2.4 线性表的应用

问题描述

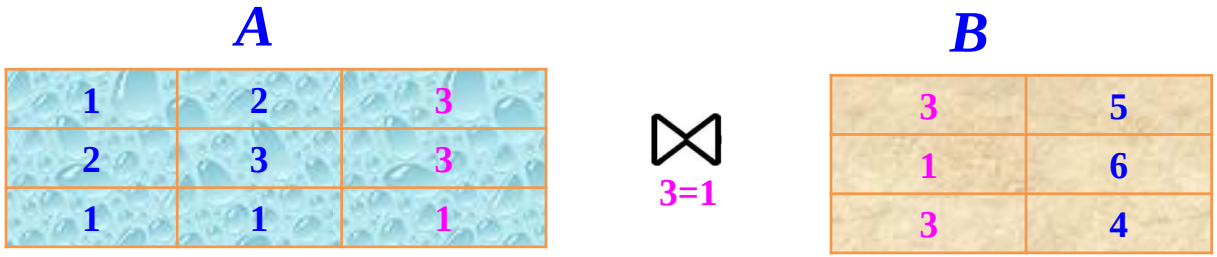
两个表自然连接问题

- 表：*m*行、*n*列。假设所有元素为整数。如：

第1列	第2列	第3列
1	2	3
2	3	3
1	1	1

一个3行3列的表

两个表自然连接



⋈

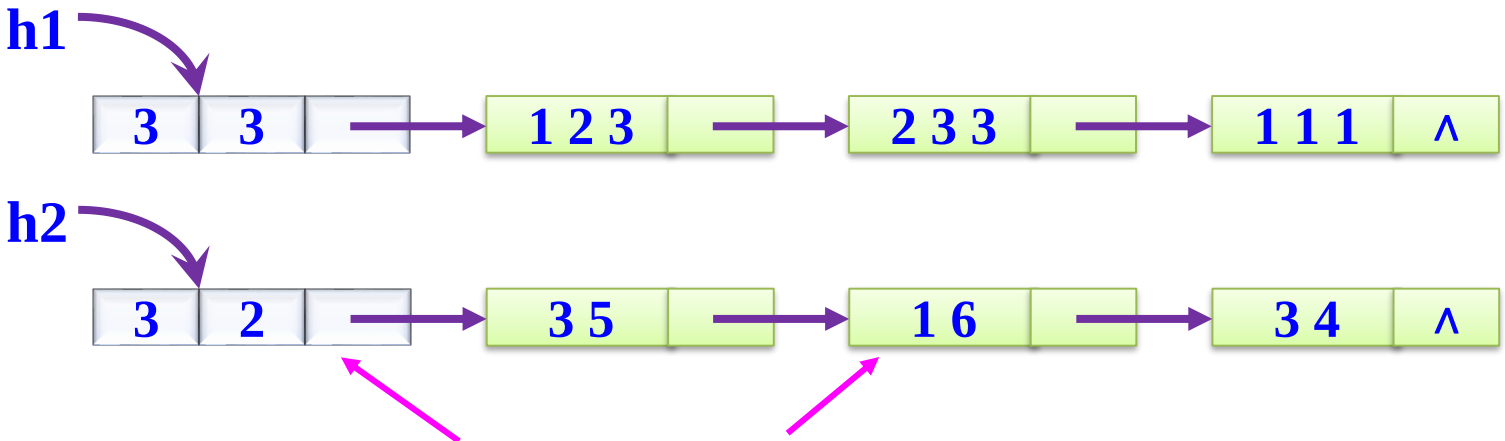
C

1	2	3	3	5
1	2	3	3	4
2	3	3	3	5
2	3	3	3	4
1	1	1	1	6

⟸ 连接结果

一般格式：$C = A \underset{i=j}{\bowtie} B$

数据组织

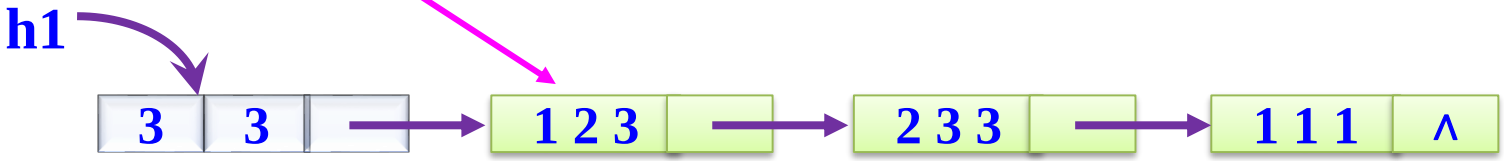


注意：头节点和数据节点的类型不同！！！

这种数据组织方式有什么好处？？？

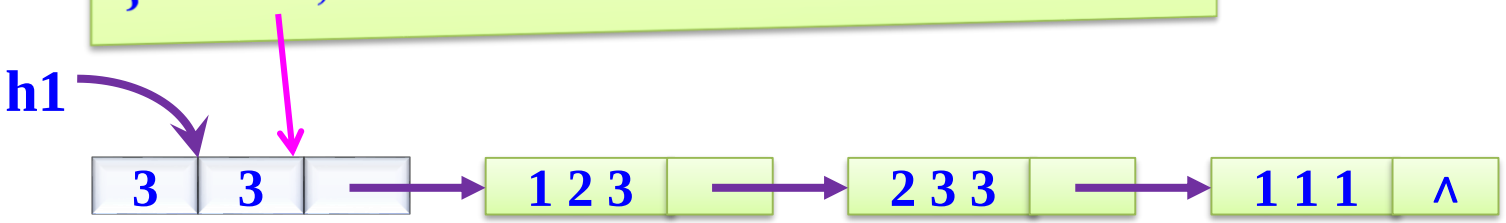
单链表中数据节点类型定义如下：

```
#define MaxCol  10                    //最大列数
typedef struct Node1                  //定义数据节点类型
{     ElemType data[MaxCol];
      struct Node1 *next;             //指向后继数据节点
} DList;
```



头节点类型定义如下：

```
typedef struct Node2        //定义头节点类型
{       int Row,Col;        //行数和列数
        DList *next;        //指向第一个数据节点
} HList;
```



顺序表和链表混合使用！！！

设计基本运算算法

① CreateTable(HList *&h)：交互式创建单链表。

② DestroyTable(HList *&h)：销毁单链表 。

③ DispTable (HList *h)：输出单链表。

④ LinkTable(HList *h1,HList *h2,HList *&h)：实现两个单链表的自然连接运算。

主程序

CreateTable(HList *&h)

DestroyTable(HList *&h)

DispTable (HList *h)

LinkTable(HList *h1,HList *h2,HList *&h)

（1）交互式创建单链表算法

```
void CreateTable(HList *&h)
{   int i,j;  DList *r,*s;
    h=(HList *)malloc(sizeof(HList));          //创建头节点
    h->next=NULL;
    printf("表的行数,列数:");
    scanf("%d%d",&h->Row,&h->Col);             //输入表的行数和列数
    for (i=0;i<h->Row;i++)                     //输入所有行的数据
    {     printf("  第%d行:",i+1);
          s=(DList *)malloc(sizeof(DList));    //创建数据节点
          for (j=0;j<h->Col;j++)               //输入一行的数据
                scanf("%d",&s->data[j]);
          if (h->next==NULL)                   //插入第一个数据节点
                h->next=s;
          else                                 //插入其他数据节点
                r->next=s;                     //将*s插入到*r节点之后
          r=s;                                 //r始终指向尾节点
    }
    r->next=NULL;                              //尾节点next域置空
}
```

采用尾插法建表

疑问： 为什么与前面尾插法建立单链表的代码不同？

回答： 因为这里头节点和数据节点类型不同，指针变量r不能同时作为头节点和数据节点的指针。这里让r指向数据节点（多个）。

（2）销毁单链表算法

```
void DestroyTable(HList *&h)
{
    DList *pre=h->next,*p=pre->next;
    while (p!=NULL)
    {       free(pre);
            pre=p;
            p=p->next;
    }
    free(pre);
    free(h);
}
```

（3）输出单链表算法

```
void DispTable(HList *h)
{    int j;
     DList *p=h->next;                    //p指向开始行节点
     while (p!=NULL)                      //扫描所有行
     {       for (j=0;j<h->Col;j++)       //输出一行的数据
                 printf("%4d",p->data[j]);
             printf("\n");
             p=p->next;                   //p指向下一行节点
     }
}
```

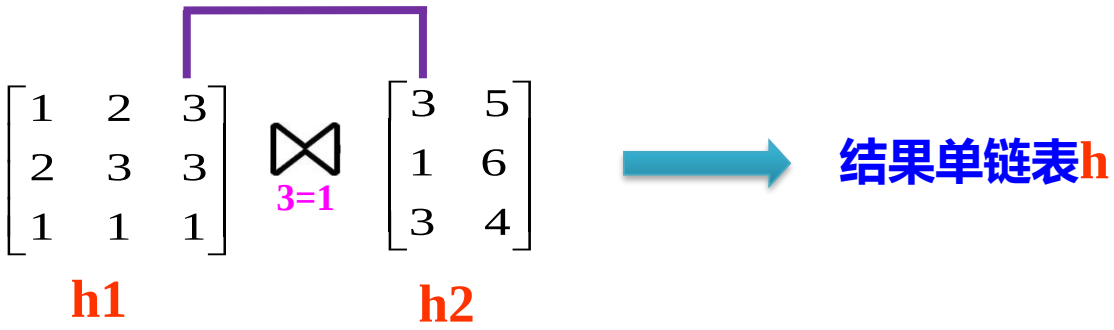
例如，输出一个表：

```
1 2 3 3 5
1 2 3 3 4
2 3 3 3 5
2 3 3 3 4
1 1 1 1 6
```

（4）表连接运算算法

p扫描h1的数据节点，q扫描h2的数据节点。

p->data[f1-1]==q->data[f2-1]

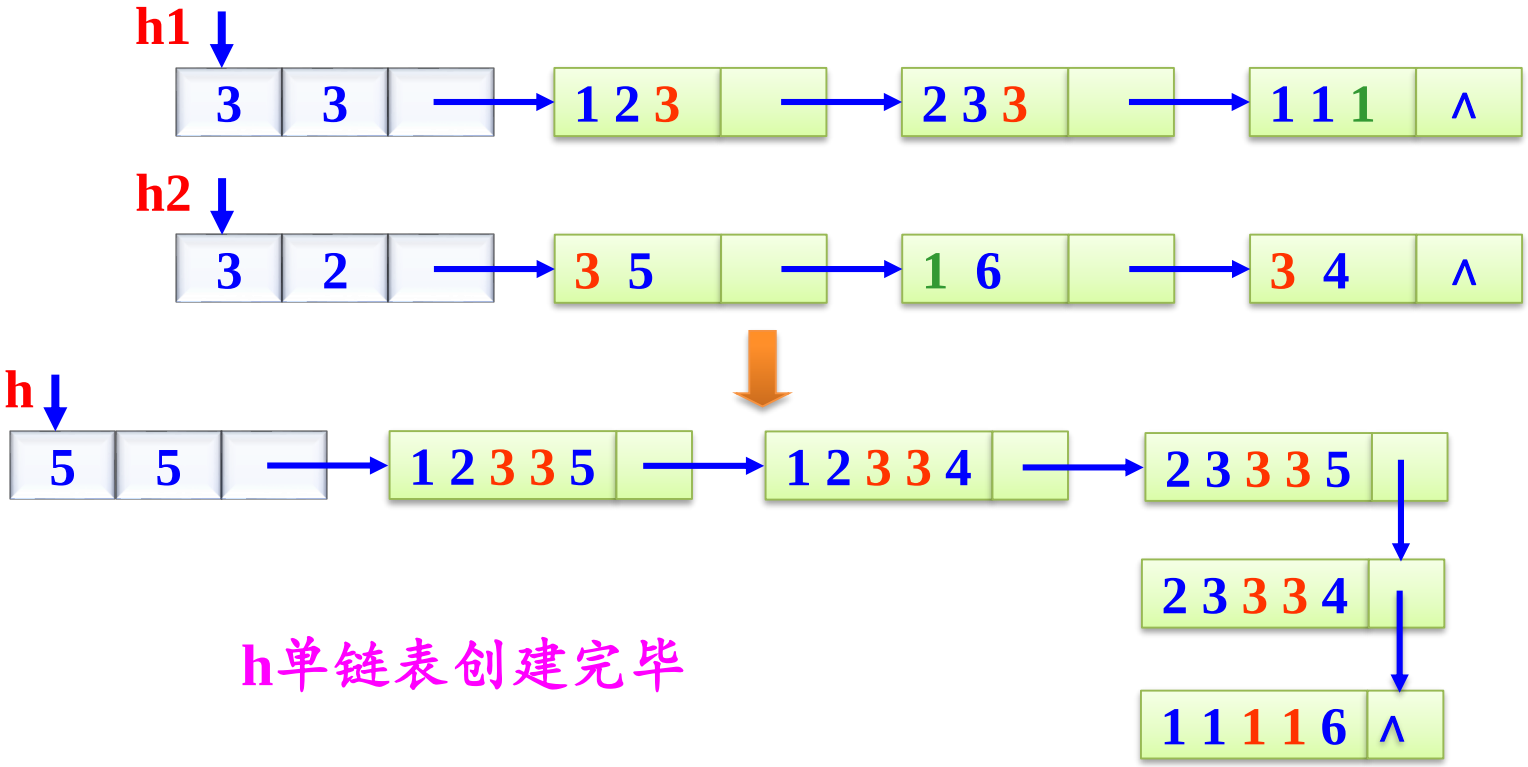


一旦条件成立，就新建一个节点插入到结果单链表h中。

结果单链表h采用尾插法建表方法创建。

两表条件连接实现的演示

连接条件为3=1。



h单链表创建完毕

```
void  LinkTable(HList *h1,HList *h2,HList *&h)
{    int i,j,k;
     DList *p=h1->next,*q,*s,*r;

     printf("连接字段是:第1个表序号,第2个表序号:");
     scanf("%d%d",&i,&j);

     h=(HList *)malloc(sizeof(HList));          //创建结果表头节点
     h->next=NULL;                              //置next域为NULL
     h->Row=0;                                  //置行数为0
     h->Col=h1->Col+h2->Col;                    //置列数为表1和表2的列数和
```

输入连接条件，如2 3表示表1的第2列和表2的第3列相等

创建头节点

```
while (p!=NULL)                                  //扫描表1
{    q=h2->next;                                 //q指向表2的开始节点
     while (q!=NULL)                             //扫描表2
     {   if (p->data[i-1]==q->data[j-1])         //对应字段值相等
         {
              s=(DList *)malloc(sizeof(DList));  //创建节点
             for (k=0;k<h1->Col;k++)             //复制表1的当前行
                  s->data[k]=p->data[k];
             for (k=0;k<h2->Col;k++)             //复制表2的当前行
                  s->data[h1->Col+k]=q->data[k];
```

条件成立，创建一个节点*s


```
            if (h->next==NULL)          //若插入第一个数据节点
                h->next=s;              //将*s插入到头节点之后
            else                        //若插入其他数据节点
                r->next=s;              //将*s插入到*r节点之后
            r=s;                        //r始终指向尾节点

            h->Row++;                   //表行数增1
        }
        q=q->next;                      //表2下移一个记录
    }
    p=p->next;                          //表1下移一个记录
}
r->next=NULL;                           //表尾节点next域置空
}
```

设计求解程序

建立如下主函数调用上述算法：

```
void main()
{    HList *h1,*h2,*h;
     printf("表1:\n");
     CreateTable(h1);                    //创建表1
     printf("表2:\n");
     CreateTable(h2);                    //创建表2
     LinkTable(h1,h2,h);                 //连接两个表
     printf("连接结果表:\n");
     DispTable(h);                       //输出连接结果
     DestroyTable(h1);                   //销毁单链表h1
     DestroyTable(h2);                   //销毁单链表h2
     DestroyTable(h);                    //销毁单链表h
}
```

运行结果

表1:
表的行数,列数: 3 3↙
  第1行: 1 2 3↙
  第2行: 2 3 3↙
  第3行: 1 1 1↙
表2:
表的行数,列数: 3 2↙
  第1行: 3 5↙
  第2行: 1 6↙
  第3行: 3 4↙
连接字段是:第1个表位序,第2个表位序: 3 1↙
连接结果表:
1 2 3 3 5
1 2 3 3 4
2 3 3 3 5
2 3 3 3 4
1 1 1 1 6

思考题：

体会利用线性表数据结构求解问题的一般过程。

——本讲完——