ISSN 1512-1232

УДК 519.685

ОБ АВТОМАТИЗАЦИИ ОБРАБОТКИ ДАННЫХ, ИМЕЮЩИХ СТРУКТУРУ ДЕРЕВА

Заркуа Теодор[1], Путкарадзе Хатуна[2]

[1]Европейский университет, г.Тбилиси, 0141, пр. Гурамишвили 76
[2]Сухумский государственный университет, г.Тбилиси, 0186, ул. Политковской 61

Аннотация

Статья посвящена попытке распространить на данные, представленные в виде дерева концепции, оправдавшие себя при автоматизации обработки функциональных выражений и опирающиеся на их бесскобочные (польские) формы записи, а также на понятие сопряженного к этим формам, введённого в практику и теорию программирования сравнительно недавно.

Авторы делают вывод о несомненной пользе исследуемого подхода, иллюстрируют этот вывод на конкретных примерах и дают в качестве заключения рекомендации о дальнейших исследованиях в данном направлении.

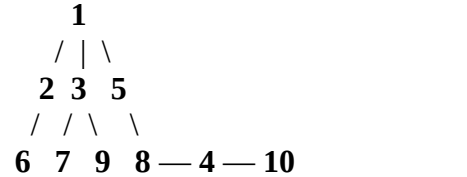
Ключевые слова: *дерево, автоматизация, функциональное выражение, алгоритм, польская запись, префиксная запись, сопряжённая запись, стек, компилятор, параллельное программирование.*

Введение

Как известно, бесскобочные формы записи функциональных вырвжений, введённые польским математиком Яном Лукасевичем (Jan Łukasiewicz) и названные в его честь польскими, создали предпосылки для разработки алгоритмов, легших в основу автоматизации обработки не только функциональных выражений, но и более сложных конструкций языков программирования. Все эти наработки сегодня интенсивно используются в теории и практике конструирования компиляторов для языков программирования. В работе [1] было введено понятие сопряженного по отношению к бесскобочным записям функциональных выражений, которое позволило ещё более упростить процесс обработки этих выражений, так как фактически позволило снять необходимость преобразования прямой польской записи в обратную и наоборот. И постольку поскольку функциональное выражение можно рассматривать как набор данных, организованный в форме дерева, возникла идея попробовать распространить на все данные древовидной структуры концепции, которые так хорошо зарекомендовали себя применительно к функциональным выражениям.

Основная часть

Рассмотрим конкретное дерево, которое в дельнейшем изложении позволит иллюстрировать алгоритмы и понятия, принеминмые к любым деревьям:



Запишем это же дерево в обычно принятой (канонической) форме:

ISSN 1512-1232

((1,{2,3,5}), (2,{6}), (3,{7,9}), (4,{10}), (5,{8}), (6,{}), (7,{}), (8,{4}), (9,{}), (10,{})) .

Как видим, здесь каждая вершина представлена парой, в которой первый элемент является идентификатором вершины (не нарушая общности, можно считать, что это его номер — натуральное число), а второй — представляет собой набор вершин, представляющих собой его соседей-потомков (очевидно, у листьев этот набор - пустой). Заметим, что не имеет никакого значения в какой последовательности идут пары, описывающие вершины дерева.

Мы акцентируемся на структуре данных и поэтому не вставили в информацию о вершине её информационную часть. Однако вполне можно считать, что информационная часть вершин расположена в отдельном ассоцированном массиве и доступна по идентификатору вершины. Соответственно, и в этом разрезе общность рассмотрения вопроса не нарушена.

Префиксной формой записи этого дерева будем считать список пар, представляющих его вершины, в каждой из которых первый элемент является идентификатором вершины, а второй элемент равен числу его соседей-потомков. Последовательность пар в списке должна соответствовать обходу дерева в глубину и слева направо, начиная с его корневой вершины. В нашем случае это будет следующая запись: **((1,3), (2,1), (6,0), (3,2), (7,0), (9,0), (5,1), (8,1), (4,1), (10,0))** .

По аналогии с вышеупомянутым понятием сопряженного применительно к функциональным выражениям, здесь сопряженным к постфиксной записи следует считать запись, полученную выписыванием элементов префиксной формы в обратном порядке — то есть, запись реверсивную по отношению к той, что мы назвали префиксной:

((10,0), (4,1), (8,1), (5,1), (9,0), (7,0), (3,2), (6,0), (2,1), (1,3)) .

Заметим, что термин постфиксная запись мы использовали исключительно для сохранения аналогии с бесскобочными формами записи функциональных выражений. В [1] относительно функциональных выражений сделан вполне однозначный вывод, что буквально все потребности автоматизации обработки полностью удовлетворяются в условиях наличия только префиксной формы. Причём, форму следует обрабатывать слева направо для реализации функциональных преобразований и следует обрабатывать справа налево, если необходимо вычислить значение выражения для конкретных значений переменных. Иногда, вместо обрабоки префиксной формы справа налево мы будем использовать её реверсивную форму, которую будем обрабатывать, естественно, слева направо. Таким образом, дальше термин постфиксная запись мы употреблять не будем и, соответственно, запись соответствующего вида определять и приводить тоже не будем.

Восстановить исходную каноническую запись из формы, реверсивной к префиксной можно следующим алгоритмом:

1. Взять пустой стек и пустой результирущий список.
2. Если в исходном списке нет очередного элемента, перейти к пункту **7**, иначе извлечь из исходного списка очередной элемент (пару).
3. Из пары, полученной в предыдущем пункте извлечь первую и вторую составляющие. Первую назовем идентификатором теущей вершины, а вторую составляющую — кратностью этой вершины.
4. Открыть новый набор (список) данных. Осуществить из стека количество считываний, равное кратности вершины. Каждый считанный элемент занести в открытый выше набор данных.
5. После завершения предыдущего пункта занести в стек идентификатор текущей вершины и добавить в результирующий список спереди пару, состоящую из идентификатора текущей вершины и полученного набора данных.
6. Перейти на пункт 2.
7. Значение результирующего списка считать окончательным и завершить выполнение алгоритма.

ISSN 1512-1232

Следующая таблица позволяет проследить за ходом выполнения описанного алгоритма для вышеуказанного конкретного случая:

Очередная пара на входе	**(10,0)**	**(4,1)**	**(8,1)**	**(5,1)**	**(9,0)**
Состояние стека	**{10}**	**{4}**	**{8}**	**{5}**	**{5,9}**
Состояние результирующего списка	**((10,{}))**	**((4,{10}), (10,{}))**	**((8,{4}), (4,{10}), (10,{}))**	**((5,{8}), (8,{4}) , (4,{10}), (10,{}))**	**((9,{}), (5,{8}), (8,{4}) , (4,{10}), (10,{}))**

Очередная пара на входе	**(7,0)**	**(3,2)**	**(6,0)**	**(2,1)**
Состояние стека	**{5,9,7}**	**{5,3}**	**{5,3,6}**	**{5,3,2}**
Состояние результирующего списка	**((7,{}), (9,{}), (5,{8}), (8,{4}) , (4,{10}), (10,{}))**	**((3,{7,9}), (7,{}), (9,{}), (5,{8}), (8,{4}) , (4,{10}), (10,{}))**	**((6,{}), (3,{7,9}), (7,{}), (9,{}), (5,{8}), (8,{4}) , (4,{10}), (10,{}))**	**((2,{6}), (6,{}), (3,{7,9}), (7,{}), (9,{}), (5,{8}), (8,{4}) , (4,{10}), (10,{}))**

После вышеприведённого, на входе остаётся единственная пара: **(1,3**). В результате обработки этой пары, в итоге получаем список:

((1,{2,3,5}), (2,{6}), (6,{}), (3,{7,9}), (7,{}), (9,{}), (5,{8}), (8,{4}) , (4,{10}), (10,{})) .

Заметим, что в данном списке важно, чтобы корневая вершина была вначале. Взаимное расположение всех остальных вершин никакого значения не имеет. В принципе, в программе вместо списка можем иметь ассоцированный массив, индексы которого соответствуют идентификаторам вершин дерева. В этом случае важно, чтобы меньший индекс соответствовал корневой вершине (либо отдельно иметь переменную, равную идентификатору корневой вершины). Заметим также, что не имеет значения в какой последовательности расположены идентификаторы соседей-потомков во втором элементе пары, описывающей вершину.

Как видим, запись, которая является реверсированной от префиксной, позволила восстановить исходную информацию о структуре дерева на основании довольно простого алгоритма. Совершенно очевидно, что эта форма записи дерева позволяет существенно облегчить большой пласт задач, которые возникают при обработке деревьев. В качестве примера разберём задачу нахождения максимальной длины пути из корневой вершины до листьев дерева.

Разумеется, в этом случае будем подразумевать, что у нас есть дополнительная информация о каждой вершинне, позволяющая определить длину любого ребра дерева. Тогда сразу же замечаем, что за основу вполне можно взять вышеприведенный алгоритм. Просто нам в стеке надо хранить пару, первый элемент которой является идентификатором текущей вершины дерева, а вот вторым элементом — наибольшее расстояние от данной вершины до своих потомков-листьев (естественно, у самих листьев этот показатель будет равен нулю). После этого в упомянутый алгоритм надо будет внести всего одно дополнение

ISSN 1512-1232

— в пункте 4 по мере считывания из стека уже пар, надо будет вычислить максимум среди сумм вторых компонентов этих пар с расстоянием от соответствующей вершины этой пары до текущей вершины и взять его (максимум) в качестве второй компоненты к идентификатору текущей вершины и имено эту пару занести в стек (в начале пункта 5.). Легко заметить, что именно таким путем в конце выполнения алгоритма мы получим не только результирующий список (каноническую форму исходного дерева), но также, в качестве второй компоненты пары, расположенной в вершине стека — ответ на задачу, поставленную выше.

Разумеется, если мы решаем задачу о наибольшем расстоянии до листьев, совсем нет необходимости формировать результирующий список. Но, само собой, держать в стеке наряду с наибольшим расстоянием от текущей вершины до листьев, необходимо также и идентификатор этой вершины. Очевидно задачу о минимальном расстоянии до листьев можно решить совершенно аналогично. И, вообще, идея этого решения подсказывает насколько удобно такое описание дерева при решении задач, связанных с обходом его вершин и ребер.

Для более определенных выводов о целесообразности использования этого формата, рассмотрим алгоритм получения префиксной формы записи дерева из канонического. Получив префиксный формат, мы, тем самым, получим и формат реверсивный к префиксному. Ведь для достижения необходимого эффекта достаточно обработать префиксный формат справа-налево!

Алгоритм для получения префиксного формата из канонического выглядит следующим образом:

1. Взять пустой стек и пустой результирущий список.
2. Извлечь из входного списка первый элемент (пару) и сделать его текущим.
3. Первый элемент текущей пары назовем идентификатором текущей вершины, а второй — набором соседей-потомков этой вершины.
4. Добавить в конец результирующего списка пару, первый элемент которой — идентификатор текущей вершины, а второй — размер набора соседей-потомков этой вершины (количество соседей-потомков).
5. Содержимое набора соседей-потомков по одному полностью записать в стек.
6. Если стек пуст, то перейти к пункту **7**. Если стек не пустой, то извлечь из него один элемент — идентификатор текущей вершины. По идентификатору текущей вершины определить набор её соседей-потомков и перейти к пункту 4.
7. Полученное значение результирующего списка посчитать окончательным результатом и завершить выполнение алгоритма.

Как видим, обе производные формы записи дерева, рассмотренные нами в данной статье получаются из канонической записи алгоритмом, быстрота выполнения которого поддаётся оценке **O(N+M)**, где **N** и **M**, соответственно, количество вершин и рёбер.

Заключение

Как видим, префиксная форма записи дерева получается из канонической очень несложным образом. С другой стороны, эта форма позволяет существенно облегчить решение целого ряда задач над деревьями, так как очень хорошо отражает суть структуры дерева. Очевидно, кроме конкретного применения описанного подхода для обработки данных, заданных деревом, имеет смысл изучить целесообразность данного подхода, вообще, к данным, заданным графами. Кроме того, следует ожидать интересных результатов применительно к параллельному программированию. Ведь граф, описывающий последовательность действий в параллельных алгоритмах, представляет собой дерево!

ISSN 1512-1232

Список литературы:

1. Т.Заркуа.Автоматизация обработки функциональных выражений на уровне алгоритмических языков. Internet-Edication-Sciense-2010. New Informational and Computer Tecnologies in Education and Sciense.p.201-206.

Article received: 2022-02-06