队列

2019年6月25日 20:05

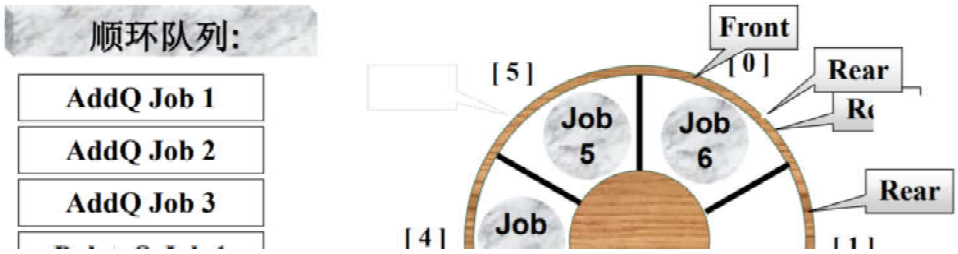
队列的定义：具有一定操作约束的线性表
插入和删除操作：只能在一端插入，而在另一端删除。
· 数据插入：入队列（AddQ）
· 数据删除：出队列（DeleteQ）
· 先来先服务 · 先进先出：FIFO

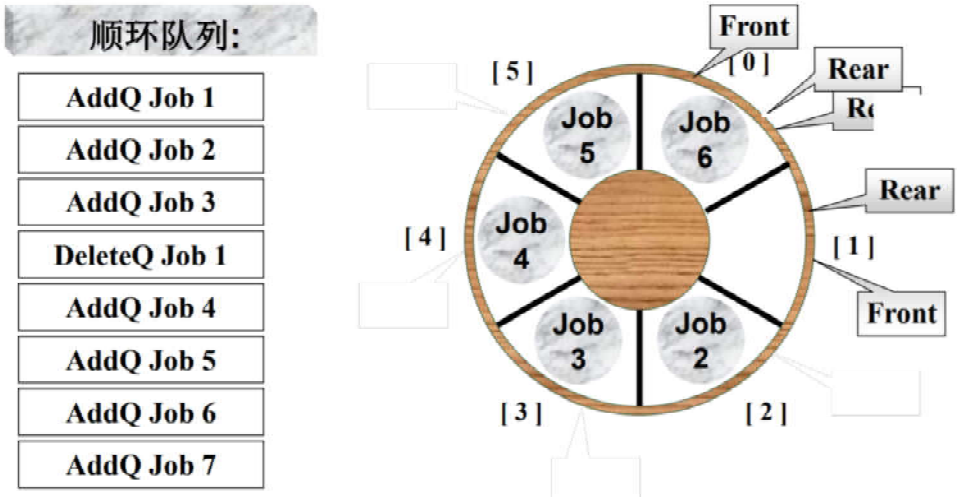
队列的抽象数据类型描述
类型名称：队列(Queue)
数据对象集：一个有0个或多个元素的有穷线性表。
操作集：长度为MaxSize的队列Q · Queue，队列元素item · ElementType
1、Queue CreatQueue(int MaxSize)：生成长度为MaxSize的空队列；
2、int IsFullQ(Queue Q, int MaxSize)：判断队列Q是否已满；
3、void AddQ(Queue Q, ElementType item)： 将数据元素item插入队列Q中；
4、int IsEmptyQ(Queue Q)： 判断队列Q是否为空；
5、ElementType DeleteQ(Queue Q)：将队头数据元素从队列中删除并返回。

队列的顺序存储实现
队列的顺序存储结构通常由一个一维数组和一个记录队列头元 素位置的变量front以及一个记录队列尾元素位置的变量rear组成。
判断为空：front==rear

```
#define MaxSize <储存数据元素的最大个数>
struct QNode {
        ElementType Data[MaxSize];
        int rear;
        int front;
};
typedef struct QNode* Queue;
```

循环队列





front==rear是空还是满?

为什么无法区分:

（1）判断空满是根据front和rear的相对关系：有六种情况：0、1、2、3、4、5 n

（2）而队列有7种情况：0（null）、1、2、3、4、5、6 n+1

解决方案:

（1）额外标记：size当前元素个数，tag当前操作（删除0 插入1）

（2）仅使用n-1个数组的空间

基本操作:

（1）入队列（rear向前移动：求余（rear+1）%size实现循环序号）

```
void AddQ(Queue PtrQ, ElementType item)
{
        if ((PtrQ->rear + 1) % MaxSize == PtrQ->front) {
                printf(“队列满”);
                return;
        }
        PtrQ->rear = (PtrQ->rear + 1) % MaxSize;
        PtrQ->Data[PtrQ->rear] = item;
}
```

（2）出队列（front向前移动：求余（front+1）%size实现循环序号）

```
ElementType DeleteQ(Queue PtrQ)
{
        if (PtrQ->front == PtrQ->rear) {
                printf(“队列空”);
                return ERROR;
        }
        else {
                PtrQ->front = (PtrQ->front + 1) % MaxSize;
                return PtrQ->Data[PtrQ->front];
        }
}
```

队列的链式实现

队列的链式存储结构也可以用一个单链表实现。插入和删除操作 分别在链表的两头进行；队列指

针front和rear应该分别指向链表的哪一头?

front表头 rear表尾

```
void AddQ(Queue PtrQ, ElementType item)
{
	if ((PtrQ->rear + 1) % MaxSize == PtrQ->front) {
		printf("队列满");
		return;
	}
	PtrQ->rear = (PtrQ->rear + 1) % MaxSize;
	PtrQ->Data[PtrQ->rear] = item;
}
struct Node {
	ElementType Data;
	struct Node* Next;
};
struct QNode { /* 链队列结构 */
	struct Node* rear; /* 指向队尾结点 */
	struct Node* front; /* 指向队头结点 */
};
typedef struct QNode* Queue;
Queue PtrQ;
```

不带头结点的链式队列出队操作的一个示例

```
ElementType DeleteQ(Queue PtrQ)
{
	struct Node* FrontCell;
	ElementType FrontElem;
	if (PtrQ->front == NULL) {
		printf("队列空"); return ERROR;
	}
	FrontCell = PtrQ->front;
	if (PtrQ->front == PtrQ->rear) /* 若队列只有一个元素 */
		PtrQ->front = PtrQ->rear = NULL; /* 删除后队列置为空 */
	else
		PtrQ->front = PtrQ->front->Next;
	FrontElem = FrontCell->Data;
	free(FrontCell); /* 释放被删除结点空间 */
	return FrontElem;
}
```

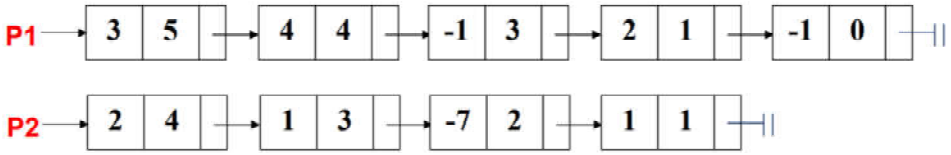
多项式加法运算

采用不带头结点的单向链表，按照指数递减的顺序排列各项

2.4 应用: 多项式加法运算

$$
\begin{aligned}
P1 &= 3X^5 + 4X^4 - X^3 + 2X - 1 \\
+\ P2 &= 2X^4 + X^3 - 7X^2 + X \\
\hline
P &= 3X^5 + 6X^4 - 7X^2 + 3X - 1
\end{aligned}
$$

主要思路：相同指数的项系数相加，其余部分进行拷贝。



算法思路：两个指针P1和P2分别指向这两个多项式第一个结点，不断循环：

- P1->expon==P2->expon: 系数相加，若结果不为0，则作为结果多项式对应项 的系数。同时，P1和P2都分别指向下一项；
- P1->expon>P2->expon: 将P1的当前项存入结果多项式，并使P1指向下一项；
- P1->exponexpon: 将P2的当前项存入结果多项式，并使P2指向下一项；

当某一多项式处理完时，将另一个多项式的所有结点依次复制到结果多项式中去。

结构：

```
struct PolyNode {
    int coef; // 系数
    int expon; // 指数
    struct PolyNode* link; // 指向下一个节点的指针
};
typedef struct PolyNode* Polynomial;
Polynomial P1, P2;
```

实现代码：

```
Polynomial PolyAdd(Polynomial P1, Polynomial P2)
{
    Polynomial front, rear, temp;
    int sum;
    rear = (Polynomial)malloc(sizeof(struct PolyNode));//为方便表头插入，先产生一 个临时空结点作为结果多项 式链表头,最后要释放
    front = rear; /* 由front 记录结果多项式链表头结点 */
    while (P1 && P2) /* 当两个多项式都有非零项待处理时 */
        switch (Compare(P1->expon, P2->expon)) {//比较p1 p2的大小
        case 1://p1大
            Attach(P1->coef, P1->expon, &rear);
            P1 = P1->link;
            break;
        case -1://p2大
            Attach(P2->coef, P2->expon, &rear);
            P2 = P2->link;
```

```
                break;
            case 0:
                sum = P1->coef + P2->coef;
                if (sum) Attach(sum, P1->expon, &rear);
                P1 = P1->link;
                P2 = P2->link;
                break;
            }
        /* 将未处理完的另一个多项式的所有节点依次复制到结果多项式中去 */
        for (; P1; P1 = P1->link) Attach(P1->coef, P1->expon, &rear);
        for (; P2; P2 = P2->link) Attach(P2->coef, P2->expon, &rear);
        rear->link = NULL;
        temp = front;
        front = front->link; /*令front指向结果多项式第一个非零项 */
        free(temp); /* 释放临时空表头结点 */
        return front;
}

void Attach(int c, int e, Polynomial* pRear)
{ /* 由于在本函数中需要改变当前结果表达式尾项指针的值， */
/* 所以函数传递进来的是结点指针的地址，*pRear指向尾项*/
        Polynomial P;
        P = (Polynomial)malloc(sizeof(struct PolyNode)); /* 申请新结点 */
        P->coef = c; /* 对新结点赋值 */
        P->expon = e;
        P->link = NULL;
        /* 将P指向的新结点插入到当前结果表达式尾项的后面 */
        (*pRear)->link = P;
```


```
        *pRear = P; /* 修改pRear值 */
```


```
}
```