期中试卷讲解

填空题（1）

一个栈的进栈序列是A、B、C，写出所有可能的出栈序列____。

答：ABC，ACB，BAC，BCA，CBA

栈：先进后出

1. 先出A
2. 先出B
3. 先出C

进栈序列有 n 个元素，那么出栈序列就有 $C_{2n}^{n}-C_{2n}^{n+1}$ 种情况

队列？双端队列？

填空题（2）

用数组q[1…m]表示的顺序存储队列，队列中元素个数最多为m，用front表示队头指针，指向队头元素的前一位置，初值为0，用rear表示队尾指针，指向队尾元素，判断队列发生假溢出的条件用C/C++语言语法表达为_____。

答：(rear==m) && (front!=0)

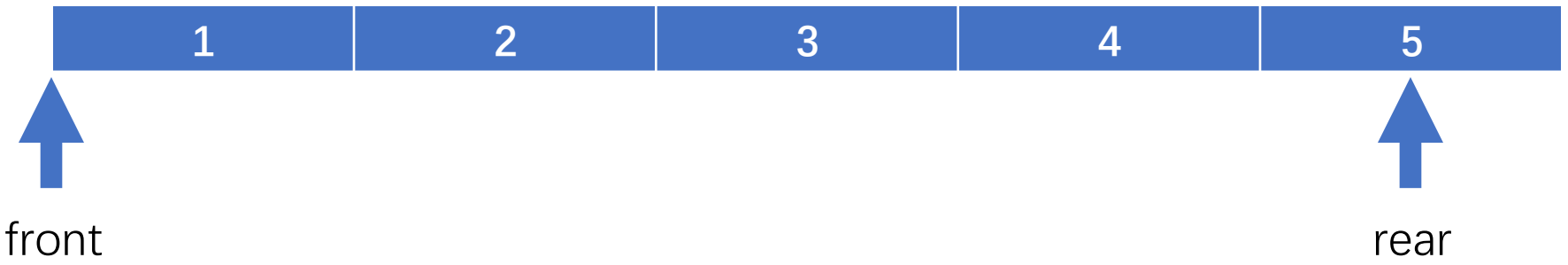
什么是假溢出？----队列存储区未满时发生溢出的现象

填空题（2）

用数组q[1…m]表示的顺序存储队列，队列中元素个数最多为m，用front表示队头指针，指向队头元素的前一位置，初值为0，用rear表示队尾指针，指向队尾元素，判断队列发生假溢出的条件用C/C++语言语法表达为_____。

答：(rear==m) && (front!=0)

什么是假溢出？----队列存储区未满时发生溢出的现象

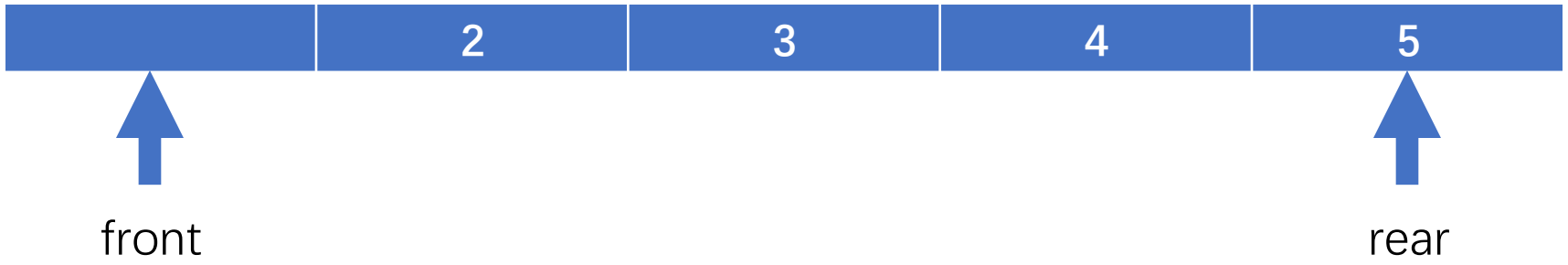


填空题（2）

用数组q[1…m]表示的顺序存储队列，队列中元素个数最多为m，用front表示队头指针，指向队头元素的前一位置，初值为0，用rear表示队尾指针，指向队尾元素，判断队列发生假溢出的条件用C/C++语言语法表达为_____。

答：(rear==m) && (front!=0)

什么是假溢出？----队列存储区未满时发生溢出的现象

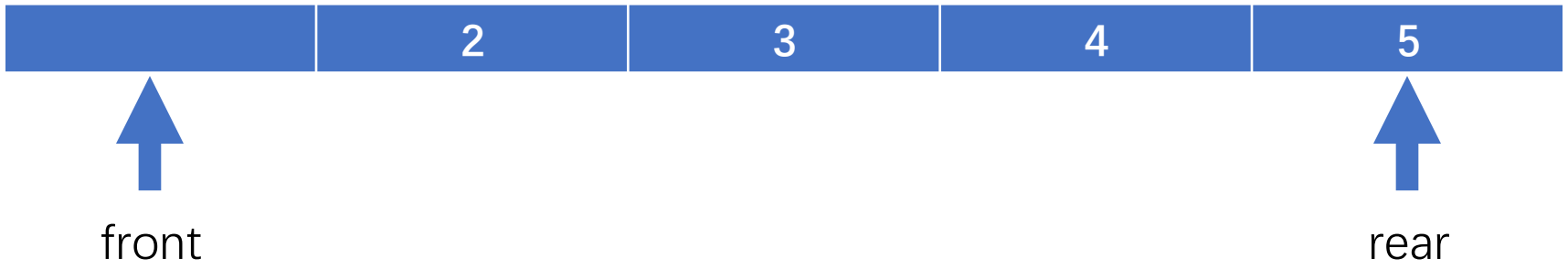


填空题（2）

用数组q[1…m]表示的顺序存储队列，队列中元素个数最多为m，用front表示队头指针，指向队头元素的前一位置，初值为0，用rear表示队尾指针，指向队尾元素，判断队列发生假溢出的条件用C/C++语言语法表达为_____。

答：(rear==m) && (front!=0)

什么是假溢出？----队列存储区未满时发生溢出的现象



解决方法：循环队列

填空题（3）

广义表GL = ((a, b, c), (d, e, f))，假设求表头操作为Head，求表尾操作为Tail，则f = _____。

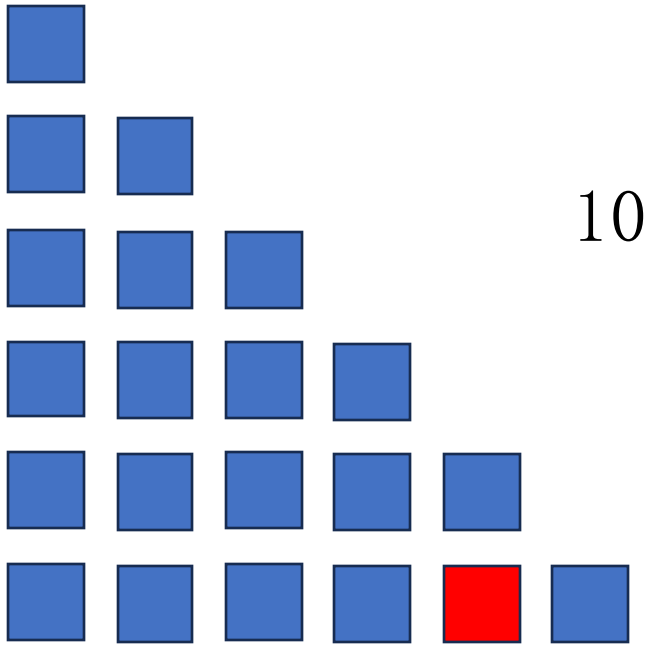
答：head[tail[tail[head[tail[GL]]]]]

1. 取表尾：tail[GL] = ((d, e, f))
2. 取 tail[GL] 的表头：head[tail[GL]] = (d, e, f)
3. 取 head[tail[GL]] 的表尾：tail[head[tail[GL]]] = (e, f)
4. 取 tail[head[tail[GL]]] 的表尾：tail[tail[head[tail[GL]]]] = (f)
5. 取 tail[tail[head[tail[GL]]]] 的表头： head[tail[tail[head[tail[GL]]]]] = f

填空题（4）

已知数组A[1…10，1…10]为对称矩阵，其中每个元素占5个单元。现将其下三角部分按行优先次序存储在起始地址为1000的连续内存单元中，则元素A[5][6]对应的地址为_____。

答：1095



1000 + 19 * 5 = 1095

填空题（5）

算术表达式(x+y)/10+a*b/c-(x-y)*(a-b)转为后缀表达式后为_____。

答：xy+10/ab*c/+xy-ab-*-

(x+y)/10+a*b/c-(x-y)*(a-b)

填空题（5）

$$(x+y)/10+a*b/c-(x-y)*(a-b)$$

扫描元素	栈状态	输出队列	说明
(	(	空	(压栈
x	(	x	x 入队
+	(+	x	+ 压栈
y	(+	xy	y入队
)	空	xy+	+入队，(弹出丢弃
/	/	xy+	/压栈
10	/	xy+10	10入队
+	+	xy+10/	栈顶/优先级高于+，弹出/入队，再压+
…	…	…	…

填空题（6）

具有4个结点的不同二叉树形态共有_____棵。

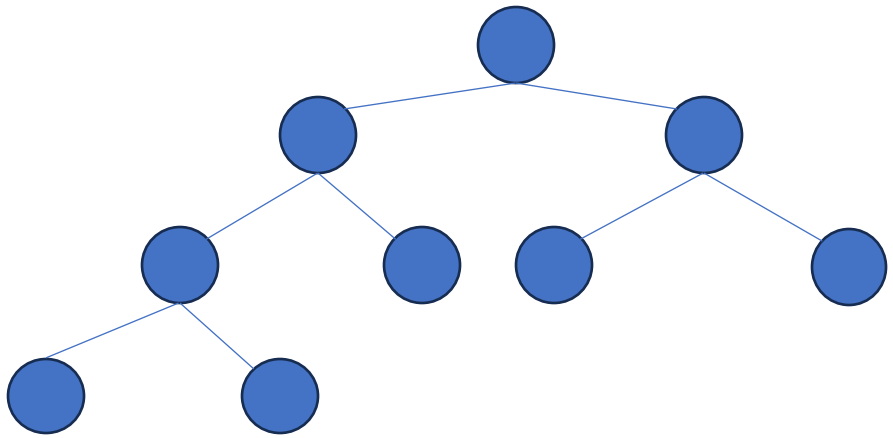
答：14或者是14*4！= 336

卡特兰数公式：$C_n = \frac{1}{n+1}\frac{(2n)!}{n!\cdot n!}$

填空题（7）

完全二叉树的第6层（根为第1层）有8个叶结点，则该树最多有_____个结点。

答：$2^6 - 1 + (2^5 - 8) * 2 = 63 + 24 * 2 = 63 + 48 = 111$



填空题（8）

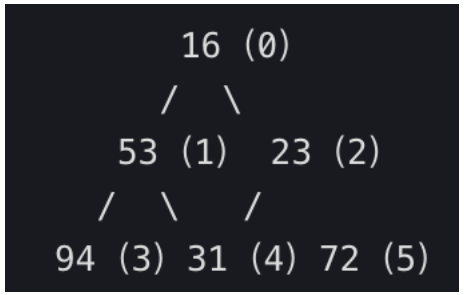
树的后根次序遍历和它通过左孩子-右兄弟方法转换的二叉树的_____遍历是等价的。

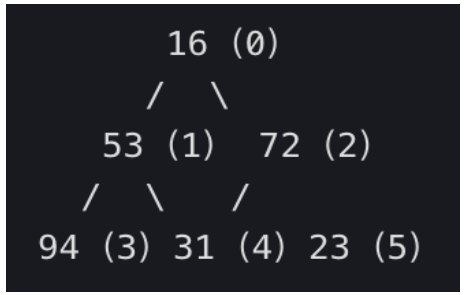
答：中序

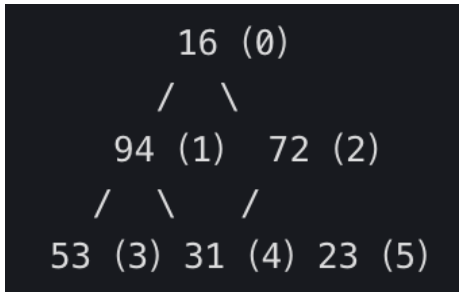
填空题（9）

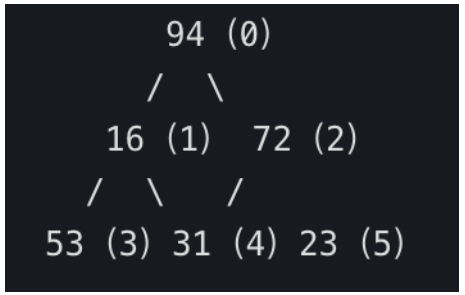
将关键码序列{16，53，23，94，31，72}转换为最大堆，所形成的二叉树最后一个叶结点的值为_____。

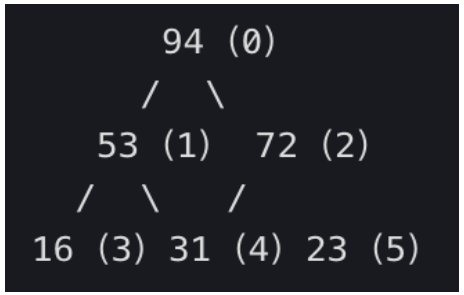
答：23











填空题（10）

已知一棵树以左孩子-右兄弟链表的形式存储，其结点结构说明如下：

```
struct node {
int data;
struct node *firstchild;
struct node *nextsibling;
      };
```

请在下面的_____处进行填空。注意：只能填一个语句，多填为0分。

```
//求出以T为根的树结点个数
int size(struct node *T) {
      if (T == NULL)
       return 0;
      else
       ____________________
}
```

答：`return (1+size(T->firstchild)+size(T-> nextsibling));`

问答题 1、分析以下函数的算法时间复杂度公式表达（规定m>k>1），并说明推算依据。

```
void f(int a[ ], int m, int k) {
  for (int i=m; i>=k; i--) {
      a[k-1]=i;
      if (k>1)  f(a, i-1,k-1);
      else  print(a);
  }
}
```

答：就是m个数中选k个数的组合，时间复杂度为O（C_{m-1}^{k-1}）（即 O（(m-1)!/[(k-1)!·(m-k)!]））。

问答题 2、设目标串T=“xxyxxxyxxxxyxyx”，模式串P=“xxyxy”。如何用最少的比较次数找到P在T中第一次出现的位置？相应的比较次数是多少？请给出具体的分析过程。

答：穷举模式匹配算法的时间复杂度为O(m*n)（其中，m为模式串长度，n为目标串长度）。KMP算法有一定改进，时间复杂度达到O(m+n)。本题也可采用从后面匹配的方法，即从右向左扫描，比较6次成功。另一种匹配方式是从左往右扫描，但是先比较模式串的最后一个字符，若不等，则模式串后移；若相等，再比较模式串的第一个字符，若第一个字符也相等，则从模式串的第二个字符开始向右比较，直至相等或者失败。若失败，模式串后移，再重复以上过程。按这种方法，本题比较18次成功。

问答题 3、 请分析并回答以下问题：

(1) 在一棵二叉树的前序遍历序列、中序遍历序列、后序遍历序列和层次序遍历序列中，任意两种序列的组合可以唯一地确定这棵二叉树吗？若不可以，有哪些组合。请说明理由。

(2) 已知一个森林的先根次序遍历序列和后根次序遍历序列分别为 ABCDEFGHIJKLMNO 和 CDEBFHIJGAMLONK， 请构造出该森林。

(1) **可以唯一确定**一棵二叉树形态的序列组合：

- 前序遍历序列+**中序**遍历序列
- 后序遍历序列+**中序**遍历序列
- 层次序遍历序列+**中序**遍历序列

中序遍历按照**左子树、根节点、右子树**的顺序遍历二叉树。因此，中序遍历可以帮助我们在树中找出每个结点的左右子树的结构。 中序遍历提供了关于树结点之间相对顺序的关键信息，并且能够明确划分每个结点的左右子树。

不可以唯一确定一棵二叉树的形态的序列组合：

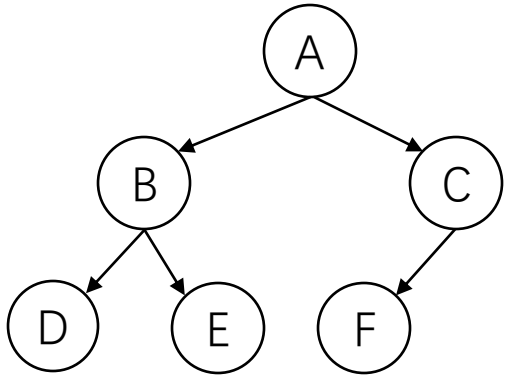
- 前序遍历序列+后序遍历序列
- 层次序遍历序列+前序遍历序列
- 层次序遍历序列+后序遍历序列

前序遍历 + 中序遍历

- 前序遍历的顺序是根结点 → 左子树 → 右子树，我们可以**从前序遍历中直接获取根结点**。
- 结合中序遍历，我们知道该根结点在中序遍历中的位置，进而可以通过**分割中序遍历得到左子树和右子树的结点**。
- 递归构建路径：一旦知道根节点，剩下的树就是左右子树的问题，而中序遍历清晰地标明了左子树和右子树的节点。

中序遍历：D B E **A** F C
前序遍历：**A** B D E C F

步骤 1：前序遍历的第一个节点是 A，所以 A 是根节点。
步骤 2：在中序遍历中，A 的左子树是 {D, B, E}，右子树是 {F, C}。
步骤 3：在前序遍历中，接下来的节点 B 是左子树的根节点，C 是右子树的根节点，进一步递归下去，直到整个树的结构完整。

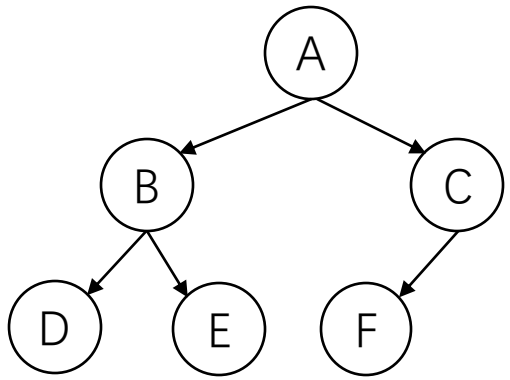


后序遍历 + 中序遍历

•后序遍历的顺序是左子树 -> 右子树 -> 根节点，这意味着我们可以通过后序遍历的最后一个节点来确定根节点。
•结合中序遍历，我们同样可以通过根节点在中序遍历中的位置来分割树，进一步递归地确定左右子树。

中序遍历：D B E A F C
后序遍历：D E B F C A

步骤 1：后序遍历的最后一个节点是 A，所以 A 是根节点。
步骤 2：在中序遍历中，A 的左子树是 {D, B, E}，右子树是 {F, C}。
步骤 3：在后序遍历中，接下来的节点 B 是左子树的根节点，C 是右子树的根节点。递归下去，最终还原出整棵树。

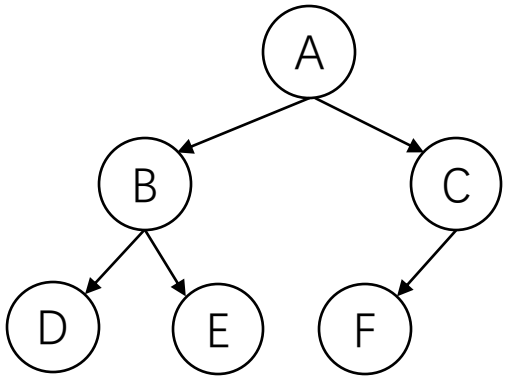


层次遍历 + 中序遍历

•层次遍历是从上到下、从左到右的顺序依次遍历节点，反映了节点的父子关系。通过层次遍历，**我们能够得知树的层级顺序以及每个节点的父节点**。
•结合中序遍历，能够帮助我们划分左右子树，进而确定树的结构。

中序遍历：D B E A F C
层次遍历：A B C D E F

步骤 1：层次遍历的第一个节点是 A，所以 A 是根节点。
步骤 2：根据中序遍历，A 的左子树是 {D, B, E}，右子树是 {F, C}。
步骤 3：层次遍历中，B 和 C 是 A 的子节点，D 和 E 是 B 的子节点，F 是 C 的子节点，最终确定整个树的结构。

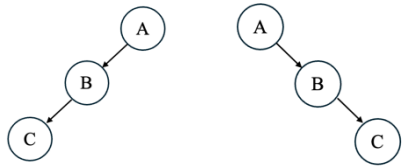


结合中序遍历和其他任意一种遍历（前序、后序或层次遍历）都能唯一确定一棵二叉树，原因在于：
•中序遍历能够帮助我们清晰地知道每个节点的左右子树。
•其他遍历（前序、后序、层次遍历）能够提供足够的父子关系和层级信息，帮助我们进一步确定树的结构。

前序遍历和后序遍历组合

前序遍历可以给出根节点，但后序遍历没有明确的区分左右子树的界限，因此，可能会存在不同的二叉树结构。

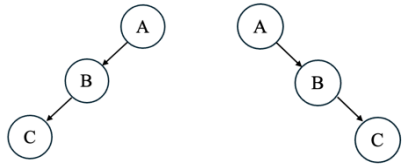
前序遍历：A B C
后序遍历：C B A



前序遍历和层次遍历组合

虽然前序遍历能给出根节点的位置，但层次遍历提供的信息过于宽泛，可能导致不同的二叉树结构。层次遍历可能不能唯一地决定左右子树的划分。

前序遍历：A B C
层次遍历：A B C

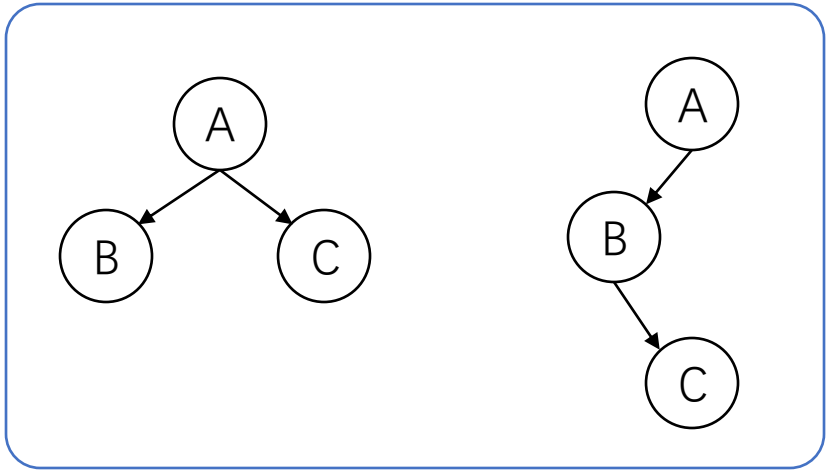


后序遍历和层次遍历组合

后序遍历和层次遍历都没有足够的信息来区分左右子树的位置。

后序遍历：B C A
层次遍历：A B C

可以构造两棵不同的二叉树



(2) 已知一个森林的先根次序遍历序列和后根次序遍历序列分别为ABCDEFGHIJKLMNO 和CDEBFHIJGAMLONK， 请构造出该森林

PPT第五章 树与二叉树

树 → 二叉树
左子结点 → 左子结点
兄弟结点 → 右子结点

5、子女-兄弟表示

- 也称为树的二叉树表示。每个结点的构造为：

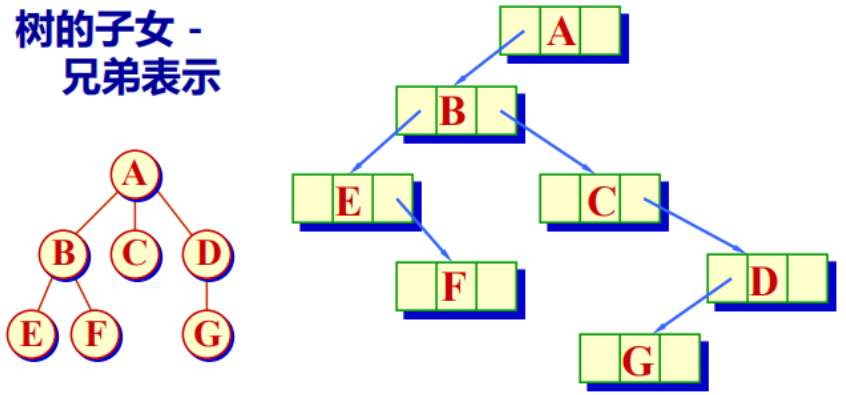
data	firstChild	nextSibling

- firstChild 指向该结点的第一个子女结点。无序树时，可任意指定一个结点为第一个子女。
- nextSibling 指向该结点的下一个兄弟。任一结点在存储时总是有顺序的。
- 若想找某结点的所有子女，可先找firstChild，再反复用 nextSibling 沿兄弟链访问。

159

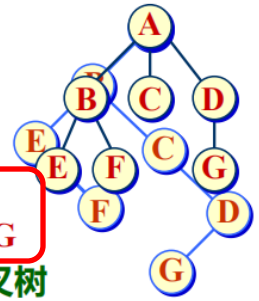
data	firstChild	nextSibling

树的子女 - 兄弟表示



树的先根次序遍历

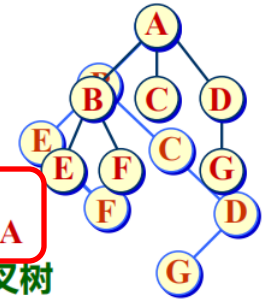
- 当树非空时
  - 访问根结点
  - 依次先根遍历根的各棵子树
- 树先根遍历 ABEFCDG
- 对应二叉树前序遍历 ABEFCDG
- 树的先根遍历结果与其对应二叉树表示的前序遍历结果相同
- 树的先根遍历可以借助对应二叉树的前序遍历算法实现



186

树的后根次序遍历

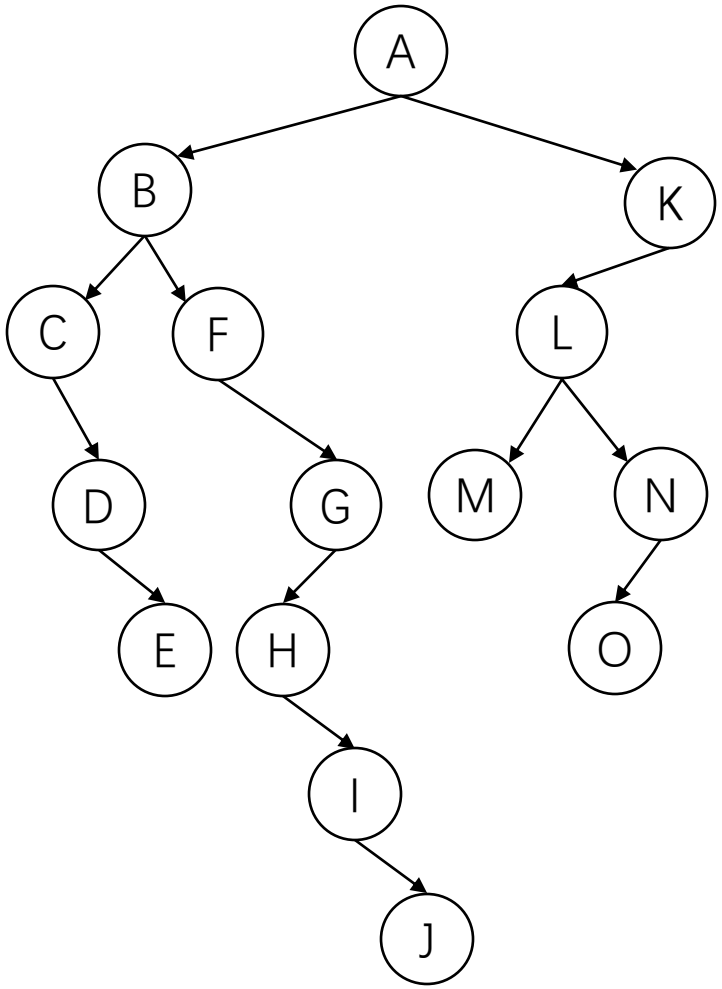
- 当树非空时
  - 依次后根遍历根的各棵子树
  - 访问根结点
- 树后根遍历 EFBCGDA
- 对应二叉树中序遍历 EFBCGDA
- 树的后根遍历结果与其对应二叉树表示的中序遍历结果相同
- 树的后根遍历可以借助对应二叉树的中序遍历算法实现

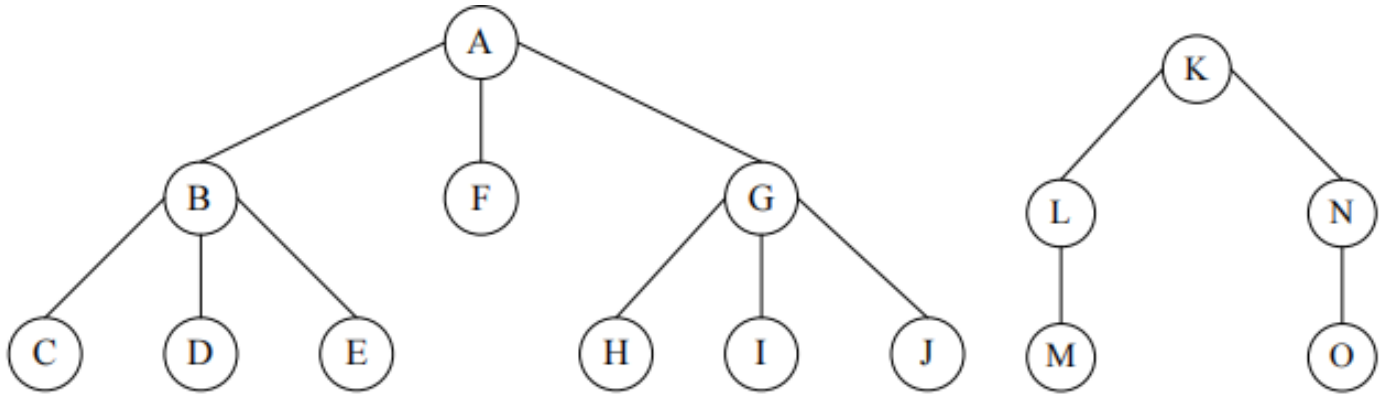


187

二叉树的先序遍历和中序遍历分别为 ABCDEFGHIJKLMNO 和 CDEBFHIJGAMLONK

ABCDEFGHIJKLMNO
CDEBFHIJGAMLONK

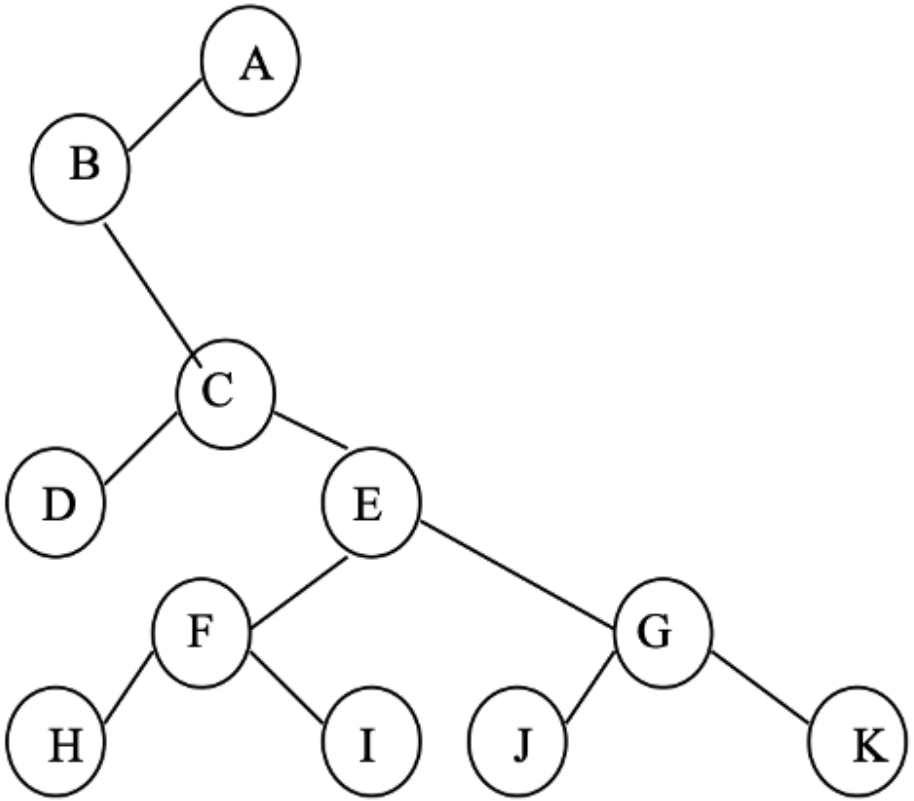




问答题 4、已知二叉树如下所示：

（1）用有向弧表示前驱和后继线索，画出树上的中序线索；

（2）在树上删除结点 E，画出改变后的中序线索化二叉树及其中的线索。



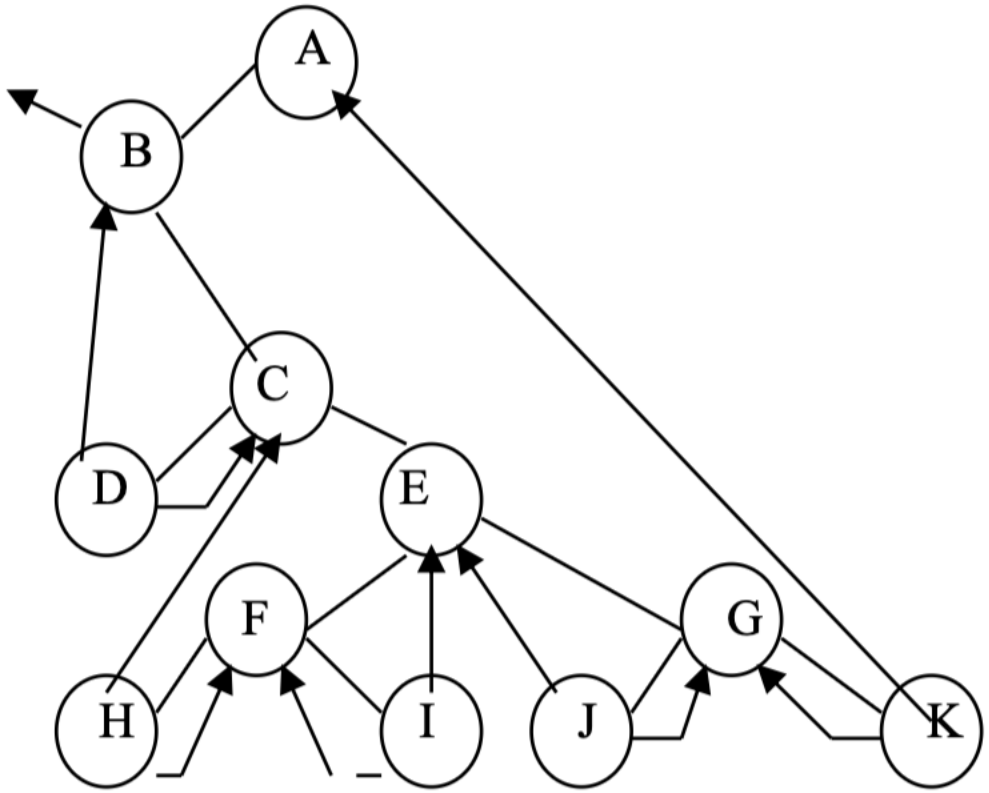
问答题 4、已知二叉树如下所示：

（1）用有向弧表示前驱和后继线索，画出树上的中序线索；

（2）在树上删除结点 E，画出改变后的中序线索化二叉树及其中的线索。

参考答案

(1)



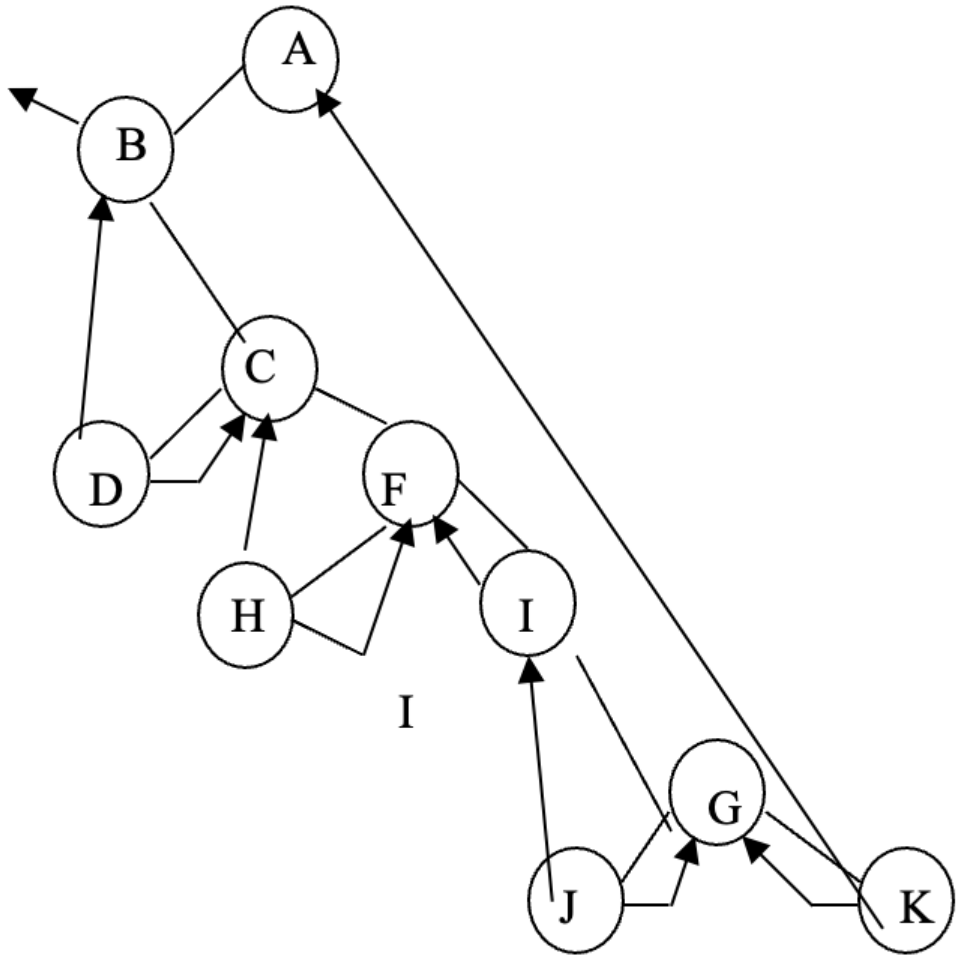
问答题 4、已知二叉树如下所示：

（1）用有向弧表示前驱和后继线索，画出树上的中序线索；

（2）在树上删除结点 E，画出改变后的中序线索化二叉树及其中的线索。

参考答案

(2)



问答题5 用于通信的电文由8个字符组成{a, b, c, d, e, f, g, h}，字符在电文中出现的频率分别为6, 18, 1, 5, 31, 2, 20, 9。试为这8个字符设计哈夫曼编码。使用0~7的3位等长二进制表示形式是另一种编码方案，对于上述实例比较两种方案的优缺点。

1, 2, 5, 6, 9, 18, 20, 31

1, 2, 5, 6, 9, 18, 20, 31

3, 5, 6, 9, 18, 20, 31

8, 6, 9, 18, 20, 31

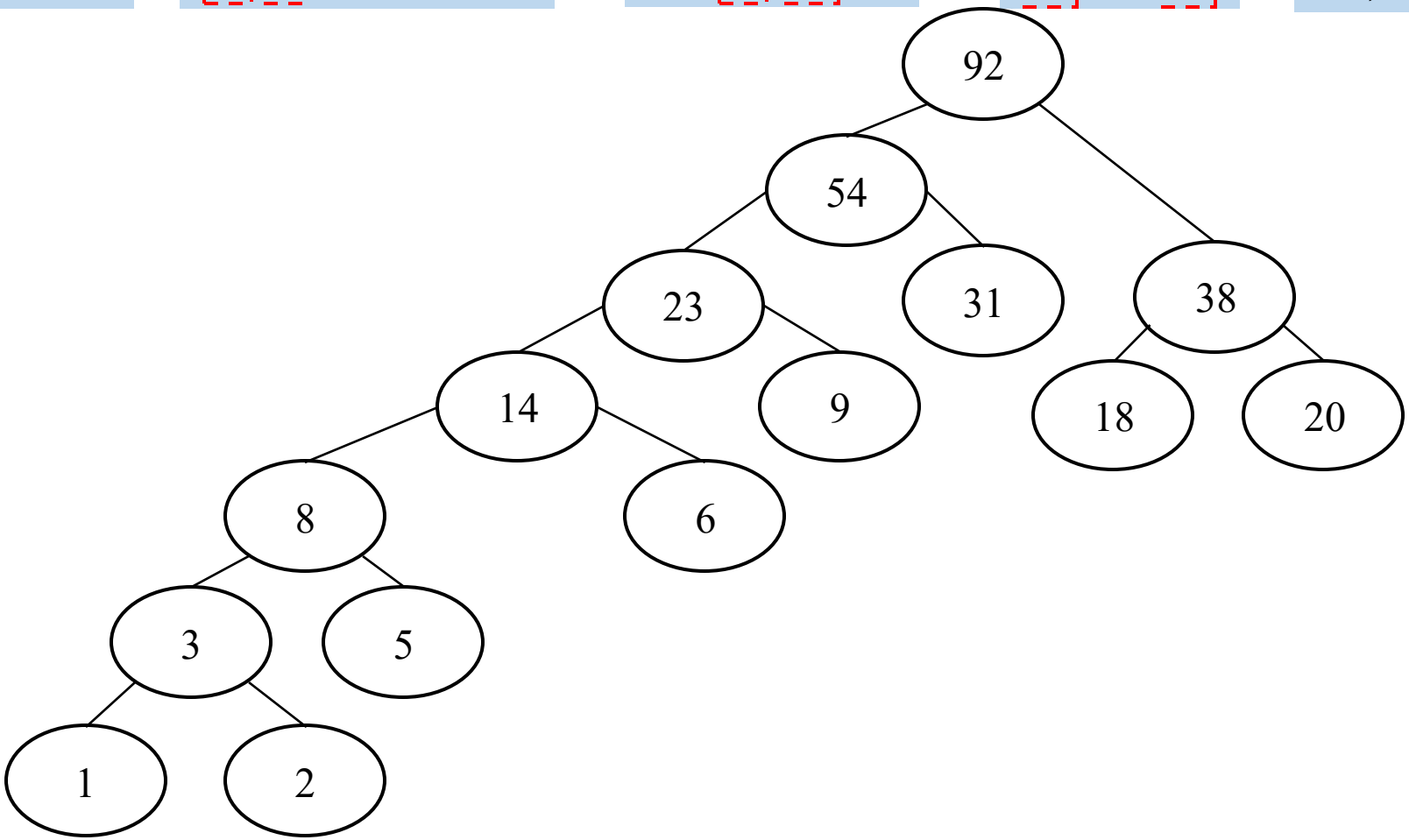
14, 9, 18, 20, 31

23, 18, 20, 31

23, 38, 31

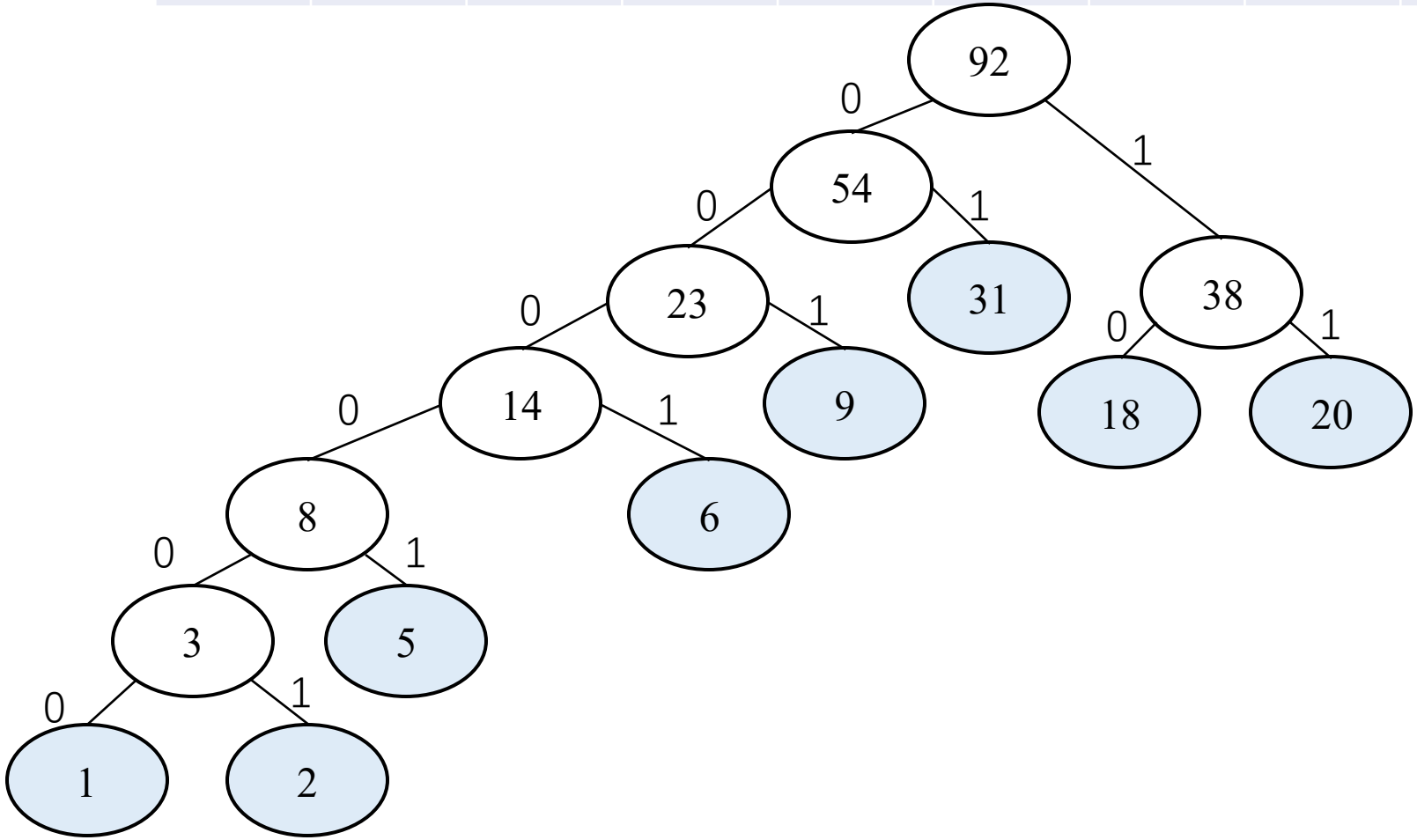
38, 54

92



问答题5 用于通信的电文由8个字符组成{a, b, c, d, e, f, g, h}，字符在电文中出现的频率分别为6, 18, 1, 5, 31, 2, 20, 9。试为这8个字符设计哈夫曼编码。使用0~7的3位等长二进制表示形式是另一种编码方案，对于上述实例比较两种方案的优缺点。

频数	6	18	1	5	31	2	20	9
哈夫曼	0001	10	000000	00001	01	000001	11	001
等长码	000	001	010	011	100	101	110	111



- 哈夫曼编码方案的带权路径长度为：
- (1+ 2) * 6 + 5 * 5 + 6 * 4 + 9 * 3 + (18+ 20 + 31)*2 =232

- 等长二进制编码方案的带权路径长度为：
- (6+18+1+5+31+2+20+9)*3 = 276

问答题5 用于通信的电文由8个字符组成{a, b, c, d, e, f, g, h}，字符在电文中出现的频率分别为6, 18, 1, 5, 31, 2, 20, 9。试为这8个字符设计哈夫曼编码。使用0~7的3位等长二进制表示形式是另一种编码方案，对于上述实例比较两种方案的优缺点。

频数	6	18	1	5	31	2	20	9
哈夫曼	0001	10	000000	00001	01	000001	11	001
等长码	000	001	010	011	100	101	110	111

Huffman

优点

•平均码长更短（本例约 2.52 vs 3），压缩更好，接近最优前缀码。
•高频字符码字很短，效率高。

缺点

•需要保存/约定码表（或重建树），实现复杂度更高。
•变长码不便于随机访问、同步；一处比特错误可能导致后续解码“跑偏”。

- 哈夫曼编码方案的带权路径长度为：
- (1+ 2) * 6 + 5 * 5 + 6 * 4 + 9 * 3 + (18+ 20 + 31)*2 =232

- 等长二进制编码方案的带权路径长度为：
- (6+18+1+5+31+2+20+9)*3 = 276

3 位等长编码

优点

•编码/解码极其简单，固定 3 位直接查表。
•易同步、易随机访问；错误一般局部影响（不会像变长码那样严重失步）。

缺点

•不利用频率差异，压缩效果差

算法题1、写一个算法，对输入的十进制非负整数num，将它的R (1<R<10)进制表示打印出来。要求算法中用到的栈采用单链表存储结构，单链表的结点类型为Node，其中，data域存放数据，link为指向后继结点的指针域。

基本思想：

采用辗转相除法，将num对应的R进制的各位数字进栈（采用头插法），然后依次出栈打印。

```
struct Node {
    Node* link;
    int data;
}
```

```
void Convert(int num, int R)
{
        Node *head=new Node;  head->link = NULL;
        Node *p;
        while (num > 0) {
            p=new Node;
            p->data=num % R;
            p->link =head->link;
            head->link =p;
            num=num/R;
        }  //结束while
        while (head->link) {
            p=head->link;
            head->link =p->link;
            printf(p->data);
        }
}
```

算法题1、写一个算法，对输入的十进制非负整数num，将它的R (1<R<10)进制表示打印出来。要求算法中用到的栈采用单链表存储结构，单链表的结点类型为Node，其中，data域存放数据，link为指向后继结点的指针域。

错误类型：

（1）采用尾插法

```
void Convert(int num, int R)
{
    Node *head = new Node;
    head->link = NULL;

    Node *p;
    Node *tail = head;

    while (num > 0) {
        p = new Node;
        p->data = num % R;
        p->link = NULL;          // 尾插：新结点的 next 置空

        tail->link = p;          // 接到尾部
        tail = p;                // 更新尾指针

        num = num / R;
    }
    while (head->link) {
        p = head->link;
        head->link = p->link;
        printf("%d", p->data);
    }
}
```

```
void Convert(int num, int R)
{
    Node *head = new Node;
    head->link = NULL;

    Node *p;
    Node *tail = head;

    while (num > 0) {
        p = new Node;
        p->data = num % R;
        p->link = NULL;          // 尾插：新结点的 next 置空

        tail->link = p;          // 接到尾部
        tail = p;                // 更新尾指针

        num = num / R;
    }
    reverse(head)                // 反转链表
    while (head->link) {
        p = head->link;
        head->link = p->link;
        printf("%d", p->data);
    }
}
```

算法题1、写一个算法，对输入的十进制非负整数num，将它的R (1<R<10)进制表示打印出来。要求算法中用到的栈采用单链表存储结构，单链表的结点类型为Node，其中，data域存放数据，link为指向后继结点的指针域。

错误类型：

（2）直接使用栈来做

```
void Convert(int num, int R)
{
    stack<int> s;
    while (num) {
        s.push(num % R);
        num /= 10;
    }
    while (!s.empty()) {
        printf("%d", s.top());
        s.pop();
    }
}
```

算法题2、已知n个结点的完全二叉树以二叉树的数组存储方式存储在数组a中，p、q为二叉树的两个不同的结点在a中的下标，设计函数“int nearestAncestor(int *a, int p, int q)”返回这两个结点的最近公共祖先结点在a中的下标。（注：最近公共祖先结点是p和q公共祖先中层数最大的结点）。（15分）

解题思路：

- 以完全二叉树形式保存二叉树，求某结点（数组编号为i）的祖先，如果此结点的序号i，则其双亲结点为i/2。
- 求祖先结点可以通过分别从p，q开始向上逐层找双亲结点，如果当前的两个双亲结点不同，则从层次较大的结点开始继续找上一层的双亲，第一个相同的双亲就是最近公共祖先。

算法题2、已知n个结点的完全二叉树以二叉树的数组存储方式存储在数组a中，p、q为二叉树的两个不同的结点在a中的下标，设计函数“int nearestAncestor(int *a, int p, int q)”返回这两个结点的最近公共祖先结点在a中的下标。（注：最近公共祖先结点是p和q公共祖先中层数最大的结点）。（15分）

参考答案

```
int nearestAncestor(int *a, int p, int q) {
    int tp = p, tq = q;
    while (tp!=tq)
        if (tp > tq)    tp = tp/2;
        else tq = tq/2;
    return p;
}
```

算法题3 给定指向二叉树根结点的指针root，二叉树为二叉链表存储方式（每个结点中有left和right两个指针指向左右孩子，整型的data属性中保存结点关键码），对于包含n个结点的二叉树，假设其中各个结点的关键码互相不重复，写出**递归和非递归**两种算法找到结点关键码的最大值，试对两种算法的**性能进行比较分析**。（15分）

```
#include <iostream>
#include <stack>
#include <climits>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
};

/********************* 递归*************************/
int maxValueRecursive(Node* root) {
    if (!root) return INT_MIN;
    int leftMax  = maxValueRecursive(root->left);
    int rightMax = maxValueRecursive(root->right);
    return max(root->data, max(leftMax, rightMax));
}
```

```
/********************非递归*******************/
int maxValueIterative(Node* root) {
    if (!root) return INT_MIN;

    int ans = INT_MIN;
    stack<Node*> st;
    st.push(root);

    while (!st.empty()) {
        Node* cur = st.top();
        st.pop();

        ans = max(ans, cur->data);

        if (cur->right) st.push(cur->right);
        if (cur->left)  st.push(cur->left);
    }
    return ans;
}
```

根节点入栈

弹出栈顶节点
（当前处理的节点）

更新最大值

子节点入栈

算法题3 给定指向二叉树根结点的指针root，二叉树为二叉链表存储方式（每个结点中有left和right两个指针指向左右孩子，整型的data属性中保存结点关键码），对于包含n个结点的二叉树，假设其中各个结点的关键码互相不重复，写出**递归和非递归**两种算法找到结点关键码的最大值，试对两种算法的**性能进行比较分析**。（15分）

典型错误

```
// 错误写法：max是值传递
int maxValueRecursive(Node* root, int max) {
    if (!root) return max; // 空节点返回当前max

    // 步骤1：更新当前层的max（仅修改局部副本）
    max = std::max(max, root->data);

    // 步骤2：递归遍历左右子树
    max = maxValueRecursive(root->left, max);  // 传值给左子树
    max = maxValueRecursive(root->right, max); // 传值给右子树

    return max;
}
```

```
void maxValueRecursive(Node* root, int& max) { // 无需返回值，直接修改引用
    if (!root) return;

    // 步骤1：更新全局max（引用指向外层变量，所有层级共享）
    if (root->data > max) {
        max = root->data;
    }

    // 步骤2：递归遍历左右子树（修改的是同一个max）
    maxValueRecursive(root->left, max);
    maxValueRecursive(root->right, max);
}
```