题目描述

Acesrc得到了n个二次函数，其中第i个二次函数是$f_i(x) = a_i x^2 + b_i x + c_i$的形式。而且第$i$个二次函数中$x$的取值应当是在区间$[l_i, r_i]$中的一个整数。我们令第$i$个函数中$x$的取值为$x_i$。

在以上的条件之外，Acesrc还受到了m个限制,每个限制都是形如$x_u \le x_v + d$的形式，其中u和v是不同函数的编号，d是一个整数。

Acesrc想要知道，在满足所有条件的情况下，能够得到的所有二次函数之和的最大值是多少。也就是说，Acesrc想要知道在合理选择每个x_i的情况下，所能得到的$\sum_{i=1}^{n} f_i(x_i)$的最大值。

输入格式

第一行包含两个正整数n和m，分别表示二次函数的个数以及额外限制条件的个数。

接下来n行，每行包含三个整数a_i, b_i, c_i,表示第i个二次函数的系数。

再接下来n行，每行包含两个整数l_i, r_i，表示第i个二次函数自变量的取值范围。

最后m行，每行包含三个数u_i, v_i, d_i，描述第i个额外限制。

输入保证存在至少一个满足所有限制的选择x_i的方式。

输出格式

输出一行，包含一个整数，表示Acesrc在合理选择每个x_i的情况下，所能得到的$\sum_{i=1}^{n} f_i(x_i)$的最大值。

样例

样例输入1

```
3 3
0 1 0
0 1 1
0 1 2
0 3
1 2
-100 100
1 2 0
2 3 0
3 1 0
```

样例输出1

```
9
```

样例输入2

```
5 8
1 -8 20
2 -4 0
-1 10 -10
0 1 0
0 -1 1
```

```
1 9
1 4
0 10
3 11
7 9
2 1 3
1 2 3
2 3 3
3 2 3
3 4 3
4 3 3
4 5 3
5 4 3
```

样例输出2

```
46
```

样例解释

在第一组样例中，$f_1(x) = x, f_2(x) = x + 1, f_3(x) = x + 2$,限制条件是$x_1 \le x_2, x_2 \le x_3, x_3 \le x_1$，所以有$x_1 = x_2 = x_3$。最优应选择$x_1 = x_2 = x_3 = 2$,此时$\sum_{i=1}^{n} f_i(x_i) = 9$

数据范围与提示

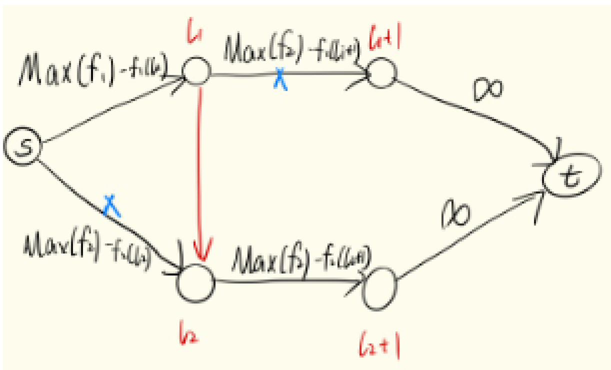
对于所有数据有

$1 \le n \le 50, 0 \le m \le 100, |a_i| \le 10, |b_i|, |c_i| \le 1000,$

$-100 \le l_i \le r_i \le 100, 1 \le u_i, v_i \le n, u_i \ne v_i, |d_i| \le 200$

对于前50%的数据，满足$\prod_{i=1}^{n}(r_i - l_i + 1) \le 2 \cdot 10^6$。

对于前100%的数据，没有附加限制。



构造如图所示的模型，其中每个函数对应一条从s到t的路径，路径上有$r_i - l_i + 1$个点$l_i, l_{i+1}, \cdots, r_i$，每条边$(u, v)$的容量是$f_{max} - f(v)$，t的入边容量为无穷大。

每个限制体现为一条不同路径之间的，权值为无穷大的边。

寻找图中的最小割，显然每个函数对应的路径上有且仅有一条边被割去，对应该函数最优解下的函数值。因为建图方式，边权代表当前方案相较于无任何限制下的最大值的损失，因此最小割对应最小损失，即最大值。限制就体现在'Z'字形的两臂上必须至少有一条边被割去，否则'Z'就成了s到t的一条路径。如图中打蓝叉的两条边就不能同时割去，对应了$x_u \le x_v + b$的限制。

注意要加上一个偏移量避免负数对最大流算法的影响。

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef unsigned long long ull;
typedef long long ll;
#define maxn 55
#define maxm 105
inline ll read()
{
    ll x = 0, k = 1;
    char c = getchar();
    while (c < '0' || c > '9') {
        if (c == '-')
            k = -1;
        c = getchar();
    }
    while (c >= '0' && c <= '9') {
        x = (x << 3) + (x << 1) + c - '0';
        c = getchar();
    }
    return x * k;
}

/*最大流模板---------------------------------*/
/*add(x,y,v,w):添加权值为v的边(x,y)和权值为w的边(y,x)*/
/* s和t分别是源和漏*/
#define inf 1e12
#define N 1000000
int s, t, la[N], et = 1, d[N], cur[N], q[N], L, R;
struct E {
    int to;
    ll flow;
    int nxt;
} e[N];
void add(int x, int y, ll v, ll w)
{
    e[++et] = (E){ y, v, la[x] }, la[x] = et;
    e[++et] = (E){ x, w, la[y] }, la[y] = et;
}
int bfs()
{
    memset(d, 0, sizeof(d));
    for (q[L = R = 0] = t, d[t] = 1; L <= R; L++)
        for (int i = la[q[L]]; i; i = e[i].nxt)
            if (e[i ^ 1].flow && !d[e[i].to])
                d[q[++R] = e[i].to] = d[q[L]] + 1;
    return d[s];
}
ll dfs(int x, ll ret)
{
    if (x == t || ret == 0)
        return ret;
    ll tmp, flow = 0;
```

```cpp
    for (int& i = cur[x]; i; i = e[i].nxt)
        if (d[x] == d[e[i].to] + 1) {
            tmp = dfs(e[i].to, e[i].flow < ret - flow ? e[i].flow : ret - flow);
            e[i].flow -= tmp, e[i ^ 1].flow += tmp, flow += tmp;
            if (ret == flow)
                return flow;
        }
    return flow;
}
ll maxflow()
{
    ll flow = 0;
    while (bfs())
        memcpy(cur, la, sizeof(la)), flow += dfs(s, inf);
    return flow;
}
/*--------------------------------------*/


int m,n,u,v,dd;
int a[maxn], b[maxn], c[maxn], l[maxn], r[maxn];
int head[maxn]; //第i个函数中的第一个可取点的编号
ll MAX[maxn]={-N};
ll ans = 0.0;

int find_former(int i, int n) //寻找当前函数中点的上一个点的编号,n为函数号,i为自变量值
{
    if(i==l[n])
        return 1;
    else
        return head[n] + i - l[n] - 1;
}
int main(void)
{
    const ll BIG = 200000; //负数偏移量
    s = 1, t = 2;
    head[0] = 2, r[0] = l[0] = 0;
    n = read();
    m = read();
    for (int i = 1;i<=n;i++)
        a[i] = read(), b[i] = read(), c[i] = read();
    for (int i = 1; i <= n; i++) {
        l[i] = read(), r[i] = read();
        head[i] = head[i - 1] + 1 + (r[i - 1] - l[i - 1]);
        for (int j = l[i]; j <= r[i];j++){
            ll tmp = a[i] * j * j + b[i] * j + c[i];
            MAX[i] = (MAX[i] > tmp) ? MAX[i] : tmp;
        }
        ans += MAX[i];
        add(s, head[i], (ll)(MAX[i]-(a[i] * l[i] * l[i] + b[i] * l[i] + c[i])) + BIG, 0);
        for (int j = l[i]; j <= r[i] - 1; j++)
             add(head[i] + j - l[i], head[i] + j - l[i] + 1, (ll)(MAX[i]-(a[i] * (j + 1) * (j + 1) + b[i] * (j + 1) + c[i])) + BIG, 0);
        add(head[i] + (r[i] - l[i]), t, inf, 0);
    }
```

```
    for (int i = 1;i<=m;i++){
        u = read(), v = read(), dd = read();
        for (int j = l[u]; j <= r[u];j++){
            int k = l[v];
            while(k!=r[v] && !(k>=j-dd)) ++k;
            if(k>=j-dd)
                add(find_former(j, u), find_former(k, v), inf, 0);
            else //函数v中所有点都不满足限制，因此这条边要连在v路径的最后，表示函数u自变量取当前值时，函数v无论怎么取值都不满足条件。
                add(find_former(j, u), head[v] + r[v] - l[v], inf, 0);
        }
    }
    cout << ans - (maxflow() - n * BIG);
    system("pause");
    return 0;
}
```