

3.1 线性表概述

数据之间存在各种关系,其中存在一种简单的“前后相继”的关系,即 A 数据是 B 数据的前驱, B 数据是 A 数据的后继。如果每个数据都有唯一的前驱元素和唯一的后继元素,那么就称为线性存储结构,即线性表。

其中,第一个数据元素无前驱,最后一个数据元素无后继。数据元素之间的关系是“一对一”的,适合存储在线性表中。线性表是一种简单、基本的数据结构,应用场景较为广泛,可以用于存储和处理一组结构相同的数据元素,如一组学生的考试成绩,也是后续章节中堆栈和队列的实现基础。

线性表有顺序和链表两种存储方式。顺序存储方式用数组实现,定义一个指定长度的数组,将数据元素依次存储在数组中,使用数组下标可以访问到指定位置的数据元素。链表存储方式创建一组带指针域和数据域的结构体,数据域中存储数据元素,指针域中存储相邻数据元素的地址。

3.2 线性表的定义及基本操作

一个线性表是 n 个具有相同特性的数据元素的有限序列。数据元素可以是简单数据类型,也可以是结构体、对象、指针、字符串等其他数据类型。但在同一个线性表中,数据类型必须统一。在线性表中,出现在 A 数据前面的数据元素称为 A 的“前驱元素”;出现在 A 的后面的元素称为 A 的“后继元素”。在线性表中, A 的前驱元素中最接近 A 的数据元素称为“直接前驱”; A 的后继元素中最接近 A 的数据元素称为“直接后继”。

线性表的基本操作如下。

创建空表: 创建一个线性表,数据元素个数初始化为 0。

插入: 确定插入的位置,在线性表中插入一个元素,元素个数加 1。

删除: 找到要删除元素的位置,删除元素,元素个数减 1。

查找: 给出元素的值,返回元素在线性表中的位置,如果元素不存在则返回空。

3.3 线性表存储结构

线性表有两种存储结构,顺序存储结构将数据存储于连续的物理空间,通过下标确定数据的位置;链式存储结构将数据存储于不连续的物理空间,通过指针确定后继数据的位置。

3.3.1 线性表的顺序存储结构

顺序存储结构用数组的形式实现。在 C++ 中,定义一个数组的时候,系统为数组分配了一个连续的物理空间。数组的名就是数组存储在内存中的首地址。线性表的数据元素的数据类型就是数组数据类型,各个数据元素的数据类型相同。

数组的长度即为数组中可以存储的数据元素的个数。数组元素的下标可以表示数据元素在线性表中的位置。

顺序存储结构的优点有:创建简单,访问效率高,可以通过下标从任意位置开始访问线性表,查找前驱元素和后继元素都很方便。

3.3.2 线性表的链表存储结构

顺序结构在使用方便的同时也存在限制:数组大小确定,不能动态增长;存储的内存需要连续的空间,随着数据元素的增加,可能无法开辟出足够的连续内存空间。而链表结构可以解决顺序结构的这些限制。链表结构在需要存储时再动态开辟一个结点空间,结点空间由以下部分组成。

数据域:每个结点存储的数据元素本身的信息。

指针域:元素之间逻辑关系的信息,后继指针存储后继结点的地址,前驱指针(可选)存储前驱结点的地址。通过指针域可以找到相邻结点的地址,提高数据查找速度。

一般来说,每个结点有一个或多个指针域,若某个结点不存在前驱或者后继,对应的指针域设为空,用常量 NULL 表示。

链表存储结构可以用于实现堆栈和队列,处理大规模数据,实现动态扩展和收缩的数据结构,还可以用于 Hash 表解决冲突。

链表有单向链表、双向链表、循环链表等形式。

3.4 线性表的实现

3.4.1 单链表

结点只包含指向后继结点的指针,适用于基本上只做单向访问的线性表。链表结点可以用结构体来描述如下。

```
//定义单链表结点
struct ListNode {
    int val;
```



单链表

```

ListNode * next;
ListNode(int x) : val(x), next(NULL) {}
};

```

其中,成员 val 用于存放结点中的有用数据,next 是指针类型的成员,指向 struct ListNode 类型数据,构建结点时可将指针域的默认值设为空。

链表头有两种设置模式:头结点模式和头指针模式。头结点模式中,head 是一个结点,next 域存储第一个结点的地址;头指针模式中,head 是一个指针,存储第一个结点的地址,如图 3.1 所示。

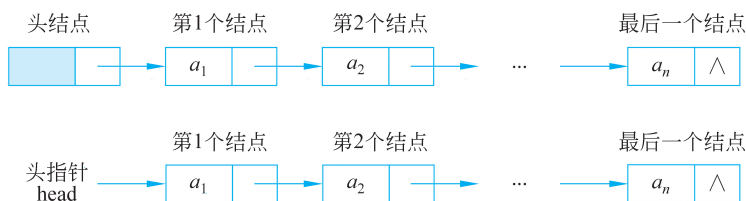


图 3.1 头结点和头指针模式下的单链表

虽然直接在栈区创建结构体变量可以用于创建链表,但是不能体现链表动态改变长度的优势,因此往往使用动态内存分配,将结点建立在堆区。下面使用动态内存分配创建一个学生成绩链表,学生的信息包含学号和成绩,并在链表中加入 4 个学生的数据。

1. 链表类的创建

首先创建链表结点结构体,设置一个头指针 head 指向链表的第一个元素。

```

struct Student {
    int id;           //学号
    float score;      //成绩
    Student * next;   //指向下一个结点的指针
};
/* 学生链表类 */
class LinkedList {
private:
    Student * head;   //指向链表的头指针

```

2. 构造函数与析构函数

构造函数创建一个空的链表,此时链表中无结点,head 指针的值为空。析构函数需要清空链表,将堆区所占用的空间进行释放。head 指针所指的数据不能直接释放,否则链表后继数据将无法找到,需要借助一个 temp 指针指向 head 指针所指的第一个元素,head 指针再进行后移,释放掉 temp 指针所指的数据元素,循环进行,直至所有数据都被释放。

```

public:
    LinkedList() {
        head = NULL;
    }
    ~LinkedList() {
        Student * temp;

```

```

while(head !=NULL) {
    temp =head;
    head =head->next;
    delete temp;
}
}

```

3. 头插法插入结点

使用头插法插入一个学生结点。首先创建一个新的学生结点,然后将学号和成绩赋给结点数据域,结点指针域赋为 head,即第一个元素的地址,最后将 head 指针指向新结点。

```

void add(int id, float score) {
    Student * newNode =new Student();
    newNode->id =id;
    newNode->score =score;
    newNode->next =head;
    head =newNode;
}

```

4. 按序插入结点

按照给出的学号,找到适合的位置插入结点,使得学号按升序排列。首先生成新结点 newNode,然后使用访问指针 p 从第一个元素开始进行比较,找到插入位置后终止循环,如图 3.2 所示。

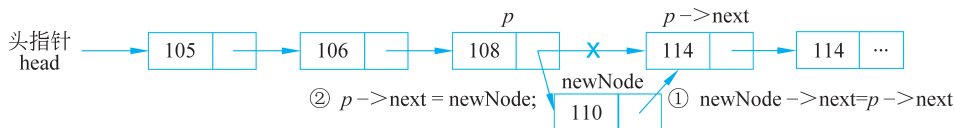


图 3.2 按序插入结点

```

void insert(int id, float score)
{
    Student * newNode =new Student();
    newNode->id =id;
    newNode->score =score;
    newNode->next =NULL;
    if(head ==NULL)
    {
        head =newNode;
    }
    else
    {
        Student * p =head;
        //使用访问指针 p 找到插入元素的位置
        while(p->next !=NULL && p->next->id <newNode->id)
        {
            p =p->next;
        }
    }
}

```

```

    }
    newNode->next = p->next;
    p->next = newNode;
}
}

```

5. 删除结点

删除结点首先要查找到指定结点,使用一个访问指针 p 对链表进行遍历,比对结点的 id 值进行查找。对结点进行删除后,要将前后的两个结点进行连接,因此需要借助一个前驱指针 $prev$ 来保存左侧结点的地址。如果删除的结点是第一个元素,那么在查找中 p 指针没有移动, $prev$ 指针的值为空,此时只需要将 $head$ 指针指向第二个元素,再释放 p 指针即可。假设要删除的是学号为 103 的学生,如图 3.3 所示。

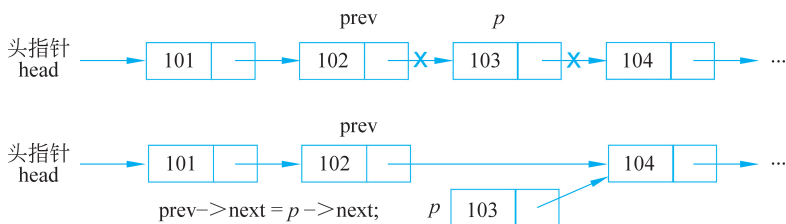


图 3.3 删除一个结点

```

void remove(int id)
{
    Student* p = head;           //设置访问指针指向第一个元素
    Student* prev = NULL;        //设置前驱指针指向访问指针前一个元素
    while(p != NULL && p->id != id) //查找 id 对应的结点
    {
        prev = p;
        p = p->next;
    }
    if(p == NULL)                 //没有找到对应的学号
    {
        cout << "Student not found." << endl;
        return;
    }
    if(prev == NULL)              //id 找到的是第一个元素
    {
        head = p->next;
    }
    else                          //其余的位置
    {
        prev->next = p->next;
    }
    delete p;                    //释放要删除的元素
}

```

3.4.2 双向链表

双向链表在每个结点中除包含数值域外,设置有两个指针域,分别用于指向其前驱结点和后继结点,这样构成的链表称为线性双向链表,简称双链表。

在双链表中具有两个指针,所以当访问过一个结点后,既可以依次向后访问每个结点,也可以依次向前访问每个结点,如图 3.4 所示。

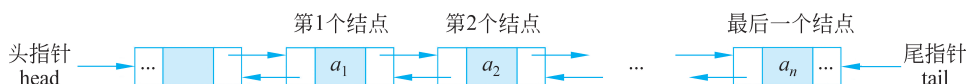


图 3.4 双向链表

双向链表可以设计一个头指针 head 指向第一个元素,一个尾指针 tail 指向最后一个元素,创建双向链表时将头尾指针都赋为空。双向链表中增加了一个指针域,成员函数中要增加相应的处理。

```
#include <iostream>
using namespace std;
struct Node           //链表结点结构体
{
    int data;
    Node* prev;        //前驱指针
    Node* next;        //后继指针
    Node(int d)        //初始化结点
    {
        data = d;
        prev = NULL;
        next = NULL;
    }
};
class DoublyLinkedList //双向链表类
{
public:
    Node* head;        //头指针
    Node* tail;        //尾指针
    DoublyLinkedList()
    {
        head = NULL;
        tail = NULL;
    }
    void append(int data) //尾插插入元素
    {
        Node* newNode = new Node(data);
        if (head == NULL)
        {
            head = newNode;
            tail = newNode;
        }
    }
};
```

```

    }
    else
    {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

void prepend(int data)           //头插插入元素
{
    Node * newNode = new Node(data);
    if(head == NULL)
    {
        head = newNode;
        tail = newNode;
    }
    else
    {
        head->prev = newNode;
        newNode->next = head;
        head = newNode;
    }
}

void print()                     //打印双向链表
{
    Node * p = head;
    while(p != NULL)
    {
        cout << p->data << " ";
        p = p->next;
    }
    cout << endl;
}
};

```

3.4.3 循环链表

循环链表形成环状结构,有 head、tail 两个固定指针,tail 结点的 next 指针指向第一个结点,在双向的循环链表中,head 的 prev 指针指向最后一个结点,这样将链表连接为环状结构。从表中任一结点出发均可找到链表中的其他结点。循环链表的结构如图 3.5 所示。双向循环链表可以由头指针找到尾指针,类中也可以不设置 tail 指针。

循环链表形成的环状结构可以解决环状结构的问题,如约瑟夫环问题。

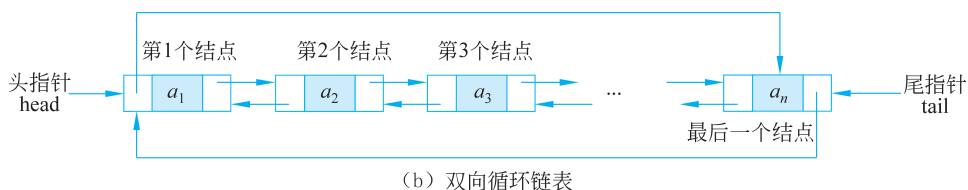
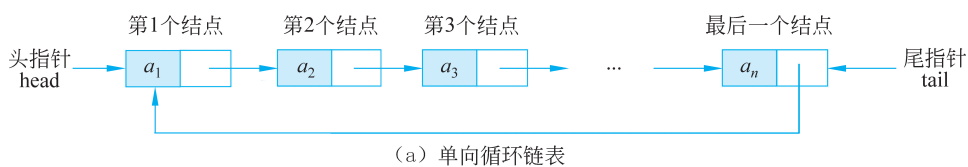


图 3.5 循环链表

3.5 能力拓展

3.5.1 判断链表中是否存在环

给定一个单链表,其中可能存在某个结点的 next 指针指向前面已经出现过的结点,此时链表中出现了环,如果使用访问指针直接进行遍历则无法达到链表的尾端。给出一个链表的头指针,要求返回链表开始入环的第一个结点。从链表的头结点开始沿着 next 指针进入环的第一个结点为环的入口结点。如果链表无环,则返回 NULL。



判断链表中
是否存在环

为了表示给定链表中的环,使用整数 pos 来表示链表尾连接到链表中的位置(索引从 0 开始)。如果 pos 是 -1,则在该链表中没有环。注意, pos 仅用于标识环的情况,并不会作为参数传递到函数中。不允许修改给定的链表。

如表 3.1 所示为判断链表是否有环的示例。

表 3.1 判断链表是否有环的示例

示 例 1	示 例 2	示 例 3
head = [1,4,2,7,5], pos = 1	head = [1,4,2,7,5], pos = 0	head = [1,4,2,7,5], pos = -1
返回索引为 1 的链表结点	返回索引为 0 的链表结点	返回 NULL
链表中有一个环,其尾部连接到第 2 个结点	链表中有一个环,其尾部连接到第 1 个结点	链表中不存在环

解题思路:

使用一个慢指针 slow 和一个快指针 fast 判断链表中是否存在环。两个指针都从 head 指针开始对链表进行访问,slow 指针每次移动一个结点,fast 指针每次移动两个结点。如果链表不存在环,那么 fast 指针将先访问到 NULL,如图 3.6 所示。

当链表存在圈时,fast 指针和 slow 指针会相遇,设 slow 指针走过的结点个数为 s,fast 指针走过的结点个数是 2s。设链表的圈外结点个数为 Nout,圈内结点个数为 Nin,圈内

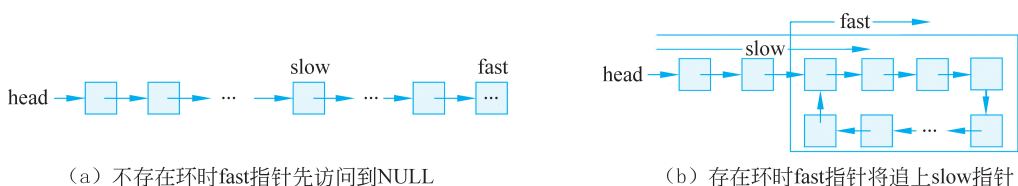


图 3.6 使用快慢指针判断是否有环

slow 指针走过的结点个数为 S_v , 如图 3.7 所示。

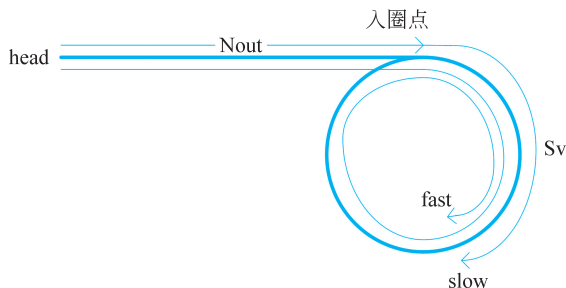


图 3.7 快慢指针相遇时的情况

当快慢指针相遇时,慢指针走过的结点数目:

$$s = N_{out} + S_v \quad (1)$$

快指针走过的结点数目(n 为快指针走过的圈数):

$$2s = N_{out} + n \times N_{in} + S_v \quad (2)$$

由(1)(2)式联立得 $N_{out} = n \times N_{in} - S_v$, 即圈外结点的数目 N_{out} 和从相遇点出发共绕 n 圈回到入圈点的距离相等。slow 指针此时停留在相遇点, 因此只需要让 slow 指针走 N_{out} 步就能刚好回到圈的入口点。可以再设置一个指针 p 从 head 出发进行访问, 当 p 和 slow 相遇时, p 走过了 N_{out} 步, slow 回到入口点, 即求得圈的入口点。

```

/* *
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode * next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 * /
class Solution {
public:
    ListNode * detectCycle(ListNode * head) {
        if(head==NULL||head->next==NULL) //特殊情况
            return NULL;
        ListNode * fast=head;
        ListNode * slow=head;
        do{
            if(fast==NULL||fast->next==NULL)
                return NULL;

```

```

        fast=fast->next->next;
        slow=slow->next;
    }while(fast!=slow);
    //退出循环时快慢指针相遇了
    ListNode * p=head;           //设置指针从 head 出发
    while(p!=slow)               //p 和 slow 将相遇在入圈点
    {
        p=p->next;               //两指针移动
        slow=slow->next;
    }
    return slow;                 //返回入圈点的地址
}
};

```

3.5.2 约瑟夫环

约瑟夫环问题：有 num 只猴子，按顺时针方向围成一圈选大王（编号从 1 到 num），从第 1 号开始报数，一直数到 target，数到 target 的猴子退出圈外，剩下的猴子再接着从 1 开始报数。就这样，直到圈内只剩下一只猴子时，这个猴子就是猴王，编程求输入 num 和 target 后，输出最后猴王的编号。

解题思路：

约瑟夫环是学习算法的一个经典问题，可以使用数组、指针、链表和数学方法解决。我们用循环链表模拟猴子报数的结构。每只猴子都看成一个结点，从 1 号到 num 号依次加入链表中。链表 head 指针指向 1 号，num 号猴子的 next 指针指向 head，使得猴子能形成环状结构，示例如图 3.8 所示。

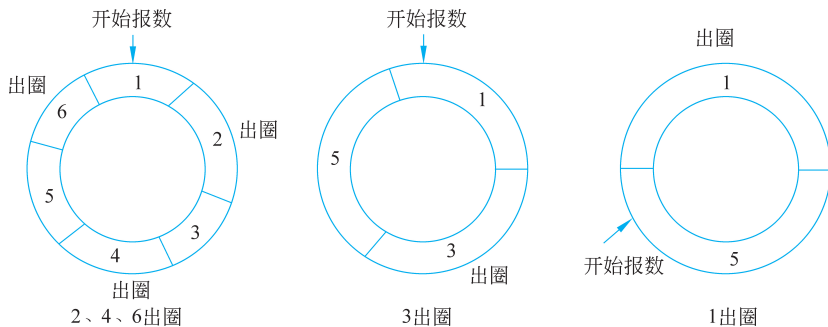


图 3.8 6 只猴子进行报数，报到 2 出圈，最后剩下 5 号

用指针 cur 从表头开始对链表进行访问模拟报数的过程，每跳一个结点进行一次报数，报到 target-1 时删除 cur 指针所指的下一个结点，最后只剩下一个结点时输出结点的编号。

```

struct Node
{
    int val;
    Node * next;
    Node(int x) : val(x), next(nullptr) {}
};

```

```
int josephus(int num, int target)
{
    //创建循环链表
    Node * head = new Node(1);
    Node * cur = head;
    for(int i = 2; i <= num; i++)
    {
        cur->next = new Node(i);
        cur = cur->next;
    }
    cur->next = head;                //形成循环
    //当圈内个数大于 1 时
    while(num > 1)
    {
        //进行报数
        for(int i = 1; i < target; i++)
        {
            cur = cur->next;
        }
        //删除结点
        Node * tmp = cur->next;
        cur->next = tmp->next;
        delete tmp;
        num--;                      //圈内数目减 1
    }
    //返回最后剩下的结点编号
    return cur->val;
}
```



习 题

1. 有带有字母 A、B、C 的三张卡片,按某种顺序排列在一排。最多可以进行一次操作,选择两张牌并交换它们。是否操作后可以变成 ABC? 是就输出“YES”,否就输出“NO”。

输入格式: 第一行包含一个整数 $t(1 \leq t \leq 10)$,代表有 t 组数据。接下来 n 行,每行包括一个长度为 3 的字符串,由 A、B、C 三个字符组成。

输出格式: “YES”或者“NO”。

2. 小 S 有一个数组,他想检查这个数组是否是一个排列。一个长度为 n 的排列指的是, $1-n$ 在这个排列中都出现且仅出现过一次。

输入格式: 第一行输入一个整数 n 。第二行输入 n 个整数,表示数组元素。

输出格式: “YES”或者“NO”。

3. 众所周知,小 S 十分挑食。这天,小 S 又来到了食堂,但他的挑食症又犯了,他不会选择热量为 k 的食物,请问他有多少种选择?

输入格式: 第一行输入一个整数 t ,代表有多少组数据。第二行输入两个整数 n 和 k , n 代表有多少个食物,不同食物的热量可能相同。第三行输入 n 个整数 a_i , a_i 表示每个食物的热量。($1 \leq t \leq 10^4, 1 \leq n \leq 10^3, 1 \leq a_i, k \leq 10^9$ 。)

输出格式：他有多少种选择，每组数据输出一个整数占一行。

4. 给定两个大小分别为 m 和 n 的数组 a 和 b 。请找出并输出这两个数组的中位数。

输入格式：第一行输入两个整数 n 和 m 。第二行输入 n 个整数，表示数组 a 。第三行输入 m 个整数，表示数组 b 。

输出格式：请输出一个浮点数，保留一位小数。

5. 森林里有 n 只猴子，它们都想当猴王，因为当猴王就不用自己找果子吃。但不是每只猴子都能当猴王，能当上猴王的只能有一只，于是便有了这样的规则： n 只猴子围成一个圈，从第 1 只猴子开始报数 $1, 2, 3, \dots$ 报出 m 的那只猴子出列，失去继续争夺猴王的机会，由下一只猴子继续从 1 开始报数，最终只剩下一只猴子得胜成为猴王。

输入格式：输入两个整数 n 和 m 。

输出格式：输出一个整数——第几只猴子获胜成为猴王。

6. 小 S 最近在学习后缀表达式，他已经把后缀表达式练得炉火纯青了，这一天他又看到了一道后缀表达式的题，他觉得太简单了，于是交给了你。后缀表达式，指的是不包含括号，运算符放在两个运算对象的后面，所有的计算按运算符出现的顺序，严格从左向右进行（不再考虑运算符的优先规则）。将给出一个后缀表达式 T ，保证 T 的长度不超过 100，且操作数仅为 $0 \sim 9$ 的整数，操作符只包含 $+$ 、 $-$ 、 $*$ ，求该后缀表达式的计算结果。

输入格式：本题的输入为单个测试用例，输入长度不超过 100，且操作数仅为 $0 \sim 9$ 的整数，操作符只包含 $+$ 、 $-$ 、 $*$ 等算术运算符的中缀表达式。

输出格式：输出一个整数，为该后缀表达式的计算结果。

7. 斯拉夫正在为朋友的生日准备礼物，礼物是 n 个数字的乘积。斯拉夫有一次机会选择 n 个数字中的一个数字来增加 1。请问斯拉夫可以送给朋友的最大礼物数是多少？

输入格式：第一行包含一个整数 t ($1 \leq t \leq 10^4$) 测试用例的数量。每个测试用例的第一行包含一个整数 n ($1 \leq n \leq 9$)。每个测试用例的第二行包含 n 个空格分隔的整数 a_i ($0 \leq a_i \leq 9$)。

输出格式：对于每个测试用例，输出一个整数。

8. 给你一个整数数组 a ，判断是否存在三元组 $a[i], a[j], a[k]$ ，满足 $i \neq j, i \neq k, j \neq k$ 并且 $a[i] \neq a[j], a[i] \neq a[k], a[j] \neq a[k]$ ，同时还满足 $a[i] + a[j] + a[k] = 0$ 。请统计三元组的数量。

输入格式：第一行输入一个整数 n ，数组 a 的长度。第二行输入 n 个整数。

输出格式：输出一个整数，为三元组的数量。

9. 给定 n 个整数 $a_1, a_2, \dots, a_n$ ，求问有多少个四元组 (i, j, k, l) 满足以下条件： $1 \leq i < j < k < l \leq n, a_i = a_k, a_j = a_l$ 。

输入格式：第一行输入一个整数 t ，表示数据组数。对于每组数据，下一行输入一个整数 n ，表示有多少个数。对于每组数据，下一行输入 n 个整数，表示 $a_1, a_2, \dots, a_n$ 。

输出格式：每组输出一个整数，为四元组的数量。

10. 小李在玩一个游戏，他有 n 个盒子，每个盒子有两个数字，小李自己也有两个数字。每个盒子可以得到的分数，是排在这个盒子前面的所有盒子的第一个数的乘积，除以这个盒

子的第二个数,然后向下取整得到的结果。小李不希望某一个盒子得到特别多的分数,所以他想请你帮他重新安排一下盒子的顺序,使得得到分数最多的盒子,所获分数尽可能的少。注意,小李的位置始终在盒子队列的最前面。

输入格式: 第一行包含一个整数 n ,表示盒子的数量。第二行包含两个整数 a 和 b ,之间用一个空格隔开,分别表示小李的两个数。接下来 n 行,每行包含两个整数 a 和 b ,之间用一个空格隔开,分别表示每个盒子的两个数。

输出格式: 一个整数,表示重新排列后的盒子队列中获分数最多的盒子所获得的分数。