#include <iostream>

#include "Interval.h"

using namespace std;

//конструктор Інтервала без параметрів

Interval::Interval(){ //якщо немає параметрів, то всі значення в інтервалі – 0. Порожній //

інтервал

is\_closed.first=0; //first - лівий кінець, second - правий, 0 - кінець відкритий, 1 –

//замкнутий. Замкнуті інтервали (відрізки) необхідні для різниці інтервалів і квадратних

//нерівностей з >= і <=

is\_closed.second=0;

type\_interval.first=0; //first - лівий кінець, second - правий, 0 - скінченне число, 1 –

//нескінченність

type\_interval.second=0;

a=0; //a,b - числа в інтервалі, a<=b

b=0;

};

//конструктор Інтервала з параметрами

Interval::Interval(bool \_cl, bool \_cr, bool \_tl, bool \_tr, float \_a, float \_b){

type\_interval.first=\_tl;

type\_interval.second=\_tr;

\_cl=\_cl&(!\_tl); //якщо лівий край – нескінченність, то зліва інтервал відкритий, бо не //містить нескінченність в собі

\_cr=\_cr&(!\_tr); //якщо лівий край – нескінченність, то аналогічно

is\_closed.first=\_cl;

is\_closed.second=\_cr;

if ((\_a>=\_b) && (!(\_tl|\_tr))&&(!(\_cl&\_cr))){ //якщо ліве число >= правого, кінці інтервала – скінченні, хоча б один з кінців – відкритий, то інтервал - порожній

a=0;

b=0;

is\_closed.first=0;

is\_closed.second=0;

}else{

if(\_a>\_b && (\_cl&\_cr) && (!(\_tl|\_tr))){ // якщо ліве число > правого, кінці інтервала – скінченні, то інтервал - порожній

a=0;

b=0;

is\_closed.first=0;

is\_closed.second=0;

}else{

a=\_a;

b=\_b;

}

}

}

//setter - функція, що присвоює інтервалу значення параметрів

void Interval::setter(bool \_cl, bool \_cr, bool \_tl, bool \_tr, float \_a, float \_b) {// setter

//аналогічний конструктору

type\_interval.first=\_tl;

type\_interval.second=\_tr;

\_cl=\_cl&(!\_tl);

\_cr=\_cr&(!\_tr);

is\_closed.first=\_cl;

is\_closed.second=\_cr;

if (\_a>=\_b && (!(\_tl|\_tr))&&(!(\_cl&\_cr))){

a=0;

b=0;

is\_closed.first=0;

is\_closed.second=0;

}else{

if(\_a>\_b && (\_cl&\_cr) && (!(\_tl|\_tr))){

a=0;

b=0;

is\_closed.first=0;

is\_closed.second=0;

}else{

a=\_a;

b=\_b;

}

}

}

//функція повертає is\_closed.first

bool Interval::get\_cl() const{

return is\_closed.first;

}

//функція повертає is\_closed.second

bool Interval::get\_cr() const{

return is\_closed.second;

}

//функція повертає type\_interval.first

bool Interval::get\_tl() const{

return type\_interval.first;

}

//функція повертає type\_interval.second

bool Interval::get\_tr() const{

return type\_interval.second;

}

//функція повертає a

float Interval::get\_a() const{

return a;

}

//функція повертає b

float Interval::get\_b() const{

return b;

}

//lMin, rMin, lMax, rMax – допоміжні функції, для знаходження перетину, об’єднання,

різниці інтервалів.

//функція знаходить крайній зліва лівий кінець двох Інтервалів

triple lMin(Interval I1, Interval I2){

triple T; //triple містить інформацію про кінець інтервала, за замовчуванням - нулі

if (I1.get\_tl() || I2.get\_tl()){ //якщо один з кінців – нескінченність, то найменший

//кінець - нескінченність

T.is\_inf=1;

return T;

}

// якщо функція не повернула результат, то обидва кінці скінченні

if (I1.get\_a()<I2.get\_a()){ //якщо число зліва в I1 < ніж в I2, то повертає інформацію про

//лівий кінець I1

T.is\_closed=I1.get\_cl();

T.num=I1.get\_a();

return T;

}

if(I1.get\_a()>I2.get\_a()){

//якщо число зліва в I2 < ніж в I1, то повертає інформацію про лівий кінець I2

T.is\_closed=I2.get\_cl();

T.num=I2.get\_a();

return T;

}

// якщо функція не повернула результат, то обидва числа зліва рівні

T.is\_closed=I1.get\_cl()||I2.get\_cl(); //якщо хоча б один з кінців замкнутий, то результат

//замкнутий. У цьому випадку замкнуті множини містять точку, яка менша будь-якої у

//відкритій множині, тому вважаються лівішими

T.num=I1.get\_a();

return T;

}

//функція знаходить крайній зліва правий кінець двох Інтервалів

triple rMin(Interval I1, Interval I2){

triple T; //triple містить інформацію про кінець інтервала, за замовчуванням - нулі

if (I1.get\_tr() && I2.get\_tr()){//якщо обидва кінці – нескінченність, то найменший –

//нескінченність

T.is\_inf=1;

return T;

}

if (!I1.get\_tr() && I2.get\_tr()){ //якщо кінець I1 – скінченний, I2 – ні, то

//наймений – I1

T.is\_closed=I1.get\_cr();

T.num=I1.get\_b();

return T;

}

if (I1.get\_tr() && !I2.get\_tr()){ //якщо кінець I2 – скінченний, I1 – ні, то

//наймений – I2

T.is\_closed=I2.get\_cr();

T.num=I2.get\_b();

return T;

}

// якщо функція не повернула результат, то обидва кінці скінченні

if (I1.get\_b()<I2.get\_b()){ //якщо число справа в I1 < ніж в I2, то повертає інформацію про правий кінець I1

T.is\_closed=I1.get\_cr();

T.num=I1.get\_b();

return T;

}

if (I1.get\_b()>I2.get\_b()){ //якщо число справа в I2 < ніж в I1, то повертає інформацію про правий кінець I2

T.is\_closed=I2.get\_cr();

T.num=I2.get\_b();

return T;

}

// якщо функція не повернула результат, то обидва числа справа рівні

T.is\_closed=I1.get\_cr()&&I2.get\_cr(); //якщо хоча б один з кінців замкнутий, то результат

//замкнутий

T.num=I1.get\_b();

return T;

}

//функція знаходить крайній справа лівий кінець двох Інтервалів

triple lMax(Interval I1, Interval I2){ //аналогічно до rMin

triple T;

if (I1.get\_tl() && I2.get\_tl()){

T.is\_inf=1;

return T;

}

if (!I1.get\_tl() && I2.get\_tl()){

T.is\_closed=I1.get\_cl();

T.num=I1.get\_a();

return T;

}

if (I1.get\_tl() && !I2.get\_tl()){

T.is\_closed=I2.get\_cl();

T.num=I2.get\_a();

return T;

}

if (I1.get\_a()>I2.get\_a()){

T.is\_closed=I1.get\_cl();

T.num=I1.get\_a();

return T;

}

if (I1.get\_a()<I2.get\_a()){

T.is\_closed=I2.get\_cl();

T.num=I2.get\_a();

return T;

}

T.is\_closed=I1.get\_cl()&&I2.get\_cl();

T.num=I1.get\_a();

return T;

}

//функція знаходить крайній справа правий кінець двох Інтервалів

triple rMax(Interval I1, Interval I2){ //аналогічно до lMax

triple T;

if (I1.get\_tr() || I2.get\_tr()){

T.is\_inf=1;

return T;

}

if (I1.get\_b()>I2.get\_b()){

T.is\_closed=I1.get\_cr();

T.num=I1.get\_b();

return T;

}

if(I1.get\_b()<I2.get\_b()){

T.is\_closed=I2.get\_cr();

T.num=I2.get\_b();

return T;

}

T.is\_closed=I1.get\_cr()||I2.get\_cr();

T.num=I1.get\_b();

return T;

}

//функція повертає довжину інтервала

//Interval.length() – допоміжна функція для знаходження довжини SetIntervals

float Interval::length() const{

if(type\_interval.first||type\_interval.second){// якщо інтервал необмежений, то повертає -1

return -1;

}else{// якщо інтервал обмежений, то повертає b-a - довжину

return b-a;

}

}

//перевизначається оператор ==, перевіряє чи рівні два Інтервали

bool operator== (const Interval& I1,const Interval& I2){

float eps=10e-7; //точність порівняння чисел типу float

//якщо інтервали рівні, то мають однакові type\_interval і is\_closed

if (I1.type\_interval!=I2.type\_interval){

return false;

}

if(I1.is\_closed!=I2.is\_closed){

return false;

}

if ((!I1.type\_interval.first)&&(!I1.type\_interval.second)){ //якщо інтервали скінченні,

//то порівнюються числа в кінцях інтервала

if(abs(I1.a-I2.a)<eps && abs(I1.b-I2.b)<eps){

return true;

}else{

return false;

}

}

if((I1.type\_interval.first)&&(!I1.type\_interval.second)){ //якщо інтервали скінченні тільки

//справа, то порівнюються числа в правих кінцях інтервала

if(abs(I1.b-I2.b)<eps){

return true;

}else{

return false;

}

}

if((!I1.type\_interval.first)&&(I1.type\_interval.second)){ //якщо інтервали скінченні тільки

//зліва, то порівнюються числа в лівих кінцях інтервала

if(abs(I1.a-I2.a)<eps){

return true;

}else{

return false;

}

}

return true;

};

//перевизначається оператор !=, перевіряє чи не рівні два Інтервали

bool operator!= (const Interval& I1,const Interval& I2){

return !(I1==I2);

}

//перевизначається оператор <<, виведення Інтервала

ostream& operator << (ostream& output,const Interval& I){

if(I.get\_cl()){ //виводить [, якщо інтервал замкнутий зліва, (, якщо відкритий

output<<"[";

}else{

output<<"(";

}

if(I.get\_tl()){//виводить –inf, якщо зліва інтервал нескінченний, число I.a , якщо

//скінченний

output<<"-inf, ";

}else{

output<<I.a<<", ";

}

if(I.get\_tr()){ ()){//виводить –inf, якщо справа інтервал нескінченний, число I.b, якщо

//скінченний

output<<"inf";

}else{

output<<I.b;

}

if(I.get\_cr()){ //виводить ], якщо інтервал замкнутий справа, ), якщо відкритий

output<<"]";

}else{

output<<")";

}

return output;

}

//перевизначається оператор >>, введення Інтервала

istream& operator >> (istream& input, Interval &I){

string a,b;

input>>a>>b; //вводяться два рядки a і b

if(a[a.length()-1]==','){//якщо в кінці a стоїть кома, то вона прибирається

a=a.substr(0,a.length()-1);

}

if (a[0]=='['){ //якщо a починається з [, то зліва інтервал закритий, якщо ні –

//відкритий

I.is\_closed.first=1;

}else{

I.is\_closed.first=0;

}

if (b[b.length()-1]==']'){ //якщо b закінчується на ], то справа інтервал закритий, якщо ні –

//відкритий

I.is\_closed.second=1;

}else{

I.is\_closed.second=0;

}

a=a.substr(1,a.length()-1); //з a і b прибираються дужки

b=b.substr(0,b.length()-1);

if (a=="-inf"){ //якщо a==”-inf”, то зліва інтервал необмежений, інакше –

//обмежений, значення рядка перетворюється на ліве число інтервала

I.type\_interval.first=1;

I.a=0;

}else{

I.type\_interval.first=0;

I.a=stof(a);

}

if (b=="inf"){ //якщо b==”inf”, то справа інтервал необмежений, інакше –

//обмежений, значення рядка перетворюється на праве число інтервала

I.type\_interval.second=1;

I.b=0;

}else{

I.type\_interval.second=0;

I.b=stof(b);

}

I.setter(I.get\_cl(), I.get\_cr(),I.get\_tl(), I.get\_tr(), I.get\_a(),I.get\_b()); //I.setter – перевіряє,

//чи правильні значення присвоєні інтервалу

return input;

}

//перевизначається оператор \*, множення (перетин) Інтервалів

Interval operator \* (const Interval& I1,const Interval& I2){

triple P1, P2; //P1, P2 – найбільший лівий і найменший правий кінці

P1=lMax(I1,I2);

P2=rMin(I1,I2);

Interval I(P1.is\_closed, P2.is\_closed, P1.is\_inf,P2.is\_inf, P1.num, P2.num);

//перетин інтервалів – інтервал з кінцями P1, P2

return I;

}

//об’єднання і різниця інтервалів може бути списком інтервалів, тому спочатку треба

//створити SetIntervals

//конструктор SetIntervals без параметрів

SetIntervals::SetIntervals(){

v=vector<Interval>(); //v – порожній вектор

}

//конструктор SetIntervals з параметром

SetIntervals::SetIntervals(vector<Interval> \_v){

// v – вектор з непорожніми значеннями з \_v. Тут невідсортовані за порядком інтервали

//якби були б відсортовані, то можливо можна було б зробити швидші алгоритми

v=vector<Interval>();

for (int i=0; i<\_v.size();++i){

if(\_v[i]!=Interval()){

v.push\_back(\_v[i]);

}

}

}

//функція знаходить кількість Інтервалів у векторі

int SetIntervals::size() const{

return v.size();

}

//функція знаходить довжину SetIntervals - сумарна довжина всіх Інтервалів у векторі

float SetIntervals::length() const{

//додає довжину всіх інтервалів у SetIntervals. Якщо хоча б один з них нескінченний, то

//повертає -1

float sum=0;

for (int i=0; i<v.size(); ++i){

float l=v[i].length();

if (l<0){

return -1;

}

sum+=l;

}

return sum;

}

//getter - повертає вектор v

vector<Interval> SetIntervals::getter() const{

return v;

}

//перевизначається оператор <<, виведення SetIntervals

ostream& operator << (ostream& output,const SetIntervals& S){

//виводяться по черзі всі інтервали в SetIntervals в дужках <>. Якщо інтервал

//порожній, то виводить Empty

vector<Interval> v=S.getter();

output<<"< ";

if(S.size()){

for (int i=0; i<S.size(); ++i){

output<<v[i]<<" ";

}

}else{

output<<"Empty ";

}

output<<">";

return output;

}

//перевизначається оператор >>, введення SetIntervals

istream& operator >> (istream& input, SetIntervals& S){

int n; //n – кількість інтервалів

vector<Interval> v;

if (&input==&cin){ //якщо ввід з консолі, то виводить в консоль “ Enter number of

//intervals:”

cout<<"Enter number of intervals: ";

}

input>>n; //вводиться кількість інтервалів

if (n==0){//якщо n==0, то інтервал порожній

S=SetIntervals(v);

return input;

}

if (&input==&cin){ //якщо ввід з консолі, то виводить в консоль "Enter intervals: "

cout<<"Enter intervals: ";

}

Interval I;

for (int i=0; i<n; ++i){ //вводяться інтервали

input>>I;

v.push\_back(I);

}

S=SetIntervals(v);

return input;

}

//перевизначається оператор +, додавання (об'єднання) Інтервалів

SetIntervals operator+ (const Interval& I1,const Interval& I2){

vector<Interval> v;

if ((I1\*I2)==Interval()){ //якщо перетин інтервалів порожній

if (I1.get\_b()<I2.get\_a()&&(!I1.get\_tr())&&(!I2.get\_tl())){

//якщо інтервали скінченні, правий кінець I1 менший лівого в I2, то додає їх у SetIntervals і повертає їх

v.push\_back(I1);

v.push\_back(I2);

return SetIntervals(v);

}

if (I2.get\_b()<I1.get\_a()&&(!I2.get\_tr())&&(!I1.get\_tl())){

//якщо інтервали скінченні, правий кінець I2 менший лівого в I1, то додає їх у SetIntervals і повертає їх

v.push\_back(I2);

v.push\_back(I1);

return SetIntervals(v);

}

//якщо функція не повернула результат, то лівий кінець одного інтервала співпадає

//з правим іншого

triple P1, P2, P3, P4; //кінці інтервалів

P1=lMin(I1,I2);

P2=lMax(I1,I2);

P3=rMin(I1,I2);

P4=rMax(I1,I2);

if (!(P2.is\_closed||P3.is\_closed)){

//якщо лівий і правий кінці інтервалів відкриті, то додає обидва інтервала в SetIntervals і

//повертає його

v.push\_back(Interval(P1.is\_closed,0, P1.is\_inf, P2.is\_inf, P1.num, P2.num));

v.push\_back(Interval(0,P4.is\_closed, P3.is\_inf, P4.is\_inf, P3.num, P4.num));

return SetIntervals(v);

}

//якщо хоча б один з кінців замкнутий, то об’єднує їх, додає в SetIntervals і

//повертає

v.push\_back(Interval(P1.is\_closed, P4.is\_closed, P1.is\_inf, P4.is\_inf, P1.num, P4.num));

return SetIntervals(v);

}else{ //якщо ітервали перетинаються, то об’єднує їх, додає в SetIntervals і повертає

triple P1, P2;

P1=lMin(I1,I2);

P2=rMax(I1,I2);

v.push\_back(Interval(P1.is\_closed, P2.is\_closed, P1.is\_inf, P2.is\_inf, P1.num, P2.num));

return SetIntervals(v);

}

};

//перевизначається оператор -, різниця Інтервалів

SetIntervals operator- (const Interval& I1,const Interval& I2){

Interval I=I1\*I2; //I – перетин I1 і I2 – який буде відніматися від I1

vector<Interval> v;

if (I==Interval()){ //якщо I – порожній, то результат – I1

v.push\_back(I1);

return SetIntervals(v);

}

if (!I.get\_tl()){ //якщо лівий кінець I – скінченний, то додає інтервал з лівим кінцем

//I1 і правим кінцем I, але замкнутість I змінюється на протилежну

v.push\_back(Interval(I1.get\_cl(),!I.get\_cl(),I1.get\_tl(),0,I1.get\_a(),I.get\_a()));

}

if (!I.get\_tr()){ //якщо правий кінець I – скінченний, то додає інтервал з лівим кінцем

//I і правим кінцем I1, але замкнутість I змінюється на протилежну

v.push\_back(Interval(!I.get\_cr(),I1.get\_cr(),0,I1.get\_tr(),I.get\_b(),I1.get\_b()));

}

return SetIntervals(v); //повертається результат

}

//перевизначається оператор \*, множення (перетин) SetIntervals

SetIntervals operator\* (const SetIntervals& S1, const SetIntervals& S2){

vector<Interval> v;

//множить кожен інтервал з S1 на інтервали з S2. Між собою інтервали S1 і S2 не перетинаються, тому перетини інтервалів з S1 і S2 теж не перетинаються

for (int i=0; i<S1.size(); ++i){

for (int j=0; j<S2.size(); ++j){

v.push\_back(S1.getter()[i]\*S2.getter()[j]);

}

}

return SetIntervals(v);

}

//функція додає Інтервал до SetIntervals (об'єднання)

SetIntervals SetIntervals::addn (const Interval& I) const {

vector<Interval> v1;

Interval I1=I;

for (int i=0; i<v.size(); ++i){

//якщо I1 перетинає v[i], то об’єднуються I1 і v[i], якщо ні, то v[i] додається до v1

if((v[i]+I1).size()==1){

I1=(v[i]+I1).getter()[0];

}else{

v1.push\_back(v[i]);

}

}

v1.push\_back(I1);

//в результаті у v1 всі інтервали, які не перетинаються з I і Інтервал, який є об’єднанням I і

//всіх інтервалів, що перетинаються з ним

return SetIntervals(v1);

}

//перевизначається оператор +, додавання (об'єднання) SetIntervals

SetIntervals operator+ (const SetIntervals& S1, const SetIntervals& S2){

//до S=S1 додаються по черзі інтервали з S2. Під час кожного кроку кількість //інтервалів в S може змінюватися

SetIntervals S=S1;

for (int j=0; j<S2.size(); ++j){

S=S.addn(S2.getter()[j]);

}

return S;

}

//перевизначається оператор -, різниця SetIntervals

SetIntervals operator- (const SetIntervals& S1, const SetIntervals& S2){

//від кожного інтервала в S=S1 віднімається інтервал з S2. Результат додається у v. Цикл //проходить по всім елементам з S2, під кінець кожного кроку S = SetIntervals(v). Під час //кожного кроку кількість інтервалів в S може змінюватися

SetIntervals S3, S=S1;

vector<Interval> v;

for (int j=0; j<S2.size(); ++j){

for (int i=0; i<S.size();++i){

S3=S.getter()[i]-S2.getter()[j];

for (int k=0; k<S3.size(); ++k){

v.push\_back(S3.getter()[k]);

}

}

S=SetIntervals(v);

v.clear();

}

return S;

}

//перевизначається оператор !=, перевіряє чи не рівні два SetIntervals

bool operator!= (const SetIntervals& S1,const SetIntervals& S2){

if(S1.size()!=S2.size()){ //Перевіряє, чи співпадають розміри інтервалів

return true;

}

for(int i=0; i<S1.size(); ++i){ //Перевіряє, чи є для кожного інтервала в S1 рівний

//йому в S2

int eq=0;

for(int j=0; j<S2.size(); ++j){

if(S1.getter()[i]==S2.getter()[j]){

eq=1;

break;

}

}

if(eq==0){

return true;

}

}

return false;

};

//конструктор Квадратної Нерівності без параметрів

QuadraticInequality::QuadraticInequality(){

a=0;

b=0;

c=0;

t="";

}

//конструктор Квадратної Нерівності з параметрами

QuadraticInequality::QuadraticInequality(float \_a, float \_b, float \_c, string \_t){

a=\_a;

b=\_b;

c=\_c;

t=\_t;

}

//функція знаходить корені квадратного рівняння ax^2+bx+c=0

//допоміжна функція для QuadrticInequality.toIntervals()

sols QuadraticInequality::roots() const{

float eps=10e-7; //точність для порівняння float

sols res; //результат

if (abs(a)<eps){

if(abs(b)<eps){

if (abs(c)<eps){//якщо a=b=c=0 – нескінченно багато розв’язків

res.num=3;

return res;

}else{ ){//якщо a=b=0 , с!=0– немає розв’язків

return res;

}

}else{// a=0, b!=0 – 1 розв’язок

res.num=1;

res.x1=-c/b;

return res;

}

}else{//a!=0

float D=b\*b-4\*a\*c; //D - дискримінант

if(D<0){ //якщо D<0 – немає розв’язків

return res;

}

if(D>0){ //якщо D>0 – 2 розв’язки

res.num=2;

if(a>0){

res.x1=(-b-sqrt(D))/(2\*a);

res.x2=(-b+sqrt(D))/(2\*a);

}else{

res.x1=(-b+sqrt(D))/(2\*a);

res.x2=(-b-sqrt(D))/(2\*a);

}

return res;

}

//якщо функція не повернула розв’язок, то D=0 - 1 розв’язок

res.num=1;

res.x1=(-b+sqrt(D))/(2\*a);

return res;

}

}

//функція знаходить значення ax^2+bx+c в точці x

//допоміжна функція для QuadrticInequality.toIntervals()

float QuadraticInequality::value(const float x) const{

return a\*x\*x+b\*x+c;

}

//функція перетворює Квадратну Нерівність в SetIntervals

SetIntervals QuadraticInequality::toIntervals() const{

sols r=this->roots(); //r - корені рівняння

vector<Interval> v;

switch(r.num){

case 3: //якщо нескінченно багато розв’язків

if(t==">"||t=="<"||t=="!="){ //порожній інтервал

return SetIntervals(v);

}else{ //(-inf, inf) – вся множина задовольняє нерівність

v.push\_back(Interval(0,0,1,1,0,0));

return SetIntervals(v);

}

case 2: //якщо 2 розв’язки

if(t=="="){//2 закритих інтервали з коренями - 2 точки

v.push\_back(Interval(1,1,0,0,r.x1,r.x1));

v.push\_back(Interval(1,1,0,0,r.x2,r.x2));

return SetIntervals(v);

}

if(t=="!="){ //3 інтервали – вся множина крім 2 точок

v.push\_back(Interval(0,0,1,0,0,r.x1));

v.push\_back(Interval(0,0,0,0,r.x1,r.x2));

v.push\_back(Interval(0,0,0,1,r.x2,0));

return SetIntervals(v);

}

//для наступних випадків при 2 розв’язках застосовується метод інтервалів – на //проміжках між коренями перевіряється більші, чи менші вони за 0. Для 2 розв’язків //достатньо перевірити 1.

if (t=="<"){

if(this->value(r.x1-1)<0){

v.push\_back(Interval(0,0,1,0,0,r.x1));

v.push\_back(Interval(0,0,0,1,r.x2,0));

}else{

v.push\_back(Interval(0,0,0,0,r.x1,r.x2));

}

return SetIntervals(v);

}

if (t=="<="){

if(this->value(r.x1-1)<0){

v.push\_back(Interval(0,1,1,0,0,r.x1));

v.push\_back(Interval(1,0,0,1,r.x2,0));

}else{

v.push\_back(Interval(1,1,0,0,r.x1,r.x2));

}

return SetIntervals(v);

}

if (t==">"){

if(this->value(r.x1-1)>0){

v.push\_back(Interval(0,0,1,0,0,r.x1));

v.push\_back(Interval(0,0,0,1,r.x2,0));

}else{

v.push\_back(Interval(0,0,0,0,r.x1,r.x2));

}

return SetIntervals(v);

}

if (t==">="){

if(this->value(r.x1-1)>0){

v.push\_back(Interval(0,1,1,0,0,r.x1));

v.push\_back(Interval(1,0,0,1,r.x2,0));

}else{

v.push\_back(Interval(1,1,0,0,r.x1,r.x2));

}

return SetIntervals(v);

}

case 1:

if(t=="="){//повертає корінь

v.push\_back(Interval(1,1,0,0,r.x1,r.x1));

return SetIntervals(v);

}

if (t=="!="){//повертає 2 інтервали – вся множина крім точки

v.push\_back(Interval(0,0,1,0,0,r.x1));

v.push\_back(Interval(0,0,0,1,r.x1,0));

return SetIntervals(v);

}

if(t==">"){

//за методом інтервалів

if(this->value(r.x1-1)>0){

v.push\_back(Interval(0,0,1,0,0,r.x1));

}

if(this->value(r.x1+1)>0){

v.push\_back(Interval(0,0,0,1,r.x1,0));

}

return SetIntervals(v);

}

if(t==">="){

if(a>0){ //якщо a>0, то всі значення >=0

v.push\_back(Interval(0,0,1,1,0,0));

return SetIntervals(v);

}

if(a<0){ //якщо a<0, тільки корінь >=0

v.push\_back(Interval(1,1,0,0,r.x1,r.x1));

return SetIntervals(v);

}

//застосовується метод інтервалів

if(this->value(r.x1-1)>0){

v.push\_back(Interval(0,1,1,0,0,r.x1));

return SetIntervals(v);

}

if(this->value(r.x1+1)>0){

v.push\_back(Interval(1,0,0,1,r.x1,0));

return SetIntervals(v);

}

}

if(t=="<"){ //аналогічно до >

if(this->value(r.x1-1)<0){

v.push\_back(Interval(0,0,1,0,0,r.x1));

}

if(this->value(r.x1+1)<0){

v.push\_back(Interval(0,0,0,1,r.x1,0));

}

return SetIntervals(v);

}

if(t=="<="){ //аналогічно до >=

if(a<0){

v.push\_back(Interval(0,0,1,1,0,0));

return SetIntervals(v);

}

if(a>0){

v.push\_back(Interval(1,1,0,0,r.x1,r.x1));

return SetIntervals(v);

}

if(this->value(r.x1-1)<0){

v.push\_back(Interval(0,1,1,0,0,r.x1));

return SetIntervals(v);

}

if(this->value(r.x1+1)<0){

v.push\_back(Interval(1,0,0,1,r.x1,0));

return SetIntervals(v);

}

}

case 0: //якщо немає розв’язків

if (t=="!="){ //(-inf, inf)

v.push\_back(Interval(0,0,1,1,0,0));

return SetIntervals(v);

}

if (t=="="){ //порожній іниервал

return SetIntervals(v);

}

if (t==">" ||t==">="){

if(this->value(0)>0){ //якщо в одній точці значення > 0, то і у всіх

v.push\_back(Interval(0,0,1,1,0,0));

return SetIntervals(v);

}else{

return SetIntervals(v);

}

}

if (t=="<" ||t=="<="){ //якщо в одній точці значення < 0, то і у всіх

if(this->value(0)<0){

v.push\_back(Interval(0,0,1,1,0,0));

return SetIntervals(v);

}else{

return SetIntervals(v);

}

}

}

return SetIntervals();

}

//перевизначається оператор <<, виведення Квадратної нерівності

ostream&operator<<(ostream&output,const QuadraticInequality& Q){

//виводиться ax^2+bx+c (тип перерахування t) 0

output<<Q.a<<"x^2";

if(Q.b>=0){

output<<"+";

}

output<<Q.b<<"x";

if(Q.c>=0){

output<<"+";

}

output<<Q.c<<Q.t<<0;

return output;

};

//перевизначається оператор >>, введення Квадратної нерівності

istream&operator>>(istream&input, QuadraticInequality& Q){

input>>Q.a>>Q.b>>Q.c>>Q.t;

return input;

};