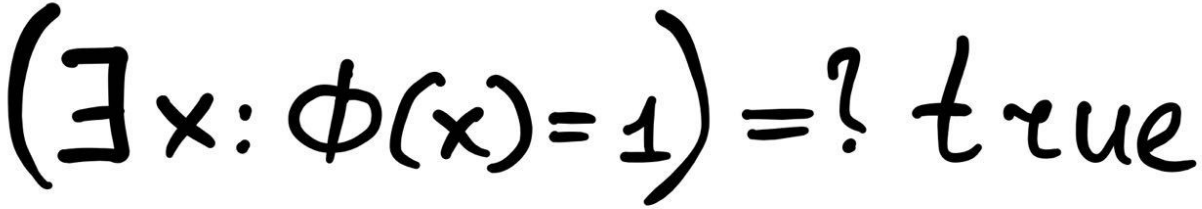
Реализация алгоритма Тарского

Дегтярев Кирилл - докладчик
Былинкин Дмитрий - докладчик

Что такое алгоритм Тарского и почему мы выбрали его ?

Оказывается, что для некоторых логических систем вопросы, записанные в виде



решаются алгоритмически

Пример такой системы - алгебра Тарского

$$TA = \langle \mathbb{R}, >, =, +, \times, 0, 1 \rangle$$

по сути в Алгебре Тарского формулы это условия на многочлены с целыми коэффициентами

Задача элиминации квантора - Quantifier elimination - на настоящий момент актуальным алгоритмом является CAD(Cylindrical Algebraic decomposition ~1970) - используется в экономике и алгебраической геометрии

Почему мы выбрали его ?

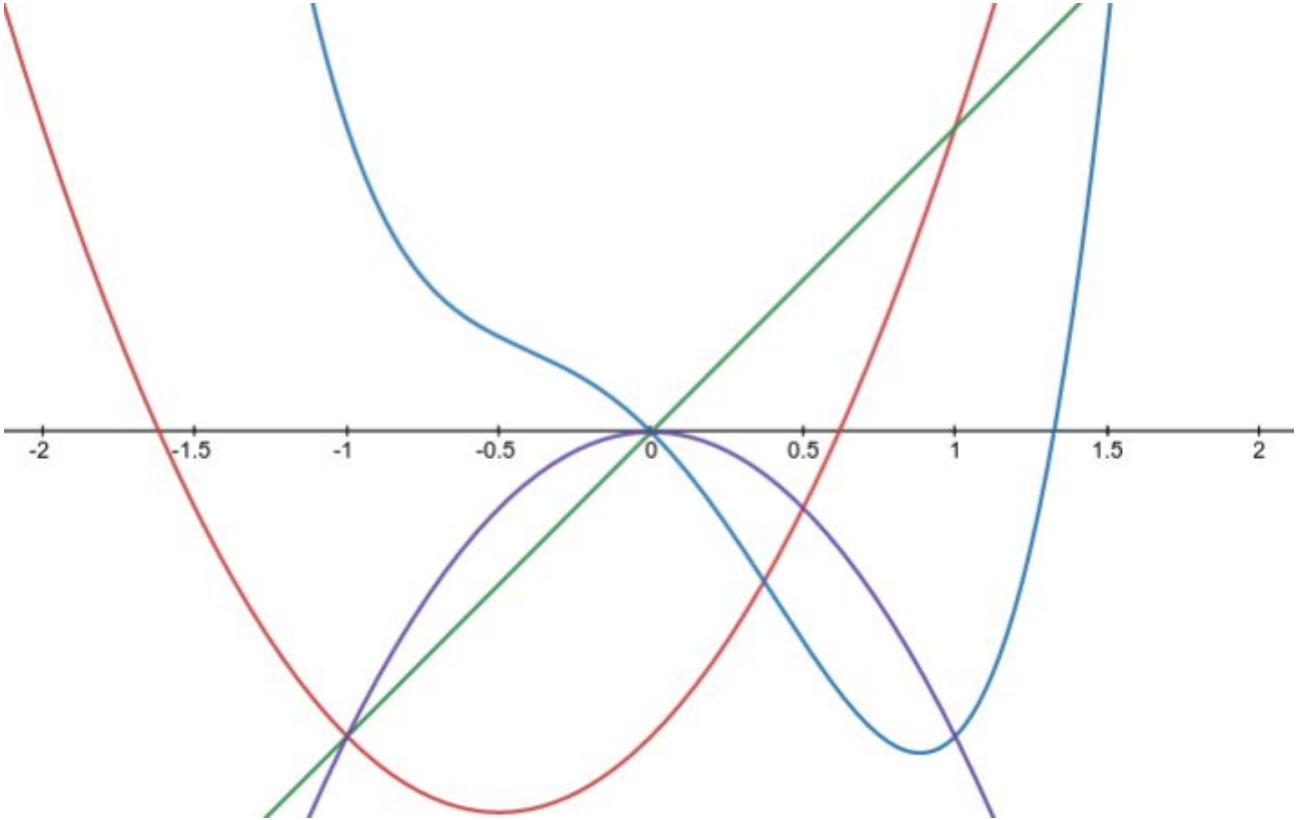
I. Идейно интересный проект - доказательство формул на компьютере
II. Пример вычислительно сложного алгоритма
III. Большое количество задач, решаемых во время написания кода - сеть, громоздкий алгоритм, парсер и т.д.
IV. Большой потенциал для дальнейшей работы

Что делает наша программа ?

Вход программы - формула в виде ДНФ

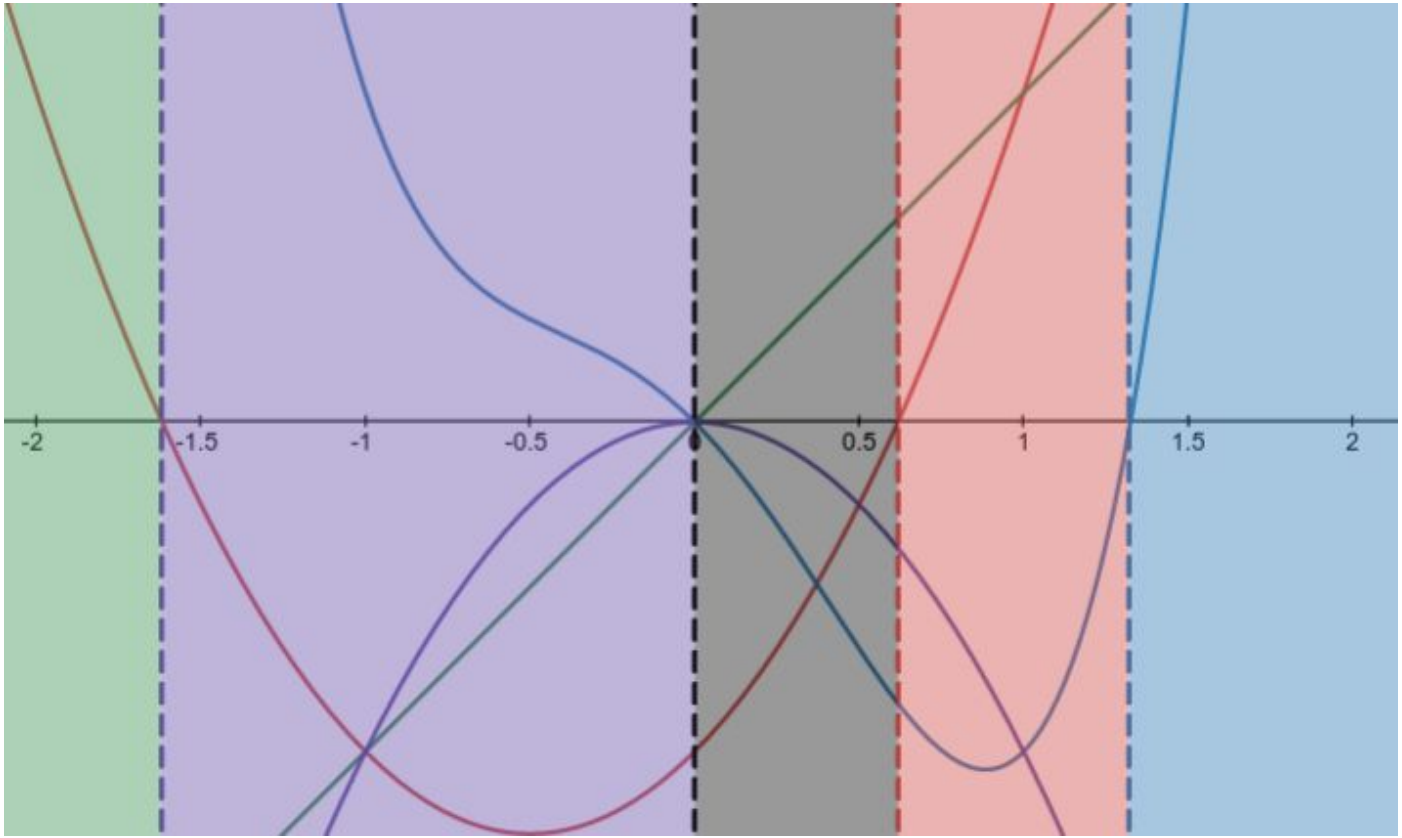
Выход программы - 1 если существует x такой, что формула верна
0, иначе

Главная идея алгоритма



Нужно найти промежутки на которых знаки всех многочленов остаются одинаковыми

Главная идея алгоритма



Задачи решенные в ходе написания программы

1. Парсинг ДНФ
2. Замыкание системы многочленов относительно взятия остатка и производной
3. Распараллеливание насыщения
4. Написание сети
5. Построение таблицы знаков для многочленов
6. Разрешение ДНФ подстановкой знаков многочлена на всех промежутках
7. Тесты по времени

Парсинг - грамматика

DNF → CON | (CON) DNF_T

DNF_T → ∪ DNF

CON → PRED CON_T

CON_T → ∧ CON | ε

PRED → Poly ORD Poly | ¬ PRED

Poly → MULT SUM

SUM → + Poly | ε

MULT → × DEGREE | CONST MULT_T

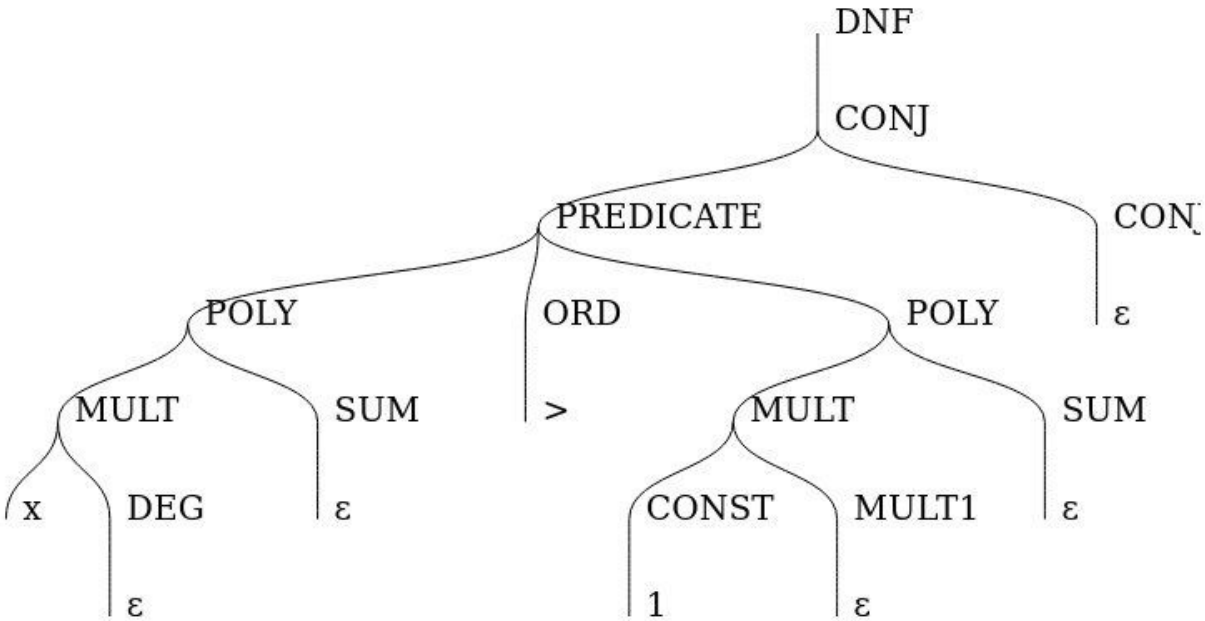
MULT_T → MULT | ε

DEGREE → ^ CONST | ε

ORD → > | =

CONST → get while analyzing lexems

Парсинг - пример дерева разбора “x>1”



Парсинг - реализация

```
void _getUpperNode(const Node* toNode); //поиск верхнего в стеке нетерминала
void _addNode(const Node* addedNode); //замена раскрытого нетерминала
const int _ifMatched(Node* nTerm, Token Term); //анализ LL-таблицы

void _makeDNF(const Node* fromNode);
void _makeConjunct(const Node* fromNode);
void _makePredicate(const Node* fromNode);

void _deleteFromNode(const Node* fromNode);
```

Насыщение системы многочленов

Будем говорить, что система многочленов является **насыщенной**, если она замкнута относительно взятия производной и взятия остатка от деления.

Насыщение - необходимый этап для построения таблицы

Насыщение системы многочленов - реализация

```
//Функция принимает некоторый вектор многочленов, возвращает насыщенную
//относительно взятия производной и взятия остатка от деления систему многочленов
//упорядоченную по возрастанию степени
std::vector<Polynom> full_saturation(std::vector<Polynom> unsaturated)
{
//Насыщаем относительно взятия остатка и взятия производной, пока насыщение не перестанет добавлять новых многочленов
        int size=0;
        while(unsaturated.size()!=size)
        {
                size=unsaturated.size();
                unsaturated=derivation_saturation(unsaturated); //Насыщение по производной
                unsaturated=mod_saturation(unsaturated);        //Насыщение по остатку
        }

        delete_constants(unsaturated);  //Удаление констант
        degree_sort(unsaturated);       //Сортировка по возрастанию степени
        return unsaturated;
}
```

Насыщение по остаткам

Ключевая структура - **матрица остатков**. Элемент i,j равен 1, если было совершено деление многочлена i на многочлен j, -1 если наоборот и 0, если деления не было вообще.

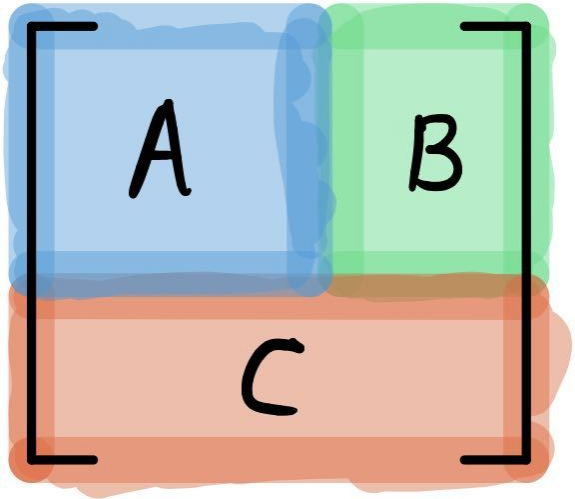
Цикл проходится по матрице и обрабатывает в ней все существующие нули, добавляя остатки от деления в систему

Самая трудозатратная операция в программе

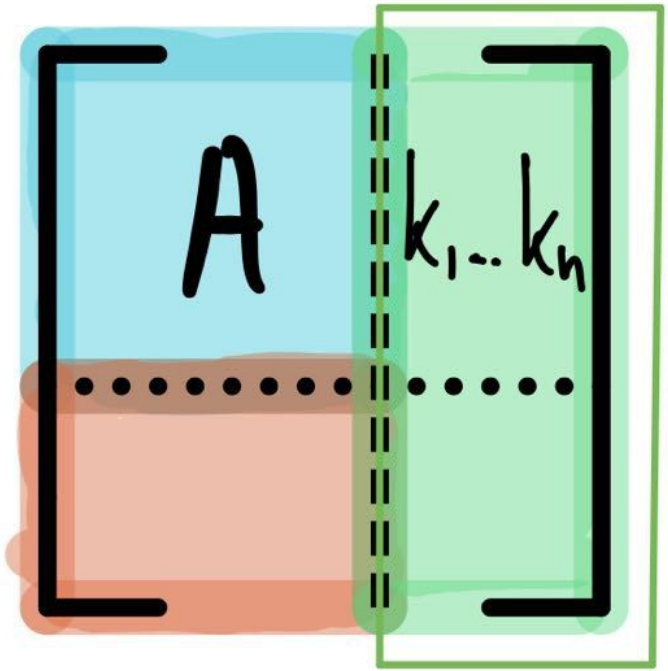
Параллельное насыщение

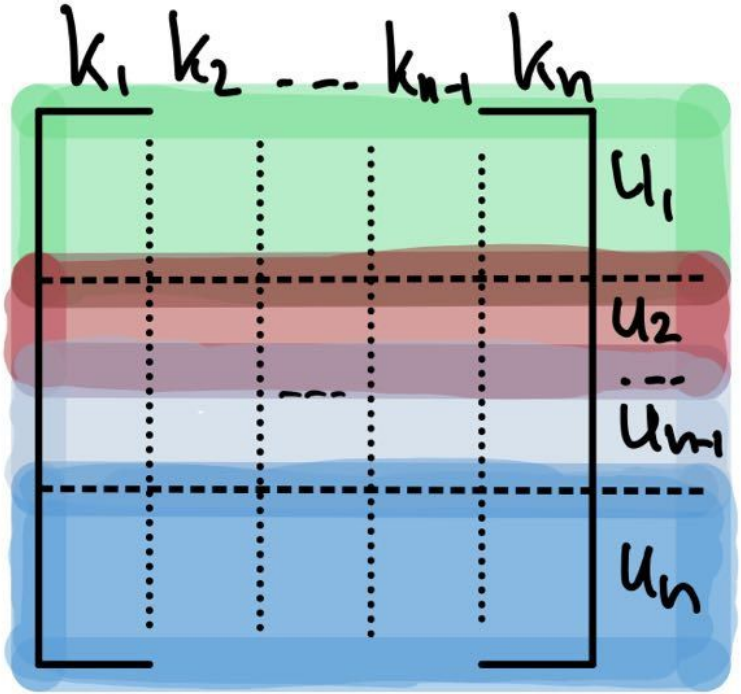
Возможный выход - распараллеливание алгоритма

Делим матрицу на куски и отдаем каждому потоку свой кусок, обратно получаем новые многочлены

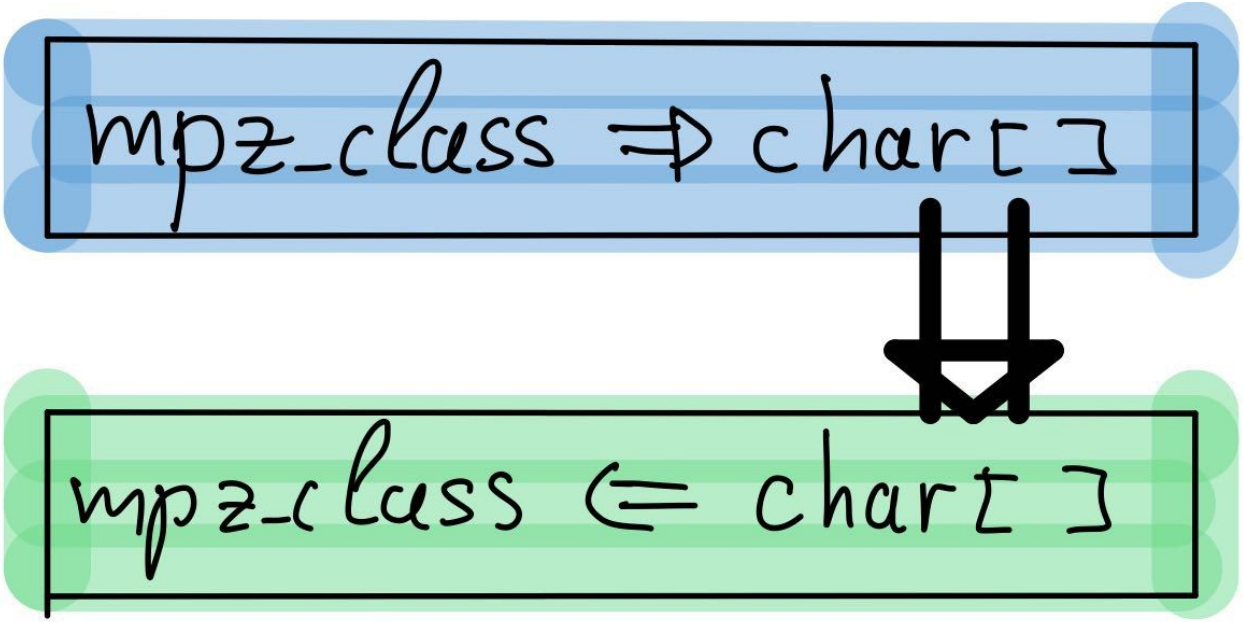


Параллельное насыщение



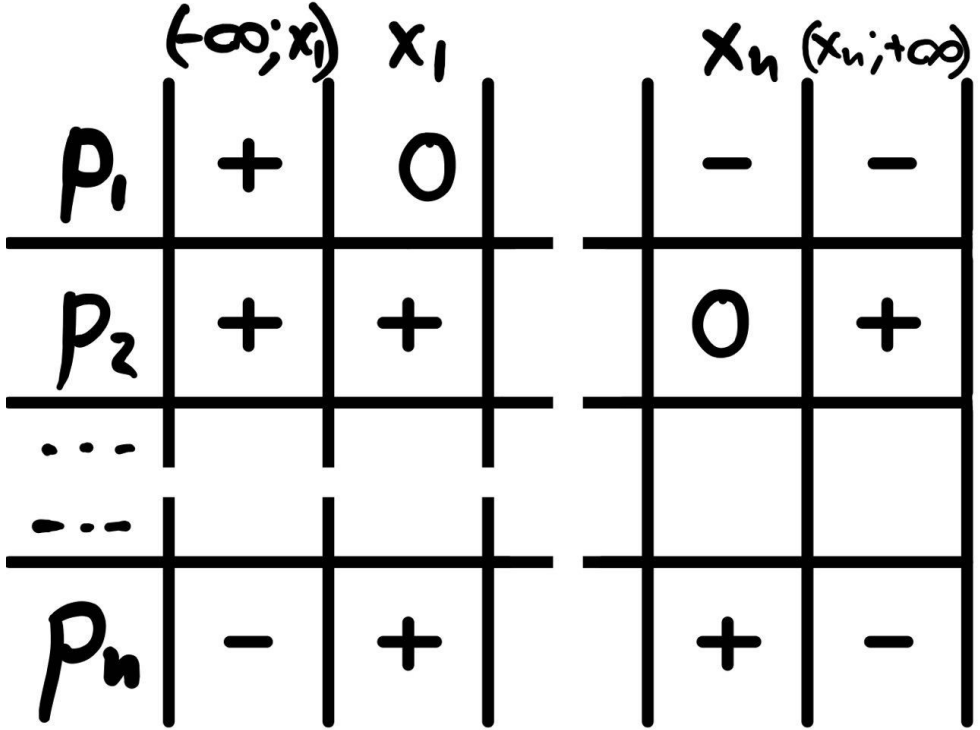


Сеть - передача многочленов



Построение таблицы знаков

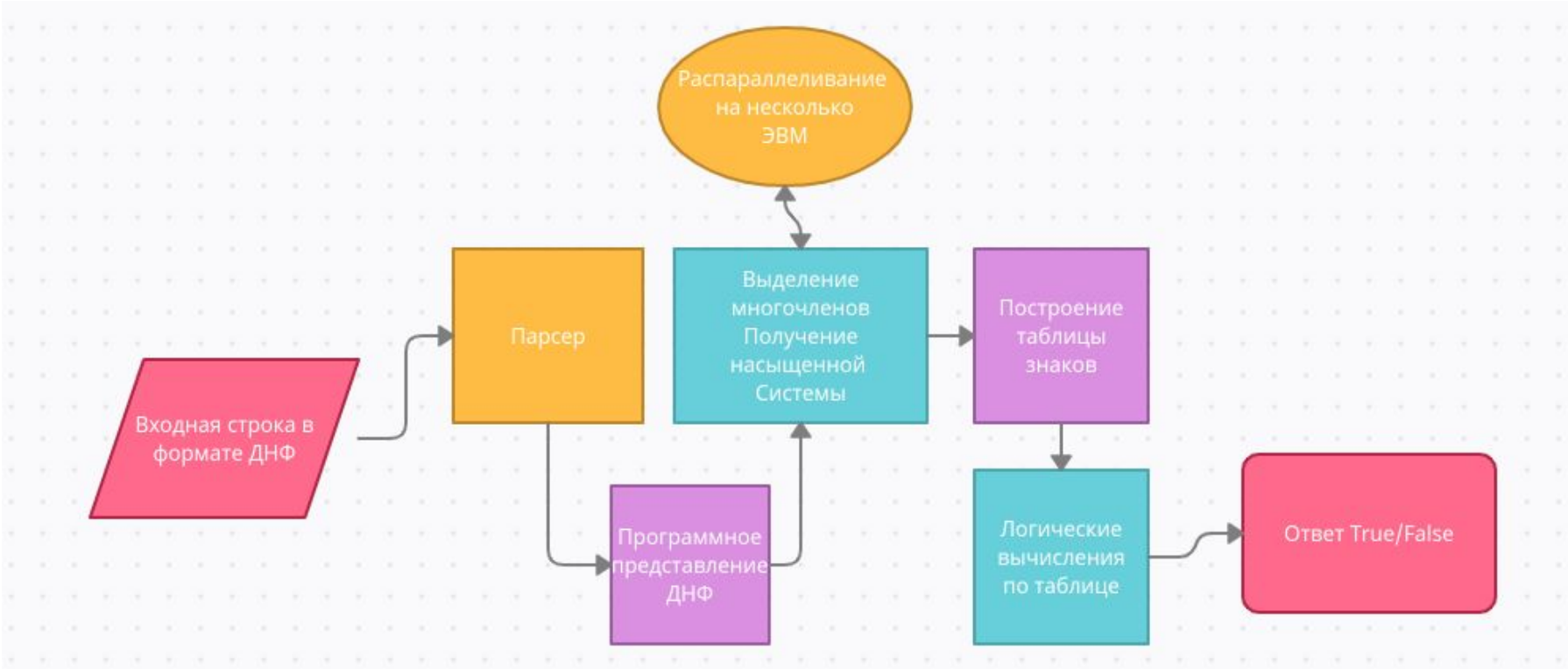
Из таблицы знаков можно получить ответ на вопрос задачи



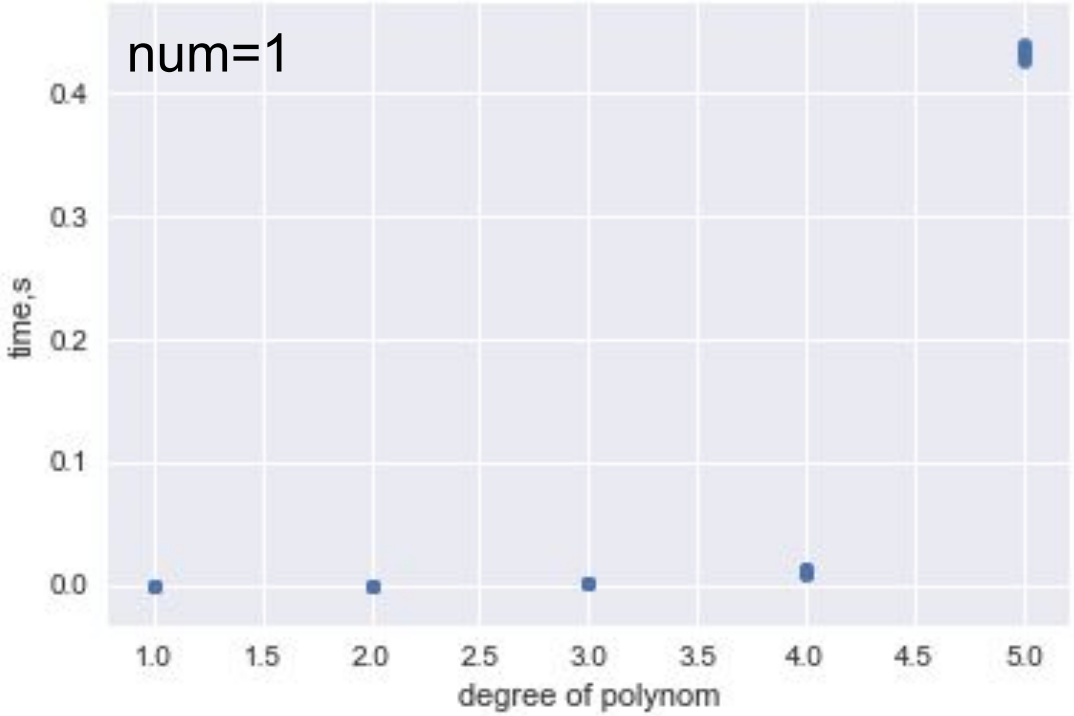
Построение таблицы знаков - реализация

```
//Эта функция принимает на вход таблицу знаков и вектор многочленов, печатает таблицу знаков
void row_print(std::vector<std::vector<int>> t,std::vector<Polynom>);

//Эта функция ставит знаки на крайних интервалах, соответствующих +inf и -inf
void inf_point(std::vector<std::vector<int>> &tars_table, std::vector<Polynom> &polynoms, int p);

//Эта функция заполняет знаки в точках нового многочлена
void set_sign_from_rem(std::vector<std::vector<int>> &tars_table, std::vector<Polynom> &polynoms,int p, int i);

//Эта функция заполняет знаки в промежутках нового многочлена
void set_sign_from_neighbor(std::vector<std::vector<int>> &tars_table, std::vector<Polynom> &polynoms, int p,int i);

//Эта функция расширяет таблицу чтобы добавить корни нового многочлена в уже лежащие в таблице многочлены
void add_new_roots(std::vector<std::vector<int>> &t, int i, std::vector<int> new_roots);

//Эта функция расширяет таблицу чтобы добавить корни нового многочлена в новый многочлен
void add_new_roots_last_row(std::vector<std::vector<int>> &t, std::vector<int> new_roots,int p);


//Функция принимает отсортированный вектор многочленов без констант и возвращет таблицу знаков
std::vector<std::vector<int>> tars_table(std::vector<Polynom> polynoms)
```

Общая схема работы

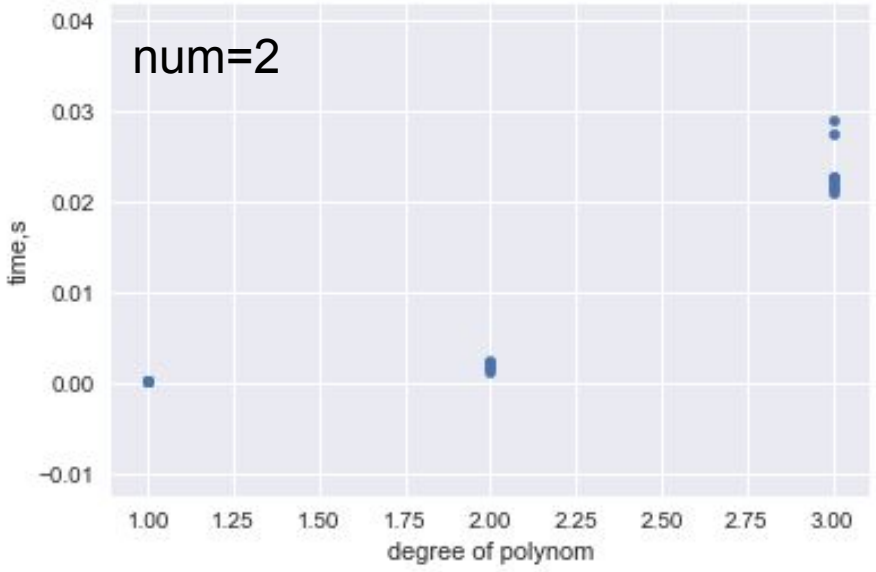


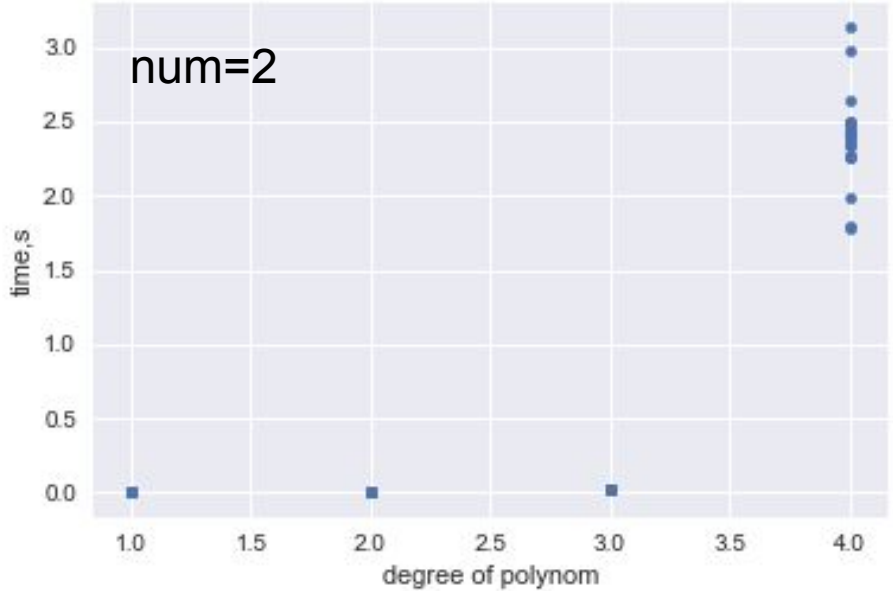
Временные тесты 1



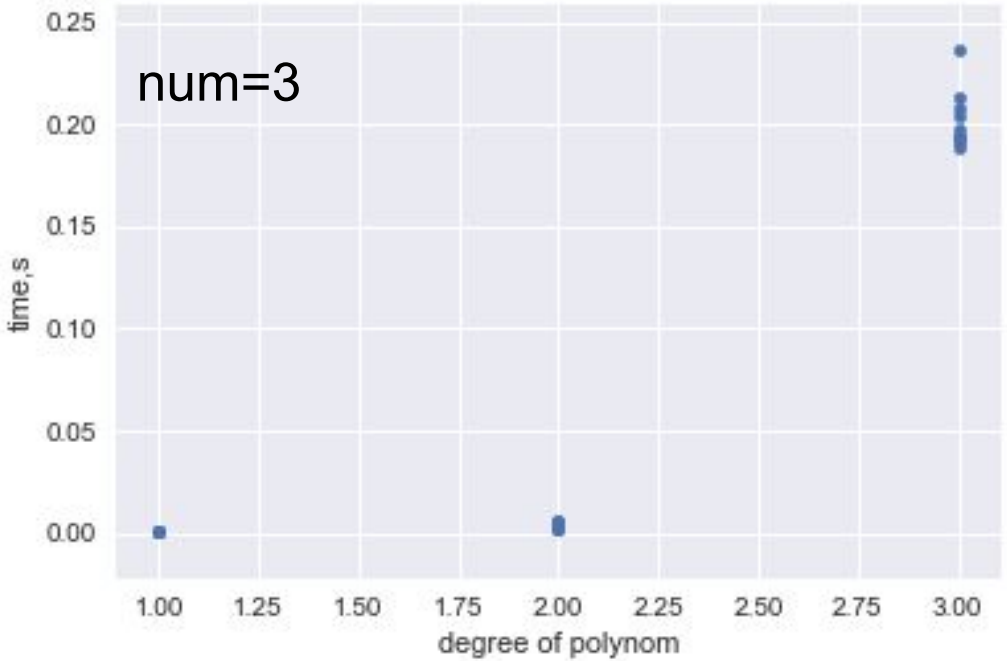
Следующая точка считается порядка часа

Временные тесты 2



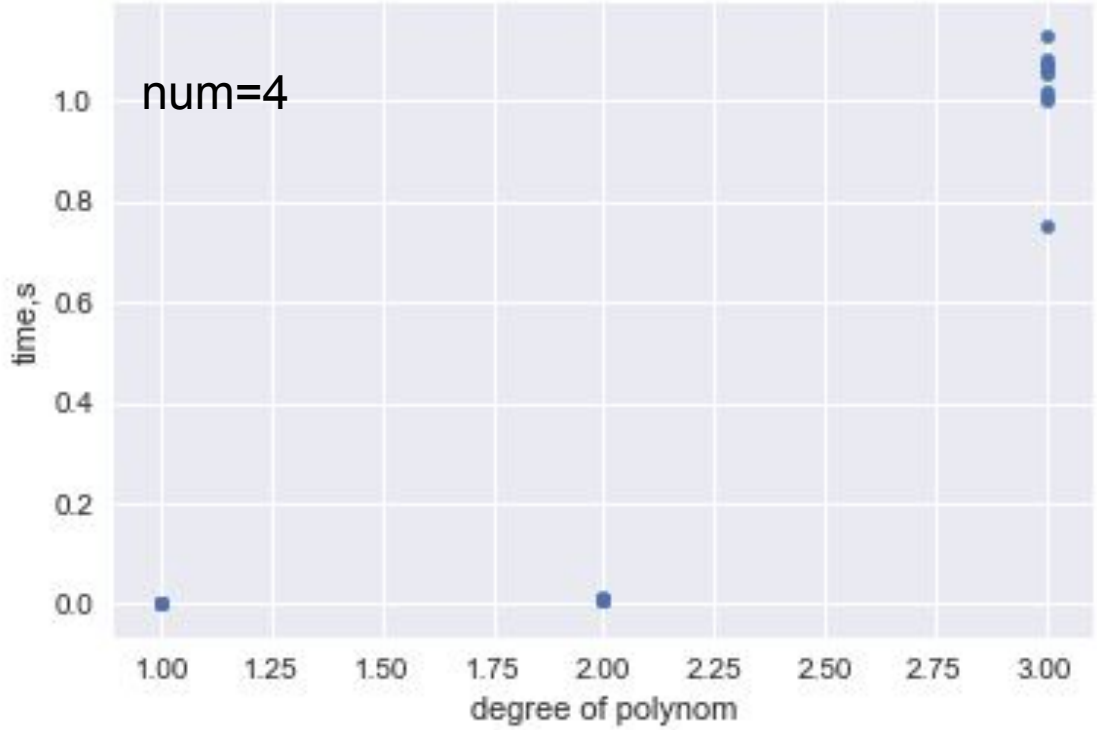


Временные тесты 3



На многочлене четвертой степени работает порядка нескольких минут

Временные тесты 4



Следующая точка считается порядка десяти минут

Заключение

Программа запустилась и работает за вменяемое время на многочленах до шестой степени включительно.

Как можно улучшить программу ? Что можно было бы добавить ?

- Возможно обобщение на случай нескольких переменных
  Дополнительные алгоритмы для деления на линейные многочлены
- Переписывание многочленов на сишные массивы
  Избежать использования чисел длинной арифметики
- Использовать как солвер базового случая CAD
- Вычисления на GPU

Спасибо за внимание

Почта для вопросов :
degtiarev.kd@phystech.edu

Отфоткать хедеры и мейн(мб часть в презу)

Класс многочлена

```
class Polynom
{
private:
        std::vector<mpz_class> _coefficients;          //Здесь лежат коэффициенты многочлена начиная с младшего
public:
        Polynom();
        Polynom(std::vector<mpz_class>);

        void add_coefficient(mpz_class coefficient);    //Добавляет старший коэффициент
        std::vector<mpz_class> get_coefficients();
        Polynom get_derivative();                       //Возвращает производную от многочлена
        void print();                                   //Печатает многочлен
        int get_degree();                               //Возвращает степень многочлена
        void delete_zeros();                            //Удаляет старшие нули, если они есть
        friend bool operator ==(Polynom,Polynom);       //Проверяет, равны ли два многочлена
        friend std::pair<Polynom,Polynom> divide(Polynom,Polynom);      //Возвращает пару частное-остаток
                                                                        //от деления многочленов
};
```

```
class Predicate
{
private:
        Polynom _polynom;
        bool _value{0};
        bool _negative;
public:
        Predicate(Polynom polynom,bool negative);
        Predicate();
        Polynom get_polynom();
        virtual bool calculate(std::vector<int> &column, std::vector<Polynom> &DNF_polynoms)=0;
        double polynom_in_point();
        bool get_value();
        void set_value(bool);
        bool get_negative();
        virtual ~Predicate()=0;
};

class Equality_predicate : public Predicate
{
public:
        Equality_predicate();
        Equality_predicate(Polynom polynom,bool negative);
        bool calculate(std::vector<int> &column, std::vector<Polynom> &DNF_polynoms) override;
        ~Equality_predicate();
};

class Greater_predicate : public Predicate
{
public:
        Greater_predicate();
        Greater_predicate(Polynom polynom, bool negative);
        bool calculate(std::vector<int> &column, std::vector<Polynom> &DNF_polynoms) override;
        ~Greater_predicate();
};
```