J - 素点的力量

大意：找出矩形区域$(x_1,y_1),(x_2,y_2)$中的所有点(x,y)（满足$x+y,x-y$均为质数）的个数。

观察1

设p_1,p_2为素数。矩形内的任意一个素点(p_1,p_2)一定在直线$x-y=p_1$上。

由此可知，矩形内素点都在对应的某条直线$x-y=p_1$被矩形所截成的线段上。

设$x-y=p_1$被矩形所截线段为L，如果L与$x+y=p_2$相交于整点，那么交点为矩形内素点。

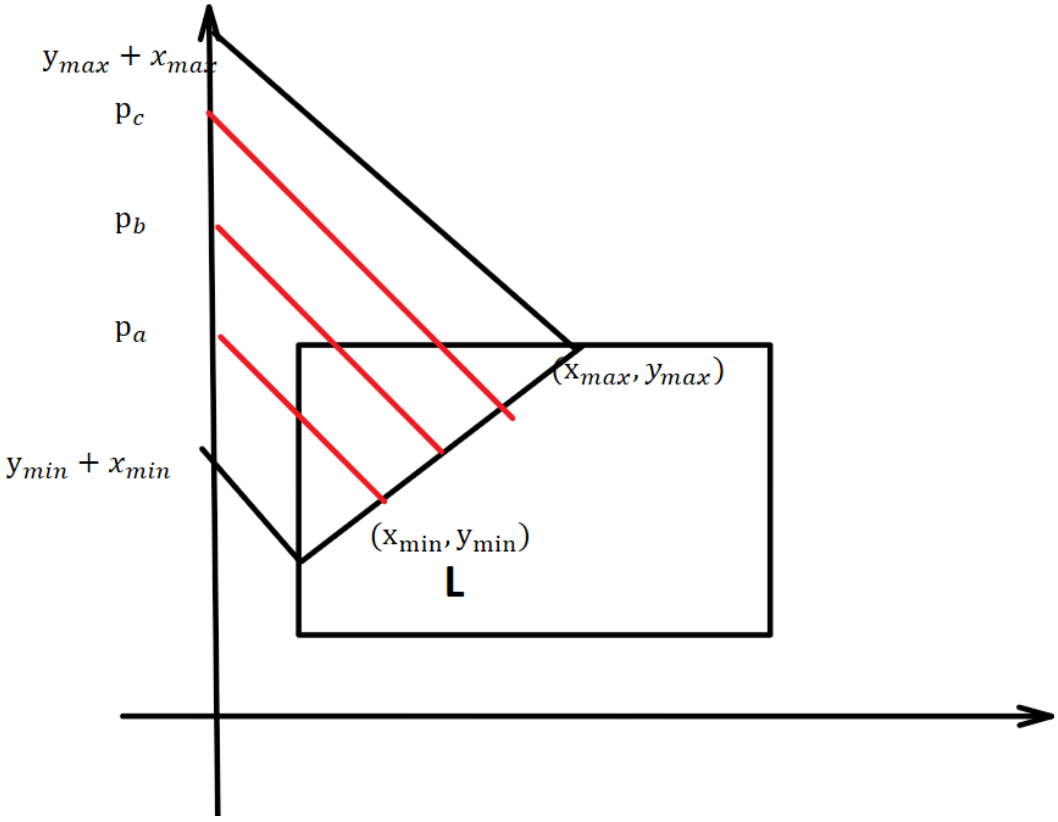
观察2

$x-y=p_1$与$x+y=p_2$相交于整点，当且仅当p_1,p_2或者同时为2，或者同时不为2.

当同时为2时，交点为$(2,0)$，一定不在矩形内，故不考虑质数为2的情况。

考虑所有的p_1，L为$x-y=p_1$被矩形所截线段。

$L(p_1\neq 2)$上的素点个数可以转化为：使得$x+y=p_2(p_2\neq 2)$与L相交的质数个数。



对于给定的$p_1\neq 2$，计算出$x-y=p_1$与矩形的交点$(x_{min},y_{min}),(x_{max},y_{max})$后，$L$上的素点个数就是$[y_{max}+x_{max},y_{min}+x_{min}]$中的非2质数个数，可以用前缀和来实现$O(1)$查询。

时间复杂度为$O(tp)$，p为需要考虑的质数个数。

预处理：

埃氏筛法：

从2遍历到n，若i是质数，则ki不是质数，从i^2开始筛去即可（比i^2更小的合数ki已经被比i更小的质数筛去），余下的数就是$[2,n]$的质数

计算前缀和来表示$(2,n]$中的质数个数。

计算交点$(x_{min},y_{min}),(x_{max},y_{max})$：

先得到$x-y=p_1$在x_1,x_2处的y值，x_1-p_1,x_2-p_1

- 若$x_1-p_1>y_2$或$x_2-p_1<y_1$，一定不相交
- 否则：
  - $x_1-p_1\le y_1$，说明$x-y=p_1$与矩形的边$y=y_1$有交点，令$y_{min}=y_1$后代回原方程得到x_{min}
  - $y_1<x_1-p_1\le y_2$，说明$x-y=p_1$与$y=y_1$有交点，x_{min}即为x_1
  - 可统一为$y_{min}=max\{x_1-p_1,y_1\}$，$x_{min}=p_1+y_{min}$
  - 类似地有$y_{max}=min\{x_2-p_1,y_2\}$，$x_{max}=p_1+y_{max}$

代码：

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

// 若i是质数，则ki不是质数，只需从i*i开始筛掉ki
// （小于i*i的非质数已经被筛掉）
ll sieveSize;
bitset<10000010> bs;
vector<int> p;

void sieve(ll upperbound) {
    sieveSize = upperbound + 1;
    bs.set();
    bs[0] = bs[1] = 0;
    for (ll i = 2; i < sieveSize; ++i) {
        if (bs[i]) {
            for (ll j = i * i; j < sieveSize; j += i) {
                bs[j] = 0;
            }
            p.push_back(i);
        }
    }
}

int t;
int x1, y1_, x2, y2;
long long s[2000006];
```

```cpp
long long calc(int p) {
    // 计算直线与矩形交点
    // x - y = p --> y = x - p
    int ymin = x1 - p;
    int ymax = x2 - p;
    // 不相交
    if (ymax < y1_ || ymin > y2) return 0;
    else {
        // 相交
        ymin = max(ymin, y1_);
        ymax = min(ymax, y2);

        // x + y = p
        // 利用前缀和计算区间内质数个数
        return s[p + 2 * ymax] - s[p + 2 * ymin - 1];
    }
}

int main() {
    cin.tie(NULL);
    ios::sync_with_stdio(false);

    sieve(2000006);
    for (int i = 3; i <= 2000000; ++i) {
        s[i] = s[i - 1];
        if (bs[i]) ++s[i];
    }
    cin >> t;
    while (t--) {
        cin >> x1 >> y1_ >> x2 >> y2;
        // 缩小搜索的上下界
        // 事实上直接（for(int i = 0; i < p.size(); ++i)）也可以过
        long long ans = 0;
        int l = lower_bound(p.begin(), p.end(), x1 - y2) - p.begin();
        if (l == 0) ++l;
        int r = lower_bound(p.begin(), p.end(), x2 - y1_) - p.begin();
        for (int i = l; i <= r; ++i) {
            ans = ans + calc(p[i]);
        }
        cout << ans << endl;
    }
    return 0;
}
```