数据结构与算法

---

1 链表

1.1 链表逆序

思路：第一次循环，当前结点为p1结点，要实现逆序，即当前结点的next指针应当指向其前结点，先用next保存其后结点p2，再令p1的next指针指向prev（此时prev为空），之后当前结点移到p2结点，其前结点变为p1，故先用prev保存p1结点。重复以上过程，直到每个结点的next指针都指向它的前结点。

```
typedef struct Node{
    int data;
    struct Node *next;
} *Link;

Link revList(Link head)
{
    Link prev = NULL, next = NULL;  //当前结点的前/后一个结点
    Link cur;   //当前节点
    if(!head || !head->next) return head;
    cur = head->next;
    while(cur != NULL){
        //next结点存储当前结点的下一结点
        next = cur->next;
        //将当前结点的next指针指向prev结点
        cur->next = prev;
        //prev结点存储当前结点
        prev = cur;
        //当前结点后移一个单位
        cur = next;
    }
    //此时prev存储未逆序前的最后一个结点，逆序后变为首元结点
    head->next = prev;
    return head;
}
```

1.2 链表排序

插入的过程中实现排序。

```c
typedef struct Node{
    int data;
    struct Node *next;
}Node;
/* 从小到大排序 */
void sortInsert(Node *head, int data){
    Node *tmp = head;  //头结点
    Node *cur = (Node *)malloc(sizeof(Node));  //要插入的结点
    if(!cur) return;
    cur->data = data;
    cur->next = NULL;
    if(!head->next){
        head->next = cur;
        return;
    }
    while(tmp->next){
        if(tmp->next->data > tmp->data){
            cur->next = tmp->next;
            tmp->next = cur;
            return;
        }
        tmp = tmp->next;
    }
    tmp->next = cur;
}
```

1.3 链表的优缺点

3 二叉树

3.1 结点数

问题：将一颗有111个结点的完全二叉树从根这一层开始，每一层从左到右依次对结点进行编号，根结点编号为1，则编号最大的非叶子结点编号为?

解答：由完全二叉树性质可知该树有7层，前六层为满二叉树，结点数为 63 个。第7层结点数为 111 - 63 = 48 个。故第六层中非叶节点个数为 48 / 2 = 24 个。前五层的结点数为 31 个，故最大非叶结点编号为 31 + 24 = 55 。

3.2 遍历

问题：若一个二叉树的前序遍历结果是abefcgd，下面哪个不可能是它的中序遍历：

A. ebfagcd B. ebafgcd C.bfaegcd D. aebfgcd。

解答：由前序遍历结果可知此二叉树的根为 a ，逐一对选项进行分析，A选项的中序遍历可知 ebf 为左子树，gcd 为右子树，结合前序遍历结果可知左子树的根结点为 b ，右子树的根结点为 c 。故A选项成立。同理B选项成立。而C选项中左子树为 bf 。结合前序遍历可知左子树不可能为 bf 。故答案为C选项。

注意：由前序遍历和中序遍历、中序遍历和后序遍历可以唯一确定一棵树，而由前序遍历和后序遍历不能唯一确定。前序遍历的第一个结点即为根结点，后序遍历的最后一个结点即为根结点。

4 算法

4.1 计算一个字节中置1的位数

```c
int comb(const char c){
    int count = 0;
    int i;
    for(i = 0; i < 8; i++){
        if((c & 1) == 1) count++;
        c >>= 1;
    }
    return count;
}
```

4.2 判断处理器是大端还是小端

大端模式，是指数据的高字节保存在内存的低地址中，而数据的低字节保存在内存的高地址中。小端则正好相反。

方法一：通过指针地址判断

```c
int check(){
    int num = 0x12345678;
    //(char*)&num获得num的低8位地址
    return (*((char *)&num) == 0x12);
}
//大端返回1，小端返回0
```

方法二：通过联合体判断

联合体union的存放顺序是所有成员都从低地址开始存放。c.a = 1，即0x00000001，若处理器为大端，则低字节0x01存放在高地址，故b的值为0，若处理器为小端，则低字节0x01存放在低地址，故b的值为1。

```c
int check(){
    union w{
        int a;
        char b;
    }c;
    c.a = 1;
    return (c.b == 1);
}
//大端返回0，小端返回1
```

4.3 分解质因数

质因数（质因子）是指能整除给定正整数的质数。例如6的质因数是3和2。把一个合数分解成若干个质因数的乘积的形式，即求质因数的过程叫做**分解质因数**。分解质因数的方法为：

- 如果这个质数恰等于n，则说明分解质因数的过程已经结束，打印出即可。
- 如果n>k，但n能被k整除，则应打印出k的值，并用n除以k的商作为新的正整数n，重复执行第一步。
- 如果n不能被k整除，则用k+1作为k的值，重复执行第一步。

```
void prim(int m){
    int n = 2;
    if(m >= n){
        while(m % n) n++;
        m /= n;
        prim(m, n)
    }
}
```