УДК 004.422.635.3

СПЕЦИАЛИЗИРОВАННЫЕ СТРУКТУРЫ ДАННЫХ ДЛЯ ТЕКСТА ПРОИЗВОЛЬНОЙ ФОРМЫ

Кузенкова Е.В. (федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»),
Кореньков Ю.Д. (федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»),
Научный руководитель – к.т.н., доцент Кореньков Ю.Д.
(федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»)

В данной работе рассматриваются особенности реализации структур данных для текстовых редакторов. Было выполнено сравнение существующих решений и предложено новое решение, являющееся улучшением совокупности рассмотренных решений.

Существует множество различных редакторов, в основе которых лежат различные структуры данных. Скорость взаимодействия редактора с пользователем напрямую зависит от временной сложности алгоритмов вставки, редактирования и удаления используемых структур данных. В ходе инкрементального семантического и синтаксического анализа важно иметь редактор с такими подлежащими структурами данных, которые бы позволили уменьшить накладные расходы на копирование и обход этих структур данных при различных операциях во время редактирования текста. Существующие решения уделяют преимущественное внимание различным операциям и сценариям редактирования, при этом зачастую проигрывая друг другу в остальных случаях. Стоит рассмотреть, какие бывают специализированные структуры данных для редакторов кода программ.

Использование в качестве подлежащей структуры данных буфера, основанного на линейном массиве, было опробовано ранее и показало существенную задержку между действием пользователя и откликом системы. К тому же такая реализация не предполагает возможность совместного редактирования текста и локального обновления состояния. Идея структуры данных Conflict-Free Replicated Data Type (CRDT) основывается на Operational Transformation (OT), которое является распространенным способом реализации асинхронного редактирования текста. Редактор здесь работает, отправляя операции, такие как вставки и удаления, между одноранговыми узлами и преобразуя их для применения к текущему тексту. К сожалению, многие реализации OT имеют проблему, заключающуюся в том, что они не всегда сохраняют порядок при удалении текста. Также стоит заметить, что напрямую представление CRDT использовать неэффективно из-за больших накладных расходов.

При представлении потенциально больших объемов текста часто используется Rope, который представляет собой иммутабельную сбалансированную древовидную структуру, использующую указатель с подсчетом атомарных ссылок для обмена данными, поэтому “копирование” любого поддерева – очень быстрая операция, имеющая сложность O(1). Очевидно, что Rope будет работать медленнее и потреблять больше памяти, чем классические строки, при этом традиционная оценка ресурсоемкости операций с ним логарифмична.

Ключевая идея внутреннего представления текста в редакторе Atom состоит в том, чтобы разделить содержимое буфера на две основные части: базовый текст, который соответствует последней версии документа, которая была прочитана или записана на диск, и так называемый патч, являющийся совокупностью несохраненных изменений, которые хранятся в отдельной разреженной структуре данных и должны быть наложены на базовое представление для получения актуального состояния редактора. Базовый текст неизменяем и хранится в одном

непрерывно выделенном блоке памяти. Для записи правок, а не для перемещения всего содержимого буфера в памяти, происходит мутация патча. Одновременно может существовать несколько слоев исправлений. Патч в самом верхнем слое всегда изменяем, но мы можем создать доступный только для чтения снимок текущего содержимого буфера, заморозив самый верхний патч и поместив новый патч на вершину стека. Изменения перетекают в этот новый патч до тех пор, пока снимок больше не будет нужен, после чего самый верхний патч может быть объединен с патчем на предыдущем слое. Однако проблема с описанным выше подходом заключается в том, что вставка изменения в список может потребовать проверки всех остальных изменений, что приводит к временной сложности $O(n^2)$. Для решения этой проблемы предлагается использовать Splay-дерево вместо простого списка, что уменьшает временную сложность операций до $O(n \bullet \log_2 n)$. Дерево должно поддерживать положение каждого узла как в новом, так и в старом координатном пространстве таким образом, чтобы была возможность эффективно обновлять положения всех последующих узлов всякий раз, когда появляется новый. Для этого, вместо связывания каждого узла с постоянным ключом, каждый узел связывается с некоторыми относительными значениями, представляющими расстояние узла от его левого предка как в старом, так и в новом координатном пространстве.

«Multiple buffer piece table with red-black tree, optimized for line model» или коротко «Piece Tree», использующаяся в редакторе VS Code, является модификацией Piece Table и основывается на сочетании красно-чёрного дерева и Rope. Тестирование показало неплохие результаты, так что такая реализация вполне приемлема и может быть принята во внимание. Оценка ресурсоемкости операций над ней также логарифмична.

На основе анализа рассмотренных решений и ряда сценариев редактирования текста в данной работе предлагается решение, основывающееся на Piece Tree, модифицированном для уменьшения ресурсоемкости операций над ним для сценариев редактирования текста с использованием «активного курсора». Такой подход, в отличии от традиционного использования структур данных посредством stateless операций над ними, позволяет значительно уменьшить число навигаций и перестроений во внутренней организации структуры данных при смежных операциях. Таким образом нижняя оценка ресурсоемкости операций редактирования в окрестности курсоров снижается до $O(1)$. В рассмотренных решениях прирост производительности ограничен достигаемым за счёт разбиения текста на фрагменты и организации этих фрагментов в структуры данных, операции вставки и удаления в которой работают быстрее. Предлагаемое решение отличается тем, что множественные активные курсоры фактически работают как дерево кэшей lookup-операций с отложенным до момента следующей абсолютной навигации сохранением результатов редактирования.

Предложенное решение показало свою эффективность и может быть использовано в решении задач построения редакторов текста произвольной формы. Кроме того, возможны улучшения и оптимизации, например, в направлении асинхронной работы.

Кузенкова Е.В. (автор) Подпись

Кореньков Ю.Д. (научный руководитель) Подпись