ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ
Deque

Минск, 2021

Deque (double-ended queue) - индексируемая двусвязная очередь, поддерживающее следующие операции, каждое из которых работает за константу:

- *push_back(x)* - добавляет x в конец очереди.
- *push_front(x)* - добавляет x в начало очереди.
- *pop_back()* - удаляет последний элемент из очереди.
- *pop_front()* - удаляет первый элемент из очереди.
- *random access* индексирование.

Простейший *deque*$<T>$ (здесь и далее за T будем считать тип данных с которым позволяет работать контейнер deque) представляет из себя некоторый массив типа T размера *capacity*, из которых задействовано лишь *size* элементов (размер *deque*).

При добавлении нового элемента в *deque* мы обращаемся к зарезервированным и еще не используемым элементам массива и, при отсутствии таковых, создаем новый массив размера *capacity*$*k$, после чего можно переместить значения старого массива в новый и, наконец, добавить новый элемент. Такой подход используется и в *vector*, что позволяет обходиться без больших расходов на память.

Но что на счет итераторов? Они при такой политики будут инвалидироваться, храня указатели на элементы старого массива. Предложенный далее алгоритм позволит *resize*'ть *deque* без инвалидации итераторов.

Немного о процессе добавления/удаления элементов из обычного *deque*.

В любой момент времени необходимо поддерживать два, скажем так, указателя:

- первый (левый) указывает на начало очереди
- второй (правый) указывает на следующий элемент после последнего в очереди

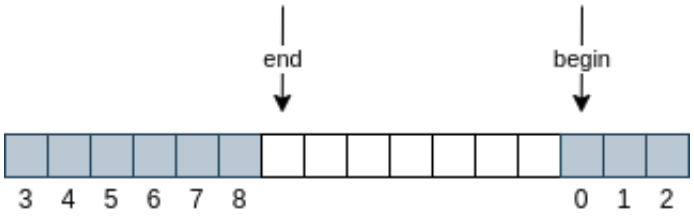
Для того чтоб добавить элемент в конец очереди достаточно положить его в ячейку массива, на которую указывает второй указатель, после чего инкрементировать его.

Для добавления элемента в начало очереди необходимо декрементировать левый указатель и положить туда добавляемый элемент.

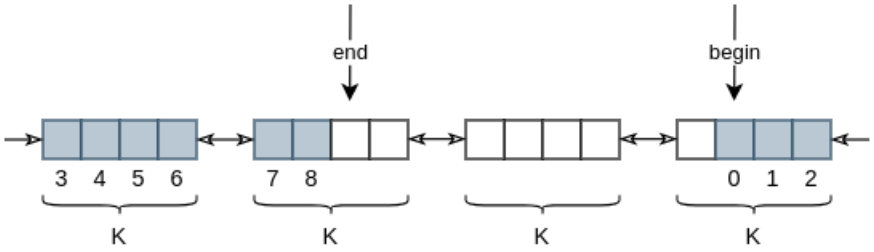
Делаем каждый раз *resize* при добавлении, когда *capacity*==*size* и получаем амортизированную сложность O(1).

Улаление - действия, обратные добавлению.

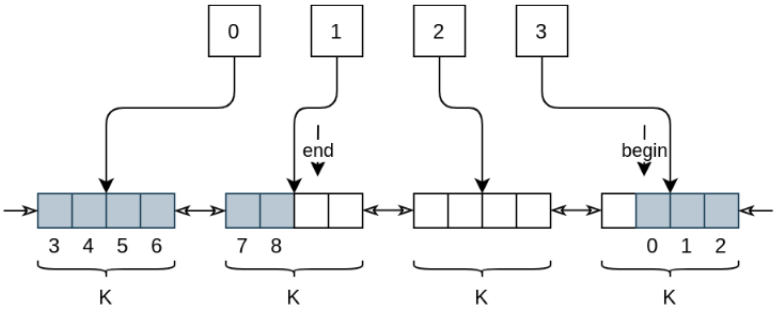
При инеркментировании/декрементировании указателей следует отметить, что первый и последний элементы выделенного массива связаны (т.е. после инкрементирования указателя на последний элемент массива он должен указывать на первый).



Чтоб получить стркутуру, где итераторы не инвалидируются разобьем выделенный массив на блоки фиксированного размера K.



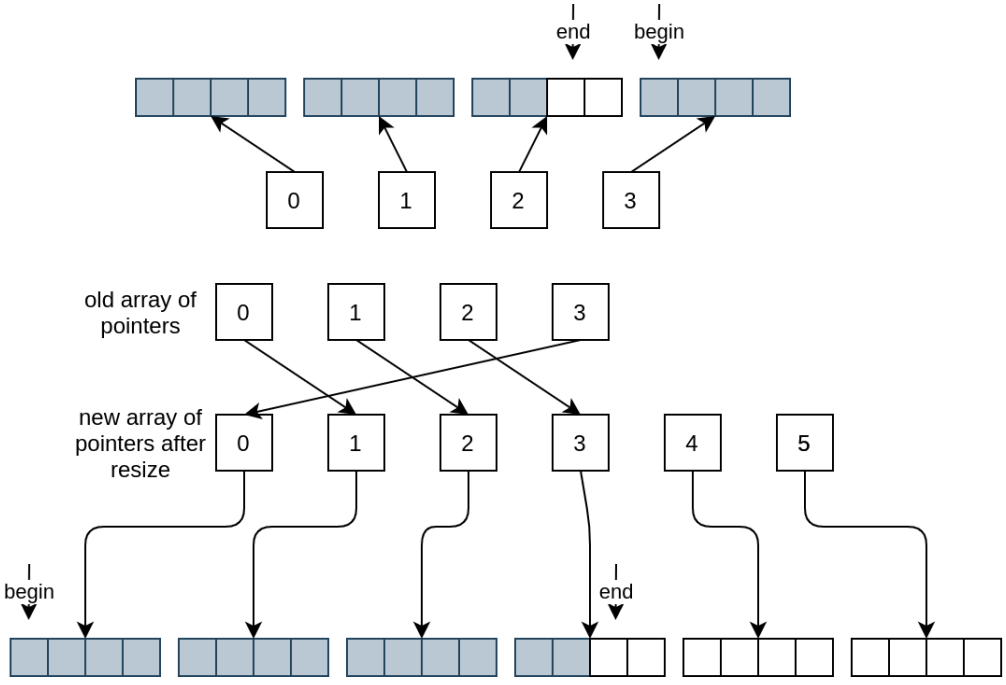
Для возможности *random access* индексирования создадим массив размера $\frac{capacity}{K}$ (*capacity* делится на K), который будет хранить указатели на блоки.



При попытке инкрементировать указатель (на объект), указывающий на последний элемент блока, то указатель в кольце из блоков перемещается на первый элемент следующего блока. Аналогично и при декременте.

resize теперь будет проводиться над массивом указателей.

Может быть ситуация, когда при добавлении необходимо сдвинуть указатель в соседний блок со свободными элементами, однако писать туда будет нельзя, так как при *resize* тот блок, в котором находится первый элемент очереди должен быть под первым указателем нового массива. Иными словами, начальный блок должен оказаться в начале, а конечный в конце. Но если в одном блоке получится, что *end* левее *begin*, то этот блок придется разбить, что тоже приведет к инвалидации итераторов. Потому следует наложить **запрет** на то, чтоб в одном блоке *end* был левее *begin*.



Задание:
Реализовать описанную структуру данных со следующими свойствами:

- *push_back* - O(1)
- *push_front* - O(1)
- *pop_back* - O(1)
- *pop_front* - O(1)
- *clear*
- *size*
- *empty*
- *random access iterator* без инвалидации при *resize*, который тоже надо будет реализовать.