2019年厦门大学铃盛杯C语言积分赛决赛题解

> 题解或许会迟到，但是永远不会缺席

A题：铃盛公司的业务

出题人：691

难度评价：**Normal**

解法:

每次扩展业务，图形的边长扩大三倍，设当前的一个字母下标为(X_i,Y_i),那么对应的扩张出的矩形范围X,Y是$[X_i*3,X_i*3+2]$，$[Y_i*3,Y_i*3+2]$,根据业务的类型（R还是C），就可以从上一个时间段的业务推出下一个时间段的业务。因此不断递推下去就可以得到答案。可以用一个二维char数组来存储答案和递推过程中的信息。

std:

```
#include<stdio.h>
#define maxn 730
char map[7][maxn][maxn];
int main(void)
{
    int w=1,h=1,n;
    scanf("%d",&n);
    map[0][0][0]='R';
    for (int i=0;i<n;++i,w*=3,h*=3)
    {
        for (int j=0;j<h;++j)
        {
            for (int k=0;k<w;++k)
            {
                if (map[i][j][k]=='R')
                {
                    map[i+1][j*3][k*3]=map[i+1][j*3][k*3+1]=map[i+1][j*3][k*3+2]=
                    map[i+1][j*3+1][k*3]=map[i+1][j*3+1][k*3+2]=
                    map[i+1][j*3+2][k*3]=map[i+1][j*3+2][k*3+1]=map[i+1][j*3+2]
[k*3+2]='R';
                    map[i+1][j*3+1][k*3+1]='C';
                }
                else
                {
                    map[i+1][j*3][k*3]=map[i+1][j*3][k*3+1]=map[i+1][j*3][k*3+2]=
                    map[i+1][j*3+1][k*3]=map[i+1][j*3+1][k*3+2]=
                    map[i+1][j*3+2][k*3]=map[i+1][j*3+2][k*3+1]=map[i+1][j*3+2]
[k*3+2]='C';
                    map[i+1][j*3+1][k*3+1]='R';
                }
            }
```

```
            }
        }
        for (int i=0;i<w;++i)
            puts(map[n][i]);
        return 0;
    }
```

B. 快速排序

出题人:LZZ

难度评价：**Hard**

解法：

显然，题目要求我们将快速排序的时间复杂度退化到$O(N^2)$。 这个快速排序的问题在于没有三划分。
三划分的含义是将原数组中小于划分点，等于划分点，大于划分点的三段分开，并只对小于和大于划分点的两段做递归。而LZZ的写法没有考虑将等于的部分划分出去（它们混在小于划分点的部分）。
因此，一个基本的思路是构造大量相同的数字。但是注意到有序检查函数的存在。因此我们需要往这些相同的数字里面掺入"沙子"。 一种正确的做法是输出一堆1，然后在倒数第二位输出0作为沙子，倒数第一位仍然输出1。我们来证明这种做法能将快速排序的时间复杂度退化到$O(N^2)$。
首先，由于1占绝对多数，因此每次选取划分点几乎都会选到1。每次选到1。1被交换到最右边进行划分，由于所有的数字都小于等于1，因此在划分时这些数字的位置几乎不会发生改变。最后一步，位于最右边的划分点1与最左边的数字交换。如果最左边的数字是1，那么序列仍然无序。递归问题被分解成一个$O(N-1)$的子问题（待排序区间左端点加1）,且每个函数（不包括递归）的复杂度是$O(N)$。如果最左边的数字是0，那么1被交换到最右边，此时序列有序，下次递归就能结束排序。
因此，只要我们将0放在倒数第二个位置，那么只要0不被抽到，则序列始终无序,平均情况下0会在$\frac{N}{2}$次递归被抽到，因此最后是时间复杂度为$O(N^2)$。
有些同学将0放在中间位置，根据之前的讨论，这样常数仅为之前的$\frac{1}{2}$,通过这道题需要一些运气。（某位同学连交7发才通过） 此外，一些其他的做法也能卡掉这道题。这里不再讨论。

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    puts("30000");
    for (int i=0;i<30000;++i)
        printf("%d ",(i==29998)?0:1);
    return 0;
}
```

C. 世界的尽头

出题人: 691

难度评价：**Hard**

解法：

注意到序列已经有序，我们考虑有序的解决问题。

我们从大到小考虑问题。如果温度最高的目标状态比当前最高的还高，就一定要升高温度，否则如果低，那需要降低温度，我们把多出来的能量存储在一个变量sum里面。 然后依次看第2大的当前状态和第2大的目标状态，第3大的当前状态和第2大的目标状态.....一直到第N大。如果需要降低温度，就存在sum里面。如果需要升高温度，就要从sum里面扣，sum扣完了还不够就需要升高温度。如果相同，显然直接跳过看下一对。
每次升温一定要付出代价，最后如果sum有剩余，也需要付出代价。
题目中说温度相差1不能交换温度。我们来证明上述过程不会出现这种情况。设序列A是源状态，B是目标状态，$A_i <= A_{i+1}, B_i <= B_{i+1}$（根据有序排列的性质），$A_i < B_i$（因为要升温），$A_{i+1} > B_{i+1}$(因为要降温),因此可知$A_{i+1} - A_i >= 2$，又因为对任意$j$有$A_j <= A_{j+1}$，所以以上的交换策略始终成立，不会出现与题设矛盾的情况。

std：

```
#include<bits/stdc++.h>
#define maxn 1000005
int a[maxn],b[maxn];
int main(void)
{
    int t;
    scanf("%d",&t);
    while (t--)
    {
        int n;
        long long sum=0,ans=0;
        scanf("%d",&n);
        for (int i=0;i<n;++i)
            scanf("%d",&a[i]);
        for (int i=0;i<n;++i)
            scanf("%d",&b[i]);
        for (int i=n-1;i>=0;--i)
        {
            int temp=b[i]-a[i];
            if (temp>sum)
            {
                ans+=temp-sum;
                sum=0;
            }
            else
                sum-=temp;
        }
        ans+=sum;
        printf("%lld\n",ans);
    }
    return 0;
}
```

D. 生 成 函 数

出题人：LRL

难度评价：**Easy**

解法：
一道"假题"。
有些同学可能会去写矩阵快速幂。然而实际上这题非常简单。
记得高中有些填空题会给一个非常大的数字（比如年份），让你计算一个有关的函数的值。这种题目显然不能硬算，而应该找规律。我们递推这个式子，发现每隔六项这个数列就会循环。所以只要算出对6的余数就可以计算出答案了。

最后，根据数据范围，记得使用long long 变量。否则会溢出。

std:

```
#include<stdio.h>
typedef long long ll;
inline ll func(const ll n,const ll a,const ll b)
{
    switch (n)
        {
        case 0:return a-b;
        case 1:return a;
        case 2:return b;
        case 3:return b-a;
        case 4:return -a;
        case 5:return -b;
        }
}
int main(void)
{
    int t;
    scanf("%d",&t);
    while (t--)
    {
        ll a,b,n;
        scanf("%lld%lld%lld",&a,&b,&n);
        n%=6;
        printf("%lld\n",func(n,a,b));
    }
    return 0;
}
```

E. 数组循环右移

出题人：LZZ

难度评价：**Very Hard**

解法： 设m为数字长度，n为右移位数，则答案为$m+gcd(n \bmod m,m)$，还要考虑特殊情况，如果$n \bmod m=0$，那么答案为0。
证明：我们注意到，右移以后，一个数字移动到下一个位置，而下一个位置的那个数字又移动到更下面的一个

位置，最后，这个递推移动过程形成了一个环。且每个元素只需要在环上移动一位即可。
我们发现，每个环的长度都是相同的，且环的数量恰为$gcd(n \bmod m, m)$。
对于一个K阶环（环上有K个元素），只需要$K+1$次赋值就可以已将其所有的元素沿环按顺时针或逆时针沿环移动一位。我们用代码演示一下（假设$k=3$）：

```
    long long temp=a[0];
    a[0]=a[1];
    a[1]=a[2];
    a[2]=temp;
```

换个方向：

```
    long long temp=a[2];
    a[2]=a[1];
    a[1]=a[0];
    a[0]=temp;
```

最后，因为一共有$gcd(n \bmod m, m)$个环，且每个环都需要相对元素总数多1次额外移动，因此答案即为$m+gcd(n \bmod m, m)$，最后注意到$n \bmod m = 0$时数组无需移动，特判答案为0。

std:

```
#include <stdio.h>
long long gcd(long long maxo, long long mino)
{
    long long temp;
    if (mino == 0)
        return maxo;
    while (temp = mino, mino = maxo % mino)
        maxo = temp;
    return temp;
}
int main(void)
{
    int t;
    scanf("%d",&t);
    while (t--)
    {
        long long m,n;
        scanf("%lld %lld\n",&m,&n);
        n%=m;
        if (n==0)
            puts("0");
        else printf("%lld\n",m+gcd(n,m));
    }
    return 0;
}
```

F. 就是这么壕

出题人：ZHR

难度评价：**Hard**

> 题意：给一个5×5的灯盘，每次可以选一个位置同时改变其周围八个灯的状态，且有5个位置不可选择。求由全灭转换到目标状态需要的最小步数。

法一

只要想到：

1. 每个灯最多操作一次，因为两次转换等于没动
2. 结果状态与操作的顺序无关

所以，每个位置只有变或不变两种选项，状态总数只有2^{20}

压位爆搜就好啦。$O(2^{20})$ 预处理，$O(1)$查询

显然，现在看到的题是削弱版。

为什么有5个位置不能动呢？还不是为了让你们顺利地爆搜。。因为验题人发现$O(2^{25})$在洛谷要跑16秒，于是连忙改题改数据。

没有剪枝的搜索毕竟不圆满——原题：如果n大一点（$>= 6$），纯爆搜就挂了

一个基本的优化思路：

如果按从上到下、从左到右的顺序枚举开关的话，实际上除第一行和第一列外，每一个位置的开关状态都由左上角决定了。

因为后面的操作都不会再影响到这个位置的灯。如果当前步没有使那个灯就位，那这个状态就不用考虑了。

这样复杂度就变成了$O(n^44^n)$(枚举$2n$个位置，检验需要n^2，其中如果用二维数组则复杂度还会增加n^2)*

标程（剪枝&&带注解）：

```
#include <stdio.h>
#include <string.h>
#define min(x, y) (x) < (y) ? (x) : (y)
int ans;
char input[5][6];
int mp[5][5];

unsigned long encode(int m[5][5]) {
        unsigned long ret = 0;
        int i, j;
        for (i = 0; i < 5; ++i)
                for (j = 0; j < 5; ++j)
                        if (m[i][j])
```

```
                                ret |= (1 <<(i * 5 + j)); //将矩阵转化为对应二进制数
        return ret;
}
void decode(unsigned long st) {
        int i;
        memset(mp, 0, sizeof(mp));
        for (i = 0; i < 25; ++i)
                if ((st >> i) & 1)
                        mp[i / 5][i % 5] = 1; //解码
        return;
}
void dfs(unsigned long st, int nowpos, int nans) { //st记录状态，nowpos表示当前考虑的点，从0到24
        if (st == 0) //到达目标状态
                ans = min(ans, nans);
        else if (nowpos != 25) {
                int l = nowpos / 5, c = nowpos % 5, chk, i; //计算当前操作的行列数
                if (nowpos == 0 || nowpos == 4 || nowpos == 12 || nowpos == 20 || nowpos == 24) { //不可选取的点，直接进下一层
                        dfs(st, nowpos + 1, nans);
                        return;
                }
                decode(st);
                for (i = l - 1; i <= l + 1; ++i)
                        if (i >= 0 && i <= 4) { //如果行未越界
                                if (c)
                                        mp[i][c - 1] ^= 1;
                                if (c < 4)
                                        mp[i][c + 1] ^= 1;
                                if (i != l) //不是中间的点
                                        mp[i][c] ^= 1;
                        }
                if (l && c)
                        chk = mp[l - 1][c - 1];
                if (!l || !c || !chk) //不转换
                        dfs(encode(mp), nowpos + 1, nans + 1);
                if (!l || !c || chk) //转换
                        dfs(st, nowpos + 1, nans);
        }
}
signed main() {
        int T, i, j;
        scanf("%d", &T);
        while (T--) {
                ans = 26;
                for (i = 0; i < 5; ++i)
                        scanf("%s", input[i]);
                for (i = 0; i < 5; ++i)
                        for (j = 0; j < 5; ++j)
                                mp[i][j] = (input[i][j] == '.' ? 1 : 0);
                dfs(encode(mp), 0, 0); //初始为给定状态，目标状态全灭
                printf("%d\n", ans == 26 ? -1 : ans);
        }
        return 0;
```

```
}
```

可以解决n略大于5的规模（<10）

btw:如果n更大怎么办？

法二：

对每个灯泡列一个关于其相邻位置状态的方程组，请自行百度高斯消元法解异或方程组获取$O(n^3)$解法