NOIP2013解题报告

——长郡中学　黄鑫

1.circle

题目描述：

n 个小伙伴（编号从 0 到 n-1）围坐一圈玩游戏。按照顺时针方向给 n 个位置编号，从 0 到 n-1。最初，第 0 号小伙伴在第 0 号位置，第 1 号小伙伴在第 1 号位置，……，依此类推。

游戏规则如下：每一轮第 0 号位置上的小伙伴顺时针走到第 m 号位置，第 1 号位置小伙伴走到第 m+1 号位置，……，依此类推，第 n－m 号位置上的小伙伴走到第 0 号位置，第 n-m+1 号位置上的小伙伴走到第 1 号位置，……，第 n-1 号位置上的小伙伴顺时针走到第 m-1 号位置。

现在，一共进行了 10^k 轮，请问 x 号小伙伴最后走到了第几号位置。

题目分析：

当前在 a 号位置的小伙伴，进行一轮游戏（即顺时针走 m 个位置）后，所处的位置显然为(a+m)mod n 设为 a'，进行两轮游戏后，则为(a'+m)mod n 即(a+2*m)mod n。那么题目所求即(x+m*10^k)mod n，我们可以先求出(m*10^k)mod n，再求出答案。那么如何快速求出(m*10^k)mod n 呢？可以用快速幂在 O(log k)的时间内得到解。

Code:

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
using namespace std;
#define ll long long

int main(){
    freopen("circle.in","r",stdin);
    freopen("circle.out","w",stdout);
    ll n, m, k, x, base = 10;
    scanf("%lld%lld%lld%lld", &n, &m, &k, &x);
    ll ans = m;
    for (; k > 0; k >>= 1, base = base * base % n)
        if (k % 2) ans = ans * base % n;
    ans = (ans + x) % n;
    printf("%lld\n",ans);
    return 0;
}
```

2.match

题目描述：

涵涵有两盒火柴，每盒装有 n 根火柴，每根火柴都有一个高度。 现在将每盒中的火柴各自排成一列， 同一列火柴的高度互不相同， 两列火柴之间的距离定义为：$\sum_{i=1}^{n}(ai-bi)^2$，其中 ai 表示第一列火柴中第 i 个火柴的高度，bi 表示第二列火柴中第 i 个火柴的高度。

每列火柴中相邻两根火柴的位置都可以交换，请你通过交换使得两列火柴之间的距离最小。请问得到这个最小的距离，最少需要交换多少次？如果这个数字太大，请输出这个最小交换次数对 99,999,997 取模的结果。

题目分析：

距离最小是什么情况呢？很容易想到是将两列火柴排序后一一对应，证明：假设距离最小的排列中存在 ai<aj 而 bi>bj，则它们贡献的距离为(ai-bi)^2+(aj-bj)^2，但交换 bi 和 bj 之后，贡献的距离变为(ai-bj)^2+(aj-bi)^2，后式减前式，得2(ai-aj)(bi-bj)<0，即后式的值小于前式，与假设矛盾。那么由于同一列火柴的高度互不相同，每根火柴对应的那根火柴就确定了。

接下来，就是解决最小交换次数了。我们令 ci 表示第一列的第 i 根火柴对应的那根火柴在第二列中的初始位置，则若 i<j 且 ci>cj,那么就必然有一次交换 ci 和 cj 两根火柴的过程，那么我们只需求出所有这样的数对（即逆序对），可以证明这就是最小交换次数。但这样还不够，因为 n 的范围有10w，所以可以用数据结构（最好是树状数组）。

事实上，交换就相当于冒泡排序的过程，我们可以模拟一遍，这也是最小交换次数。从左到右扫一遍 a，若当前 ai 所对应的的那根火柴在 di 位置，则从 di 一路走到 i 位置，路径上的火柴位置都+1，于是我们可以用线段树区间修改 lazy tag 的方法，也可以做到 O(nlog n)，在考场上我就是这么做的。

Code:

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cstring>
using namespace std;
#define ll long long

const int maxn = 100000 + 100, mo = 99999997;

struct node{
    int d,w;
}a[maxn], b[maxn];

int tr[maxn] = {0}, n;

bool cmp(node a, node b){
    return a.d < b.d;
}

int ask(int x){
    int tmp = 0;
    for (; x > 0; x -= x & (-x)) tmp += tr[x];
    return tmp;
}

void add(int x){
    for (; x <= n; x += x & (-x)) tr[x]++;
}

int main(){
    freopen("match.in","r",stdin);
    freopen("match.out","w",stdout);
    scanf("%d",&n);
    for (int i = 1; i <= n; i++) scanf("%d",&a[i].d), a[i].w = i;
```

```
    for (int i = 1; i <= n; i++) scanf("%d",&b[i].d), b[i].w = i;
    sort(a+1, a+n+1, cmp); sort(b+1, b+n+1, cmp);
    int c[maxn], ans = 0;
    for (int i = 1; i <= n; i++) c[a[i].w] = b[i].w;
    for (int i = n; i; i--) ans = (ans + ask(c[i])) % mo, add(c[i]);
    printf("%d\n",ans);
    return 0;
}
```

以上是求逆序对的方法，另附第二种方法的代码链接（考场上的源代码，pascal 版的）：
http://hi.baidu.com/immortal_hx/item/1e63c72b32fcf9999c63d159

3.truck

题目描述：

A 国有 n 座城市，编号从 1 到 n，城市之间有 m 条双向道路。每一条道路对车辆都有重量限制，简称限重。现在有 q 辆货车在运输货物， 司机们想知道每辆车在不超过车辆限重的情况下，最多能运多重的货物。

题目分析：

可以证明任意可以互相到达的两点之间所有边权最小值最大的路径中必有一条在这两个点所在连通图的最大生成树上，根据 prim 算法的过程很容易想到这点。换个说法，就是一个连通图的最大生成树上的路径，必然是路径两端点的边权最小值最大的路径。求最大生成树可以用 kruskal 算法。求完最大生成树之后，就只要对每次询问求出两点之间的树上路径的边权最小值了。

树上路径的边权最小值怎么求呢？每次 dfs 或 bfs？根据数据范围肯定会超时。我们必须想一个快捷的方法。联想到最近公共祖先（lca），可以用类似的方法求解。即 f[i,j]记录 i 结点向上走2^j 步后所到达的祖先，g[i,j]记录 i 结点向上走2^j 步的路途中的边权最小值。首先，对于询问的两个点 a、b，将它们移动到相同的深度，然后同时向上走，直到走到它们的最近公共祖先，用一个变量记录走的过程中的边权最小值即可。

Code：

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cstring>
using namespace std;
#define ll long long

const int maxm = 50000 + 10, maxn = 10000 + 10, inf = 100001;
int n, m, fa[maxn], edge[2*maxn], lev[maxn] ;
int next[2*maxn], len[2*maxn], head[maxn] , f[maxn][14], g[maxn][14];
struct node{
    int u, v, w;
}e[maxm];

bool cmp(node a, node b){
    return a.w > b.w;
}
```

```
int find(int x){
	if (fa[x] != x) fa[x] = find(fa[x]);
	return fa[x];
}

int min(int a, int b){
	return (a < b) ? a : b;
}

void dfs(int x, int level){
	lev[x] = level;
	for (int i = 1; i <= 13; i++){
		f[x][i] = f[f[x][i-1]][i-1];
		g[x][i] = min(g[x][i-1], g[f[x][i-1]][i-1]);
	}
	for (int i = head[x]; i; i = next[i])
		if (!lev[edge[i]]){
			f[edge[i]][0] = x;
			g[edge[i]][0] = len[i];
			dfs(edge[i], level+1);
		}
}

int lca(int a, int b){
	if (lev[a] > lev[b]) swap(a, b);
	int ans = inf;
	for (int i = 13; i+1; i--)
		if (lev[f[b][i]] >= lev[a]){
			ans = min(ans, g[b][i]);
			b = f[b][i];
		}
	if (a == b) return ans;
	for (int i = 13; i+1; i--)
		if (f[a][i] != f[b][i]){
			ans = min(ans, min(g[a][i], g[b][i]));
			a = f[a][i]; b = f[b][i];
		}
	ans = min(ans, min(g[a][0], g[b][0]));
	return ans;
}

int main(){
	freopen("truck.in","r",stdin);
	freopen("truck.out","w",stdout);

	scanf("%d%d", &n, &m);
	for (int i = 1; i <= m; i++) scanf("%d%d%d", &e[i].u, &e[i].v, &e[i].w);

	sort(e+1, e+m+1, cmp);
	for (int i = 1; i <= n; i++) fa[i] = i;
	int tot = 0;
```

```
    for (int i = 1; i <= m; i++){
        int x = find(e[i].u), y = find(e[i].v);
        if (x != y){
            edge[++tot] = e[i].v; len[tot] = e[i].w; next[tot] = head[e[i].u]; head[e[i].u] = tot;
            edge[++tot] = e[i].u; len[tot] = e[i].w; next[tot] = head[e[i].v]; head[e[i].v] = tot;
            fa[x] = y;
        }
    }

    for (int i = 1; i <= n; i++)
        if (!lev[i]) dfs(i, 1);

    int q, x, y;
    scanf("%d",&q);
    for (int i = 1; i <= q; i++){
        scanf("%d%d", &x, &y);
        if (find(x) != find(y)) printf("-1\n");
        else printf("%d\n",lca(x, y));
    }
    return 0;
}
```

4.block

题目描述：

春春幼儿园举办了一年一度的“积木大赛”。今年比赛的内容是搭建一座宽度为o的大厦，大厦可以看成由o块宽度为1的积木组成，第i块积木的最终高度需要是 hi 。

在搭建开始之前，没有任何积木（可以看成o块高度为 0 的积木）。接下来每次操作，小朋友们可以选择一段连续区间$[M, R]$，然后将第M块到第R块之间（含第 L 块和第 R 块）所有积木的高度分别增加1。

小N是个聪明的小朋友，她很快想出了建造大厦的最佳策略，使得建造所需的操作次数最少。但她不是一个勤于动手的孩子，所以想请你帮忙实现这个策略，并求出最少的操作次数。

题目分析：

首先，将[l,r]变成右边是开区间的[l,r)，那么不可能有一个点同时包含区间左端点和右端点（如果同时含有，合并两个区间显然更优），那么我们只需确定每个点上是左端点还是右端点，或者都没有（当这个点的高度和左右两个点的高度相同时），如果 h(i)>h(i-1)，则这个点显然要新加 h(i)-h(i-1)个左端点，反之，则当前点存在 h(i-1)-h(i)个右端点，统计总共加入了多少个左端点即可。

Code：

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cstring>
using namespace std;
#define ll long long
```

```
int main(){
    freopen("block.in","r",stdin);
    freopen("block.out","w",stdout);
    int n, ans = 0, x, now = 0;
    scanf("%d",&n);
    for (int i = 1; i <= n; i++){
        scanf("%d",&x);
        if (x > now) ans += x - now;
        now = x;
    }
    printf("%d\n",ans);
    return 0;
}
```

5.flower

题目描述：

花匠栋栋种了一排花，每株花都有自己的高度。花儿越长越大，也越来越挤。栋栋决定把这排中的一部分花移走，将剩下的留在原地，使得剩下的花能有空间长大，同时，栋栋希望剩下的花排列得比较别致。

具体而言，栋栋的花的高度可以看成一列整数 $h1, h2, \cdots, hn$。设当一部分花被移走后，剩下的花的高度依次为$g1, g2, \cdots, gn$，则栋栋希望下面两个条件中至少有一个满足：

条件 A：对于所有的$1 \leqslant i \leqslant m2$，有$g2i > g2i-1$， 同时对于所有的 $1 \leqslant i < m2$，有$g2i > g2i+1$；

条件 B：对于所有的$1 \leqslant i \leqslant m2$，有$g2i < g2i-1$，同 时对于所有的$1 \leqslant i < m2$，有$g2i < g2i+1$。

注意上面两个条件在$n = 1$时同时满足，当$n > 1$时最多有一个能满足。

请问，栋栋最多能将多少株花留在原地。

题目分析：

根据两个条件，最后剩下的花必然是波浪形的，那么我们只要根据输入序列的波浪形就可以判断最多能留下多少株，即找出拐点的个数（包括第一个和最后一个）。

另一个方法是 DP，f[i]表示以 i 为最后一株，且 i 是波浪的上端的最大值，g[i]则为以 i 为波浪的下端的最大值，f[i]=max(g[j])+1，满足 j<i 且 hj<hi，g[i]的转移类似。但由于数据较大，需要用线段树优化。

Code：

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cstring>
using namespace std;
#define ll long long
const int maxn = 100000+100;

int main(){
    freopen("flower.in","r",stdin);
    freopen("flower.out","w",stdout);
```

```
    int n, a[maxn], next[maxn], last[maxn];
    memset(last, 0, sizeof(last));
    scanf("%d",&n);
    for (int i = 1; i <= n; i++){
        scanf("%d",&a[i]);
        last[i] = i-1;
        next[i-1] = i;
        if (a[i] == a[i-1]) last[i] = last[i-1];
        next[last[i]] = i;
    }
    int ans = 2, s = next[1], t = last[n];
    while (a[s] == a[1]) s = next[s];
    while (a[t] == a[n]) t = last[t];
    for (int i = s; i <= t; i++)
        if ((a[i] > a[last[i]] && a[i] > a[next[i]])||
                (a[i] < a[last[i]] && a[i] < a[next[i]])) ans++;
    printf("%d\n",ans);
    return 0;
}
```

上面的方法是求拐点的，DP 的代码链接（同样也是 pascal 的考场源代码）：
http://hi.baidu.com/immortal_hx/item/029bb21f90ea7818b98a1a10

6.puzzle

题目描述：

小 B 最近迷上了华容道，可是他总是要花很长的时间才能完成一次。于是，他想到用编程来完成华容道：给定一种局面， 华容道是否根本就无法完成，如果能完成， 最少需要多少时间。

小 B 玩的华容道与经典的华容道游戏略有不同，游戏规则是这样的：

1. 在一个 n*m 棋盘上有 n*m 个格子，其中有且只有一个格子是空白的，其余 n*m-1 个格子上每个格子上有一个棋子，每个棋子的大小都是 1*1 的；
2. 有些棋子是固定的，有些棋子则是可以移动的；
3. 任何与空白的格子相邻（有公共的边）的格子上的棋子都可以移动到空白格子上。

游戏的目的是把某个指定位置可以活动的棋子移动到目标位置。

给定一个棋盘，游戏可以玩 q 次，当然，每次棋盘上固定的格子是不会变的， 但是棋盘上空白的格子的初始位置、 指定的可移动的棋子的初始位置和目标位置却可能不同。第 i 次玩的时候， 空白的格子在第 EXi 行第 EYi 列，指定的可移动棋子的初始位置为第 SX i 行第 SYi 列，目标位置为第 TXi 行第 TYi 列。

假设小 B 每秒钟能进行一次移动棋子的操作，而其他操作的时间都可以忽略不计。请你告诉小 B 每一次游戏所需要的最少时间，或者告诉他不可能完成游戏。

题目分析：

初看这道题，很容易让人想到广搜，但仔细观察数据范围，就会发现广搜是过不了的。为什么呢？因为广搜搜索了很多无用状态和重复状态。这里的重复状态不是平常所说的可通过判重剪枝的重复状态，而是每次询问都重复搜索的重复状态。为什么这么说呢，让我们分析一下。

显而易见的是，要让指定棋子移动朝一个方向移动，必须先将空格移动到指定棋子的那

个方向，然后棋子才能迈出步伐。而普通的广搜中，每次都要搜索一遍将空格移动到指定位置。并且，只有空格在指定棋子的隔壁的状态才是有用的，所以可以先预处理出棋子在[i,j]位置时，空格在 k 方向，要将棋子往 h 方向移动一格的最小步数，记为 Move[i][j][k][h]，然后用 A[i][j][k]表示一个状态，那么 Move[i][j][k][h]即为 A[i][j][k]状态到 A[i'][j'][h']的步数，其中 i',j'为朝 h 方向移动后指定棋子的坐标，h'为 h 的反方向（想一想，为什么？），那么我们就相当于从 A[i][j][k]到 A[i'][j'][h']连了一条边权为 Move[i][j][k][h]的边，每次询问时，先将空格移动到指定棋子的隔壁，然后再通过各种状态之间的转移（用 SPFA 实现），求出到目标位置的最小移动次数。

Code：

```
#include<cstdio>
#include<cstdlib>
#include<algorithm>
#include<cstring>
using namespace std;

const int maxn = 40 + 2, inf = 1000000, maxl = 10000;
const int D[4][2] = {{1,0},{-1,0},{0,1},{0,-1}};

int n, m, ask, V = 0, l, r, aimx, aimy, inx = 0, e = 0;
int Map[maxn][maxn], A[maxn][maxn][4], next[maxl*10], len[maxl*10];
int old[maxn][maxn], qu[maxl], inq[maxl], dist[maxl], head[maxl], edge[maxl*10];
struct node{
    int x, y;
}q[maxn*maxn];

void init(){
    freopen("puzzle.in","r",stdin);
    freopen("puzzle.out","w",stdout);
    scanf("%d%d%d", &n, &m, &ask);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++){
            scanf("%d",&Map[i][j]);
            for (int k = 0; k < 4; k++) A[i][j][k] = ++V;
        }
}

bool bfs(int ll, int rr){
    for (int i = ll; i <= rr; i++)
        for (int k = 0; k < 4; k++)
            if (Map[q[i].x+D[k][0]][q[i].y+D[k][1]]){
                q[++r].x = q[i].x+D[k][0];
                q[r].y = q[i].y+D[k][1];
                if (old[q[r].x][q[r].y] == inx) r--; else old[q[r].x][q[r].y] = inx;
                if (q[r].x == aimx && q[r].y == aimy) return 1;
            }
    l = rr;
    return  0;
}

int find(int sx, int sy, int tx, int ty){
```

```
    if (sx == tx && sy == ty) return 0;
    int step = 1;
    l = 0; r = 1; inx++;
    q[1].x = sx; q[1].y = sy;
    aimx = tx; aimy = ty;
    bool bb;
    while  (l < r && (bb = !bfs(l+1,r))) step++;
    if (bb)    return inf; else return step;
}

void link(int s, int t, int l){
    edge[++e] = t; next[e] = head[s]; len[e] = l; head[s] = e;
}

void prepare(){
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (Map[i][j])
                for (int k = 0; k < 4; k++)
                    if (Map[i+D[k][0]][j+D[k][1]]){
                        link(A[i][j][k],A[i+D[k][0]][j+D[k][1]][k^1], 1);
                        for (int h = 0; h < 4; h++)
                            if (k != h && Map[i+D[h][0]][j+D[h][1]]){
                                Map[i][j] = 0;
                                int move = find(i+D[k][0],j+D[k][1],i+D[h][0],j+D[h][1])+1;
                                if (move < inf)
                                    link(A[i][j][k], A[i+D[h][0]][j+D[h][1]][h^1], move);
                                Map[i][j] = 1;
                            }                       }
}

int spfa(int s, int t){
    inx++; l = 0; r = 1; qu[1] = s;
    memset(dist, 127, sizeof(dist));
    dist[s] = 0;
    while (l != r){
        l = (l % maxl) + 1;
        for (int i = head[qu[l]]; i; i = next[i])
            if (dist[qu[l]]+len[i] <= dist[edge[i]]){
                dist[edge[i]] = dist[qu[l]] + len[i];
                if (inq[edge[i]] != inx){
                    inq[edge[i]] = inx;r = (r % maxl) + 1;qu[r] = edge[i];
                }
            }
        inq[qu[l]] = 0;
    }
    return dist[t] < inf ? dist[t] : -1;
}

void work(){
    int kx,ky,sx,sy,tx,ty;
    for (int i = 1; i <= ask; i++){
```

```
        scanf("%d%d%d%d%d%d",&kx,&ky,&sx,&sy,&tx,&ty);
        if (sx == tx && sy ==ty){
            printf("0\n"); continue;
        }
        if (!Map[sx][sy]||!Map[tx][ty]){
            printf("-1\n"); continue;
        }
        int S = ++V, T = ++V;
        Map[sx][sy] = 0;
        for (int k = 0; k < 4; k++)
            if (Map[sx+D[k][0]][sy+D[k][1]]){
                int d = find(kx,ky,sx+D[k][0],sy+D[k][1]);
                if (d < inf) link(S,A[sx][sy][k],d);
            }
        Map[sx][sy] = 1;
        for (int k = 0; k < 4; k++)
            if (Map[tx+D[k][0]][ty+D[k][1]]) link(A[tx][ty][k],T,0);
        printf("%d\n",spfa(S,T));
    }
}

int main(){
    init();
    prepare();
    work();
}
```