1.成绩

算法分析

据说这题当年官方评测时也闹出过乌龙。小数会产生异变，比如以下代码：

$$\text{int} \quad a = 80 * 0.2$$

赋值号右侧是实数，最后结果可能会为16.000001或15.999999，赋值给a的话，a的值可能是16或15，产生错误。解决方法：化乘为除（A,B,C 都是 10的整数倍），或定义为double，最后结果保留0位输出。

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#define ll long long
using namespace std;
int main()
{
	int a, b, c;
	cin>>a>>b>>c;
	int ans = a * 20 / 100  + b * 30 / 100 + c * 50 / 100;
	cout<<ans<<endl;
	return 0;
}
```

2.图书管理员

算法分析

n和q是小于1000的，直接枚举就可以。需要预处理10的次方。

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#define ll long long
using namespace std;
int x[10], sbook[1010];
int main()
{
	x[0] = 1;
	for (int i = 1; i <= 8; ++i) x[i] = x[i-1] * 10;
	int n, q;
	scanf("%d%d", &n, &q);
	for (int i = 1; i <= n; ++i) scanf("%d", &sbook[i]);
	sort(sbook + 1, sbook + n + 1);
	int len, num;
	for (int i = 1; i <= q; ++i)
	{
		scanf("%d%d", &len, &num);
		int ans = -1;
		for (int j = 1; j <= n; ++j)
		{
			if (sbook[j] % x[len] == num)
			{
				ans = sbook[j];
				break;
			}
		}
		printf("%d\n", ans);
	}
	return 0;
}
```

3.棋盘

算法分析

一个很容易想到的思路是：将无颜色的点拆点，然后转图论求最短路。红色用1，黄色用2，无色用0。坐标(i, j)对应的点编号为$i * (m - 1) + j$，无色的点比如点(x, y)是无色的，拆点后编号$x * (m - 1) + y$的点是红色，$x * (m - 1) + y + m * m$的点是黄色。相反地，给定一个点的编号，也可以求出它的对应的坐标。有几个细节：

1.变色是一种花费，走过去是否要花费还要看两者的颜色是否相同，颜色不相同的话，也得要有花费。

2.无色的点，只能被变色，变色后可以连接其相邻的红色或黄色的点，不能连接无色的了。因此，对应变色的无色的点，可以用vis数组标记下。

3.建图过程。逐个枚举点u，v是和其相连的点：

u是红色，v是红色，则边权为0；v是黄色，则边权为1；v是无色，对v拆点，u到v边权是0，u到$v + m * m$边权为1，标记无色的点v。

u是黄色，v是红色，则边权为1；v是黄色，则边权为0；v是无色，对v拆点，u到v边权是1，u到$v + m * m$边权为0，标记无色的点v。

u是无色，并且该点被标记过，说明该点被变色过，可以向四周扩展。v是红色，则u到v边权为0，$u + m * m$到v边权为1；v是黄色，则u到v边权为1，$u + m * m$到v边权为0。

4.最短路过程中，如果访问到一个点是无色的，则最短路的代价要加2。以下用的是dij+堆优化。

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <queue>
#include <utility>
#define ll long long
using namespace std;
int a[110][110], m, n, vis[110][110];
int dx[4] = {-1, 0, 1, 0},
        dy[4] = {0, 1, 0, -1};
struct node
{
        int next, to, val;
}edg[100010];
int h[20010], cnt, v[20010];
int dis[20010], pre[20010];
priority_queue< pair<int, int> > q;
void sadd(int u, int v, int val)
{
        ++cnt;
        edg[cnt].next = h[u];
        edg[cnt].to = v;
        edg[cnt].val = val;
        h[u] = cnt;
}
void dij()
{
        memset(dis, 0x3f, sizeof(dis));
        q.push(make_pair(0, 1));
        dis[1] = 0;
        pre[1] = 0;
        while (q.size())
        {
                int u = q.top().second; q.pop();
                if (v[u]) continue;
                v[u] = 1;
                for (int i = h[u]; i; i = edg[i].next)
                {
                        int t = edg[i].to, w;
                        if (v[t]) continue; // 如果t是白点，则跳过

```

```
                        if (t > m * m) w = t - m * m;
                        else w = t;
                        int x, y;
                        if (w % m == 0)
                        {
                                x = w / m; y = m;
                        }else
                        {
                                x = w / m + 1; y = w - w / m * m;
                        }
                        if (a[x][y] == 0) // 无颜色的点
                        {
                                if (dis[t] > dis[u] + edg[i].val + 2 )  //
                                {
                                        dis[t] = dis[u] + edg[i].val + 2;
                                        q.push(make_pair(-dis[t], t));
                                }
                        }else
                        {
                                if (dis[t] > dis[u] + edg[i].val)
                                {
                                        dis[t] = dis[u] + edg[i].val ;
                                        q.push(make_pair(-dis[t], t));
                                }
                        }
                }
        }
}
int main()
{
        scanf("%d%d", &m, &n);
        int x, y, c;
        for (int i = 1; i <= n; ++i)
        {
                scanf("%d%d%d", &x, &y, &c);
                a[x][y] = ++c; // 1:红色  2：黄色  0:无色
        }
        for (int i = 1; i <= m; ++i)
                for (int j = 1; j <= m; ++j)
                {
                        for (int k = 0; k < 4; ++k)
                        {
                                x = i + dx[k], y = j + dy[k];
                                if (x < 1 || x > m || y < 1 || y > m) continue;
                                if (a[i][j] == 1)
                                {
                                        if (a[x][y] == 1) sadd(m * (i - 1) + j, m * (x - 1) + y, 0);
                                        else if (a[x][y] == 2) sadd(m * (i - 1) + j, m * (x - 1) + y, 1);
                                        else
                                        {
                                                sadd(m * (i - 1) + j, m * (x - 1) + y, 0); //
                                                sadd(m * (i - 1) + j, m * (x - 1) + y + m * m, 1);  //
                                                vis[x][y] = 1; // 该点被拆过
                                        }
                                }else if (a[i][j] == 2)
                                {
                                        if (a[x][y] == 1) sadd(m * (i - 1) + j, m * (x - 1) + y, 1);
                                        else if (a[x][y] == 2) sadd(m * (i - 1) + j, m * (x - 1) + y, 0);
                                        else
                                        {
                                                sadd(m * (i - 1) + j, m * (x - 1) + y, 1); //
                                                sadd(m * (i - 1) + j, m * (x - 1) + y + m * m, 0);  //
                                                vis[x][y] = 1; // 该点被拆过
                                        }
                                }else if (vis[i][j]) // 该点被拆过
                                {
                                        if (a[x][y] == 1)
                                        {
```

```
                                        sadd(m * (i - 1) + j, m * (x - 1) + y, 0);
                                        sadd(m * (i - 1) + j + m * m, m * (x - 1) + y, 1);
                                }
                                else if (a[x][y] == 2)
                                {
                                        sadd(m * (i - 1) + j, m * (x - 1) + y, 1);
                                        sadd(m * (i - 1) + j + m * m, m * (x - 1) + y, 0);
                                }
                        }
                }
        dij();
        if (a[m][m] == 0)
        {
                if (dis[m * m] == 0x3f3f3f3f && dis[m * m * 2] == 0x3f3f3f3f) printf("-1\n");
                else printf("%d\n", min(dis[m * m], dis[m * m * 2]));
        }else
        {
                if (dis[m * m] == 0x3f3f3f3f ) printf("-1\n");
                else printf("%d\n", dis[m * m]);
        }
        return 0;
}
```

算法拓展

1.bfs求最短路。以上方法建图之后，边权为0或1，用bfs+双端队列也可以求出最短路，时间效率更高。

2.dfs+剪枝。因为可以向四个方向移动，所以记忆化有后效性。可以考虑dfs。$dfs(int\ x, int\ y, int\ val, int\ c)$：已经搜索到$(x, y)$处，最短路为$val$，$(x, y)$的颜色为c。无色的点不能向无色的点走。设全局变量minval记录最值。

一个思维上的优化：如果u是有颜色的，v是无色的，v变色后和u的颜色一样即可，效果不会更差。

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#define ll long long
using namespace std;
int a[110][110], m, n, vis[110][110];
int f[110][110];
int dx[4] = {-1, 0, 1, 0},
        dy[4] = {0, 1, 0, -1};
// f[x][y]:(x, y)到(m, m)的最小花费
int minval = 1e8;
void dfs(int x, int y, int val, int c)
{
        if (val > minval) return;
        if (x == m && y == m)
        {
                minval = min(minval, val);
                return;
        }
        for (int k = 0; k < 4; ++k)
        {
                int sx = x + dx[k], sy = y + dy[k];
                if (sx < 1 || sx > m || sy < 1 || sy > m) continue;
                if (vis[sx][sy]) continue;
                if (a[x][y] == 0 && a[sx][sy] == 0) continue;
                vis[sx][sy] = 1;
                if (a[x][y])
                {
                        if (a[sx][sy])
                        {
                                if (f[sx][sy] > f[x][y] + (a[x][y] == a[sx][sy] ? 0 : 1))
                                {
```

```
                                        f[sx][sy] = f[x][y] + (a[x][y] == a[sx][sy] ? 0 : 1);
                                        dfs(sx, sy, f[sx][sy], a[sx][sy]);
                                }
                        }else
                        {
                                if (f[sx][sy] > f[x][y] + 2)
                                {
                                        f[sx][sy] = f[x][y] + 2;
                                        dfs(sx, sy, f[sx][sy], a[x][y]);
                                }
                        }
                }else
                {
                        if (f[sx][sy] > f[x][y] + (c == a[sx][sy] ? 0 : 1))
                        {
                                f[sx][sy] = f[x][y] + (c == a[sx][sy] ? 0 : 1);
                                dfs(sx, sy, f[sx][sy], a[sx][sy]);
                        }

                }
                vis[sx][sy] = 0;
        }
}
int main()
{
        scanf("%d%d", &m, &n);
        int x, y, c;
        for (int i = 1; i <= n; ++i)
        {
                scanf("%d%d%d", &x, &y, &c);
                a[x][y] = ++c; // 1:红色  2：黄色  0:无色
        }
        memset(f, 0x3f, sizeof(f));
        vis[1][1] = 1;
        f[1][1] = 0;
        dfs(1, 1, 0, a[1][1]);
        if (minval != 1e8) printf("%d\n", minval); else printf("-1\n");
        return 0;
}
```

4.跳房子

算法分析

对于花费的金币g，可以计算出机器人每次弹跳的距离[s, t]。弹跳是在一条线上进行的。下面就好想了。dp。设f[i]：跳到位置i时的最大分数。则：

$$f[i] = \max\{f[j] + i的分数\}$$

j的取值范围是：[i − t, i − s]

要在决策j中取最大值的f[j]，而且决策j有一定的区间范围，用单调队列优化dp。

需要注意的一点：不是所有的位置都能被跳到，只有被更新过的位置才能作为下一个的决策。代码中用ssav数组存待考察的决策，对于其中的决策，如果满足条件，则入队列。

如果枚举的g不满足，则枚举更大的g。容易证明，枚举g，答案具有单调性。可以二分枚举。整体时间复杂度O(nlogn)。

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
#include <queue>
#define ll long long
using namespace std;
```

```cpp
struct node
{
	int dis, val;
}a[500010];
ll f[500010];
int ssav[500010], sl, sr, n, d, k;
int q[500010];
ll chk(int g)
{
	memset(f, 0, sizeof(f));
	memset(q, 0, sizeof(q));
	int l, r, s, t;
	if (g < d)
	{
		s = d - g; t = d + g;
	}else
	{
		s = 1; t = d + g;
	}
	// 每次跳的距离区间[s, t]
	l = r = 1;
	q[1] = 0;
	f[0] = 0;
	sl = 1; sr = 0;
	int i;
	for (i = 1; i <= n; ++i)
	{
		for (int k = sl; k <= sr; ++k)
		{
			if (a[ ssav[k] ].dis <= a[i].dis - s )
			{
				++sl;
				while (l <= r && f[ssav[k]] >= f[q[r]]) --r;
				q[++r] = ssav[k];
			}else break;
		}
		while (l <= r && a[ q[l] ].dis  < a[i].dis - t) ++l;
		if (l <= r && (a[ q[l] ].dis >= a[i].dis - t && a[ q[l] ].dis <= a[i].dis - s))
		{
			ssav[++sr] = i;
			f[i] = f[q[l]] + a[i].val;
		}
	}
	ll ans = -1e12;
	for (int i = 1; i <= n; ++i) ans = max(ans, f[i]);
	return ans >= k;
}
int main()
{
	scanf("%d%d%d", &n, &d, &k);
	for (int i = 1; i <= n; ++i) scanf("%d%d", &a[i].dis, &a[i].val);
	int l = 0, r = 500000 + 1;
	while (l < r)
	{
		int g = (l + r) >> 1;
		if (chk(g)) r = g; else l = g + 1;
	}
	if (l == 500000 + 1) printf("-1\n"); else printf("%d\n", l);
	return 0;
}
```