Problem D. Knapsack and Queries

Input file:	*standard input*
Output file:	*standard output*
Time limit:	10 seconds
Memory limit:	1024 mebibytes

First, you are given the positive integer MOD.

You have a knapsack, that is empty at first.

You have to perform Q queries.

- In each query, you have first to perform either **ADD** or `REMOVE` operation, and then perform `FIND`.
- For **ADD** operation, you are given positive integers w and v. You put the cookie with weight w and value v to the knapsack.
- For **REMOVE** operation, you take out the cookie with smallest weight from the knapsack and eat it.
- After each **ADD** or **REMOVE** you perform **FIND** operation: given positive integers l and r, you answer the follow question: can you choose cookies from that knapsack so that the $l \le (X \bmod MOD) \le r$ (X is sum of the weight of the selected cookies)?. If you can't, output -1. Otherwise output the maximum sum of the value of the selected cookies.

Input

Input is given in the following format:

MOD

Q

$t'_1\ w'_1\ v'_1\ l'_1\ r'_1$

$t'_2\ w'_2\ v'_2\ l'_2\ r'_2$

...

$t'_Q\ w'_Q\ v'_Q\ l'_Q\ r'_Q$

Constraints:

$0 \le t'_i, w'_i, v'_i, l'_i, r'_1 \le 2^{30} - 1$, $2 \le MOD \le 500$, $1 \le Q \le 100\,000$.

Queries are encrypted as described later. You can get t_i, w_i, v_i, l_i, r_i by decryption $t'_i, w'_i, v'_i, l'_i, r'_i$.

You may assume that $1 \le w_i, v_i \le 10^9$, $0 \le l_i \le r_i \le MOD - 1$, $t_i = 1$ for **ADD+FIND** query, $t_i = 2$ for **REMOVE+FIND** query (and in this case $w_i = v_i = 0$), that the cookie, which is given by **ADD**, is *heavier* than any cookies which added by the previous **ADD**, and that when executing **REMOVE**, the knapsack isn't empty.

We prepared the decryption code with C++11 (or later), Java, D, C#. Use class `Crypto` for decryption. The code of class `Crypto` and examples of its usage can be uploaded from `http://opentrains.mipt.ru/~ejudge/crypto.zip` for C++11 (or later), Java, D, C#.

Here is the example for C++:

```
#include <cstdint> //uint8_t, uint32_t

class Crypto {
public:
    Crypto() {
```

```
        sm = cnt = 0;
        seed();
    }

    int decode(int z) {
        z ^= next();
        z ^= (next() << 8);
        z ^= (next() << 16);
        z ^= (next() << 22);
        return z;
    }

    void query(long long z) {
        const long long B = 425481007;
        const long long MD = 1000000007;
        cnt++;
        sm = ((sm * B % MD + z) % MD + MD) % MD;
        seed();
    }
private:
    long long sm;
    int cnt;

    uint8_t data[256];
    int I, J;

    void swap_data(int i, int j) {
        uint8_t tmp = data[i];
        data[i] = data[j];
        data[j] = tmp;
    }

    void seed() {
        uint8_t key[8];
        for (int i = 0; i < 4; i++) {
            key[i] = (sm >> (i * 8));
        }
        for (int i = 0; i < 4; i++) {
            key[i+4] = (cnt >> (i * 8));
        }

        for (int i = 0; i < 256; i++) {
            data[i] = i;
        }
        I = J = 0;

        int j = 0;
        for (int i = 0; i < 256; i++) {
            j = (j + data[i] + key[i%8]) % 256;
            swap_data(i, j);
        }
    }
```

```
    uint8_t next() {
        I = (I+1) % 256;
        J = (J + data[I]) % 256;
        swap_data(I, J);
        return data[(data[I] + data[J]) % 256];
    }
};
```

The decryption process works in the next way:

- First, you make the instance of `class Crypto`.
- For each query first call the `decode` function in order of t', w', v', l', r'. The return values are t, w, v, l, r. Then perform the query and call the `query` function with the result of the **FIND**.

The sample C++ code:

```
#include <cstdio>
#include <cstdlib>
#include <cstdint> //uint8_t, uint32_t

class Crypto {
    ...
};

int main() {
    int MOD, Q;
    scanf("%d %d", &MOD, &Q);
    Crypto c;
    for (int i = 0; i < Q; i++) {
        int t, w, v, l, r;
        scanf("%d %d %d %d %d", &t, &w, &v, &l, &r);
        t = c.decode(t);
        w = c.decode(w);
        v = c.decode(v);
        l = c.decode(l);
        r = c.decode(r);
        if (t == 1) {
            (add candy(w, v))
        } else {
            (delete candy)
        }
        long long ans = (answer for query(l, r));
        c.query(ans);
        printf("%lld\n", ans);
    }
}
```

Note that class `Crypto` consume time about 200 to process $Q = 100\,000$.

Output

For each query print the result of **FIND** operation.

Examples

standard input	standard output
10 7 281614559 249378726 433981056 466775634 683612866 727071329 787572584 591471796 328464426 757737734 279580343 240336097 538846427 808491898 224313807 222498984 42804452 371605808 667115067 791865961 68683864 1045549765 515479514 1067782238 349547144 907343711 381772625 149003422 879314974 953881571 883899098 700164610 414212891 752949213 972845634	10 0 -1 21 -1 11 111

standard input	standard output
7 20 281614559 249378726 433981094 466775639 683612870 59536386 999828879 241246766 434670565 174365647 172060134 848462699 857413429 182122460 807914643 808426426 600772095 829463884 974102196 354283529 370037909 1024921880 664216868 194331103 140834169 917331875 242953442 205247688 335469789 1055568137 823475244 641321246 617915164 160300810 1073617378 892669150 939175632 904628449 606339993 1059849410 829170894 436718235 288920513 228195002 55212938 772189413 373108543 94133155 610930061 513937768 986619331 175674265 812546186 865335970 605634588 880196843 1071068047 723408215 587598264 380801783 393196081 141080294 584230885 135343295 661927186 5740819 967233824 22597607 888639499 467454437 365679801 515258603 989059385 962028117 761163096 357270919 737051059 569528959 935653628 70506031 869282414 947492121 280522456 96822010 856514221 155948699 826430734 291243254 381421299 617876780 980891674 833928389 1048677341 522527723 223764850 50617939 963598173 281959650 499436870 47455938	0 134 90 158 -1 22 238 269 179 189 121 53 41 41 -1 58 -1 84 -1 149

Note

The result of decoding Sample 1:

```
10
7
1 5 10 5 5
2 0 0 0 9
1 7 10 2 4
1 12 11 9 9
```

```
2 0 0 1 1
1 22 10 2 3
1 32 100 4 4
```

The result of decoding Sample 2:

```
7
20
1 5 44 0 1
1 11 90 0 3
2 0 0 3 4
1 18 68 1 6
1 25 32 2 3
1 31 22 2 3
1 32 26 1 5
1 36 31 3 6
2 0 0 2 5
1 43 10 3 6
2 0 0 5 6
2 0 0 3 4
2 0 0 2 4
2 0 0 1 5
2 0 0 3 5
1 49 48 0 4
2 0 0 1 5
1 50 36 0 6
1 56 48 3 5
1 59 17 3 5
```