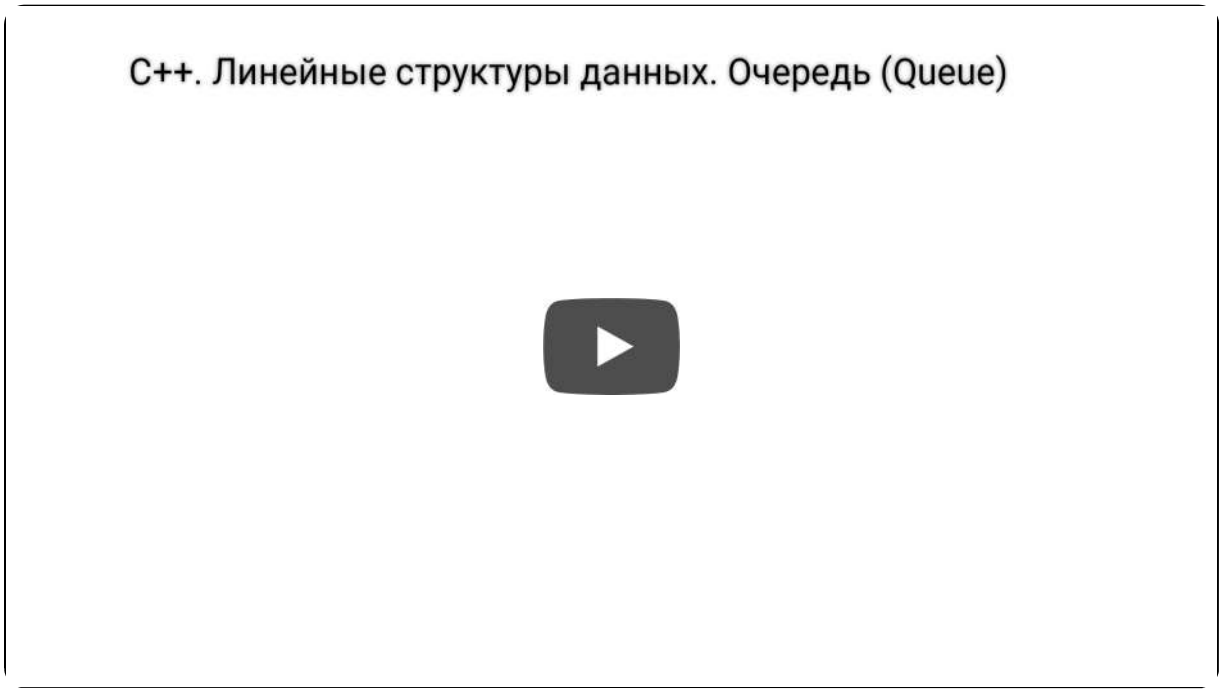
Очередь



Очередь

Другая структура данных, похожая на стек, — это очередь (queue). Её назначение понятно из названия. Элементы выстраиваются один за другим, новые элементы добавляются в конец очереди, а удаляются элементы из начала очереди. Данная структура работает по принципу “первый пришёл — первый ушёл” — FIFO.

Очередь поддерживает следующие операции:

- Добавить элемент в конец очереди (push)
- Удалить элемент из начала очереди (pop)
- Узнать значение первого элемента очереди (front)
- Узнать размер очереди (size)

Все операции с очередью должны выполняться за $O(1)$.

Подумаем, как можно хранить очередь. Давайте, как в случае со стеком, будем хранить элементы очереди в массиве, добавляя новые элементы в конец массива. Но чтобы не перемещать все элементы очереди при удалении первого элемента, будем просто считать, что начало очереди смещается, то есть позиция удаляемого элемента не используется. Мы будем запоминать

позицию, где начинается очередь. Тогда при операциях удаления элементов из очереди очередь будет двигаться дальше от начала массива, а место удалённых элементов будет потерянной областью памяти. Чтобы бороться с этой проблемой, есть два подхода. Один подход: давайте логически замкнём массив, в котором будет храниться очередь в кольцо, то есть за последним элементом будет идти первый. Тогда когда очередь доползёт до конца выделенного массива, она начнёт расти с начала, то есть мы будем повторно использовать места, занятые удалёнными элементами. Размер такого массива должен позволять разместить столько элементов, сколько одновременно могут храниться в очереди в любой момент времени, но не обязательно — все добавленные в очередь элементы.

Другой подход — регулярное сжатие свободного места. Если в начале очереди накопилось слишком много неиспользованных ячеек (например, столько же, сколько элементов в самой очереди), то можно сразу удалить все эти ячейки, скопировав все элементы очереди в начало массива. То есть мы будем операцию реального удаления элементов выполнять редко, но удалять сразу же много элементов, тогда средняя сложность в пересчёте на одну операцию удаления будет $O(1)$.

Есть и другой интересный способ реализации очереди. Если последовательность элементов положить в стек, а потом излечь, то она развернётся в обратном порядке. Если это повторить ещё раз, то последовательность элементов опять развернётся и порядок будет исходным. Очередь же не меняет порядок элементов, поэтому можно реализовать очередь с использованием двух стеков.

Возьмём два стека: s_1 и s_2. Операцию добавления будем всегда делать в первый стек. А удалять элементы будем из второго стека. Но при этом второй стек изначально пустой, все добавленные элементы будут находиться в первом стеке. Если необходимо удалить элемент, а второй стек пуст, то все элементы из первого стека перенесём во второй стек. При этом самый первый добавленный элемент, который был внизу первого стека, окажется на вершине второго стека и выйдет из очереди первым. Если s_2 не пуст, достаём элементы из него. Как только s_2 окажется снова пустым, повторяем ту же операцию.

Кажется, что при таком подходе изменится вычислительная сложность операций очереди, потому что операция удаления элементов может

выполняться долго, так как придётся все элементы перекладывать из одного стека в другой. Но такие операции будут происходить редко, и средняя сложность одной операции оказывается невелика. Действительно, пусть мы n элементов добавили в очередь и n элементов удалили, как-то чередуя эти операции. Тогда каждый элемент сначала был добавлен в стек s_1, потом перенесён в стек s_2, потом удалён из стека s_2, то есть мы выполнили 2 операции добавления элемента в стек и 2 операции удаления, всего — $4n$ операций со стеком для реализации $2n$ операций с очередью. Итого на одну операцию с очередью в среднем требуется $O(1)$ операций со стеком.

Эта идея может показаться сложной для применения на практике, но мы используем её для реалиазации другой структуры — очереди с минимумом или максимумом, то есть очереди, которая поддерживает ещё один запрос — узнать значение наименьшего или наибольшего элемента в очереди. Сложно реализовать операцию взятия минимума, когда из структуры данных удаляется один элемент. Пусть мы знали, что наименьшее значение всех элементов в структуре равно m и из структуры был удалён элемент, равный m. Какое стало теперь наименьшее значение во всей структуре? Оно может оказаться как m, так и больше m.

Но раньше мы рассмотрели, как реализовать стек с минимумом, теперь для реализации очереди с минимумом мы будет использовать два стека с минимумом, то есть минимумы в стеках будут поддерживаться автоматически. Чтобы узнать значение минимума в очереди, нужно взять минимум из минимальных значений в каждом из двух стеков.