Павлов Дмитрий Алексеевич

РЕВЕРСИВНЫЙ ПОИСК И ПЕРЕБОР ПОДДЕРЕВЬЕВ ГРАФА

Аннотация

Реверсивный поиск – один из алгоритмов, идея которого «витает в воздухе» десятилетиями и который часто изобретается заново программистами, знакомыми с такими алгоритмами, как поиск в ширину (BFS) и поиск в глубину (DFS). В 1996 г. алгоритму было дано имя и довольно общая формулировка [1], с которой его можно применить ко множеству различных задач. В статье показано ещё одно применение этого алгоритма – перебор поддеревьев произвольного графа.

Ключевые слова: реверсивный поиск, перебор подграфов, перебор поддеревьев.

ВВЕДЕНИЕ

Большинство комбинаторных и оптимизационных проблем содержат в себе задачу обхода вершин некоего связного графа. Граф, как правило, имеет большой размер и не задан явно, иначе перебор его вершин не составил бы труда. Известна лишь какая-то начальная вершина; также для каждой полученной в процессе перебора вершины можно узнать список смежных с ней вершин. Задача проста: посетить все вершины графа по одному разу.

Существуют два классических алгоритма для решения этой задачи: поиск в глубину (*depth-first search*) и поиск в ширину (*breadth-first search*). Оба алгоритма требуют, кроме памяти для хранения «фронта» (стека или очереди) вершин, которые будут посещены, дополнительной памяти для запоминания тех вершин, которые уже посещены.

АЛГОРИТМ РЕВЕРСИВНОГО ПОИСКА

При некотором условии можно написать рекурсивный алгоритм обхода, не требующий запоминания посещённых вершин. Алгоритм был предложен в 1996 году и называется «реверсивный поиск» (*reverse search*) [1]. Для его работы требуется, чтобы для каждой вершины v графа была определена соседняя вершина $f(v)$, имеющая максимальный «приоритет». Функция f должна быть задана так, чтобы любой путь вида

$$v \to f(v) \to f((f(v)) \to \ldots$$

заканчивался, причем обязательно в одной и той же вершине v^*. Для определённости положим $f(v^*) = v^*$.

Простейший пример: пронумеруем вершины графа натуральными числами ($p(v)$) и определим $f(v)$ как вершину, имеющую максимальный номер среди множества $nei(v)$ всех соседей v. Конечно, не любая нумерация будет соответствовать выщеуказанному критерию. На множестве V вершин графа функция p должна иметь единственный локальный максимум, он же и глобальный (рис. 1).

Рассмотрим граф T, множество вершин которого совпадает с V, а ребро между u и w есть, когда $u = f(w)$ или $w = f(u)$. Несложно показать, что этот граф будет деревом. На рисунке рёбра дерева обозначены стрелками, ведущими от v к $f(v)$, по возрастанию номера вершины.

Собственно алгоритм реверсивного поиска очень прост (см. алгоритм 1).

Шаги 1 и 2 предназначены для обнаружения корня дерева, а шаги 3 и 4 обходят это дерево в глубину. Обойдя все вершины дерева, мы тем самым обойдём все верши-

ны исходного графа. Обход дерева делается в направлении, противоположном возрастанию функции p, отсюда название алгоритма: реверсивный, то есть обратный поиск.

После этого примера может создаться впечатление, что алгоритм вообще не имеет смысла, так как все заботы, связанные с отбрасыванием уже посещённых вершин, переложены на функцию f, и посчитать её не проще, чем обойти все вершины графа. Это не так: на многих графах, заданных неявно с использованием терминов какой-либо предметной области, функция f тривиально формулируется в тех же терминах. Авторы в своей статье приводят алгоритмы для перебора:

– триангуляций точек на плоскости,

– остовных деревьев графа,

– остовных деревьев графа на плоскости с непересекающимися рёбрами,

– связных индуцированных подграфов графа,

– вершин n-мерного выпуклого многогранника, заданного гиперплоскостями граней,

– других объектов.

(Кроме того, рассматривается модификация алгоритма, допускающая наличие нескольких «локальных максимумов» в графе поиска).

ПЕРЕБОР ПОДДЕРЕВЬЕВ ГРАФА

Построим алгоритм перебора всех поддеревьев произвольного графа G, основанный на реверсивном поиске. Множество вершин V графа поиска соответствует множеству поддеревьев G. Мы будем считать, что между вершинами u и w есть ребро, если u получается из w добавлением или

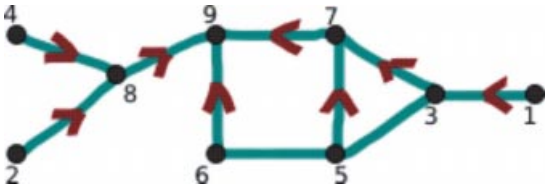

Рис. 1

удалением одного ребра из графа G. Допустим, что рёбра графа G как-то упорядочены. Будем считать, что $u = f(w)$, если u получается из w удалением максимального ребра (не считая тех рёбер, после удаления которых получается два дерева вместо одного). Нетрудно понять, что вершина v^* в таком случае будет соответствовать пустому подграфу G.

Построим, например, граф поиска всех поддеревьев графа из трёх вершин (рис. 2).

Каждый узел графа поиска соответствует одному из поддеревьев нашего графа; пустое поддерево не является исключением (рис. 3).

Направления стрелок соответствуют удалению рёбер с максимальным номером в исходном графе. Поддеревья из одной вершины не учитываются; можно сказать, что мы рассматриваем «рёберные» поддеревья.

Шаги 1 и 2 алгоритма перебора поддеревьев соответствуют шагам 3 и 4 общего алгоритма реверсивного поиска, так как известно, что начальное поддерево – пустое, и искать его не надо. Кроме того, держать поддеревья на стеке не требуется: можно вместо поддерева класть на стек рёбра исходного графа. Каждое ребро кладётся на стек дважды: первый раз – для добавления в поддерево, второй раз – для удаления из него (см. алгоритм 2).

Алгоритм 1

1. Выбрать произвольную вершину v
2. Если $v \neq f(v)$, то присвоить $v \leftarrow f(v)$ и перейти к шагу 1
3. Положить на стек вершину v
4. Пока стек не пуст:
    1. Снять со стека вершину u
    2. «Посетить» u
    3. Для каждой вершины $w \in nei(u)$:
        - Если $f(w) = u$, то положить w на стек

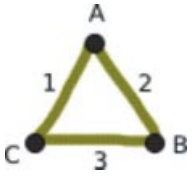

Рис. 2

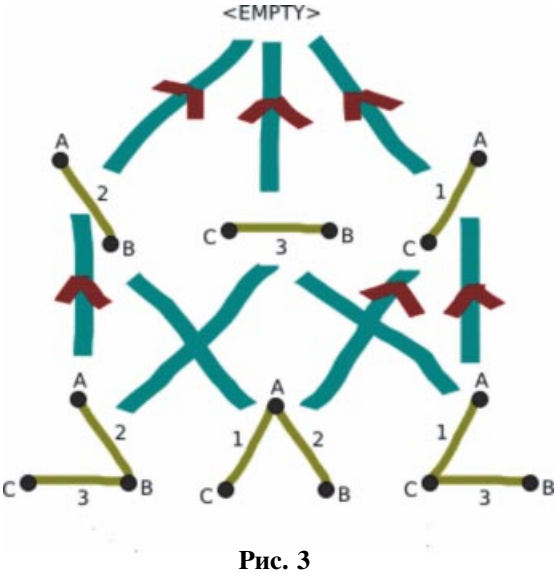

Рис. 3

Поддерево S' получается добавлением ребра h к S. «Соседи» S' в графе поиска – это поддеревья, полученные удалением одного ребра из S'. Поскольку соседи также должны являться деревьями, то из S' можно удалять только «висячие» рёбра, то есть такие, которые имеют вершину со степенью 1. $S = f(S')$, если h является максимальным из всех таких рёбер. Можно сказать, что h должно быть максимальным ребром из всех, которые могли быть добавлены в S' на последнем шаге. В этом и только в этом случае мы переходим к поддереву S'. Это даёт гарантию, что ни одно поддерево не будет посещено дважды. При обходе в ширину или в глубину избежать повторений так просто бы не получилось.

Алгоритм 2

1. Для каждого ребра e графа G
   - Положить e на стек
2. $S \leftarrow$ пустой граф
3. Пока стек не пуст:
   - Снять ребро j со стека
   - Если $j \in S$:
     - Удалить j из S
   - Иначе:
     - Добавить j в S
     - «Посетить» S
     - Положить j на стек
     - Для каждого ребра h графа G, соседнего с одним из рёбер S и не принадлежащего S:
       - $S' \leftarrow S + h$
       - Если S' – дерево и если h – максимальное ребро S', содержащее вершину степени 1, то положить h на стек

Литература

1. *David Avis, Komei Fukuda.* Reverse search for enumeration. Discrete applied mathematics 65, pp. 21–46, 1996.

Abstract

The idea of reverse search was in the air for decades, and lot of programmers were reinventing it without giving it a name, while the similar algorithms such as breadth-first search and depth-first search were widely known. In 1996 the general algorithm was presented in [1], which was shown to be suitable for variety of problems. In the present paper another application of this algorithm is shown: enumeration of the subtrees of an arbitrary graph..

Павлов Дмитрий Алексеевич,
аспирант кафедры прикладной математики ФМФ СПбГПУ,

dmitry.pavlov@gmail.com