

Dalian University 3rd qualification trail of ACM school team (2019)

Tutorial

大连大学第三届 ACM 校队选拔赛 题解



主办方：大连大学 ACM 程序设计工作室，大连大学科技协会

题解撰写：大连大学出题组

题解中所有代码均可使用 C99 标准编译运行

2019 年 11 月 23 日

Problem A. Fibonacci sequence I (Tutorial)

题目思路

输出 1 1 2 3 5 即可

参考代码

```
#include <stdio.h>

int main()
{
    printf("1 1 2 3 5");
    return 0;
}
```

Problem B. Fibonacci sequence II (Tutorial)

题目思路

根据题中公式递推即可，注意数据类型应为 *long long int*，否则会溢出。

参考代码

```
#include <stdio.h>
#define maxn 100
#define _for(i,a,b) for(register ll i = (a);i < b;i++)
typedef long long int ll;

ll fi[maxn];

int main()
{
    fi[1] = fi[2] = 1;
    _for(i,3,90)
        fi[i] = fi[i-1] + fi[i-2];

    int ask;
    scanf("%d",&ask);
    printf("%lld",fi[ask]);
    return 0;
}
```

Problem C. GAME (*Tutorial*)

题目思路

因为 RG 战队要在双循环比赛中赢下四场，即六场中要赢下四场，而赢下一场的条件是失去与对方 ATK 值相应的 HP，因为 ATK 值固定，而且 HP 值只增不减，所以只要找 ATK 值最小的两只战队，将他们的 ATK 值相加后乘以 2，若 HP 值大于等于两数之和的两倍，则 RG 战队能够赢得决赛门票，否则不能。

参考代码

```
#include <stdio.h>

int main()
{
    int a[3];
    int R;
    for (int i = 0; i < 3; ++i) {
        scanf("%d", &a[i]);
    }
    scanf("%d", &R);
    //将三数排序
    for (int i = 0; i < 3; ++i) {
        for (int j = i+1; j < 3; ++j) {
            if (a[i]>a[j])
            {
                int t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
        }
    }

    if (R>=(a[0]+a[1])*2)
        printf("NB\n");
    else
        printf("NOOB\n");
    return 0;
}
```

Problem D. Solve equations (*Tutorial*)

题目思路

因为三元一次不等式的系数已经确定，所以通过手动计算后不难得出：

$$X = b - a, \quad Y = \frac{a - (c - a) - (b - a)}{2}, \quad Z = c - a$$

参考代码

```
#include <stdio.h>

int main()
{
    int a,b,c;
    scanf("%d%d%d", &a, &b, &c);
    printf("%d %d %d\n", b-a, (a-(c-a)-(b-a))/2, c-a);
    return 0;
}
```

Problem E. GPA Calculation (Tutorial)

题目思路

此题正确解读计算公式后用循环模拟计算过程即可。

每个大 $\sum_{i=1}^n$ 符号表示 i 从 1 遍历到 n (包括 n)。则我们先从 $A_{k,i}$ 出发，因为题目给的也是每个学生各门课的成绩。

$(\frac{A_{k,i}}{10} - 5) \times C_i$ 其实就是第 k 个学生第 i 门课的**单科绩点**，对每门课的单科绩点进行累加后除以 $\sum_{i=1}^m C_i$ (总学分) 就能得到该学生的**平均绩点**，则将所有学生的**平均绩点**相加后除以班级总人数就是题目所求的**班级平均绩点**。

时间复杂度 $O(nm)$

参考代码

```
#include <stdio.h>
#define _for(i,a,b) for(int i = (a);i < b;i++)
#define maxm 100+3
#define maxn 100000+3
//n个人, m门课
int n,m;
//课程总学分
double csum = 0;
//C[i]表示第i门课的学分
int C[maxm];
//相当于二维数组, A[i].grade[j]表示第i个学生第j门课的成绩
struct st
{
    int grade[maxm];
};
struct st A[maxn];
int main()
{
    scanf("%d%d",&n,&m);
    _for(i,1,m+1)
    {
        scanf("%d",&C[i]);
        csum += C[i];
    }
    _for(i,1,n+1)
    {
        _for(j,1,m+1)
        {
            scanf("%d",&A[i].grade[j]);
            //计算单科绩点
            A[i].grade[j] = A[i].grade[j]<60?50:A[i].grade[j];
        }
    }
    double ans = 0;
    _for(k,1,n+1)
    {
        ans += A[k].grade[m];
    }
    ans /= n;
    printf("%.2f",ans);
}
```

```
{  
    double sum = 0;  
    //计算sum, 即每个学生的平均绩点  
    _for(i,1,m+1)  
        sum += ((A[k].grade[i]/10.0)-5.0)*c[i];  
    sum /= Csum;  
    //ans即为班级总绩点  
    ans += sum;  
}  
//除以n为班级平均绩点, 记得数据类型, 保留两位小数  
printf("%.2f\n",ans/n);  
return 0;  
}
```

Problem F. Move on!Move on! (Tutorial)

题目思路

题目有点长，读完以后我们可以总结出，只有在以下两种情况，食物和汽油储备才有可能增加或减少：

1. 从一个点到另一个点的过程中，食物和汽油储备会减少
2. 当到达一个点的时候，食物和汽油储备会增加

所以我们可以逐个补给点，逐条路去模拟行进过程。在第一个点肯定要初始化 B_{food} 和 B_{petrol} ，注意初始时的值不是 M_{food} 和 M_{petrol} 。然后考虑能否到达第二个点，若不能到达，则直接跳出循环输出 No 即可，如果能到达，就将当前食物和汽油储备减去距离 D ，然后加上下个点 N_{food} 和 N_{petrol} 的值，注意不要超出最大的携带量。若最后能成功到达终点，则输出 Yes 即可。

参考代码

```
#include <stdio.h>
#define _for(i,a,b) for(int i = (a);i < (b);i++)
#define maxn 300000+39
int N,Bf,Bp,Mf,Mp;
int d[maxn];
struct P
{
    int food;
    int petrol;
};
struct P p[maxn];
int main()
{
    scanf("%d%d%d%d%d", &N, &Bf, &Bp, &Mf, &Mp);
    _for(i,1,N+2)
        scanf("%d", &d[i]);

    _for(i,1,N+1)
        scanf("%d%d", &p[i].food, &p[i].petrol);

    int flag = 1;
    _for(i,1,N+2)
    {
        //行进
        Bf -= d[i], Bp -= d[i];
        //判断是否中道崩殂
        if(Bf<0 || Bp<0)
        {
            flag = 0;
            break;
        }
        //补给
        Bf += p[i].food, Bp += p[i].petrol;
        //考虑最大储备量
        if(Bf>Mf)
```

```
Bf = Mf;  
if(Bp>Mp)  
    Bp = Mp;  
//若已到达倒数第二个点且现有储备能够支持她们到达终点，直接退出循环  
if(i==N && Bf-d[i+1]>=0 && Bp-d[i+1]>=0)  
    break;  
}  
if(flag)  
    printf("Yes\n");  
else  
    printf("No\n");  
return 0;  
}
```

Problem G. Find your "MIKU" (Tutorial)

题目思路

本题没有什么好的办法，观察数据范围，猛男就是要大力搜索即可。

具体怎么搜呢？我们将整个图读入一个三维数组以后，枚举每一个元素，若某个元素为字母 M ，则它有可能成为单词 $MIKU$ 的开头，那我们就以它为起点开始往六个方向寻找。考虑所有六个方向，若找到了字母 I ，那我们就能继续找下去……找到什么地方为止呢？当然是找到了字母 U 的时候，这时候我们就可以将结果数 +1 然后返回。因为最多找四个，且递归能很好的执行这种搜索，返回的任务，所以我们考虑使用 dfs （深度优先搜索）来完成。具体细节和实现都写在参考代码里了。

参考代码

```
#include <stdio.h>
#include <string.h>
#define _for(i,a,b) for(int i = (a);i < b;i++)
#define maxn 50
int N;
char m[maxn][maxn][maxn];
int vis[maxn][maxn][maxn];
int rnt = 0;
int dx[] = {0,0,0,0,-1,1};
int dy[] = {0,0,-1,1,0,0};
int dz[] = {-1,1,0,0,0,0};
char toFind[] = {'M','I','K','U'};
//判断是否越界
int valid(int x,int y,int z)
{
    return x>=0&&x<N&&y>=0&&y<N&&z>=0&&z<N;
}
//(x,y,z)为当前点坐标，pos是当前正在搜寻toFind[pos]的字母
void dfs(int x,int y,int z,int pos)
{
    //全都找到了，结果数加一，返回
    if(pos==4)
    {
        rnt++;
        return ;
    }
    //考虑所有6种情况
    _for(i,0,6)
    {
        //(nx,ny,nz)为尝试的新坐标
        int nx = x+dx[i];
        int ny = y+dy[i];
        int nz = z+dz[i];
        //如果新坐标坐标位置没有越界，且没有被访问过，且就是我们正在搜寻的字母
        if(valid(nx,ny,nz) && !vis[nx][ny][nz] && toFind[pos]==m[nx][ny][nz])
        {
            vis[nx][ny][nz] = 1;
            dfs(nx,ny,nz,pos+1);
            vis[nx][ny][nz] = 0;
        }
    }
}
```

```
//标记已访问
vis[nx][ny][nz] = 1;
//深搜递归进行访问
dfs(nx,ny,nz, pos+1);
vis[nx][ny][nz] = 0;
}
}

int main()
{
    //vis数组清零
    memset(vis,0,sizeof(vis));
    scanf("%d",&N);
    _for(i,0,N)
        _for(j,0,N)
            _for(k,0,N)
                scanf(" %c",&m[i][j][k]);

    //逐个遍历正方体中元素并假设以该元素开头进行深搜
    _for(i,0,N)
        _for(j,0,N)
            _for(k,0,N)
                if(m[i][j][k]=='M')
                    dfs(i,j,k,1);
    printf("%d",rnt);
    return 0;
}
```

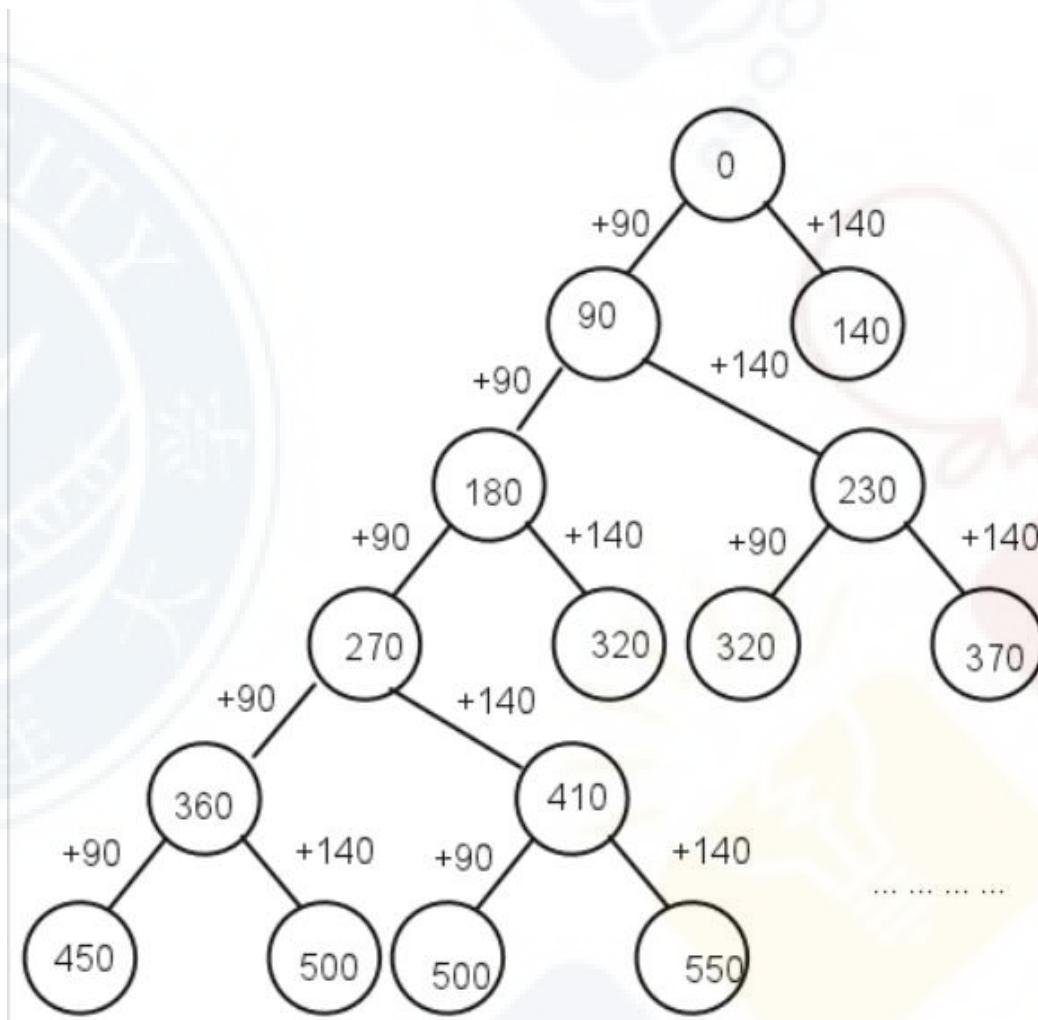
Problem H. Running On The South Playground (Tutorial)

题目思路

观察操场平面图和跑操路线，我们不难得出一条结论：

在任意一点，下一步都可以选择走 140m 或 90m，且不管选择哪个，到达下一个点后依然可以进行 140m 和 90m 的选择。

因此可以到达的距离为 90m, 140m, 90m + 90m, 90m + 140m, 140m + 140m.....可以到达的距离形成了一棵二叉树，图示如下：



如果我们想求出这棵二叉树上所有节点，其实就是求出了所有可到达的距离，然后我们根据可达距离回答即可。但是求出这棵树的时间复杂度为 $O(2^{\frac{n}{90}})$ ，对于最大的数据，需要执行 $T_n \approx 2^{50000}$ 次运算，显然不可能。

我们观察这棵二叉树，可以发现其中有些节点是重复的，比如 320, 500，实际上这些节点我们只需要求解一次即可。我们可以考虑遍历所有米数，若当前米数为 x ，则若 $x - 90$ 或者 $x - 140$ 可以到达，那 x 不就也可以到达吗？这里运用了动态规划的思想，将米数看为状态进行转移，而不用穷举所有节点。

这样我们就可以预处理一个数组 $dp[]$, $dp[x]$ 若等于 1 则说明 可到达, 等于 0 则说明不可到达。对于每个询问, 我们期望在 $O(1)$ 时间复杂度下回答询问, 我们现在只能做到 $O(1)$ 查询某个米数 x 是否能直接到达, 怎么办? 我们还需要处理一个 $ans[]$ 数组, 让 $ans[x]$ 表示对于某个要求米数 x , 大于等于 x 的最小可到达米数。所以我们从后往前遍历一遍处理过的 $dp[]$ 数组, 若能到达 y_2 , 则在从后往前遇到下一个 y_1 ($y_1 < y_2$) 前, 所有 $y_1 < x \leq y_2$ 的要求数, 都需要跑 y_2 才能合格, 因此最终 $ans[]$ 数组就处理完了。对于输入 x 答案就是 $ans[x]$ 。

注意 要求为 0 的时候需要输出 0 而不是 90。

接下来我们就可以在 $O(1)$ 时间复杂度下处理询问, 总时间复杂度为 $O(n + T)$, 其中 n 为最大米数, T 为询问次数。

参考代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define _for(i,a,b) for(int i = (a);i < b;i++)
#define _rep(i,a,b) for(int i = (a);i > b;i--)
#define maxn 500000+3
int dp[maxn];
//答案数组
int ans[maxn];
int T;
//预处理
void preDeal()
{
    memset(dp,0,sizeof(dp));
    memset(ans,0,sizeof(ans));
    //初始化, 90m和140m可达
    dp[90] = dp[140] = 1;
    //dp[x]表示 x 可达
    _for(i,141,maxn)
    if(dp[i-90] || dp[i-140])
        dp[i] = 1;

    int cur = maxn;
    //从后往前遍历dp数组, 处理ans数组
    _rep(i,maxn,0)
    {
        if(dp[i])
            cur = i;
        ans[i] = cur;
    }
}
int main()
{
    preDeal();
    scanf("%d",&T);
    while(T--)
    {
        int t;
        scanf("%d",&t);
        printf("%d\n",ans[t]);
    }
}
```

```
return 0;  
}
```

Problem I. 39 CREATOR (Tutorial)

题目思路

本题作为正式赛压轴题，还是有一定难度的。我们首先观察混合规则，如果一直加入一种药，比如一直加入蓝药，则瓶中的药剂状态为 $0g \rightarrow 39g$ 蓝药 $\rightarrow 39g$ 红药 $\rightarrow 0g$ 。怎么样？瓶中的药又回到了初始状态！于是我们可以联想到溢出以及取余运算。我们用 -39 到 39 作为药瓶状态， -39 为 $39g$ 红药，而 39 为 $39g$ 蓝药，然后我们将这段区间加上 39 ，就变成了用 0 到 78 表示药瓶状态，这样我们就可以用数组下标表示药瓶的状态了。

然后我们考虑倒药剂的过程，用取余来模拟这一过程，根据之前的分析我们不难发现对于状态 $x (0 \leq x \leq 78)$ ，若加入药剂会形成 $x + a (-39 \leq a \leq 39)$ 的状态，然后我们将此状态 $((x + a) + 79) \% 79$ 后就成了合法状态。什么意思呢？举两个例子：

1. 瓶子里已经有 $39g$ 蓝药了，再往里倒 $3g$ 蓝药，则 $x = 39$ ， $a = 3$ ，所以 $((x + a) + 79) \% 79 = 2$ ， 2 的含义我们之前已经提到过，将 $2 - 39 = -37$ 即瓶中现在有 $37g$ 红药。
2. 瓶子里已经有 $38g$ 红药了，再往里倒 $5g$ 红药，则 $x = 1$ ， $a = -5$ ，所以 $((x + a) + 79) \% 79 = 75$ ，将 $75 - 39 = 36$ 即瓶中现在有 $36g$ 蓝药。

搞清楚这个问题以后，我们尝试进行 dfs （深度优先搜索），选取方案无非就是左左右右的组合，有没有感觉和 *Problem H. Running On The South Playground* 的每次选 90 或者 140 很像？但是很明显，如果枚举所有左右情况，就像在问题 *H* 里枚举 90 和 140 一样，时间复杂度是 2 的指数次幂，一定会超时，不过我们还是将程序放出，方便大家对 dfs 有一个更加清晰的认识：

参考程序 (DFS解法，会超时)

```
#include <stdio.h>
#define maxn 500+3
#define maxm 78+3
#define MOD 1000000007
#define _for(i,a,b) for(int i = (a);i < b;i++)
#define _rep(i,a,b) for(int i = (a);i > b;i--)

int N;
int a[maxn];
int rnt = 0;

//l,e,ri代表目前所剩瓶子的左右区间，now为目前瓶子装药情况
void dfs(int le,int ri,int now)
{
    //凑出来39g蓝药，结果就加一
    if(now==78)
        rnt ++,rnt %= MOD;
    //就剩一个瓶子了
    if(le==ri)
    {
        //试试这个瓶子倒进来后能不能凑出来39g蓝药，能的话结果就加一
        if(now+a[le]==78)
            rnt ++,rnt %= MOD;
    }
}
```

```

    }
    //对当前所有的瓶子考虑取左端或者右端
    dfs(l[e+1], r[i], (now+a[l[e]+79])%79);
    dfs(l[e], r[i-1], (now+a[r[i]+79])%79);
}
int main()
{
    scanf("%d", &N);
    _for(i, 1, N+1)
        scanf("%d", &a[i]);

    dfs(1, N, 39);
    printf("%d\n", rnt);
    return 0;
}

```

再进行观察思考，我们发现对于“还剩下的区间” $[l[e], r[i]]$ ，若当前瓶中药剂 now 已经是 39g 蓝药， dfs 会将结果数 +1，这就是程序低效的原因。如果我们对于还剩下的区间 $[l[e], r[i]]$ ，能将从更大区间转移过来的方案数都加上，那效率将会大大提升。

$dp[l[e]][r[i]][K]$ 代表 $[l[e], r[i]]$ 为现在剩余的一排瓶子，目前瓶子装了 K 状态的药的方案数。显然，对于区间 $[l[e], r[i]]$ ，只可能从 $[l[e] - 1, r[i]]$ 和 $[l[e], r[i] + 1]$ 转移而来，因为只能从两端取。那我们不难想到 $\{l[e], r[i], (K + a[l[e] - 1] + 79)\}$ 这个状态的方案数可以由 $\{l[e] - 1, r[i], K\}$ 这个状态转移而来，同理， $\{l[e], r[i], (K + a[r[i] + 1] + 79)\}$ 可以由 $\{l[e], r[i] + 1, K\}$ 这个状态转移而来。

转移方程如下：

```

dp[l[e]][r[i]][(k+a[l[e]-1]+79)%79] += dp[l[e]-1][r[i]][k];
dp[l[e]][r[i]][(k+a[r[i]+1]+79)%79] += dp[l[e]][r[i]+1][k];

```

初始状态呢？对于满区间，即没有取的时候，那自然什么方案都没有。对于 $[1, n - 1]$ 和 $[2, n]$ 这两种区间，很明显方案数为 1，且要根据 $a[N]$ 和 $a[1]$ 的情况去赋值，于是我们有初始化方案：

```

dp[1][N-1][a[N]+39] = dp[2][N][a[1]+39] = 1;

```

同时我们需要注意，对于每一段区间的 39g 蓝药 状态，我们都应该加入 ans ，因为可以不取满。

最后的最后，因为 dp 数组没法模拟取完的场景，所以对于 $dp[i][i][K]$ 其实是只剩下 $a[i]$ 这一个瓶子没取时候的情况，所以如果 $a[i]$ 里面装的是蓝药，那就只可能由 $dp[i][i][78 - a[i]]$ 转移而来，也就是本来已经是蓝药了，加上这瓶药刚好凑够 39g 蓝药。如果 $a[i]$ 这个瓶子里是红药，则只可能本来瓶子里装的就是红药，加上这瓶药以后负得正恰巧变成 39g 蓝药了，因此从 $dp[i][i][-a[i] - 1]$ 转移而来。

最后不要忘记取余。

时间复杂度 $O(78 * n^2) = O(n^2)$ 。

虽然程序不长，但是思维量并不小。一个算法竞赛选手需要具备很多素质，但最重要的特质永远是独立思考，这是程序设计竞赛的灵魂，能够看到这里都是好样的，给看到这里的你点个赞，也希望这次比赛能够激发你 coding 的热情，培养你独立思考的能力。

最后预祝大家能够学业进步，期末加油！

参考程序

```
#include <stdio.h>
#define maxn 500+3
#define maxm 78+3
#define MOD 1000000007
#define _for(i,a,b) for(int i = (a);i < b;i++)
#define _rep(i,a,b) for(int i = (a);i > b;i--)

int N;
int a[maxn];
int dp[maxn][maxn][maxm];
int main()
{
    scanf("%d",&N);
    _for(i,1,N+1)
        scanf("%d",&a[i]);

    if(N==1)
    {
        if(a[1]==39)
            printf("1\n");
        else
            printf("0\n");
        return 0;
    }

    int ans = 0;
    dp[1][N-1][a[N]+39] = dp[2][N][a[1]+39] = 1;
    _rep(len,N-2,0)
        _for(le,1,N-len+2)
        {
            int ri = le+len-1;
            _for(k,0,79)
            {
                dp[le][ri][(k+a[le-1]+79)%79] += dp[le-1][ri][k];
                dp[le][ri][(k+a[ri+1]+79)%79] += dp[le][ri+1][k];
                dp[le][ri][(k+a[le-1]+79)%79] %= MOD;
                dp[le][ri][(k+a[ri+1]+79)%79] %= MOD;
            }
            ans += dp[le][ri][78];
            ans %= MOD;
        }

    _for(i,1,N+1)
    {
        if(a[i]>0)
            ans += dp[i][i][78-a[i]];
        else
            ans += dp[i][i][-a[i]-1];
        ans %= MOD;
    }
    printf("%d\n",ans);
    return 0;
}
```

