А. И. Свито

ПЕРСИСТЕНТНЫЕ СТРУКТУРЫ ДАННЫХ

Введение

Для решения определённого класса геометрических задач, а также при разработке некоторых прикладных инструментов, возникла необходимость в структурах, которые могут запоминать свои предыдущие состояния и, при необходимости, возвращаться к ним. Хорошими примерами использования таких структур являются сервисы геопозиционирования, система управления версиями (git) или microsoft office online, где в процессе создания документа мы можем вернуться к любой его версии. Впервые структуры данных, поддерживающих такую функциональность, были представлены Driscoll, Sarnak, Sleator, и Tarjans' в статье 1986 года. В той же статье им было дано название — Персистентные структуры данных (англ. Persistent data structures) [1]. Персистентная структура данных - это структура, сохраняющая свои предыдущие состояния при каждой модификации, таким образом, обеспечивая возможность работы с её состоянием в любой временной отрезок. Другими словами, персистентные структуры данных позволяют осуществлять своеобразные "путешествия во времени". Из-за того, что при каждой модификации структура сохраняет свою прошлую версию, мы можем работать не только с "настоящим но и с данными из прошлых состояний структуры, т. е. её "прошлым". Чтобы различать одну и ту же структуру из разных временных промежутков, каждое её состояние идентифицируется особым образом (в виде числа, вектора чисел или хеша), данный идентификатор для персистентных структур называется ее версией.

I. Типы персистентности

В зависимости от функциональности, предоставляемой такими структурами, их классифицируют на персистентные и ретроактивные. Основное различие между ними в том, что при модификации персистентной структуры в ее "прошлом"состоянии сохраняется исходная цепь модификаций, а от старой версии структуры ответвляется новая. При этом версия структуры в "настоящем"остаётся неизменной. Другими словами, изменения прошлых версий структуры создают новые "временные линии оставляя за пользователем возможность получать доступ к старым, как видно на рис.1.

Ретроактивные структуры данных работают иначе. При внесении изменений в прошлое состояние, вся "временная линия"просчитывается заново, и состояние структуры в "настоящем"меняется.

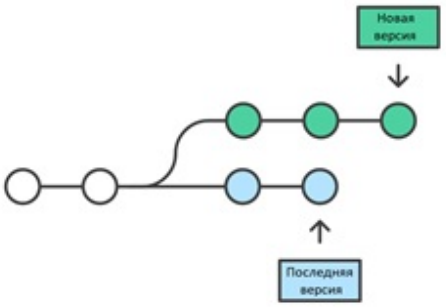

Рис. 1 – Граф версий для полностью персистентных структур, представляющий из себя дерево

Сами же персистентные структуры данных подразделяются на:

- Частично персистентные (можно редактировать только последнюю версию структуры);
- Полностью персистентные (можно редактировать любую версию структуры в любой временной ветке);
- Конфлюэнтные (можно сливать временные ветки в одну);
- Функциональные.

Разница между частично персистентными и полностью персистентными структурами в том, что в частично персистентных мы можем модифицировать лишь текущую версию структуры, в её "настоящем а в полностью персистентных структурах мы можем модифицировать любую версию структуры в любой временной линии — новая версия прошлого состояния просто создаст новую ветвь в графе версий (рис. 1). Цепь изменений частично персистентной структуры представляет из себя обычную линию. Тут версиями структуры могут являются обычные натуральные числа.

II. Реализация персистентных структура

Существует три основных подхода к тому, чтобы сделать структуру персистентной: полное копирование, копирование пути, «толстые» узлы. Самый простой из них - это полное копирование. При любом изменении мы делаем полную копию прошлого состояния структуры, и оставляем указатель на неё в таблице версий. Указатель на текущее состояние мы передвигаем на новую версию. Такой подход страдает от вполне очевидного ряда недостатков, основными из которых выступают огромное потребление мощностей CPU и памяти. К тому же, затраты памяти

бессмысленны, так как, как правило, при модификации какого-то конкретного элемента большинство элементов в структуре остаются неизменными, но, тем не менее, дублируются и сохраняются в памяти. Второй подход – это копирование пути, сокращает количество копируемых элементов. Суть подхода в том, что копируется не вся структура, а лишь те её элементы, которые являются сильно связанными с модифицируемым, то есть, являющиеся звеньями между указателями на структуру и на модифицируемый элемент. Например, для бинарного дерева поиска, мы будем копировать все родительские элементы по отношению к модифицируемому, как видно на рис.2.

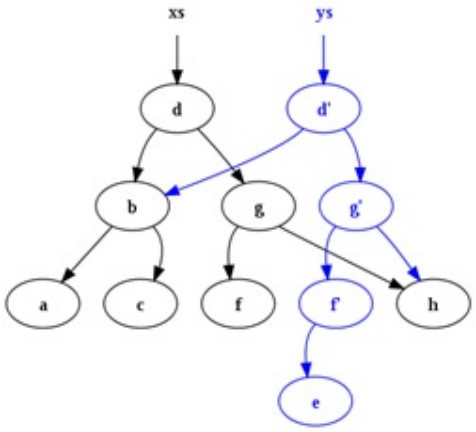

Рис. 2 – При копировании путей дублируются только родительские узлы (d, g, f)

Данный подход позволяет сэкономить значительное количество памяти. Однако можно заметить, что часть не модифицированных компонентов всё равно дублируется. Третий подход – метод «толстых» узлов, сводит издержки памяти к минимуму, однако он применим не ко всем структурам данных, а лишь к тем, которые можно представить в виде "машины указателей"(англ. Pointer machine). Машина указателей - структура данных с выполняющимися тремя условиями:

- Вся структура состоит только из указателей и узлов с данными.
- Все узлы структуры фиксированного размера, то есть состоят из константного количества полей.
- Для каждого указателя есть обратный указатель.

При выполнении данных условий, мы можем преобразовать структуру к виду, когда в каждом узле структуры хранится «лог изменений». То есть, при каждой модификации структуры, вместо того чтобы реально перезаписывать значение поля в «толстом» узле, мы вносим новую запись в «лог» узла, в этой записи содержится: версия, когда было произведена модификация и новое значение. При обращении к конкретной версии структуры, мы ищем в узле подходящую версию и возвращаем значение из «лога». При переполнении «лога» узла, создаётся новый узел, текущим значением которого является значение максимальной версии старого узла. Несмотря на то, что при переполнении узлов в худшем случае мы должны будем обойти всю структуру, при подавляющем большинстве модификаций все операции над узлом будут происходить за O(1). Таким образом, нетрудно доказать, что амортизированная сложность любых операция над «толстыми» узлами будет O(1).

Выводы

Персистентные структуры данных — довольно новая и неизученная тема, поэтому существует ещё огромное количество открытых вопросов и теорий, на которые ещё не дан ответ или которые не было доказаны. Несмотря на относительную сложность, подобные структуры активно применяются при написании геометрических алгоритмов, а также при построении различного рода самовосстанавливающихся систем, ведь они могут сделать «дешёвую» копию самой себя. В любом случае, персистетные и ретроактивные структуры данных и алгоритмы для работы с ними определённо стоят более подробного изучения.

1. J.R. Driscoll, N.Sarnak, D.D.Sleator, R.E.Tarjan (1986). "Making data structures persistent". Продолжение STOC '86. Материалы восемнадцатого ежегодного симпозиума ACM по теории вычислений. Страницы 109-121.
2. David Karger, Persistent Data Structures // Лекция массачусетского технологического института, 9 сентября 2005
3. Дополнительные главы алгоритмов. Лекции Андрея Станкевича – https://www.lektorium.tv/lecture/14321
4. Персистентные структуры данных. Лекции Павла Маврина –http://logic.pdmi.ras.ru/csclub/node/2734/

Свито Александр Игоревич, студент 3-го курса факультета компьютерных систем и сетей Белорусского государственного университета информатики и радиоэлектроники, alexandervirk@gmail.com.

Научный руководитель: Свито Игорь Леонтьевич, доцент кафедры теоретических основ электротехники Белорусского государственного университета информатики и радиоэлектроники, кандидат технических наук, svito@bsuir.by