# Обобщенная реализация укорененных Rake-and-Compress деревьев

У Цзюньфэн, магистрант кафедры компьютерных технологий Университета ИТМО, fengone7@gmail.com

Буздалов М. В., к.т.н., доцент кафедры компьютерных технологий Университета ИТМО, mbuzdalov@gmail.com

## Аннотация

Описывается реализация укорененных (имеющих выделенный корень) Rake-and-Compress деревьев, поддерживающая пакетные операции обновления, в виде контейнера на языке C++. Отличие от существующих реализаций таких деревьев, известных авторам, состоит в том, что одновременно реализовано три особенности. Во-первых, дерево является контейнером с произвольными типами пометок на вершинах и ребрах, соответственно поддерживаются запросы на сумму реберных пометок на пути между двумя вершинами и на сумму вершинных пометок в поддереве одной вершины. Во-вторых, снято ограничение на максимальную степень вершины. В-третьих, структура данных занимает $\Theta(n)$ памяти, где $n$ — число вершин в дереве. Приведено время работы построения и выполнения запросов к дереву согласно различным шаблонам. Результаты демонстрируют среднюю сложность выполнения одной операции, равную $O(\log n)$.

## Введение

Структуры данных для поддержания динамических деревьев имеют прямое приложение к эффективному решению многих практическим задач, таких как поиск максимального потока [2]. Такие структуры данных должны поддерживать лес деревьев, операции обновления (добавление и удаление ребер, а в случае укорененных деревьев — подвешивание вершины к вершине и отцепление вершины). Также, как правило, на вершинах, ребрах, или и тех, и других имеются пометки. Структуры данных также должны уметь отвечать на запросы вида «достижима ли вершина $A$ из вершины $B$ по ребрам деревьев», «найти длину пути из вершины $A$ в вершину $B$», «найти сумму пометок на пути из вершины $A$ в вершину $B$», «найти сумму пометок в поддереве

вершины $A$» и другие. С практической точки зрения, полезно также уметь строить такие структуры данных параллельно, а также параллельно применять совокупности, или пакеты, операций обновления.

Среди множества структур данных для динамических деревьев, таких как Link-Cut деревья [4], структура данных Rake-and-Compress Tree [3] — одна из наиболее подходящих структур данных, теоретически поддерживающая параллельное построение и параллельное обновление (включая не только добавление и удаление листьев дерева, но и произвольные подвешивания и отцепления вершин), а также запросы как на путях, так и на поддеревьях.

Идея Rake-and-Compress деревьев, в изложении для укорененных деревьев, заключается в следующем. Для дерева последовательно строится несколько «уменьшенных» копий дерева, причем $i$-тая копия получается из $(i-1)$-ой применением ко всем допустимым вершинам одной из следующих операций:

- Rake. Вершина, имеющая родителя и не имеющая детей, удаляются вместе с ребром, ведущим к родителю (рис. 1).
- Compress. Вершина, имеющая родителя и ровно одного ребенка, удаляется вместе со смежными ребрами, а вместо этого ребро проводится от ребенка удаляемой вершины к родителю этой вершины (рис. 2). При этом родитель не должен подвергаться операции Compress, а ребенок не должен подвергаться ни операции Compress, ни операции Rake.

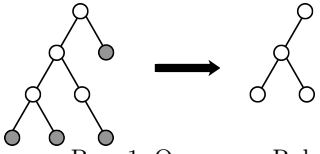

Рис. 1: Операция Rake

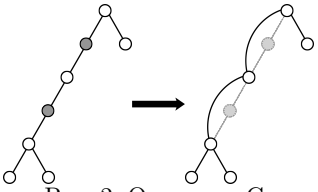

Рис. 2: Операция Compress

С каждым раундом число вершин в дереве, если оно больше единицы, уменьшается как минимум в $8/7$ раз в каждой компоненте связности. В конце работы для каждой компоненты связности остается ровно одна вершина, для чего требуется произвести $O(\log n)$ раундов (рис. 3).

Такое многоуровневое представление дерева позволяет отвечать на запросы на путях и поддеревьях за время, пропорциональное числу раундов (а следовательно, $O(\log n)$), даже если дерево изначально было

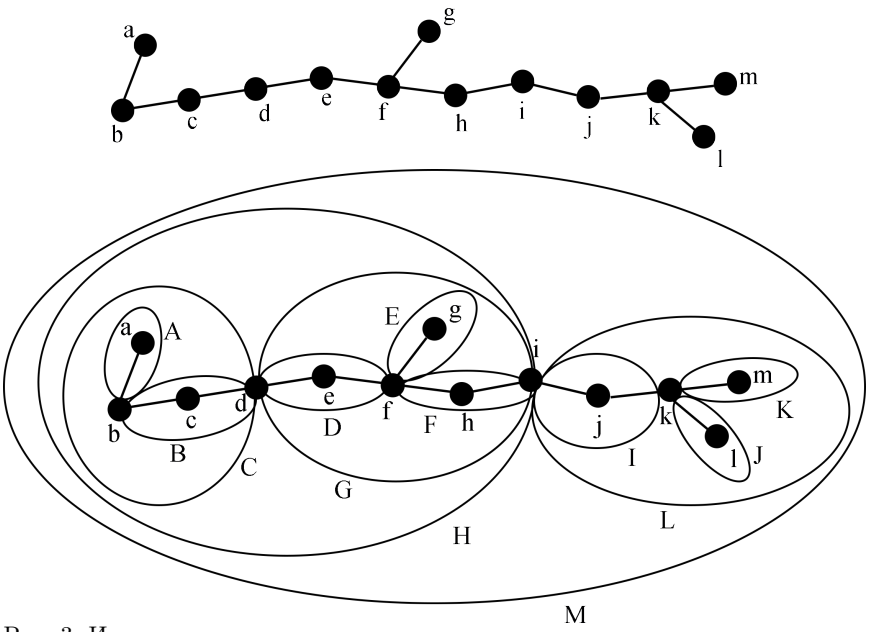

Рис. 3: Исходное дерево и группировка вершин, полученная последовательным применением операций Rake и Compress

несбалансировано (например, существенно вытянутое). В то же время решение о выполнении операции Rake или Compress можно сделать на основе локальной информации (о самой вершине и ее ближайших соседях), что позволяет эффективно распараллеливать операции в рамках каждого раунда. Эффективное применение операций обновления также возможно с использованием этой структуры данных [1].

## Описание реализации

Основной проблемой имеющихся реализаций является константное ограничение на максимальную степень вершины [1]. В настоящей работе его предлагается преодолевать с помощью следующего приема. Каждая вершина оригинального дерева разбивается на две вершины — *вершину данных* (или «нижнюю» вершину) и *вершину связей* (или «верхнюю» вершину). Вершина данных всегда имеет родителем соответствующую вершину связей. При этом все дети той или иной вершины организуются в двоичное дерево поиска, корнем которого служит вершина

данных родителя, а остальными вершинами — вершины связей детей. В качестве двоичного дерева поиска использовалось декартово дерево [5]. При таком подходе степень каждой вершины не превосходит трех, что позволяет использовать алгоритмы для вершин с константной степенью. На рис. 4 приведено оригинальное дерево и дерево, получающееся путем удвоения вершин и организации детей каждой вершины в виде дерева.

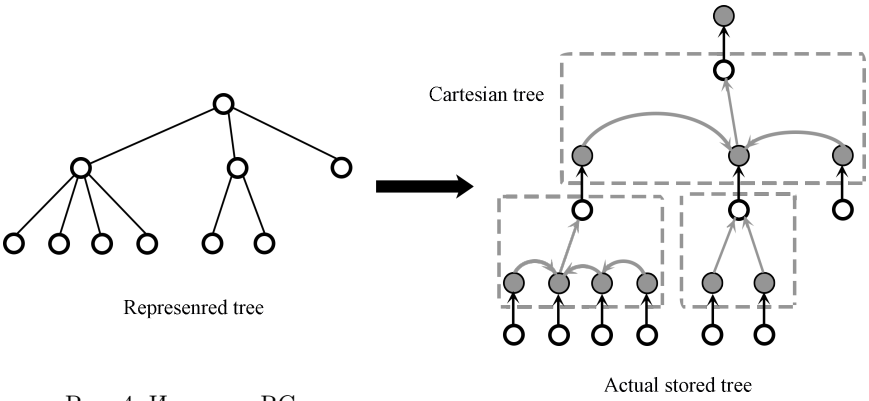


Рис. 4: Исходное RC-дерево и дерево, которое хранится в памяти

Каждая вершина оригинального дерева снабжена меткой, тип которой является коммутативным моноидом, при этом структура данных поддерживает операцию «вычислить моноидную сумму меток всех вершин в поддереве данной вершины». Аналогично, каждое ребро оригинального дерева снабжено двумя метками (метка, соответствующая направлению от ребенка к родителю, или «вверх», и аналогичная метка «вниз»), тип которых совпадает и является моноидом (возможно, некоммутативным), при этом структура данных поддерживает операцию «вычислить моноидную сумма меток всех ребер на пути от вершины $A$ к вершине $B$ для данных вершин $A$ и $B$». При удвоении вершин, метки вершин и ребер, выходящих вверх из вершин оригинального дерева, переходят в соответствующие метки соответствущих вершин данных, в то время как метки вершин и выходящих вверх ребер у вершин связей равны нейтральным элементам соответствующих моноидов. Таким способом достигается корректность выполнения запросов независимо от структуры деревьев, организующих детей вершин.

Исходный код реализации дерева, а также нижеследующих экспериментов, располагается в репозитории GitHub по адресу `https://github.com/feng7/pasl/tree/pdt/example/rc-tree`.

## Эксперименты

Были выполнены эксперименты по измерению времени построения разработанной реализации и выполнения ею запросов на следующих тестах:

1. дерево-«палка»: $n$ вершин, при этом $i$-тая вершина является ребенком $(i-1)$-ой вершины при $i > 0$.

2. дерево-«пучок»: $n$ вершин, при этом $i$-тая вершина является ребенком вершины 0 при $i > 0$.

3. дерево-«два пучка»: $n$ вершин при $n = 2m$, $m$ целое, при этом вершина 0 является родителем вершин с 1 по $(m-1)$, вершина $m$ является родителем вершин с $(m+1)$ по $(n-1)$, а вершины 0 и $m$ дополнительно соединены.

4. построение дерева-«палки» в десять этапов, каждый из которых состоит в присоединении $n/10$ вершин.

После каждого построения дерева также проводилось $n$ запросов к поддереву и $n$ запросов на пути. В табл. 1 приведены времена работы структуры данных на тестах, в секундах, для различных $n$. По результатам можно сделать вывод, что средняя сложность одной операции составляет $O(\log n)$.

Таблица 1: Результаты экспериментов

| Тест | $n = 100$ | $n = 10^3$ | $n = 10^4$ | $n = 10^5$ | $n = 10^6$ |
|---|---|---|---|---|---|
| 1 | $4.3 \cdot 10^{-4}$ | $4.6 \cdot 10^{-3}$ | $8.0 \cdot 10^{-2}$ | $1.4 \cdot 10^{0}$ | $1.4 \cdot 10^{1}$ |
| 2 | $1.4 \cdot 10^{-4}$ | $1.0 \cdot 10^{-3}$ | $1.3 \cdot 10^{-2}$ | $2.4 \cdot 10^{-1}$ | $2.6 \cdot 10^{0}$ |
| 3 | $1.6 \cdot 10^{-4}$ | $1.3 \cdot 10^{-3}$ | $1.3 \cdot 10^{-2}$ | $2.0 \cdot 10^{-1}$ | $2.8 \cdot 10^{0}$ |
| 4 | $3.4 \cdot 10^{-3}$ | $9.7 \cdot 10^{-3}$ | $1.5 \cdot 10^{-1}$ | $3.2 \cdot 10^{1}$ | $5.3 \cdot 10^{1}$ |

## Заключение

В работе продемонстрирована реализация укорененных Rake-and-Compress деревьев, обобщенная по типу меток на вершинах и ребрах и моноидным операциям с ними, поддерживающая пакетные операции обновления произвольного вида (прикрепление вершины, отцепление вершины, изменение свойства). В реализации снято ограничение на максимальную степень вершины, присутствующее в предшествующих работах. Асимптотическая эффективность реализации продемонстрирована на нескольких сценариях работы.

Следующим шагом работы является модификация описанной реализации, поддерживающая параллельное выполнение операций обновления. Основной проблемой видится корректная работа с памятью, в частности с выделением памяти разными потоками, а также параллельная работа со структурой данных, описывающей измененные вершины. Авторы считают указанные проблемы преодолимыми и ожидают, что в ближайшее время описанная реализация будет успешно модифицирована надлежащим образом.

## Литература


[1] U. Acar, G. Blelloch, R. Harper, J. Vittes, and S. Woo. Dynamizing static algorithms with applications to dynamic trees and history independence. In *Proceedings of SODA*, pages 531–540, 2004.

[2] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

[3] J. Reif and S. Tate. Dynamic parallel tree contraction. In *Proceedings of SPAA*, pages 114–121, 1994.

[4] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[5] J. Vuillemin. A unifying look at data structures. *Communications of ACM*, 23(4):229–239, 1980.