Дэк

© И. И. Гридасов

19 октября 2016

Реализация структуры Дек:

Храним буфер(массив фиксированной длины), в который будем записывать элементы. Будем хранить его размер *size*. Также будем хранить два индекса(или указателя), которые указывают на первый и последний элемент в массиве. Если количество элементов стало больше, чем размер буфера, то создадим буфер в 2 раза большей длины и перекопируем элементы из старого буфера в новый, старый буфер удалим. В процессе жизни Дека будем поддерживать инвариант: Если пройти по элементам с левого указателя до правого(считая, что с последнего элемента буфера мы переходим в первый), мы получим все элементы Дека в нужном порядке, начиная с первого и заканчивая последним. Тогда за $O(n)$ мы можем пройтись по элементам Дека в правильном порядке(с первого до последнего).

Описание операций:

Resize: Когда, кол-во элементов, которые нужно хранить превысило размер буфера, то нужно создавать новый, больший буфер. Например, создадим массив в 2 раза большей длины, пройдёмся по элементам Дека и перекопируем их в новый буфер, установив левый указатель в начало нового буфера, а правый на последний записанный элемент. Время $O(n)$ в него входит создание буфера из $2n$ элементов и перекопирование n элементов в новый буфер.

push_back: увеличиваем указатель на последний элемент на 1 и записывем новое число в эту ячейку, в случае если вышли за границу массива(за правую), то перемещаем указатель в начало массива и записываем новое число в данную ячейку. Если же правый указатель ещё до операции стоял там же, где и первый, то значит весь буфер уже заполнен и нужно запустить Resize и уже после добавить элемент, выше указанным

способом. Не забыть сделеать $size++$

pop_back: просто уменьшаем правый указатель на 1, при переходе через левую границу переходим в последний элемент. $size$ -= *1*. Время работы $O(1)$

Аналогично выполняются операции $push\_front$ и pop_front, при добавлении левый указатель сдвигается влево, а при удалении вправо.

$operator[]$: Пусть arr - буфер, $left, size$ - левый индекс и размер буфера соответственно. Тогда $i-$ элемент находится в ячейке $arr[(left + i) mod size]$. время $O(1)$

Доказательство Асимптотики. Покажем, что амортизационное время работы $push\_back$ и $push\_front$ $O(1)$.

Метод Бухгалтерского учёта: Повесим на каждое число по 3 монетки. Пусть на непосредственное добавление элемента хватает одной монетки. Далее каждый элемент либо может быть удален, тогда ещё одной монетки хватит на обеспечение операции pop_back и pop_front, Так как они выполняются за $O(1)$. Итого потратили только 2 из 3. Либо может дожить до перехода в новый буфер, но тогда заметим, что успело накопиться хотя бы $2 * (\frac{size\_buff}{2}) = size\_buff = size = n$ монет, которые не удалялись, Так как при создании буфера в нём было лишь $\frac{size\_buff}{2}$ элементов. А значит, с новодобавленных элементов можно собрать монетки на операцию $Resize$. Так как эта операция выполняется за $O(n)$, то можно выбрать монетку такой, чтобы n монет хватило на эту операцию.

Метод потенциалов: В методе потенциалов, главную роль играет нахождение функции от состояния, такой, что $\Phi(D_0) = 0$ и $\Phi(D_i) \geqslant 0$. В данном примере, как и в динамическом массиве, подойдёт $\Phi(D_i)$ равная, кол-ву элементов в структуре. Тогда, если $\Delta\Phi(i) = \Phi(i) - \Phi(i-1)$, то амотризационная стоимость вычисляется как $a_i = t_i + \Delta\Phi(i)$. Где t_i - непосредственно время, необходимое для выполнения операции. В нашем случае для операции $Resize$ можно считать $t_i = n$, где n - кол-во элементов в Деке, так как асимптотика данной операции $O(n)$. А для других операций $t_i = 1$(без учёта возможного пересоздания буфера). Можно взять $\Phi(i) = 2\frac{size*size}{size\_buff}$. Тогда для операций удаления:

$$a_i = t_i + 2 * \left(\frac{(size-1)*(size-1)}{size\_buff} - \frac{size*size}{size\_buff} \right) \leqslant t_i = 1.$$

Для операции добавления элемента без $Resize$:

$$a_i = t_i + 2 * \left(\frac{(size+1)*(size+1)}{size\_buff} - \frac{size*size}{size\_buff} \right) =$$
$$t_i + 2 * \left(\frac{2*size+1}{size\_buff} \right) \leqslant t_i + 2 * \frac{2*(size+1)}{size\_buff}$$
$$\leqslant t_i + 2*2 = t_i + 4 = 5$$

А для операций добавления с Resize:

$$a_i = t_i + 2 * \left(\frac{(size+1)*(size+1)}{size\_buff\_new} - \frac{size*size}{size\_buff} \right) =$$
$$t_i + 2 * \left(\frac{(size+1)*(size+1)}{2*size\_buff} - \frac{size*size}{size\_buff} \right) = t_i + \frac{-size*size+2*size+1}{size\_buff} =$$
$$t_i - size + 2 + \frac{1}{size} \leqslant t_i - size + 3 = size - size + 3 = 3$$

Таким образом все $a_i \leqslant 5$, А значит амортизационное время работы каждой операции $O(1)$, что и требовалось показать.