(LDE)_Linear_Diophantine_Equation.md

idea

- its like extended Euclidian
- but here we have a , b , c and want to get x , y
- a x + b y = c
- we will find : 1 sol & all sol & number of sol in range & sol with min x+y
- note : if a = b = 0 -> c = o -> no. sol = inf , c != 0 -> no. sol = 0

work

- we need to calc a x + b y = c
- we first use extended euclidean to find sol (x0,y0) for a $x0$ + b y0 = gcd(a,b) = g
- x = x0 * (c / g)
- y = y0 * (c / g)
- so if c is divisible by g (gcd(a,b)) the integer sol exists

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
// we call this
// if there is no sol the LDE fun return false we must check it
// c must be div by gcd
bool LDE(int a, int b, int c, int &x0, int &y0, int &gc) {
     gc = gcd(abs(a), abs(b), x0, y0);
    if (c % gc) {
        return false;
    }
    x0 *= c / gc;
    y0 *= c / gc;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
```

code for all sol write this in main

```
    int a,b,c,x,y,g;
    cin >> a >> b >> c;
    if(!LDE(a,b,c,x,y,g))
        cout << -1 << endl;
    vector<int> all_x,all_y;
    for (int i = 1; i < 10000; ++i) {
        all_x.push_back(x + i * b / g);
        all_y.push_back(y - i * a / g);
    }
```

min x+y in range

- x + y = x0 + y0 + k * (b-a) / g
- if a < b, we need to select smallest possible value of k
- if a > b, we need to select largest possible value of k
- if a = b, all solution will have the same sum

number of sol in range

```
void shift_solution(int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int LDE_range(int a, int b, int c, int minx, int maxx, int miny, int maxy) {
    int x, y, g;
    if (!LDE(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;
    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;
    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;
    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;
    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;
    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;
    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);
    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

note

```
        int dec = ceil(double(-x) / (b/g));
        int inc = floor(double(y) / (a/g));
        if(inc < dec)
            cout << -1 << endl;
```