Интересная структура

© И. И. Гридасов

22 октября 2016, 19 декабря 2016

Чтобы избавиться от недостатков предыдущей версии, при запросе удаления элемента не будем удалять его, а будем просто помечать, что такой элемент удалён. При этом операция добавления останется той же, и по уже доказанному, её амортизационная оценка $O(\log n)$. Но так как могут быть элементы одинкаково значения, то чтобы асимптотика поиска не ухудшилась, будем поддерживать инвариант: в группе среди элементов одинакового значения, сначала идут неудалённые, а потом удалённые.

Тогда операция удаления примет следующий вид: Сначала находим группу, в которой находится элемент, который надо удалить. Это делается прямым проходом по всем группа за $O(\log n)$. Теперь в этой группе бин-поиском ищем самый правый неудалённый элемент с таким значением. Помечаем его как удалённый.

Перестройка: Так как мы не удаляем элементы, а помечаем, то фактически элементов, которые занимают место в структуре, больше, чем число неудалённых элементов. Для того, чтобы кол-во помеченных элементов в структуре было не слишком велико и чтобы не испортилась асимптотика операций, будем как-только кол-во помеченных элементов стало равно кол-ву непомеченных, то делаем перестройку. Тогда перед каждой перестройкой было сделано хотя бы n, (где $2n$ - сейчас всего элементов в структуре) операций удаления. Так как перестройка делается за $O(n \log n)$, то мы можем придумать монетки, так чтобы с каждой операции удаления мы брали по $\log n$ монеток. Функция поиска также работает за нужную асимптотику, так как кол-во групп осталось $O(\log n)$. Операция добавления по предыдущим док-ствам осталась работать за $O(\log n)$.

Общее представление структуры:

Храним переменную size или n - кол-во элементов в структуре. Будем хранить список указателей на начало каждого набора в порядке увеличения степени двойки. При этом буфер каждого набора пусть будет в 2 раза больше, чем кол-во элементов, которое должен хранить. Таким образом для числа 19, будем хранить список из 5 указателей на буферы, размером 2, 4, 8, 16, 32 соотвественно. В первом будет лежать 1 элемент, во втором 2 и в последнем 16 элементов, все они лежат в первых ячейках соответствующих буферов. Оставшиеся буферы пока пустуют(но они есть). При таком представлении структуры, кол-во занимаемой памяти $\leqslant 4n$, так как будем поддерживать инвариант, что последний буфер заполнен, а значит так как суммарно все буферы без последнего занимают на 2 ячейки меньше, чем один последний буфер, то суммарная длина всех буферов $\leqslant$ 2 * длина последнего = 4 * кол-во эл-тов в последнем буфере $\leqslant$ 4 * кол-во всех эл-тов(занятых ячеек) во всех буферах = $4n$.

Функция Search(x):

Перебираем все наборы массивов, в каждом из них запускаем бин-поиск. Кол-во наборов $O(\log n)$, длина каждого набора $O(n)$, следовательно бин-поиск отработает за время $O(\log n)$. Итоговая асимптотика операции: $O(\log^2 n)$.

Функция Insert(x):

Пусть первый буфер пуст, то есть size делится на 2. (Проверку на пустоту далее можно делать поддерживая переменную $tmp = size$, а при переходе к следующей итерации делать $tmp/ = 2$). Тогда просто поставим этот элемент на первое место в первом буфере. Если же нет, то сольём его в первый буфер, так как буфер на 2 эл-та так можно сделать. Теперь если второй буфер пуст, то просто перекопируем эти два элемента туда. Иначе проделаем тоже самое, сольём эти 2 элемента с элементами из второго буфера во второй буфер. Слияние делаем, так чтобы второй буфер заполнялся с конца, это позволит нам не создавая новой памяти просто прилить один буфер к другому. Если же мы такими действиями дошли до последнего буфера, то есть теперь он заполнен не на половину как должен а полностью, то создаем новый буфер размером в 2 раза больше этого буфера, перекопируем все элементы в него. добавляем указатель на него в конец списка указателей на буферы. Операция выполняется за $O(2^i)$, где i - кол-во единиц в конце числа n в его двоичном представлении. $2^1 + \ldots + 2^{i+1} + 2^{i+1} = 2^{i+2} + ^{i+1} -1 = O(2^i)$, где 2^i тратится на слияние полного $i-1-$ буфера, заполненного полностью, и $i-$, заполненного "нормально", то есть на половину, в каждом из них

по 2^{i-1} эл-ов. И последнее 2^{i+1} - перенос эл-ов из заполненного буфера в первую половину следующего. И ещё может потребоваться создать новый буфер размером 2^{i+2}, но это также делается за $O(2^i)$. Увеличиваем $size$ на 1.

Функция Delete(x):

За $O(\log n)$ найдём какому набору принадлежит эл-т, который надо удалить. Если нет непустых буферов, меньших, чем тот, в котором надо удалить, то просто раскидываем его элементы, во все меньшие буферы по порядку, пропустив элемент который надо удалить. Это мы сделали за $O(2^i)$, где i - кол-во нулей в конце числа в двоичном представлении числа n. Так же так как теперь последний буфер пуст, то освободим память, которую он занимал. Так как её размер 2^{i+1}, то её удаление выполняется за $O(2^i)$. Если же наш буфер не меньший, то возьмём самый маленький непустой буфер и один его эл-т запишем на место эл-та, который надо удалить. А оставшиеся раскидаем на меньшие действия вышеописанным алгоритмом. Осталось бин-поиском найти место для эл-та, который мы поставили на место удаляемого. Асимптотика бин-поиска $O(\log n)$. Асимптотика всей операции $O(\log n + 2^i)$, где i - кол-во нулей в конце числа в двоичном представлении числа n. Уменьшаем $size$ на 1.

Докажем, что учётное время работы каждой операции $O(\log n)$:

Воспользуемся методом бухгалтерского учёта для операции Insert. Пусть при поступлении новой монетки она приносит кол-во монеток равное числу существующих буферов(включая пустые) + 1. Это число есть $O(\log n)$ благодаря тому, что последний буфер точно заполнен эл-тами. Мы кладём на каждый буфер по монетке и 1 остаётся. Тогда при оплате за новый запрос, возьмём монетки с последней единицы, которую мы затронули при запросе. Если этой единицы не было то есть $size$ делится на 2(младший бит в двоичном представлении равен 0), то потратим отложенную монетку. Иначе возьмём монетки с самой правой единицы, которая участвовала в операции, пусть её номер i(начиная с 1). На ней накопилось 2^i монет, первая пришла с числа вида 100...0(i)..., а предпоследняя с 111...1(i).... Значит этих монет хватит на совершение операции, так как её асимптотика $O(2^i)$. Отсюда получаем, что амортизированная стоимость операции $O(\log n)$

Аналогичное док-ство можно провести для операции Delete. Найдём, что амортизированная оценка $O(2^i)$, где i - кол-во нулей с конца в двоичном представлении числа n равна $O(\log n)$. Теперь добавляем в каждую операцию ещё $O(\log n)$(бин-поиск и поиск нужной группы), получаем,

что амортизированная стоимость каждой операции $O(\log n)$*.

*Замечание, заметим, что эта амортизационная оценка показывает лишь, что при применении большого числа insert без delete в среднем операция будет работать за $O(\log n)$. Но при чередование операций insert и delete неверно, что в среднем операция будет за $O(\log n)$. Достаточно взять $n = 2^k$, и поочерёдно делать delete и insert, тогда каждая операция будет за $\Theta(n)$, и значит амортизационное время каждой есть $\Theta(n)$. К сожалению, как сделать так, чтобы и в этом случае амортизационное время было $O(\log n)$ не ясно.