第九章 线性结构

数据结构概述

概念和术语

- **数据结构**：数据对象中数据元素之间的结构关系
- **数据**：计算机能识别、存储和处理的符号的集合
- **数据元素**：数据的基本单位，由一个或多个数据项组成。又称为结点、顶点、记录、表目等
- **数据项**：具有独立含义的数据的最小单位，又称为域、字段
- **数据对象**：具有相同性质的数据元素集合

例如：职工情况表
职工情况表是一种数据结构，每个职工情况为一数据元素，职工的每项基本信息为一个数据项，部门的所有职工构成一个数据对象

主要研究内容

- 1.数据的逻辑结构：
  四种基本结构——集合结构、线性结构(一一对应)、树形结构(层次结构)、图形结构(网状结构)
- 2.数据的存储结构：　(是逻辑结构在计算机内存储器中的实现)
- 顺序方式：存将数据元素按照某种顺序放到**一片连续的存储单元**内，逻辑关系由存储器中的**相对位置**来体现　(优点：存储密度高、快速访问；缺点：插入或删除时效率低，存储空间需预先分配)
  链接方式：元素间的逻辑关系由附加的指针域表示（如：单链表）　(优点：插入/删除元素时，不会引起大量元素的移动；可动态申请和释放结点空间；缺点：不能实现随机访问，指针域会耗费一些存储空间)
  索引方式：关键字后放地址，地址被存到索引表中（优点：检索速度快；缺点：增加的索引表会占存储空间，插入/删除元素要修改索引表)
  散列方式：数据元素的存储位置是用它的关键字计算出来的（如：哈希表、杂凑表等）(优点：快；缺点：如果散列函数不好，可能会出现存储单元的冲突)
- 3.数据的运算
  运算：对数据对象中的数据所进行的加工和处理
  常用的运算：插入、删除、查找、排序、更新
  特点：数据运算定义在逻辑结构上，实现在存储结构上、实现用算法描述

算法

算法时间复杂度

一般把程序运行时语句执行的次数（不包括说明语句）作为估算程序执行时间的量度

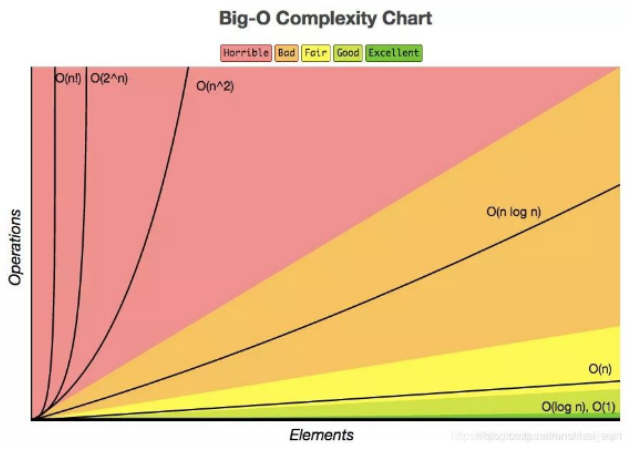
常见的时间复杂度如图所示：

执行次数函数	阶	非正式术语
12	O(1)	常数阶
2n+3	O(*n*)	线性阶
$3n^2+2n+1$	$O(n^2)$	平方阶
$5\log_2 n+20$	O(log*n*)	对数阶
$2n+3n\log_2 n+19$	O(*n*log*n*)	nlogn 阶
$6n^3+2n^2+3n+4$	$O(n^3)$	立方阶
2^n	$O(2^n)$	指数阶

所耗时间从小到大依次是：

$$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)<O(2^n)<O(n!)<O(n^n)$$

我们可以画一个函数图像清晰的看每个复杂度的时间对比：



线性表

定义

线性表是由n(n ⩾ 0)个相同类型的数据元素e0、e1 ... en-1组成的有限序列。
线性表中元素的个数称为表的 **长度** ，长度为0的表为空表，元素在表中的序号称作元素的下标

常用的基本运算

建空表、求表长、查找、插入、删除、排序等

线性表的顺序存储结构（顺序表）

- 以顺序方式存储的线性表称为顺序表
- 可以用 **一维数组** 实现顺序表

```
char table[26];
//或
char *table = new char[26];
```

字符型顺序表类的定义：

```
#include <iostream.h>

class SeqList{
public:
        SeqList(int m = 10);        //构造函数
        ~SeqList(){delete[]element;}   //析构函数
        bool IsEmpty(){return length==0;}   //判断是否为空
        int Length(return length;)   //返回表长
        bool Find(int i,char &x);   //把下标为i的元素取至x
        int Search(const char &x);   //返回x在表中的下标
        bool Insert(int i,const char &x);   //在下标i处插入元素x
        bool Delete(int i,char &x); //返回下标为i的元素至x，并删除之
        void ClearList(){length=0;}   //将表清空
        void output(ostream &out)const;   //输出表中所有元素的值
        friend ostream& operator<<(ostream &out,const SeqList &x); //重载"
<<"
private:
        int Maxsize;     //线性表的容量
        int length;      //线性表的长度
        char *element;

}

SeqList::SeqList(int m){
        elemet = new char[m];
        Maxsize=m;
        length=0;
}

bool SeqList::Find(int i,char &x){
        if(i<0||i>length-1) return false;
        x=element[i];
        return true;
}
```

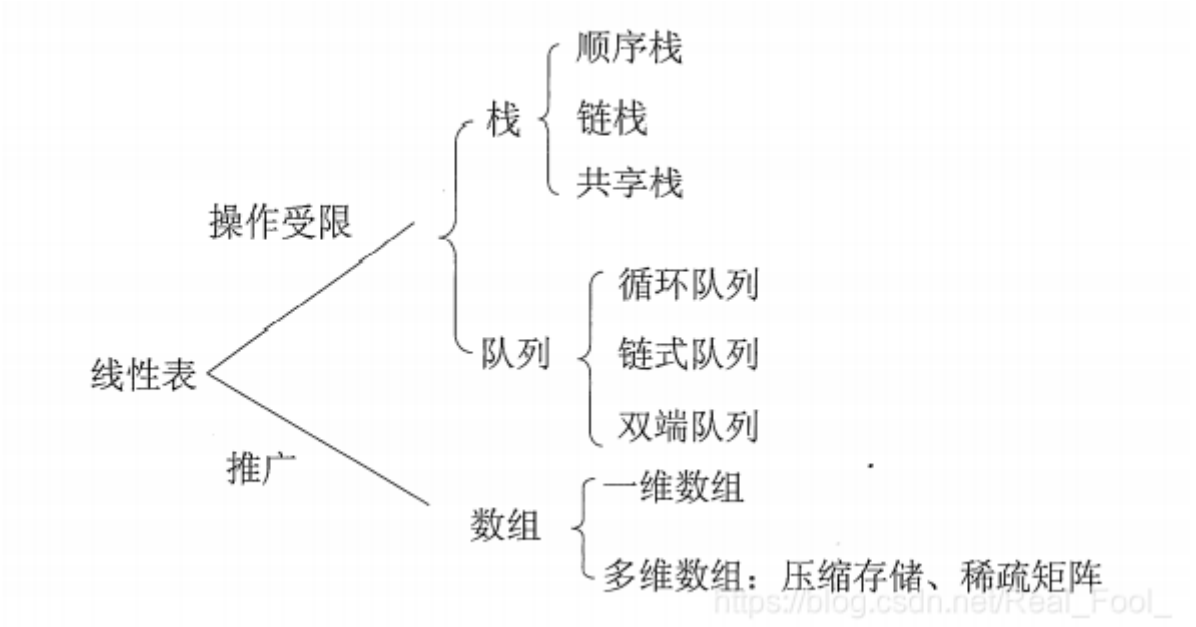
```
int SeqList::Search(const char &x){
        for(int i = 0;i<length;i++)
                if(element[i]==x) return i;
        return -1;

bool SeqList::Insert(int i,const char &x){
        if(i<0||i>length-1) return false;  //下标越界
        if(length==Maxsize) return false;  //表已满
        for(int k = length-1;k≥i;k--) element[k+1]=element[k];
        element[i]=x;
        length++;
        return true;
}
bool SeqList::Delete(int i,const char &x){
        if(Find(i,x)){
                for(int k = i;k<length-1;k++) element[k]=element[k+1];
                length--;     //最后一个数据还是自身，这没有关系，因为顺序关系的线
性表的有效范围完全取决于length这个变量，与放的内容无关。要区别于字符串
                return true;
        }
        else return false;    //下标越界
}

void SeqList::output(ostream &out)const{
        for(int i = 0;i<length;i++) out<<element[i]<<" ";
}

ostream & operator<<(ostream &out,const SeqList &x){
        x.output(out);
        return out;
}


int main(){
        char str[]="1C++ 2FORTRAN 3PASCAL 4BASIC";
        SeqLis L(100);  //建空表
        for(int i = 0;str[i]≠0;i++)     //把数据插入线性表中
                if(L.Insert(i,str[i])==false){
                        cout<<"插入异常"<<endl;
                        break;
                }
        cout<<L<<endl;  //利用重载的插入运算符显示表中的内容
        char x = 0;
        int i = 0;
        while(i<L.Length()){
                L.Find(i,x);
                if(x≥'0'&&x≤'9')
                        L.Delete(i,x);
                else
                        i++;
```

```
        }
        cout<<L<<endl;
        return 0;
}
```

线性表的链接存储结构（链表）

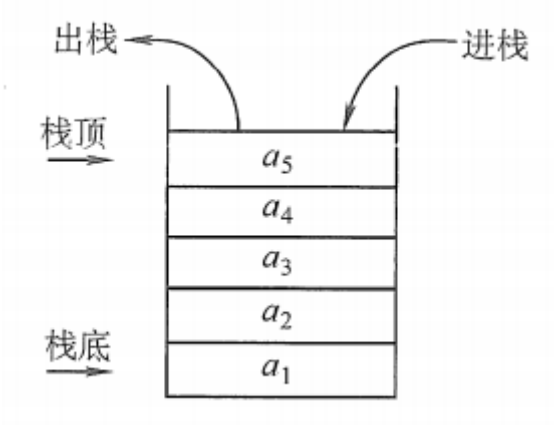
- 以链接方式存储的线性表称为链表
- 常用的链表有：单链表、双联表和循环链表等

栈



栈的概念

1.定义：
栈是限定只能在**表的同一端**进行插入和删除运算的═线性表



2.栈的术语：
栈顶：允许进行插入和删除的一端
栈底：与栈顶相对的一端
入栈：想栈顶插入一个元素
出栈：从栈顶取出一个元素

3.站的特点：
栈中的元素的变化是按**后进先出**原则进行，因此又称栈为**后进先出(Last In First Out)**表

4.栈的基本运算：
入栈、出栈、判栈空、取栈顶元素、置栈空（清空）

例题1：入栈序列为1,2,3；栈容量为2，可能的出栈序列有哪些
123,132,213,231
例题2：入栈序列为12345；出栈序列为14a5b,a=? b=?
a=3,b=2

顺序栈

- 以顺序方式存储的栈称为顺序栈
- 顺序栈可以用一维数组实现
- 栈顶指示器(top)：指示当前栈顶位置
- 栈容量(Maxsize)：栈中最多可以存放的元素个数
  空栈：top=-1；栈满：top=Maxsize-1
- 运算：判栈空、判栈满、入栈、出栈、取栈顶元素、置栈空

```
using namespace std;
class segstack{
public:
        seqstack(int m=10){    //构造函数，建立空栈
        stk = new char[m];    //动态申请栈空间
        Maxsize = mtop =-1;
        ~seqstack(){
                delete []stk;
        }  //析构函数

        bool IsEmpty(){
                return top == -1;
        }//判栈是否为空

        bool IsFu11(){
        return top==Maxsize -1;
```

```
        }//判栈是否为满

        bool Push(const char &x){     //进栈
                if (IsFu11())
                        return false;      //栈满
                top++;
                stk[top]= x;
                return true;
        }

        bool Pop(){          //出栈，对栈顶元素不做处理
                if (IsEmpty())
                        return false;    //栈空
                else{
                        top--;
                        return true;
                }
        }

        bool Top(char &x){      //取栈顶元素
                if (IsEmpty())
                        return false;    //栈空
                else {
                        x= stk[top];
                        return true;
                }
        }
        void clearstack(){      //置栈空
                top =-1;
        }
private:
        int top;
        int Maxsize;
        char* stk;
}
```

```
#include "seqstack.h"  //顺序栈类的定义和实现
void Base_conversion(long x,int base){
    char digit[]="0123456789ABCDEF",remainder,ch;
  SeqStack stk;   //缺省栈容量为10
   int temp=x;
  while(temp!=0){
     remainder=digit[temp%base];
       if(stk.Push(remainder)==false){cout<<"栈满\n
     temp/=base;
   }
   cout<<x<<"对应的"<<base<<"进制数是: ";
  while(!stk.IsEmpty()){
    stk.Top(ch);
    cout<<ch;
    stk.Pop();
```

例：利用栈实现将一个非负的十进制整数转换为B（2≤B≤16)进制整数。

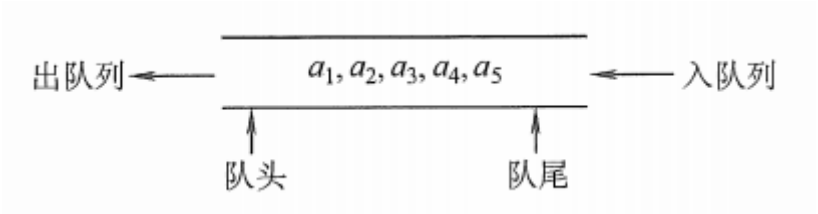
```
}
void main(){
   long num;
   int B;
    char response;
   do{
      cout<<"输入一个非负的十进制整数和转换基数B(2≤B≤16
      cin>>num>>B;
      Base_conversion(num,B);
        cout<<endl;
        cout<<"继续吗(Y/N)?";
        cin>>response;
   }while(response=='Y'||response=='y');
}
```

队列

队列的概念

（要求较低）

1.定义：
只能在表的一端进行插入，而在另一端进行删除的线性表



2.特点:
队列中元素的变化是按照 **先进先出** 原则进行，因此又称队为 **先进先出(First In First Out)表**

3.基本运算:
入队、出队、判队空、判队满、取队头元素、置队空

顺序队列

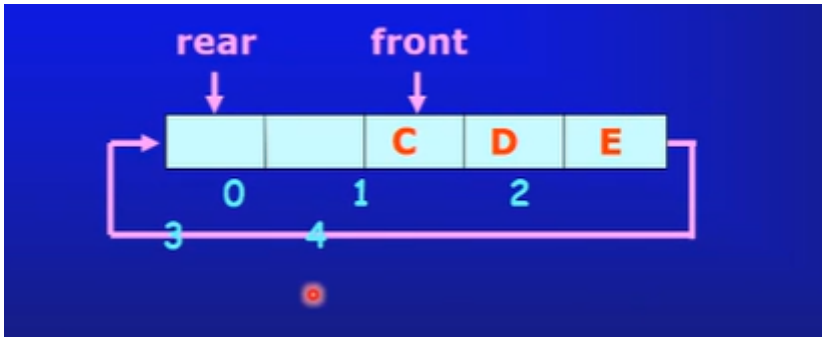
- 以顺序方式存储的队称为顺序队
- 顺序队可以用一维数组实现
- 队头指示器(front)：指示当前队头位置
- 队尾指示器(rear)：指示将要入队元素所在的位置

假溢出：当队中还有空间可放数据，但不能执行入队操作

顺序循环队列

解决假溢出问题的三种方法:

- 建立一个足够大的存储空间
- 平移元素
- **循环队列方式**，队头、队尾指针循环移动



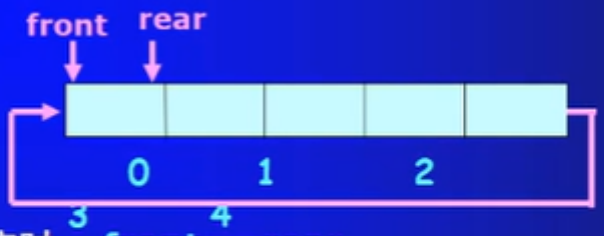
讨论：顺序循环队列的操作

1.入队、出队和取队头元素：

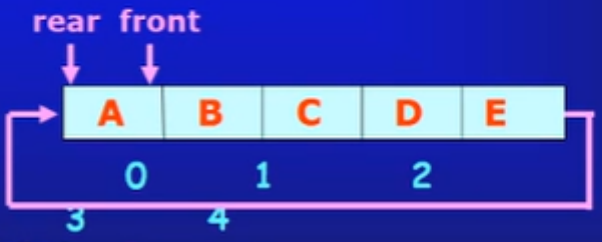
- 入队： elem[rear]=x; rear=（rear+1）%Maxsize;
- 出队：front=（front+1）%Maxsize;
- 取队头元素： x=elem[front];
- 求队列长度：
  - 方法1：(Maxsize+rear-front)%Maxsize
  - 方法2：设一个计数变量(count),由入队和出队操作控制计数

循环顺序队列（3）

2.判队空和判队满的方法



循环队列队空时： front==rear



循环队列队满时： front==rear

2.判队空和判队满的方法

方法1： 利用front和rear的值判断“队空”和“队满”

队空时： front==rear

队满时：(rear+1)%Maxsize==front

缺点：浪费一个数据元素空间

方法2： 利用当前队列元素个数（count）判断：

队空：count==0

队满：count==Maxsize

要求：记忆**队空、入队、出队、队满、队列长度

数组