Тураев Тимур, 504 (SE)
1.10.2013

4.5 Разработать Persistent2List с операциями за логарифм.

1. Возьмем уже готовую персистентную структуру – массив. В нем операция доступа и операция замены проводится за O(log|S|). Можно построить список на этом массиве: будем хранить в каждой ячейке массива структуру с полями: указатель на реальные данные которые хотим связать в список (или можно сразу данные), индекс предыдущего элемента и индекс следующего элемента списка. Таким образом операция At(S) за логарифм даст нам указатель на элемент, на который указыавет указатель. Указателем будет индекс в массиве.

Предыдущий и следующий элементы в массиве получаются за O(1) – просто переходим по соответствующим «индексам-указателям», а вот возвращение элемента массива дает нам операция At(S), которая работает за логарифм.

Перенос указателя на начало делается также за O(1) + получения элемента массива (At(S)) за O(log|S|).

Удаление элемента из списка делается почти аналогично обычному двусвязному списку: аккуратное перекидывание указателей (снова будет работать операция At(s)) за O(log|S|) + операция «замена» - тоже логарифм.

Вставка – аналогично, но вставлять мы будем в конец массива (для этого можно завести отдельный индекс-указатель, который будет показывать на первое «свободное» место). Может оказаться так, что места в массиве не хватит – тогда можно расширить его (применяя для оценки те же рассуждения, которые проводили на лекции при доказывании почему время работы вставки в «расширяющийся массив» равно не, казалось бы O(n^2), а O(1); применяя амортизационный анализ) – таким образом время работы O(log|S|) амортизационно.

Персистентность обеспечит массив.

2. Можно строить тот же список на бинарном дереве, причем будем хранить элементы списка только в листьях дерева. А в каждом внутреннем узле будем хранить число листьев в его поддереве. Таким образом значение в корне есть число листьев, которое просто равно числу элементов в списке в текущий момент.

Переход в начало списка простой – двигается от корня в самый левый узел, это делается за O(h) = O(log|S|).

Переход в следующий и предыдущий делается тоже несложно, только нужно еще вместе с указателем хранить «номер» элемента в списке, на который указывает указатель и его обновлять: если вставили элемент – то прибавить единицу, если удалили – то вычесть. Передвинули указатель на следующий элемент – тоже прибавили единицу и т.д.

Переход по дереву тогда будет такой: идем от корня (храним, стало быть, указатель на корень каждой «версии дереве») и смотрим сколько листьев в левом и правом поддереве. Если «номер» следующего элемента не больше числа листьев в левом поддереве – значит он определенно там. Если больше – значит надо искать в правом. И потом запускаем спуск рекурсивно (при этом не забыть либо запомнить сколько мы уже прошли, либо вычитая это число из параметра функции поиска: к примеру если нам нужен 10й элемент, а левый сын корня равен 8, то нам надо найти второй элемент в правом поддереве корня)

Вставка и удаление проделываются как обычная вставка и удаление в бинарное дерево, вот только для сохранения асимптотической оценки придется реализовывать это на каком-нибудь самобалансирующемся дереве: либо АВЛ, либо красно-черном. Тогда вставка и удаление будет гарантированно на логарифм (в среднем).

Персистентность будет делать аналогично персистентной куче – когда при вставке и удалении будем обновлять ветку (при поворотах дерева и при обновлении значений в узлах дерева на плюс один (при добавлении элемента) или минус один (при удалении)) – будем копировать ее и устанавливать указатели на «соседа» той испорченной ветки. При этом появится два корня, и будем хранить в отдельном массиве-истории версий ссылки на корни в различные моменты времени.