真题总结

23软件专硕

辗转相除法，也称欧几里得算法，是一种高效的求最大公约数（Greatest Common Divisor，简称GCD）的算法。它的基本原理是：两个正整数a和b（假设a>b），它们的最大公约数等于a除以b的余数c和b之间的最大公约数。

具体步骤如下：

1. 设定两个正整数a和b，且a>b。
2. 将a除以b，得到余数c。
3. 将b赋值给a，将c赋值给b。
4. 重复步骤2和3，直到b为0。
5. 当b为0时，a的值即为这两个数的最大公约数。

以下是辗转相除法的C语言实现：

```c
#include <stdio.h>

// 函数声明
int gcd(int a, int b);

int main() {
    int num1, num2, result;

    // 用户输入两个正整数
    printf("Enter two positive integers: ");
    scanf("%d %d", &num1, &num2);

    // 调用函数计算最大公约数
    result = gcd(num1, num2);

    // 输出结果
    printf("GCD of %d and %d is %d\n", num1, num2, result);

    return 0;
}

// 辗转相除法函数定义
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

2019软件专硕

输出无向无权连通图以v为顶点的最短路径长度为k的所有结点

```
int visited[N] = {0};        //判断是否访问过
int isout[N] = {0};          //判断是否输出

void dfsfindallnode(int a[N][N], int i, int k){
    if(k == 0){              //函数的终止条件
        if(isout[i] == 0){
            printf("可以到达节点%d\n",i);
            isout[i] = 1;
            return;          //结束当前层递归
        }
    }
    for(int j = 0;  j < N; j++){
        if(a[i][j] == 1 && visited[j]==0){//如果可以到达这个点的话
            visited[j]=1;
            dfsfindallnode(a,j,k-1);
            visited[j]=0;                //回溯还原
        }
    }
}
```

2016软件专硕

带头结点的双链表l，每个节点有4个数据成员，前驱节点llink，后继节点rlink，数据成员data，访问频度freq，且已知双向链表L中节点一直按访问频度递减的顺序排列且频繁访问的节点一直靠近表头，初始状态l中的所有节点freq的节点都为0，对双链表的locate操作，每操作一次，将数据值x的节点访问频度+1，设计一个算法，对双链表l的locate操作，要求操作后L中的节点仍然按照频度的递减顺序排列
思路：

- 先找到结点值为x的节点，频度加1，然后断链
  - 如果是该节点前一个节点是头结点，不用操作
- 然后找到需要插入的节点，分情况讨论
  - 插在最后一个，只需接三条链
  - 不是插在最后一个，需要接四条链

```
typedef struct dnode{
    int data;
    int freq;
    struct dnode *llink, *rlink;
}*dlist;

void Locate(dlist head, int x){
    dlist p = head->rlink,q;
    while(p != NULL && p->data != x) p = p->rlink;          //找到节点值为x的节点
    p->freq++;                                              //访问频度加1
    if(p!=NULL){                                            //找到节点
        p->freq++;                                          //频度加1
        if(p->llink == head) return;                        //第一个节点，不用调整
        q = p;                                              //保存需要操作的节点

        p->llink->rlink = p->rlink;                         //断开p左右节点的链并接上
        if(p->rlink!=NULL)                                  //右链得保证存在才能接
            p->rlink->llink = p->llink;
```

```
        p = q->llink;                                           //从q的前一个结点找到q的
freq大的节点
        while(p != head && p->freq < q->freq) p = p->llink; //找到需要插入的节点
        if(p->rlink==NULL){                                     //q插到p节点前 ,注意这里
的插法
            q->llink = p;
            q->rlink = p->rlink;
            p->rlink = q;
        }
        else{                                                   //不是插到最后一个节点
            p->rlink->llink = q;
            q->llink = p;
            q->rlink = p->rlink;
            p->rlink = q;
        }
    }
}
```