- 第4章 34.
- 一个nxn的矩阵T是一个等对角矩阵（Toeplitz matrix）当且仅当对于所有的i和j都有T（i，j）=T（i-1，j-1），其中i〉1，j〉1
- 证明一个等对角矩阵最多有2n-1个不同的元素
- 给出一种映射模式，把一个等对角矩阵映射到一个大小为2n-1的一维数组中。
- 采用（2）中的映射模式设计一个C++类Toeplitz，用一个大小为2n-1的一维数组来存储对角矩阵，要求给出两个共享成员函数store和Retrieve。
- 编写一个共享成员函数，用来对两个按(2)中所给方式存储的等对角矩阵进行乘法运算，所得到的结果存储在一个二维数组中，该函数的时间复杂性是多少？

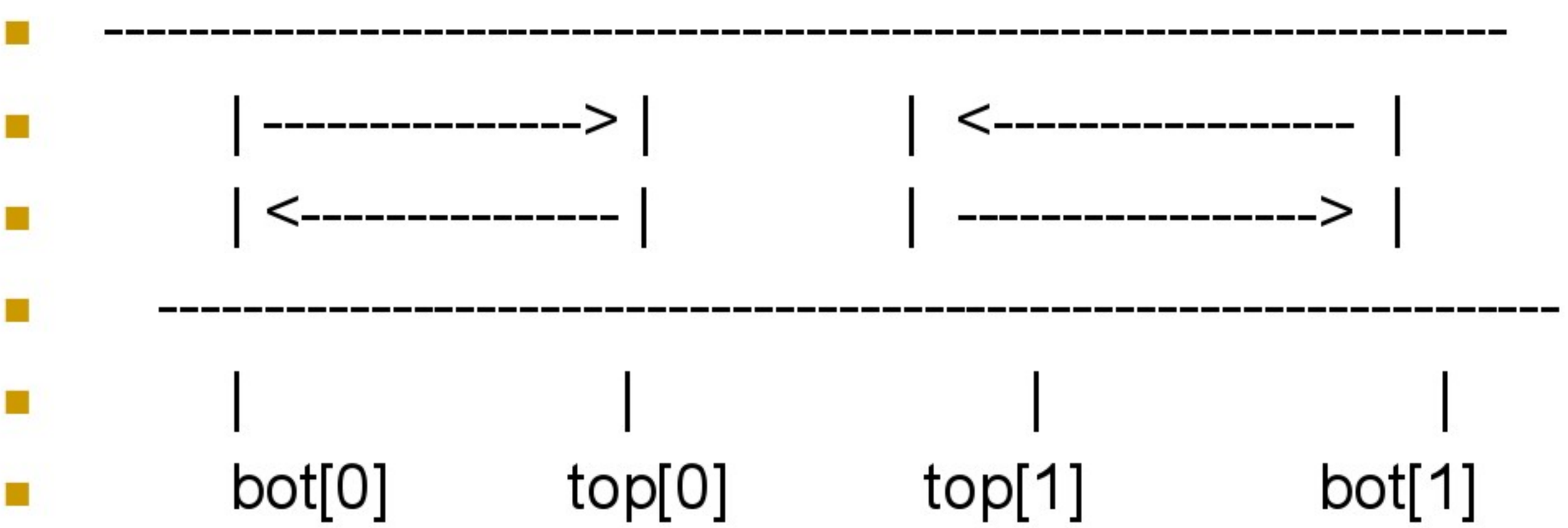
- （1）
- 证明：由于对所有的I和j都有T(I,j)=T(I-1,j-1),即对角线上的元素均相等,
- 对下三角（包括最长对角线），有n条等值线，若是这n条等值线的值互不相等，则有n个不同的元素。
- 对上三角（不包括最长对角线），有n-1条等值线，若这些等值线的值互不相等，则有n-1个不同的元素
- 对于此矩阵，若是上三角的元素和下三角的元素均不相等，此时矩阵含有的不同元素最多，即最多有2n-1个不同的元素。

- （2）映射关系为：
- 按对角线映射，从低对角线到高对角线：
- map（i，j）= n-（i-j+1)
- = n+j-i-1
-
- 按行映射：map（i，j） = n+i-j-1　　i>j
- = j- i　　i<=j

- 按列映射：map（i，j）= i-j　　i>j
- = n+j-i-1　　i<=j

第5章　关于双栈

两个栈共享一个栈空间



```
template <class T>
class DblStack
{
private:
   int top[2], bot[2]; //双栈的栈顶指针和栈底指针
   T *Elements;   //栈数组
   int m;    //栈最大可容纳元素个数
public:
   DblStack( int sz =10 );     //初始化双栈, 总体积m的默认值为10
   ~DblStack() { delete [] Elements; }  //析构函数
   DblStack<T>& Push( T &x, int i);   //把x插入到栈i的栈顶
   DblStack<T>& Pop(int i, T &x,);       //弹出位于栈i栈顶的元素
    T Top(int i);       //返回栈i栈顶元素的值
   bool IsEmpty(int i) const { return top[i] == bot[i]; }  //判栈i空否, 空则返回true, 否则返回false
   bool IsFull() const { return top[0]+1 == top[1]; }   //判栈满否, 满则返回true, 否则返回false
};
```

```
template <class T>
DblStack<T> :: DblStack (int sz)
{
 //建立一个最大尺寸为sz的空栈。
 m = sz;
 top[0] = -1;
 bot[0] = -1;
 top[1] = sz;
 bot[1] = sz;
 Elements = new T[m];    //创建栈的数组空间
 }
```

```
template <class T>
DblStack<T>&  DblStack<T> :: Push (const T &x, int i)
{
 //如果IsFull()，则报错；否则把x插入到栈i的栈顶
   If  (IsFull())  throw  NoMem();      //栈满
   if (i == 0)
      Elements[++top[0]] = x;    //栈0情形：栈顶指针先加1, 然后按此地址进栈
   else
      Elements[--top[1]] = x;    //栈1情形：栈顶指针先减1, 然后按此地址进栈
}
```

```
template <class T>
DblStack<T>&  DblStack<T> :: Pop(int i, T &x,)
{
//弹出位于栈i栈顶的元素
  if (IsEmpty(i))   throw  OutofBounds();    //判栈空否
  if (i == 0)
    x=elements[top[0]--];     //栈0情形：栈顶指针减1
  else
    x=elements [top[1]++];     //栈1情形：栈顶指针加1
}
```

```
template <class T>
T DblStack<T> ::Top(int i)
{
 //若栈不空则函数返回该栈栈顶元素。
   if (IsEmpty(i))    throw  OutofBounds();    //判栈空否
   return Elements[top[i]]; //返回栈顶元素的值
}
```

- 4.
- 修改程序6-1中的Queue类，使得队列的容量与数组queue的大小相同。为此，可引入另外一个私有成员LastOp来跟踪最后一次队列操作。可以肯定，如果最后一次队列操作为Add，则队列一定不为空；如果最后一次队列操作为Delete，则队列一定不会满。因此，当front=rear时，可使用LastOp来区分一个队列是空还是满。试测试修改后的代码。

```
template<class T>
class Queue {
public:
    Queue(int MaxQueueSize = 10);
    ~Queue() {…}
    bool IsEmpty() const
    bool IsFull() const
    T First() const; //返回队首元素
    T Last() const; // 返回队尾元素
    Queue<T>& Add(const T& x);
    Queue<T>& Delete(T& x);
private:
    int front; //与第一个元素在反时针方向上相差一个位置
    int rear; // 指向最后一个元素
    int MaxSize; // 队列数组的大小
    T *queue; // 数组
};
```

```
Queue<T>::Queue(int MaxQueueSize)
{                   ……
                    Lastop=’’;
          }
Queue<T>& Queue<T>::Add(const T& x)
{         ……
                    last0p=″add″;
          ……
}
Queue<T>& Queue<T>::Delete(T& x)
{         ……
                        last0p=″ delete ″;
          ……
}
```

bool IsEmpty() const

```
{if(rear==front && last0p==″ delete ″)
            return true;
    else return false;}
```

bool IsFull() const

```
 {if (rear==front && lastop==″ add ″)
        return true;
   else return false}
```

- 第7章
- 1.采用公式化描述方法定义C++类SortedList。要求提供与SortedChain中同样的成员函数。编写所有函数的代码，并用合适的数据测试代码。

- 10. 指出在线性开型寻址散列中进行顺序访问所存在的困难。
- 16. 指出在链表散列中进行顺序访问所存在的困难。
- 18.从低层开发ChainHashTable类。定义自己的类HashNode，其中包含data域和link域，不要使用链表类的任何版本。测试代码。
- class ChainHashTable {

  .....

  private:

  int D; // 位置数

  HashNode <E,K> **ht; // 桶数组，每个桶以一个头节点开头

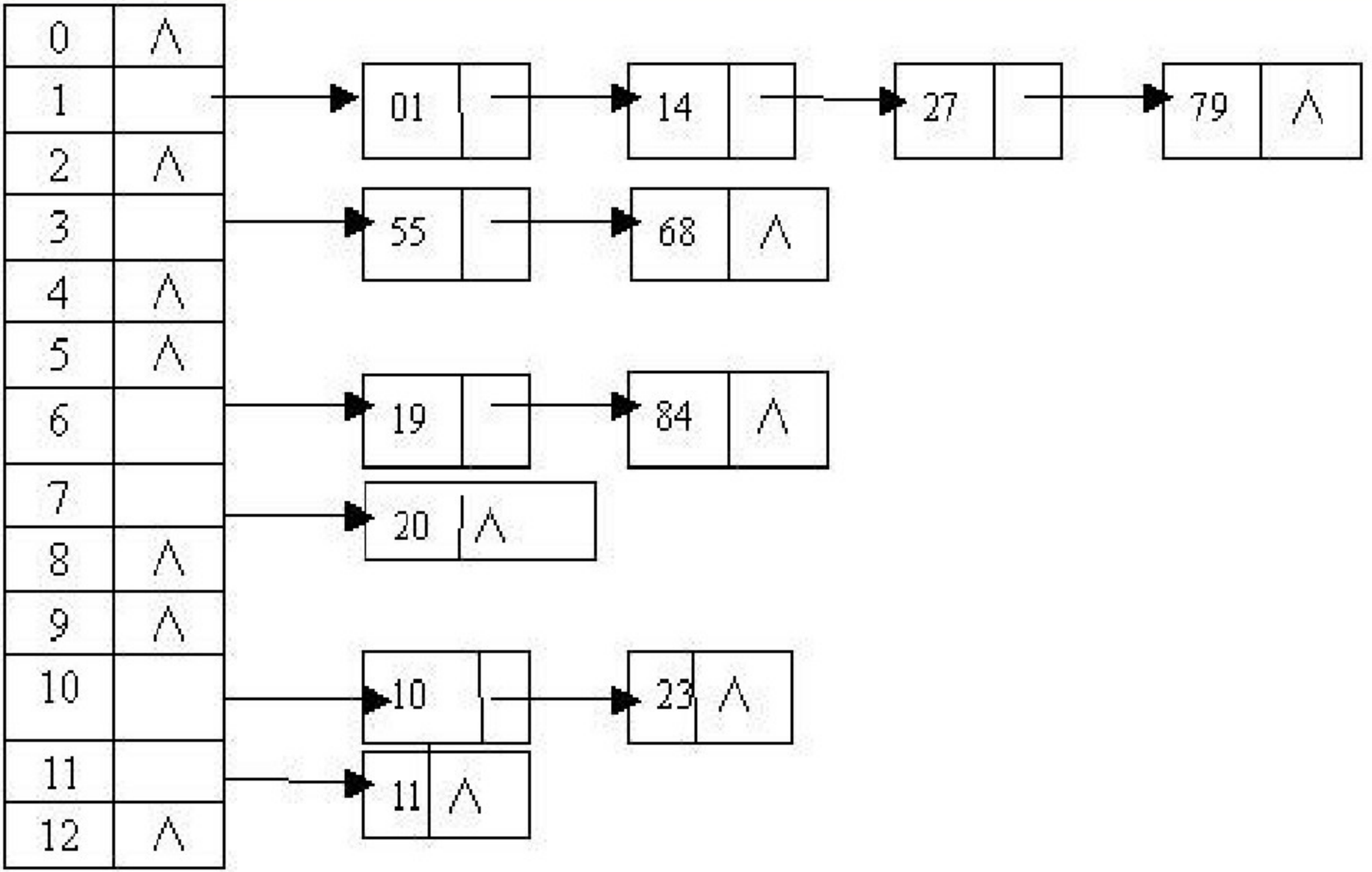
  };

- 1.设有一组关键字：{19，14，23，01，68，20，84，27，55，11，10，79}，采用散列函数：
- H（key）=key mod 13，采用线性开型寻址方法解决溢出。
- 要求：在0∽12的散列地址空间中对该关键字序列构造散列表（给出散列表的存储结构）。搜索元素27，55所花的比较次数各是多少？
- 2.有关键字集合：{19，14，23，01，68，20，84，27，55，11，10，79}，散列函数：H（key）=key mod 13，采用链地址散列方法解决溢出。要求：
- 构造散列表；搜索元素27，55所花的比较次数各是多少？

元素	19	14	23	01	68	20	84	27	55	11	10	79
H(key)	6	1	10	1	3	4	6	1	3	11	10	1

	14	01	68	20	27	19	84	55	79	23	11	10
0	1	2	3	4	5	6	7	8	9	10	11	12

搜索元素27，55所花的比较次数: 5；6。



ASL=1/12（1*6+2*4+3*1+4*1）=1.75