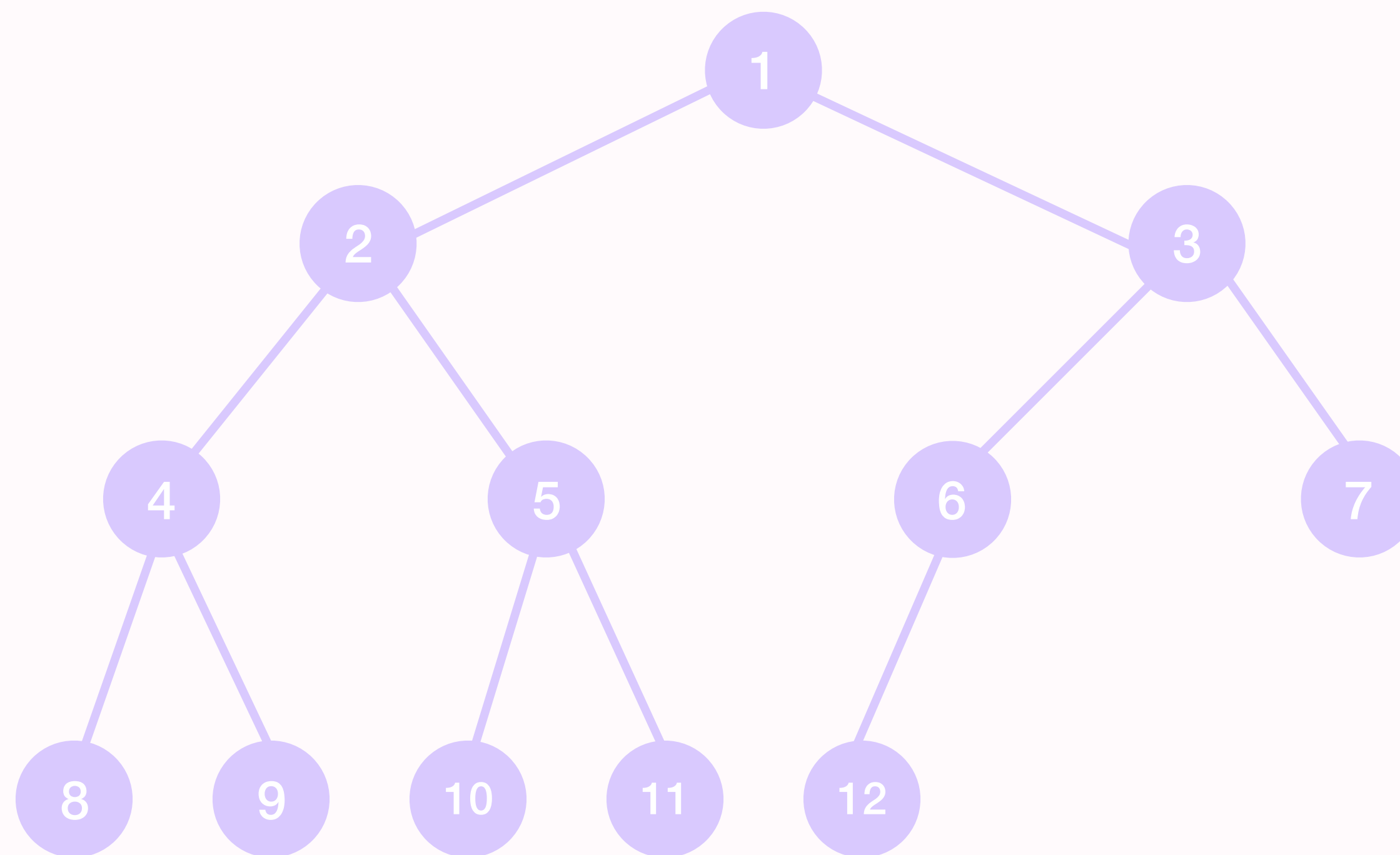
一棵高度为 h ，并且含有 2^h-1 个结点的二叉树。



特殊的二叉树

完全二叉树：

设一棵高度为 h ，有 n 个结点的二叉树，当且仅当其每一个结点都与高度为 h 的满二叉树中编号为其 $1 \sim n$ 的结点——对应时，称为完全二叉树。



特殊的二叉树

二叉排序树：

一棵二叉树或空二叉树，具有如下性质：

左子树上所有结点的关键字均小于根结点的关键字

右子树的所有结点的关键字均大于根结点的关键字

特殊的二叉树

平衡二叉树：

树上任一结点的左子树和右子树的深度之差不超过1。

二叉树的性质

1. 非空二叉树上叶子结点数等于度为2的结点数加1，即 $N_0 = N_2 + 1$ 。
2. 非空二叉树上第K层上至多有 2^{k-1} 个结点($k \geq 1$)。
3. 高度为H的二叉树至多有 $2^H - 1$ 个结点($H \geq 1$)。
4. 对完全二叉树按从上到下、从左到右的顺序依次编号1, 2, ..., N, 则:
 - 当 $i > 1$ 时，结点i的双亲编号为 $i/2$ 下取整
 - 当 $2i \leq N$ 时，结点i左孩子编号为 $2i$ ，否则无左孩子
 - 当 $2i+1 \leq N$ 时，结点i右孩子编号为 $2i+1$ ，否则无右孩子
 - 结点i所在层次（深度）为 $\log_2 i$ 下取整+1
5. 具有N个($N > 0$)结点的完全二叉树高度为 $\log_2(N+1)$ 上取整或 $\log_2 N$ 下取整+1

例题5-4 /

下列关于二叉树的说法中，正确的是（ ）。

- A. 度为2的有序树就是二叉树。
- B. 含有n个结点的二叉树的高度为 $\lceil \log_2 n \rceil + 1$ 。
- C. 在完全二叉树中，若一个结点没有左孩子，则它必是叶结点。
- D. 在任意一棵非空二叉树排序树中，删除某结点后又将其插入，则所得二叉排序树与删除前原二叉排序树相同。

解析5-4 /

C

- A.二叉树是有序树，在二叉树中，若某个结点只有一个孩子结点，则这个孩子结点的左右次序是确定的；而在度为2的有序树中，若某个结点只有一个孩子结点，则这个孩子结点就无须区分其左右次序。
- B.其仅当为完全二叉树时才有意义，对于任意一棵二叉树，高度可能为 $\lceil \log_2 n \rceil + 1 \sim n$ 。
- C.根据完全二叉树的定义，在完全二叉树中，若有度为1度结点，则只可能有一个，且该结点只有左孩子无右孩子。
- D.在二叉排序树中插入结点时，一定插入在叶结点的位置，故若先删除分支结点再插入，则会导致二叉排序树的重构，其结果就不再相同。

例题5-5 /

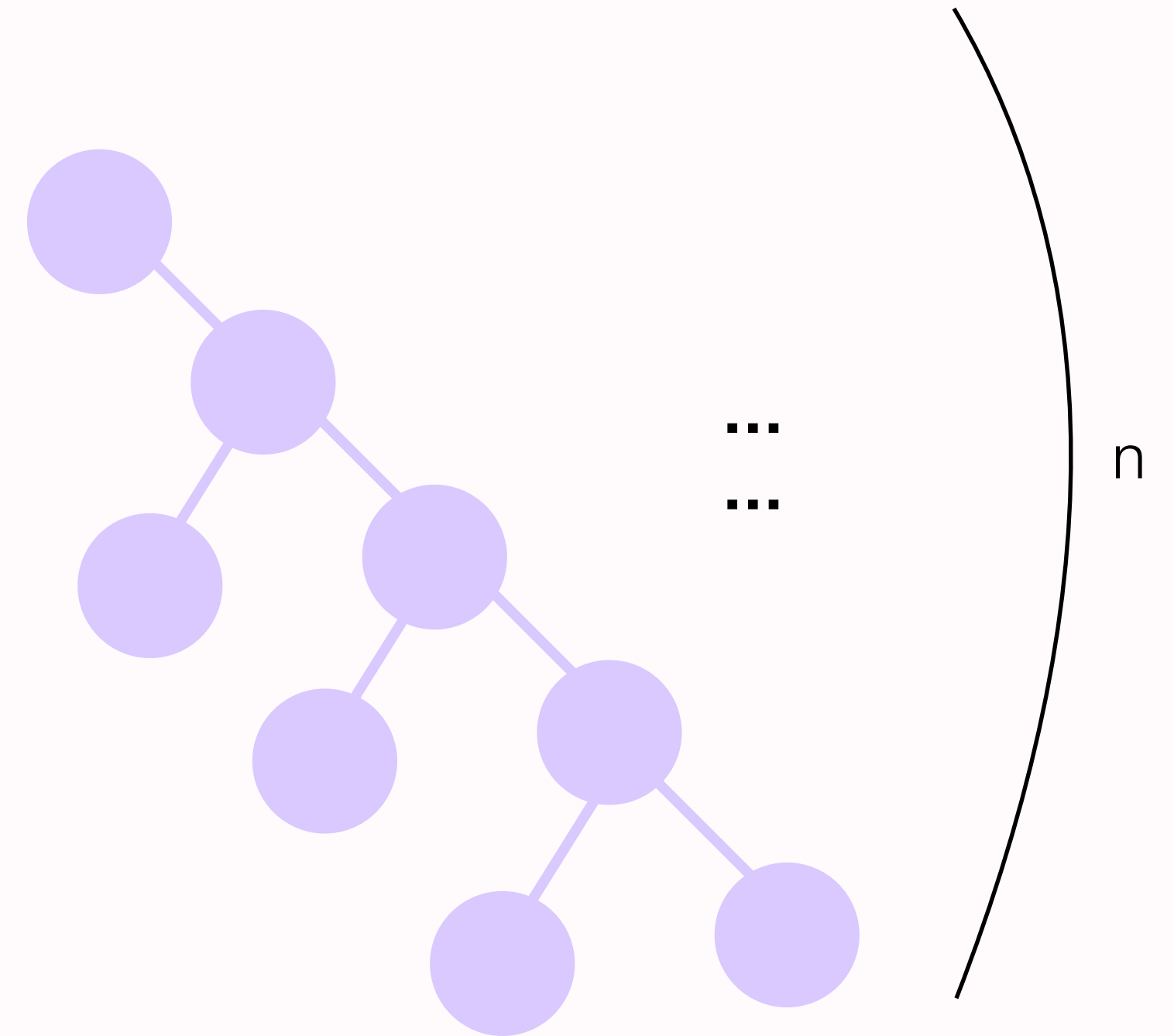
设高度为 h 的二叉树上只有度为0和度为2的结点，则此类二叉树中所包含的结点数至少为（ ）。

- A. h
- B. $2h-1$
- C. $2h+1$
- D. $h+1$

解析5-5 /

B

结点最少的情况如图，除根结点层只有1个结点外，其他 $h-1$ 层均有两个结点，结点总数= $2(h-1)+1=2h-1$ 。



二叉树的存储结构「顺序存储」

即用一个数组来存储一棵二叉树。

最适合于完全二叉树和满二叉树，用于存储一般二叉树会浪费大量的存储空间。

二叉树的存储结构「链式存储」

即用一个链表来存储一棵二叉树，二叉树中每一个结点用链表的一个链结点来存储。

包含数据域data，左指针域lchild，右指针域rchild。



二叉树的存储结构「链式存储」

链式存储的结构体定义



```
typedef struct BiTNode
{
    ElemType data; // 数据域
    struct BiTNode *lchild, *rchild; // 左、右孩子指针
} BiTNode, *BiTree
```

例题5-6 /

一棵有 n 个结点的二叉树采用二叉链存储结点，其中空指针数为（ ）。

- A. n
- B. $n+1$
- C. $n-1$
- D. $2n$

解析5-6 /

B

非空指针数=总分支数= $n-1$ ，

空指针数= $2 \times$ 结点总数-非空指针数= $2n - (n-1) = n+1$ 。

二叉树

小节1 / 基本概念

小节2 / 实体操作

小节3 / 具体应用

二叉树的遍历「先序遍历」

如果二叉树为空，什么也不做，否则：

1. 访问根结点
2. 先序遍历左子树
3. 先序遍历右子树



```
void PreOrder(BiTtree T){  
    if(T!=NULL){  
        visit(T); //访问根结点  
        PreOrder(T->lchild); //递归遍历左子树  
        PreOrder(T->rchild); //递归遍历右子树  
    }  
}
```

二叉树的遍历「中序遍历」

如果二叉树为空，什么也不做，否则：

1. 中序遍历左子树
2. 访问根结点
3. 中序遍历右子树



```
void InOrder(BiTtree T){  
    if(T!=NULL)  
    {  
        InOrder(T->lchild)//递归遍历左子树  
        visit(T);//访问根结点  
        InOrder(T->rchild);//递归遍历右子树  
    }  
}
```

二叉树的遍历「后序遍历」

如果二叉树为空，什么也不做，否则：

1. 后序遍历左子树
2. 后序遍历右子树
3. 访问根结点

```
void PostOrder(BiTtree T)
{
    if(T!=NULL)
    {
        PostOrder(T->lchild); //访问左子树
        PostOrder(T->rchild); //递归遍历右子树
        visit(T); //递归遍历根结点
    }
}
```

二叉树的遍历「递归算法和非递归算法的转换」

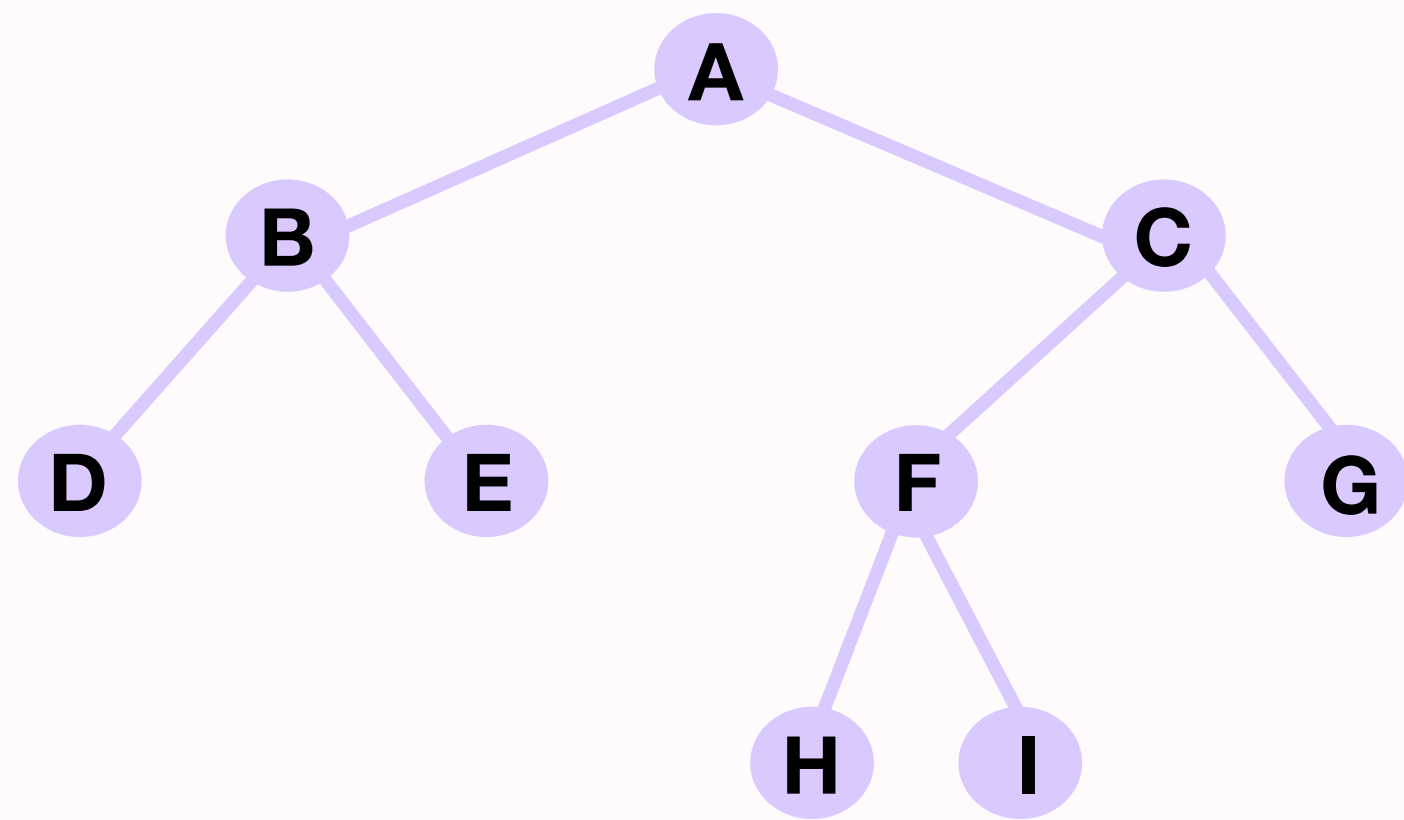
可以借助栈，将二叉树的递归遍历算法转换为非递归算法。

1. 扫描（并非访问）根结点的所有左结点，并将它们一一进栈
2. 出栈一个结点*p，则访问它
3. 扫描该结点的右孩子结点，将其进栈
4. 再扫描该右孩子结点的所有左结点并一一进栈
5. 如此继续，直至栈空

```
void InOrder(BiTree T)
{
    //二叉树中序遍历的非递归算法，算法需要借助一个栈
    InitStack (S) ; //初始化栈
    BiTree p=T; //p是遍历指针
    while(p||!IsEmpty(S)) //栈不空或p不空时循环
    {
        if(p) //根指针进栈，遍历左子树
        {
            Push(S,p); //每遇到非空二叉树先向左走
            p=p->lchild;
        }
        else
        {
            Pop(S,p); //根指针退栈，访问根结点，遍历右子树
            visit(p); //退栈，访问根结点
            p=p->rchild; //再向右子树走
        }
    }
}
```

二叉树的遍历「层次遍历」

按照箭头所指方向，按照1、2、3、4的层次顺序，对二叉树中各个结点进行访问。



```
void LevelOrder(BiTree T)
{
    InitQueue(Q); // 初始化辅助队列
    BiTree p;
    Enqueue(Q, T); // 将跟结点入队
    while (!IsEmpty(Q)) { // 队列不空循环
        Dequeue(Q, p); // 队头元素出队
        visit(p); // 访问当前p所指向结点
        if (p->lchild != NULL)
            Enqueue(Q, p->lchild); // 左子树不空，则左子树入队列
        if (p->rchild != NULL)
            Enqueue(Q, p->rchild); // 右子树不空，则右子树入队列
    }
}
```

例题5-7 /

对二叉树的结点从1开始进行连续编号，要求每个结点的编号大于其左、右孩子的编号，同一结点的左、右孩子中，其左孩子的编号小于右孩子的编号，可采用（ ）次序的遍历实现编号。

- A. 先序遍历
- B. 中序遍历
- C. 后序遍历
- D. 层次遍历

解析5-7 /

C

对每个顶点从1开始按序编号，要求结点编号大于其左、右孩子编号，并且左孩子编号小于右孩子编号。编号越大说明遍历顺序越靠后，因此，三者遍历顺序为先左子树再右子树后根结点，4个选项中仅后序遍历满足要求。

线索二叉树的基本概念

若无左子树，令lchild指向其前驱结点；
若无右子树，令rchild指向其后继结点。

ltag 0 lchild域指示结点的左孩子
 1 lchild域指示结点的前驱

rtag 0 rchild域指示结点的右孩子
 1 rchild域指示结点的后继



线索二叉树的存储结构



```
typedef struct ThreadNode
{
    ElemType data; // 数据元素
    struct ThreadNode *lchild, *rchild; // 左、右孩子指针
    Int ltag, rtag; // 左、右线索标志
} ThreadNode, *ThreadTree;
```

线索二叉树的构造

对二叉树的线索化，实质上就是遍历一次二叉树。

在遍历过程中，检查当前结点左右指针域是否为空，若为空，将它们改为指向前驱结点或后继结点的线索。

线索二叉树的遍历

求中序线索二叉树中中序序列下的第一个结点



```
ThreadNode *Firstnode(ThreadNode *p)
{
    while(p->Itag == 0) p=p->Ichild; //最左下结点 (不一定是叶结点)
    return p;
}
```

线索二叉树的遍历

求中序线索二叉树中结点p在中序序列下的后继结点



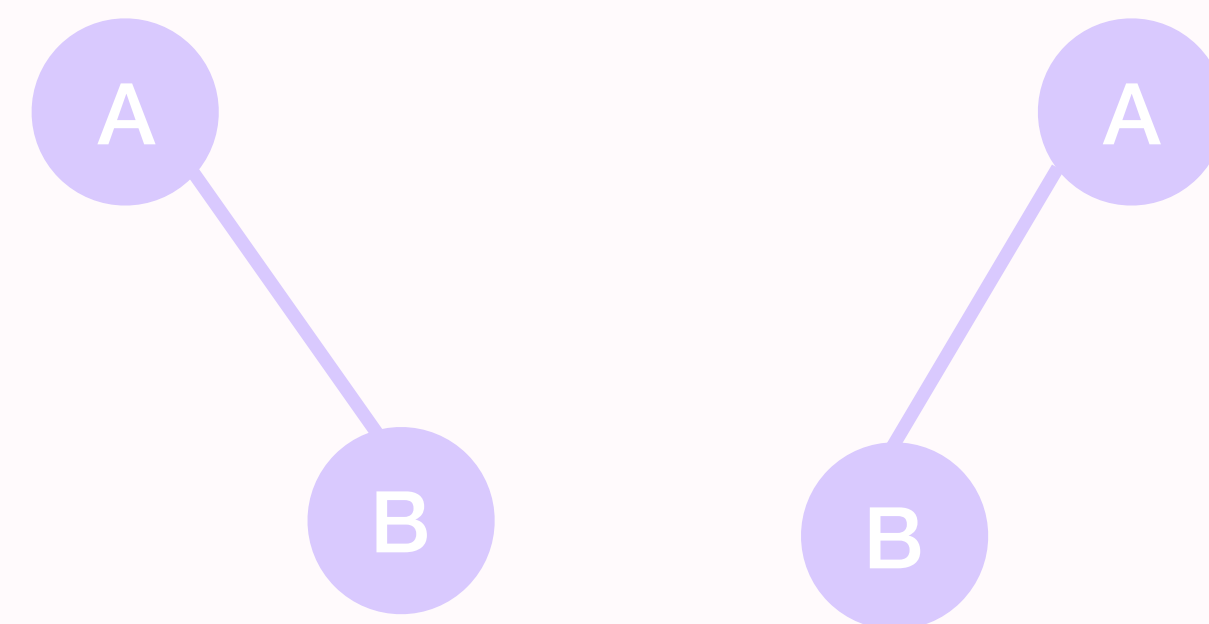
```
ThreadNode *Nextnode(ThreadNode *p)
{
    if(p->rtag == 0) return Firstnode(p->rchild); //rtag==1 直接返回后继线索
    else return p->rchild;
}
```

例题5-8 / 下列序列中，不能唯一地确定一棵二叉树的是（ ）。

- A. 层次序列和中序序列
- B. 先序序列和中序序列
- C. 后序序列和中序序列
- D. 先序序列和后序序列

解析5-8 / **D**

先序序列为NLR，后序序列为LRN，虽然可以唯一确定树的根结点，但无法划分左右子树。例如，先序遍历为AB，后序遍历为BA，则其对应的二叉树如右图所示。



二叉树

小节1 / 基本概念

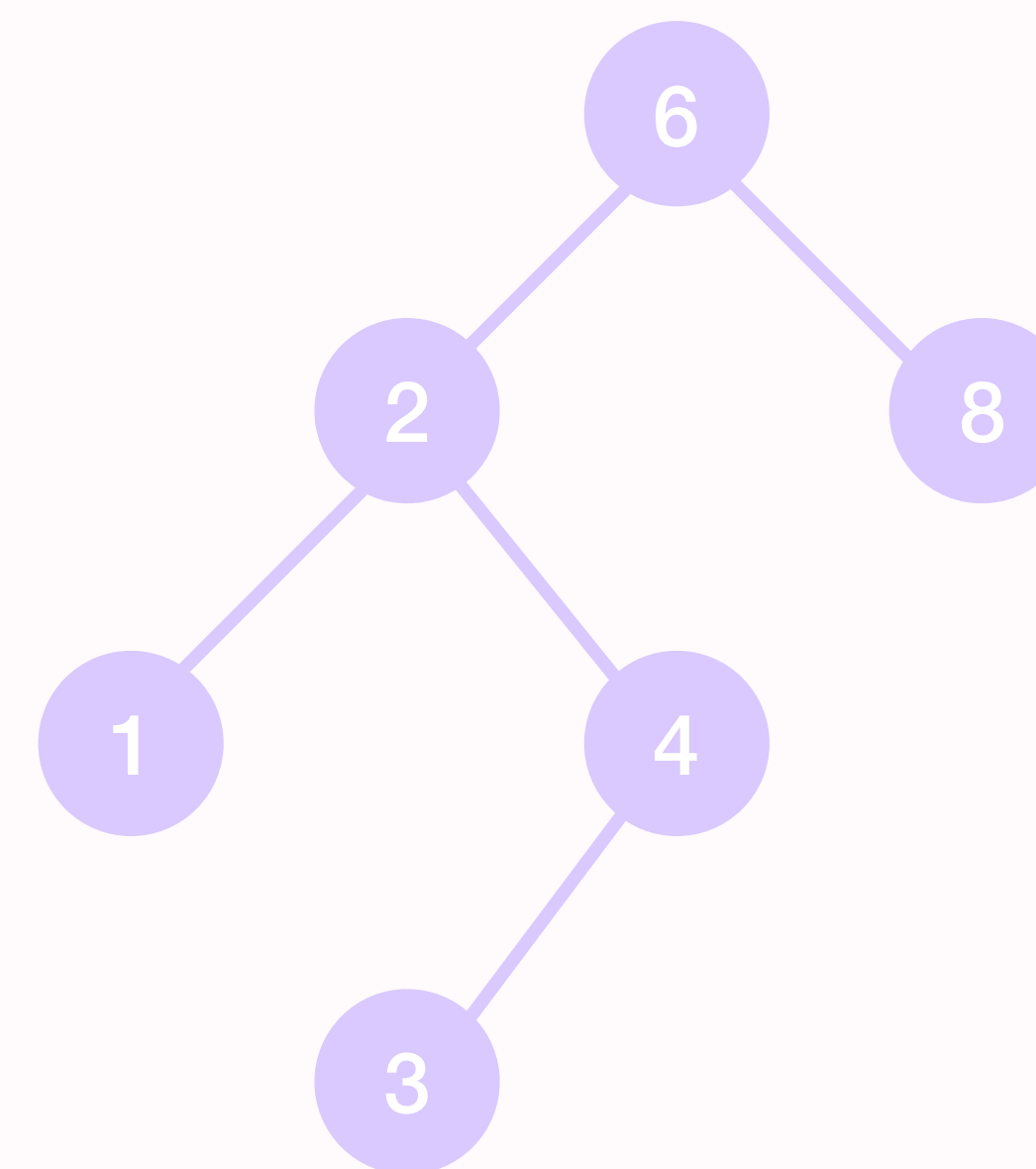
小节2 / 实体操作

小节3 / 具体应用

二叉树的应用「排序二叉树」

排序二叉树的定义

1. 若左子树非空，则左子树上所有结点关键字值均小于根结点的关键字值。
2. 若右子树非空，则右子树上所有结点关键字值均大于根结点的关键字值。
3. 左、右子树本身也是一棵二叉排序树。



二叉树的应用「排序二叉树」

排序二叉树的查找

从根结点开始，沿某一个分支逐层向下进行比较

若二叉树非空，将给定值与根结点的关键字比较，若相等，则查找成功；若不等，则当根结点的关键字大于给定关键字时，在根结点的左子树中查找，否则在根结点的右子树中查找。

二叉树的应用「排序二叉树」

排序二叉树的插入

原二叉排序树为空，则直接插入结点；

否则，若关键字 k 小于根结点关键字，则插入到左子树中，

若关键字 k 大于根结点关键字，则插入到右子树中。

二叉树的应用「排序二叉树」

排序二叉树的构造

每读入一个元素，就建立一个新结点，
若二叉排序树非空，则将新结点的值与根结点的值比较，
如果小于根结点的值，则插入到左子树中，
否则插入到右子树中；
若二叉排序树为空，则新结点作为二叉排序树的根结点。

二叉树的应用「排序二叉树」

排序二叉树的删除

1. 如果被删除结点 z 是叶结点，则直接删除，不会破坏二叉排序树的性质。
2. 若结点 z 只有一棵左子树或右子树，则让 z 的子树成为 z 父结点的子树，替代 z 的位置。
3. 若结点 z 有左、右两棵子树，则令 z 的直接后继（或直接前驱）替代 z ，然后从二叉排序树中删去这个直接后继（或直接前驱）。

二叉树的应用「平衡二叉树」

平衡二叉树的定义

在插入和删除二叉树结点时，保证任意结点的左、右子树高度差的绝对值不超过1，这样的二叉树称为平衡二叉树。

结点左右子树与右子树的高度差为该结点的平衡因子，则平衡二叉树结点的平衡因子的值只可能是-1、0或1。

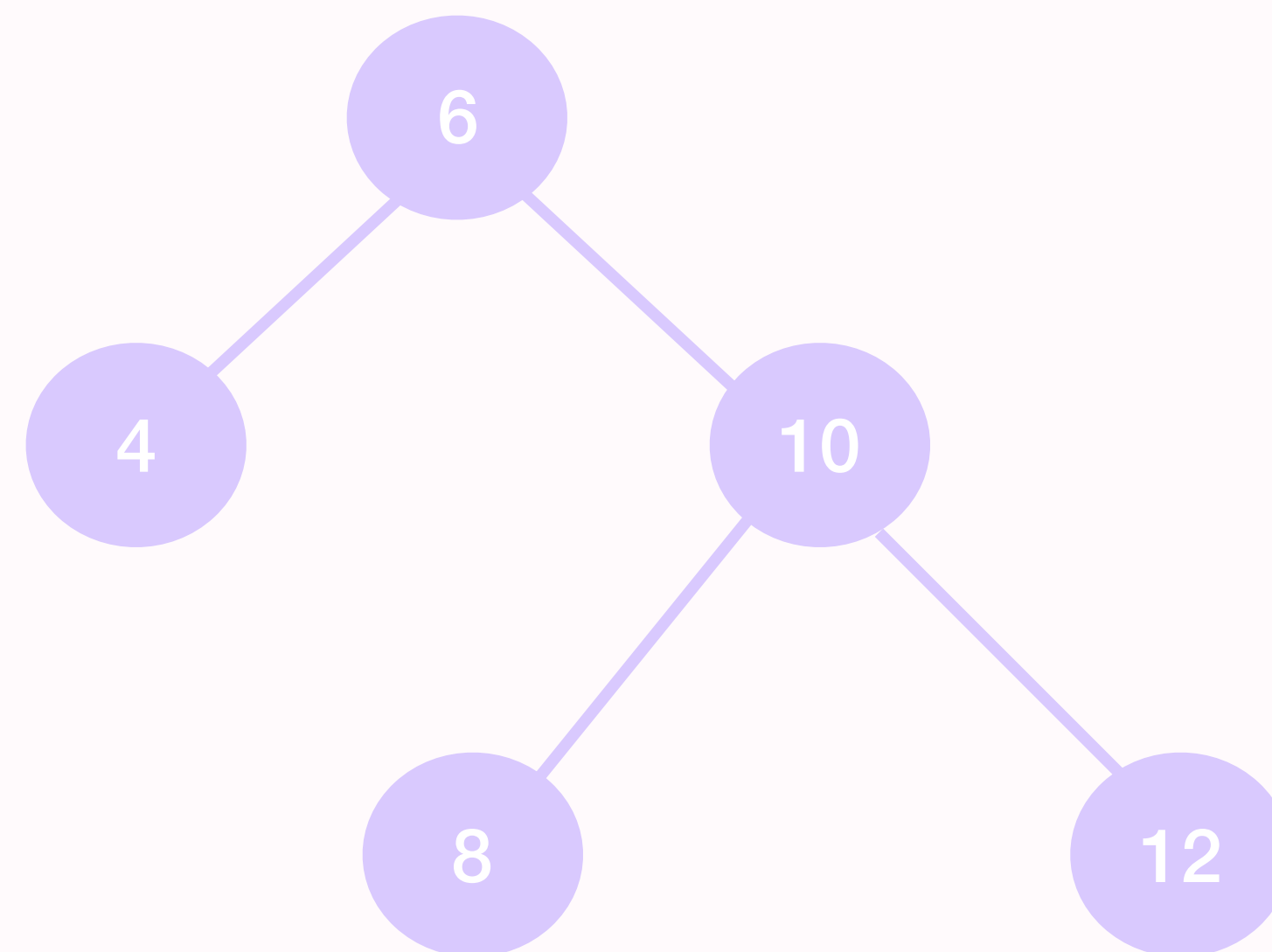
二叉树的应用「平衡二叉树」

平衡二叉树的插入

1. LL平衡旋转
2. RR平衡旋转
3. LR平衡旋转
4. RL平衡旋转

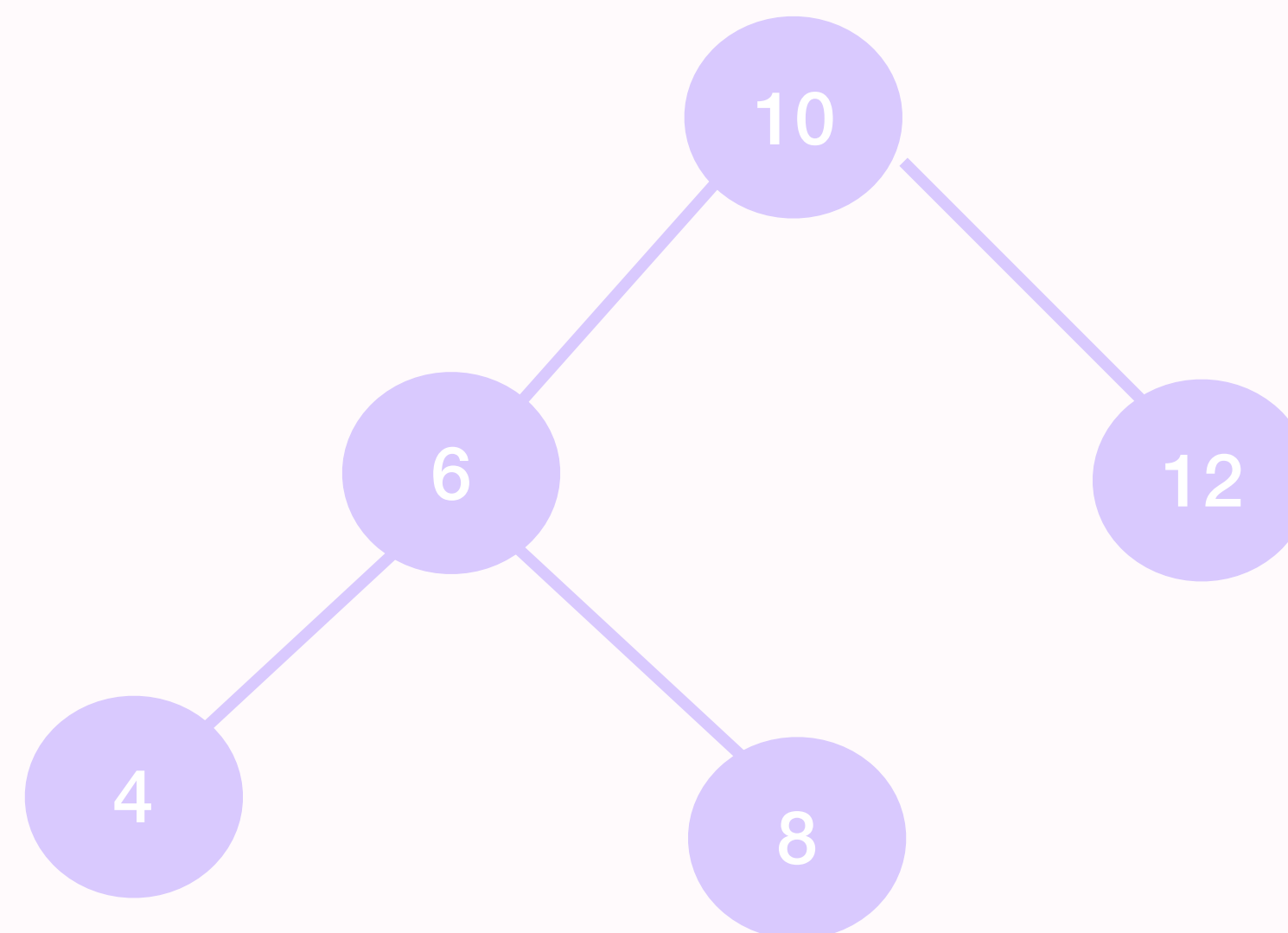
二叉树的应用「平衡二叉树」

LL平衡旋转



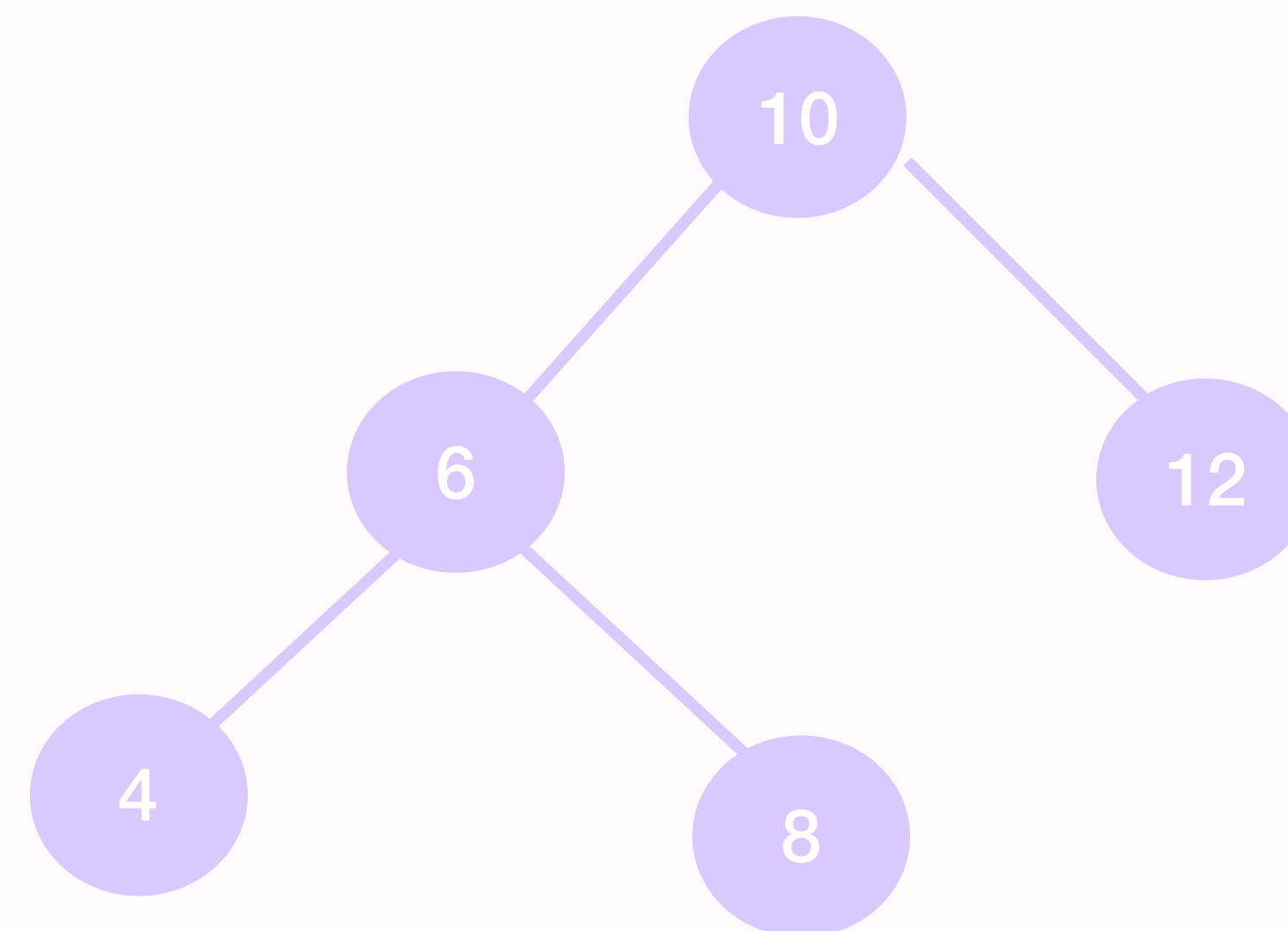
二叉树的应用「平衡二叉树」

LL平衡旋转



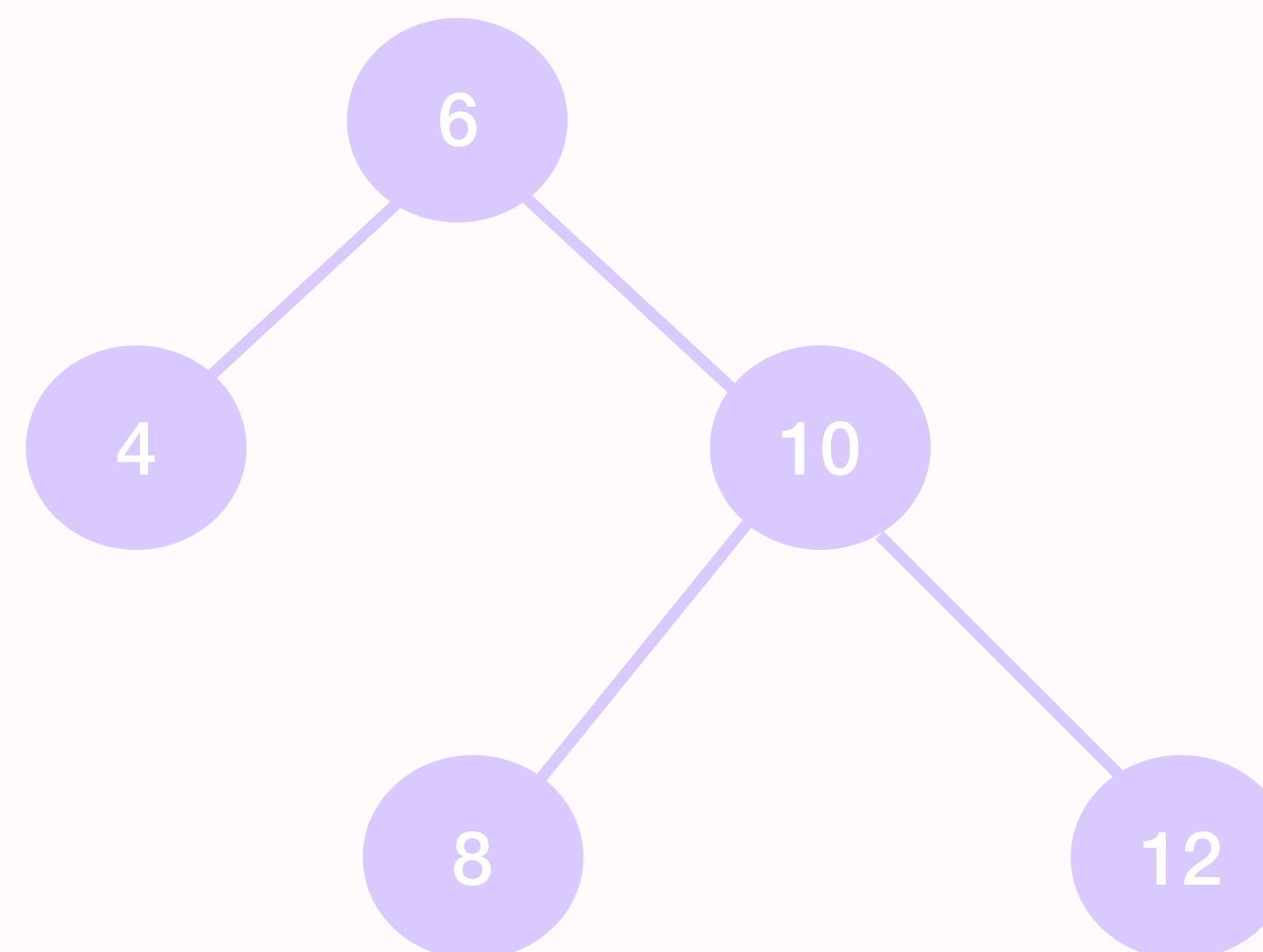
二叉树的应用「平衡二叉树」

RR平衡旋转



二叉树的应用「平衡二叉树」

RR平衡旋转



二叉树的应用「平衡二叉树」

平衡二叉树的查找

过程和二叉排序树相同，在查找过程中和给定值进行比较的关键字个数不超过树的深度。

含有 n 个结点的平衡二叉树的最大深度为 $O(\log_2 n)$ 。

平衡二叉树的平均查找长度为 $O(\log_2 n)$ 。

二叉树的应用「哈夫曼树」

哈夫曼树的定义

在含有N个带权叶子结点的二叉树中，其中带权路径长度（WPL）最小的二叉树称为哈夫曼树，也称为最优二叉树。

二叉树的应用「哈夫曼树」

哈夫曼树的构造

1. 将这N个结点分别作为N棵仅含有一个结点的二叉树，构成森林F。
2. 构造一个新结点，并从F中选取两棵根结点权最小的树作为新结点的左、右子树，并且将新结点的权值置为左、右子树上根结点的权值之和。
3. 从F中删除刚才选出的两棵树，同时将新得到的树加入F中。
4. 重复步骤2和3，直至F中只剩下一棵树为止。

二叉树的应用「哈夫曼树」

哈夫曼树的特点

1. 权值越大的结点，距离根结点越近。
2. 树中没有度为1的结点。
3. 树的带权路径长度最短

二叉树的应用「哈夫曼编码」

哈夫曼树的特点

1. 权值越大的结点，距离根结点越近。
2. 树中没有度为1的结点。
3. 树的带权路径长度最短。

树、森林

小节1 / 概念

小节2 / 操作

小节3 / 应用

树、森林

小节1 / 概念

小节2 / 操作

小节3 / 应用

森林的定义

森林： 是 $m(m \geq 0)$ 棵互不相交的树的集合。

只要把树的根结点删去就成了森林。

树的存储结构「顺序存储结构」

双亲存储结构

采用一组连续空间来存储每个结点。

在每个结点中增设一个伪指针，指示其双亲结点在数组中的位置。

优点：可以很快得到每个结点的双亲结点

缺点：求结点的孩子是需要遍历整个结构



```
#define MAX_TREE_SIZE 100//树中最多结点数
typedef struct{//树的结点定义
    ElemType data;//数据元素
    int parent;//双亲位置域
}PTNode;
```



```
typedef struct{//树的类型定义
    PTNode nodes[MAX_TREE_SIZE];//双亲表示
    int n;//结点数
}PTree;
```

树的存储结构「链式存储结构」

孩子表示法

将每个结点的孩子结点都用单链表链接起来形成一个线性结构。

N个结点就有N个孩子链表。

优点：寻找子女的操作非常直接

缺点：寻找双亲的操作需要遍历N个结点中孩子链表指针域所指向的N个孩子链表

树的存储结构「链式存储结构」

孩子兄弟存储结构

以二叉链表作为树的存储结构。

每个结点包括三部分：结点值、指向结点第一个孩子结点的指针、指向结点下一个兄弟结点的指针。



```
typedef struct CSNode{
    ElemType data; //数据域
    struct CSNode *firstchild, *nextsibling; //第一个孩子和右兄弟指针
}CSNode, *CSTree;
```

优点：可以方便地实现树转换为二叉树的操作，易于查找结点的孩子

缺点：从当前结点查找其双亲结点比较麻烦

例题5-8 /

利用二叉链表存储森林，则根结点的右指针是（ ）。

- A. 指向最左兄弟
- B. 指向最右兄弟
- C. 一定为空
- D. 不一定为空

解析5-8 /

D

森林与二叉树具有对应关系，因此，我们存储森林的时候应先将森林转换成二叉树，转换的方法就是“左孩子右兄弟”，与树不同的是，如果存在第二棵树，二叉链表的根结点的右指针指向的是森林中第二棵树的根结点。若此森林只有一棵树，那么根结点的右指针为空。因此，右指针可能为空也可能不为空。

树、森林

小节1 / 概念

小节2 / 操作

小节3 / 应用

树转换为二叉树

1. 将同一结点的各孩子结点用线串起来
2. 将每个结点的分支从左到右除了第一个以外，其余的都剪掉
3. 调整结点使之符合二叉树的层次结构

树转换为二叉树

1. 将同一结点的各孩子结点用线串起来
2. 将每个结点的分支从左到右除了第一个以外，其余的都剪掉
3. 调整结点使之符合二叉树的层次结构

树转换为二叉树

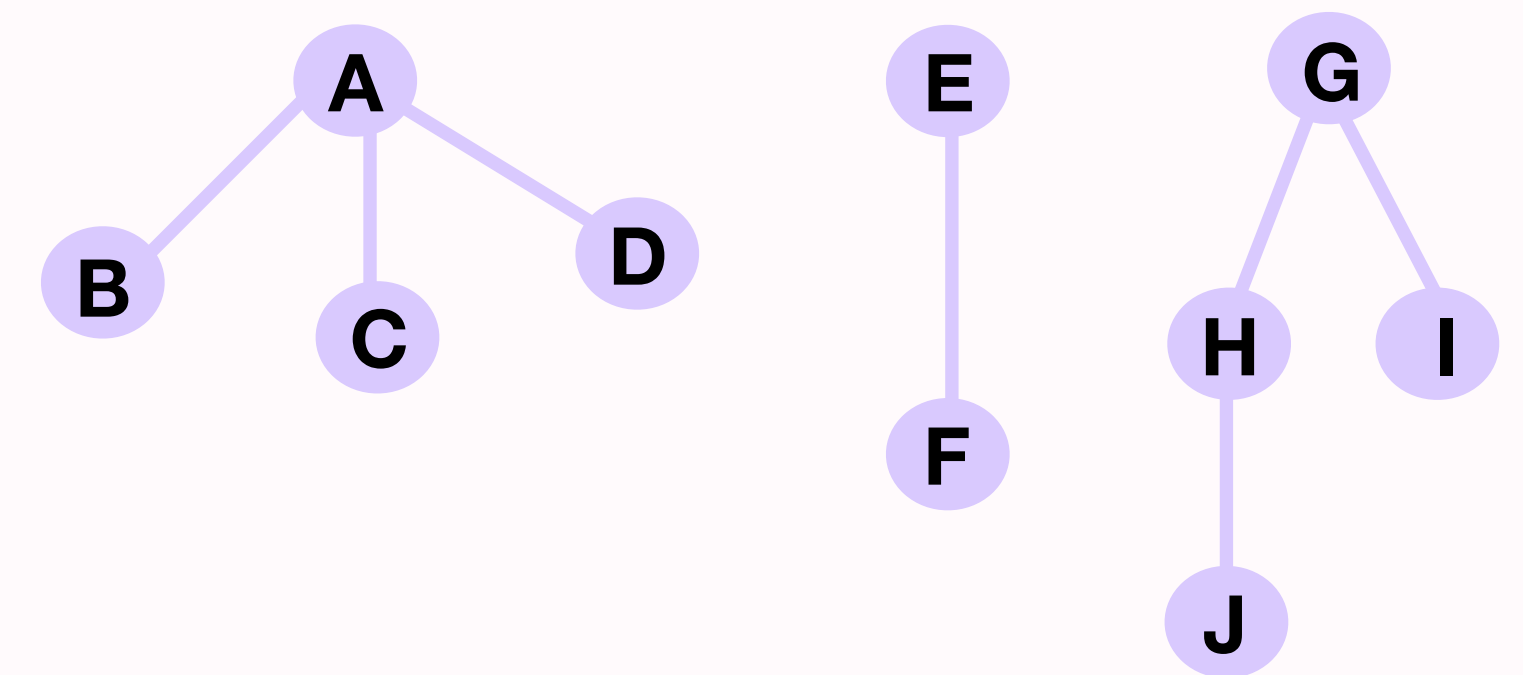
1. 将同一结点的各孩子结点用线串起来
2. 将每个结点的分支从左到右除了第一个以外，其余的都剪掉
3. 调整结点使之符合二叉树的层次结构

二叉树转换为树

- 1.先把二叉树从左上到右下分为若干层
2. 找到每一层结点在其上一层的父结点
- 3.将每一层的结点和其父结点相连

森林转换为二叉树

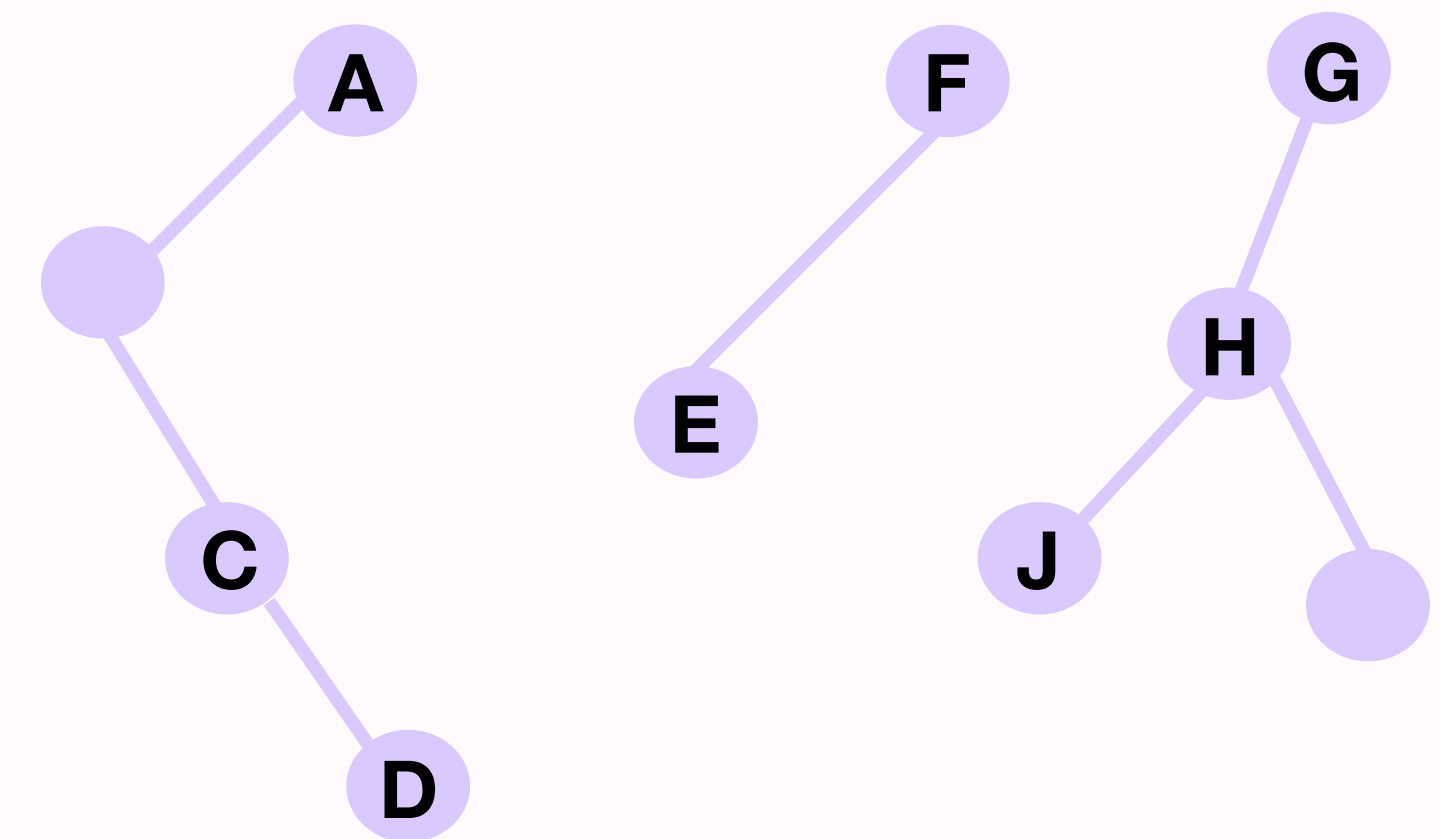
- 1.先将森林中的三棵树分别转换为二叉树
2. 将第二棵二叉树作为第一棵二叉树根的右子树，
将第三棵二叉树作为第二棵二叉树根的右子树



拥有三棵树的森林

森林转换为二叉树

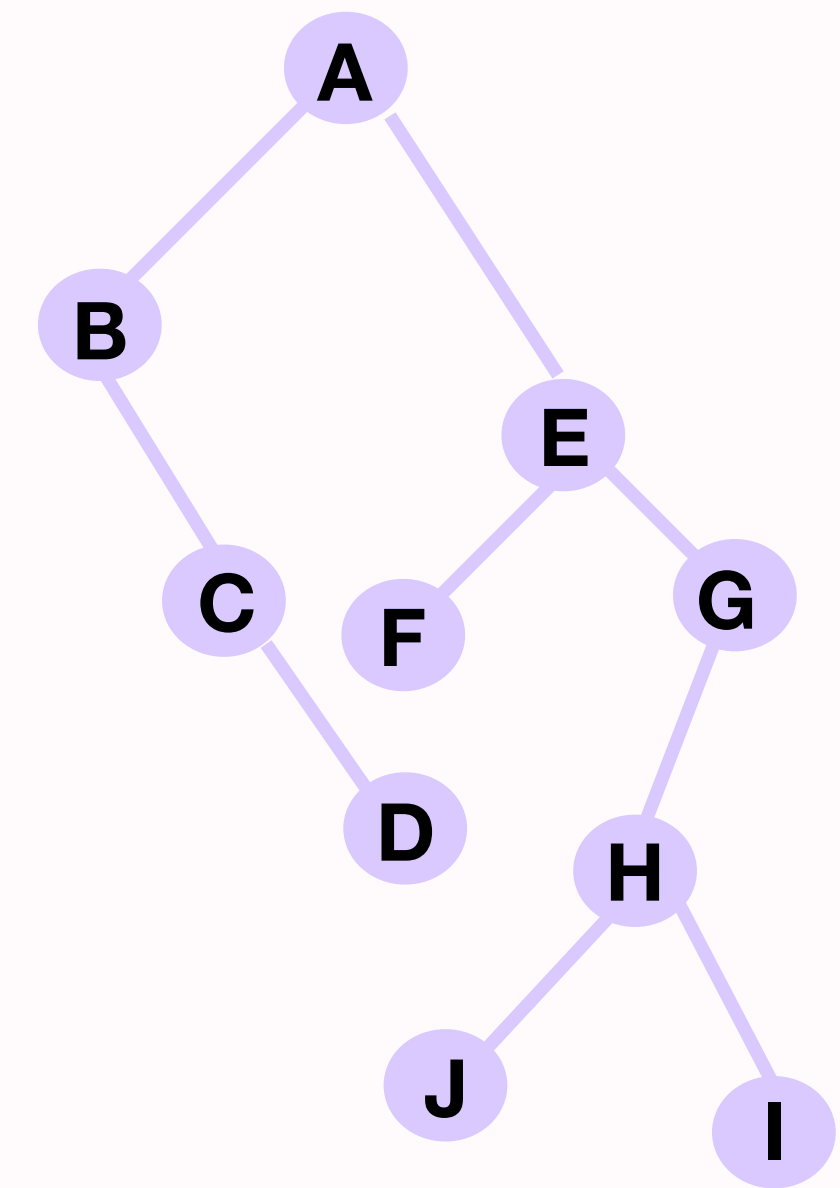
- 1.先将森林中的三棵树分别转换为二叉树
2. 将第二棵二叉树作为第一棵二叉树根的右子树，
将第三棵二叉树作为第二棵二叉树根的右子树



森林中每一棵树都转换为二叉树

森林转换为二叉树

- 1.先将森林中的三棵树分别转换为二叉树
2. 将第二棵二叉树作为第一棵二叉树根的右子树，
将第三棵二叉树作为第二棵二叉树根的右子树



森林中每一棵树都转换为二叉树

二叉树转换为森林

- 1.若二叉树非空，则二叉树根及其左子树作为第一棵二叉树的形式
2. 二叉树根的右子树看作是一个由除第一棵树外的森林转换后的二叉树
3. 直至最后一棵没有右子树的二叉树为止

例题5-9 /

已知一棵有2011个结点的树，其叶结点个数为116，该树对应的二叉树中无右孩子的结点个数是（ ）。

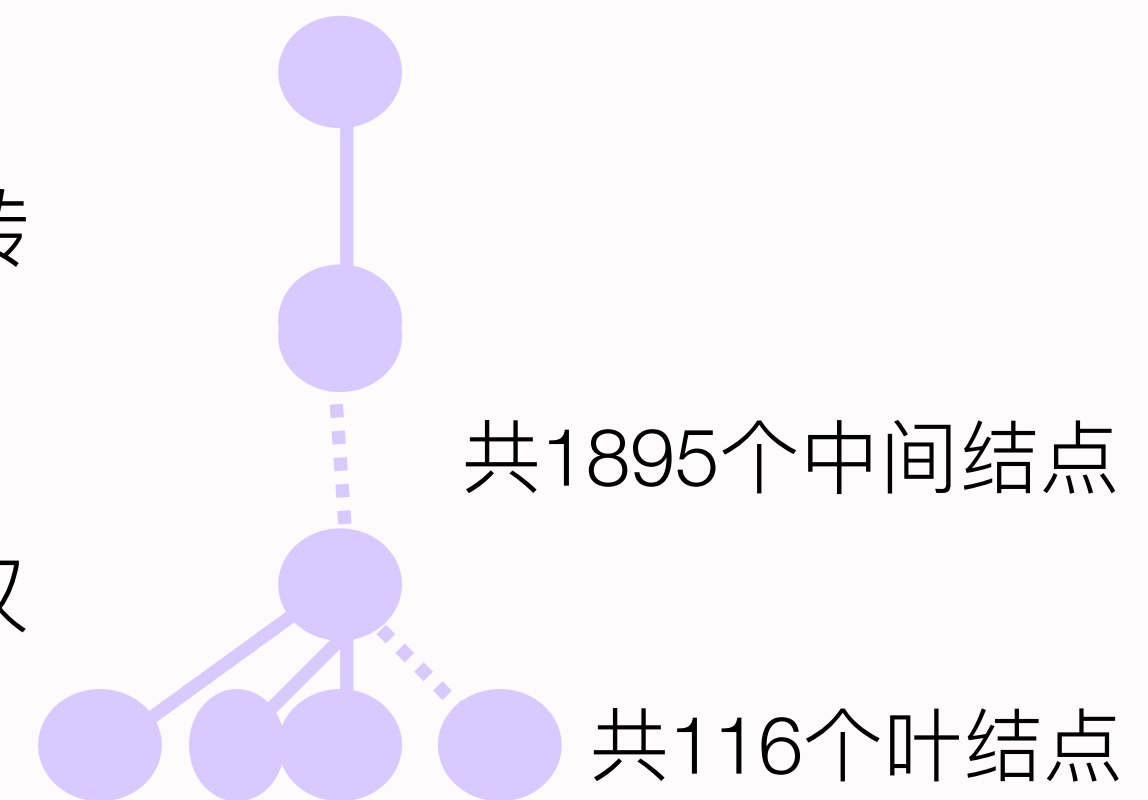
- A. 115
- B. 116
- C. 1895
- D. 1896

解析5-9 /

D

树转二叉树时，树的每个分支结点的所有子结点中的最右子结点无右孩子，根结点转换后也没有右孩子，因此，对应对二叉树中无右孩子的结点个数=分支结点数+1=2011-116+1=1896。

通常本题应采用特殊法求解，设题意中的树如右图所示的结构，则对应的二叉树中仅有前115个叶结点有右孩子，故无右孩子的结点个数=2011-115=1896。



树的遍历

先序遍历：先访问根结点，然后先序遍历每一棵子树

后序遍历：先后序遍历根结点的每一棵子树，然后再访问根结点

森林的遍历

先序遍历：先访问森林中第一棵树的根结点，然后先序遍历第一棵树中根结点的子树，最后先序遍历森林中除第一棵树以外的其他树。

后序遍历：先后序遍历第一棵树中根结点的子树，然后访问第一棵树的根结点，最后后序遍历森林中除去第一棵树以外的森林。

树、森林

小节1 / 概念

小节2 / 操作

小节3 / 应用

并查集

并查集的结构体定义



```
#define SIZE 100  
int UFSets[SIZE]; //集合元素数组 (双亲指针数组)
```

并查集

并查集的初始化操作



```
void Initial(int S[ ])  
{  
    for(int i = 0; i<size; i++)//每个自成单元元素集合  
        S[i]=-1;  
}
```

并查集

Find操作：在并查集S中查找并返回包含元素x的树的根



```
int Find(int S[ ], int x)
{
    while(S[x]>=0)//循环寻找x的根
        x=S[x];
    return x;
}
```



并查集

Union操作：求两个不相交子集合的并集



```
void Union(int S[ ], int Root1, int Root2)//要求Root1和Root2不同, 且表示子集合的名字
{
    S[Root2] = Root1;//将根Root2连接到另一根Root1下面
}
```

树与二叉树

斐多课堂  数据结构  第五讲
Phaedo Classes