士谔2019-第八次上机题解

A. 直角三角形

解题思路

本题考察了多组数据读入和简单逻辑判断。 对于给定组数的多组数据，可以用类似`while(n--)`的语句便利地进行n次循环。 另外注意运算符的优先级，虽然`==`确实比`||`优先级高，但还是推荐将等式打上括号（求稳），尤其是在考试中。 最后，对于字符串输出（尤其是靠后的题），建议直接复制题中文本，以防万一。

AC代码

```
#include<stdio.h>
int main()
{
    int a,b,n,c;
    scanf("%d",&n);
    while(n--){
        scanf("%d%d%d",&a,&b,&c);
        if((a*a+b*b==c*c)||(b*b+c*c==a*a)||(a*a+c*c==b*b)){
            printf("Yes\n");
        }
        else printf("No\n");
    }
    return 0;
}
```

B. 算算交流电

解题思路

本题考察了浮点运算。

将复数的除法计算式化为代码即可。注意如果想获得两个整型变量相除的小数结果，需要对分母或分子进行强制类型转换，或乘以`1.0`，

AC代码

```
#include<stdio.h>
int main()
{
    int a,b,c,d;
    double e,f;
    scanf("%d%d%d%d",&a,&b,&c,&d);
    e = (double)(a*c+b*d) / (c*c+d*d);
    f = (double)(b*c-a*d) / (c*c+d*d);
    printf("%.2f %.2f",e,f);
    return 0;
}
```

C. Corpse别摸鱼了！

解题思路

每次输入后与999比对，如相等则输出上一个输入并停止程序即可。最后输出数据个数。

AC代码

```c
#include <stdio.h>
int main(){
    int find = 999;
    int n = 0, a, pre;
    //读取多组数据
    while(scanf("%d", &a) != EOF){
        if(a == find){
            printf("%d", pre);
            return 0;
        }
        //保存上一次输入
        pre = a;
        n++;
    }
    printf("%d", n);
    return 0;
}
```

D. bnoeq

解题思路

本题需要计算有符号整数二进制表示下1的个数。需要注意如果使用`a = a/2`移位，符号位在最后一次计算时会被忽略掉，即`-1 / 2`结果为0。要特殊考虑负数。

AC代码

```c
#include <stdio.h>
int main(){
    int t = 0;
    scanf("%d", &t);
    while(t--){
        int a, b, i;
        scanf("%d%d", &a, &b);
        int cnta = 0, cntb = 0;
        //注意这里不可以使用 while(a != 0)
        //负数符号右移最终会保持为111...1，即-1
        //或将a的类型设置为unsiged无符号整型
        for(i = 0; i <= 31; i++){
            //取最后一位
            cnta += a & 1;
            a >>= 1;
        }
        for(i = 0; i <= 31; i++){
            cntb += b & 1;
            b >>= 1;
        }
        printf(cnta == cntb ? "branch\n": "in-order\n");
    }

}
```

E. 狭缝与干涉条纹

解题思路

本题是个数学题，可能观察到干涉条纹现象的条件有两个，一个是狭缝的宽度小于等于1，另一个是狭缝和双棱镜的夹角小于等于1度。

对于第一个条件，我们只需要计算$p1$和$p2$之间的距离即可。

对于第二个条件，首先我们根据$p1$和$p3$计算狭缝的斜率，然后使用$atan$函数求狭缝和x轴正向的夹角θ_1。由于$atan$函数的值域是$(-\pi/2, \pi/2)$，为了之后比较大小方便，将$(-\pi/2, 0)$区间映射到$(\pi/2, \pi)$区间，即加π，并将弧度转化为角度（即$*180/\pi$），对于k同理可求得θ_2，最后判断θ_1和θ_2的夹角。

AC代码

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

#define pie 3.1415926535

int main() {
    double x1, y1, x2, y2, x3, y3, x4, y4, k, d;
    double theta1, thetak;
    while (scanf("%lf%lf%lf%lf%lf%lf%lf%lf%lf", &x1, &y1, &x2, &y2, &x3, &y3, &x4, &y4, &k) != EOF) {
        d = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
        if (d > 1) {
            printf("nonoNO\n");
            continue;
        }
        theta1 = atan((y3 - y1) / (x3 - x1)) * 180 / pie;
        theta1 = theta1 < 0 ? theta1 + 180 : theta1;
        thetak = atan(k) * 180 / pie;
        thetak = thetak < 0 ? thetak + 180 : thetak;
        if (fabs(theta1 - thetak) <= 1) {
            printf("| | |\n");
            continue;
        }
        if (theta1 > thetak) {
            printf("Rotate the slit clockwise.\n");
        }
        else {
            printf("Rotate the slit counterclockwise\n");
        }
    }
    return 0;
}
```

F. 找零点

解题思路

本题大体上和之前出过的题类似，核心方法是使用二分法找零点，区别是：本题要考虑$a = 0$的特殊情况，当$a = 0$时，函数是二次函数，又因为题中说只存在一个零点，所以一定是不变号零点，而二分法求零点只能求变号零点，所以本题要特判。

AC代码

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

int a, b, c, d;

double f(double x) {
    return a * x * x * x + b * x * x + c * x + d;
}

int main() {
    scanf("%d%d%d%d", &a, &b, &c, &d);
    double l = -100, r = 100, x;
    if (a == 0 && b != 0)
        printf("%.8f", (double) -c / (2 * b));
    else {
        while (f(l) * f(r) >= 0) {
            l *= 2;
            r *= 2;
        }
        while (1) {
            x = (l + r) / 2;
            if (fabs(f(x)) < 1e-9) break;
            if (f(x) * f(l) > 0 && f(x) * f(r) < 0) {
                l = x;
            }
            else {
                r = x;
            }
        }
        printf("%.8f", (l + r) / 2);
    }
    return 0;
}
```

G. ab+ba

解题思路

本题判断这两个数是否互为逆，首先要求是绝对值位数相同。对于高精度加减的位数比较，可以使用$strlen()$函数，当首空间为负号时，仅比较后面数字部分长度相等，可以用$(strlen(a) - strlen(b)) == 1$，也可以用$strlen(\&a[1]) == strlen(b)$来判断一负一正的两个数长度相同。在长度相同的基础上进行高精度加减运算并判断是否互为逆。在进行高精度减的时候，可以先用$strcmp$函数判断两个数的大小，使用大数减小数计算比较方便。

AC代码

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

char result[1000];

char *sub(char *a, char *b) {
    memset(result, 0, sizeof(result));
    int la = strlen(a), i, r;
    for (i = la - 1; i >= 0; i--) {
        r = a[i] - b[i];
        if (r < 0) {
            a[i - 1] -= 1;
        }
        result[i] = (r + 10) % 10 + '0';
    }
    for (i = 0; i < la; i++) {
        if (result[i] != '0') break;
    }
    return result + i;
}

int main() {
    char sa[1000], sb[1000];
    char *a, *b;
    int signa, signb;
    int i, t, la, lb;
    scanf("%d", &t);
    while (t--) {
        scanf("%s%s", sa, sb);
        a = (sa[0] == '-') ? sa + 1 : sa;
        b = (sb[0] == '-') ? sb + 1 : sb;
        signa = (sa[0] == '-') ? 1 : 0;
        signb = (sb[0] == '-') ? 1 : 0;
        la = strlen(a);
        lb = strlen(b);
```

```
        if (la != lb) printf("illegal operation\n");
        else {
            int flag = 1;
            for (i = 0; i < la; i++) {
                if (a[i] + b[i] - '0' - '0' != 9) {
                    printf("illegal operation\n");
                    flag = 0;
                    break;
                }
            }
            if (flag == 0) {
                continue;
            }
            if (signa == 1 && signb == 1) {
                printf("-");
                for (i = 0; i < la; i++) printf("9");
                printf("\n");
            }
            else if (signa == 0 && signb == 0) {
                for (i = 0; i < la; i++) printf("9");
                printf("\n");
            }
            else if (signa == 0 && signb == 1) {
                if (strcmp(a, b) >= 0)
                    printf("%s\n", sub(a, b));
                else printf("-%s\n", sub(b, a));
            }
            else {
                if (strcmp(b, a) >= 0)
                    printf("%s\n", sub(b, a));
                else printf("-%s\n", sub(a, b));
            }
        }
    }
}
```

H. k关键字排序

解题思路

本题AC代码采用了冒泡排序，时间复杂度$O(kn^2)$。

在读入时，可以令第$k+1$个关键字为物品编号。在进行冒泡排序时，对于每两组数据，可以从前往后遍历k个关键字的值（注意自己加的编号不要参与比较），直到出现两组数据第i个关键字的值不相等，如果前面一组的第i个关键字的值大于后面一组的第i个关键字的值，则这两组数据需要交换，否则两组数据不用交换。如果遍历后发现两组数据完全相同的话也不用交换。

AC代码

```
#include <stdio.h>

int main() {
    int i, j, n, k, l;
    int a[1100][20];
    scanf("%d%d", &n, &k);
    for (i = 0; i < n; i++) {
        for (j = 0; j < k; j++) scanf("%d", &a[i][j]);
        a[i][j] = i + 1;
    }
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            int flag = 0;
            for (l = 0; l < k; l++) {
                if (a[j][l] < a[j + 1][l]) break;
                if (a[j][l] > a[j + 1][l]) {
                    flag = 1;
                    break;
                }
            }
            if (flag == 1) {
                int temp;
                for (l = 0; l <= k; l++) {
                    temp = a[j][l];
                    a[j][l] = a[j + 1][l];
                    a[j + 1][l] = temp;
                }
            }
        }
    }
    for (i = 0; i < n; i++)  printf("%d ", a[i][k]);
    return 0;
}
```

I. 连连看

解题思路1

题目描述的规则与一些连连看游戏相同。如所有位置均不可被消除，则游戏结束。可以认为答案与消除顺序无关。模拟每一次查找与消除过程即可，复杂度为$O((n*m)^3)$。

AC代码1

```
#include <stdio.h>
int mat[15][15];
//标记每次搜索完成的位置，不重复搜索
int done[15][15] = {};
int cnt = 0, n, m;
int inRange(int i, int j){
    return i >= 0 && j >= 0 && i < n && j < m;
}
//返回 0：不可被消除 1：可被消除
//cur：剩余转弯次数，dir：方向，(i,j):坐标，num:起始点数字
int dfs(int num, int i, int j, int dir, int cur){
    if(num == 0 || done[i][j] || cur < 0 || !inRange(i, j)) return 0;
    if(mat[i][j] == num) {
        mat[i][j] = 0;
        cnt+=2;//每次消除2个
        return 1;
    }
    if(mat[i][j] != 0) return 0;
    done[i][j] = 1;
    //递归搜索，注意不可反向走，方向改变时cur-1
    //找到立即返回1，否则继续搜索
    if(dir != 1 && dfs(num, i + 1, j, 0, cur - (dir != 0)))return 1;
    if(dir != 0 && dfs(num, i - 1, j, 1, cur - (dir != 1)))return 1;
    if(dir != 3 && dfs(num, i, j + 1, 2, cur - (dir != 2)))return 1;
    if(dir != 2 && dfs(num, i, j - 1, 3, cur - (dir != 3)))return 1;
    return 0;
}
int main(){
    int i, j;
    scanf("%d %d", &n, &m);

    for (i = 0; i < n; ++i) {
        for (j = 0; j < m; ++j) {
            scanf("%d", &mat[i][j]);
        }
    }
    //每次检查矩阵每一个位置是否可被消除，如在一次检索中所有位置均不可被消除，则结束检索，否则重新检索
```

```
    while (1){
        int flag = 0;
        for (i = 0; i < n; ++i) {
            for (j = 0; j < m; ++j) {
                int cur = 2;
                //搜索前重置
                memset(done, 0, sizeof(done));
                //每个起点向四个方向搜索
                if(dfs(mat[i][j], i + 1, j, 0, cur)) {
                    mat[i][j] = 0, flag = 1;
                    //消除后不再搜索
                    continue;
                }
                memset(done, 0, sizeof(done));
                if(dfs(mat[i][j], i - 1, j, 1, cur)){
                    mat[i][j] = 0, flag = 1;
                    continue;
                }
                memset(done, 0, sizeof(done));
                if(dfs(mat[i][j], i, j + 1, 2, cur)){
                    mat[i][j] = 0, flag = 1;
                    continue;
                }
                memset(done, 0, sizeof(done));
                if(dfs(mat[i][j], i, j - 1, 3, cur)){
                    mat[i][j] = 0, flag = 1;
                    continue;
                }
            }
        }
        if(flag == 0) break;
    }
    printf("%d", cnt);
}
```

解题思路2

如果两个数能被消去，那么需要其中一个数能够走到另一个数的位置，可以看成他们在路径中间的某一个点汇合。

考虑到拐点数不超过2，那么路径走法本质只有两种情况：

- 两个数先走到同一列然后在这一列的某个位置汇合。
- 两个数先走到同一行然后在这一行的某个位置汇合。

每条路径都可以由看成三个部分组成。

举个例子，两个点分别为$(1,3),(3,2)$，如果他们在第2列汇合，走法为：

- (1, 3) $\to$ (2, 3)
- (3, 2) $\to$ (2, 2)
- (2, 2) $\to$ (2, 3)

同学们可以通过画一些图来帮助自己理解。

那么我们的做法就变成了枚举汇合的行或者列，然后检查这条路径上是否全是0。具体细节可以参考代码。

AC代码2

```
#include<stdio.h>
int n, m, ver, mat[20][20], px[110][2], py[110][2];

int checkRow(int row, int l, int r, int v) {
    if (l > r) {
        int t = l; l = r; r = t;
    }
    int i;
    for (i = l; i <= r; ++i)
        if (mat[row][i] != 0 && mat[row][i] != v)
            return 0;
    return 1;
}

int checkLine(int line, int l, int r, int v) {
    if (l > r) {
        int t = l; l = r; r = t;
    }
    int i;
    for (i = l; i <= r; ++i)
        if (mat[i][line] != 0 && mat[i][line] != v)
            return 0;
    return 1;
}
```

```
int find() {
    int i, line, row;
    for (i = 1; i <= 100; ++i)
        if (px[i][0]) {
            int ok = 0;
            for (line = 1; line <= m && !ok; ++line)
                if (checkRow(px[i][0], py[i][0], line, i)
                    && checkRow(px[i][1], py[i][1], line, i)
                    && checkLine(line, px[i][0], px[i][1], i))
                    ok = 1;

            for (row = 1; row <= n && !ok; ++row)
                if (checkLine(py[i][0], px[i][0], row, i)
                    && checkLine(py[i][1], px[i][1], row, i)
                    && checkRow(row, py[i][0], py[i][1], i))
                    ok = 1;

            if (ok) {
                mat[px[i][0]][py[i][0]] = mat[px[i][1]][py[i][1]] = 0;
                px[i][0] = px[i][1] = py[i][0] = py[i][1] = 0;
                return i;
            }
        }
    return -1;
}

int main()
{
    int i, j, x;
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= m; ++j) {
            scanf("%d", &x);
            mat[i][j] = x;
            if (!px[x][0]) {
                px[x][0] = i;
                py[x][0] = j;
            } else {
                px[x][1] = i;
                py[x][1] = j;
            }
        }
    int ans = 0, r;
    while ((r = find()) >= 0) ans += 2;
    printf("%d\n", ans);
    return 0;
}
```

J. 区间

解题思路

本题考察了结构体和贪心+前后缀和的思想。

要将区间分为两个子集，并且使交集长度之和最大，一个符合直觉的“贪心”的想法是：将彼此比较靠近的区间进行合并。如何衡量彼此比较靠近呢？自然就是排序了。

因此一个具体的思路是，将区间以左端点为第一关键字，右端点为第二关键字进行排序。排序后就要枚举一个分割位置`k`了，记排序后数组名是`v`，那么按照这种思路得到的最大答案就是 $\max\limits_{k\in[1,n]}\{\text{len}(\bigcap\limits_{i=1}^{k}v[i])+\text{len}(\bigcap\limits_{i=k+1}^{n}v[i])\}$。我们可以利用前后缀交（此处利用到了用前后缀和快速求区间和的思想），使用单层循环求出上述答案。

上述答案是否就是最终答案？还不一定。如果某种取法是最优的，且此取法没有在上面的枚举过程中考虑到，那么有两种情况：

- 1.这种取法取出的某个集合在`v`中是连续的一段，但不以1开始或以n结束。此时另外一个集合在`v`中就被割为了两段。不失一般性地，记A集合是`v`数组中连续的一段，B集合是被割开的，左半边为`B1`，右半边为`B2`。
  - 若 $B1\bigcap B2=\varnothing$，则答案为 $\text{len}(\bigcap\limits_{i=1}^{\#A}A_i)$。这种情况下**答案可能会更新**。注意到 $\text{len}(\bigcap\limits_{i=1}^{\#A}A_i)<=\min\limits_{x\in A}\{\text{len}(x)\}$，因此实际上只需考虑 $\max\limits_{k\in[1,n]}\{\text{len}(v[i])\}$ 对最终答案的影响。
  - 若 $B1\bigcap B2\neq\varnothing$，我们可以把A和B1合并，或者把A和B2合并，这两者答案的更大者一定不会比原答案更小。这种情况下**答案不会受到影响**。
- 2. 这种取法取出的A、B两个集合在`v`中都不是连续的一段。易证（也是分空集讨论）这种情况下通过交换把其中一集合转化为`v`中连续的一段，一定不会比原答案更小，因此这种情况下**答案也不会受到影响**。

因此最终答案就是 $\max\limits_{k\in[1,n]}\{(\text{len}(\bigcap\limits_{i=1}^{k}v[i])+\text{len}(\bigcap\limits_{i=k+1}^{n}v[i])),\ \text{len}(v[i])\}$。

AC代码

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

```
#define MN 200007
#define MIN(a, b) ((a)<(b)?(a):(b))
#define MAX(a, b) ((a)>(b)?(a):(b))
#define cmpfp_cast(fp) (int(*)(const void *, const void *))(fp)
```

```
typedef struct {int l, r;} Itv;
Itv v[MN], p_sum[MN], s_sum[MN];
```

```c
int cmp(const Itv *p, const Itv *q)
{
    return p->l-q->l ? p->l-q->l : p->r-q->r;
}
int itvlen(const Itv i)
{
    return i.l <= i.r ? i.r-i.l+1 : 0;
}
Itv inter(const Itv i1, const Itv i2)
{
    return (Itv)
    {
        .l = MAX(i1.l, i2.l),
        .r = MIN(i1.r, i2.r)
    };
}
```

```c
int main()
{
    int n, i, j;
    int max_len = 0;

    scanf("%d", &n);
    for (i=1; i<=n; ++i)
    {
        scanf("%d %d", &v[i].l, &v[i].r);
        max_len = MAX(max_len, itvlen(v[i]));
    }

    qsort(v+1, n, sizeof(*v), cmpfp_cast(cmp));
    p_sum[1] = v[1];
    s_sum[n] = v[n];
    for (i=1, j=n; i<n; ++i, --j)
    {
        p_sum[i+1] = inter(p_sum[i], v[i+1]);
        s_sum[j-1] = inter(s_sum[j], v[j-1]);
    }

    for (i=1; i<n; ++i)
        max_len = MAX(max_len, itvlen(p_sum[i]) + itvlen(s_sum[i+1]));

    return printf("%d", max_len), 0;
}
```