线性表

2019年6月25日 18:17

线性表定义：由同类型的数据元素构成的有序的线性结构

长度：表中元素个数

空表：没有元素

表头、表尾

线性表的基本操作：

数据对象集：线性表是n个元素构成的有序序列

操作集：线性表L · List，整数i表示位置，元素X · ElementType，

线性表基本操作主要有：

1、List MakeEmpty()：初始化一个空线性表L；

2、ElementType FindKth(int K, List L)：根据位序K，返回相应元素；

3、int Find(ElementType X, List L)：在线性表L中查找X的第一次出现位置；

4、void Insert(ElementType X, int i, List L)：在位序i前插入一个新元素X；

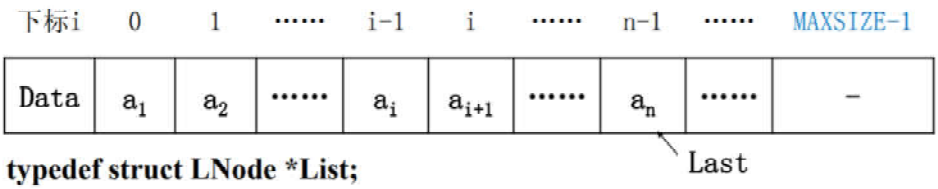
5、void Delete(int i, List L)：删除指定位序i的元素；

6、int Length(List L)：返回线性表L的长度n。

线性表的实现

（一）顺序存储

利用数组的连续存储空间顺序存放线性表的各元素




```
typedef struct LNode *List;
struct LNode{
        ElementType  Data[MAXSIZE];
        int  Last;
} ;
struct LNode L;
List  PtrL;
```

访问下标为 i 的元素：L.Data[i] 或 PtrL->Data[i]

线性表的长度：L.Last+1 或 PtrL->Last+1

主要操作实现：

1.初始化

```
list MakeEmpty()
{
    List PtrL;
    PtrL=(List)malloc(sizeof(struct LNode));
    PtrL->Last=-1;
    return PtrL;
}
```

2.查找

1. 初始化（建立空的顺序表）

```
List MakeEmpty( )
{    List PtrL;
     PtrL = (List )malloc( sizeof(struct LNode) );
     PtrL->Last = -1;
     return PtrL;
}
```

查找成功的平均比较次数为 (n +1)/2，平均时间性能为 O(n)。

2. 查找

```
int Find( ElementTy…, List PtrL )
{   int i = 0;
    while( i <= PtrL->Last && PtrL->Data[i]!= X )
        i++;
    if (i > PtrL->Last)  return -1; /* 如果没找到，返回-1 */
    else  return i;              /* 找到后返回的是存储位置 */
}
```

3.插入(第i个位置插入值为X元素)

平均移动次数：n/2

时间复杂度：O（n）

```
void Insert(ElementType X, int i, List PtrL)
{
        int j;
        if (PtrL->Last == MAXSIZE - 1) { /* 表空间已满，不能插入*/
                printf( " 表满 " );
                return;
        }
        if (i < 1 || i > PtrL->Last + 2) { /*检查插入位置的合法性*/
```

```
        printf(" 位置不合法 ");
        return;
    }
    for (j = PtrL->Last; j >= i - 1; j--)
        PtrL->Data[j + 1] = PtrL->Data[j]; /*将 ai ~ an倒序向后移动*/
    PtrL->Data[i - 1] = X; /*新元素插入*/
    PtrL->Last++; /*Last仍指向最后元素*/
    return;
}
```

4.删除

平均移动次数：（n-1）/2

时间复杂度：O（n）

```
void Delete(int i, List PtrL)
{
    int j;
    if (i < 1 || i > PtrL->Last + 1) { /*检查空表及删除位置的合法性*/
        printf("不存在第 % d个元素", i);
        return;
    }
    for (j = i; j <= PtrL->Last; j++)
        PtrL->Data[j - 1] = PtrL->Data[j]; /*将 ai+1 ~ an顺序向前移动*/
    PtrL->Last--; /*Last仍指向最后元素*/
    return;
}
```

（二）链式存储

不要求逻辑上相邻的两个元素物理上也相邻；通过“链”建 立起数据元素之间的逻辑关系。 • 插入、删除不需要移动数据元素，只需要修改“链” 。

创建链表：

```
typedef struct LNode* List;
struct LNode {
    ElementType Data;
    List Next;
};
struct Lnode L;
List PtrL;//表头
```

1.求表长（遍历）O（n）

```
int Length(List PtrL)
{
    List p = PtrL; /* p指向表的第一个结点*/
    int j = 0;
    while (p) {
        p = p->Next;
        j++; /* 当前p指向的是第 j 个结点*/
    }
    return j;
}
```

2.查找

（1）按序号：找到第k个

```
List FindKth(int K, List PtrL)
{
        List p = PtrL;
        int i = 1;
        while (p != NULL && i < K) {
                p = p->Next;
                i++;
        }
        if (i == K) return p;
        /* 找到第K个，返回指针 */
        else return NULL;
        /* 否则返回空 */
}
```

（2）按值查找

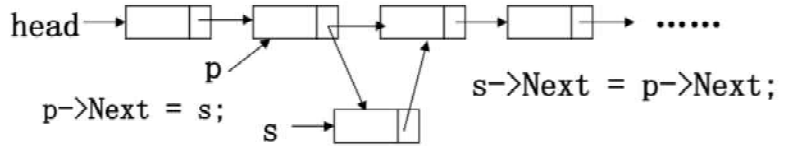
```
List Find(ElementType X, List PtrL)
{
        List p = PtrL;
        while (p != NULL && p->Data != X)
                p = p->Next;
        return p;
}
```

3.插入（在第 i-1(1≤i≤n+1)个结点后插入一个值为X的新结点)

（1）先构造一个新结点，用s指向；

（2）再找到链表的第 i-1个结点，用p指向；

（3）然后修改指针，插入结点（p之后插入新结点是 s)



思考：修改指针的两个步骤如果交换一下，将会发生什么？

代码：

```
List Insert(ElementType X, int i, List PtrL)
{
        List p, s;
        if (i == 1) { /* 新结点插入在表头 */
                s = (List)malloc(sizeof(struct LNode)); /*申请、填装结点*/
                s->Data = X;
                s->Next = PtrL;
                return s; /*返回新表头指针*/
        }
        p = FindKth(i - 1, PtrL); /* 查找第i-1个结点 */
        if (p == NULL) { /* 第i-1个不存在，不能插入 */
                printf(" 参数i错 " );
                return NULL;
        }
        else {
                s = (List)malloc(sizeof(struct LNode)); /*申请、填装结点*/
```

```
            s->Data = X;
            s->Next = p->Next; /*新结点插入在第i-1个结点的后面*/
            p->Next = s;
            return PtrL;
        }
}
```

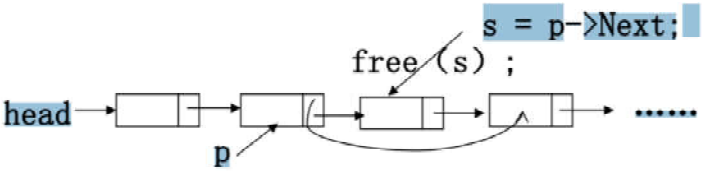
4.删除（删除链表的第 i (1≤i≤n)个位置上的结点)

（1）先找到链表的第 i-1个结点，用p指向；

（2）再用指针s指向要被删除的结点（p的下一个结点）；

（3）然后修改指针，删除s所指结点;

（4）最后释放s所指结点的空间。



p->Next = s->Next;

思考：操作指针的几个步骤如果随意改变，将会发生什么？

代码：

```
List Delete(int i, List PtrL)
{
        List p, s;
        if (i == 1) { /* 若要删除的是表的第一个结点 */
            s = PtrL; /*s指向第1个结点*/
            if (PtrL != NULL) PtrL = PtrL->Next; /*从链表中删除*/
            else return NULL;
            free(s); /*释放被删除结点 */
            return PtrL;
        }
        p = FindKth(i - 1, PtrL); /*查找第i-1个结点*/
        if (p == NULL) {
            printf("第 % d个结点不存在", i - 1); return NULL;
        }
        else if (p->Next == NULL) {
            printf("第 % d个结点不存在", i); return NULL;
        }
        else {
            s = p->Next; /*s指向第i个结点*/
            p->Next = s->Next; /*从链表中删除*/
            free(s); /*释放被删除结点 */
            return PtrL;
        }
}
```

（三）线性表的应用