线性表

1.顺序表的结构定义

```
typedef struct
{
    int data[maxSize];
    int length;
}Sqlist;
```

简单形式：

```
int A[maxSize];
int n;
```

2.单链表结点形式

```
typedef struct LNode
{
    int data;
    struct LNode *next;
}LNode;
```

3.双链表结点形式

```
typedef struct DLNode
{
    int data;
    struct DLNode *prior;
    struct DLNode *next;
}DLNode;
```

4.顺序表的查找

```
int LocateElem(Sqlist L, int e)
{
    int i;
    for(i = 1; i <= L.length; ++i)
        if(e == L.data[i])
            return i;
    return 0;
}
```

5.顺序表的插入

```
int insert(Sqlist &L, int p, int e)
{
    int i;
    if(p < 1 || p > L.length + 1 || L.length == maxSize - 1)
        return 0;
    for(i = L.length; i >= p; --i)
        L.data[i+1] = L.data[i];
    L.data[p] = e;
    ++(L.length);
    return 1;
}
```

6.顺序表的删除

```
int delete(Sqlist &L, int p, int &e)
{
    inti ;
    if(p < 1 || p > L.length)  return 0;
    e = L.data[p];
    for(i = p; i < L.length; ++i)
        L.data[i] = L.data[i+1];
    --(L.length);
    return 1;
}
```

7.单链表的尾插法

```
void CreatelistR(LNode *&C, int a[], int n)
{
    LNode *s, *r;
    int i;
    C = (LNode *)malloc(sizeof(LNode));
    C->next = NULL;
    r = C;
    for(i = 1; i <= n; ++i)
    {
        s = (LNode *)malloc(sizeof(LNode));
        s->data = a[i];
        r->next = s;
        r = r->next;
    }
    r->next = NULL;
}
```

8.单链表的头插法

```
void CreatelistF(LNode *&C, int a[], int n)
{
    LNode *s;
    int i;
    C = (LNode *)malloc(sizeof(LNode));
    C->next = NULL;
    for(i = 1; i <= n; ++i)
    {
        s = (LNode *)malloc(sizeof(LNode));
        s->data = a[i];
        s->next = C->next;
        C->next = s;
    }
}
```

9.单链表的删除

```
q = p->next;
p->next = p->next->next;
free(q);
```

栈

声明：

```
int stack[maxSize];
int top = -1;
```

进栈：

```
stack[++top] = x;
```

出栈：

```
x = stack[top--];
```

队列

声明：

```
int data[maxSize];
int front, rear;
```

队空：`rear == front;`

队满：`(rear + 1) % maxSize == front;`

进队：

```
rear = (rear + 1) % maxSize;
data[rear] = x;
```

出队：

```
front = (front + 1) % maxSize;
x = data[front];
```

【**注**：进队、出队中两个语句的顺序依据题目而定。】

二叉树

1.链式存储结构

```
typedef struct BTNode
{
    char data;
    struct BTNode *lchild;
    struct BTNode *rchild;
}
```

2.先序遍历

```
void preorder(BTNode *p)
{
    if(p != NULL)
    {
        visit(p);
        preorder(p->lchild);
        preorder(p->rchild);
    }
}
```

3.中序遍历

```
void inorder(BTNode *p)
{
    if(p != NULL)
    {
        inorder(p->lchild);
        visit(p);
        inorder(p->rchild);
    }
}
```

4.后序遍历

```
void postorder(BTNode *p)
{
    if(p != NULL)
    {
        postorder(p->lchild);
        postorder(p->rchild);
        visit(p);
    }
}
```

非遍历

```
void postorder(BTNode *T)
{
    BTNode *S[maxSize];            //栈
    int top = -1;
    BTNode *p = T;
    BTNode *r = NULL;
    while(p || top != -1)
    {
        if(p)
        {
            S[++top] = p;
            p = p->lchild;
        }
        else
        {
            p = S[top--];
            if(p->rchild && p->rchild != r)
            {
                p = p->rchild;
                S[++top] = p;
                p = p->lchild;
            }
            else
            {
                p = S[--top];
                visit(p);
                r = p;
                p = NULL;
            }
        }
    }
}
```

5.层次遍历

```
void level(BTNode *p)
{
    int front, rear;
    BTNode *que[maxSize];          //定义一个循环队列，
                                   //用来记录将要访问的层次上的结点
    front = 0;
    rear = 0;
    BTNode *q;
    if(p != NULL)
    {
        rear = (rear + 1) % maxSize;
        que[rear] = p;                 //根结点入队
        while(front != rear)
        {
               front = (front + 1) % maxSize;
               q = que[front];         //队结点出队
               visit(q);
               if(q->lchild != NULL)
               {
                    rear = (rear + 1) % maxSize;
                    que[rear] = q->lchild;
               }
               if(q->rchild != NULL)
               {
                    rear = (rear + 1) % maxSize;
                    que[rear] = q -> rchild;
               }
        }
    }
}
```

图

1.邻接矩阵的结构型定义

```
typedef struct
{
    int no;
    char info;
}VertexType;                        //顶点类型

typedef struct
{
    int edges[maxSize][maxSize];
    int n,e;                        //分别为顶点数和边数
    VertexType vex[maxSize];        //存放结点信息
}MGraph;                            //图的邻接矩阵类型
```

2.邻接表的结构型定义

```
typedef struct ArcNode
{
    int adjvex;                     //该边所指向的结点的位置
    struct ArcNode *nextarc;        //指向下一条边的指针
}ArcNode;                           //边的类型

typedef struct VNode
{
    char data;
    ArcNode *firstarc;              //指向第一条边的指针
}VNode;                             //点的类型
typedef struct
{
    VNode adjlist[maxSize];         //邻接表
    int n,e;                        //顶点数和边数
}AGraph;                            //图的邻接表类型
```

3.邻接表 深度优先搜索遍历

```
int visit[maxSize];

void DFS(AGraph *G, int v)
{
    ArcNode *p;
    visit[v] = 1;
    访问顶点 v;
    p = G->adjlist[v].firstarc;
    while(p != NULL)
    {
        if(visit[p->adjvex] == 0)
            DFS(G,p->adjvex);
        p = p->nextarc;
    }
}

void dfs(AGraph *g)
{
    int i;
    for(i = 1; i <= g->n; ++i)
        if(visit[i] == 0)
            DFS(g,i);
}
```

4.邻接表 广度优先搜索遍历

```
void BFS(AGraph *G, int v, int visit[maxSize])
{
    ArcNode *p;
    int que[maxSize], front = 0, rear = 0;    //队列
    int j;
    访问结点 v;
    visit[v] = 1;
    rear = (rear + 1) % maxSize;
    que[rear] = v;
    while(front != rear)
    {
        front = (front + 1) % maxSize;
        j = que[front];
        g = G->adjlist[j].firstarc;
        while(p != NULL)
        {
            if(visit[p->adjvex] == 0)
            {
                访问 p->adjvex;
                visit[p->adjvex] = 1;
                rear = (rear + 1) % maxSize;
                que[rear] = p->adjvex;
            }
            p = p->nextarc;
        }
    }
}

void bfs(AGraph *g)
{
    int i;
    for(i = 1; i <= g->n; ++i)
        if(visit[i] == 0)
            BFS(g, i, visit);
}
```

PV 操作

1. 生产者-消费者问题

```
full = 0; empty = n; mutex = 1;
Producer()
{
    while(true)
    {
        生产;
        P(empty);
        P(mutex);
        放入缓冲池;
        V(mutex);
        V(full);
    }
}
Consumer()
{
    while(true)
    {
        P(full);
        P(mutex);
        取出产品;
        V(mutex);
        V(empty);
        消费;
    }
}
```

2.读者-写者问题

(1)读者优先

```
rmutex = 1; mutex = 1;
int readcount = 0;
reader()
{
    while(true)
    {
        P(rmutex);
        if(readcount == 0)
            P(mutex);
        readcount++;
        V(rmutex);
        进行读操作;
        P(rmutex);
        readcount--;
        if(readcount == 0)
            V(mutex);
        V(rmutex);
    }
}
writer()
{
    while(true)
    {
        P(mutex);
        进行写操作;
        V(mutex);
    }
}
```

(2)公平情况

```
mutex = 1; rmutex = 1; wmutex = 1;
int readcount = 0;
reader()
{
    while(true)
    {
        P(wmutex);
        P(rmutex);
        if(readcount == 0)
            P(mutex);
        readcount++;
        V(rmutex);
        V(wmutex);
        进行读操作;
        P(rmutex);
        readcount--;
        if(readcount == 0)
            V(mutex);
        V(rmutex);
    }
}
writer()
{
    while(true)
    {
        P(wmutex);
        P(mutex);
        进行写操作;
        V(mutex);
        V(wmutex);
    }
}
```

(3)写者优先

```
mutex = 1; rmutex = 1; wmutex = 1; readable = 1;
int readcount = 0, writecount = 0;
reader()
{
    P(readable);
    P(rmutex);
    if(readcount == 0)
        P(mutex);
    readcount++;
    V(rmutex);
    V(readable);
    读操作;
    P(rmutex);
    readcount--;
    if(readcount == 0)
        V(mutex);
    V(rmutex);
}
writer()
{
    P(wmutex);
    if(writecount == 0)
        P(readable);
    writecount++;
    V(wmutex);
    P(mutex);
    写操作;
    V(mutex);
    P(wmutex);
    writecount--;
    if(writecount == 0)
        V(readable);
    V(wmutex);
}
```

3.哲学家进餐问题

```
Fork[5] = {1, 1, 1, 1, 1};
philosoper(int i)
{
    while(true)
    {
        思考;
        想吃饭;
        if(i % 2 != 0)
        {
            P(Fork[i]);
            P(Fork[(i + 1) % 5]);
            进餐;
            V(Fork[i]);
            V(Fork[(i + 1) % 5]);
        }
        else
        {
            P(Fork[(i + 1) % 5]);
            P(Fork[i]);
            进餐;
            V(Fork[(i + 1) % 5]);
            V(Fork[i]);
        }
    }
}
```

4.理发师问题

```
int chairs = n + 1;
ready = 0; finish = 1; mutex = 1;
barber()
{
    while(true)
    {
        P(ready);
        理发;
        P(mutex);
        chairs++;
        V(mutex);
        V(finish);
    }
}
customer()
{
    P(mutex);
    if(chairs > 0)
    {
        chairs--;
        V(mutex);
        V(ready);
        P(finish);
    }
    else
        V(mutex);
}
```