

Lecture 4: 线性表的顺序实现

4.1 顺序表的定义

4.1.1 基本概念

概念：顺序表的定义

- 概念：
 - 顺序表的方式实现线性表顺序存储。
 - 把逻辑上相邻的元素存储在物理位置上也相邻的存储单元中，元素之间的关系由存储单元的邻接关系来体现
- 定义：
 - $LOC(L) + n * \text{数据元素的大小}$
- 注意
 - LOC (L)表示线性表第一个元素的存放位置 (Location)
 - N 表示第 n 个元素
- 补充：
 - C 语言中查找一个数据元素大小 `sizeof(ELemType)`

4.1.2 静态分配

代码：定义

```
#define MaxSize 10 //定义最大长度
typedef struct{
    ElemType data[MaxSize]; //用静态的“数组”存放数据元素
    int length; //顺序表的当前长度
}SqList; //顺序表的类型定义(静态分配方式)
```

代码：实现

```
#include <stdio.h>
#define MaxSize 10 //定义最大长度
typedef struct{
    int data[MaxSize]; //用静态的“数组”存放数据元素
    int length; //顺序表的当前长度
}SqList; //顺序表的类型定义(静态分配方式)
//基本操作——初始化一个顺序表
void InitList(SqList &L){
    for(int i=0;i<MaxSize;i++)
        L.data[i]=0; //将所有数据元素设置为默认初始值
    L.length=0; //顺序表初始长度为0
}
```

```
int main() {
    SqList L;    //声明一个顺序表
    InitList(L); //初始化顺序表
    //.....未完待续，后续操作
    return 0;
}
```

- 由于静态分配内存空间不变
- 数据存满，将会溢出
- 初始空间分配过多，会浪费内存空间
- 由于内存中会有遗留的脏数据，所以要设置默认初始值 (0)

4.1.3 动态分配

代码：顺序表的实现 - 动态分配

```
#define InitSize 10    //顺序表的初始长度
typedef struct{
    ElemType *data;    //指示动态分配数组的指针
    int MaxSize;       //顺序表的最大容量
    int length;        //顺序表的当前长度
}SeqList;             //顺序表的类型定义(动态分配方式)
```

- C: `malloc`, `free` 函数
 - `L.data = (ElemType*) malloc (sizeof (ElemType)\* InitSize)`
 - 注意: `malloc` 函数返回一个指针需要强制转型为你定义的数据元素类型指针
- C++: `new`, `delete` 关键字
 - `L.data = new ElemType[InitSize];`

代码

```
#include <stdlib.h> //标准库
#define InitSize 10 //默认的最大长度
typedef struct{
    int *data;    //指示动态分配数组的指针
    int MaxSize;  //顺序表的最大容量
    int length;   //顺序表的当前长度
}SeqList;

void InitList(SeqList &L){
    //用malloc函数申请一片连续的存储空间
    L.data=(int *)malloc(InitSize*sizeof(int));
    L.length=0;
    L.MaxSize=InitSize;
}

//增加动态数组的长度
void IncreaseSize(SeqList &L,int len){
    int *p=L.data;
```

```

    L.data=(int *)malloc((L.MaxSize+len)*sizeof(int));
    for(int i=0;i<L.length;i++){
        L.data[i]=p[i];        //将数据复制到新区域
    }
    L.MaxSize=L.MaxSize+len; //顺序表最大长度增减len
    free(p);                  //释放原来的内存空间
}
int main(){
    SeqList L;                //声明一个顺序表
    InitList(L);              //初始化顺序表
    //...往顺序表中随便插入几个元素...
    IncreaseSize(L,5);
    return 0;
}

```

概念：顺序表的特点

1. 随机访问， 即可以在 $O(1)$ 时间内找到第 i 个元素
2. 存储密度高， 每个节点只存储数据元素
3. 拓展容量不方便 (即使采用动态分配的方式， 拓展长度的时间复杂度也比较高)
4. 插入、删除操作不方便， 需要移动大量元素

4.2 顺序表的插入与删除

4.2.1 插入

概念： ListInsert(&L,i,e)

- 表示插入操作
- 在表 L 中的第 i 个位置上插入指定元素 e ;

代码：定义

```

#define MaxSize 10 //定义最大长度
typedef struct{
    ElemType data[MaxSize]; //用静态的“数组”存放数据元素
    int length;              //顺序表的当前长度
}SqlList;                  //顺序表的类型定义

```

代码：实现

```

//代码实现
#define MaxSize 10 //定义最大长度
typedef struct{
    int data[MaxSize]; //用静态的“数组”存放数据元素
    int length;         //顺序表的当前长度
}SqlList;              //顺序表的类型定义

bool ListInsert(SqlList &L,int i,int e){
    if(i<1||i>L.length+1) //判断i的范围是否有效

```

```

        return false;
    if(L.length>=MaxSize)        //当前存储空间已满，不能插入
        return false;
    for(int j=L.length;j>=i;j--) //将第i个元素及其之后的元素后移
        L.data[j]=L.data[j-1];
    L.data[i-1]=e;                //在位置i处放入e
    L.length++;                  //长度加1
    return true;
}

int main() {
    SqList L;        //声明一个顺序表
    InitList(L);     //初始化顺序表
    //...此处省略一些代码，插入几个元素
    ListInsert(L,3,3)
    return 0;
}

```

分析：时间复杂度分析

- 最好情况： $O(1)$
- 最坏情况： $O(n)$
- 平均情况：
 - 假设新元素插入到任何一个位置的概率相同，即 $i = 1, 2, 3, \dots, length + 1$ 的概率都是 $p = \frac{1}{n+1}$
 - $i = 1$ ，循环 n 次； $i = 2$ ，循环 $n - 1$ 次； $i = 3$ ，循环 $n - 2$ 次，...， $i = n + 1$ 时，循环 0 次
 - 平均循环次数 = $np + (n - 1)p + (n - 2)p + \dots + 1 \cdot p = n \frac{n+1}{2} \frac{1}{n+1} = \frac{n}{2}$

4.2.2 删除

概念：ListDelete(&L,i,&e)

- 删除操作：删除表 L 中第 i 个位置的元素；

代码：实现

```

bool ListDelete(SqList &L,int i,int &e){
    if(i<1||i>L.length)        //判断i的范围是否有效
        return false;
    e=L.data[i-1];                //将被删除的元素赋值给e
    for(int j=i;j<L.length;j++) //将第i个位置后的元素前移
        L.data[j-1]=L.data[j];
    L.length--;                  //线性表长度减1
    return true;
}

int main() {
    SqList L;        //声明一个顺序表
    InitList(L);     //初始化顺序表

```

```
//...此处省略一些代码，插入几个元素
int e = -1;    //把变量e把删除的元素“带回来”
if(ListDelete(L, 3, E))
    printf("已删除第3个元素，删除元素之值为%d\n", e);
else
    printf("位序i不合法，删除失败\n");
return 0;
}
```

- 平均循环次数 = $\frac{n-1}{2}$ $\rightarrow$ 平均复杂度: $O(n)$

获取笔记持续更新可加微信: WandD2277