Дерево выражений или дерево синтаксического разбора. Алгоритм формирования дерева.

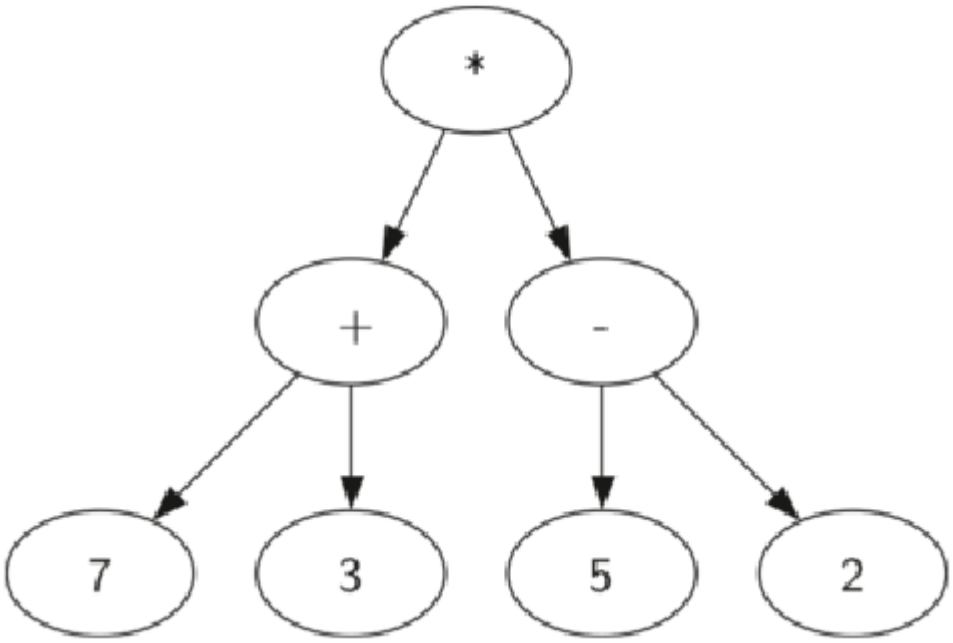


Рисунок 1. Дерево синтаксического разбора выражения (((7+3)*(5-2)))

Выражение может содержать скобки, которые определяют порядок выполнения операций в выражении. Так на рисунке представлено такое дерево.
Благодаря скобкам мы знаем, что здесь сначала нужно вычислить сумму и разность.
Выражение со скобками можно определить рекурсивной зависимостью:
формула::=цифра|(формула операция формула)
Примеры таких формул (3*(2+5)) или (((7+3)*(5-2)))

Может быть выражение, в котором нет скобок, при построении дерева необходимо учитывать порядок выполнения операций в соответствии с их приоритетностью.
Иерархия дерева помогает лучше понять порядок вычисления выражения в целом.
Так в при вычислении значения выражения (((7+3)*(5-2))) перед тем, как найти стоящее на самом верху произведение, требуется произвести сложение и вычитание в поддеревьях.
Первая операция - левое поддерево - даст 10, вторая - правое поддерево - 3.
Используя свойство иерархической структуры, мы можем просто заменить каждое из поддеревьев на узел, содержащий найденный ответ. Эта процедура даст упрощённое дерево, показанное на рисунке 2.

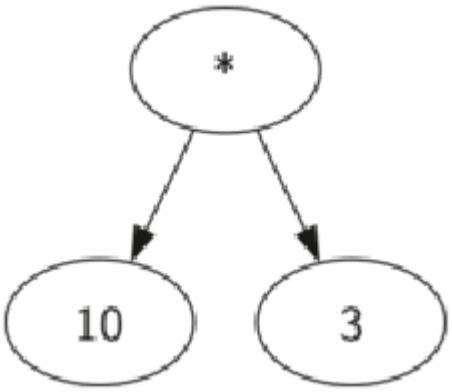


Рисунок 2: Упрощённое дерево синтаксического разбора для (((7+3)*(5-2)))

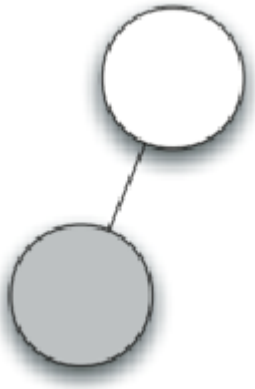
Рассмотрим алгоритмы
• Как построить дерево разбора для математического выражения с полной расстановкой скобок.
• Как вычислить выражение, хранящееся в дереве разбора.
• Как записать оригинальное математическое выражение из дерева разбора.
Первый шаг при построении дерева синтаксического разбора - это разбить строку выражения на список символов (токенов). Их насчитывается четыре вида:
левая скобка, правая скобка, оператор и операнд.
Мы знаем, что прочитанная левая скобка всегда означает начало нового выражения и, следовательно, необходимо создать связанное с ним поддерево. И наоборот, считывание правой скобки сигнализирует о завершении выражения.
Так же известно, что операнды будут листьями и потомками своих операторов.
Наконец, мы знаем, что каждый оператор имеет обоих своих потомков.
Используя эту информацию, определим следующие правила:
1. Если считан токен '(' - добавляем новый узел, как левого потомка текущего, и спускаемся к нему вниз.
2. Если считанный один из элементов списка ['+','-','/','*'], то устанавливаем корневое значение текущего узла равным оператору из этого токена. Добавляем новый узел на место правого потомка текущего и спускаемся вниз по правому поддереву.
3. Если считанный токен - число, то устанавливаем корневое значение равным этой величине и возвращаемся к родителю.
4. Если считан токен ')', то перемещаемся к родителю текущего узла.

Проверим приведённые выше правила на выражении ((3 + (4 * 5))) и разобьём его следующим образом: ('(', '3', '+', '(', '4', '*', '5' ,')',')'). Начинать будем с дерева разбора, содержащего пустой корневой узел.

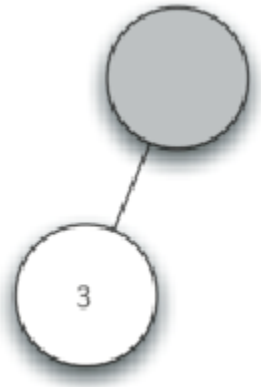
Рисунок 3 демонстрирует структуру дерева и содержимое по мере определения каждого нового токена.

Пустое -корень

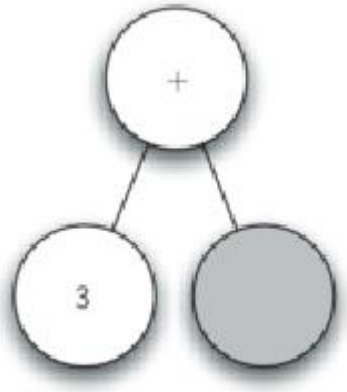
Создаем левое, так как была (



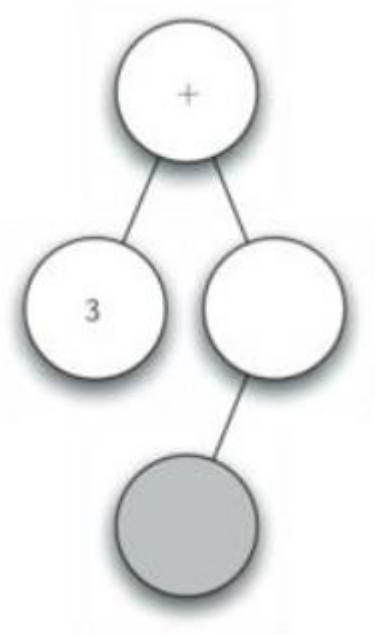
Выбираем из выражения цифру 3 и записываем в узел левого поддерева операнд
3



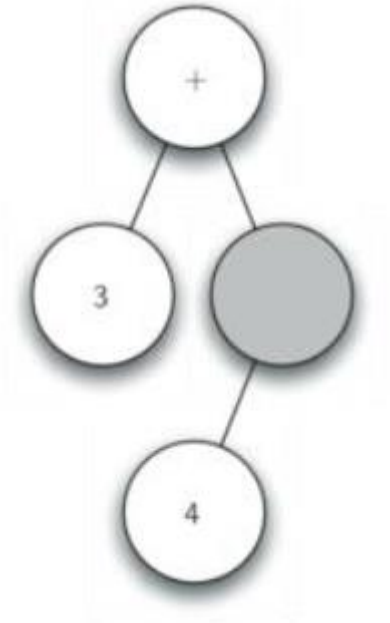
Выбираем из выражения знак операции + и записываем в текущий корневой узел



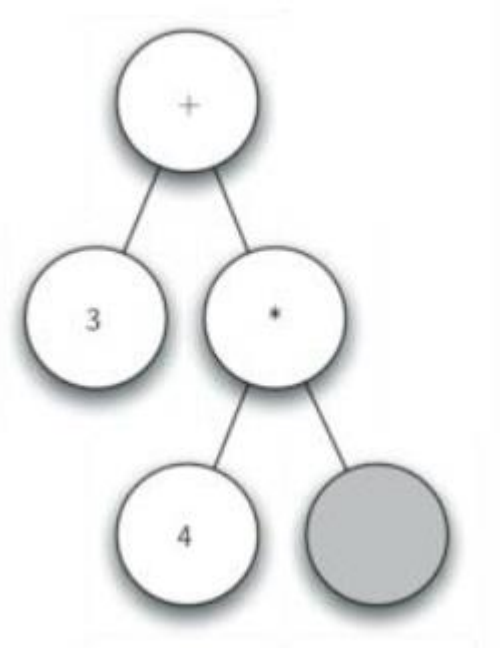
Выбираем из выражения (, это правый операнд операции +) тогда создаем корень правого поддерева пока пустой



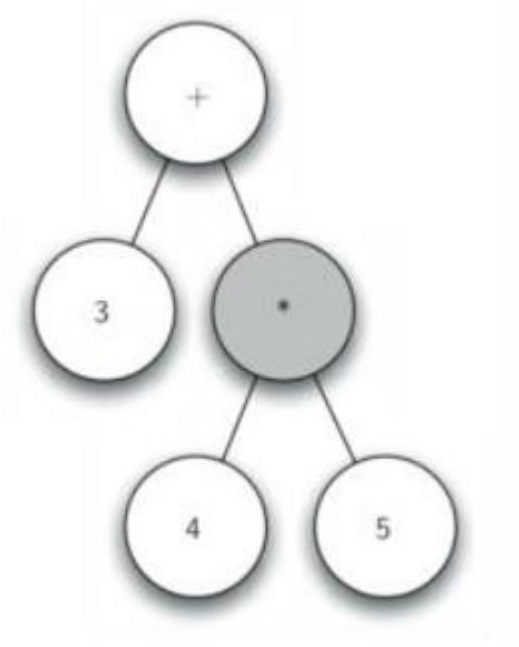
Выбираем из выражения цифру 4, создаем у текущего корня левое поддерево и записываем в него 4



Выбираем из выражения знак операции * и записываем в текущий корневой узел



Выбираем из выражения цифру 5, создаем у текущего корня правое поддерево и записываем в него 5



Выбираем из выражения), разбор выражения в скобке завершен

Рисунок 3 - Трассировка построения дерева разбора. Используя этот рисунок, сформулируем алгоритм формирования дерева выражения со скобками.

а) Создаём пустое дерево.
б) Читаем (в качестве первого токена. По правилу 1 создаём новый узел, как левого потомка корня. Делаем его текущим.

в) Считываем следующий токен - 3. По правилу 3 устанавливаем значение текущего узла в 3 и перемещаемся обратно к его родителю.
г) Следующим считываем +. По правилу 2 устанавливаем корневое значение текущего узла в + и добавляем ему правого потомка. Этот новый узел становится текущим.
д) Считываем (. По правилу 1 создаём для текущего узла левого потомка. Этот новый узел становится текущим.
е) Считываем 4. По правилу 3 устанавливаем значение текущего узла равным 4. Делаем текущим родителя 4.
ж) Считываем следующий токен - *. По правилу 2 устанавливаем корневое значение текущего узла равным * и создаём его правого потомка. Он становится текущим.
з) Считываем 5. По правилу 3 устанавливаем корневое значение текущего узла в 5, после чего текущим становится его родитель.
и) Считываем). По правилу 4 делаем текущим узлом родителя *.
к) Наконец, считываем последний токен -). По правилу 4 мы должны сделать текущим родителя +. Но такого узла не существует, следовательно, мы завершили разбор дерева.

Из примера выше очевидна необходимость отслеживать не только текущий узел, но и его предка. Интерфейс дерева предоставляет нам способы получить потомков заданного узла - с помощью методов getLeftChild и getRightChild, - но как отследить родителя?

Два подхода
Подход 1. При не рекурсивной реализации алгоритма построения дерева выражений
Простым решением для этого станет использование стека в процессе прохода по дереву. Перед тем, как спуститься к потомку узла, последний узел мы кладём в стек. Когда же надо будет вернуть родителя текущего узла, мы вытолкнем из стека нужный элемент.
Используя описанные выше правила совместно с операциями из Stack и BinaryTree, мы готовы написать функцию для создания дерева синтаксического разбора.
Подход 2. При рекурсивной реализации стек будет использован системный, а ссылку на родителя будет хранить узел, т.е. структура узла содержит три указателя: на левое и правое деревья, и на родителя.
При создании левого и правого поддерева будем сохранять ссылку на их родителя.