электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.021

Вычисление значений математических выражений с использованием метода Рутисхаузера

***Халайджи А.К.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы и сети»

Научный руководитель: Иванова Г.С., д.т.н., профессор
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Компьютерные системы и сети»
v.suzev@bmstu.ru

Задача разбора математических выражений возникла при появлении первых компиляторов. Сами выражения могут быть достаточно сложными, в связи с чем возникает проблема расстановки приоритетов операций в выражении и дальнейшего пооперационного их выполнения. При этом для выражений необходимо проводить анализ, подчиняясь синтаксису языка программирования, на котором они описаны. Например, структура обычного математического выражения значительно отличается от структуры SQL-запроса, в котором помимо обычных операндов и операций также могут быть функции и служебные слова. С такой задачей успешно справляются компиляторы при разборе программного кода, однако в некоторых случаях необходимо использовать отдельные алгоритмы для разбора и вычисления математических выражений, поступающих на вход программы. Ввод выражений во время работы программы повышает её гибкость и универсальность по сравнению с путём прописывания конкретного выражения в программном коде. Примером может являться программа решения систем уравнений, в которых на входе программы подразумевается заранее неизвестная система выражений.

Существует множество алгоритмов разбора выражений. Чаще всего используются алгоритмы, основанные на грамматиках, например, стековые алгоритмы (одним из таких является алгоритм Бауэра-Замельзона). Такие алгоритмы предоставляют обобщённый механизм разбора выражений, причём не только математических, но и, например, конструкций языков программирования. Таких алгоритмов и их реализаций существует огромное множество и найти их не предоставляет никакой сложности. В частности, о них упомянуто в [1].

Помимо алгоритмов, основанных на теории грамматик, есть не менее известный метод Рутисхаузера [2], позволяющий вычислить значение корректно сформированного математического выражения, состоящего лишь из скобок, операций (+, -, *, /) и самих операндов. При этом предполагается, что все незначимые символы (пробелы и т.п.) удаляются перед применением алгоритма вычисления значения выражения. Если в стековых методах порядок выполнения операций определяется грамматикой и таблицей предшествования, то в рассматриваемом методе он учитывается с помощью использования полной скобочной формы.

В целом, метод можно условно разделить на две части: подготовительную и итерационную. В подготовительной части выражение приводится к полной скобочной форме. Если рассматривать этот этап как отдельную задачу, то она не является сложной. Подходы к её решению могут быть разными. Например, можно использовать метод Вунге-Пратта [3].

При разборе выражения может возникнуть проблема, заключающаяся в том, что необходимо различать унарные операции «+» и «-» от двуместных операций сложения и вычитания соответственно. В корректном с точки зрения синтаксиса математическом выражении унарные операции могут встретиться в ограниченном количестве случаев: в начале выражения или после открытия скобки. Поэтому их поиск не вызывает особых трудностей и проблема легко устраняется. Другим решением этой проблемы вместо использования специальной таблицы литералов можно предложить замену всех унарных операций бинарными: для этого к каждой такой операции в качестве левого операнда для заменяющей операции сложения/вычитания необходимо добавить 0. Так, унарная операция «-5» будет заменена на «0-5». В любом из рассмотренных вариантов решение оказывается простым и не вызывает особых затруднений и усложнений всего метода Рутисхаузера в целом. Последним подготовительным действием является расстановка значений индексов для каждого символа выражения (под символами здесь подразумевается операнд, операция или скобка).

Во второй части метода, итерационной, на каждой итерации производится поиск максимальной «тройки», её свёртка и замена в выражении на её значение. Полное описание классического метода Рутисхаузера здесь не приводится в силу того, что он доступен, т.е. его можно легко и быстро найти. Поэтому будет проведён лишь краткий его анализ.

Несомненным достоинством метода Рутисхаузера является то, что его понимание не требует знаний из теории грамматик и доступно любому, даже человеку, не имеющему большого опыта в работе с алгоритмами. Также из-за отсутствия необходимости веде-

ния стеков как в методах, основанных на грамматиках, рассматриваемый метод может во многих случаях быть эффективнее. Однако он имеет и недостатки:

1) так как на каждой итерации необходимо просматривать всё выражение целиком (или некоторую более сложную структуру данных) для поиска максимальной “тройки”, то при достаточно большом размере выражения по количеству входящих в него символов алгоритм в целом выполняется медленнее, чем стековые алгоритмы, в которых весь разбор может быть проведён за один полный проход выражения;

2) по сравнению с методами, использующими теорию грамматик, рассматриваемый метод имеет худшую способность обнаружения ошибок в исходном выражении, поскольку не учитывает позиционную взаимосвязь входящих в него символов. Вместо этого он опирается на поиск и свёртку «троек». Например, если в выражении существует некорректная комбинация («)(» или «5+*3» и т.п.), то ошибка будет найдена на одной из итераций, когда её при анализе нельзя будет отнести ни к одному из типов «троек». Если такая комбинация находится в начале выражения, то при анализе стековыми алгоритмами она найдётся на одной из первых итераций, в то время как при работе рассматриваемого алгоритма она может быть вычислена, когда всё оставшееся выражение будет уже вычислено. Но так как ошибка будет найдена, то итоговое значение всего выражения не будет вычислено, то есть все вычисления «хвоста» выражения были сделаны впустую. Таким образом, если выражение было большим, то на нахождение ошибки в целом будет затрачено намного больше времени, чем было бы затрачено при работе стекового алгоритма;

3) область применения метода Рутисхаузера (исторически одного из первых для вычисления значений математических выражений) – простейшие математические выражения с числами, скобками и арифметическими операциями. Поэтому для анализа сложных математических выражений, в которых могут быть функции, он не подходит. В этом легко можно убедиться, если попробовать расставить индексы по классическому методу для выражения: «F(P)», где F – функция, а P – её аргумент: после нумерации всего выражения счётчик индекса равен 1, а не 0, то есть с точки зрения классического метода данное выражение не является корректным;

Однако необходимость вычисления значений стандартных математических функции встречается в математических выражениях достаточно часто. Причем, как правило, количество аргументов не больше двух. Поэтому при дальнейшем рассмотрении работы модификации метода Рутисхаузера для решения поставленной задачи будут использоваться следующие обозначения: F() – функция без параметров (например, e() или pi()), F(P) – функция с одним параметром, где P – литерал или переменная – параметр, напри-

мер, sin(X) или fact(5) и F(P;P) – функция с двумя параметрами, например, log(a;b) или pow(a;-2). При этом для компактности выражения будут представлены как строки, в которых F – функция, P – параметр, C – вычисленное значение «тройки».

Для решения поставленной задачи было расширено множество типов «троек» и видов их свёртки. Это было обусловлено введением нового вида символов в выражении - «F» и «;» для разделения параметров. В частности для символа «;» решение было следующим - так как без него невозможно отделить параметры функции, то его можно считать частью составной конструкции «параметры функции» и при нумерации символов индекс проставлять таким же, что и для аргументов (по этому же соображению правый аргумент имеет тот же индекс, что и левый). Также было пересмотрено индексирование скобок, так как они тоже частично сменили свою функциональность - если они идут после символа «F», то они отождествляются с ним, то есть им присваивается тот же индекс, так как в рассматриваемой модели у любой функции есть пара скобок для перечисления аргументов вне зависимости от числа последних для того, чтобы отличать имя функции от имени параметра.

Таким образом, после тщательного рассмотрения всех возможных вариантов были выявлены следующие комбинации индексов:

1) n:n:n+1:n – F(<параметр>). Например, sin(a);

2) n+1:n:n+1 – <операнд 1><операция><операнд 2>. Например, a+b;

3) n-1:n:n+1:n – (<значение>). Например, (a) - первый индекс (n-1) необходимо учитывать для того, чтобы отличать данную «тройку» от «тройки» первого типа;

4) n-1:n-1:n:n:n:n-1 – F(<параметр1>;<параметр2>). Например, log(n;a);

5) n:n:n – F(). Например, e() - исключение (у второй скобки индекс тот же).

Отсюда видно, что существует неоднозначность между «тройками» 4-го и 5-го типов, так как последовательность индексов, описывающая 5-ый тип является подпоследовательностью индексов, описывающей 4-ый тип. Однако устранение этой неоднозначности не является большой проблемой, так как функции без параметров всегда имеют конкретный синтаксис: «F()» и могут быть вычислены заранее, а, следовательно, можно их вычислить на подготовительном этапе и заменить в исходном выражении все конструкции вида «F()» на «C». Отыскать все такие конструкции несложно и замену можно осуществить ещё до основных действий на подготовительном этапе метода.

Анализ четырёх оставшихся комбинаций индексов показывает, что они сами по себе не влияют на нумерацию остальных символов в том смысле, что при вставке или удалении любой из «троек» индексы уже пронумерованных символов не изменятся. А из это-

го следует, что с помощью таких «троек» возможно сконфигурировать любое сколь угодно сложное математическое выражение (количество аргументов у функций в которых не превышает двух).

Следует отметить, что метод применим и к выражениям, в состав которых также входят функции с тремя и более параметрами. Так, выражение «F(<параметр 1>;<параметр 2>;...;<параметр n>)» будет иметь следующие индексы: n-1:n:n:n:n:.....:n:n-1. Но в таком случае вероятность ошибки при свёртке возрастает, потому что нет гарантии, что эта последовательность индексов соответствует корректной «тройке» (например, «F(P;;)» не является такой). Следовательно, необходимо проводить дополнительные проверки на корректность числа параметров у функции при свёртке.

С учётом этой модификации этапы, из которых состоит классический алгоритм, остаются теми же, но на подготовительном этапе необходимо дополнительно вычислять все функции без параметров, входящих в выражение, а на итерационном этапе – производить поиск и свёртку сложных «троек». Поиск можно реализовать методом последовательного перебора, который в свою очередь является доказательством того, что не существует других видов «троек». При этом как и в классическом методе, если предположение о типе «тройки» оказалось не верным, то необходимо проверить следующее предположение по алгоритму, который будет приведён ниже. Если ни одна из комбинаций не подошла, то это сигнализирует о наличии ошибки в исходном математическом выражении.

На очередной итерации необходимо найти первый элемент i с максимальным индексом n. Необходимо проверить i+1 элемент. Если он также имеет индекс n, то это может быть либо ошибка, либо «тройка» четвёртого типа. Для проверки на наличие ошибки необходимо проверить все индексы, образующие эту «тройку», а при свёртке проконтролировать, что сворачиваемая функция может иметь два параметра. (потому что, например, выражение «cos(x;y)» некорректно, когда «log(x;y)» корректно).

Если индекс элемента i+1 не равен n, то он не может быть равен n+1 (т.к. максимальный индекс в строке n), поэтому он может быть равен только n-1. Тогда необходимо проверить i+2 элемент. Если его индекс равен n, то тем самым этот элемент замыкает «тройку-операцию». Если он равен n-1, то это сигнал об ошибке, потому что тогда получается «тройка» вида: «n:n-1:n-1», где в качестве «n-1:n-1» может быть или «<параметр>;», или «F(». Но так как и параметр, и функция являются символами, которые не уменьшают значение счётчика, то они не могут быть в строке, потому что тогда бы на i-ом месте должен бы стоять индекс n-2, а там находится индекс n, то есть индекс уменьшается от i-го до (i+1)-го элемента, следовательно данная комбинация недопустима. Если индекс

(i+2)-го элемента равен n-2, то это ни о чём явно не говорит (например, это может быть последовательность закрывающихся скобок, то есть он будет проанализирован на одной из следующих итераций). Поэтому необходимо проверить i-1 элемент. Его индекс не может быть равным n, потому что тогда бы (i-1)-й элемент был бы первым максимальным элементом строки. По той же причине, его индекс не может быть больше, чем n. Поэтому единственным возможным значением индекса является n-1. Видно, что теперь выстроилась «тройка», которая есть как в первом типе, так и в третьей. Поэтому необходимо проверить также и (i-2)-й символ для того, чтобы точно определить, к какому типу относится рассматриваемая «тройка». Если индекс этого элемента n-1, то это функция с одним аргументом. При свёртке необходимо проверить, допустим ли у этой функции только один параметр. Если нет, то это сигнал об ошибке. Если же индекс равен не n-1, то он может быть равен только n-2. Тогда i-1:i:i+1 элементы образуют «тройку-скобки». Таким образом, доказано, что других корректных типов «троек» нет.

В алгоритме были задействованы элементы i+2 и i-2. Следует отметить, что они существуют всегда, если количество символов в строке больше 1. В самом деле, так как любая корректная «тройка» состоит как минимум из трёх символов, то в крайнем случае, если на очередной итерации в строке останется только одна «тройка», то первый и терминальный нули, которые всегда проставляются при индексации на подготовительном этапе, могут являться этими элементами. В случае, если на очередной итерации строка состоит меньше, чем из трёх символов, то, если она состоит ровно из одного символа, и это P (или C), то значение P (или C) и является значением всего выражения (такая ситуация возникает на последней итерации). В любом другом случае строка некорректна (в этом также можно убедиться методом перебора всех возможных комбинаций из одного – двух символов). Наконец, в вырожденном случае, когда строка пустая, её следует воспринимать как ошибочную и не подвергать анализу.

Ниже представлен алгоритм, записанный на псевдокоде:

```
Поиск символа выражения с максимальным индексом n
Запоминание номера i этого символа
Если индекс (i+1) символа = n
  то
    Если количество параметров функции = 2
      то Свёртка F(a;b)
      иначе Установка флага наличия ошибки
```

```
        Всё–если
    иначе
        Если индекс (i+1) символа = n-1
            то
                Если индекс (i+2) символа = n
                    то Свёртка «тройки-операции»
                    иначе
                        Если (i+2) символа = n-2
                            то
                                Если (i-2) символ = n-2
                                    то Свёртка «тройки-скобок»
                                    иначе
                                        Если количество параметров функции = 1
                                            то Свёртка F(a)
                                            иначе Установка флага наличия ошибки
                                        Всё–если
                                Всё–если
                        Всё–если
                Всё–если
        Всё–если
Всё–если
```

Пример. Пусть дано выражение в полной скобочной форме: (F((P*F())/F(P;P)))

Номер шага																					
1		(	F	(	(	P	*	F	(	)	)	/	F	(	P	;	P	)	)	)	
	0	1	2	2	3	4	3	4	4	4	3	2	3	3	4	4	4	3	2	1	0
2		(	F	(	(	P	*	C	)	/	F	(	P	;	P	)	)	)			
	0	1	2	2	3	4	3	4	3	2	3	3	4	4	4	3	2	1	0		
3		(	F	(	(	C	)	/	F	(	P	;	P	)	)	)					
	0	1	2	2	3	4	3	2	3	3	4	4	4	3	2	1	0				
4		(	F	(	C	/	F	(	P	;	P	)	)	)							
	0	1	2	2	3	2	3	3	4	4	4	3	2	1	0						

5			(	F	(	C	/	C	)	)
	0	1	2	2	3	2	3	2	1	0
6			(	F	(	C	)	)		
	0	1	2	2	3	2	1	0		
7			(	C	)					
	0	1	2	1	0					
8 - результат			C							
	0	1	0							

Таким образом, были рассмотрены основные подходы к решению задачи вычисления математических выражений, одним из которых является метод Рутисхаузера. Проведён анализ этого метода, а также предложена модификация, позволяющая расширить его область применения до математических выражений, содержащих математические функции. При этом алгоритм не исправляет недостатков классического метода, связанных с большой вычислительной сложностью на больших выражениях, возможностью чёткого отслеживания ошибок в выражении (как в алгоритмах, основанных на грамматиках) и с необходимостью приведения выражения к полной скобочной форме. Тем не менее он может оказаться полезным для разработчиков любого уровня подготовленности для разбора относительно небольших математических выражений.

Список литературы

1. Обратная польская нотация. Режим доступа: http://algolist.ru/syntax/revpn.php (дата обращения 28.04.2014)
2. Алгоритм Рутисхаузера. Режим доступа: http://algolist.ru/syntax/parsear.php (дата обращения 28.04.2014)
3. Prett Parsing — метод Вогана Пратта для разбора выражений. Режим доступа: http://habrahabr.ru/post/50349/ (дата обращения 28.04.2014)