Мир, — 1990 — С. 255.

Персистентные структуры данных и их эффективная реализация

Соколов Борис Сергеевич
Крымский федеральный университет имени В.И. Вернадского
Таврическая академия
факультет математики и информатики
кафедра информатики (группа 601-И)
e-mail: sokolov031193@gmail.com

Персистенные структуры данных (англ. *persistent data structure*) – это структуры данных, которые при внесении в них каких-то изменений сохраняют все свои предыдущие состояния и доступ к этим состояниям.

Уровни персистентности

Есть несколько уровней персистентности:

- частичная (англ. *partial*),
- полная (англ. *full*),
- конфлюэнтная (англ. *confluent*),
- фунциональная (англ. *functional*).

В частично персистентных структурах данных к каждой версии можно делать запросы, но изменять можно только последнюю версию структуры данных.

В полностью персистентных структурах данных можно менять не только последнюю, но и любую версию структур данных, также к любой версии можно делать запросы.

Конфлюэнтные структуры данных позволяют объединять две структуры данных в одну (деревья поиска, которые можно сливать).

Функциональные структуры данных полностью персистентны по определению, так как в них запрещаются уничтожающие присваивания, т.е. любой переменной значение может быть присвоено только один раз и изменять значения переменных нельзя. Если структура данных функциональна, то она и конфлюэнтна, если конфлюэнтна, то и полностью персистентна, если полностью персистентна, то и частично персистентна. Однако бывают структуры данных не функциональные, но конфлюэнтные.

Способы преобразования структур данных в персистентные

Есть несколько способов сделать любую структуру персистентной:

- полное копирование (англ. *full copy*) когда при любой операции изменения полностью копируется структура данных и в получившуюся новую копию вносятся изменения,
- копирование пути (англ. *path copying*),
- метод «толстых» узлов (англ. *fat node*).

Рассмотрим для начала частичную персистентность. История изменений структуры данных линейна, в любой момент времени можно обратиться к любой версии структуры данных, но поменять возможно только последнюю версию. В нашем понимании структурой данных будет называться набор узлов, в которых хранятся какие-то данные, и эти узлы связаны ссылками.

Метод копирования пути

Пусть есть сбалансированное дерево поиска. Все операции в нем делаются за $O(h)$, где h — высота дерева, а высота дерева $O(\log n)$, где n — количество вершин. Пусть необходимо сделать какое-то обновление в этом сбалансированном дереве, например, добавить очередной элемент, но при этом нужно не потерять старое дерево. Возьмем узел, в который нужно добавить нового ребенка. Вместо того чтобы добавлять нового ребенка, скопируем этот узел, к копии добавим нового ребенка, также скопируем все узлы вплоть до корня, из которых достижим первый скопированный узел вместе со всеми указателями.

Все вершины, из которых измененный узел не достижим, мы не трогаем. Количество новых узлов всегда будет порядка логарифма. В результате имеем доступ к обоим версиям дерева.

Так как рассматривается сбалансированное дерево поиска, то поднимая вершину вверх при балансировке, нужно делать копии всех вершин, участвующих во вращениях, у которых изменились ссылки на детей. Таких всегда не более трех, поэтому ассимптотика $O(\log n)$ не пострадает. Когда балансировка закончится, нужно дойти вверх до корня, делая копии вершин на пути.

Этот метод хорошо работает на стеке, двоичных (декартовых, красно-черных) деревьях. Но в случае преобразования очереди в персистентную операция добавления будет очень дорогой, так как элемент добавляется в хвост очереди, который достижим из всех остальных элементов. Также не выгодно применять этот метод и в случае, когда в структуре данных имеются ссылки на родителя.

Метод «толстых» узлов

Пусть в структуре данных есть узел, в котором нужно сделать изменения (например, на рисунке ниже в первой версии структуры данных в узле X есть поле $a = 3$, а во второй версии это поле должно быть равно 4), но при этом нужно сохранить доступ к старой версии узла X и не нужно экономить время. В таком случае можно хранить обе версии узла X в большом комбинированном узле.

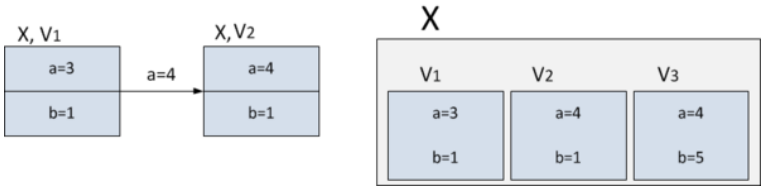

Рис. 1

В примере выше в этом «толстом» узле будет храниться первая версия V_1, у которой $a = 3$ и вторая версия V_2, у которой $a = 4$. Если далее последуют еще какие-то изменения (например, поле b узла X станет равно 5) добавим в толстый узел X еще одну версию – V_3.

Пусть нужно сделать запрос ко второй версии структуры данных (на рисунке выше это запрос $X.a-?$). Чтобы сделать этот запрос, нужно зайти в

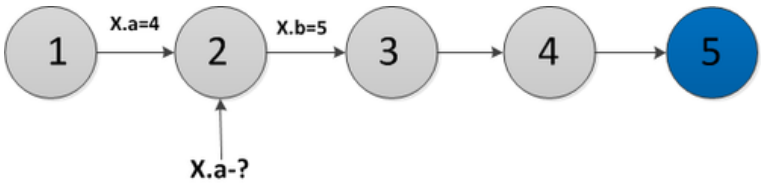

Рис. 2

узел X и найти в списке версий максимальную версию, которая меньше или равна версии запроса (в примере на рисунке это версия 2), и в этой версии узла найти значение поля a (в примере $a = 4$). Чтобы быстро найти нужную версию в списке версий, хранящихся в «толстом» узле, нужно хранить их в виде дерева. Тогда мы сможем за логарифм найти нужную версию и к ней обратиться. Значит, все операции, которые будут производиться на этой структуре данных, будут домножаться на логарифм от числа версий.

Структура толстого узла может быть и другой: к каждой вершине можно хранить лог ее изменений, в который записывается версия, в которой произошло изменение, а также само изменение. Такая структура толстого узла рассмотрена ниже, в разделах об общих методах получения частично и полностью персистентных структур данных. Лог может быть организован поразному. Обычно делают отдельный лог для каждого поля вершины. Когда что-то меняется в вершине, то в лог соответствующего поля записывается это изменение и номер версии, с которой данное изменение произошло. Когда нужно обратиться к старой версии, то двоичным поиском ищут в логе последнее изменение до этой версии и находят нужное значение. Метод fat node дает замедление $\log t$, где t — число изменений структуры данных; памяти требуется $n + t$, где n — число вершин в структуре данных.

Список литературы

[1] Дополнительные главы алгоритмов. Лекции Андрея Станкевича – https://www.lektorium.tv/lecture/14321

[2] Персистентные структуры данных. Лекции Павла Маврина – http://logic.pdmi.ras.ru/csclub/node/2734/