

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
ГЛАВА 1. АЛГЕБРА И ИДЕАЛЫ. БАЗИСЫ ГРЁБНЕРА И АЛГОРИТМ БУХБЕРГЕРА.	9
1.1 Кольца, алгебры и идеалы.....	9
1.2 Многочлены от одной переменной. Деление многочленов от многих переменных.	11
1.3 Базисы Грёбнера. Алгоритм Бухбергера.	12
1.4 Свободное программное обеспечение для реализации алгоритма Бухбергера.	14
ГЛАВА 2. РЕАЛИЗАЦИЯ АЛГОРИТМА БУХБЕРГЕРА.	16
2.1 Представление многочленов от многих переменных на языке JavaScript.	16
2.2 Пользовательский интерфейс программы.	17
2.3 Преобразование входных данных для дальнейшей обработки.	19
2.4 Реализация вспомогательных функций для алгоритма Бухбергера.	22
2.5 Реализация алгоритма Бухбергера и построение минимального базиса Грёбнера.	26
2.6 Преобразование вычисленных данных для вывода в HTML документ.	27
2.7 Сравнение полученного приложения с другими реализациями алгоритма Бухбергера.	29
ЗАКЛЮЧЕНИЕ.....	31
ЛИТЕРАТУРА	32

ПРИЛОЖЕНИЯ	34
Приложение 1. Программный код вспомогательных функций, предназначенных для работы с многочленами и одночленами.	34
Приложение 2. Программный код функций, осуществляющих построение базиса Грёбнера и построение минимального базиса Грёбнера.	41
Приложение 3. Программный код функций осуществляющих преобразование входных данных для дальнейшей обработки в приложении	44
Приложение 4. Программный код функций осуществляющих преобразование полученных данных для вывода в HTML-документ	49

ВВЕДЕНИЕ

Известно, что конечный набор многочленов от одной переменной с коэффициентами из какого-либо поля (например, рациональными, действительными или комплексными) всегда имеет наибольший общий делитель, поэтому система алгебраических уравнений, представленная

такими многочленами, $\begin{cases} f_1(x) = 0 \\ \vdots \\ f_n(x) = 0 \end{cases}$ равносильна одному уравнению $d(x)=0$,

где $d(x) = \text{НОД}(f_1(x), \dots, f_n(x))$. Для конечного набора многочленов от нескольких переменных наибольший общий делитель не всегда существует, поэтому равносильный переход от системы уравнений, представленных такими многочленами, к одному уравнению не всегда возможен. Тем не менее, всегда возможен равносильный переход к системе, представленной более простым набором многочленов. О нахождении такого набора и идёт речь в данной работе. Такой набор назван базисом Грёбнера, а алгоритм, позволяющий его найти, алгоритмом Бухбергера, по фамилии его автора, профессора университета города Линца (Австрия) Бруно Бухбергера.

Ниже приводится отрывок из беседы с Бруно Бухбергером начальника сектора Лаборатории информационных технологий Объединённого института ядерных исследований Владимира Петровича Гердта, опубликованной в [5].

“Впервые в Советский Союз Бруно Бухбергер приехал в 1970 году. Первого мая. Естественно, он стал свидетелем праздничных шествий, лозунгов, красных знамен и маршевых песен (некоторые из них он вспоминает и сейчас). Наверно, это было важно в тот момент - видеть воодушевленные радостные лица советских людей, потому что одним из аргументов за предстоящий год работы в Дубне стало желание узнать, как живут ученые за "железным занавесом". Вторым аргументом была возможность заниматься любимым делом.

- В то время, - рассказывает профессор Бухбергер, - я занимался в основном алгоритмической математикой. Тогда математика была, за редкими исключениями, как говорится, "чистая", не связанная с компьютерными технологиями. С другой стороны, уже тогда, более 30 лет назад, существовали большие вычислительные машины, но они не были в центре внимания математиков. Я же хотел соединить математику и информатику. К тому времени моя кандидатская диссертация, сделанная в 1965 году в университете Инсбрука (Тироль), была посвящена разработке универсального алгоритмического метода, позволяющего, в частности, решать системы нелинейных алгебраических уравнений. Я назвал свой метод методом базисов Грёбнера, в честь моего руководителя диссертации, известного тирольского алгебраического геометра Вольфганга Грёбнера. Это был как раз пример, когда математика помогла разработать универсальный метод, который сейчас используется во всех компьютерных алгебраических системах для решения множества задач - математических, промышленных, физических, технологических.

Но 33 года назад ни математики, ни специалисты по информатике не были заинтересованы в применении этого метода, потому что для математиков компьютер не существовал как инструмент работы, а информатики использовали компьютер только для работы с числами. В середине 70-х годов сотрудники ЛИТ (тогда она называлась Лаборатория вычислительной техники и автоматизации) одними из первых в Восточной Европе начали проводить исследования по преобразованию на компьютере символьных математических выражений, то есть заниматься тем, что сейчас называется компьютерной алгеброй. С начала 80-х годов в ЛВТА/ЛИТ эти работы ведутся в секторе компьютерной алгебры, который и возглавляет профессор Владимир Гердт. Но когда Бруно Бухбергер приехал в Советский Союз, это было очень экзотическое направление и, по большому счету, ждало своего часа.

- В Австрии я работал в вычислительном центре и был очень загружен текущей работой. Тогда в СССР Дубна была единственным местом, куда официально брали на работу западных специалистов. Я написал заявку, мне дали стипендию на год. В ОИЯИ мне предоставили все условия для интенсивной плодотворной научной работы над своими математическими методами. Здесь я работал как математик, получил свободу творчества. Великолепно! Это был очень важный год в моей жизни.

Я познакомился с Бруно в 1983 г., - рассказывает Владимир Гердт, - когда он приехал в Дубну на конференцию по программированию и математическим методам решения физических задач и представил доклад по методу базисов Грёбнера. Тогда мы применяли методы компьютерной алгебры для проверки интегрируемости (в смысле обратной задачи рассеяния) нелинейных эволюционных уравнений, и в ряде случаев условия интегрируемости сводились к системам чисто алгебраических уравнений. Поэтому метод Бруно позволил нам значительно продвинуться в нашей работе, и с тех пор одним из главных направлений работ сектора является развитие алгоритмов и программ вычисления базисов Грёбнера, а также их применение к решению различных физических задач. После 1983 г. мы встречались неоднократно на международных конференциях и во время моих визитов в Австрию.

Тогда же, в начале 70-х метод базиса Грёбнера и в Дубне ещё не нашел признания. В то время физики нуждались, прежде всего, в быстрых программах подсчёта большого количества экспериментальных данных. Тем не менее, Бруно морально поддерживали. Заместитель директора лаборатории Н.Н.Говорун сказал Бруно - занимайся, я уверен, у тебя получится, ты своего добьешься. Признание пришло только через шесть лет после работы в России и более чем через десять после создания метода. Произошло это весьма забавно, историю об этом профессор Бухбергер рассказал почти анекдотичную.

- Это было в 1976 году. Я должен был выступать с докладом на конференции в Германии. Доклад был посвящен абстрактной теории алгоритмов. Ко мне подошел молодой профессор, физик, и сказал - извините, но на ваш доклад я не приду. Я подумал, наверно, он очень вежливый человек и извиняется, что не может присутствовать на моем выступлении. Но все оказалось не так. Он сказал, что мог бы прийти, но ему совсем не кажется интересным то, о чем я буду говорить, потому что в его науке есть очень серьезные проблемы и никто не знает, как их решить. Я попросил его привести пример. Он сказал - задана система многочленов, и задан еще один многочлен. Можно ли алгоритмически проверить, лежит ли этот многочлен в идеале, порожденном этим набором многочленов? Чисто математическая задача. Для меня это было впервые, когда физик правильно поставил математическую задачу. Я подумал, что это как раз то, чем я уже занимался несколько лет назад и даже опубликовал статью. Физики ведь не всегда читают математические журналы. Когда я ему об этом сказал, он не поверил и заявил - эта задача неразрешима алгоритмически. Но у меня уже был алгоритм, и эта задача стала бы его практическим приложением! Я прислал ему свою работу, через пять дней он мне позвонил и сказал, что это верно, удивительно, но верно. Потом он написал статью, и с тех пор это направление начало быстро развиваться. Меня пригласили на конференцию, было показано, что этот метод работает. С тех пор написаны тысячи статей о моем методе, а если бы я получал по одному центу от каждого приложения, которое делается методом базиса Грёбнера, то был бы очень богатым человеком!

- Мы тоже принимаем и развиваем технику базиса Грёбнера, - продолжает тему Владимир Гердт, - как я отметил выше, и у нас есть модификации алгоритма, свои программы. Мы называем свои алгоритмы инволютивными и соответствующие базисы также инволютивными, поскольку это понятие восходит к исследованиям начала прошлого века по приведению в инволюцию систем дифференциальных уравнений в частных

производных. Как мы показали, инволютивные базисы являются базисами Грёбнера специального вида, и исследование взаимосвязи инволютивных алгоритмов и алгоритма Бухбергера, чем мы сейчас занимаемся, дает реальную надежду на повышение вычислительной эффективности обоих методов. И мы пригласили Бруно Бухбергера для обсуждения научных вопросов.“

Это было написано в 2005 году, и после этого было опубликовано немало работ по дальнейшему изучению, применениям и модификациям алгоритма Бухбергера.

Цель моей квалификационной работы: составление программы, реализующей алгоритм Бухбергера средствами свободного программного обеспечения.

Задачи, решаемые при выполнении работы:

- изучение теоретического материала;
- составление программы.

ГЛАВА 1. АЛГЕБРА И ИДЕАЛЫ. БАЗИСЫ ГРЁБНЕРА И АЛГОРИТМ БУХБЕРГЕРА.

1.1 Кольца, алгебры и идеалы.

Группой называется множество G с заданной бинарной операцией $*$, удовлетворяющей следующим условиям:

1. $\forall a, b \in G \quad a * b \in G$

2. Ассоциативность:

$$\forall a, b, c \in G \quad (a * b) * c = (a * b) * c = a * b * c$$

3. Существует нейтральный элемент по заданной операции:

$$\exists e \in G \quad \forall a \in G \quad a * e = e * a = a$$

4. Существует обратный элемент по заданной операции:

$$\forall a \in G \quad \exists \tilde{a} \quad a * \tilde{a} = \tilde{a} * a = e$$

Кольцом называется множество K , в котором определены две бинарные операции, называемые сложением $(+)$ и умножением (\cdot) , и удовлетворяющие следующим условиям:

1. Относительно операции сложения K является абелевой группой, т.е. выполнены следующие условия:

a. $\forall a, b, c \in K \quad (a + b) + c = (a + b) + c = a + b + c$

b. $\forall a, b \in K \quad a + b = b + a$

c. $\exists 0 \in K \quad \forall a \in K \quad a + 0 = a$

d. $\forall a \in K \quad \exists \tilde{a} \in K \quad a + \tilde{a} = \tilde{a} + a = 0$

2. $\forall a, b, c \in K \quad a \cdot (b \cdot c) = (a \cdot b) \cdot c$

3. $\forall a, b, c \in K \quad a \cdot (b + c) = a \cdot b + a \cdot c \quad (a + b) \cdot c = a \cdot c + b \cdot c$

Если в кольце есть нейтральный элемент по умножению, называемый единицей и обозначаемый 1 , то кольцо называется *кольцом с единицей*.

Если операция умножения коммутативна, то есть $\forall a, b \in K \quad a \cdot b = b \cdot a$, то кольцо называется *коммутативным*.

Поле называется коммутативное кольцо с единицей, в котором каждый ненулевой элемент обратим, то есть имеет обратный.

Алгеброй над полем F называют кольцо, являющееся одновременно линейным пространством над полем F .

Идеалом кольца K называется подмножество $I \subset K$, для которого выполняются следующие условия:

1. $\forall a, b \in I \quad a + b \in I$
2. $\forall a \in I, f \in K \quad fa \in I$

Если кольцо некоммутативно, то, в зависимости от порядка сомножителей в формулировке свойства 2, различают левосторонние и правосторонние идеалы. В коммутативном кольце, очевидно, любой идеал является и левосторонним, и правосторонним. В дальнейшем рассматриваемые кольца предполагаются коммутативными.

Для любого множества $M \subset K$ можно рассмотреть порождённый им идеал $I(M)$, состоящий из всевозможных конечных сумм вида $\lambda_1 m_1 + \dots + \lambda_r m_r$, где $\lambda_i \in K$, $m_i \in M$.

Базисом идеала I называется семейство $\{a_\alpha\}$, $a_\alpha \in I$, такое, что любой элемент $a \in I$ можно представить в виде $a = \lambda_1 a_{\alpha_1} + \dots + \lambda_t a_{\alpha_t}$, где $\lambda_i \in K$. Если идеал I обладает конечным базисом, то он называется *конечно порождённым*.

Пусть K - некоторое поле (например, \mathbb{Q} , \mathbb{R} или \mathbb{C}) или кольцо Z , $K[x_1, \dots, x_n]$ - кольцо многочленов от n переменных с коэффициентами из K . (Если K – поле, то $K[x_1, \dots, x_n]$, очевидно, алгебра над K .)

Теорема 1. (Теорема Гильберта о базисе) В кольце $K[x_1, \dots, x_n]$ любой идеал конечно порождён.

1.2 Многочлены от одной переменной. Деление многочленов от многих переменных.

Алгоритм нахождения базиса идеала для многочленов от одной переменной над полем K реализуется с помощью деления с остатком одного многочлена на другой. Деление многочленов с остатком производится следующим образом. Пусть $f(x) = a_n x^n + \dots + a_0$, $g(x) = b_m x^m + \dots + b_0$, причём $n \geq m$. Положим $f_1(x) = f(x) - \frac{a_n x^n}{b_m x^m} g(x)$. Если $\deg f_1 > \deg g$, то применим к f_1 аналогичную процедуру и так далее. В итоге получим $f = qg + r$, где $\deg g > \deg r$ (или $r = 0$), где многочлены r и q определены однозначно.

Любой идеал I в кольце $K[x]$ многочленов над полем от одной переменной главный, т. е. порожден одним элементом.

Для того чтобы определить деление с остатком для многочленов от многих переменных введём лексикографическое упорядочивание, т.е. будем считать что моном $x^\alpha = x_1^{\alpha_1} \cdot \dots \cdot x_n^{\alpha_n}$ старше монома $x^\beta = x_1^{\beta_1} \cdot \dots \cdot x_n^{\beta_n}$, если $\alpha_1 = \beta_1, \dots, \alpha_k = \beta_k, \alpha_{k+1} > \beta_{k+1}$ (возможно $k = 0$).

Запись $f = a_\alpha x^\alpha + \dots$ будет означать что $a_\alpha x^\alpha$ старший член многочлена f , то есть x^α старший моном, входящий в f .

Пусть $f = a_\alpha x^\alpha + \dots$ и $g = b_\beta x^\beta + \dots$ два многочлена от n переменных. Если некоторый член $c_\gamma x^\gamma$ многочлена f делится на x^β , то составим $f_1 = f - \frac{c_\gamma x^\gamma}{b_\beta x^\beta} g$. Если в многочлене f_1 найдётся некоторый член, который делится на x^β , то применим к f_1 аналогичное преобразование и так далее. Для того чтобы этот процесс сходился за конечное число шагов, в качестве $c_\gamma x^\gamma$ будем брать старший из всех членов многочленов f , делящихся на x^β . Тогда порядок старшего члена f , делящегося на x^β , будет строго убывать. А любая строго убывающая последовательность мономов от n переменных конечна.

Деление с остатком многочлена f на несколько многочленов f_1, \dots, f_s определяется аналогично. В результате получим $f = u_1 f_1 + \dots + u_s f_s + r$, где у многочлена r нет членов, делящихся на старшие мономы многочленов f_1, \dots, f_s . Говорят, что r остаток от деления многочлена f на многочлены f_1, \dots, f_s . Многочлен r определён не однозначно.

1.3 Базисы Грёбнера. Алгоритм Бухбергера.

Пусть $K[x_1, \dots, x_n]$ -- кольцо многочленов над полем от n переменных, I -- идеал в нём.

Базис Грёбнера образуют (ненулевые) многочлены $g_1, \dots, g_t \in I$, если старший член любого (ненулевого) многочлена $f \in I$ делится на старший член одного из многочленов g_1, \dots, g_t .

Приведём некоторые факты о базисах Грёбнера.

Теорема 2. Многочлены g_1, \dots, g_t образуют базис Грёбнера идеала I тогда и только тогда, когда выполняется одно из следующих эквивалентных условий:

- (а) $f \in I \leftrightarrow$ остаток от деления f на g_1, \dots, g_t равен 0;
- (б) $f \in I \leftrightarrow f = \sum h_i g_i$ и старший моном многочлена f равен старшему из произведений старших мономов h_i и g_i ;
- (в) идеал $L(I)$, порождённый старшими членами элементов идеала I , порождён старшими членами многочленов g_1, \dots, g_t .

Следствие из теоремы 2. Если g_1, \dots, g_t -- базис Грёбнера идеала I , то многочлены g_1, \dots, g_t порождают идеал I .

Теорема 3. У любого ненулевого идеала $I \subset K[x_1, \dots, x_n]$ есть базис Грёбнера.

Теорема 4. Ненулевые многочлены g_1, \dots, g_t образуют базис Грёбнера идеала I тогда и только тогда, когда остаток от деления любого $f \in I$ на g_1, \dots, g_t определён однозначно.

Пусть $f = a_\alpha x^\alpha + \dots$ и $g = b_\beta x^\beta + \dots$ и x_γ – наименьшее общее кратное (далее НОК) мономов x^α и x^β . Положим $S(f, g) = \frac{x_\gamma}{a_\alpha x^\alpha} - \frac{x_\gamma}{b_\beta x^\beta} g$; многочлен $S(f, g)$ строится таким образом, чтобы старшие члены двух его составляющих сократились.

Теорема 5. (Бухбергер) Многочлены g_1, \dots, g_t образуют базис Грёбнера тогда и только тогда, когда остаток от деления многочлена $S(g_i, g_j)$ на g_1, \dots, g_t равен нулю при всех $i \neq j$.

Алгоритм Бухбергера вычисления базиса Грёбнера идеала, порождённого многочленами f_1, \dots, f_s , построен следующим образом. Вычислим остатки от деления многочленов $S(f_i, f_j)$ на f_1, \dots, f_s и все ненулевые остатки добавим к набору f_1, \dots, f_s . Повторим эту процедуру для полученного набора многочленов и так далее. Очевидно, что эта последовательность операций завершится за конечное число шагов, а согласно теореме Бухбергера в итоге получим базис Грёбнера идеала, порождённого многочленами f_1, \dots, f_s .

В случае, когда f_1, \dots, f_s – линейные полиномы, алгоритм Бухбергера сводится к алгоритму Гаусса; в случае, когда f_1, \dots, f_s – полиномы от одной переменной, сводится к алгоритму Евклида.

Базис Грёбнера g_1, \dots, g_t называется минимальным, если $g_i = x^{\alpha_i} + \dots$ и мономы x^{α_i} и x^{α_j} не делятся друг на друга при $i \neq j$. У любого идеала I есть минимальный базис Грёбнера. В самом деле, пусть g_1, \dots, g_t – некоторый базис Грёбнера идеала I и $g_i = x^{\alpha_i} + \dots$. Если x^{α_1} делится на x^{α_2} , то g_2, \dots, g_t – базис Грёбнера идеала I . Действительно, если $f = x^\alpha + \dots \in I$, то согласно определению базиса Грёбнера x^α делится на x^{α_i} при некотором i . Но x^{α_1} делится на x^{α_2} , поэтому x^α делится на x^{α_i} при $i \geq 2$. Это означает, что g_2, \dots, g_t – базис Грёбнера идеала I . Последовательно убирая многочлены, старшие мономы которых делятся на старшие мономы других многочленов,

от базиса Грёбнера g_1, \dots, g_t можно перейти к минимальному базису Грёбнера.[1]

Из определения идеала, порождённого некоторым набором элементов, легко следует, что общие корни многочленов, порождающих идеал в кольце многочленов, являются корнями любого многочлена, принадлежащего этому идеалу. Поэтому замена набора многочленов, порождающих идеал, на более простой (в каком-либо смысле) набор, порождающий тот же идеал, соответствует упрощению системы алгебраических уравнений (далее САУ), представленных первоначальным набором многочленов. Таким более простым набором являются базисы Грёбнера. Таким образом, они позволяют упростить САУ. Базисы Грёбнера применяются также для исследования САУ на совместность, для выяснения эквивалентности двух САУ, для установления конечности числа решений САУ; в литературе описано много других приложений базисов Грёбнера.

1.4 Свободное программное обеспечение для реализации алгоритма Бухбергера.

Целью выпускной квалификационной работы является реализация алгоритма Бухбергера средствами свободного программного обеспечения. На сегодняшний день существует большое разнообразие операционных систем (в дальнейшем ОС). Разрабатывать приложение для каждой ОС нецелесообразно, поэтому кроссплатформенность также является одной из приоритетных задач.

Проблема кроссплатформенности успешно решается использованием для разработки приложения веб-технологий таких как HTML, JavaScript и CSS.

HTML (HyperText Markup Language) – язык разметки гипертекста, представляет собой набор тегов, описывающих структуру документа. Также HTML позволяет размещать изображения, гиперссылки и многое другое.

Тег может описывать весь документ, задавать способ форматирования текста, определять какой элемент должен быть на месте тега или тип информации заключённой в него.

Атрибуты сообщают браузеру, каким образом должен отображаться тот или иной элемент страницы. Атрибуты позволяют сделать более разнообразным внешний вид информации, добавляемой с помощью одинаковых тегов.

CSS (Cascading Style Sheets) – каскадные таблицы стилей, с помощью которых можно задавать точные характеристики практически всех элементов Web-страницы. Задать стиль можно тремя способами: встроить определение стиля в тег, встроить определения стилей в заголовок HTML-документа или вынести таблицу стилей в отдельный файл. Файл с таблицей стилей представляет собой обычный текстовый файл и, как правило, имеет расширение css. Если для одного элемента задано несколько стилей, то применяется каскадирование, которое определяет приоритет того или иного стиля.

JavaScript – скриптовый язык программирования с синтаксисом, немного подобным синтаксисам языков C, Perl и Python.

JavaScript позволяет создавать приложения, выполняемые на стороне клиента, т.е. эти приложения выполняются браузером на компьютере пользователя. Программы (сценарии) на этом языке обрабатываются встроенным в браузер интерпретатором.

С самого начала своего развития язык JavaScript применялся для написания различных клиентских сценариев. Они широко применялись для решения таких задач, как, например, проверка информации, введенной пользователем в форму, перед ее отправкой на сервер или программирование ответных реакций на действия пользователя, делающие веб-страницы интерактивными.

Сегодня с помощью языка JavaScript создаются уже целые веб-приложения. Хорошие примеры таких веб-приложений можно найти среди

сервисов компании Google, например Google Calendar — многофункциональный органайзер в веб-браузере и Google Doc&Spreadsheet — текстовый и табличный редактор, позволяющий работать с офисными документами прямо в окне веб-браузера. Язык JavaScript применяется не только в интернете, но и в таких программах, как, например, Adobe Dreamweaver, Adobe Acrobat Reader и Adobe Photoshop для расширения их возможностей.

jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также jQuery предоставляет удобный API по работе с Ajax. Одним из преимуществ этой библиотеки является её открытый исходный код.

ГЛАВА 2. РЕАЛИЗАЦИЯ АЛГОРИТМА БУХБЕРГЕРА.

2.1 Представление многочленов от многих переменных на языке JavaScript.

Представление многочленов начнём строить с одночленов. Одночлен характеризуется двумя параметрами — коэффициентом и набором степеней переменных. Представим одночлен в виде объекта с двумя свойствами:

- `cof` -- коэффициент одночлена
- `stepen` -- массив степеней переменных

Тогда одночлен $3x_1^3x_2^4$ будет объявлен следующим образом:

```
var mon = new Object;  
mon[`cof`] = 3;  
mon[`stepen`] = [3, 4];
```

Многочлен представим в виде объекта в котором в качестве ключей будем использовать порядковый номер одночлена (начиная с нуля), а в

качестве значений сам моном (его представление). Дополнительно зададим свойство `length` в котором будем хранить количество одночленов в данном многочлене.

Пусть имеется многочлен $3x_1^3x_2^4 + 6x_1^2x_2^3$ и его одночлены заданы как `mon1` и `mon2`, тогда рассматриваемый многочлен будет объявлен следующим образом:

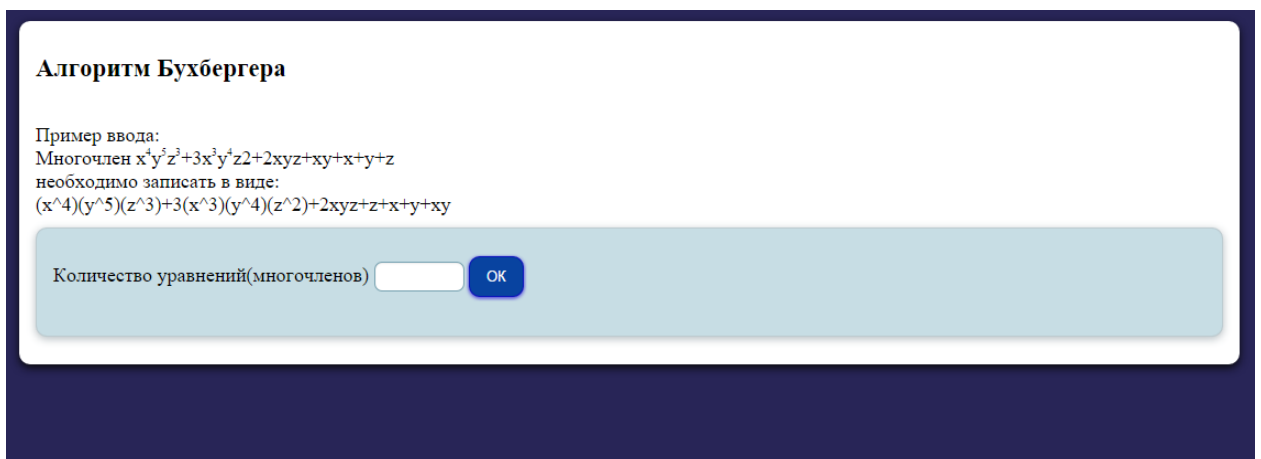
```
Var pol = Object;  
pol[1] = mon1;  
pol[2] = mon2;
```

Аналогично зададим систему многочленов.

2.2 Пользовательский интерфейс программы.

Пользовательский интерфейс реализуется в качестве HTML страницы.

Так как заранее неизвестно количество вводимых полиномов, необходимо чтобы пользователь ввёл эту информацию. Для этого используем текстовое поле и кнопку ОК (элемент `button`), которую пользователь должен нажать после ввода числа многочленов (рис. 1).



Алгоритм Бухбергера

Пример ввода:
Многочлен $x^4y^5z^3 + 3x^3y^4z^2 + 2xyz + xy + x + y + z$
необходимо записать в виде:
 $(x^4)(y^5)(z^3) + 3(x^3)(y^4)(z^2) + 2xyz + z + x + y + xy$

Количество уравнений(многочленов)

Рис. 1. Стартовое окно приложения.

После нажатия на кнопку ОК, происходит запуск функции counts, которая генерирует необходимое количество полей для ввода информации и отображает кнопку запуска алгоритма.

Для ввода многочленов используется текстовые поля (элемент input с типом text). Запуск алгоритма Бухбергера происходит при нажатии на кнопку СТАРТ (рис. 2).

Алгоритм Бухбергера

Пример ввода:
Многочлен $x^4y^2z^3+3x^3y^4z^2+2xyz+xy+x+y+z$
необходимо записать в виде:
 $(x^4)(y^5)(z^3)+3(x^3)(y^4)(z^2)+2xyz+z+x+y+xy$

Количество уравнений(многочленов)

= 0

= 0

= 0

Рис. 2. Окно приложения с полями для ввода многочленов.

Входные данные и базис Грёбнера, полученный в результате работы программы, выводятся в отдельный блок (рис. 3).

Алгоритм Бухбергера

Пример ввода:

Многочлен $x^5y^3z^3+3x^3y^4z^2+2xyz+xy+x+y+z$

необходимо записать в виде:

$(x^4)(y^5)(z^3)+3(x^3)(y^4)(z^2)+2xyz+z+x+y+xy$

Количество уравнений(многочленов)

= 0

= 0

= 0

Исходная система:

$xy-z^2-z = 0$

$x^2-x-yz = 0$

$xz-y^2-y = 0$

Минимальный базис Грёбнера:

$xy-z^2-z = 0$

$x^2-x-yz = 0$

$xz-y^2-y = 0$

$y^2+yz+y-z^2-z = 0$

$2yz = 0$

$2z^3+2z^2 = 0$

Рис. 3. Окно приложения после вычисления минимального базиса Грёбнера.

2.3 Преобразование входных данных для дальнейшей обработки.

Для ввода переменных используются латинские буквы A,..., Z, a,..., z (регистр учитывается); обозначения степени используется знак ^, если показатель равен единице, то его можно не записывать; для отделения одной переменной с показателем от другой используются скобки, а коэффициент записывается в начале выражения.

Таким образом, многочлен $x^5y^3 + 4x^4z$ будет записан в следующем виде: $(x^5)(y^3)+4(x^4)z$.

Данные именно такого вида вводятся в текстовые поля, изображённые на рис. 2.

Данные считанные JS из HTML документа являются строковыми.

Содержимое текстовых полей (рис.2) считывается в массив line. Каждый элемент этого массива содержит строковое представление соответствующего многочлена.

Далее функцией `parse`, каждый элемент массива `line` разбивается знаками `+` и `-` на массив одночленов и записывается в результирующий массив под соответствующим индексом. Результат работы этой функции сохраняется в переменной `pars_line`.

Для работы с массивами показателей в представлениях одночленов, необходимо чтобы они удовлетворяли следующим условиям:

1. Каждый массив показателей должен содержать показатели для всех переменных из введённых полиномов. Если переменная отсутствует в одночлене, то для неё записывается нулевой показатель.
2. Элементы в массивах показателей должны быть упорядочены, то есть показатели для одной и той же переменной в разных массивах должны иметь одинаковые индексы.

Чтобы обеспечить выполнение условий, а также запомнить и упорядочить все введённые переменные, введём понятие эталонного массива.

Эталонным массивом переменных назовём массив, содержащий все переменные введённых полиномов без повторений в упорядоченном виде. Этот массив составляется функцией `etalon` следующим образом:

1. Создаётся пустой массив `result`.
2. В массиве `pars_line` перебираются все строковые представления мономов. В них функция находит символы `A,..., Z`, `a,..., z`, и если найденного символа нет в массиве `result`, то символ добавляется в `result`.
3. Массив `result` сортируется по алфавиту и возвращается.

Результат работы этой функции сохраняется в переменной `etalon_arr`.

Далее в функцию `convert` передаются массивы `pars_line` и `etalon_arr`. Функцией перебираются все строковые представления мономов. В каждом сначала выделяется коэффициент и записывается во временную переменную `cof`, если коэффициент не найден, то записывается единица. Затем создаётся временный массив `tmp_step`, в который будут записываться показатели для

каждого одночлена. В строковом представлении одночлена ищется i -тая переменная из массива `etalon_arr`. Возможные случаи:

- Если переменная не найдена, то в массив `tmp_step` по i -тому индексу записывается нуль.
- Если переменная найдена, и поле неё не стоит символ «^», то в массив `tmp_step` по i -тому индексу записывается единица.
- Если переменная найдена, и за ней следует символ «^», то по i -тому индексу записываются все числовые символы идущие после «^» до символа «)».

Далее формируется объект для одночлена, описанный в пункте 2.1:

```
var tmp_mon = new Object();  
tmp_mon['cof'] = cof;  
tmp_mon['step'] = tmp_step;
```

Для многочлена создаётся временный объект `tmp_pol`. Каждое сформированное представление одночлена записывается в `tmp_pol` по индексу, равному индексу соответствующего строкового представления одночлена.

Для лексикографического упорядочивания полученного полинома, описанного в пункте 1.2, используется функция быстрой сортировки `qsort`. Сравнение одночленов (без учёта коэффициента) осуществляется функцией `comparson`. Эта функция принимает два одночлена и возвращает «+1» если первый одночлен старше второго, «-1» если второй старше первого, «0» если одночлены подобны.

Каждое отсортированное представление многочлена записывается в объект `result` по индексу, равному индексу соответствующего массива строковых представлений одночленов.

Функция `convert` возвращает объект `result`, соответствующий представлению системы, описанному в пункте 2.1. Результат работы функции сохраним в переменной `system`.

2.4 Реализация вспомогательных функций для алгоритма Бухбергера.

В алгоритме Бухбергера используются следующие операции над многочленами и одночленами:

- сумма двух многочленов;
- сравнение двух одночленов;
- произведение многочлена и одночлена;
- частное двух одночленов;
- деление многочленов с остатком;
- НОК двух мономов;
- проверка двух многочленов на равенство.

Сумма многочленов реализуется функцией `sum_pol`, которая принимает два многочлена `a` и `b`, и суммирует их следующим образом:

1. Создаётся объект `res` и предполагается $i = 0, \dots, a.length$, $j = 0, \dots, b.length$, $k = 0, \dots, n$.
2. Пока i не примет значение `a.length-1` и j не примет значение `b.length-1`, с помощью функции `comparsion` сравниваются i -тый одночлен из `a` и j -тый одночлен из `b`. Возможные варианты:
 - если функций `comparsion` вернула «1» (т.е `a[i]` старше чем `b[j]`), то в объект `res` по k -тому индексу записывается `a[i]`, переходим к следующему k . Если i равен `a.length-1`, то в `res` записываются элементы из `b` начиная с j -того. Переходим к следующему i ;
 - если функций `comparsion` вернула «-1» (т.е `a[i]` старше чем `b[j]`), то в объект `res` по k -тому индексу записывается `b[j]`, переходим к следующим j и k . Если j равен `b.length`, то

переходим к следующему i и в `res` дописываются элементы из `a` начиная с i -того;

- если функций `comparsion` вернула «0» (т.е. $a[i]$ и $b[j]$ подобны), то, если сумма коэффициентов $a[i][\text{'cof'}]$ и $b[j][\text{'cof'}]$ не равна нулю, то в объект `res` по k -тому индексу записывается одночлен, у которого коэффициент равен сумме $a[i][\text{'cof'}]$ и $b[j][\text{'cof'}]$, а массив показателей равен $a[i][\text{'cof'}]$. Переходим к следующему j . Если j равен `b.lengt`, то переходим к следующему i и в `res` дописываются элементы из `a`, начиная с i -того. Если i равен `a.length-1`, то в `res` дописываются элементы из `b` начиная с j -того.

3. В `res.length` записывается количество элементов в одночленах в `res`.

Функция `sum_pol` возвращает объект `res` соответствующий представлению многочлена, описанному в пункте 2.1, и являющийся суммой двух входных многочленов.

Произведение многочлена и одночлена реализуется функцией `multi`, принимающей параметры -- одночлен `mon` и `pol` -- многочлен, следующим образом. Создается пустой объект `pol`, в него по i -тому индексу ($i = 0, \dots, \text{pol.length}$) записывается одночлен следующего вида:

- коэффициент равен произведению коэффициента i -того одночлена из `pol` и коэффициента из `mon`;
- массив показателей получен суммированием каждого элемента массива показателей i -того одночлена из `pol` с элементом с таким же индексом массива показателей из `mon`.

В `pol.length` записываем количество одночленов входного одночлена. Функция возвращает объект `pol`, являющийся многочленом.

Частное двух одночленов реализовано функцией `divisionMon`. Функция принимает два одночлена `a` и `b` и, если деление возможно, возвращает объект `res`, являющийся их частным, полученный следующим образом:

- коэффициентом является частное коэффициентов одночленов a и b ;
- каждый элемент массива `res['stepen']` является разностью элементов массивов показателей из a и b с такими же индексами.

Если частное не определено, например, в случае если коэффициент делителя равен нулю, функция вернёт `false`.

Функция `sofDev` считает множитель $\frac{c_\gamma x^\gamma}{b_\beta x^\beta}$ из формулы многочленов f_1, \dots, f_n для нахождения остатка от деления в случае многочленов от многих переменных. В качестве параметров она принимает два многочлена: `pol1` – делимое, `pol2` – делитель. Функция перебирает все одночлены делимого, начиная со старшего. Для рассматриваемого одночлена из делимого функцией `divisionMon` вычисляется частное этого одночлена и старшего одночлена делителя и записывается в переменную `sof`. Если `sof` отлична от `false`, то функция возвратит переменную `sof`. В противном случае происходит переход к следующему одночлену делимого. Если множитель не найден, то функция вернёт `false`.

Остаток от деления многочленов вычисляется функцией `residue`, принимающей в качестве параметров два объекта: a – делимое, b – делитель. В переменную `res` записывается делимое, а в переменную `sof` – множитель для `res` и b , полученный с помощью функции `sofDev`. Затем объявляется переменная `tmp` для хранения произведения делителя и полученного множителя. Далее, пока переменная `s` не примет значение `false`, происходят следующие действия:

1. `s['sof']` умножается на -1 (это необходимо чтобы в дальнейшем получить разность многочленов с помощью функции `multi`).
2. В переменную `tmp` записывается произведение s и b (множителя и делителя), полученное с помощью функции `multi`.

3. Если `res.lenght` (количество одночленов в многочлене `res`) равняется нулю, то остаток от деления равен нулю и функция возвращает объект `res`. В противном случае в переменную `c` записывается множитель для `res` (полученного на шаге 3) и `b`.

В итоге функция вернёт объект `res` являющийся остатком от деления `a` на `b`.

НОК мономов (НОК одночленов без учёта коэффициентов) реализовано функцией `nokMon` принимающей в качестве параметров одночлены `a` и `b`. Функция возвращает одночлен `res`, полученный следующим образом:

- в качестве коэффициента записана единица, так как для обработки данных необходимо, чтобы у любого ненулевого одночлена должен быть коэффициент:
- в массив степеней для каждой переменной записывается её наибольший показатель из `a['stepen']` и `b['stepen']`.

Проверка двух многочленов на равенство осуществляется функцией `eq`, принимающей в качестве параметра два многочлена `p1` и `p2`. Сначала функция сравнивает количество одночленов в `p1` и `p2` и если оно разное, то возвращает `false`. Далее функция поочерёдно перебирает многочлены из `p1` и `p2` с одинаковыми индексами. В каждой паре одночленов, сравниваются коэффициенты, если они не равны, то функция вернёт `false`. В противном случае поэлементно сравниваются массивы степеней рассматриваемых одночленов, если они не равны, то функция вернёт `false`. Затем происходит переход к следующей паре одночленов. Если перебор одночленов закончен, то входные многочлены равны и функция вернёт `true`.

Ввиду некоторых особенностей языка JavaScript также была написана функция `clone` создающая копию представления системы.

2.5 Реализация алгоритма Бухбергера и построение минимального базиса Грёбнера.

Алгоритм Бухбергера реализован функцией `algorithm`. Эта функция принимает в качестве параметра представление системы и возвращает базис Грёбнера построенный для неё (возвращаемый базис также является представлением системы). Построение базиса Грёбнера происходит следующим образом:

1. В переменные `res` и `tmp` записываются копии исходной системы, полученные с помощью функции `clone`.
2. Для i -того и j -того ($i \neq j$) многочленов из `res` выполняются следующие действия:
 - 2.1 В переменную `tmp_nok` записывается НОК старших мономов из `res[i]` и `res[j]`, полученное с помощью функции `nokMon`.
 - 2.2 В переменную `s1` записывается результат последовательного выполнения следующих операций. Сначала, с помощью функции `divisionMon`, вычисляется частное полученного НОК и старшего одночлена из `res[i]`. Затем, с помощью функции `multi`, вычисляется произведение `tmp_nok` и `res[i]`.
 - 2.3 `tmp_nok['cof']` умножается на «-1», это необходимо чтобы в дальнейшем получить разность многочленов.
 - 2.4 В переменную `s2` записывается результат последовательного выполнения следующих операций. Сначала, с помощью функции `divisionMon`, вычисляется частное полученного `tmp_nok` и старшего одночлена из `res[j]`. Затем, с помощью функции `multi`, вычисляется произведение `tmp_nok` и `res[j]`.

- 2.5 В переменную s записывается сумма многочленов s_1 и s_2 , полученная с помощью функции `sum_pol`. Получили S -полином для i -того и j -того многочленов из res .
- 2.6 В переменную s записываем остаток от деления S -полинома на многочлены из res .
- 2.7 Если многочлен s имеет не нулевую длину и не содержится в системе tmp , то он добавляется в tmp .
3. Если величины $res.length$ и $tmp.length$ неравны (т.е. в на шаге 2 были найдены ненулевые остатки от деления многочленов $S(f_i, f_j)$ ($i \neq j$) на f_1, \dots, f_s и добавлены в tmp), то в res записывается копия tmp и происходит переход к шагу 2. Если же величины равны (т.е. в на шаге 2 не были найдены ненулевые остатки от деления многочленов $S(f_i, f_j)$ ($i \neq j$) на f_1, \dots, f_s), то функция вернёт систему res , являющуюся базисом Грёбнера.

Построение минимального базиса Грёбнера реализовано функцией `minimization`, которая принимает систему sys , являющуюся базисом Грёбнера. Попарно перебираются все многочлены из sys . Если старший одночлен i -того многочлена делится на старший одночлен j -того многочлена при $i \neq j$, то коэффициенты всех одночленов из i -того многочлена и $sys[i].length$ приравниваются к нулю. Функция вернёт систему sys которая в итоге будет являться минимальным базисом Грёбнера.

2.6 Преобразование вычисленных данных для вывода в HTML документ.

Для вывода полученные данные преобразуются в строку содержащую HTML элементы.

Функция `monHtml` преобразует одночлен. Она принимает два параметра: `monom` – представление одночлена и `etalon` – эталонный массив переменных и возвращает строковую переменную `str`, полученную следующим образом.

Объявляется переменная `str`, содержащая пустую строку. Затем проверяется значение коэффициента одночлена. Если он равен «-1», то в `str` записывается « - ». Если коэффициент равен «1», то в `str` ничего не записывается. Если коэффициент равен «0», то функция возвратит пустую строку. В других случаях в `str` записывается коэффициент одночлена.

Далее перебираются все элементы массива степеней. Если i -тый элемент массива не равен нулю, то в `str` дописывается i -тая переменная из эталонного массива и элемент `span` с классом `stepen`, содержащий данный элемент. Если все элементы массива степеней нулевые и коэффициент одночлена равен «1» или «-1», то в `str` дописывается единица.

Функция `polHtml` преобразует многочлен в строку. Она принимает два параметра: `polinom` – многочлен и `etalon` – эталонный массив переменных и возвращает значение `false` или строковую переменную `str`, полученную следующим образом.

Если `polinom.length` меньше нуля, то функция вернёт `false`. В противном случае в переменную `str` записывается одночлен с индексом «0», преобразованный в строку с помощью функции `monHtml`. Затем перебираются остальные одночлены. Если коэффициент i -того одночлена больше нуля, то в `str` дописывается «+» и этот одночлен, преобразованный в строку. В противном случае в `str` дописывается этот одночлен, преобразованный в строку.

Функция `sysHtml` преобразует систему в строку. Она принимает два параметра: `bas` – система и `etalon` – эталонный массив переменных и возвращает строковую переменную `str`, полученную следующим образом.

Объявляется переменная `str`, содержащая пустую строку. Перебираются все многочлены системы. В переменную `tmp` записывается i -

тый многочлен системы, преобразованный к строке с помощью функции polHtml. Если tmp отлична от false, то в str дописывается элемент div с классом pol, содержащий значение переменной tmp и строку «=0».

Чтобы показатель степени корректно отображался на HTML-странице, для элементов с классом stepen были написаны следующие стили:

```
font-face: 10px;
position: relative;
bottom: 8px;
text-align: left.
```

2.7 Сравнение полученного приложения с другими реализациями алгоритма Бухбергера.

Реализации алгоритма Бухбергера присутствуют во многих пакетах для систем компьютерной алгебры. Произведём сравнение полученного приложения с некоторыми из них. Для сравнения были выбраны две популярные системы компьютерной алгебры. Результат сравнения приведён в таблице 1. В таблице указано время построения базиса Грёбнера для системы 1.

Название приложения	Поддерживаемые платформы	Время	Минусы приложения	Плюсы приложения
Пакет grobner для системы компьютерной алгебры MAPLE	Windows	0.2109	Приложение является платным; не кроссплатформенное; необходимо затратить время, чтобы изучить синтаксис и разобраться с интерфейсом; занимаемый объём дискового пространства превышает 300МБ; при введении данных необходимо дополнительно указывать список используемых	Содержит функции позволяющие работать с многочленами; для построения базиса Грёбнера можно использовать не только алгоритм Бухбергера.

			переменных.	
Пакет grobner для системы компьютер- ной алгебры MAXIMA	Windows, Linux	0.2131	Имеются версии только для двух платформ; необходимо затратить время, чтобы изучить синтаксис и разобраться с интерфейсом; занимаемый объём дискового пространства превышает 200МБ; при введении данных необходимо дополнительно указывать список используемых переменных.	Является свободным программным обеспечением; содержит функции позволяющие работать с многочленами; для построения базиса Грёбнера можно использовать не только алгоритм Бухбергера.
Алгоритм Бухбергера	Кроссплатформенное	0.2117	Для работы приложения необходим браузер с поддержкой JavaScript; реализован только алгоритм Бухбергера;	Для реализации использовано свободное программное обеспечение; кроссплатформенное; занимаемый объём дискового пространства не превышает 1МБ; программа сама считывает список переменных из введённых данных; имеет интуитивно понятный пользовательский интерфейс.

Таблица 1.

$$\begin{cases} xy - z^2 - z = 0 \\ x^2 - x - yz = 0 \\ xz - y^2 - y = 0 \end{cases} \quad (1)$$

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы были решены следующие задачи:

- изучена научно-методическая литература по теме исследования;
- проведен анализ рассматриваемой предметной области;
- средствами свободного программного обеспечения реализован алгоритм Бухбергера.

Сравнение с другими реализациями алгоритма Бухбергера показало конкурентно способность написанного приложения.

ЛИТЕРАТУРА

1. Прасолов, В.В. Многочлены / В.В. Прасолов — М.: МЦНМО., 2001. — 336 с.
2. Компьютерная алгебра: Символьные и алгебраические вычисления: Пер. с англ. / Бухбергер Б., [и др.]; под ред. Б. Бухбергера, Дж. Коллинза, Р. Лооса. — М.: Мир, 1986.— 392 с.
3. The jQuery Foundation [Электронный ресурс]. — Режим доступа: <https://jquery.org/license> (дата обращения 12.03.2015)
4. Software Licenses in Plain English [Электронный ресурс]. — Режим доступа: <https://tldrlegal.com/license/mit-license> (дата обращения 12.03.2015)
5. Мялковская, Г. Алгоритм Бухбергера, или Метод раскручивающейся спирали / Г. Мялковская // Знание-сила. - 2005. - N 1. - С. . 49-53.
6. Панкратьев, Е.В. Элементы компьютерной алгебры / Е.В. Панкратьев — М.: Интернет-университет информационных технологий, 2007. – 248 С.
7. David Sawyer McFarland JavaScript & jQuery: The Missing Manual, 2nd Edition, O'Reilly Media, 2011. – 536 s.
8. Флэнаган, Д. JavaScript. Подробное руководство / Д. Флэнаган. – М.: Символ-плюс., 2012. – 1080 с.
9. Макконнелл, С. Совершенный код. Практическое руководство по разработке программного обеспечения / С. Макконнелл. – СПб.: Русская Редакция, Питер, 2005. – 896 с.
10. Лёзин, И.А. Решение систем полиномиальных уравнений на ЭВМ / И.А. Лёзин // Программные продукты и системы. - № 3, 2012. - с.22-25.
11. Многочлены от нескольких переменных и алгоритм Бухбергера на Haskell [Электронный ресурс]. — Режим доступа: <http://habrahabr.ru/post/177237/> (дата обращения 23.03.2015)

12. Матросов, Д.Ш., Поднебесова, Г.Б. Элементы абстрактной и компьютерной алгебры / Д.Ш. Матросов, Г.Б. Поднебесова — М.: Академия, 2004. - 240 с.
13. Жуков, К.Д. О возможности применения методов базисов Гребнера в фактор-кольцах / К.Д. Жуков. // Обозрение прикладной и промышленной математики. - Т.19, вып. 2,3, 2012.
14. Кокс, Д. Идеалы, многообразия и алгоритмы. Введение в вычислительные аспекты алгебраической геометрии и коммутативной алгебры: Пер. с англ. / Д. Кокс, Дж. Литтл, Д. О`Ши — М.: Мир, 2000. - 687 с.
15. Прохоренок, Н. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н. Прохоренок — СПб.: БХВ-Петербург, 2010. - 912 с.

ПРИЛОЖЕНИЯ

Приложение 1. Программный код вспомогательных функций, предназначенных для работы с многочленами и одночленами.

```
function comparison (a,b){  
    for (m=0;m<a['stepen'].length; m++){  
        if( a['stepen'][m] > b['stepen'][m] ){ return 1; }  
        if( a['stepen'][m] < b['stepen'][m] ){ return -1; }  
    }  
    return 0;  
};
```

```
function sum_pol(a,b){  
    var j = 0;  
    var k = 0;  
    var i = 0;  
    var res = new Object();  
    for( i = 0; (i < a.length ) && (j < b.length); i++ ){  
        q = comparison( a[i], b[j] );  
        res[k] = new Object();  
        switch(q){  
            case 1: //a[i]>b[j]  
                res[k] = a[i];  
                k++;  
                if(i == (a.length-1)){  
                    while(j<b.length){  
                        res[k] = b [j];  
                        k++;  
                        j++;  
                    }  
                }  
            }  
        }  
    }
```

```

    }
    break;
case -1: //a[i]<b[j]
    res[k] = b[j];
    j++;
    k++;
    i--;
    if( j == b.length ){
        i++;
        while ( i < a.length ){
            res[k] = a [i];
            k++;
            i++;
        }
    }
    break;
case 0: //a[i]==b[j]
    if((a[i]['cof'] + b[j]['cof']).toFixed(10)! = 0){
        res[k]['cof'] = a[i]['cof'] + b[j]['cof'];
        res[k]['stepen'] = a[i]['stepen'];
        k++;
    }
    j++;
    if(j == b.length){
        i++;
        while(i < a.length){
            res[k] = a [i];
            k++;
            i++;
        }
    }

```

```

    }
    if(i == (a.length - 1)){
        while( j < b.length ){
            res[k] = b [j];
            k++;
            j++;
        }
    }
    break;
}
}
res.length = k;
return res;
}

```

```

function multi(mon, poll){
    var i = 0;
    var j = 0;
    var pol = new Object;
    for ( i = 0; i < poll.length; i++ ){
        pol[i] = new Object();
        pol[i]['cof']= poll[i]['cof'] * mon['cof'];
        pol[i]['stepen'] = new Array();
        for( j=0; j < mon['stepen'].length; j++ ){
            pol[i]['stepen'][j] = poll[i]['stepen'][j] + mon['stepen'][j];
        }
    }
    pol.length = poll.length;
    return pol;
}

```

```

function divisionMon(a,b){
    var res = new Object();
    var i = 0;
    var cof ;
    if( (b['cof'] != 0) && (a['cof'] != 0) ){
        res['cof'] = a['cof']/ b['cof'];
    }else{
        return false;
    }
    res['stepen'] = new Array();
    for( i = 0; i < a['stepen'].length; i++){
        if(a['stepen'][i] >= b['stepen'][i]){
            res['stepen'][i] = a['stepen'][i]-b['stepen'][i]
        }else{
            return false;
        }
    }
    return res;
}

```

```

function cofDev(pol1,pol2){
    var i = 0;
    for (i = 0; i < pol1.length; i++){
        cof = divisionMon(pol1[i],pol2[0]);
        if(cof){ return cof; }
    }
    return false;
}

```

```

function residue(a, b){
    var res = a;
    var c = cofDev(res, b);
    var tmp;
    while(c != false){
        c['cof'] *= -1;
        tmp = multi(c,b);
        res = sum_pol(res, tmp);
        if(res.lenght == 0){
            return res;
        }
        c = cofDev(res, b);
    }
    return res;
}

```

```

function nokMon(a,b){
    var i = 0;
    var res = new Object();
    res['cof'] = 1;
    res['stepen'] = new Array();
    for( i = 0; i < a['stepen'].length; i++){
        if( a['stepen'][i] > b['stepen'][i] ){
            res['stepen'][i] = a['stepen'][i];
        }else{
            res['stepen'][i] = b['stepen'][i];
        }
    }
    return res;
}

```

```

function clone(a){
    var res = new Object();
    var i = 0;
    var j = 0;
    var k = 0;
    for( i = 0; i < a.length; i++ ){
        var tmp_pol = new Object();
        for (j = 0; j<a[i].length; j++){
            var tmpmon = new Object();
            tmpmon['cof'] =a[i][j]['cof'];
            var tmp_step = new Array();
            for(k=0; k<a[i][j]['stepen'].length; k++){
                tmp_step[k] = a[i][j]['stepen'][k];
            }
            tmpmon['stepen'] = tmp_step;
            tmp_pol[j] = tmpmon;
        }
        res[i] = tmp_pol;
        res[i].length = a[i].length;
    }
    res.length = a.length;
    return res;
}

```

```

function eq(p1,p2){
    var i = 0;
    var j = 0;
    if(p1.length == p2.length){
        for(i = 0; i < p1.length; i++){

```

```

    if( (p1[i]['cof'] !=0 ) && (p2[i]['cof'] !=0 ) ){
        if( (p1[i]['cof']).toFixed(10) != (p2[i]['cof']).toFixed(10) ){
            return false;
        }
        for(j=0;j<p1[i]['stepen'].length; j++){
            if(p1[i]['stepen'][j]!=p2[i]['stepen'][j]){
                return false;
            }
        }
    }
    return true;
}
return false;
}

```

**Приложение 2. Программный код функций, осуществляющих
построение базиса Грёбнера и построение минимального базиса
Грёбнера.**

```
function minimisation(sys){
    var i=0;
    var j = 0;
    var k = 0;
    var flag = 0;
    for(i = sys.length-1; i >= 0; i--){
        for(j = sys.length-1; j >= 0; j--){
            if(i != j){
                flag = divisionMon(sys[i][0], sys[j][0]);
                if(flag){
                    for(k = 0; k < sys[i].length; k++){
                        sys[i][k]['cof'] = 0;
                    }
                    sys[i].length = 0;
                }
            }
        }
    }
    return sys;
}
```

```
function algoritm(sys){
    var res = clone(sys);
    var tmp = clone(sys);
    var i = 0;
    var j = 0;
```



```

var k = 0;
var m = 0;
var l = 0;
while( 1 ){
    for( i = 0; i < res.length; i++ ){
        for( j = 0; j < res.length; j++ ){
            if( i != j ){
                var tmp_nok = nokMon(res[i][0],res[j][0]); //nok
                var s1 = multi( divisionMon( tmp_nok,res[i][0] ),res[i] );
                tmp_nok['cof'] *= -1;
                var s2 = multi( divisionMon( tmp_nok,res[j][0] ),res[j] );
                var s = sum_pol( s1,s2 );
                for( k=0; ( k<res.length )&&(s != 0)&&(s.length != 0); k++){
                    s = residue( s, res[k] );
                }
                var flag = true;
                if((s != 0) && (s.length != 0)){
                    if(s[0]['cof']<0){
                        for(l = 0;l < s.length; l++){
                            s[l]['cof'] *= -1;
                        }
                    }
                }
            }
        }
        for(m = 0; m < tmp.length; m++){
            if(eq(tmp[m], s)){
                flag = false;
            }
        }
        if( (s.length != 0) && flag){
            tmp[tmp.length] = s;

```

```
        tmp.length++;
    }
}
}
}
if( res.length != tmp.length ){
    res = clone(tmp);
}else{
    return res;
}
}
```

**Приложение 3. Программный код функций осуществляющих
преобразование входных данных для дальнейшей обработки в
приложении**

```
function parse(line){
    var i = 0;
    var j = 0;
    var k = 0;
    var pars_line=new Array();
    for ( i = 0; i<line.length; i++){
        pars_line[i] = line[i].split("+");
        length = pars_line[i].length;
        for(j=0; j<length; j++){
            min_pars = pars_line[i][j].split("-");
            if(min_pars != pars_line[i][j]){
                pars_line[i][j] = min_pars[0];
                var len = pars_line[i].length-1;
                for( k = 1; k < min_pars.length; k++){
                    pars_line[i][pars_line[i].length]=("-"+min_pars[k]);
                }
            }
        }
    }
    return pars_line;
};

function etalon(arr){
    var result = new Array();
    var flag = true;
```

```

var i =0;
var j =0;
var l = 0;
var k = 0;
for(i=0;i<arr.length; i++){
    for(j=0; j<arr[i].length;j++){
        for(k=0;k<arr[i][j].length;k++){
            if(/[a-zA-Z]/.test(arr[i][j][k])){
                for(l=0;l<result.length; l++){
                    if(result[l]==arr[i][j][k]){
                        flag=false;
                    }
                }
                if(flag==true){
                    result.push(arr[i][j][k]);
                }
                flag = true;
            }
        }
    }
}
return result.sort();
};

```

```

function convert(arr,et_arr){
    var i = 0;
    var j = 0;
    var k =0;
    var result = new Object();
    for(i = 0; i<arr.length; i++){
        var tmp_pol = new Object();

```

```

for(j = 0; j < arr[i].length; j++){
    var tmp_mon = new Object();
    var tmp_step = new Array();
    k = 0;
    cof = "";
    if(arr[i][j][k] == '-'){
        k ++;
        cof += '-'
    }
    while( (/[0-9]/.test( arr[i][j][k] ))&&(k<arr[i][j].length) ){
        cof+=arr[i][j][k];
        k++;
    }
    if(cof==""){cof='1';}
    if(cof=="-"){cof='-1';}
    tmp_mon['cof']=+cof;
    for(l=0; l<et_arr.length; l++){
        var perem = false;
        for(k=0; k<arr[i][j].length; k++){
            if( arr[i][j][k] == et_arr[l]){
                perem = true;
                if(k+1<arr[i][j].length){
                    if(arr[i][j][k+1] == "^"){
                        var stepen="";
                        k+=2;
                        while((/[0-9]/.test(arr[i][j][k]))||(arr[i][j][k])=='-
')&&(k<arr[i][j].length)){
                            stepen+=arr[i][j][k];
                            k++;

```



```

if (f_compare == undefined) {
    f_compare = function(a, b) {return ((a == b) ? 0 : ((a > b) ? 1 : -1));};
};

if (f_change == undefined) {
    f_change = function(a, m, n) { var c = a[m]; a[m] = a[n];a[n] = c; };
};

var qs = function (z, r) {
    var m = z,
        n = r,
        x = a[z];
    while(m <= n) {
        while(f_compare(a[m], x) == 1) {m++;}
        while(f_compare(a[n], x) == -1) {n--;}
        if(m <= n) {f_change(a, m++, n--);}
    };
    if(z < n) {qs(z, n);}
    if(m < r) {qs(m, r);}
};

qs(0, a.length-1);
};

```

Приложение 4. Программный код функций осуществляющих преобразование полученных данных для вывода в HTML- документ

```
function monHtml(monom, etalon){
    var str = "";
    var flagCof = false; //флаг будет true если коэффициент единица
    var flagStep = true; //false если хотя-бы одна степень не нулевая
    switch(monom['cof']){
        case -1:
            str = '-';
            flagCof = true;
            break;
        case 1:
            flagCof = true;
            break;
        case 0:
            return str;
            break;
        default:
            str += monom['cof'];
            break;
    }
    var i = 0;
    for(i=0; i < monom['stepen'].length; i++){
        if(monom['stepen'][i] != 0){
            flagStep = false;
            str += etalon[i]
            if(monom['stepen'][i] != 1){
                str += '<span class="stepen">' + monom['stepen'][i] + '</span>';
            }
        }
    }
}
```



```

        }
    }
}
if(flagCof && flagStep){
    str+='1';
}
return str;
}

function polHtml(polinom, etalon){
    var i = 0;
    if(polinom.length > 0){
        var str = monHtml(polinom[0], etalon);
        for( i=1; i < polinom.length; i++){
            if(polinom[i]['cof'] > 0){
                str += '+' + monHtml(polinom[i],etalon);
            }else{
                Str += monHtml(polinom[i],etalon);
            }
        }
        return str;
    }else{
        return false;
    }
}
}

```

```

function sysHtml(bas, etalon){
    var str = "";
    var i = 0;
    var tmp_str = false;

```

```
for(i = 0; i < bas.length; i++){  
    tmp_str = polHtml(bas[i], etalon);  
    if(tmp_str){  
        str += '<div class="pol">' + tmp_str + ' = 0 </div>'  
    }  
}  
return str;  
}
```