数据结构 A 课后题答案
2023-2024-2 合订本

王骏峣 *2024* 年 *6* 月 *22* 日 于珞珈山
wjyyy1@whu.edu.cn

目录

注意事项

教材：数据结构教程（C++ 语言描述）李春葆等
助教：王骏峣
本学期的数据结构 A 是机考，在机考时会有选择题和程序设计题。选择题会对概念和细节有一定的考察，建议考前多看一下书，复习一些不常见但常用的概念（如 ASL），尤其是对查找次数、循环边界这种是否要 $+1, -1$ 的地方多加注意。
对于程序设计题，一般而言是有部分分的，不要看到自己不会的题目就直接放弃，可以先写一些基础的代码，通过样例，然后逐步完善。同时，吸取期中考试的教训，注意文件流的输入输出。相关介绍在本文档的最后一页应该也还有。如果实在不会写编程题，至少也要输出个样例提交上去吧！不过还是希望大家冷静审题，都能写出来。
最后祝所有同学考试顺利，能够在珞珈山度过愉快的四年！

数据结构 A
作业 1 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：绪论、算法分析
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

某算法的空间复杂度为 $O(1)$，则（　　）。
A. 该算法执行所需辅助空间大小与问题规模 n 无关
B. 该算法执行所需全部空间大小与问题规模 n 无关
C. 该算法执行不需要任何辅助空间
D. 该算法执行不需要任何空间

答案：A

解析：参考课件「第 1 讲-绪论」第 87 页，一个算法的存储量包括形参所占空间和临时变量所占空间。在对算法进行存储空间分析时，只考察临时变量所占空间。

参考课本 25 页，空间复杂度是对一个算法在执行过程中临时占用的存储空间的量度。

因此考虑带**辅助空间**的选项。其中 $O(1)$ 表示线性复杂度，例如使用了常数个辅助变量，也就是辅助空间大小与问题规模 n 无关，故选 A。

习题 2 求时间复杂度

已知 n 为正整数，以下算法的时间复杂度为（　　）。

```
void fun(int n)
{
    int i,j,s=0;
    for(i=1;i<=n;i=i*3)
        for(j=i/3;j<=i;j++)
            s=s+j;
}
```

答案：$O(n)$

解析：分析算法复杂度，就是考虑这段代码一共执行了多少次语句。

方法 1：本题为填空题，可以直接本地代码跑出结果。

```
void fun(int n)
{
    int i,j,s=0;
    int cnt=0;
    for(i=1;i<=n;i=i*3)
        for(j=i/3;j<=i;j++)
        {
            s=s+j;
            cnt++;
        }
    cout<<n<<" "<<cnt<<endl;
}
```

输入 $n=100$，得到结果 $cnt=86$；输入 $n=1000$，得到结果 $cnt=736$。

大致分析出算法复杂度和 n 线性相关，复杂度不会超过 $O(n)$。熟练后可以推测出结果为 $O(n)$。

方法 2：求和算法分析。外层循环执行了 $\log_3 n$ 次，每次循环的时候的 i 带入内层循环。现在在进行第几次外层循环，i 就是 3 的几次方。也就是说，第 k

次外层循环时，$i = 3^k$。

因此内层的 j 从 $i/3$ 到 i 一共是 $\frac{2}{3}i + 1$ 次，复杂度中常数忽略之后可认为执行了 $O(i)$ 次。

所以算法分析的结果是：

$$\begin{aligned} T(n) &= \sum_{k=1}^{\log_3 n} (\frac{2}{3}i + 1) \\ &\approx \sum_{k=1}^{\log_3 n} i \\ &= \sum_{k=1}^{\log_3 n} 3^k \\ &= \frac{1 - 3^{\log_3 n}}{1 - 3} (\text{等比数列求和}) \\ &= O(n) \end{aligned}$$

习题 3 最大因子

【问题描述】
给定一组共 $n(1 \leq n \leq 5000)$ 个整数，整数的取值范围为 1 到 20000，请确定具有最大素数因子的整数（请记住，素数只有因子 1 和自身，整数 7 是素数，而整数 6 可以被 2 和 3 整除，它不是素数）。
【输入形式】
第一行为单个整数 n，接下来共 n 个整数，每行包含一个整数。
【输出形式】
在一行中输出具有最大素数因子的那个整数，如果有多个，则输出最早出现在输入文件中的一个。
【样例输入】
4
36
38
40
42
【样例输出】
38
【样例说明】
测试数据的文件名为 `in.txt`
【评分标准】
该题目有 5 个测试用例，每通过一个测试用例，得 20 分。

答案：

```
#include<iostream>
using namespace std;
//定义函数prime，返回x的最大因子
int prime(int x)
{
    for(int i=x;i>=2;i--)
    {
```

```
8           //如果i是x的因子，就判断i是不是质数
9           if(x%i==0)
10          {
11              bool is_prime=true;
12              for(int j=2;j*j<=i;j++)
13                  if(i%j==0)
14                      is_prime=false;
15              if(is_prime)
16                  return i;
17          }
18      }
19      return 0;
20  }
21  int main()
22  {
23      int n,ans=0,max_prime=0;
24      //输入n，定义ans和max_prime存储当前答案
25      //和答案所拥有的最大因子
26      ifstream fin("in.txt");
27      //使用文件流从in.txt读取数据
28      fin>>n;
29      for(int i=1;i<=n;i++)
30      {
31          //读入n次数字x
32          int x;
33          fin>>x;
34          //从函数中获取x的最大因子
35          int new_prime=prime(x);
36          //如果x的最大因子比之前出现过数的最大因子更大
37          //就更新答案和最大因子
38          if(new_prime>max_prime)
```

```
39          {
40              ans=x;
41              max_prime=new_prime;
42          }
43      }
44      cout<<ans;
45      return 0;
46  }
```

解析：对于 n 个数据，需要找到其中所有数中拥有**最大质数因子**的一个。所以算法流程为：

1. 对每个数找出它的因子，并返回其中最大的质数。（函数 `int prime(x)`）
2. 比较所有返回值，拥有最大返回值，并最早出现的一个就是答案。

注意算法复杂度，如果发生时间超限，可以考虑优化算法。例如提前把所有质数存储在一个列表里。

习题 4 回文数

【问题描述】

判断一个整数是否是回文数。例如，121 是回文数，而 −121 不是回文数。

题目要求设计如下满足要求的函数：

```
class Solution {
public:
    bool isPalindrome(int x)
    {  ...  }
};
```

【输入形式】

每个测试用例输入一行，是一个整数。

【输出形式】

对于每个测试用例输出一行，输出 0 表示不是回文数，或者输出 1 表示是回文数。

【样例输入】

12345

【样例输出】

0

【样例说明】

测试用例中的每一行代表一个待测试的整数，测试数据的文件名为 in.txt。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```
class Solution {
    public:
        bool isPalindrome(int x)
        {
            //如果是负数，负号不会有对应，直接返回否
```

```
6               if(x<0)
7                   return false;
8               //定义可变数组/链表，每次放x的一位进去
9               vector<int> v;
10              //取出x的最低位，并除以10，就能取出所有位
11              while(x>0)
12              {
13                  v.push_back(x%10);
14                  x/=10;
15              }
16              //从前往后数第i位，应该对应从后往前第i位
17              //下标从0开始，到v.size()-1结束
18              for(int i=0;i<v.size();i++)
19                  if(v[i]!=v[v.size()-i-1])
20                      return false;
21              return true;
22          }
23      };
```

总结

题目：绪论

日期：2024 年 3 月 11 日

批改人：王骏峣

邮箱：wjyyy1@126.com

各位同学写填空题也需要注意过程，作业中的填空题如果答案错误也会酌情给分。

习题 1：选择题注意找题目中的关键词，并注意参考上课讲的内容。

习题 2：大部分同学没有分析对复杂度，建议使用数学推导，并使用代码验证。另外，课本中的 O 均为大写，代表复杂度上界，小写 o 有其他含义。本次作业中写小写 o 的同学被扣掉 1 分（该题满分 10 分）。

习题 3：注意本次提交需要用到文件输入，而输出是直接用 `cout` 或 `printf` 给到标准输出中的。对基础代码结构不太了解的同学需要注意复习程序设计，或在群里提问。

习题 4：注意本题需要处理 $n < 0$ 的情况。建议使用题目要求的格式进行代码补全。

数据结构 A
作业 2 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：顺序表与单链表
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

下列对顺序存储的有序表 (长度为 n) 实现给定操作的算法中平均时间复杂度为 $O(1)$ 的是（　　）。
A. 查找包含指定值元素的值
B. 插入包含指定值元素的算法
C. 删除第 i 个元素的算法
D. 获取第 i 个值的算法

答案：D

解析：顺序表（顺序存储）：方便随机访问，不方便插入和删除。
链表（链式存储）：方便插入和删除，不方便随机访问。
顺序存储的有序表即有序数组。

A 中，在有序数组中查找包含指定值的元素的位置最快的方法是用二分查找，时间复杂度为 $O(\log n)$。

B 中，在有序数组中查找包含指定值的元素最快的方法是首先用二分查找找到插入元素的位置，时间复杂度为 $O(\log n)$，然后把位于插入元素后面的元素的位置后移一格，平均时间复杂度为 $O(n)$。

C 中，删除第 i 个元素的最快的方法是首先利用顺序表支持随机访问的性质

找到第 i 个元素并删除，时间复杂度为 $O(1)$，然后把位于插入元素后面的元素的位置前移一格，平均时间复杂度为 $O(n)$。

D 中，获取第 i 个元素利用顺序表支持随机访问的性质找到第 i 个元素并访问即可。

习题 2 单选题

以下关于单链表叙述正确的是（　　）

i. 结点除自身信息以外还包括指针域，存储密度小于顺序表

ii. 找第 i 个结点的时间为 $O(1)$

iii. 在插入、删除运算时不必移动结点

A. 仅 i、iii

B. 仅 i、ii

C. 仅 ii、iii

D. 仅 i、ii、iii

答案：A

解析：i 项正确，顺序表存储时可以直接用下标记录相对位置，而单链表需要存储“下一个位置”的信息。

ii 项错误，找第 i 个结点的平均时间为 $O(n)$，因为单链表只能从表头开始查找 i 次，不支持随机访问。

iii 项正确，插入、删除运算时不必移动结点，只需要修改指针即可，原来的数据可以保持不变。

习题 3 设计链表

【问题描述】

请设计一个整数链表包含如下操作（含 n 个结点的链表中结点的序号或者索引为 $0 \sim n-1$）：

`1 val`：在链表的第一个元素之前添加一个值为 val 的结点，插入后新结点将成为链表的第一个结点。

`2 val`：将值为 val 的结点添加到链表的最后一个结点之后。

`3 i val`：在链表中的第 i 个结点之前添加值为 val 的结点。如果 i 等于链表的长度，则该结点将附加到链表的末尾。如果 i 大于链表长度，则不会插入结点。如果 i 小于 0，则在头部插入结点。

`4 i`：如果索引 i 有效，则删除链表中的第 i 个结点。

`5 i`：获取链表中第 i 个结点的值并且输出，如果索引 i 无效则返回 -1。

【输入形式】

每个测试用例由一系列的操作过程组成，以 -1 表示结束。

【输出形式】

对于每个测试用例，若测试用例中有操作 5，则在一行中输出对应的整数。

【样例输入】

```
1 1
2 2
2 3
2 4
2 5
2 7
3 5 6
5 0
5 1
5 2
5 3
5 4
5 5
5 6
4 0
4 1
5 1
5 10
-1
```

【样例输出】

```
1
2
3
4
5
6
7
4
-1
```

【样例说明】

测试用例中的每一行代表一个功能，第一个数字为功能编号，后面的数据为该功能的参数。比如 `3 5 6`，表示的是第 3 个功能，在链表中的第 5 个结点之前添加值为 6 的结点。测试数据的文件名为 `in.txt`。

【评分标准】

该题目有 5 个测试用例，每通过一个测试用例，得 20 分。

答案：

```
#include<iostream>
using namespace std;
struct LinkNode              //单链表结点类型
{
    int data;            //存放int类型数据元素
    LinkNode* next;              //下一个结点的指针
    LinkNode():next(NULL) {}     //构造函数
    LinkNode(int d):data(d),next(NULL) {}
};
int main()
{
    LinkNode *head=new LinkNode();
    int op,x;//输入操作符，如果为-1就停止
    ifstream fin("in.txt"); //使用文件流从in.txt读取数据
    fin>>op;
    while(op!=-1)
    {
        if(op==1)//op=1表示头插法
        {
            fin>>x;
            LinkNode* s=new LinkNode(x);
            s->next=head->next;
            head->next=s;
        }
        if(op==2)//op=2表示尾插法
        {
            fin>>x;
            LinkNode* p=head;
            while(p->next!=NULL)//直到找到尾节点
                p=p->next;
            p->next=new LinkNode(x);
        }
        if(op==3)//op=3表示在第i节点前插入
        {
            int i;
            fin>>i>>x;
            LinkNode* p=head;//找到第i-1节点
            for(int j=0;j<i&&(p!=NULL);j++)
                p=p->next;//超过尾节点也会停下
            if(p!=NULL)//如果超过尾节点，就不插入
```

```
            {
                LinkNode* s=new LinkNode(x);
                s->next=p->next;
                p->next=s;
            }
        }
        if(op==4)//删除第i节点
        {
            int i;
            fin>>i;
            LinkNode* p=head;//找到第i-1节点
            for(int j=0;j<i&&(p->next!=NULL);j++)
                p=p->next;//到达尾节点也会停下
            if(p->next!=NULL)//如果到达尾节点，就不操作
                p->next=p->next->next;
        }
        if(op==5)//输出第i节点
        {
            int i;
            fin>>i;
            LinkNode* p=head;
            for(int j=0;j<=i&&(p!=NULL);j++)
                p=p->next;//到达尾节点也会停下
            if(p!=NULL)//如果到达尾节点，就不操作
                cout<<p->data<<endl;
            else
                cout<<-1<<endl;
        }
        fin>>op;
    }
    return 0;
}
```

解析：参考课件第二讲中的 2.3.2 单链表，建立一个带头结点的单链表，然后根据输入的操作符进行操作。

操作 1 为头插法，操作 2 为尾插法，操作 3 为插入，操作 4 为删除，操作 5 为输出。注意尾节点不要越界，即 p 不为空时才能操作当前节点、p->next 不为空时才能操作下一节点。

考虑到本题中数据均为 int，可以把课件中的 <T> 去掉，直接用 int data 代替。或者按课件中写法，将 LinkNode<T> 换成 LinkNode<int>。

习题 4 两数之和

【问题描述】

给定一个整数数组 $nums$ 和一个目标值 $target$，请你在该数组中找出和为目标值的那两个整数，并返回他们的数组下标。例如，给定 $nums = [2, 7, 11, 15]$，$target = 9$，返回结果为 $[0, 1]$。若没有答案，则返回 $[-1, -1]$。题目要求设计 `twoSum()` 函数：

```
class Solution {
    public:
        vector<int> twoSum(vector<int>& nums, int
            target)
        {  ...  }
};
```

【输入形式】

每个测试用例由两行数据构成，第一行给出整数数组，元素之间以空格隔开，可以有重复元素；第二行给出求和的目标值。

【输出形式】

返回累加和为目标值的两个整数在数组中的下标值，不限顺序。若没有答案，则返回 $[-1, -1]$。

【样例输入】

```
2 7 11 15
9
```

【样例输出】

```
0 1
```

【样例说明】

输入的整数数组有四个整数，分别为 $2, 7, 11, 15$，目标值为 9。数组中的 2 和 7 的累加和为 9，因此返回这两个整数在数组中的下标地址，即 $0, 1$。测试数据文件名为 `in.txt`。

【评分标准】

共 10 个测试用例，每通过一个测试得 10 分。

答案：

```cpp
#include<iostream>
#include<vector>
#include<fstream>
#include<sstream>
using namespace std;
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        vector<int> ans;
        for(int i=0;i<nums.size();i++)
            for(int j=i+1;j<nums.size();j++)
                if(nums[i]+nums[j]==target)
                {
                    ans.push_back(i);
                    ans.push_back(j);
                    return ans;
                }
        ans.push_back(-1);
        ans.push_back(-1);
        return ans;
    }
};
int main()
{
    int n,x,target;
    vector<int> num;
    ifstream fin("in.txt");
    //使用文件流从in.txt读取数据
    string line;
    getline(fin,line);
    //因为要读入一行，所以使用getline函数，再用fin读入line这一字符串
    stringstream ss(line);
    //再将这一字符串转化为字符串流，用类似的>>方法读入数据
    while(ss>>x)
        num.push_back(x);
    //target只有一个数，所以直接用fin读入即可
    fin>>target;
    Solution s;
    vector<int> ans=s.twoSum(num,target);
    cout<<ans[0]<<" "<<ans[1];
    return 0;
}
```

解析：使用二重循环，找到两个相加等于 target 的元素即可，注意不要找到相同位置的数。

本题可以不使用 `vector` 返回结果，把解题过程写在主函数中更加简便。

对于输入流的常见用法，可以引入 `<fstream>`，使用 `ifstream fin("in.txt")` 定义 `in.txt` 对应的流。

使用 `fin>>x` 可以向 `x`（可为数值或字符串）读入一段数据，直到下一个空格/换行为止，因此可以读入以空格分开的数字或字符串。

使用 `getline(fin,line)` 可以向 `line`（应为字符串）读入一段数据，直到换行为止，这时一整行数据都在 `line` 中。

此时可以直接将 `line` 作为字符串利用，如果要再读入 `line` 中的数据，可以引入 `<sstream>`，使用 `stringstream ss(line)` 定义 `line` 对应的流。

总结

题目：线性表 1

日期：2024 年 3 月 29 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1、2：按书上和课件上内容理解单链表和顺序表的区别和使用。

习题 3：本题以单链表实现为主，可参考课件或书本，出现问题及时跟着报错信息进行尝试。部分同学出现了代码 90% 以上一致的情况，这些同学写完后一定要把这些内容落实为自己的知识。后续作业再出现完全一致的情况将扣除相应题目的分数。

习题 4：注意本题不要找到相同位置的数，总体较为容易。后续增加了文件输入输出的指导，在之后的作业题目中可以参考。感谢张智皓同学提出的建议。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 3 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：线性表
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 集合并集运算

【问题描述】
给你两个集合 A 和 B，要求 $\{A\}+\{B\}$。注意同一个集合中不会有两个相同的元素。
【输入形式】
每组输入数据分为三行，第一行有两个数字 n，m（$0 < n, m \leq 10000$），分别表示集合 A 和集合 B 的元素个数。后两行分别表示集合 A 和集合 B，每个元素为不超出 `int` 范围的整数，两个元素之间有一个空格隔开。
【输出形式】
针对每组数据输出一行数据，表示合并后的集合，要求从小到大输出，两个元素之间有一个空格隔开。
【样例输入】

```
1 2
1
2 3
```

【样例输出】

```
1 2 3
```

【样例说明】
第一行的两个数字表示集合 A 和集合 B 的元素个数，后面两行分别表示

集合 A 和集合 B 中的整数元素，两个元素之间用空格隔开。测试数据存放在 in.txt 文件中。

【评分标准】

共 10 个测试用例，每通过一个测试得 10 分。

答案：

```
#include<iostream>
#include<fstream>
using namespace std;
struct LinkNode             //单链表结点类型
{
    int data;            //存放int类型数据元素
    LinkNode* next;              //下一个结点的指针
    LinkNode():next(NULL) {}     //构造函数
    LinkNode(int d):data(d),next(NULL) {}
};
int main()
{
    LinkNode *ahead=new LinkNode();//链表a
    LinkNode *bhead=new LinkNode();//链表b
    int n,m,x;//输入操作符，如果为-1就停止
    ifstream fin("in.txt"); //使用文件流从in.txt读取数据
    fin>>n>>m;
    for(int i=1;i<=n;i++)//插入链表a
    {
        fin>>x;
        LinkNode *p=ahead;
        while(p!=NULL)
        {
            if(p->next!=NULL&&p->next->data<x)//说明还未找到比x大的位置
                p=p->next;
            else//找到比x大或链表末尾
            {
                LinkNode *t=new LinkNode(x);
                t->next=p->next;
                p->next=t;
                break;
            }
        }
```

```
34      }
35      for(int i=1;i<=m;i++)//插入链表b
36      {
37          fin>>x;
38          LinkNode *p=bhead;
39          while(p!=NULL)
40          {
41              if(p->next!=NULL&&p->next->data<x)//说明还未找到比x大的位置
42                  p=p->next;
43              else//找到比x大或链表末尾
44              {
45                  LinkNode *t=new LinkNode(x);
46                  t->next=p->next;
47                  p->next=t;
48                  break;
49              }
50          }
51      }
52      ahead=ahead->next;//把表头向后移动
53      bhead=bhead->next;//将两组中最小的数据进行比较
54      while(ahead!=NULL&&bhead!=NULL)
55      {
56          if(ahead->data==bhead->data)//相等时同时输出并后移
57          {
58              cout<<ahead->data<<" ";
59              ahead=ahead->next;
60              bhead=bhead->next;
61          }
62          else if(ahead->data<bhead->data)//不相等时输出较小的一个
63          {
64              cout<<ahead->data<<" ";
65              ahead=ahead->next;
66          }
67          else
68          {
69              cout<<bhead->data<<" ";
70              bhead=bhead->next;
71          }
72      }
73      while(ahead!=NULL)//清空剩余的链表
74      {
75          cout<<ahead->data<<" ";
```

```
76          ahead=ahead->next;
77      }
78      while(bhead!=NULL)
79      {
80          cout<<bhead->data<<" ";
81          bhead=bhead->next;
82      }
83      return 0;
84  }
```

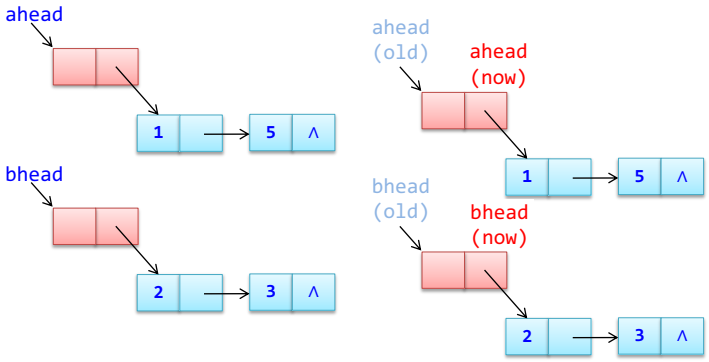
解析：参考课件第三讲中的 2.5.2 节例题，对输入的集合进行排序后，通过二路归并。

注意输入数据不一定是有序的，所以在插入的时候需要按顺序插入。或者对构建好的链表进行插入排序。

此外还可以使用 STL 中的 `list` 容器或 `vector` 容器，输入后使用 `sort()` 函数进行排序。排好序之后使用二路归并得到结果输出即可。

具体二路归并就是指将两个序列进行排序之后，每次找**两个序列的最小值中更小的那个**，并将其原先队列的指针后移一位。

而因为一开始头指针自身不包含 data 数据，需要将自己移到下一个位置才能开始进行处理，如下图所示。



习题 2 归并排序

【问题描述】

有一个含 n（$n \leq 200000$）个整数的无序序列，采用链表的二路归并排序实现递增排序。

【输入形式】

一行字符串，包含多个整数，每个数之间用空格分开。

【输出形式】

递增排序的结果，每个数之间用空格分开。

【样例输入】

```
9 4 7 6 2 5 8 1 3
```

【样例输出】

```
1 2 3 4 5 6 7 8 9
```

【样例说明】

测试数据的文件名为 in.txt，输出文件名为 out.txt。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例得 10 分

答案：

```cpp
#include<iostream>
#include<fstream>
using namespace std;
struct LinkNode              //单链表结点类型
{
    int data;            //存放int类型数据元素
    LinkNode* next;              //下一个结点的指针
    LinkNode():next(NULL) {}     //构造函数
    LinkNode(int d):data(d),next(NULL) {}
};
void merge_sort(LinkNode* head,int len)//传入链表头指针和长度，进行归并
{
    //如果len为1，则说明不需要再排序
    if(len==1)
        return;
    //将链表划分为两个长度相近的部分，将复杂度尽快减少
    int half=len/2;
```

```
18      LinkNode* p=head;//将p定位到第half个元素
19      for(int i=1;i<=half;i++)
20          p=p->next;
21      LinkNode* left_head=new LinkNode();
22      LinkNode* right_head=new LinkNode();
23      //让righthead指向第half+1个位置
24      //并截断前面一半的链表
25      left_head->next=head->next;
26      right_head->next=p->next;
27      p->next=NULL;
28      //接下来对两个链表分别递归排序
29      merge_sort(left_head,half);
30      merge_sort(right_head,len-half);
31      //两个子排序结束后，此时两个head指向的链表均为有序
32      LinkNode* new_head=new LinkNode();
33      LinkNode* new_tail=new_head;
34      //接下来开始二路归并即可（参考上一题）
35      left_head=left_head->next;//把表头向后移动
36      right_head=right_head->next;//将两组中最小的数据进行比较
37      while(left_head!=NULL&&right_head!=NULL)
38      {
39          if(left_head->data<=right_head->data)
40          {
41              new_tail->next=new LinkNode(left_head->data);
42              new_tail=new_tail->next;//使用尾插法插入左边元素
43              left_head=left_head->next;
44          }
45          else
46          {
47              new_tail->next=new LinkNode(right_head->data);
48              new_tail=new_tail->next;//使用尾插法插入右边元素
49              right_head=right_head->next;
50          }
51      }
52      while(left_head!=NULL)
53      {
54          new_tail->next=new LinkNode(left_head->data);
55          new_tail=new_tail->next;//使用尾插法插入左边元素
56          left_head=left_head->next;
57      }
58      while(right_head!=NULL)
59      {
```

```
60          new_tail->next=new LinkNode(right_head->data);
61          new_tail=new_tail->next;//使用尾插法插入右边元素
62          right_head=right_head->next;
63      }
64      //最后将head指向new_head才能结束
65      head->next=new_head->next;
66  }
67  int main()
68  {
69      LinkNode *head=new LinkNode();//创建链表
70      int n=0,x;//输入链表，统计长度
71      ifstream fin("in.txt"); //使用文件流从in.txt读取数据
72      ofstream fout("out.txt"); //使用文件流将数据输出到out.txt
73      while(fin>>x)//使用头插法插入链表
74      {
75          LinkNode *p=new LinkNode(x);
76          p->next=head->next;
77          head->next=p;
78          n++;
79      }
80      merge_sort(head,n);//递归进行归并排序
81      head=head->next;
82      while(head!=NULL)
83      {
84          fout<<head->data<<" ";
85          head=head->next;
86      }
87      return 0;
88  }
```

解析：归并排序指的是将一段数列以中间为界限分为两部分，然后对两部分分别进行排序，最后将两部分合并。

归并排序分为三个步骤：

1. 将数列划分为两部分；2. 递归地分别对两个子序列进行归并排序；3. 合并两个子序列。

因为合并两个有序子序列的复杂度很低，所以通过归并排序可以在$O(n\log n)$的时间内完成排序任务。

而此次作业我们输入的是链表，所以在对链表进行拆分的时候需要从中间断开，然后将两个表头传入函数，分别进行排序。

习题 3 归并排序

【问题描述】

有 M 个人，编号分别为 1 到 M，玩约瑟夫环游戏，最初时按编号顺序排成队列；每遍游戏开始时，有一个正整数报数密码 N，队列中人依次围坐成一圈，从队首的人开始报数，报到 N 的人出列，然后再从出列的下一人开始重新报数，报到 N 的人出列；重复这一过程，直至所有人出列，完成一遍游戏，所有出列的人形成新队列；游戏可能玩很多遍，每遍有新报数密码。求若干遍游戏完成后队列次序。本题要求使用单链表实现，程序要求采用模块化设计，格式规范，有合适注解。

【输入形式】

每个测试用例包含若干个正整数（至少 1 个），第一个正整数为玩游戏人数 M，后续每个正整数为每遍游戏报数密码；报数密码可能为 1。

【输出形式】

每个测试用例结果占一行，每个编号占 4 位。

【样例输入】

```
10 3 5 2
```

【样例输出】

```
   4   6   5   2   9   1   3   7   8  10
```

答案：

```cpp
#include<iostream>
#include<fstream>
using namespace std;
struct LinkNode              //单链表结点类型
{
    int data;            //存放int类型数据元素
    LinkNode* next;              //下一个结点的指针
    LinkNode():next(NULL) {}     //构造函数
    LinkNode(int d):data(d),next(NULL) {}
};
int main()
{
    LinkNode *head=new LinkNode();//创建链表
    LinkNode *tail=head;//链表尾指针
    int m,n;//输入链表，统计长度
```

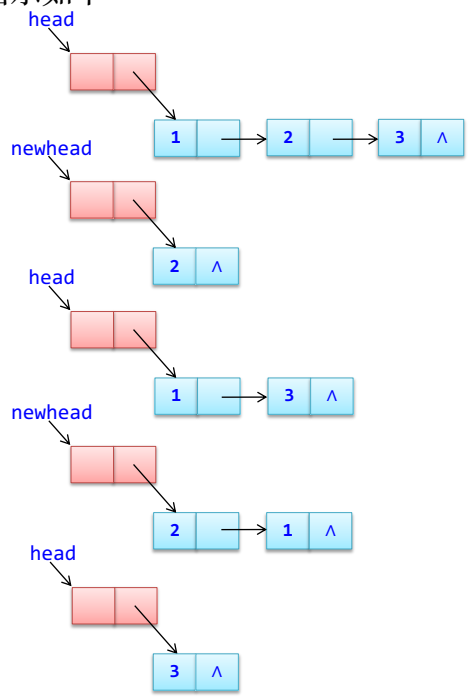
```
16      cin>>m;
17      for(int i=1;i<=m;i++)
18      {
19          tail->next=new LinkNode(i);
20          tail=tail->next;//使用尾插法
21      }
22      while(cin>>n)//使用头插法插入链表
23      {
24          LinkNode *p=head;
25          LinkNode *new_head=new LinkNode();
26          LinkNode *new_tail=new_head;//创建新的环，仍使用尾插法
27          while(head->next!=NULL)//说明环内仍有数字
28          {
29              //跳转n-1次
30              for(int i=1;i<n;i++)
31              {
32                  if(p->next!=NULL)//跳向p->next
33                      p=p->next;
34                  else
35                      p=head->next;
36              }
37              //跳n次之后p->next就是要被删掉的
38              if(p->next!=NULL)//尾插法将p->next插入新链表
39              {
40                  new_tail->next=new LinkNode(p->next->data);
41                  new_tail=new_tail->next;
42                  p->next=p->next->next;
43              }
44              else//然后删掉p->next
45              {
46                  new_tail->next=new LinkNode(head->next->data);
47                  new_tail=new_tail->next;
48                  head->next=head->next->next;
49              }
50  //              cout<<new_tail->data<<" ";
51          }
52          head=new_head;
53  //          cout<<endl;
54      }
55      head=head->next;
56      while(head!=NULL)
57      {
```

```
58          cout<<setw(4)<<head->data;//保证宽度为4
59          head=head->next;
60      }
61      return 0;
62  }
```

解析：本题需要模拟约瑟夫环，难点主要在于删除相应节点并拼接到新的链表上。

每次需要报数时就新建一个链表，报到第 n 个数字就删除当前指向的节点。因为要删除某个节点，所以应该保留**“要删除的点的”**上一个指针。此时把**要删除的点**用尾插法加入新的链表以便下一次报数。

图示如下：



总结

题目：线性表 2

日期：2024 年 4 月 5 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：根据对题目意思的理解，对链表进行排序。然后合并两个有序链表需要用到归并的思想，这个较为容易（同时也出现在了第一次实验里，有细微不同）。

习题 2：延续习题 1 的归并思想，本题需要降低复杂度，关于归并排序的具体做法可以自行学习原理，或参考对数组的排序过程，最终写出链表的归并排序。部分同学对函数的理解不够具体，建议加强复习。

习题 3：注意本题是标准输入输出（不涉及文件），本地调试时可以阶段性输出来检验每次约瑟夫环的更新是否正确（参考答案中被注释掉的输出语句）。

在之后的作业中，如果具体实现方法与题目要求有出入但获得了分数（如使用链表的题目用了数组模拟），会酌情进行扣减，望大家注意。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 4 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：栈
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

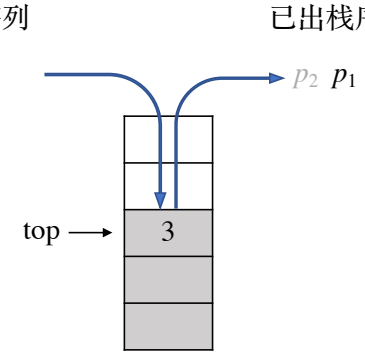
一个栈的入栈序列为 1，2，3，...，n，其出栈序列为 p_1，p_2，...，p_n，若 $p_2 = 3$，则 p_3 可能取值的个数是（　　）。

A. $n - 1$
B. $n - 2$
C. $n - 3$
D. 无法确定

答案：A

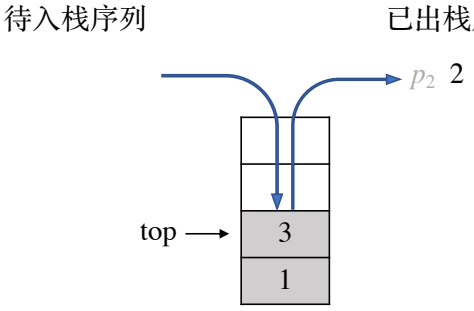
解析：在栈内，已知 $p_2 = 3$，说明**第二个出栈的元素是 3**。此时一定可以确定的是，**1，2 已经入栈过**，而 $4 \sim n$ 的入栈情况未知。

图中左侧待入栈序列中的数字一定 > 3。

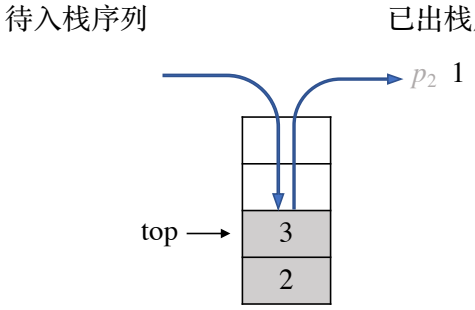


因此只要在 3 出栈后，放入任意个后面的数，都可以做到在 3 的下一个出栈，所以 $4 \sim n$ 都可能作为 p_3 出现。

此时再考虑 $1, 2$ 是否有可能作为 p_3。如果 2 作为 p_3，则为下图的情况，1 入栈、2 入栈、2 出栈。



如果 1 作为 p_3，则为下图的情况，1 入栈、2 入栈、2 出栈。



所以 $1 \sim 2, 4 \sim n$ 都有可能是 p_3。

习题 2 单选题

由两个栈共享一个数组空间的好处是（　　）。

A. 减少存取时间，降低上溢出发生的几率

B. 节省存储空间，降低上溢出发生的几率

C. 减少存取时间，降低下溢出发生的几率

D. 节省存储空间，降低下溢出发生的几率

答案：B

解析：上溢指的是栈满，共享栈空间表示两个栈分别占据一段空间的两端。因此可以互相调节存储空间，只有两栈都满时才会发生上溢，所以降低了上溢的发生几率。

习题 3 多选题

假如栈的入栈顺序是 a, b, c, d，下面 4 个选项中可能是它的出栈顺序的是（　　）。

A. a, c, b, d

B. b, c, d, a

C. c, d, b, a

D. d, c, a, b

答案：ABC

解析：参考本次习题 1 和例 3.6。一方面可以使用代码验证，另一方面可以自行画图验证。当我们有 $p_1, ..., p_{i-1}$ 时，要让下一个出栈的元素是 p_i，可能进行的操作是唯一的。

定义入栈序列为 a，出栈序列为 b，模拟一个栈。

若栈空，则无法操作出栈，这时将 a 的下一个元素入栈。

否则，检验 b 中下一个元素，相等时可以直接出栈，若不相等则只能继续往栈内加入元素。

当入栈序列已空，仍未匹配到出栈序列的下一个元素，则说明序列不可能出现。

A：a 入，a 出，b 入，c 入，c 出，b 出，d 入，d 出。

B：a 入，b 入，b 出，c 入，c 出，d 入，d 出，a 出。

C：a 入，b 入，c 入，c 出，d 入，d 出，b 出，a 出。

D：a 入，b 入，c 入，d 入，d 出，c 出，b 出**不来**，因此不选。

习题 4 填空题

把中缀表达式 `3+(5-2)*6` 转化为后缀表达式时，需要的顺序栈容量至少是（　　），得到的后缀表达式是（　　）（用 `#` 表示一个数字串结束）。

答案：3，`3#5#2#-6#*+`

解析：参考课本 3.1.7 部分，模拟入栈顺序。设 *postexp* 是后缀表达式。

遍历中缀表达式，遇到数字符，则将连续的数字符末尾加上“#”后添加到 *postexp* 后面；遇到“(”，将其进栈；遇到“)”，退栈运算符并添加到 *postexp*，直到

退栈的是“(”为止（该左括号不添加到 *postexp* 中）；

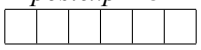
遇到运算符 op_2，将其跟栈顶运算符 op_1 的优先级进行比较，只有当 op_2 的优先级高于 op_1 的优先级时才直接将 op_2 进栈，否则将栈中“(”（如果有）之前[1]的优先级等于或高于 op_2 的运算符均退栈并添加到 *postexp*，再将 op_2 进栈。

模拟这一过程，栈内元素如下所示：

1. 遇到数字 **3**，直接添加到后缀表达式 *postexp*

原始串：**3**+(5-2)*6

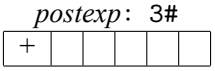
postexp：3#



2. 遇到运算符 +，发现栈内没有符号，可以进栈

原始串：3+(5-2)*6

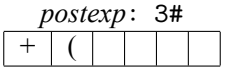
postexp：3#



3. 遇到左括号 **(**，将其进栈

原始串：3+**(**5-2)*6

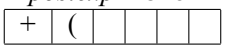
postexp：3#



4. 遇到数字 **5**，直接添加到后缀表达式 *postexp*

原始串：3+(**5**-2)*6

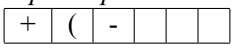
postexp：3#5#



5. 遇到运算符 **-**，发现栈顶是左括号，直接入栈

原始串：3+(5**-**2)*6

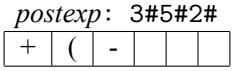
postexp：3#5#



6. 遇到数字 **2**，直接添加到后缀表达式 *postexp*

原始串：3+(5-**2**)*6

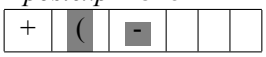
postexp：3#5#2#



7. 遇到右括号**)**，退栈，将运算符添加到后缀表达式，直到下一个左括号

原始串：3+(5-2)**)***6

postexp：3#5#2#-



栈内元素数量最多为 3，后缀表达式：3#5#2#-6#*+

[1]这里的“之前”指遇到括号之前，也就是在栈顶操作

8. 遇到运算符 *，发现比栈顶的 + （op_1）优先级高，将运算符 * 入栈
原始串：3+(5-2)*6
postexp：3#5#2#-

+	*				

9. 遇到数字 **6**，直接添加到后缀表达式
原始串：3+(5-2)*6
postexp：3#5#2#-6#

+	*				

10. 最后依次出栈，得到后缀表达式
postexp：3#5#2#-6#*+

习题 5 逆波兰表达式求值

【问题描述】
根据逆波兰表示法，求表达式的值。有效的运算符包括 +，-，*，/。每个运算对象可以是整数，也可以是另一个逆波兰表达式。假设给定逆波兰表达式总是有效的，换句话说，表达式总会得出有效数值且不存在除数为 0 的情况。其中整数除法只保留整数部分。
【输入形式】
每个样例是一行，为有效的表达式，每个数字和运算符号之间用“,”隔开
【输出形式】
表达式的计算结果。
【样例输入】
2,1,+,3,*
【样例输出】
9
【样例说明】
测试数据的文件名为 in.txt。
【评分标准】
该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include<iostream>
#include<fstream>
#include<stack>
using namespace std;
```

```
stack<int> s;
int main()
{
    ifstream in("in.txt");
    char ch;
    while(in>>ch)
    {
        if(ch>='0'&&ch<='9')//此时读入的是数字
        {
            int x=ch-'0';//将数字存入x
            in>>ch;
            while(ch!=',')//数字结束后会有逗号承接下一个内容
            {
                x=x*10+ch-'0';//读入x的下一位
                in>>ch;
            }
            s.push(x);
        }
        else//输入的最后一个内容一定是运算符
        {
            if(ch=='+')
            {
                int x=s.top();//取出栈顶两个元素
                s.pop();
                int y=s.top();
                s.pop();
                s.push(x+y);
            }
            else if(ch=='-')
            {
                int x=s.top();
                s.pop();
                int y=s.top();
                s.pop();
                s.push(y-x);//注意减数关系
            }
            else if(ch=='*')
            {
                int x=s.top();
                s.pop();
                int y=s.top();
                s.pop();
```

```
47                  s.push(x*y);
48              }
49              else if(ch=='/')
50              {
51                  int x=s.top();
52                  s.pop();
53                  int y=s.top();
54                  s.pop();
55                  s.push(y/x);//保留整数
56              }
57              else if(ch!=',')//如果是逗号，说明还有内容，否则没有
58                  break;
59          }
60      }
61      cout<<s.top();//最后剩余一个栈内元素
62      return 0;
63  }
```

解析：逆波兰表达式求值的过程是模拟栈的运算过程，遇到数字直接入栈，遇到运算符则将栈顶两个元素出栈进行运算，将结果入栈。

需要注意的是，当运算符是减法和除法时，栈顶的元素是被减数/被除数。

本答案利用了对字符的读入，将字符转化为数字，再将数字入栈。在读入数字时，需要注意数字可能是多位数，因此需要循环读入直到遇到逗号。

习题 6 合并栈操作

【问题描述】

栈是一种具有后进先出的数据结构。可合并栈是支持“merge”操作的栈。三种操作的说明如下：

1. push A x：将 x 插入栈 A 中。
2. pop A：删除栈 A 的顶部元素。
3. merge A B：合并栈 A 和 B。

其中，“merge A B”操作后栈 A 包含 A 和 B 之前的所有元素，B 变为空，新栈中的元素根据先前的进栈时间重新排列，就像在一个栈中重复“push”操作一样。给定两个可合并栈 A 和 B，请执行上述操作。

【输入形式】

测试用例的第一行包含一个整数 $n(0 < n \leq 10^5)$ 表示操作个数，接下来的 n 行每行包含一条指令 push、pop 或 merge，栈元素是 32 位整数。A 和 B 最初都是空的，并且保证不会对空栈执行 pop 操作。以 $n = 0$ 表示输入结束。

【输出形式】

对于每个 pop 操作，在一行中输出对应的出栈元素。

【样例输入】

```
9
push A 0
push A 1
push B 3
pop A
push A 2
merge A B
pop A
pop A
pop A
```

【样例输出】

```
1
2
3
0
```


【样例说明】

测试数据的文件名为 in.txt。

【评分标准】

该题目有 5 个测试用例，每通过一个测试用例，得 20 分。

答案:

```cpp
#include<iostream>
#include<fstream>
#include<stack>
using namespace std;
stack<pair<int,int>> A,B;
int main()
{
    ifstream in("in.txt");
    string op;
    int n;
    in>>n;
    for(int i=1;i<=n;i++)
    {
        in>>op;
        if(op=="push")//需要两个输入
        {
            string s;
            int x;
            in>>s>>x;
            if(s=="A")//放入时间戳和元素
                A.push(make_pair(i,x));
            else
                B.push(make_pair(i,x));
        }
        else if(op=="pop")
        {
            string s;
            in>>s;
            if(s=="A")
            {
                cout<<A.top().second<<endl;
                A.pop();//注意元素存储在second里
            }
            else
            {
                cout<<B.top().second<<endl;
                B.pop();
            }
        }
        else//merge
```

```
41          {
42              string x,y;
43              in>>x>>y;
44              stack<pair<int,int>> C;//新栈，放入时是倒序
45              while(!A.empty()&&!B.empty())
46              {
47                  //倒序，所以时间戳大的先入C
48                  if(A.top().first<B.top().first)
49                  {
50                      C.push(B.top());
51                      B.pop();
52                  }
53                  else
54                  {
55                      C.push(A.top());
56                      A.pop();
57                  }
58              }
59              while(!A.empty())//归并的剩余部分
60              {
61                  C.push(A.top());
62                  A.pop();
63              }
64              while(!B.empty())
65              {
66                  C.push(B.top());
67                  B.pop();
68              }
69
70              while(!C.empty())//把C倒回A
71              {
72                  if(x=="A")//B变空
73                      A.push(C.top());
74                  else
75                      B.push(C.top());
76                  C.pop();
77              }
78          }
79      }
80      return 0;
81  }
```

解析：栈是先进后出的，但是同一时刻在栈内的元素，一定满足越靠近栈底，入栈时间越早。因此可以利用之前学过的**归并**进行 `merge` 操作。

所以使用 `pair` 存储栈内元素，`pair` 的第一个元素是入栈时间，第二个元素是栈内元素。在合并栈时，将两个栈的元素合并到一个新的栈中，按照入栈时间排序即可。

注：`pair` 是一种 STL 标准库自带的“二元结构体”。它的两个内部变量分别叫 `first` 和 `second`。

总结

题目：栈

日期：2024 年 4 月 19 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：可以考虑自行赋值 $p_2 = 3$，逐个尝试 p_3 检验是否可能出现相应的值。错的同学还是有一些，建议自行画图验证。

习题 2：按书上例 3.7 理解。注意上溢下溢的概念区分。

习题 3：手动模拟和代码验证都可以尝试，考试时鼓励使用代码验证与手动模拟互相核对。

习题 4：仍然是对书上例题的理解，强烈建议手动模拟计算过程。如果觉得麻烦可以标注每个变量的变化过程。

习题 5：难点在读入多位数，可以参考答案练习不同类型的数据输入（习题 6 也涉及字符串的读入）。

习题 6：入栈时间戳是关键思想。作业中一般对大家复杂度没有太高要求，但是可以时常联想归并的做法。归并是常用的合并有序数列的低复杂度做法。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 5 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：队列
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

已知环形队列存储在一维数组 $A[0..n-1]$ 中，且队列非空时 *front* 和 *rear* 分别指向队头元素和队尾元素。若初始时队列空，且要求第一个进入队列的元素存储在 $A[0]$ 处，则初始时 *front* 和 *rear* 的值分别是（　　）

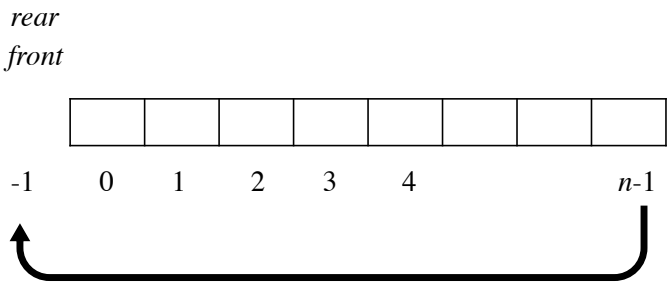
A. 0，0
B. 0，$n-1$
C. $n-1$，0
D. $n-1$，$n-1$

答案：B

解析：题目要求：若初始时队列空，且要求第一个进入队列的元素存储在 $A[0]$ 处。

而参考书上对队列的定义，*front* 指向队首元素的前一个位置，队尾指针 *rear* 指向队尾元素。本题要求 *front* 指向队首元素，队尾指针 *rear* 指向队尾元素。所以初始状态下应有 $front = rear + 1$。

队列的下一个元素会被插入队尾 $rear+1$ 的位置，因此可以确定 $rear = n-1$。又因为队列为空，所以 $front = rear + 1$。此时循环需要对 $MaxSize$ 取余，所以 $front = 0$。



习题 2 单选题

设环形队列的存储空间是 $a[0..20]$，且当前队头指针（f 指向队首元素的前一个位置）和队尾指针（r 指向队尾元素）的值分别是 8 和 3，则该队列中的元素个数为（　　）

A. 5

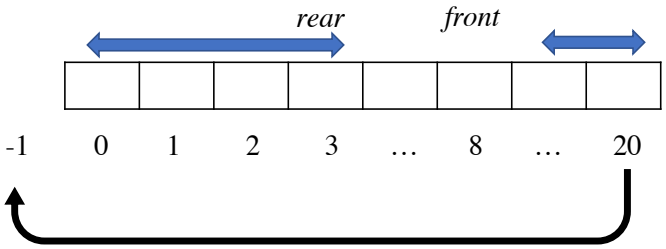
B. 6

C. 16

D. 17

答案：C

解析：$f=8, r=3$，从 f 的下一个位置开始排布元素，直到 r 的位置，因此 $[9,20]$ 和 $[0,3]$ 都是有值的。一共为 $(20-9+1)+(3-0+1)=16$ 个。



习题 3 单选题

栈和队列的共同特点是（　　）。

A. 都是先进后出

B. 都是先进先出

C. 只允许在端点处插入和删除元素

D. 没有共同点

答案：C

解析：栈是先进后出，队列是先进先出。在脑海里模拟两个管道，其中一个是单边出入的，另一边被封起来；另一个是一边入一边出的。

习题 4 填空题

循环队列 $A[0,10]$，存储容量是 10，队首指针 f，队尾指针 r，初始状态（　　）。假设经过一系列入队和出队操作后，$r=f=5$，然后又出队 1 个元素，则现在循环队列中的元素个数为（　　）

答案：$r=f$，9 或 0（下溢错误）

解析：第二问答案给的是 9 或 0，我认为 9 更对一些，只填 9 的同学我给了满分，而只填 0 的同学我扣掉了一分。

首先初始状态应该填的是 f,r 两变量的状态。要注意本题是循环队列，初始状态下 $f=r$，默认赋值为 0。因此写 $f=r$、$f=0,r=0$ 的同学都算对。

$r=f=5$ 时，考虑两种情况：队空或队满（恰好 10 个元素）。如果队满，**又出队 1 个元素**则剩下 9 个。如果队空，严格意义上不能**又出队 1 个元素**，但是如果一定要再出的话那就仍然是空队列。

因此本题第二个空填 9 或 0。如果要填 0 可以标注“下溢错误”。

习题 5 团队队列

【问题描述】

在团队队列中每个成员都属于一个团队，如果一个成员进入队列，它首先从头到尾搜索队列，以检查它的一些队友（同一队的成员）是否已经在队列中，如果是，它会进入到该团队的后面，如果不是，它会从尾部进入队列并成为新的最后一个成员。成员出队是按常规队列操作，按照出现在队

列中的顺序从头到尾进行处理。你的任务是编写一个模拟这样的团队队列的程序。

【输入形式】

每个测试用例都以团队个数 t 开始（$1 \leq t \leq 1000$），然后是团队描述，每个描述包含属于团队的成员个数和成员编号列表，成员编号为 0 到 999999 之间的整数，一个团队最多可以包含 1000 个成员。然后是一系列命令，有三种不同的命令：

1. ENQUEUE p：成员 p 进入队列。
2. DEQUEUE：队列中第一个成员出来并将其从队列中删除。
3. STOP：当前测试用例结束。

【输出形式】

对于每个 DEQUEUE 命令，以单独一行输出出队的成员。

【样例输入】

```
2
3 101 102 103
3 201 202 203
ENQUEUE 101
ENQUEUE 201
ENQUEUE 102
ENQUEUE 202
ENQUEUE 103
ENQUEUE 203
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
STOP
```

【样例输出】

101
102
103
201
202
203

【样例说明】

测试数据的文件名为 in.txt。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include<iostream>
#include<fstream>
#include<queue>
using namespace std;
queue<int> Q;
queue<int> q[1010];
int team[1000000];
//team[i]表示i属于哪个队伍
int main()
{
    ifstream in("in.txt");
    string op;
    int t;
    in>>t;
    for(int i=1;i<=t;i++)
    {
        int n,x;
        in>>n;
        for(int j=1;j<=n;j++)
        {
            in>>x;
            team[x]=i;
        }
    }
    in>>op;
    while(op!="STOP")
```

```
    {
        if(op=="ENQUEUE")
        {
            int x;
            in>>x;
            if(q[team[x]].empty())
            //或写成 tt=team[x] 并判断 q[tt] 是否为空也可以
                Q.push(team[x]);
            q[team[x]].push(x);
        }
        else//否则就是DEQUEUE
        {
            //DEQUEUE的内容一定是Q里第一个团队的第一个元素
            cout<<q[Q.front()].front()<<endl;
            q[Q.front()].pop();
            if(q[Q.front()].empty())//如果这个团队没有元素了，就出队
                Q.pop();
        }
        in>>op;
    }
    return 0;
}
```

解析：本题更适合用链表写，插入元素时可以依次从前到后找到队列中是否有相同团队的元素，如果有则插入到相同团队的最后，否则插入到队列尾部。

但是因为团队之间不会相互干扰，所以可以将每个团队分开进行排列，并额外使用一个队列 Q 存储团队之间的顺序。每次插入成员时，先检查所属团队队列中是否有元素，如果有则直接插入到团队队列的最后，否则将团队插入到额外队列尾部，再将元素插入团队队列。

当一个团队队列被清空时，它也要被从额外队列中被移除了。

习题 6 扔钉子

【问题描述】

年度学校自行车比赛开始了，ZL 是这所学校的学生，他太无聊了，因为他不能骑自行车！因此，他决定干预比赛，他通过以前的比赛视频获得了选手的信息，一个选手第一秒可以跑 F 米，然后每秒跑 S 米。每个选手有一条直线跑道，ZL 每秒向跑的最远的运动员跑道扔一个钉子，在自行车胎爆炸之后，该选手将被淘汰。如果有多个选手是 NO.1，则他总是选择 *ID* 最小的选手扔钉子。

【输入形式】

每个测试用例的第一行包含一个整数 $n(1 \leq n \leq 50000)$，表示选手人数，然后跟随 n 行，每行包含第 i 个选手的两个整数 $F_i(0 \leq F_i \leq 500)$，$S_i(0 < S_i \leq 100)$，表示该选手第一秒可以跑 F_i 米，然后每秒跑 S_i 米，i 是玩家从 1 开始的 *ID*。

【输出形式】

输出 n 个数字，以空格分隔，第 i 个数字是选手的 *ID*，该选手将在第 i 秒结束时被淘汰。

【样例输入】

```
3
100 1
100 2
3 100
```

【样例输出】

```
1 3 2
```

【样例说明】

测试数据的文件名为 `in.txt`。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include<iostream>
#include<fstream>
#include<vector>
```

```cpp
#include<queue>
using namespace std;
struct player
{
    int start,id;
    friend bool operator <(player a,player b)
    {
        if(a.start!=b.start)//如果可以通过速度区分，就返回速度大小关系
            return a.start<b.start;
        return a.id>b.id;//如果速度相等，就返回id的反向关系
    }
};
priority_queue<player> q[105];//q[i]包含了速度为i的所有人
int main()
{
    ifstream in("in.txt");
    int n;
    in>>n;
    for(int i=1;i<=n;i++)
    {
        player x;
        int speed;
        in>>x.start>>speed;
        x.id=i;
        q[speed].push(x);
    }
    for(int i=0;i<n;i++)//i=0表示第1秒。第i+1秒时每个人的位置是f+s*i
    {
        int farthest_j=0;
        int dis=0;
        for(int j=1;j<=100;j++)
            if(!q[j].empty())
            {
                player now=q[j].top();
                if(now.start+i*j>dis)
                {
                    dis=now.start+i*j;
                    farthest_j=j;
                }
                else if(now.start+i*j==dis)
                {//距离相等，则比较id
                    if(now.id<q[farthest_j].top().id)
```

```
                        farthest_j=j;
                }
            }
        //最后farthest_j队伍里跑得最远的会被淘汰
        cout<<q[farthest_j].top().id<<" ";
        q[farthest_j].pop();
    }
    return 0;
}
```

解析：根据题意，S_i 的取值只有 100 种。因此 S_i 相同的人的前后位置是相对固定的，淘汰顺序也因此固定。所以可以将 S_i 相同的选手放入同一个排好序的队列 q_{S_i}，可以使用优先队列 `priority_queue`。

每次被淘汰的人一定是从所有队列 q 的最大值中再选出它们的最大值。所以每次将所有优先队列的最大值进行比较，把最大的弹出。

举例理解一下就是现在有 100 个已经从前到后排好序的队伍，我们要找所有人里面最靠前的一个，那么这一个人一定在它自己队列里也是最靠前的。因此我们只需要每次找出所有队伍里面最靠前的一个，当把它删掉（弹出优先队列）时，它后面的元素会自动补位，而优先队列就可以做到这一点。

此处涉及两个关键字的比较，可以用结构体**重载运算符**来解决，也可以使用 **pair**，用法可以参考程序设计课程。如果对代码细节有问题可以咨询助教。

colback=cyan!10,colframe=cyan!70!blue,

题目：队列

日期：2024 年 4 月 21 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：注意对概念的理解，相信大家都能够掌握队列的用法，但是如果考试考概念的话还是要注重细节。感谢黄冰豪同学对本题解析的指正。

习题 2：注意存储空间最末元素具体是哪一个，同时也要注意循环队列和普通队列的区别。

习题 3：同样是对概念的理解，如果这样的性质题目想不清楚就多画图来帮助理解。

习题 4：填空题只有写了我才能想办法给大家一点思考分，不写的话我是无法给分的。只填 9 的同学可以认为是思考过循环队列的，这题可以得分。填空题看出有思考的同学我会酌情加分。（但是好像跟整次作业比起来，一分的差距不是很大）

习题 5：直接对题目模拟是一种做法，但是可能有超时的风险。如果能用更聪明更简单的做法达到目的，更鼓励大家尝试简便做法。在作业截止前提交次数是不受限制的，可以多次提交来验证自己的不同思路。

习题 6：此题和 https://www.luogu.com.cn/problem/U31965 有很相似的地方，有余力的同学可以考虑做一下并体会。题目难度有一点点大，需要多思考并和之前学过的内容进行结合。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 6 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：串
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

以下关于串的叙述中正确的是（　　）
A. 串是一种特殊的线性表
B. 串中元素只能是字母
C. 空串就是空白串
D. 串的长度必须大于零

答案：A

解析：A 选项，串实际上可以认为是“字符数组”，也就是一种线性表，它比线性表的要求更严格。

B 选项，字符包含“字母、数字和其他字符”，这里的其他字符可以是标点符号等。同时广义的字符也可以是汉字。

C 选项，空串是长度为 0 的串，而空白串可以指的是**只包含空格的串**。

D 选项，串的长度可以为 0，也就是空串。

习题 2 单选题

两个字符串相等的条件是（　　）

A. 串的长度相等

B. 含有相同的字符集

C. 串的长度相等并且对应的字符相同

D. 串的长度和对应字符都一致并且存储结构相同

答案：C

解析：串只关注内容，不关注存储结构。只需要保证逐个对应字符都相等即可。

习题 3 单选题

若串 $S =$“software”，其不同子串的个数（　　）

A. 37

B. 36

C. 8

D. 9

答案：A

解析：参考例 4.1，子串的定义是串内部连续的一段字符，而“software”内部的字符两两不同，因此不用考虑重复的情况（注意如果有重复的话按题目要求确定是否统计）。

长度为 1 的子串有 8 个（分别为“s”、“o”、“f”、“t”、“w”、“a”、“r”、“e”）。

长度为 2 的子串有 7 个（分别为“so”、“of”、“ft”、“tw”、“wa”、“ar”、“re”）。

长度为 3 的子串有 6 个（分别为“sof”、“oft”、“ftw”、“twa”、“war”、“are”）。

……

长度为 7 的子串有 2 个（分别为“softwar”、“oftware”）。

长度为 8 的子串有 1 个（分别为“software”）。

不要忘记还有 1 个空串。

$8 + 7 + 6 + \cdots + 2 + 1 + 1 = 37$

习题 4 单选题

在 KMP 模式匹配中用 *next* 数组存放模式串的部分匹配信息，当模式串位 j 与目标串 i 比较时两字符不相等，则 i 的位移方式是（　　）

A. $i = next[j]$

B. i 不变

C. $i = 0$

D. $i = i - j + 1$

答案：B

解析：*next* 数组存放的是最长相同前后缀。也就是当前的串如果失配了，可以尝试某个和当前后缀相等的前缀，保留一部分已经匹配的内容。

因此当**模式串位** j 出现失配时，j 自行变为 $next[j]$，仍然与 i 进行比较。注意审题 !!! 减少被题目误导的机会。

习题 5 两串的最长相同前后缀

【问题描述】

给定两个字符串 s_1 和 s_2，求最长的 s_1 前缀 ss 使得 ss 为 s_2 的最长后缀，输出该字符串和其长度。

【输入形式】

输入的每个测试用例由两行构成，第一行为 s_1，第二行为 s_2。假设所有输入的数据均为小写字母。

【输出形式】

每个测试用例的输出由单行组成，其中包含最长的字符串，该字符串是 s_1 的前缀和 s_2 的后缀，后面跟着该前缀的长度。如果最长的此类字符串是空字符串，则输出应为 0。s_1 和 s_2 的长度最多为 50000。

【样例输入】

```
aaariemann marjorieaaa
```

【样例输出】

```
aaa 3
```

【样例说明】

输入的第一行字符串 s_1 =‘aaariemann’，第二行字符串 s_2 =‘marjorieaaa’。

s_1 的前缀和 s_2 的后缀最长相等字符串为 aaa，因此输出 aaa 3，而不是 a 1。测试数据存放在 in.txt 文件中。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int nextval[100100];
int main()
{
    ifstream in("in.txt");
    string s1,s2;
    in>>s1>>s2;
    s1+=s2;//把s2连接到s1后面
    //求新s1的next数组
    int i=0,k=-1;
    nextval[0]=-1;
    while(i<s1.length())
    {
        if(k==-1||s1[i]==s1[k])
        {
            i++;
            k++;
            if(s1[i]==s1[k])
                nextval[i]=k;
            else
                nextval[i]=nextval[k];
        }
        else
            k=nextval[k];
    }
    int common=nextval[s1.length()-1]+1;
    if(common>min(s1.length(),s2.length()))
        common=min(s1.length(),s2.length());
    cout<<s1.substr(0,common)<<" "<<common;
    return 0;
}
```

解析：本题需要求出“s_1 的前缀与 s_2 的后缀相等的最长长度”。最简单的方法是逐个比较 $s_1[0..i]$ 和 $s_2[length-i-1..length-1]$ 是否相等。但是这样的时间复杂度是 $O(n^2)$，正常情况下会超时（查阅大家代码后发现这样做也没有超时）。

实际上本题是需要大家使用 KMP 算法的 *nextval* 数组。首先将 s_2 连接到 s_1 后面，然后求出新串的 *next* 数组。

nextval$[i]$ 的含义就是 $\max\{k|0<k<i, \text{“}s_0s_1\dots s_{k-1}\text{”}=\text{“}s_{i-k}s_{i-k+1}\dots s_{i-1}\text{”}\}$，刚好对应了 s_1+s_2（此处 $+$ 指连接操作）的前缀和后缀相等的情况。

最后输出长度 *nextval*$[s_1.length-1]+1$ 即可。

但是要注意，拼接后的 *next* 值是有可能超出 s_1 和 s_2 的长度的，思考 $s_1=$“aaaaaa” 和 $s_2=$“aaaaaa” 拼接的情况。所以要判断使结果不超过 $\min(s_1.length, s_2.length)$。

习题 6 压缩字符串

【问题描述】

给定一组字符，使用原地算法将其压缩。压缩后的长度必须始终小于或等于原数组长度。数组的每个元素应该是长度为 1 的字符（不是 `int` 整数类型）。在完成原地修改输入数组后，返回数组的新长度。

【输入形式】

输入一组字符，字符间以空格隔开。

【输出形式】

输出一组字符，重复的字符用重复次数替代。每个数字在数组中都有它的位置。输出字符间以空格隔开。

【样例输入】

```
a a b b c c c
```

【样例输出】

```
a 2 b 2 c 3
```

【样例说明】

“a a” 被 “a 2” 替代。“b b” 被 “b 2” 替代。“c c c” 被 “c 3” 替代。测试数据存放在 `in.txt` 文件中。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include<iostream>
#include<fstream>
#include<stack>
using namespace std;
stack<int> s;
int main()
{
    ifstream in("in.txt");
    char c,pre=0;
    int cnt=0;
    while(in>>c)//逐字符输入，没有字符了就停止（输入流会忽略空格）
    {
        if(c==pre)//如果仍然相等，就继续统计
            cnt++;
        else
        {
            if(pre!=0)//pre=0表示是第一个字符
            {
                cout<<pre<<" ";
                if(cnt>1)//cnt=1时就不输出数量
                {
                    while(cnt)//使用栈把字符数量分空格输出
                    {
                        s.push(cnt%10);
                        cnt/=10;
                    }
                    while(!s.empty())//最高位会在栈顶
                    {
                        cout<<s.top()<<" ";
                        s.pop();
                    }
                }
            }
            cnt=1;
            pre=c;
        }
    }

    cout<<pre<<" ";//注意最后一个字符在结尾处，无法被作为pre判断
    if(cnt>1)
    {
        while(cnt)
```

```
43          {
44              s.push(cnt%10);
45              cnt/=10;
46          }
47          while(!s.empty())
48          {
49              cout<<s.top()<<" ";
50              s.pop();
51          }
52      }
53      return 0;
54  }
```

解析：本题把串以字符的形式分开输入，考察对相同字符的判断。当存在相同字符时才用数字把出现的数量替代，从而满足题目中**始终小于或等于原数组长度**的要求。注意当数量 ≥ 10 时需要把每位数字隔开，因为每次只能方便地用取模获得最低位，所以还要用栈将这个顺序取反。

总结

题目：串

日期：2024 年 4 月 22 日（于江汉路）

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：概念题，注意概念细节、定义区分，善于使用排除法。

习题 2：串是特殊的线性表，而对串进行比较只需要管内容是否相等即可。

习题 3：本题一方面要注意是否有相等字符（子串），如果有就需要排除。此外还有空串的情况需要考虑。和集合的概念类似，如果题目提到“真子串”也要注意不能把整个字符串算进去。

*习题 3 思考题：字符串“hongkong”的*不同非空真子串*一共有多少个?*

习题 4：注意区分模式串和匹配串的定义，在具体实现时区分函数 `GetNextval()` 和 `KMPval()`。

习题 5：本题使用枚举匹配是可以通过的，但是更建议大家向 KMP 算法的方向思考。

习题 6：注意本题的测试数据中，输入数据是由空格分开的串，此外输出的数字也是由空格分开的。大家平时写作业遇到问题时可以找助教帮大家尽快排除问题。

本次作业经查重，无代码雷同情况，感谢同学们的辛苦付出！

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 7 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：数组与递归
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

设有数组 $A[i,j]$，数组中的每个元素长度为 3 个字节，i 的值为 1 到 8，j 的值为 1 到 10，数组从内存首地址 BA 开始以列为主序顺序存放，元素 $A[5,8]$ 的存储首地址为（　　）。

A. $BA+141$
B. $BA+180$
C. $BA+222$
D. $BA+225$

答案：B

解析：参考教材 5.1.2 数组的存储结构，题目说**“以列为主序”**进行存放，因此 $A[5,8]$ 前面还有 $1\sim7$ 列的各个元素，以及 $A[1,8]\sim A[4,8]$。共有 $7\times8+4=60$ 个元素，每个元素占据 3 个字节，共为 180 字节的偏移量。

在做题的时候，你可以认为内存首地址 BA 存储的是 $A[1,1]$，偏移量为 0，内存地址为 $BA+0$。所以计算出 $A[5,8]$ 为 $BA+180$。

习题 2 单选题

设某二维数组 $a[10][20]$ 采用顺序存储方式，每个数组元素占用 1 个存储单元，$a[0][0]$ 的存储地址为 200，$a[6][2]$ 的存储地址是 322，则该数组（　　）。

A. 只能按行优先存储

B. 只能按列优先存储

C. 按行优先存储或按列优先存储均可

D. 以上都不对

答案：A

解析：每个数组元素占用 1 个存储单元，所以 $a[6][2]$ 前面应该有 122 个元素。假设按行优先存储，$a[6][2]$ 前面有 $6 \times 20 + 2 = 122$ 个元素；假设按列优先存储，$a[6][2]$ 前面有 $2 \times 10 + 6 = 26$ 个元素。

按列存储与题目所给条件矛盾，因此只能按行优先存储。

习题 3 填空题

设有一个带宽为 5 的 10 阶对角阵 A，采用压缩存储方式，$A[1][1]$ 的存储地址为 1，每个元素占一个地址空间，若以行序为主序，$A[8][6]$ 的存储地址是（　　）；若以列序为主序，$A[8][6]$ 的存储地址是（　　）。

答案：33，27

解析：参考教材 5.2.3，带宽为 5 的 10 阶对角阵 A 应该如下所示，半带宽 $b = 2$。

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} & a_{7,9} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{8,6} & a_{8,7} & a_{8,8} & a_{8,9} & a_{8,10} \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{9,7} & a_{9,8} & a_{9,9} & a_{9,10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{10,8} & a_{10,9} & a_{10,10} \end{bmatrix}$$

其特征为有 5 条斜向右下的线上是有元素的，其他为 0。压缩存储时只考虑非零元素，按照顺序存储在（一维）压缩数组里。

如果只是填空题，挨个数出 $A[8][6]$ 前面有多少个元素就可以了。当以行为主序，就一行一行数（从第一行到最后一行左到右）；当以列为主序，就一列一列数（从第一列到最后一列上到下）。

如果本题是编程题，可以考察给定 b 和下标 (i, j)，求出 $A[i][j]$ 在压缩矩阵中的的存储地址。前 b 行中，第 i 行有 $b+i$ 个元素，后 b 行中，第 i 行有 $b-(n-i)+1$ 个元素。其余中间各行有 $2b+1$ 个元素。

因此第 i 行第 j 列以行为主序存储在压缩数组中的位置为：

$$\begin{cases} \sum_{k=1}^{i-1}(b+k)+j, & i \le b \\ \frac{(3b+1)b}{2}+\sum_{k=1}^{i-b-1}(2b+1)+j-(i-b-1), & b < i \le n-b \\ \frac{(3b+1)b}{2}+(n-2b)(2b+1)+\sum_{k=1}^{i-n+b}(b-(n-k)+1)+j-(i-b-1), & i > n-b \end{cases}$$

计算方式主要是计算前面的行共有多少非零元素，加上当前行前面已经出现过的非零元素个数。（不作详细解释，感兴趣可以单独询问助教）。

习题 4 N 皇后 II

【问题描述】

n 皇后问题研究的是如何将 n 个皇后放置在 $n \times n$ 的棋盘上，并且使皇后彼此之间不能相互攻击。给定一个整数 n，返回 n 皇后不同的解决方案的数量。要求设计如下函数：

```
class Solution{
public:
    int totalNQueens(int n)
    { ... }
};
```

【输入形式】

输入一个整数 n。

【输出形式】

输出 n 皇后问题的解决方案的数量。

【样例输入】

6

【样例输出】

4

【样例说明】

6 皇后问题有 4 个不同的求解方案。测试数据存放在 `in.txt` 文件中。

【评分标准】

共 10 个测试用例，每通过一个测试得 10 分。

答案：

```
#include<cmath>
#include<cstdio>
#include<cstring>
#include<iostream>
#include<fstream>
using namespace std;
class Solution {
    int cnt;        //累计解个数
    int q[50];      //q[i]存放第i行放置皇后的列号
```

```
public:
    int totalNQueens(int n)
    {
        cnt=0;//初始化并清空
        memset(q,0,sizeof(q));
        queen(1,n);
        return cnt;
    }
    bool place(int i,int j)      //测试(i,j)位置能否摆放皇后
    {
        for(int k=1;k<i;k++)       //k=1~i-1是已放置了皇后的行
        //如果前面有人在j列放了皇后，就false，结束当前测试
            if(q[k]==j||(abs(q[k]-j)==abs(i-k)))//同一列或同一对角线
                return false;
        //没有遇到异常情况就返回true，可以放置
        return true;
    }
    void queen(int i,int n)//已放置1~i-1的皇后，正在放置第i行
    {
        if(i>n)        //所有皇后放置结束
        {
            cnt++;       //统计一个解
            return;
        }
            else
            {
                for(int j=1;j<=n;j++)    //在第i行上试探每一个列j
                if(place(i,j))     //如果(i,j)可以放置皇后
                {
                    q[i]=j;          //修改值，表示第i行的皇后放在了第j列
                    queen(i+1,n);    //每一层有多个放皇后的可能
                }//所以循环会多次进入queen(i+1,n)
            }
    }
};
int main()
{
    ifstream in("in.txt");
    int n;
    in>>n;
    Solution obj;
    cout<<obj.totalNQueens(n);
```

```
52      return 0;
53      }
```

解析：本题需要计算 n 皇后的方案数。在搜索过程中，需要时刻注意不要和前面放置过的皇后产生冲突，为了记录这样的信息，我们把每一行已经放置的皇后存储在数组 q 中。

每次尝试将第 i 行的皇后放在第 j 列，需要保证前面没有其他的 $q[k] = j$，且没有对角线冲突，即 $|q[k] - j| = |k - i|$。如果满足条件，就可以继续放置下一行的皇后，否则就需要回溯到上一层重新放置上一层的皇后进行测试。

习题 5 母牛的故事

【问题描述】
有一头母牛，它每年年初生一头小母牛。每头小母牛从第四个年头开始，每年年初也生一头小母牛。请编程实现在第 n 年的时候，共有多少头母牛?
【输入形式】
输入一个整数 n（$0 < n < 55$），n 的含义如题目中描述。
【输出形式】
输出在第 n 年的时候母牛的数量。
【样例输入】
5
【样例输出】
6
【样例说明】
第 5 年时共有 6 头母牛。测试数据存放在 `in.txt` 文件中。
【评分标准】
该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```
1   #include<iostream>
2   #include<fstream>
3   using namespace std;
4   long long func(int n)
5   {
6       //第n年的牛数量为上一年（n-1年）的数量
7       //加上这一年已经可以生产小牛的母牛数量
```

```
8       //return func(n-1)+func(n-3);
9       //问题在于，n不能一直缩小，所以要在n足够小的时候定义递归边界
10      if(n<=3)//前三年，只有第一头牛能生产
11          return n;
12      return func(n-1)+func(n-3);
13  }
14  int main()
15  {
16      ifstream in("in.txt");
17      int n;
18      in>>n;
19      cout<<func(n);
20      return 0;
21      }
```

解析：详见代码注释。推导过程为 $f_i = f_{i-3} + f_{i-1}$。考虑到本章内容为递归，所以以递归的形式实现，即 $f(n) = f(n-3) + f(n-1)$。

为了防止递归越界，我们要在 n 足够小的时候特判递归边界，让其停止。

个人认为题目描述不够清楚，实际上第一头牛在 $n = 2$ 时才开始生小牛。所以有 $f_1 = 1, f_2 = 2, f_3 = 3$。从 $n = 4$ 开始就满足上面的递推式子了。

建议大家在期末考试遇到这种看到样例解释仍然不明白的问题及时提问。

总结

题目：数组和递归

日期：2024 年 5 月 9 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1、2：概念题，注意数组存储的方式和行列优先级。

习题 3：主要考察对对角矩阵压缩存储的掌握，建议对其定义进行进一步的了解和分析。

习题 4：基础的递归搜索，也是深度优先搜索（dfs）的基础。注意不同层搜索之间的制约、产生的变化和影响。

习题 5：考察简单的递归，类似“斐波那契数列”。本题题目描述较为难懂，建议优先使用样例解释进行输入输出数据的核对，思路没问题后再开始代码编写。

本次作业经查重，有两位同学的代码有相似情况，考虑到部分同学已转入 B 班，仅扣除 20% 的分数并提出提醒。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 8 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：树
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 多选题

在一棵度为 4 的树 T 中，若有 20 个度为 4 的结点，10 个度为 3 的结点，1 个度为 2 的结点，10 个度为 1 的结点，则树 T 的叶节点个数是（　　）。

A．41
B．82
C．113
D．122

答案：B

解析：题目说树 T 的度为 4，因此每个节点度的大小不会超过 4。根据教材 7.1.4，**树的节点个数等于所有节点度数和加一**。也就是说，除了根节点以外，每个节点有一个唯一的父节点。那么一个度对应一个**子节点**，根节点没有父节点，所以要额外加 1。

因此答案应该是 $20 \times 4 + 10 \times 3 + 1 \times 2 + 10 \times 1 + 1 = 82$。

习题 2 多选题

具有 12 个叶节点的二叉树中有（　　）个度为 2 的节点

A. 8

B. 9

C. 10

D. 11

答案：D

解析：二叉树意味着每个节点最多有两个子节点。分类讨论一下，

- 如果一个节点有 0 个子节点，就说明它是叶子；
- 如果有 1 个子节点，说明它不会影响叶子节点；
- 如果有 2 个子节点，它会增加一个分支数量，也就是会多一个叶子节点。

考虑到根节点在没有子节点的时候就是 1 个叶子，每多一个度为 2 的点就会增加一个叶子节点，因此叶子节点的数量是度为 2 的节点数量 +1。所以具有 12 个叶节点的二叉树中有 11 个度为 2 的节点。

习题 3 多选题

已知一棵完全二叉树的第 6 层（设根为第 1 层）有 8 个叶结点，则完全二叉树的结点个数最多是（　　）

A. 39

B. 52

C. 111

D. 119

答案：C

解析：完全二叉树在第六层的节点数不会超过 $2^5 = 32$。如果此时叶子节点个数并未到达 32 个，说明这棵完全二叉树为 **6 或 7** 层。

当这棵完全二叉树为 6 层时，最左边 8 个叶子节点在第 6 层，加上前五层是 $8 + (2^5 - 1) = 41$ 个节点。

当这棵完全二叉树为 7 层时，第 6 层最右边 8 个是叶子节点，因此左边 $32-8 = 24$ 个是非叶子节点，那么第 7 层有 48 个节点，加上前六层共 $48+(2^6-1) =$

111 个节点。

习题 4 多选题

已知 n 个结点的二叉树具有最小路径长度时，其深度为 k，那么第 k 层上的结点数为（　　）

A. $n-2^{k-2}+1$

B. $n-2^{k-1}+1$

C. $n-2^{k}+1$

D. $n-2^{k-1}$

答案：B

解析：二叉树具有最小路径长度时，说明此时**深度最低**。也就是尽可能将二叉树的形态构建为完全二叉树，当深度为 k 就要尽可能保证前 $k-1$ 层全满。

前 $k-1$ 层的节点个数为 $\sum_{i=1}^{k-1}2^{i-1}=2^{k}-1$。因此第 k 层剩余 $n-2^{k-1}+1$ 个节点。

习题 5 树的简单问题

【问题描述】

假设二叉树中的每个结点值为单个整数，采用二叉链结构存储，假定每颗二叉树不超过 2000 个节点。设计算法完成

（1）按从左到右的顺序输出二叉树的叶子结点

（2）按从右到左的顺序输出二叉树的叶子结点

（3）输出二叉树所有的节点，按照从根节点开始，逐层输出，同一层按照从右向左的顺序

【输入形式】

每个测试是一颗二叉树的括号表示法字符串。

【输出形式】

第一行是按从左到右的顺序输出二叉树的叶子结点，结点之间用空格隔开

第二行是按从右到左的顺序输出二叉树的叶子结点，结点之间用空格隔开

第三行是输出二叉树所有的节点，按照从根节点开始，逐层输出，同一层按照从右向左的顺序，结点之间用空格隔开

【样例输入】

```
1(2(4,5),3)
```

【样例输出】

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

【样例说明】

测试数据的文件名为 in.txt。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <queue>
#include <stack>
#include <fstream>
#include <algorithm>

using namespace std;

/* Definition for a binary tree node.*/
struct BTNode
{
    int data;
    BTNode *lchild;
    BTNode *rchild;
    BTNode() : lchild(NULL), rchild(NULL) {}
    BTNode(int x) : data(x), lchild(NULL), rchild(NULL) {}
};
class BTree
{
    BTNode *r;

public:
    BTree()
    {
        r = NULL;
    }
```

```
29      void CreateBTree(string str) // 从str创建二叉树
30      {
31          stack<BTNode *> st; // 定义名为st的栈
32          BTNode *p;
33          bool flag;
34          int i = 0;
35          while (i < str.length()) // 用i从前到后遍历str的每个字符
36          {
37              switch (str[i])
38              {
39              case '(':
40                  st.push(p);  // 如果是'(', 则将p压入栈中
41                  flag = true; // flag=true说明去找左子节点
42                  break;
43              case ')':
44                  st.pop(); // 否则出栈
45                  break;
46              case ',':
47                  flag = false; // 如果遇到逗号就将flag置为false, 去找右子节点
48                  break;
49              default: // 遇到数字就进行读入操作
50                  int sum = 0;
51                  while (str[i] <= '9' && str[i] >= '0')
52                  {
53                      sum = sum * 10 + str[i] - '0';
54                      i++;
55                  }
56                  i--;
57                  p = new BTNode(sum); // 将读到的数字存入当前节点p
58                  if (r == NULL)       // 如果根节点为空, 则将p设为根节点
59                      r = p;
60                  else // 否则进行查找, 将p插入子节点
61                  {
62                      if (flag && !st.empty()) // flag=true表示p插入左子节点
63                          st.top()->lchild = p;
64                      else if (!st.empty()) // 否则将p插入右子节点
65                          st.top()->rchild = p;
66                  }
67                  break;
68              }
69              i++; // 进行下一个字符的读取
70          }
```

```
71      }
72
73      void DispBTree() // 输出二叉树，从根节点开始遍历
74      {
75          DispBTree1(r);
76      }
77      void DispBTree1(BTNode *b) // 作为DispBTree的搜索函数
78      {
79          if (b != NULL)
80          {
81              cout << b->data; // 输出当前节点（先序遍历）
82              if (b->lchild != NULL || b->rchild != NULL)
83              {
84                  cout << "(";            // 输出子节点前打"("
85                  DispBTree1(b->lchild); // 递归输出左子节点
86                  if (b->rchild != NULL)
87                      cout << ",";        // 如果有右子节点，在左右子节点之间输
                            出","
88                  DispBTree1(b->rchild); // 递归输出右子节点
89                  cout << ")";            // 输出子节点后打")"
90              }
91          }
92      }
93      void LeftToRight() // 从左往右输出叶子节点
94      {
95          LeftToRight1(r);
96      }
97      void LeftToRight1(BTNode *b) // 作为LeftToRight()的搜索函数
98      {
99          if (b != NULL)
100         {
101             if (b->lchild == NULL && b->rchild == NULL) // 判断是叶子节点，就
                    输出
102                 cout << b->data << " ";
103             else if (b->lchild != NULL)
104                 LeftToRight1(b->lchild);
105             LeftToRight1(b->rchild);
106         }
107     }
108
109     void RightToLeft() // 从右往左输出叶子节点
110     {
```

```
111         RightToLeft1(r);
112     }
113     void RightToLeft1(BTNode *b) // 作为RightToLeft的辅助函数
114     {
115         if (b != NULL)
116         {
117             if (b->lchild == NULL && b->rchild == NULL) // 如果是叶子节点就输
                出
118                 cout << b->data << " ";
119             else if (b->rchild != NULL) // 先去遍历右子节点
120                 RightToLeft1(b->rchild);
121             RightToLeft1(b->lchild);
122         }
123     }
124     void AllLevel() // 逐层输出节点
125     {
126         AllLevel1(r);
127     }
128     void AllLevel1(BTNode *b) // 作为AllLevel的辅助函数
129     {
130         BTNode *p;
131         queue<BTNode *> qu;
132         qu.push(b);
133         while (!qu.empty())
134         {
135             p = qu.front();
136             qu.pop();                // 将p从队列中弹出
137             cout << p->data << " "; // 输出p的数据
138             if (p->rchild != NULL)  // 将右子节点压入队列
139                 qu.push(p->rchild);
140             if (p->lchild != NULL) // 将左子节点压入队列
141                 qu.push(p->lchild);
142         }
143     }
144 };
145 int main()
146 {
147     string s;
148     BTree bt;
149     vector<string> v;
150     bool result;
151     ifstream infile("in.txt");
```

```
152     if (!infile)
153     {
154         cerr << "Error opening file";
155         exit(EXIT_FAILURE);
156     }
157     while (getline(infile, s))
158     {
159         v.push_back(s);
160     }
161     infile.close();
162     infile.clear();
163     bt.CreateBTree(v[0]);
164     bt.LeftToRight();
165     cout << endl;
166     bt.RightToLeft();
167     cout << endl;
168     bt.AllLevel();
169 }
```

解析：本题难点在于解析输入的代表二叉树的字符串。处理需要关注的问题主要是：遇到左括号就深入一层，遇到右括号就撤回一层。

这个过程可以用栈来模拟，也可以把所有节点的父子关系都保存下来。我们在读取字符串的过程中需要知道**当前处理的节点位置、当前节点的父节点、当前节点的左右孩子节点**。

总结

题目：树

日期：2024 年 5 月 13 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：需要熟练掌握度的概念。

习题 2：注意对二叉树的理解和分类讨论。

习题 3：完全二叉树在同一层可能有两种形态：全满或未满。在讨论这种问题时需要对两种都注意到。

习题 4：弄明白二叉树具有最小路径长度代表的含义，接着对二叉树进行数学统计和计算即可。

习题 5：对于这种题目，需要熟练应用字符串和栈/递归。对于树的遍历，建议熟练掌握递归和非递归的方法。

本次作业经查重，有两组同学的代码有相似情况，已扣除 20% 的分数。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 9 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：树
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

设有 13 个值，用它们组成一棵哈夫曼树，则该哈夫曼树共有（ ）个结点
A.13
B.12
C.26
D.25

答案：D

解析：若有 13 个值，说明有 13 个叶子节点。每次合并两个叶子节点会**增加一个节点并减少一个联通块**。一开始是 13 个联通块，需要合并 12 次到 1 个联通块。

因此需要增加 $13-1=12$ 个节点，所以哈夫曼树共有 $13+12=25$ 个节点。

习题 2 单选题

某二叉树的先序序列和后序序列正好相反，则该二叉树一定是（ ）
A. 空或者只有一个结点
B. 完全二叉树
C. 二叉排序树
D. 高度等于其结点数

答案：D

解析：先序序列的遍历顺序是：【当前、左子树、右子树】；后序序列的遍历顺序是：【左子树、右子树、当前】。

为了完全倒转过来，我们需要让【当前、左子树、右子树】变成【右子树倒序、左子树倒序、当前】，并和【左子树、右子树、当前】保持一致。

所以只要保证左右子树只有其中一棵，就可以了，此时除了叶子节点，所有点的度数都为 1，高度等于其结点数。

习题 3 单选题

一颗完全二叉树上有 1001 个结点，其中叶子结点的个数是（　　）

A.250

B.501

C.254

D.505

答案：B

解析：考虑完全二叉树的定义，每一层、每一个节点一定要填满了才能开始填下一层、下一个节点。那么第 i 层一定会有 2^{i-1} 个节点。

第一层 1 个，第二层 2 个，第三层 4 个，……，第 k 层 2^{k-1} 个。

观察等比数列得到，$1+2+2^2+\cdots+2^9=1023$，所以 1001 个节点的完全二叉树的高度为 9。此时撤掉最后的 $1023-1001=22$ 个节点，为第 8 行空出 11 个叶子，此时第 9 行剩余 $512-22=490$ 个叶子。

因此叶子数量为 501。

习题 4 单选题

如果将一棵有序树 T 转换为二叉树 B，那么 T 中结点的层次序列对应 B 的（　　）序列

A. 先序遍历

B. 中序遍历

C. 层次遍历

D. 以上都不对

答案：D

解析：参考教材 7.8.1。此题可以用排除法进行判断

二叉树无论先序遍历、中序、后序，都会分离左右两棵子树。而有序树是有层次的，左右子树可能是混合连接的，所以要先排除 A、B 两项。

当一个节点有很多孩子时，它的孩子会顺着当前的右子树**逐层加深**排布，因此也无法满足层次遍历。所以选 D。

习题 5 由先序和中序序列产生后序序列

【问题描述】

由二叉树的先序序列和中序序列构造二叉树并求其后序序列。

【输入形式】

每个测试用例的第一行包含一个整数 n（$1 \leq n \leq 1000$）表示二叉树的节点个数，所有节点的编号为 $1 \sim n$，后面两行分别给出先序序列和中序序列。可以假设构造出的二叉树是唯一的。

【输出形式】

对于每个测试用例，输出一行表示其后序序列。

【样例输入】

```
9
1 2 4 7 3 5 8 9 6
4 7 2 1 8 5 9 3 6
```

【样例输出】

```
7 4 2 8 9 5 6 3 1
```

【样例说明】

测试数据的文件名为 `in.txt`。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include<iostream>
#include<fstream>
#include<vector>
using namespace std;
vector<int> pre,ins;
void suc(int l1,int r1,int l2,int r2)
{
```

```
    if(l1>r1||l2>r2)//递归边界，此时树空
        return;
    //考虑先序遍历第一个一定是「当前的根节点」
    //也就是在l2-r2中找到pre[l1]所在的位置
    int i;
    for(i=l2;i<=r2;i++)
    {
        if(ins[i]==pre[l1])
            break;
    }
    //此时i就是中序遍历的根节点，前面为左孩子，右边为右孩子
    //左孩子大小为i-l2
    //右孩子大小为r2-i
    //后序遍历为：左、右、当前
    suc(l1+1,l1+i-l2,l2,i-1);
    suc(r1-r2+i+1,r1,i+1,r2);
    //此处注意左右长度保持一致，右子树长度更好确定
    cout<<pre[l1]<<" ";
}
int main()
{
    ifstream in("in.txt");
    int n,x;
    in>>n;//输入n
    for(int i=0;i<n;i++)
    {
        in>>x;//输入当前数字并将其插入链表最后，作为数组
        pre.push_back(x);
    }
    for(int i=0;i<n;i++)
    {
        in>>x;//输入当前数字并将其插入链表最后，作为数组
        ins.push_back(x);
    }
    //该函数表示将先序遍历l1-r1与中序遍历l2-r2进行对应，并输出后序
    suc(0,n-1,0,n-1);
    return 0;
}
```

解析：如果已知先序遍历，那么先序遍历的第一个位置就是根节点，此时在中序遍历中找到这一根节点，将中序遍历左边部分划分为左子树，右边部分划分

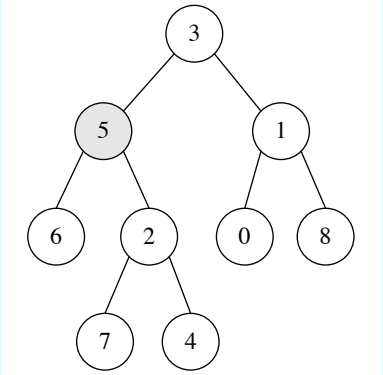
为右子树。这时先序遍历也可以按照子树大小进行划分，递归执行上述内容直到划分完毕。

习题 6 二叉树中距离为 k 的结点问题

【问题描述】

给你二叉树的根结点 *root*、树中一个结点 *target* 和一个正整数 k，求二叉树中距离 *target* 结点为 k 的所有结点。假设二叉树中的所有结点值都唯一。输入为顺序存储方式表示的二叉树字符串，如果节点为空，则输入‘#’。

例如，输入 $root = [3,5,1,6,2,0,8,\#,\#,7,4], target = 5 \square k = 2$，输出为 $[7,4,1]$。对应的二叉树如图所示。



其中离目标结点 5 的距离为 2 的结点是 7、4、1。假设给定的二叉树非空，树中每个结点有唯一值，结点值位于 0 到 500 之间，*target* 是其中的一个结点，$0 \le k \le 1000$。

要求设计如下成员函数：

```
1
2 class Solution {
3 public:
4     vector<int> distanceK(TreeNode* root, TreeNode*
        target, int k)
5     {   ...   }
6 };
```

【输入形式】

每个测试用例由三行，第一行是由一对方括号 [] 括起来的顺序存储的二

叉树节点数据，每个节点用“,”隔开，如果是空节点，则存入“#”。第二行为目标节点数据，第三行为距离。

【输出形式】

用一对方括号 [] 将满足要求的节点括起来后输出，如有多个节点，用“,”隔开，节点输出顺序：优先输出目标节点的子孙节点，其次是目标节点兄弟节点的子孙节点，最后是目标节点的祖父节点（如果有多个祖父节点的子孙节点，则从最近的开始输出），如果在同一层从左边到右边输出。

【样例输入】

```
[1,2,3,4,5,6,7,8,9,10,11,12,13,#,#,14,15,16,17,18,19,#,#,
#,#,#,#,#,#,#,#,20,21,22,23,24,25,26,27,28,29]
4
3
```

【样例输出】

```
[20,21,22,23,24,25,26,27,10,11,3]
```

【样例说明】

测试数据的文件名为 in.txt。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```cpp
#include<iostream>
#include<fstream>
#include<vector>
#include<queue>
#include<stack>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
int target,dis;
vector<int> tree;
queue<TreeNode*> q;//存储每个节点，用于跳转Node
```

```
stack<pair<TreeNode*,int>> s;//存储祖先节点指针和遍历方向
TreeNode* pos;//target所在的指针
bool printed=false;//是否打印过，以此判断是输出逗号还是括号
void dfs2(TreeNode* x,int d)//d表示找距离当前深度为d的点
{
    if(x->val==-1)//空节点直接结束
        return;
    if(d==0)
    {
        if(!printed)
        {
            printed=1;
            printf("[");
        }
        else
            printf(",");
        cout<<x->val;
        return;
    }
    dfs2(x->left,d-1);
    dfs2(x->right,d-1);
}
void dfs(TreeNode* x)
{
    if(x==NULL||x->val==-1)
        return;
    if(x==pos)
    {
        dfs2(x,dis);//当前点子树中距离为dis的打印出来
        //此时可以把它的祖先全部进行遍历
        while(!s.empty()&&dis>0)
        {
            dis--;//每向上一层，需要找的位置就少一个
            if(dis==0)
                dfs2(s.top().first,0);
            if(s.top().second)
                dfs2(s.top().first->left,dis-1);
            if(s.top().first)
                dfs2(s.top().first->right,dis-1);
            s.pop();
        }
        return;
```

```
58      }
59      s.push(make_pair(x,0));
60      dfs(x->left);
61      if(!s.empty())//如果找到答案了就可能是空栈
62          s.pop();
63      s.push(make_pair(x,1));
64      dfs(x->right);
65      if(!s.empty())//如果找到答案了就可能是空栈
66          s.pop();
67  }
68  int main()
69  {
70      ifstream in("in.txt");
71      string s;
72      in>>s;
73      in>>target>>dis;
74      int now=0;//现在的数字
75      for(int i=1;i<s.size();i++)//跳过[
76      {
77          if(s[i]>='0'&&s[i]<='9')
78              now=now*10+s[i]-'0';//进位读取当前数字
79          else if(s[i]=='#')
80              now=-1;//如果是空为止，就放-1代表空
81          else
82          {
83              tree.push_back(now);//将当前数字存储到树上
84              now=0;//清空now
85          }
86      }
87      int fa=0,son=1;//依次找到每个节点对应的父子关系
88      TreeNode* root=new TreeNode(tree[0]);//创建根节点
89      pos=root;//target的位置（初始化为root）
90      //如果son没有找到target，就说明它在root
91      q.push(root);//需要遍历的父节点
92      while(son<tree.size())//遍历所有子节点，寻找父节点
93      {
94          TreeNode* Node=q.front();//当前节点指针
95          q.pop();
96          Node->left=new TreeNode(tree[son]);//优先找左孩子
97          q.push(Node->left);
98          if(tree[son]==target)//如果遇到target就保存位置
99              pos=Node->left;
```

```
100         son++;//找下一个孩子
101         if(son<tree.size())
102         {
103             Node->right=new TreeNode(tree[son]);
104             q.push(Node->right);
105             if(tree[son]==target)
106                 pos=Node->right;
107         }
108         son++;
109         fa++;
110     }
111     //从根节点开始dfs，用栈存储经历过的点，方便回溯
112     dfs(root);
113     printf("]");//格式收尾
114     return 0;
115 }
```

解析：本题可以采用 BFS 宽度优先搜索的方式构建输入的树，也可以用下标对树上的节点进行映射。

此后可以把整棵树的相连关系构建出来，从 *target* 出发走 k 步，输出对应的节点。但是要注意先输出自己子树中的点，再输出父节点、祖先节点。

或者按照上面答案中写的，用栈的形式，依次处理当前节点、父节点、祖先节点。对于当前节点，找到距离为 k 的点，对于父节点，找到**另一棵子树中**距离为 $k-1$ 的点，以此类推。

总结

题目：树 2

日期：2024 年 5 月 20 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：考察哈夫曼树的定义。

习题 2：考察二叉树的（三种）遍历方式。

习题 3：对完全二叉树的掌握，主要是要构建出完全二叉树的形状。

习题 4：考察二叉树和多叉树的互换，以及二叉树的（三种）遍历方式。

习题 5：考察先序遍历的特点，以及树的构建和遍历。如果一边构建一边遍历不是很熟练的话，可以先构建再遍历。

习题 6：考察对二叉树的构建、逻辑关系处理，以及树的深度和广度优先搜索。

本次作业经查重，没有代码相似的同学。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 10 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：树
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

一颗完全二叉树上有 1001 个结点，其中叶子结点的个数是（ ）
A.250
B.501
C.254
D.505

答案：B

解析：注意完全二叉树指的是前 $h-1$ 层全满，第 h 层从左到右依次填满的二叉树。所以第 h 层的叶子节点个数为 2^{h-1}，第 $h-1$ 层的叶子节点个数为 2^{h-2}，以此类推。

那么 1001 是介于 $2^9(512)$ 和 $2^{10}(1024)$ 之间的，所以二叉树停止在第 10 层。前 9 层的节点个数为 $\sum_{i=1}^{9} 2^{i-1} = 511$，因此第 10 层叶子节点个数为 $1001-511 = 490$，那么第 9 层有 $490/2 = 245$ 个节点不是叶子，剩余 $256-245 = 11$ 个叶子节点。

因此共有 $490+11 = 501$ 个叶子节点。

（更详细的过程可参考第九次作业）

习题 2 单选题

如果将一棵有序树 T 转换为二叉树 B，那么 T 中结点的层次序列对应 B 的（　　）序列

A. 先序遍历

B. 中序遍历

C. 层次遍历

D. 以上都不对

答案：D

解析：参考教材 7.8.1。此题可以用排除法进行判断

二叉树无论先序遍历、中序、后序，都会分离左右两棵子树。而有序树是有层次的，左右子树可能是混合连接的，所以要先排除 A、B 两项。

当一个节点有很多孩子时，它的孩子会顺着当前的右子树**逐层加深**排布，因此也无法满足层次遍历。所以选 D。

习题 3 单选题

某二叉树的先序序列和后序序列正好相反，则该二叉树一定是（　　）

A. 空或者只有一个结点

B. 完全二叉树

C. 二叉排序树

D. 高度等于其结点数

答案：D

解析：先序序列的遍历顺序是：【当前、左子树、右子树】；后序序列的遍历顺序是：【左子树、右子树、当前】。

为了完全倒转过来，我们需要让【当前、左子树、右子树】变成【右子树倒序、左子树倒序、当前】，并和【左子树、右子树、当前】保持一致。

所以只要保证左右子树只有其中一棵，就可以了，此时除了叶子节点，所有点的度数都为 1，高度等于其结点数。

习题 4 单选题

设有 13 个值，用它们组成一棵哈夫曼树，则该哈夫曼树共有（　　）个结点

A.13

B.12

C.26

D.25

答案：D

解析：若有 13 个值，说明有 13 个叶子节点。每次合并两个叶子节点会**增加一个节点并减少一个联通块**。一开始是 13 个联通块，需要合并 12 次到 1 个联通块。

因此需要增加 $13-1=12$ 个节点，所以哈夫曼树共有 $13+12=25$ 个节点。

习题 5 单选题

设森林 F 对应的二叉树为 B，B 有 m 个结点，B 的根结点为 p，p 的右子树结点个数为 n，森林 F 中第一棵树的结点个数是（　　）

A. $m-n$

B. $m-n-1$

C. $n+1$

D. 条件不足，无法确定

答案：A

解析：参考教材 7.8.2，**转换后二叉树的根节点**是森林中第一棵树的根节点，因为在将森林从多棵二叉树拼接成一棵二叉树之前，所有的根结点会连成兄弟。

那么第一棵树的根结点就是整个二叉树的根结点，所以第一棵树的结点个数就是整个二叉树的结点个数减去右子树的结点个数，即 $m-n$。

习题 6 多选题

已知一棵二叉树的前序序列为 BACDEGHF，中序序列为 CADBHGEF，则其后序序列为（　　）

A. CAHGDFEB

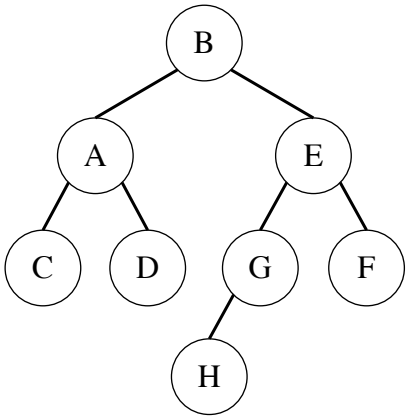
B. CDABGHFE
C. CDAHGFEB
D. HGFEBCDA

答案： C

解析：前序遍历的顺序是：【当前、左子树、右子树】；中序遍历的顺序是：【左子树、当前、右子树】；后序遍历的顺序是：【左子树、右子树、当前】。

所以可以利用**前序遍历的第一个节点**确定根节点，再用这个节点在中序遍历中区分左右子树，具体过程如下。

- 前序：BACDEGHF，中序：CADBHGEF，推出根节点为 B
  - 左子树前序 ACD，右子树前序 EGHF
  - 左子树中序 CAD，右子树中序 HGEF
- 左子树根节点推出为 A，右子树根节点推出为 E
  - 左子树的左子树就是 C，右子树是 D
  - 右子树的左子树为 HG，右子树为 F
    - 容易看出 H 是 G 的左孩子，构造出如下的二叉树：



注：前序和后序是等效的，有前序中序可以推后序，有后序中序也可以推前序。但有前序后序时，推出的中序不唯一。

习题 7 畅通工程问题

【问题描述】

某省调查城镇交通状况，得到现有城镇道路统计表，表中列出了每条道路直接连通的城镇。省政府“畅通工程”的目标是使全省任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要互相间接通过道路可达即可）。问最少还需要建设多少条道路?

【输入形式】

每个测试用例的第 1 行给出两个正整数，分别是城镇数目 n（$n < 1000$）和道路数目 m，随后的 m 行对应 m 条道路，每行给出一对正整数，分别是该条道路直接连通的两个城镇的编号。为简单起见，城镇从 1 到 n 编号。注意两个城市之间可以有多条道路相通，也就是说:

```
3 3
1 2
1 2
2 1
```

这种输入也是合法的。

【输出形式】

对每个测试用例，在一行里输出最少还需要建设的道路数目。

【样例输入】

```
3 3
1 2
1 3
2 3
```

【样例输出】

```
0
```

【样例说明】

测试数据的文件名为 `in.txt`。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案:

```cpp
#include<iostream>
#include<fstream>
using namespace std;
const int MAXN=1005;
int parent[MAXN]; //定义并查集数组
int rnk[MAXN];  //存储并查集的秩，方便按秩合并降低复杂度
int n,m;
void Init()     //初始化并查集，所有人都是自己的根
{   for (int i=1;i<=n;i++)
    {   parent[i]=i;
        rnk[i]=0;
    }
}
int Find(int x)                     //并查集查找根节点
{   if (x!=parent[x])
        parent[x]=Find(parent[x]);  //查找时进行路径压缩
    return parent[x];
}
void Union(int x,int y)             //合并x和y所在的连通块
{   int rx=Find(x);
    int ry=Find(y);
    if (rx==ry)                     //如果两个在同一连通块就不做操作
        return;
    if (rnk[rx]<rnk[ry])
        parent[rx]=ry;              //rx秩更低，应该并给ry
    else
    {   if (rnk[rx]==rnk[ry])       //秩相等，合并后会加1
            rnk[rx]++;
        parent[ry]=rx;              //否则ry合并给rx
    }
}
int main()
{   ifstream txtfile;
    txtfile.open("in.txt");
    if(!txtfile){
        cout << "error open in.txt!" << endl;
        return 1;
    }
    txtfile>>n;
    txtfile>>m;
    Init();                     //并查集需要初始化
    for (int i=1;i<=m;i++)      //共有m条边
```

```
    {   int a,b;
        txtfile>>a;
        txtfile>>b;
        Union(a,b);
    }
    int ans=0;
    for (int i=1;i<=n;i++)        //连通块个数统计为ans
        if (parent[i]==i)         //一旦发现一个根节点
            ans++;                //就说明有一个独立连通块
    cout<<ans-1;                  //需要添加的道路为ans-1条
    return 0;
}
```

解析：本题问还需要多少条道路能将所有城镇连接起来。若城镇目前不相连，说明整张图有超过一个连通块。易于理解的是，**每当两个连通块之间添加一条边时**，整张图的连通块个数就会减少 1。

因此本题要做的是把连通块个数统计出来，最后输出连通块个数 -1。

统计连通块个数有两种方法：一是使用并查集计算有多少节点是并查集树的根，这就是连通块个数；二是先把图建出来，计算需要多少次 BFS/DFS 才能将图遍历完，总共进入 BFS/DFS 的次数就是连通块个数。

上述代码里使用了**并查集**处理，但考虑到并查集可能是非必学内容（我不知道 !! 以老师通知为准），下给出图搜索判断连通块个数的方法：

（以 DFS 为例，仅作参考，无文件输入输出；如要改为 BFS，使用队列即可）

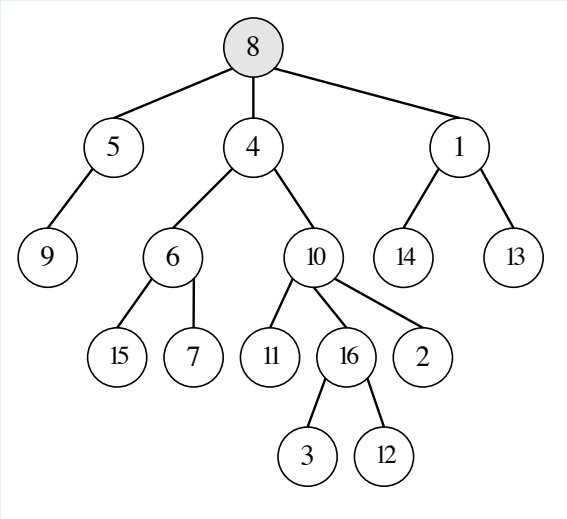
```
#include<iostream>
#include<vector>
using namespace std;
vector<int> v[1050];
int vis[1050];
void dfs(int x)
{
    for(int i=0;i<v[x].size();i++)
        if(!vis[v[x][i]])
        {
            vis[v[x][i]]=1;
            dfs(v[x][i]);
        }
}
int main()
{
```

```
    int n,m,u,w;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        cin>>u>>w;//说明u,w相连
        v[u].push_back(w);
        v[w].push_back(u);//向邻接表添加边
    }
    int cnt=0;//统计连通块个数
    for(int i=1;i<=n;i++)
        if(!vis[i])
        {
            dfs(i);
            ++cnt;
        }
    cout<<cnt-1;
    return 0;
}
```

习题 8 求树中两个结点的最近公共祖先（LCA）

【问题描述】

如图所示是一棵有根树，图中每个结点用 1 ∼ 16 的整数标识，结点 8 是树根。如果结点 x 位于根结点到结点 y 之间的路径中，则结点 x 是结点 y 的祖先。如果结点 x 是结点 y 和结点 z 的祖先，则结点 x 称为两个不同结点 y 和 z 的公共祖先。如果 x 是 y 和 z 的共同祖先并且在所有共同祖先中最接近 y 和 z，则结点 x 被称为结点 y 和 z 的最近公共祖先，如果 y 是 z 的祖先，那么 y 和 z 的最近共同祖先是 y。例如结点 16 和 7 的最近公共祖先是结点 4，结点 2 和 3 的最近公共祖先是结点 10，结点 4 和 12 的最近公共祖先是结点 4。编写一个程序，找到树中两个不同结点的最近公共祖先。



【输入形式】

每个测试用例的第一行为树中结点数 n（$2 \leq n \leq 10000$），所有结点用整数 $1 \sim n$ 标识，接下来的 $n-1$ 行中的每一行包含一对表示边的整数，第一个整数是第二个整数的父结点。请注意，具有 n 个结点的树具有恰好 $n-1$ 个边。每个测试用例的最后一行为两个不同整数，需要计算它们的最近公共祖先。

【输出形式】

为每个测试用例输出一行，该行应包含最近公共祖先结点的编号。

【样例输入】

```
16
1 14
8 5
10 16
5 9
4 6
8 4
4 10
1 13
6 15
10 11
```

6 7
10 2
16 3
8 1
16 12
16 7

【样例输出】

4

【样例说明】

测试数据的文件名为 in.txt。

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例，得 10 分。

答案：

```
#include<iostream>
#include<fstream>

using namespace std;
const int MAXN=10005;
int parent[MAXN];    //和并查集不同的是，parent只存储当前节点的直接父亲
int Level(int x)     //求取x的深度
{   int cnt=0;
    while(x!=-1)     //直到跳转到根节点才停下
    {   x=parent[x];//x跳转到其父亲
        cnt++;       //每跳一层，cnt+1
    }
    return cnt;
}
int solve(int x,int y)       //求x与y的最近公共祖先
{   int lx=Level(x);         //先判断x和y的深度
    int ly=Level(y);
    while (lx>ly)            //至少要跳到同级
    {   x=parent[x];
        lx--;
    }
    while (ly>lx)            //另一种深度相对情况
    {   y=parent[y];
```

```
        ly--;
    }
    while (x!=y)              //直到x和y遇到相等的位置就停下
    {   x=parent[x];          //这里就是他们的公共祖先（最近）
        y=parent[y];
    }
    return x;
}

int main()
{   int n,a,b,x,y;
    ifstream txtfile;
    txtfile.open("in.txt");
    if(!txtfile){
        cout << "error open in.txt!" << endl;
        return 1;
    }
    txtfile>>n;
    for (int i=0;i<=n;i++)            //初始化，没有被更新到的就是根节点
        parent[i]=-1;
    for (int i=1;i<n;i++)             //n个节点的树，共有n-1条边
        {   txtfile>>a;
            txtfile>>b;
            parent[b]=a;
        }
    txtfile>>x>>y;
    int ans=solve(x,y);
    cout<<ans;
}
```

解析：关于求取最近公共祖先，最简单的方法是找出两个点的所有祖先，**然后返回深度最大的一个**，但是这样的话显然复杂度太高，并且没有用到重复性信息。

当两个点深度不同时，它们的深度差距不会带来任何祖先相关的信息，因此需要先将深度调整到一致的水平，也就是让更深的（深度更大的）节点先往上调整，直到两个点深度一样。

此后，两个节点同时向上跳，也一定同时找到一个公共祖先，这时所找到的就是最近的那个公共祖先。因此使用一个 `while` 来控制两个节点的跳转，每次向上跳一层。一旦遇到相同的情况，就直接输出。

注意任意两个点一定有公共祖先，因为根节点是所有人的祖先（包括它自己）。

总结

题目：树 3

日期：2024 年 6 月 17 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：考察完全二叉树的概念（注意与满二叉树进行区分）。

习题 2：考察层次遍历，这个概念较为冷门，建议多掌握这种不太常见但是可能考到的概念。

习题 3：考察先序遍历的特点，以及树的构建和遍历。如果一边构建一边遍历不是很熟练的话，可以先构建再遍历。

习题 4：考察哈夫曼树的概念、原理和构建。

习题 5：对于森林、多叉树、二叉树的互转，需要知道步骤（尤其是思路，代码可以暂缓理解）。注意兄弟之间的顺序和优先级区分。

习题 6：注意掌握前序、中序、后序的原理与特点，会利用其性质解决问题。

习题 7：判断连通块个数的多种方法可以尽量掌握，但主要是需要看出这道题需要大家处理连通块并统计个数。相信代码部分并不是特别有难度，但是还是需要多注意细节。

习题 8：最近公共祖先的朴素算法，注意向上跳的细节，要先调整深度到一致。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 11 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：图
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

用 Dijkstra 算法求一个带权有向图 G 中从顶点 0 出发的最短路径，在算法执行的某时刻，$S=\{0,2,3,4\}$，下一步选取的目标顶点可能是（　　）

A. 顶点 2
B. 顶点 3
C. 顶点 4
D. 顶点 7

答案：D

解析：Dijkstra 算法中，一般会把节点分为 S 和 T 两个集合部分，S 是已经被更新过的，**并且不会被再次更新的那些节点**，而 T 是还未更新的节点，Dijkstra 算法需要在 T 里面找到离源点最近的那一个更新至 S，并利用这个节点更新 T 中的距离。

因此只有顶点 7 有可能被作为这个新点被更新，而 $2,3,4$ 已经在 S 集合里，没有更近的点可以更新它们了。

习题 2 单选题

若一个有向图中的顶点不能排成一个拓扑序列，则可断定该有向图（　　）

A. 是个有根有向图

B. 是个强连通图

C. 含有多个入度为 0 的顶点

D. 含有顶点数目大于 1 的强连通分量

答案：D

解析：仍然使用排除法。先考虑拓扑序列的定义，一般而言是指**有向无环图**。也就是说，**不能有环**，但可以不连通（分成多块），也可以有多个源点（入度为 0 的点）。

按照定义，依次检查选项：

A：不能排成拓扑序列的图可以是无根的有向图，图出入度为 0 的点不一定唯一。

B：如果图强连通，说明图里任意两个点可以互相到达。这样的条件太苛刻，而当一个图无法拍成拓扑序列时，只需要有自环以外的环即可。

C：含有多个入度为 0 的顶点与图是否能拓扑无关。

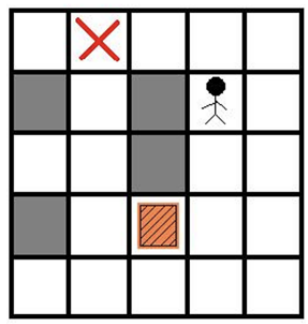
D：顶点数目等于 1 的强连通分量表示自环，不影响拓扑排序结果；而如果有顶点数目大于 1 的强连通分量，说明有环，此时就不能完成拓扑排序前后序依赖的梳理了。

习题 3 推箱子

【问题描述】

推箱子是一个很经典的游戏，今天我们来玩一个简单版本。在一个 $n \times m$ 的房间里有一个箱子和一个搬运工，搬运工的工作就是把箱子推到指定的位置。注意，搬运工只能推箱子而不能拉箱子，因此如果箱子被推到一个角上（如下图所示），那么箱子就不能再被移动了，如果箱子被推到一面墙上，那么箱子只能沿着墙移动。

现在给定房间的结构，箱子的位置，搬运工的位置和箱子要被推去的位置，请你计算出搬运工至少要推动箱子多少格。



【输入形式】

输入数据的第一行是两个正整数 n 和 m（$2 \leq n, m \leq 7$），代表房间的大小，然后是一个 n 行 m 列的矩阵，代表房间的布局，其中 0 代表空的地板，1 代表墙，2 代表箱子的起始位置，3 代表箱子要被推去的位置，4 代表搬运工的起始位置。

【输出形式】

对于每组测试数据，输出搬运工最少需要推动箱子多少格才能将箱子推到指定位置，如果不能推到指定位置则输出 -1。

【样例输入】

```
5 5
0 3 0 0 0
1 0 1 4 0
0 0 1 0 0
1 0 2 0 0
0 0 0 0 0
```

【样例输出】

```
4
```

【样例说明】

对于输入的房间布局和箱子、搬运工起始位置，搬运工最少需要推动箱子 4 格才能将箱子推到指定位置。测试数据存放在 `in.txt` 文件中。

【评分标准】

该题目有 10 个测试用例，每通过一个测试得 10 分。

答案：

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <queue>
using namespace std;
#define MAXN 10
#define INF 0x3f3f3f3f
struct QNode          //队中元素的类型
{ int x, y;           //搬运工的坐标
    int bx, by;       //箱子的坐标
    int step;         //推箱子步数
};
int dx[]={1,-1,0,0};         //x方向偏移量
int dy[]={0,0,1,-1};         //y方向偏移量
int grid[MAXN][MAXN];
int len[MAXN][MAXN][MAXN][MAXN];  //搬运工的坐标和箱子的坐标
int m,n;
bool Judgeperson(QNode p)      //判断位置p中搬运工的坐标是否有效
{   return p.x>=0 && p.x<m && p.y>=0 && p.y<n
            && grid[p.x][p.y]!=1 && p.step < len[p.x][p.y][p.bx][p.by];
}
bool Judgebox(QNode p)       //判断位置p中箱子的坐标是否有效
{   return p.bx>=0 && p.bx<m && p.by>=0 && p.by<n && grid[p.bx][p.by]!=1
            && p.step<len[p.x][p.y][p.bx][p.by];
}
int BFS(QNode st)         //从st开始搜索
{   queue<QNode> qu;
    qu.push(st);
    len[st.x][st.y][st.bx][st.by]=0;
    int ans=INF;
    while (!qu.empty())
    {   QNode p=qu.front(); qu.pop();    //出队一个元素p
        if(grid[p.bx][p.by]==3)      //找到箱子的目标位置
        {   ans=min(ans,p.step);
            continue;        //队不空时继续搜索
        }
        for(int di=0;di<4;di++)
        {   QNode np=p;       //搬运工向前走一步
            np.x+=dx[di];
            np.y+=dy[di];
            if (Judgeperson(np))      //判断搬运工走一步是否合法
            { if(np.x==np.bx && np.y==np.by) //搬运工和箱子重合
```

```
43                  {   np.bx+=dx[di];     //箱子沿着di方位走一步
44                      np.by+=dy[di];
45                      np.step++;      //推动箱子一次
46                      if (Judgebox(np))    //判断箱子前进是否合法
47                      {   len[np.x][np.y][np.bx][np.by]=np.step;
48                          qu.push(np);
49                      }
50                  }
51                  else         //搬运工和箱子尚未重合
52                  {   len[np.x][np.y][np.bx][np.by]=np.step;
53                      qu.push(np);
54                  }
55              }
56          }
57      }
58      return ans;
59  }
60  int main()
61  {
62      ifstream inFile;
63      inFile.open("in.txt", ios::in);
64      if(!inFile){
65          cout << "error open in.txt!" << endl;
66      }
67
68      QNode st;          //初始状态
69
70      memset(len,0x3f,sizeof(len));     //len所有元素初始化为INF
71      while (!inFile.eof()) {
72          inFile >> m >> n;
73          for (int i = 0; i < m; i++)
74              for (int j = 0; j < n; j++) {
75                  inFile >> grid[i][j];
76                  if (grid[i][j] == 4)      //搬运工的初始位置
77                  {
78                      st.x = i;
79                      st.y = j;
80                      st.step = 0;
81                  } else if (grid[i][j] == 2)     //箱子的初始位置
82                  {
83                      st.bx = i;
84                      st.by = j;
```

```
                }
            }
        int ans = BFS(st);
        if (ans == INF) ans = -1;
        printf("%d\n", ans);
    }
    return 0;
}
```

解析：本题主要考察的是广度优先搜索 BFS。本题有两个关键变量，工人位置坐标和箱子位置坐标。题目中最多一共只会出现 $(nm)^2$ 种状态，也就是箱子行数 × 箱子列数 × 工人行数 × 工人列数。

因此我们对于每个状态，记录从开始状态到它的距离。每次搜索时，检查工人是否能向四个方向（上下左右）移动，移动是否会影响箱子位置，移动的方向用 $(0,1),(0,-1),(1,0),(-1,0)$ 表示。

可以用 *len*[工人行][工人列][箱子行][箱子列] 来存储从出发点到这个状态需要经过的**最小步数**，如果更新相邻点时发现可以做到以更小的步数到达这个点，就更新这个点的新距离。

本题主要复杂之处在于**检查工人是否能向四个方向移动，移动是否会影响箱子位置**，需要较为细致的逻辑判断。要注意输入时的数据应当以合适的格式存储。

习题 4 找最小费用环

【问题描述】

杭州有 n 个景区，景区之间有一些双向的路来连接，现在万先生想找一条旅游路线，这个路线从 A 点出发并且最后回到 A 点，假设经过的路线为 v_1，v_2，...，v_k，v_1，那么必须满足 $k>2$，就是说除了出发点以外至少要经过两个其他不同的景区，而且不能重复经过同一个景区。现在万先生需要你帮他找一条这样的路线，并且花费越少越好。

【输入形式】

第一行是两个整数 n 和 m（$n \le 100$，$m \le 1000$），代表景区的个数和道路的条数。接下来的 m 行每行包括 3 个整数 a，b，c，代表 a 和 b 之间有一条通路，并且需要花费 c 元（$c \le 100$）。

【输出形式】

对于每个测试用例，如果能找到这样一条路线的话，输出花费的最小值。如果找不到的话，输出”It's impossible.”。

【样例输入】

```
3 3
1 2 1
2 3 1
1 3 1
```

【样例输出】

```
3
```

【样例说明】

输入的数据包含 3 个景区和 3 条道路，可以找到一条旅游路线，其最小代价为 3。测试数据存放在 in.txt 文件中。

【评分标准】

该题目有 10 个测试用例，每通过一个测试得 10 分。

答案：

```cpp
#include<iostream>
#include <fstream>
#include<algorithm>
using namespace std;
#define INF 0x3f3f3f3f
typedef long long LL;
int n,m;
LL mat[105][105];            //邻接矩阵数组
LL A[105][105];             //存放两顶点之间最短路径长度的A数组
LL Floyd()              //Floyd求最小环长度
{ LL ans=INF;
    for(int k=1;k<=n;k++)
    { for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                if(i!=k && j!=k && i!=j)       //保证i,j,k不相同，环中至少3个顶点
                    ans=min(ans,A[j][i]+mat[i][k]+mat[k][j]); //求最小环长
        for(int i=1;i<=n;i++)          //更新A[i][j]
            for(int j=1;j<=n;j++)
                A[i][j]=min(A[i][j],A[i][k]+A[k][j]);
```

```
    }
    return ans;
}
int main()
{
    ifstream inFile;
    inFile.open("in.txt", ios::in);
    if(!inFile){
        cout << "error open in.txt!" << endl;
        return 1;
    }
    if(!inFile.eof())
    {
        inFile >> n >> m;
        for(int i=1;i<=n;i++)          //mat和A初始化
            for(int j=1;j<=n;j++)
                A[i][j]=mat[i][j]=INF;
        int a,b;
        LL c;
        for(int i=1;i<=m;i++)
        { inFile >> a >> b >> c;
            mat[a][b]=min(mat[a][b],c);       //取最小值
            mat[b][a]=mat[a][b];
            A[a][b]=A[b][a]=mat[a][b];
        }
        LL ans=Floyd();
        if (ans==INF)
            printf("It's impossible.\n");
        else
            printf("%lld\n",ans);
    }
    inFile.close();
    return 0;
}
```

解析：Floyd 算法主要是用中间点更新其余点之间的最短路的，每当一个点作为中间承接点 k 更新时，最短路径上就会有一个点被安放在指定的位置，直到最短路径上所有点都归位。

而 Floyd 并没有避免掉“自环”、“二元环”这两种情况，所以我们要手动避免 Floyd 更新出环的情况。避免的方式就是当且仅当 $i \neq j, j \neq k$ 时更新，这样就一

定不会更新出小于三个点的环了。此外也要排除自环的情况，这样最后每个点的 $A[i][i]$ 就是题目所求“环路”的结果了。

习题 5 图最短路径算法

【问题描述】

给定 n 个村庄，如果村庄 i 与 j 之间有路联通，则将 i，j 之间连上边，边的权值 W_{ij} 表示这条路的长度。

现在选定一个村庄建医院，设计一个算法求出该医院应该建在哪个村庄，才能使距离医院最远的村庄到医院的路程达到最短。

【输入形式】

第一行输入村庄个数 N（$3 < N < 10$）

之后输入邻接矩阵，如果 i，j 之间没有直接的通路，W_{ij} 为 0

W_{11}，W_{12}，$\ldots W_{1n}$

$\ldots$

W_{n1}，W_{n2}，$\ldots W_{nn}$

【输出形式】

输出选定的村庄的序号

【样例输入】

```
4
0 3 4 0
3 0 0 2
4 0 0 1
0 2 1 0
```

【样例输出】

```
2
```

答案：

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int A[15][15];
int main()
{

```

```
    ifstream inFile;
    inFile.open("in.txt", ios::in);
    if(!inFile){
        cout << "error open in.txt!" << endl;
        return 1;
    }
    if(!inFile.eof())
    {
        int n;
        inFile >> n;
        for(int i=1;i<=n;i++)          //A输入的初始化
            for(int j=1;j<=n;j++)
            {
                inFile>>A[i][j];
                if(A[i][j]==0)       //说明不连通
                    A[i][j]=1e9;
            }
        for(int k=1;k<=n;k++)        //Floyd算法
            for(int i=1;i<=n;i++)
                for(int j=1;j<=n;j++)
                    if(A[i][j]>A[i][k]+A[k][j])
                        A[i][j]=A[i][k]+A[k][j];
        int mn=1e9,mnn=0;//记录全局最大值中的最小值
        for(int i=1;i<=n;i++)
        {
            int mx=0;
            for(int j=1;j<=n;j++)
                if(A[i][j]>mx)
                    mx=A[i][j];
            if(mx<mn)
            {
                mn=mx;
                mnn=i;
            }
        }
        cout<<mnn;
    }
    inFile.close();
}
```

解析：最简单的做法仍然是 Floyd，因为它可以找到任意两点之间的距离，此时只需要判断到这个点最远的村庄距离，并进行比较就可以了。

对于 $\max_{1\le j\le n} F[i][j]$，要找到使这个值最小的 i，就是本题的答案。

Dijkstra 和 Bellman-Ford 等算法也可以做，但是稍显麻烦。每次仍然是对每个点求单源最短路，并记录最大值和更新。

总结

题目：图

日期：2024 年 6 月 18 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：对 Dijkstra 的概念掌握，概念和代码并重，都要能够理解。

习题 2：根据拓扑排序的原理，进行排除和判断。

习题 3：推箱子是一道有一定难度的复杂题目，以 BFS 的思路为基础，记录路径的长度和更新。题目主要障碍在于判断是否能够通行。有同学代码存在雷同，给这部分同学本题扣掉了一半的分数。

习题 4：考察对 Floyd 算法的应用和改进，不要只局限于传统的算法，要理解其本质。

习题 5：本题重点在于计算出最短路之后如何对其合理应用，并高效处理计算结果。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 12 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：查找
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

下列关键字序列不可能是二叉排序树的查找路径的为（　　）
A.95，22，91，24，94，71
B.92，20，91，34，88，35
C.21，89，77，29，36，38
D.12，25，71，68，33，34

答案：A

解析：对于二叉排列树的任一子树，它的左子树上的节点都大于根，右子树上的节点都小于根。因此可以从前到后根据节点数字逐步缩小范围。

以 A 选项为例，**从 95 开始**，22 小于 95，所以查找范围确定在 $(-\infty, 22]$ 或 $[22, 95]$（分别可能是 22 的左右子树）。

此后遇到 91，所以把范围定在 $[22, 91]$ **或** $[91, 95]$。

接下来遇到 24，发现 24 落在 $[22, 91]$，所以范围确定在 $[22, 24]$ 或 $[24, 91]$。

接下来遇到 94，发现 94 大于 91，无论如何也不会落在 $[22, 24]$ 或 $[24, 91]$，所以 A 选项不可能是二叉排列树的查找路径。

分析 B 选项是同样的，**先确定在** $(-\infty, 20]$ **或** $[20, 92]$，**接着遇到 91**，分裂为 $[20, 91]$ 或 $[91, 92]$，**再遇到 34**，分裂为 $[20, 34]$ 或 $[34, 91]$，**接着 88**，分裂为 $[34, 88]$ 或 $[88, 91]$，**再遇到 35**，落在 $[34, 88]$。查找到最后一个元素的正确位置，所以 B 正确。

分析 C 选项，**先确定在** $[21,89]$ **或** $[89,\infty)$，**接着遇到 77**，分裂为 $[21,77]$ 或 $[77,89]$，**再遇到 29**，分裂为 $[21,29]$ 或 $[29,77]$，**接着 36**，分裂为 $[29,36]$ 或 $[36,77]$，**再遇到 38**，落在 $[36,77]$。C 正确。

D 选项同理，**先确定在** $[12,25]$ **或** $[25,\infty)$，**接着遇到 71**，分裂为 $[25,71]$ 或 $[71,\infty)$，**再遇到 68**，分裂为 $[25,68]$ 或 $[68,71]$，**接着 33**，分裂为 $[25,33]$ 或 $[33,68]$，**再遇到 34**，落在 $[33,68]$。C 正确。

所以要看当前查找元素是否能在目前可能的范围内找到，如果找不到，那么这个序列就不可能是二叉排列树的查找路径。

习题 2 单选题

下列选项中，错误的是（　　）。

A. 红黑树中，任何一个结点的左右子树高度之差不超过 2 倍

B. AVL 树中，任何一个结点的左右子树高度之差不超过 1

C. 红黑树的查找效率要优于 AVL 树

D. 红黑树和 AVL 树的查找、插入和删除操作的最坏时间复杂度相同

答案：C

解析：排除法，并比较概念。

A 项：红黑树保证了从任一节点到其每个叶子的所有路径包含相同数目的黑色节点，而两个黑色节点之间最多有一个红色节点，因此最长路径不超过最短路径的两倍。

B 项：此选项描述了 AVL 树的一个核心属性，即任何节点的左右子树高度之差不超过 1。这是 AVL 树定义的基础，用于确保树的平衡性，从而优化查找效率。这个描述是正确的。

C 项：AVL 树通常在查找操作上比红黑树更有效。红黑树在插入和删除操作中会进行更少的调整，可能在这些操作上比 AVL 树稍优，但如果要考虑查找效率，因为 AVL 树和红黑树的平衡性不同，并不能做出严格的判断。

D 项：尽管 AVL 树和红黑树在平衡策略上有所不同，但它们都提供 $\log(n)$ 的时间复杂度保证，因为两者都是自平衡的二叉搜索树，可以让树的深度维持在对数级别。

习题 3 单选题

从 19 个元素中查找其中某个元素，如果最多进行 5 次元素之间的比较，则采用的查找方法只可能是（　　）。

A. 折半查找

B. 分块查找

C. 顺序查找

D. 都不可能

答案：A

解析：$n = 19$，折半查找的元素最多比较次数 $= \lceil \log_2(n+1) \rceil = 5$，顺序查找和分块查找所需元素比较次数会更多。

参考教材 9.2.2，折半查找的比较树高度是 $\lceil \log_2(n+1) \rceil = 5$，从根节点出发，刚好在每一个节点处比较一次，因此最多比较次数是 5。

而对于分块查找，一般查找的时间复杂度是 $O(\sqrt{n})$，因为会将整个序列分成约 $\sqrt{n}$ 块，需要先用 $O(\sqrt{n})$ 次左右找到所在块，然后还要具体找到位置，所以最多比较次数会超过 5。

补充说明：对于 $n = 19$，分块查找当把数据分为 4 块或 5 块时，可以把最多查找次数控制在 9。最坏情况下需要先花 $\sqrt{n}$ 次找到所在块，然后再花 $\sqrt{n}$ 找到具体位置。

顺序查找最坏需要用 n 次，因此也不可能。

习题 4 单选题

采用分块查找，如果线性表一共有 625 个元素，查找每个元素的概率相同，假设采用顺序查找来确定结点所在的块，每块分为（　　）个结点最佳

A.9

B.25

C.6

D.625

答案：B

解析：结论是块数取 $\sqrt{n}$ 附近的值，下为分析过程。

采用分块查找的时候，我们的目的是最小化整体查找的平均时间。在这种情况下，我们假设线性表被分成若干个块，并且采用顺序查找来确定元素所在的块，然后在该块内使用顺序查找来找到具体的元素。

如果线性表共有 n 个元素，分成 b 个块，每个块有 k 个元素，其中 $k = n/b$。平均查找时间 T 可以表示为**确定块的时间加上在块内查找的时间**：

$$T = \frac{b}{2} + \frac{k}{2}$$

我们希望最小化这个时间。将 k 代为 n/b 得：

$$T = \frac{b}{2} + \frac{n/b}{2}$$

为了找到最优的 b（即每个块的大小 k），我们对 T 关于 b 进行求导，并令导数等于 0 找到最小值。

$$\frac{\mathrm{d}}{\mathrm{d}b}(\frac{b}{2} + \frac{n/b}{2}) = 0$$

计算后得：

$$\begin{aligned} \frac{1}{2} - \frac{n}{2b^2} &= 0 \\ b^2 &= n \\ b &= \sqrt{n} \end{aligned}$$

在本题中 $n = 625$，所以 $b = \sqrt{625} = 25$。这表示最佳的情况是每块包含 25 个元素。因此选 B。

另一种理解方式是，查找块的位置要查询 a 次，在块内确定位置要查询 b 次，总查询次数为 $a + b$。而 ab 是恒定情况下，要最小化 $a + b$，显然是基本不等式取最值时最优，也就是 $a = b = \sqrt{ab} = \sqrt{n}$

习题 5 单选题

数据元素有序元素个数较多，且元素固定不变的情况下，应该采用（　　）法

A. 折半查找

B. 分块查找
C. 二叉排序树查找
D. 顺序查找

答案：A

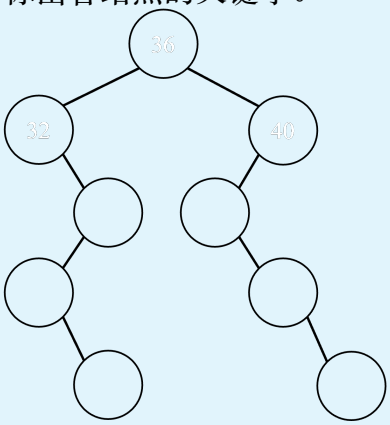
解析：在处理有序且固定不变的大量数据元素时，折半查找（也称为二分查找）是非常高效的查找方法，其时间复杂度为 $O(\log n)$。这是因为折半查找可以在每一步将查找范围缩小到一半，从而大幅度减少比较的次数。

分块查找的效率比折半查找稍低，但较为方便统计其他信息，适用于数据一定程度上动态变化的情况。

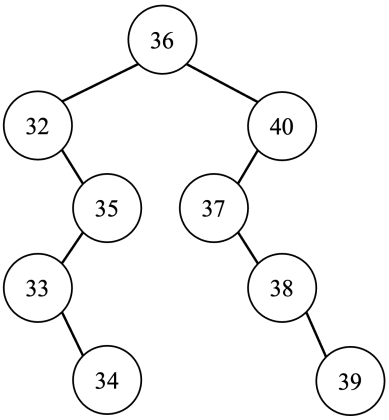
二叉排序树查找的效率相对折半查找不够有保证，顺序查找则效率更低。

习题 6 简答题

一棵二叉排序树的结构如图所示，其中各结点的关键字依次为 $32 \sim 40$，请标出各结点的关键字。

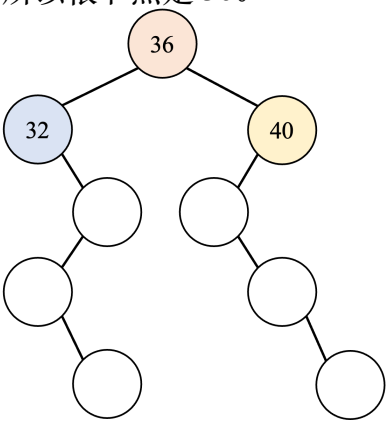


答案：



解析：本题有两种主要的思路可以参考。第一种是递归地看当前节点左子树有多少节点、右子树有多少节点。因为元素都在 $32 \sim 40$ 的范围，所以每次可以确定根节点的值是多少。

也就是在下图中，根节点左边是 4 个节点，说明 $32 \sim 40$ 里有 4 个数字比它小，所以根节点是 36。



另一种方式是先找到最小值。在二叉排序树中，最小值一定是顺着根节点的左儿子一直走下去，直到左儿子为空。所以 32 应该填在图上蓝色位置。

接着可以找到次小值，如果最小值有右儿子，那么最小值就在右子树里面，同样顺着根节点往左走即可；如果没有右儿子，则需要进行回溯。这种方式较为复杂，如果是填空题的话需要多做验证。

同理最大值 40 也可以用这种方式确定出来。

习题 7 求中位数

【问题描述】

给定 n 个整数 $x_1, x_2, \ldots, x_n$，计算每对整数的差值 $|x_i - x_j| (1 \le i < j \le n)$，可以得到 $\mathrm{C}(n, 2)$ 个差值，现在你的任务是尽快找到这些差值的中位数。注意在此问题中，如果差值的个数 m 为偶数，则中位数定义为第 $m/2$ 个最小数，例如 $m = 6$ 时需要求第 3 个最小数。

【输入形式】

在每个测试用例中，第一行给出 n，然后给出 n 个整数，分别代表 x_1，x_2，$\ldots$，x_n $(x_i \le 1000000000$，$3 \le n \le 100000)$。

【输出形式】

在单独的行中输出中位数。

【样例输入】

```
4
1 3 2 4
```

【样例输出】

```
1
```

【样例说明】

测试数据的文件名为 `in.txt`

【评分标准】

该题目有 10 个测试用例，每通过一个测试得 10 分。

答案：

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
const int MAXN=1000005;
int x[MAXN];
int n,m;
bool Judge(int mid)      //如果bigcnt<=m/2就满足条件，返回true
{
    int bigcnt=0;      //x+n作为尾端指针减去mid所在的位置，得到的是大于mid的个数
    for(int i=0;i<n;i++)
```

```
12          bigcnt+= x+n-upper_bound(x+i+1,x+n,x[i]+mid);
13      return bigcnt<=m/2;
14  }
15  int BinSearch()       //二分查找
16  {   int low=0,high=x[n-1]-x[0];
17      while(high-low>1)     //直到low和high相差1时，停止二分
18      { int mid=(low+high)/2;
19          if (Judge(mid))
20              high=mid;
21          else
22          low=mid;
23      }
24      return high;
25  }
26  int main()
27  {
28      freopen("in.txt","r",stdin);
29      scanf("%d",&n);
30      for(int i=0;i<n;i++)
31          scanf("%d",&x[i]);
32      m=n*(n-1)/2;
33      sort(x,x+n);
34      int ans=BinSearch();
35      printf("%d\n",ans);
36      return 0;
37  }
```

解析：为了确定中位数，我们可以使用二分判断的方式。假设我们检查的值为 *mid*，就可以去判断差值超出 *mid* 的数对有多少个。因此先对输入数据进行排序，这样每个数与其他数的差值大小就更方便计算。使用 `upper_bound` 可以快速找出与当前数差值超出 *mid* 的数边界位置。最后如果超出 *mid* 的数对大于 $m/2$，就将 *mid* 定为下界，否则定为上界。

习题 8 前 m 大的数

【问题描述】

给定一个包含 N（$N \leq 3000$）个正整数的序列，每个数不超过 5000，对它们两两相加得到 $N \times (N-1)/2$ 个和，求出其中前 M 大的数（$M \leq 1000$）并按从大到小的顺序排列。

【输入形式】

输入可能包含多组数据，其中每组数据包括两行，第一行两个数 N 和 M，第二行 N 个数，表示该序列。

【输出形式】

输入可能包含多组数据，其中每组数据包括两行，第一行两个数 N 和 M，第二行 N 个数，表示该序列。

【样例输入】

```
4 4
1 2 3 4
```

【样例输出】

```
7 6 5 5
```

【样例说明】

测试数据的文件名为 in.txt

【评分标准】

该题目有 5 个测试用例，每通过一个测试得 20 分。

答案：

```cpp
#include<iostream>
#include<unordered_map>
using namespace std;
#define MAXN 3005       //数据数量最大值
#define MAXV 10001      //数据差值可能的最大值
int a[MAXN];
int main()
{
    int n,m;
    freopen("in.txt","r",stdin);
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        unordered_map<int,int>hmap;//将int作为关键字进行映射
        for(int i=0;i<n;i++)
            cin>>a[i];
        for(int i=0;i<n;i++)
            for(int j=i+1;j<n;j++)
                hmap[a[i]+a[j]]++;
        bool first=true;
        int i=MAXV; //从大到小查询是否有这个元素
```

```
21          while(i>0&&m>0)
22          {
23              if(hmap[i]!=0)//如果这个元素出现过，就把出现次数-1，输出一下
24              { if(first)
25                  { cout<<i;
26                      first=false;
27                  }
28                  else cout<<' '<<i;
29                  hmap[i]--;
30                  m--;
31              }
32              else i--;
33          }
34          cout<<endl;
35      }
36  }
```

解析：因为序列长度只有 3000，所以可以用 n^2 次将所有数对的和进行相加。又因为数的大小是不超过 5000 的，相加结果不会超过 10000。那么就从 10000 从大到小找计算结果是否出现，出现过多少次就输出多少次，直到输出够 M 次。

当然，这里基本没有考察查找的知识点，主要是对 map/unordered_map 数据结构的应用。如果期末考试要求大家不使用 STL，本题可能需要大家自己写哈希表来完成 map 的功能。

本题还有一种思路，就是对所有数进行排序，然后利用二分查找来确定有多少个数对的和大于等于某个值 *mid*，这样可以用减少时间复杂度做到 $O(n\log^2 n)$，感兴趣的同学可以找助教学习这种做法。

总结

题目：查找

日期：2024 年 6 月 21 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：本题主要考察对二叉排序树的查找路径的理解，需要掌握二叉排序树的性质和特点。

习题 2：考察对 AVL 树和红黑树的理解，主要是概念，基本不会考代码。所以要区分这些高级数据结构特点和它们之间的区别。

习题 3：考察对不同查找方法的复杂度掌握情况，此外还需要对查找细节有一些了解，才能确定到具体的数字。期末考试的填空题更有可能考的是数字，所以希望大家能够尽量了解各算法的细节。

习题 4：主要是对分块的理解。其实这个题可以盲猜，或者记住结论，分块就是开根号。

习题 5：考察对不同查找方式的理解。

习题 6：考察对二叉排序树的理解，并希望大家掌握一部分查找、建树的技巧。

习题 7：二分查找有相当多的例题和模型，可以多体会。二分的题目最主要的类型是：**使得** *xxx* **满足条件时，** *yyy* **最小**。这时只需要大家完成检验的 `judge` 函数就可以了。

习题 8：主要考察哈希表的建立或者对 `map` 的掌握。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
作业 13 参考答案

作业情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：排序
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

习题 1 单选题

对序列（15，9，7，8，20，-1，4）进行排序，进行一趟后数据的排列变为（4，9，-1，8，20，7，15），则采用的是（　　）排序

A. 简单选择排序
B. 冒泡排序
C. 希尔排序
D. 快速排序

答案：C

解析：A 项，选择排序的过程需要把最小的元素找到并放在前面，所以排除；

B 项，冒泡排序过程也较容易模拟，显然不符合这一过程，可以排除；

D 项，快速排序需要找一个基准数字，但可以发现这一趟排序没有符合要求的数字。

这一趟排序中，第 1,4,7；2,5；3,6 这三组元素都变得有序，可以是增量为 3 的希尔排序。

习题 2 单选题

数据序列（8，9，10，4，5，6，20，1，2）只能是（　　）算法的两趟排序后的结果

A. 简单选择排序

B. 冒泡排序
C. 直接插入排序
D. 堆排序

答案：C

解析：排除法，比较各算法细节。

A 项：如果是选择排序，那么每次会找最小值放在前面，两趟过后前面两个位置一定分别是最小值和次小值。可以排除。

B 项：冒泡排序是指每趟交换相邻的两个元素，而且最大值一定会在第一趟被冒到最右侧，此时 20 仍然不在最右的位置，所以排除。

C 项：如果是直接插入排序，那么第一趟排序后，前两个元素一定是有序的，符合题意。

D 项：堆排序的根节点一定是最小值，可以排除。

习题 3 单选题

在下列排序方法中，若待排序的数据已经有序，花费时间反而最多的是（　　）

A. 快速排序
B. 希尔排序
C. 冒泡排序
D. 堆排序

答案：A

解析：快速排序在待排序的数据已经有序时，其性能可能会变得最差，考虑到其对数据基准点（pivot）的选择方式。

快速排序的时间复杂度和其基准点的选择密切相关。在常见的快速排序实现中，如果每次都选择最左边或最右边的元素作为基准点，当这个数组已经有序时，就会导致分割不平衡，无法砍半以达到优化复杂度的目的。每次划分只能减少一个元素，这会导致算法退化为 $O(n^2)$ 的时间复杂度。

冒泡排序的最坏复杂度永远都是 $O(n^2)$，不受影响。

希尔排序建立在插入排序基础上，如果有序时算法会更快。

堆排序的时间复杂度无论如何不会超过 $O(n \log n)$，不受影响。

习题 4 单选题

在下列排序方法中，执行时间不受数据初始状态影响，总为 $O(n\log_2 n)$ 的是（　　）

A. 堆排序

B. 冒泡排序

C. 简单选择排序

D. 快速排序

答案：A

解析：对于堆排序，无论如何都要检查每一个位置是否满足堆的条件，所以其时间复杂度是 $O(n\log n)$。

对于冒泡排序，当数据有序时只需要 $O(n)$；当数据无序时仍然需要 $O(n^2)$，两个都不满足题意。

简单选择排序的时间复杂度无论如何都是 $O(n^2)$，不满足题意。

快速排序的时间复杂度是 $O(n\log n)$，但是在数据有序时可能会退化为 $O(n^2)$（参考上一题），所以也不满足题意。

习题 5 单选题

在下列排序方法中，某一趟结束后未必能选出一个元素放在其最终位置上的是（　　）

A. 堆排序

B. 冒泡排序

C. 直接插入排序

D. 快速排序

答案：C

解析：A 项，堆排序的根节点一定是最小/最大值，所以有确定元素。

B 项，最大的数字一定在第一趟就到了最后一个位置，所以有确定元素。

C 项，插入排序在第 i 趟只会考虑前 $i+1$ 个数据，而后面的元素还可能对已经排好的那些元素造成比较大的影响，比如出现了最小的元素，就会把已排序的所有值都往后移动，因此错误。

D 项，一趟快速排序之后，基准点的位置就确定了，因为左边都是比它小的，

右边都比它大。

习题 6 快速排序

【问题描述】

有一个含 n（$n \leq 200000$）个整数的无序序列，采用快速排序实现递增排序

【输入形式】

一行字符串，包含多个整数，每个数之间用空格分开。

【输出形式】

递增排序的结果，每个数之间用空格分开。

【样例输入】

```
9 4 7 6 2 5 8 1 3
```

【样例输出】

```
1 2 3 4 5 6 7 8 9
```

【样例说明】

测试数据的文件名为 `in.txt`，输出文件名为 `out.txt`

【评分标准】

该题目有 10 个测试用例，每通过一个测试用例得 10 分

答案：参考快速排序的代码如下：

```cpp
#include<bits/stdc++.h>
#include <string>
#include <stdio.h>
#include <vector>

using namespace std;
int partion(vector<int>& nums, int start, int end)
{   //分治合并过程
    int target = nums[start];
    int i = start, j = end;
    while (i < j)
    {
        while (i<j && nums[j]>target) j--;
        while (i < j && nums[i] <= target) i++;
        if (i < j)
            swap(nums[i], nums[j]);
    }
    swap(nums[i], nums[start]);
```

```
    return i;
}

void quickSort(vector<int>& nums, int start, int end)
{   //快速排序递归过程
    if (start < end)
    {
        int i = partion(nums, start, end);
        quickSort(nums, start, i - 1);
        quickSort(nums, i + 1, end);
    }
}

int main()
{
    freopen("in.txt", "r", stdin);
    vector<int> nums;
    int num;
    while (cin >> num)
    {
        nums.push_back(num);
    }
    freopen("out.txt", "w", stdout);
    quickSort(nums, 0,nums.size()-1);
    int i = 0;
    for (; i < nums.size() - 1; i++)
    {
        cout << nums[i] << ' ';
    }
    cout << nums[i] << '\n';;
    return 0;

}
```

解析：请注意掌握快速排序的算法和思路，并对其实现有比较扎实的理解。快速排序的基本思路是选取一个基准点，然后将比基准点小的数放在左边，比基准点大的数放在右边，然后递归地对左右两边进行排序。

但为了避免时间过长，每次选择基准点可以不按上述代码写，而是可以随机在**区间内**选择一个点，以免退化为 $O(n^2)$ 的时间复杂度。

习题 7 希尔排序

【问题描述】
有一个含 n（$n \leq 200000$）个整数的无序序列，采用希尔排序实现递增排序
【输入形式】
一行字符串，包含多个整数，每个数之间用空格分开。
【输出形式】
递增排序的结果，每个数之间用空格分开。
【样例输入】
`9 4 7 6 2 5 8 1 3`
【样例输出】
`1 2 3 4 5 6 7 8 9`
【样例说明】
测试数据的文件名为 `in.txt`，输出文件名为 `out.txt`
【评分标准】
该题目有 10 个测试用例，每通过一个测试用例得 10 分

答案：参考快速排序的代码如下：

```
#include <iostream>
#include <string>
#include <fstream>
#include <stdio.h>
#include <vector>

using namespace std;
void shellSort(vector<int>& R, int n)
{
    int d = n / 2;
    while (d > 0)//选择增量
    {
        for (int i = d; i < n; i++)
        {
            if (R[i] < R[i - d])
            {
                int tmp = R[i];
                int j = i - d;
                do
```

```
                {
                    R[j + d] = R[j];
                    j = j - d;
                } while (j >= 0 && R[j] > tmp);
                R[j + d] = tmp;
            }
        }
        d = d / 2;
    }
}

int main()
{
    freopen("in.txt", "r", stdin);
    vector<int> nums;
    int num;
    while (cin >> num)
    {
        nums.push_back(num);
    }
    freopen("out.txt", "w", stdout);
    shellSort(nums, nums.size());
    int i = 0;
    for (; i < nums.size() - 1; i++)
    {
        cout << nums[i] << ' ';
    }
    cout << nums[i] << endl;;
    return 0;

}
```

解析：希尔排序的增量变化有很多实现方式，掌握其中一个即可。主要是要注意插入排序的过程不要写错，然后难点在于下标和增量的变化。

习题 8 快速排序

【问题描述】

有一个含 n（$n \leq 200000$）个整数的无序序列，设计一个算法利用快速排序思路求前 10 个最大的元素。

【输入形式】

一行字符串，包含多个整数，每个数之间用空格分开。

【输出形式】

前 10 个最大的元素，按递减排序，用空格分开。

【样例输入】

```
1 5 32 4 6 8 9 4 7 6 55 1 3 65 24
```

【样例输出】

```
65 55 32 24 9 8 7 6 6 5
```

【样例说明】

测试数据的文件名为 in.txt，输出文件名为 **out.txt**

【评分标准】

该题目有 10 个测试用例，每通过一个测试得 10 分。

答案：实现细节同习题 6，此处仅给出输入输出部分。

```
int main()
{
    freopen("in.txt", "r", stdin);
    vector<int> nums;
    int num;
    while (cin >> num)
    {
        nums.push_back(num);
    }
    freopen("out.txt", "w", stdout);
    quickSort(nums, 0, nums.size() - 1);
    int i = nums.size() - 1;
    for (; i > nums.size() - 10; i--)
    {
        cout << nums[i] << ' ';
    }
    cout << nums[i] << endl;;
    return 0;

}
```

解析：算法部分为快速排序，在输入输出时和其他题有一点点区别，可以注意一下。最后只按从大到小顺序输出 10 个数字即可。

习题 9 前 m 大的数

【问题描述】

给你 n 个整数，请按从大到小的顺序输出其中前 m 大的数。

【输入形式】

每组测试数据有两行，第一行有两个数 n 和 m（$0 < n$，$m < 1000000$），第二行包含 n 个各不相同，且都处于区间 $[-500000, 500000]$ 的整数。

【输出形式】

对每组测试数据按从大到小的顺序输出前 m 大的数。

【样例输入】

```
5 3
3 -35 92 213 -644
```

【样例输出】

```
213 92 3
```

【样例说明】

对于输入的 5 个整数，前 3 大的整数分别是 213、92、3。测试数据存放在 `in.txt` 文件中。

【评分标准】

该题目有 10 个测试用例，每通过一个测试得 10 分。

答案：如果想学习 `sort` 的使用，可以参考下面的代码：

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <functional>
#include <algorithm>

using namespace std;

int main()
{
    ifstream inFile;
    inFile.open("in.txt", ios::in);

    if(!inFile){
        cout << "error open in.txt!" << endl;
```

```
        return 1;
    }

    int n,m,x;
    vector<int> myv;
    if(!inFile.eof())
    {
        inFile >> n >> m;
        myv.clear();
        for(int i=0;i<n;i++)
        {
            inFile >> x;
            myv.push_back(x);
        }
        sort(myv.begin(),myv.end(),greater<int>());
        for(int i=0;i<m;i++)
        { if (i>0) printf(" ");
            printf("%d",myv[i]);
        }
        printf("\n");
    }
    inFile.close();
    return 0;
}
```

解析：本题思路同前面第八题。只需要输出前 m 个数即可。可以使用库中的 sort 函数，也可以自行实现快速排序。

总结

题目：排序

日期：2024 年 6 月 22 日

批改人：王骏峣

邮箱：wjyyy1@126.com

习题 1：对各种排序过程的掌握。

习题 2：考察排序过程和一部分逆向排序思维，主要是要考虑是否有没想到的情况。

习题 3：注意对快速排序**缺点**的掌握，快速排序运行速度很快，但缺点就是容易被有序数据这种常见问题卡掉复杂度。

习题 4, 5：对概念的理解。

习题 6, 7：对不同排序算法思路的检查，以及对排序算法的实现。

习题 8, 9：对不同算法实际应用的考核。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。

数据结构 A
期中考试参考答案与分析

试题情况

教材：数据结构教程（C++ 语言描述）李春葆等
题目范围：线性表、栈、队列、串
邮箱：wjyyy1@126.com
授课教师：彭蓉 教授
助教：王骏峣

试题 1 链表重排

【问题描述】
给定一个单链表 $L_1 \to L_2 \to \cdots \to L_{n-1} \to L_n$，请编写程序将链表重新排列为 $L_n \to L_1 \to L_{n-1} \to L_2 \to \cdots$。例如：给定 L 为 $1 \to 2 \to 3 \to 4 \to 5 \to 6$，则输出应该为 $6 \to 1 \to 5 \to 2 \to 4 \to 3$。
【输入形式】
每个输入包含 1 个测试用例。每个测试用例第 1 行给出第 1 个结点的地址和结点总个数，即正整数 $N(\leq 10^5)$。结点的地址是 5 位非负整数，NULL 地址用 -1 表示。
接下来有 N 行，每行格式为：

```
Address Data Next
```

其中 Address 是结点地址；Data 是该结点保存的数据，为不超过 10^5 的正整数；Next 是下一结点的地址。题目保证给出的链表上至少有两个结点。
【输出形式】
对每个测试用例，顺序输出重排后的结果链表，其上每个结点占一行，格式与输入相同。
【样例输入】

```
00100 6
00000 4 99999
```

00100 1 12309
68237 6 -1
33218 3 00000
99999 5 68237
12309 2 33218

【样例输出】

68237 6 00100
00100 1 99999
99999 5 12309
12309 2 00000
00000 4 33218
33218 3 -1

【类库使用要求】

可以使用 STL 类库。

【输入类型】

标准输入

【输出类型】

标准输出

【评分标准】

10 个测试用例，按通过比例评分。

【解题思路】（仅供参考，可以有其他解法）

定义链表结点：包含三元组，*addr* 存储该结点本身的地址（可以定义为 `int` 或 `string` 类型，如果是 `int` 类型要注意位数不足时的输出时格式），*value* 存该结点序号，*next* 为指向下一结点的指针。

从输入数据构建链表，根据 *value* 值将其插入到链表相应位置；

按要求重排链表；

输出重排后的链表，输出内容为：“当前节点.addr”“当前结点.value”“下一结点.addr(尾节点输出 -1）”

答案：

```
#include<iostream>
using namespace std;
```

```
int da[100000],ne[100000],node[100000];
bool renewed[100000];//表示这个地址的next被重置过
int main()
{
    int n,head;
    cin>>head>>n;
    for(int i=1;i<=n;i++)
    {
        int ad;//输入地址
        cin>>ad;
        cin>>da[ad]>>ne[ad];
        //将data和next放入相应地址
    }
    int now=head;
    for(int i=1;i<=n;i++)
    {
        node[i]=now;
        now=ne[now];//依次梳理L1L2L3分别是谁
    }
    //让Ln.next指向L1
    //让L1.next指向Ln-1
    //以此类推，Ln-i指向Li+1，Li指向Ln-i
    int t=n,forward=0;
    while(!renewed[t])//避免重复赋值
    {
        renewed[t]=1;
        if(forward==1)
        {
            if(renewed[n-t])
            {//如果发现下一个是找过的位置，就该停止了
                ne[node[t]]=-1;
                break;
            }
            ne[node[t]]=node[n-t];
            forward=0;
            t=n-t;
        }
        else
        {
            if(renewed[n-t+1])
            {//如果发现下一个是找过的位置，就该停止了
                ne[node[t]]=-1;
```

```
                break;
            }
            ne[node[t]]=node[n-t+1];
            forward=1;
            t=n-t+1;
        }
    }
    head=node[n];//从头（Ln）开始输出
    while(head!=-1)
    {
        if(head<10000)
            cout<<0;//不足5位补齐5位
        if(head<1000)
            cout<<0;
        if(head<100)
            cout<<0;
        if(head<10)
            cout<<0;
        cout<<head<<" ";
        cout<<da[head]<<" ";
        if(ne[head]>=0&&ne[head]<10000)
            cout<<0;//注意-1不用补齐
        if(ne[head]>=0&&ne[head]<1000)
            cout<<0;
        if(ne[head]>=0&&ne[head]<100)
            cout<<0;
        if(ne[head]>=0&&ne[head]<10)
            cout<<0;
        cout<<ne[head]<<endl;
        head=ne[head];
    }
    return 0;
}
```

解析：题目要求将链表重新排列为 $L_n \to L_1 \to L_{n-1} \to L_2 \to \cdots$，所以要将每个位置的 *next* 进行修改，同时把表头放在最前面进行输出。

首先输入的链表不是按输入顺序排布的，需要先找到表头，然后依次梳理 $L_1, L_2, \cdots$。然后再将 L_n 放在最前，让其 *next* 指向 L_1（的地址），接着让 L_1 的 *next* 指向 L_{n-1}（的地址），以此类推。

所以本题需要把每个节点初始的编号找到，然后修改它的 *next* 指向即可。最

后输出的时候要注意格式，一方面，每个节点的地址都是 5 位数，不足的要补零；另一方面要按照重排后的链表顺序 $L_n, L_1, \cdots$ 进行输出。

如果理解有问题可以考虑把上述代码的过程中的 *now* 和 *t* 逐个打印出来观察。

或直接参考题目给出的思路。

试题 2 超市模拟

（注：题目好像没有说读入输出方式，按标准输入输出）

【问题描述】

模拟超市排队的行为。最初，有 n 个队列（1，2，...，n），每个队列都有一些顾客。之后可能会发生两个事件：

ENTERS：顾客到达队列。如果队列在 1 到 n 之间，则顾客到达该队列的末尾。否则，该事件将被忽略；

LEAVES：顾客离开队列。如果队列在 1 到 n 之间，并且该队列不为空，则该队列的第一个顾客将离开该队列。否则，该事件将被忽略。

【类库使用要求】可以使用 STL 类库

【输入形式】

1. 输入从队列的数量 n（严格意义上为正的自然数）开始。
2. 按照 n 行，每个队列一行，每个行按照顾客到达队列的顺序，列出顾客名字。
3. 然后空一行。
4. 事件描述（ENTERS 或者 LEAVES）后面，跟着顾客名字和和队列序号（正整数）。

【输出形式】

1. 首先，按离开的顺序打印离开队列的顾客姓名。
2. 然后，按顺序打印 n 个队列的最终内容。

【样例输入】

```
5
Lisa Tom
John
Jerry Mary
```

```
Eric

LEAVES 1
LEAVES 2
ENTERS Harry 2
```

【样例输出】

```
DEPARTS
Lisa
John
FINAL CONTENTS
queue 1: Tom
queue 2: Harry
queue 3: Jerry Mary
queue 4:
queue 5: Eric
```

答案:

```cpp
#include<iostream>
#include<queue>
#include<sstream>
using namespace std;
queue<string> q[100010];
int main()
{
    int n;
    cin>>n;
    string line;
    getline(cin,line);
    for(int i=1;i<=n;i++)
    {
        getline(cin,line);//读入一行
        istringstream in(line);//转为输入流
        string name;
        while(in>>name)
            q[i].push(name);
```

```
    }
    string op;
    cout<<"DEPARTS"<<endl;
    while(cin>>op)
    {
        if(op=="ENTERS")//入队
        {
            string name;
            int q_num;
            cin>>name>>q_num;
            if(q_num<=n&&q_num>=1)//需要判断是否合法
                q[q_num].push(name);
        }
        else
        {
            int q_num;
            cin>>q_num;//需要判断范围合法、队伍非空
            if(q_num<=n&&q_num>=1&&!q[q_num].empty())
            {
                cout<<q[q_num].front()<<endl;
                q[q_num].pop();
            }
        }
    }
    cout<<endl<<"FINAL CONTENTS"<<endl;
    for(int i=1;i<=n;i++)
    {
        cout<<"queue "<<i<<": ";
        while(!q[i].empty())//输出队伍内容
        {
            cout<<q[i].front()<<" ";
            q[i].pop();
        }
        cout<<endl;
    }
    return 0;
}
```

解析：本题可以模拟若干个字符串（string）队列，但是注意输入时要将一整行转为一个字符输入流（istringstream），然后逐段读入。

此外，cin 会忽略空格和空行，如果需要读入空行或包含空格的一整行，需

要使用 `getline`。

试题 3 栈处理数据序列

【问题描述】

从 `txt` 文件输入一个整数序列 $a_1, a_2, a_3, \ldots, a_n$，试编写算法实现，用栈结构存储输入的整数，当 $a_i \neq -1$ 时，将 a_i 进栈，当 $a_i = -1$ 时，使用 `txt` 文件输出栈顶整数并出栈，栈空间为 20，算法应对异常情况时输出错误信息（栈满输出 999、栈空下溢输出 -999 等）。

【类库使用要求】不允许使用 STL 中的容器类，比如 `stack`、`vector`、`list` 等，栈的操作需要自行编写代码。

【输入形式】

文件输入，文件名为 `in.txt`

【输出形式】

文件输出，文件名为 `out.txt`

【样例输入 1】

```
1 2 3 -1 -1 -1 -1
```

【样例输出 1】

```
3
2
1
-999
```

【样例输入 2】

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1
```

【样例输出 2】

```
999
```

【样例输入 3】

```
1 2 3 4 -1 -1 -1 4 5 -1
```

【样例输出 3】

```
4
3
2
```

5

答案：

```cpp
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
    int s[25],tp=0,x;//tp指栈顶指针，tp=0表示为空
    ifstream in("in.txt");//输入流（用于替代cin）
    ofstream ou("out.txt");//输出流（用于替代cout）
    while(in>>x)
    {
        if(x==-1)
        {
            if(tp==0)//无法再出栈，栈空异常
            {
                ou<<"-999";
                return 0;
            }
            ou<<s[tp--]<<endl;//出栈同时弹出顶上元素
        }
        else
        {
            if(tp==20)//无法再入栈，栈满异常
            {
                ou<<"999";
                return 0;
            }
            s[++tp]=x;
        }
    }
    return 0;
}
```

解析：用数组模拟一个栈，用一个整型变量指向栈顶元素的下标，当这个下标为 0 时表示没有元素，即栈空；当这个下标为 20（栈空间大小）时，栈满。

栈空时如果要出栈，就是在让 0 接着减小，这是不合理的，输出 -999；栈满时如果要入栈，就是在让 20 接着增大，这也我们不想看到的，输出 999。

注：指向栈顶元素或栈顶元素的下一个位置都可以，但是要注意区分栈空和

栈满时的情况。同时要注意数组要尽量开大一点，以免数组越界。而栈是否越界就由我们代码里的逻辑来判断。

试题 4 将偶数移至奇数之前，保持相对次序不变

【问题描述】
从 txt 文件读入一个整数数组，数组长度在 0 到 200 之间，设计一个算法，将所有偶数移动到所有奇数的前面，要求它们的相对次序不变，使用 txt 文件输出移动后的结果。
【类库使用要求】不允许使用 STL 中的容器类，比如 queue、list、vector、stack 等，如用到队列、链表的操作需要自行编写代码。
【输入形式】
文件输入，文件名为 in.txt
【输出形式】
文件输出，文件名为 out.txt
【样例输入】
1 2 3 4 5 6 7 8
【样例输出】
2 4 6 8 1 3 5 7

答案：

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
    int q0[205],cnt0=0;//q0按顺序存储偶数，cnt0存储偶数个数
    int q1[205],cnt1=0;//q1按顺序存储奇数，cnt1存储奇数个数
    ifstream in("in.txt");
    ofstream ou("out.txt");
    int x;
    while(in>>x)
    {
        if(x&1)
        {
            cnt1++;
            q1[cnt1]=x;
```

```
            //或合并为q1[++cnt1]=x;
        }
        else
        {
            cnt0++;
            q0[cnt0]=x;
            //或合并为q0[++cnt0]=x;
        }
    }
    for(int i=1;i<=cnt0;i++)
        ou<<q0[i]<<" ";
    for(int i=1;i<=cnt1;i++)
        ou<<q1[i]<<" ";
    return 0;
}
```

解析：实现两个队列，当输入数据是奇数则放入第二个队列，当输入数据是偶数则放入第一个队列。

最后先按顺序输出/弹出第一个队列，然后按顺序弹出第二个队列即可。

总结

题目：2023-2024 学年度第二学期期中考试

日期：2024 年 4 月 22 日

助教：王骏峣

邮箱：wjyyy1@126.com

期中测试大部分同学的问题出在输入输出上。第二次作业中我已经讲解过文件输入输出的基础方法，如果还是不太清楚的话可以直接记住以下几点：

- 标准输入输出的头文件是 `#include<iostream>`，文件输入输出的头文件是 `#include<fstream>`
- 标准输入输出为 `cin>>` 和 `cout<<`，箭头方向如果是输入就指向变量，如果是输出就从变量引出来，理解记忆。
- 当输入从标准输入变成文件输入时，就把 `cin>>` 换成我们自定义的 `in>>`。
  - **读入文件**：使用 `ifstream` 类，`ifstream in("in.txt")`，然后使用 `in>>x` 读入数据。
- 当输出从标准输出变成文件输出时，就把 `cout<<` 换成我们自定义的 `ou<<`。（这里不用 `out` 是担心 `out` 和标准库命名冲突）
  - **输出文件**：使用 `ofstream` 类，`ofstream ou("out.txt")`，然后使用 `ou<<x` 读入数据。

各位同学如有问题欢迎及时在群里提出，或者通过邮件/QQ 联系我。