C5-2021级航类-第五次上机题解

LJF算距离

题目分析

只需要在输入结束后对数组中每一个元素进行遍历，并与当前的最小值进行比较即可。

示例代码

```
#include <stdio.h>
#include <stdlib.h>
int n,ans=3005,x;
int a[1005];
int main(void)
{
    scanf("%d",&n);
    for(int i=0;i<n;i++)
        scanf("%d",a+i);
    scanf("%d",&x);
    for(int i=0;i<n;i++)
        if(abs(a[i]-x)<ans)
            ans=abs(a[i]-x);
    printf("%d\n",ans);
    return 0;
}
```

简单的查找

题目分析

由于本题数据规模较小，对于每一组输入的l, r，我们只需要遍历a[l]到a[r]的每一个数然后找出第3大的就可以了。

找第3大数的方法：

定义三个int型变量`max1,max2,max3`，用来保存第1大、第2大、第3大的数，初值赋为-2147483648。这一步是不可或缺的，因为这n个数的范围在$[-2147483648, 2147483647]$，于是我们需要让max们小于等于最小值从而保证之后的迭代是正确的。

或者，也可以将`a[l]、a[l+1]、a[l+2]`这三个值按大小分别赋给`max1,max2,max3`，只不过需要对这三个数的大小进行分类讨论。

之后我们用`for`循环遍历`a[l]~a[r]`：

```
int i;
for(i=l; i<=r; ++i){

}
```

对于当前的`a[i]`，我们用它来更新`max1,max2,max3`的值：

```
    if(a[i] > max1){
        max3 = max2;
        max2 = max1;
        max1 = a[i];
    }
    else if(a[i] > max2){
        max3 = max2;
        max2 = a[i];
    }
    else if(a[i] > max3)    max3 = a[i];
```

遍历结束后我们输出`max3`就可以了。

示例代码

```
#include<stdio.h>

const int minv = -2147483648;

int n, m;
int a[5005];

int main(){
    int i, j;
```

```
    scanf("%d%d", &n, &m);
    for(i=1; i<=n; ++i)
        scanf("%d", &a[i]);
    for(j=1; j<=m; ++j){
        int l, r, max1=minv, max2=minv, max3=minv;

        scanf("%d%d", &l, &r);

        for(i=l; i<=r; ++i)
            if(a[i] > max1){
                max3 = max2;
                max2 = max1;
                max1 = a[i];
            }
            else if(a[i] > max2){
                max3 = max2;
                max2 = a[i];
            }
            else if(a[i] > max3)    max3 = a[i];

        printf("%d\n", max3);
    }
    return 0;
}
```

宋老师的名次预测3.0

题目分析

在 `名次预测2.0` 的基础上，将每组猜对人数顺序存储在一个数组中，然后对数组元素进行一次冒泡排序，注意是从大到小排序

参考代码

```
#include <stdio.h>
#include <stdlib.h>
int ans[1005];
void guess(int cnt)
{
    int num = 0, tmp;
    for (int i = 1; i <= 50; i++)
    {
        scanf("%d", &tmp);
        num += (tmp == i);
    }
    ans[cnt] = num; //将猜对人数存入数组中
}
void bubble_sort(int a[], int n) //冒泡排序
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (a[j] < a[j + 1]) //从大到小排序
            {
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
}
int main()
{
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
        guess(i);

    bubble_sort(ans, n);

    for (int i = 0; i < n; i++)
```

```
        printf("%d\n", ans[i]);

    return 0;
}
```

参考代码2

计数排序的写法，原理可参考这个

```
#include <stdio.h>
#include <stdlib.h>
int ans[55];
void guess(int cnt)
{
    int num = 0, tmp;
    for (int i = 1; i <= 50; i++)
    {
        scanf("%d", &tmp);
        num += (tmp == i);
    }
    ans[num]++; // 计数
}

int main()
{
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; i++)
        guess(i);

    for(int i=50;i>=0;i--)
    {
        for(int j=0;j<ans[i];j++)
        printf("%d\n",i);
    }

    return 0;
}
```

实矩阵乘法简单版

问题分析

用二维数组模拟矩阵，用线性代数中学到的知识，利用循环模拟下面的公式即可。

需要注意的点是存储答案的矩阵需要先将每一个元素初始化为0。

$$(AB)_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{ip} b_{pj}$$

参考代码

```
#include<stdio.h>
int a[110][110]={0},b[110][110]={0},ans[110][110]={0};
int main()
{
    int i,j,k,l,a1,b1,a2,b2;
    scanf("%d%d%d%d",&a1,&b1,&a2,&b2);
    for(i=1;i<=a1;i++)
    {
        for(j=1;j<=b1;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    for(i=1;i<=a2;i++)
    {
        for(j=1;j<=b2;j++)
        {
            scanf("%d",&b[i][j]);
        }
    }
    for(i=1;i<=a1;i++)
        {
            for(j=1;j<=b2;j++)
            {
                for(k=1;k<=b1;k++)
                {
                    ans[i][j]+=a[i][k]*b[k][j];
                }
            }
        }
    for(i=1;i<=a1;i++)
    {
        for(j=1;j<=b2;j++)
        {
            printf("%d ",ans[i][j]);
```

```
        }
        printf("\n");
    }
    return 0;
}
```

上 α 分位点

方程已经给出，是普通的解方程题目。

二分法：

```
/*
 Author: 张京泽(20356)
 Result: AC Submission_id: 4297252
 Created at: Sun Apr 10 2022 22:59:39 GMT+0800 (China Standard Time)
 Problem: 5405  Time: 2 Memory: 2424
*/
#include<stdio.h>
#include<math.h>

#define eps 1e-13

int main()
{
    double a,b;
    while(scanf("%lf",&a)!=EOF)
    {
        b=0;
        double min=-10,max=10;
        double ans=erf(b/sqrt(2));
        while(fabs(ans-1+2*a)>eps)
        {
            if(ans>1-2*a)
            {
                max=b;
                b=(min+max)/2;
            }
            else if(ans<1-2*a)
            {
                min=b;
                b=(min+max)/2;
            }
            ans=erf(b/sqrt(2));
        }
        printf("%.8lf\n",b);
    }
    return 0;
}
```

牛顿法：

```
#include<stdio.h>
#include<math.h>
```

```
double sqr5=sqrt(0.5);

//正态Phi(x)=0.5+0.5*erf(x/sqrt(2))=1-alpha
//0.5-0.5*erf(x/sqrt(2))-alpha=0
double Normal(double alpha)
{
    if(alpha<0)
    {
        return INFINITY;
    }
    if(alpha>1)
    {
        return -INFINITY;
    }
    if(alpha>0.5)
    {
        return -Normal(1-alpha);
    }
    double x,y;
    for(x=0,y=0.5-alpha;fabs(y)>1e-16;y=0.5*(1-erf(sqr5*x))-alpha)
    {
        x+=y/(sqr5*exp(-x*x/2));
    }
    return x;
}

int main()
{
    double alpha;
    while(scanf("%lf",&alpha)!=EOF)
    {
        printf("%.8lf\n",Normal(alpha));
    }
}
```

Java今天吃什么

题目解析

本题魔改自课件例6-9。

本题首先需要注意的是读入，要正确处理第一行的数字最后的换行符，不要让换行符影响到后面的读入。读入菜谱时，由于事先不知道行数，需要采用HINT中的方法之一进行循环读入，同时统计读入的行数。读入的总行数便是菜谱的总页数。

在存储菜谱的时候也要注意，我们可以使用一个`char`类型的二维数组存放多个字符串，数组的第一维的大小应该大于要存储的字符串的总个数；数组的第二维的大小**应该至少比字符串的最长长度大1**，因为所有字符串都应该以一个“看不见的”`\0`结尾。

在读入完数据之后，即可开始计算Java在下个月 y 号会看哪一页菜谱。计算方法和课件例6-9相似，将当月天数减去今日日期，再加上今日看的菜谱页数的下月的日期，最后整体对菜谱总页数求余。

在输出菜谱的某一页，也就是输出具体的字符串的时候，要按照题目要求，输出一个菜名换一个行。可以采用的一种方法是逐个输出字符串中的字符，当这个字符是`,`的时候，改成输出一个`\n`即可。

示例代码

```
#include <stdio.h>
#define ROW 110
#define LEN 110

int main() {

    int x, n, y;
    scanf("%d %d %d", &x, &n, &y);
    getchar(); // 处理行末换行符
    char s[ROW][LEN]; // 由于之后使用的是fgets读入，会读进换行符\n，因此第二维的长度至少是102，这里开得稍大一些，比较安全
    int t = 0;
    while (fgets(s[t], LEN, stdin) != NULL) {
        t++;
    }
    n = (31 - x + y + n - 1) % t; // 由于菜谱的第一页实际上存储在s[0]中，因此n需要减1
    for (int i = 0; s[n][i] != 0; i++) { // 逐字符输出第n页菜谱，当遇到\0时停止。由于\0的ascii码是0，所以写成0也一样
        if (s[n][i] == ',') { // 遇到,就输出\n
            printf("\n");
        } else { // 否则直接输出原字符
            printf("%c", s[n][i]);
        }
    }

    return 0;
}
```

爱偷懒的球场管理员

题目分析

本题是一个合并区间的问题。先将所有的区间按照左端点从小到大排序，然后依次将能合并的区间合并，使用了简单的贪心思想。
本题需要注意的是数据范围需要使用long long。

示例代码

```
#include<stdio.h>

int main() {
    long long a[1010][2],ans[1010][2];
    int n,idx=0;
    scanf("%d",&n);
    if (n!=0) {
        int i,j;
        for(i=0;i<n;i++) scanf("%lld %lld",&a[i][0],&a[i][1]);


        long long temp;
        for(i=0;i<n-1;i++) {
            int flag = 0;
            for(j=0;j<n-i-1;j++) {
                if (a[j][0]>a[j+1][0]) {
                    flag = 1;
                    temp = a[j][0];
                    a[j][0] = a[j+1][0];
                    a[j+1][0] = temp;
                    temp = a[j][1];
                    a[j][1] = a[j+1][1];
                    a[j+1][1] = temp;
                }
            }
            if(!flag) break;
        }


        for(i=0;i<n;i++) {
            if (idx==0) {
                ans[idx][0] = a[i][0];
                ans[idx][1] = a[i][1];
                idx++;
            } else {
                if (ans[idx-1][1] < a[i][0]) {
                    ans[idx][0] = a[i][0];
                    ans[idx][1] = a[i][1];
```

```
                    idx++;
                } else {
                    if (ans[idx-1][1] < a[i][1]) {
                        ans[idx-1][1] = a[i][1];
                    }
                }
            }
        }

        for(i=0;i<idx;i++) {
            printf("%lld %lld\n",ans[i][0],ans[i][1]);
        }
    }
    return 0;
}
```

方阵数局

题目分析

本题是《直线数据》的升级版，主要考察的是点在二维数组中的“移动”。

本题的易错点仍然是在数组中“走死”的情况，由于0在此题中也有了步数含义，因此“走死”只会发生在几个方格来回绕的情况（已有提示）

示例代码

```
//
// Created by moc85 on 2022/3/31.
//

#include <stdio.h>
#include <math.h>

int matrix[20][20];
int path[20][20];

int main()
{
    int m = 0, n = 0, i = 0, j = 0, arrive = 0, pathLength = 0;
    double r = 0;
    scanf("%d%d", &m, &n);
    for (i = m - 1; i >= 0; i--) {
        for (j = 0; j < n; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    i = 0;
    j = 0;
    while (i >= 0 && i < m && j >= 0 && j < n) {
        if (i == m - 1 && j == n - 1) {
            arrive = 1;
            break;
        }
        if (path[i][j] == 1) {
            break;
        }
        int temp = matrix[i][j];
        path[i][j] = 1;
        if (temp % 2 == 0) {
            if (temp % 3 == 0) {
                j++;
            } else {
```

```
                i++;
            }
        } else {
            if (temp % 3 == 0) {
                i--;
            } else {
                j--;
            }
        }

        pathLength++;

        double tmp = sqrt(i * i + j * j);

        if (i >= 0 && i < m && j >= 0 && j < n) {
            if (tmp > r) {
                r = tmp;
            }
        }
    }

    if (arrive) {
        printf("True\n");
        printf("%d", pathLength);
    } else {
        printf("False\n");
        printf("%.3f", r);
    }

    return 0;
}
```

zhnの数字零

题目分析

如果要使得乘积结尾出现0，那么必然是$2*5$使得结尾出现0。所以每个数对答案的贡献实际上就是这个数中包含的2的个数还有这个数中包含的5的数对答案的贡献。

我们设计如下的状态：$dp[i][j]$表示取出i个数字，这些数字中有j个因子5，最多有多少个2的个数。

可以得出如下的转移方程：$dp[i][j]=max(dp[i][j],dp[i-1][j-cnt5[i]]+cnt2[i])$

示例代码

```
#include <stdio.h>
#include <string.h>
#define ll long long
const int INF = 0x3f3f3f3f;
int n,k;
ll dp[205][205*65];
int cnt2[505],cnt5[505];
int max(int a,int b){
    int temp=a;
    if(a<b) temp=b;
    return temp;
}
int min(int a,int b){
    int temp=a;
    if(a>b) temp=b;
    return temp;
}
int main(){
    scanf("%d%d",&n,&k);
    for(int i=1;i<=n;i++){
        long long x;
        scanf("%lld",&x);
        while(x%2==0){
            cnt2[i]++;
            x/=2;
        }
        while(x%5==0){
            cnt5[i]++;
            x/=5;
        }
    }
    memset(dp,-1,sizeof(dp));
    dp[0][0]=0;
    ll sum=0;
    for(int i=1;i<=n;i++){
        sum+=cnt5[i];
```

```
        for(int j=min(i,k);j>=1;j--){
            for(int kk=sum;kk>=cnt5[i];kk--){
                dp[j][kk]=max(dp[j][kk],dp[j-1][kk-cnt5[i]]+cnt2[i]);
            }
        }
    }
    ll ans=0;
    for(int i=0;i<=sum;i++){
        ans=max(ans,min(1ll*i,dp[k][i]));
    }
    printf("%lld",ans);
    return 0;
}
```

图图玩狼人杀2.0

题目分析

把比较难的问题分步解决，逐个击破，是解决这道题的关键。

观察数据范围，玩家最多 18 人，狼人数最多 6 人，因此基本思想还是枚举；

第一步，从 n 名玩家选出 $wolf$ 头狼，是经典的 n 选 m 组合问题，这一过程使用递归实现，见如下代码：

```
#include <stdio.h>

int n, m, totalStep, ans[10];

void dfs(int step, int now) {
    int i;
    if (step == totalStep) {
        for (i = 0; i < totalStep; i++)
            printf("%d ", ans[i]);
        printf("\n");
        return;
    }
    for (i = now; i <= n; i++) {
        ans[step] = i;
        dfs(step + 1, i + 1);
    }
}

int main() {

    // 18选6，这里的n和m可以自行改动
    n = 18;
    m = 6;

    totalStep = m;
    dfs(0, 1);

    return 0;
}
```

第二步，将选出来的 m 个数假定为狼人（只需稍微改动上面的代码），再检验合理性即可，完整代码如下：

示例代码

```
#include <stdio.h>

int n, m, idTutu, numOfWolf, totalStep, realIdentity[20], say[20], found = 0,
numOfTruth;
```

```c
void dfs(int step, int now);
int check(void);

int main() {

    int i;
    scanf("%d%d%d%d", &n, &m, &idTutu, &numOfWolf);
    totalStep = numOfWolf;
    numOfTruth = n - m;

    for (i = 1; i <= n; i++)
        realIdentity[i] = 1;        // 初始假定每位玩家都是好人

    for (i = 1; i <= n; i++)
        scanf("%d", &say[i]);

    dfs(0, 1);

    if (found == 0)
        printf("Still can't find.\n");

    return 0;
}

void dfs(int step, int now) {

    int i;
    if (step == totalStep) {
        if (check()) {              // check检验合理性
            if (found == 0) {
                found = 1;
                printf("Thank God I know!\n");
            }
            for (i = 1; i <= n; i++)
                if (realIdentity[i] == -1)
                    printf("%d ", i);
            printf("\n");
        }
        return;
    }

    for (i = now; i <= n; i++) {
        if (i == idTutu)        // 图图为好人，故不参与枚举
            continue;
        realIdentity[i] = -1;
        dfs(step + 1, i + 1);
        realIdentity[i] = 1;
    }
```

```
}

int check(void) {

    int i, cnt = 0;
    for (i = 1; i <= n; i++) {
        if (say[i] > 0) {
            cnt += realIdentity[say[i]] == 1;
        } else {
            cnt += realIdentity[-say[i]] == -1;
        }
    }

    if (cnt == numOfTruth) {
        if ((say[idTutu] > 0 && realIdentity[say[idTutu]] == 1)     // 图图说的是真话
            || (say[idTutu] < 0 && realIdentity[-say[idTutu]] == -1)) {
            return 1;
        }
    }

    return 0;
}
```

虚假的签到题

问题分析

本题脱胎于经典的摩尔投票算法

题中内存限制使得程序无法把所有数全存下来再找众数，应考虑如何用更少的变量维护有关众数的信息。

关键突破口在于众数数量大于$\frac{n}{2}$，即出现的次数大于其他所有数字出现次数之和。

因此，如果每次可以将一对答案和非答案的数同时删除，那么最后剩下的数一定是答案。

由于内存限制，我们只能在读入数据的同时“在线”进行一系列操作，而不能等到所有数据全部输入后再进行处理。

所以，我们可以只记录当前认定的“答案”是哪个数，以及到目前为止“答案”出现的次数cnt，下一个读入的数如果是“答案”则$cnt++$，否则$cnt--$。这样，如果最后$cnt>0$，则说明我们认定的“答案”就是本题最终的答案。

那么问题来了，我们要认定哪个数为答案呢？如果$cnt<=0$了怎么办？

对于第一个问题，我们自然而然地想到可以将第一个读进来的数假定为答案，之后再不断更新这个数；

而如果第二个问题所述情况出现了，就需要我们更新这个假定的答案。好消息是，由于本题众数的数量大于其他数的数量，我们在任意成对地删除原数组中不同的数字后，通过对剩下的数组再进行上文的一一对应删除策略，仍然可以找到正确答案。因此，如果出现了$cnt<=0$的情况，只需完全舍弃之前的数组，将新读入的数当成新的假定答案，再重复与之前相同的的操作即可。

参考代码

```
#include<stdio.h>
int n,x,ans,cnt;
int main(){
    scanf("%d",&n);
    while(n--){
        scanf("%d",&x);
        if(x!=ans){
            cnt--;
            if(cnt<=0){
                ans=x;
                cnt=1;
            }
        }
        else cnt++;
    }
    printf("%d",ans);
    return 0;
}
```