2.1 线性表的定义和基本操作

线性表的定义

- 线性表是具有相同数据类型的 n(n≥0) 个数据元素的有限序列
- 特点
    - 表中元素个数有限
    - 表中元素具有逻辑上的顺序性，表中元素有其先后次序
    - 表中元素都是数据元素，每个元素都是单个元素
    - 表中元素的数据类型都相同，这意味着每个元素占有相同大小的存储空间
    - 表中元素具有抽象性

什么是线性表？🤔

有点像函数

函数：一个 x 值只能对应一个 y 值 线性表：一个索引值只能有一个映射值

线性表的基本操作

详细文档可以参考 c++容器list 的相关函数

c++代码示例

- 初始化 构造器 constructor
- 求表长 length()或 size()
- 清空 empty()
- 获取元素
- 插入
- 删除
- ...

2.2 线性表的顺序表示

顺序表的定义 Sequence List

- *线性表*的顺序存储 -> 顺序表
- 顺序表的特点是表中元素的逻辑顺序与其物理顺序相同
- 顺序表的结构类型定义
    - 静态分配
    - 动态分配
    - C 的动态分配语句
    - `L.data = (ElemType*)malloc(sizeof(ElemType)*InitSize;`
    - C++的动态分配语句
    - `L.data = new ElemType[InitSize];`

顺序表上基本操作的实现

- 插入操作
  - 最好情况：表尾插入，时间复杂度为$O(1)$
  - 最坏情况：表头插入，时间复杂度为$O(n)$
  - 平均情况：$E_{ins} = \frac 1{n+1} \sum^{n+1}_{i=1}(n-i+1) = \frac 1{n+1}(n+\cdots + 1 + 0) = \frac n2$ ，时间复杂度为$O(n)$

```
// 插入操作代码
bool ListInsert(SqList &L, int i, ElemType e) {
    if (i < 1 || i > L.length + 1 )          // i值不合法
        return false;
    else if (L.length >= MAXSIZE)            // 当前存储空间已满
        return false;
    for (int j = L.length ; j >= i ; j--)
        L.elem[j] = L.elem[j - 1];           // 插入位置及之后位置后移
    L.elem[i - 1] = e;                       // 将新元素放入第i个位置
    L.length++;                              //表长增加1
    return true;
}
```

- 删除操作
  - 最好情况：删除表尾元素，时间复杂度为$O(1)$
  - 最坏情况：删除表头元素，时间复杂度为$O(n)$
  - 平均情况：$E_{del} = \frac 1n \sum^n_{i=1}(n-i)=\frac 1n \frac {(n-1)n}2 = \frac{n-1}2$ ，时间复杂度为$O(n)$

```
// 删除操作代码
bool ListDelete(SqList& L, int i, ElemType e) {
    if (i < 1 || i > L.length)             // 判断i值是否合理
        return false;
    e = L.data[i - 1];
    for (int j = i; j < L.length; j++)
        L.elem[j - 1] = L.elem[j];
    L.length--;
    return true;
}
```

- 按值查找（顺序查找）
  - 最好情况：查找的元素就在表头，时间复杂度为$O(1)$
  - 最坏情况：查找的元素在表尾（或不存在），时间复杂度为$O(n)$
  - 平均情况：$E_{find} = \frac 1n \sum^n_{i=1}i= \frac{n+1}2$ ，时间复杂度为$O(n)$

```
// 按值查找（顺序查找）代码
int LocateElem(SqList &L, ElemType e){
    int i;
    for(i = 0; i < L.length; i++){
        // TODO
```

```
    }
}
```

习题

2.3 线性表的链式表示

单链表的定义

- 线性表的链式存储又称单链表
- 结点的类型定义

```
typedef struct LNode{     // 声明结点的类型和指向结点的指针类型
  ElemType data;         //结点的数据域
  struct LNode *next;    // 结点的指针域
}LNode,*LinkList;          // LinkList为指向结构体LNode的指针类型
```

- 通常用头指针来表示一个单链表
- 单链表上基本操作的实现
- 采用头插法建立单链表
    - 时间复杂度$O(n)$

```
void CreateList_Head(LinkList &L,int n){
    L = new LNode;
    L->next = NULL; //L指向头结点，头结点指向NULL指针
  for(int i = 0;i < n; i++){
    p = new LNode;
    cin >> p->data;
    p->next = L->next;
      L->next = p;
  }
}
```

- 采用尾插法建立单链表
    - 时间复杂度$O(n)$

```
void CreateList_Tail(LinkList& L, int n) {
  L = new LNode;
  L->next = NULL;
  LNode* r;
  LNode* p;
  r = new LNode; r = L;
```

```
  for (int i = 0; i < n; i++) {
    p = new LNode; p->next = NULL;
    cin >> p->data;
    r->next = p;
    r = p;
  }
}
```

- 按序号查找结点值
    - 时间复杂度$O(n)$

```
Status GetElem_L(LinkList L,int i, ElemType &e){
  LNode* p = L->next;
  int j = 1;
  if(i == 0) return L;        // 若i等于0，则返回头结点
  if(i < 1) return NULL;      // 若i无效，则返回头结点
  while(p && j < i){
    p = p->next;
    j++;
  }
  e = p->data;
  return OK;
}
```

- 按值查找表结点
    - 时间复杂度$O(n)$

```
int LocateElem_L_return_num(LinkList L,ElemType e){
  LNode* p = L->next;
  int j = 1;
    while(p && p->data != e){
      p = p->next;
      j++;
    }
    if(p) return j;
    else return 0;
}
```

- 插入结点操作
    - 时间复杂度$O(n)$，开销主要在查找第i-1个元素

```
Status ListInsert_L(LinkList& L, int i, ElemType e) {
  LNode* p = L;
  LNode* s;
  int j = 0;
```

```
  while (p && j < i - 1) {
    p = p->next;
    j++;
  }
  if (p || j > i - 1) return ERROR;
  s = new LNode;
  s->data = e;
  s->next = p->next;
  p->next = s;
  return OK;
}
```

- 删除结点操作
    - 时间复杂度$O(n)$

```
bool ListDelete_L(LinkList& L, int i, ElemType &e) {
  LNode* p = L;
  LNode* q = new LNode;
  int j = 0;
  while (p->next && j < i - 1) {
    p = p->next;
    j++;
  }
  if (!(p->next) || j > i - 1) return false;

    q->next = p->next;
  e = q->data;
    delete q;
  return true;
}
```

- 求表长操作

双链表

- 结点类型定义

```
typedef struct DuLNode{
  Elemtype data;
  struct DuLNode *prior,*next;
}DuLNode, *DuLinkList;
```

- 双链表的插入操作
    - 时间复杂度$O(n)$

```
int ListInsert_DuL(DuLinkList &L, int i, ElemType e) {
    DuLNode* p, *s;
    //在带头结点的双向循环链表L中第i个位置之前插入元素e
    if (!(p = GetElemP_DuL(L, i))) return ERROR;

    s = new DuLNode;
    s->data = e;

    s->prior = p->prior;
    p->prior->next = s;
    s->next = p;
    p->prior = s;
    return OK;
}// ListInsert_DuL
```

- 双链表的删除操作
    - 时间复杂度$O(n)$

```
int ListDelete_DuL(DuLinkList& L, int i, ElemType &e) {
    DuLNode* p;
    //在带头结点的双向循环链表L中删除第i个元素，并返回e
    if (!(p = GetElemP_DuL(L, i))) return ERROR;
    e = p->data;
    p->prior->next = p->next;
    p->next->prior = p->prior;
    delete p;
    return OK;
}//ListDelete_DuL
```

- 循环链表
    - 循环单链表
    - 循环双链表
- 静态链表
    - 结点类型定义

```
#define MaxSize 100
typydef struct{
    ElemType data;
    int next;
}SlinkList[MaxSize];
```

- 顺序表和链表的比较
    - 存储（读写）方式

    - 逻辑结构与物理结构
    - 查找、插入和删除操作
    - 空间分配
    - 在实际应用中应该怎样选取存储结构呢？
        - 基于存储的考虑
        - 基于运算的考虑
        - 基于环境的考虑

习题

- 7 给定有n个元素的一维数组，建立一个有序单链表的最低时间复杂度是?→$$O(n\log_2n)$$ 先建立再排序，时间复杂度是$O(n^2)$； 先排序再建立，排序的时间复杂度是$O(n\log_2n)$，建立的时间复杂度是$O(n)$
- 15【2016】
- 21 一个链表最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素，选用？最节省时间 A 不带头结点的单循环链表 B 双链表 C 不带头结点且有尾指针的单循环链表 D 单链表→C
- 23【2016】
- 25 某线性表用带头结点的循环单链表存储，头指针为head，当head->next->next=head 成立时，线性表长度可能是？→0或者1
- 综合题21【2009】已知一个嗲有表头结点的单链表，结点结构为data|link 假设该链表只给出了头指针list。在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第k个位置上的结点（k为正整数）。若查找成功，算法输出该结点的data域的值，并返回1；否则，只返回0。要求： ① 描述算法的基本设计思想 ② 描述算法的详细实现步骤 ③ 根据设计思想和实现步骤，采用程序设计语言描述算法，关键之处请给出简要注释
- 22【2012】
- 23【2015】
- 25【2019】