Оглавление

[Общие понятия 2](#_Toc130326323)

[Односвязный список 3](#_Toc130326324)

[Двусвязный список 5](#_Toc130326325)

[Бинарное дерево 6](#_Toc130326326)

[Стек 8](#_Toc130326327)

[Очередь 9](#_Toc130326328)

[Кольцевая очередь 10](#_Toc130326329)

[Очередь с приоритетом 10](#_Toc130326330)

[Deque (Дек) 12](#_Toc130326331)

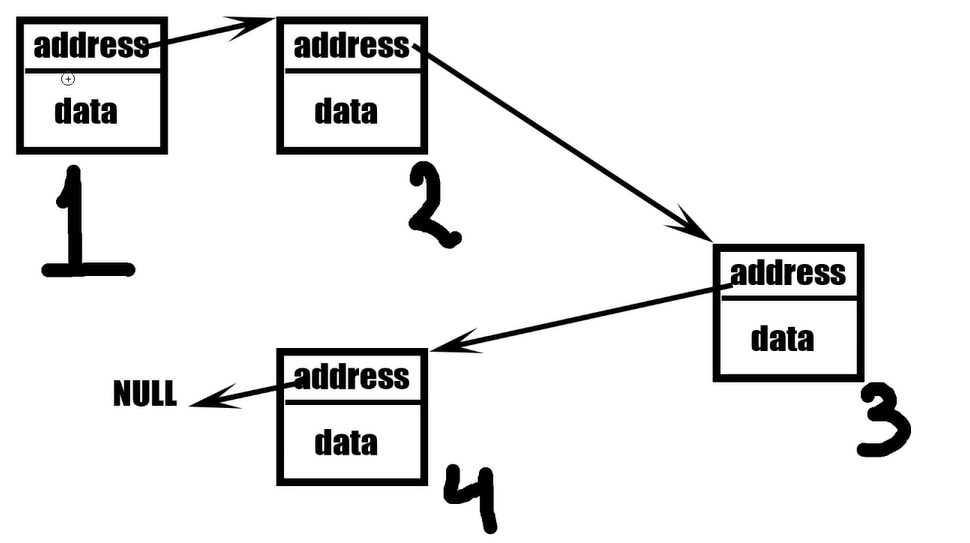
# Общие понятия

Динамические структуры данных предназначены для удобного хранения и манипулирования данными

# Односвязный список

Элемент односвязного списка – это **класс**, который содержит 2 поля: **адрес и данные**.

В отличии от массива (непрерывная область памяти), поле «адрес» хранится не адрес текущего элемента, а адрес **следующего**. Таким образом, мы можем убрать необходимость последовательного выделения памяти под ячейки нашего контейнера, т.е. мы можем убрать слово «непрерывная» из определения.



Таким образом, **Список –** это совокупность элементов, в котором каждый элемент знает адрес следующего элемента

Адрес последнего элемента – **NULL**

В отличии от массива нам не нужно удалять весь список, если мы хотим добавить еще один элемент в конец, а лишь дописываем N-1 адрес новой ячейки.

То же самое с удалением. Мы не создаем новый список, как с массивом, а просто, например, при удалении ячейки «3» в ячейку «2» записываем адрес ячейки «4» и подтираем данные в ячейки «3»

При добавлении элемента в середину массива порядок действий точно такой же, как и с удалением (только без удаления ячеек)

**Однако**:

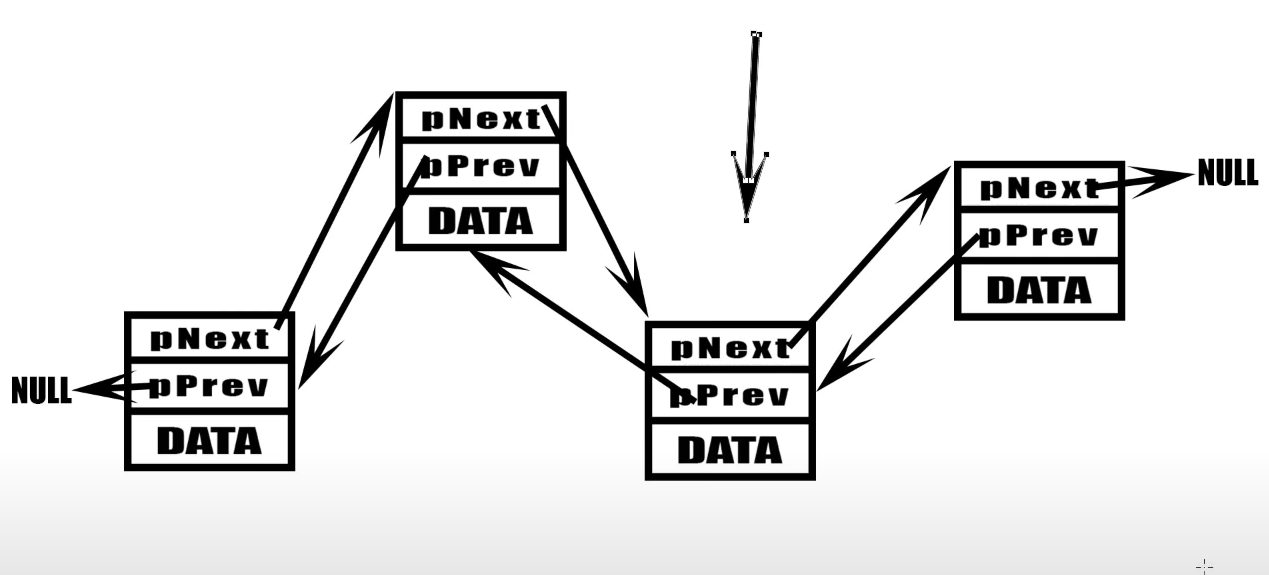
Если мы хотим получить 4 элемент, нам необходимо пройтись сперва по следующему пути 0-1-2-3-4; нельзя сразу перескочить на 4 элемент. Чем дальше в конец, тем медленее доступ к данным.

# Двусвязный список

Ячейка состоит теперь из 3 полей: добавляется адрес предыдущей ячейки



Восполняет недостаток односвязного списка, который связан с долгим доступом к последним элементам списка: поиск может осуществляться с конца списка (если ближе к концу, ищем с конца, иначе ищем с начала



# Бинарное дерево

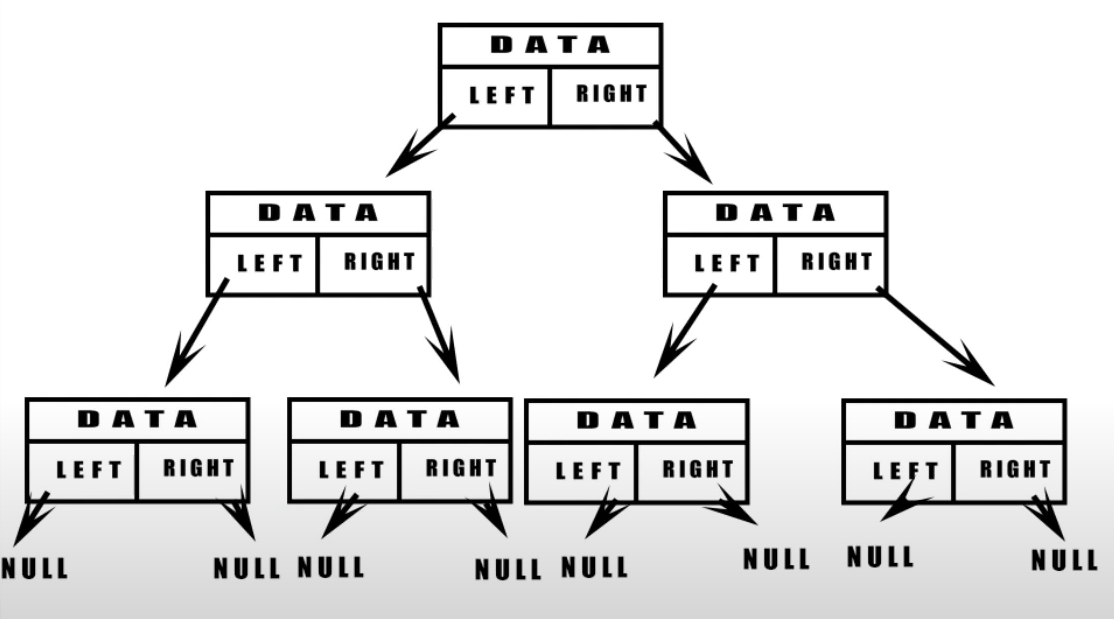
Состоит из узлов (элементов).

Каждый узел явялется родителей двух других узлов.

Элемент, у которого нет родителей является **корнем**

Элемент, у которого нет детей является **листьями**

Бинарное дерево называется бинарным, потому что каждый узел бинарного дерева может иметь максимум 2 потомков

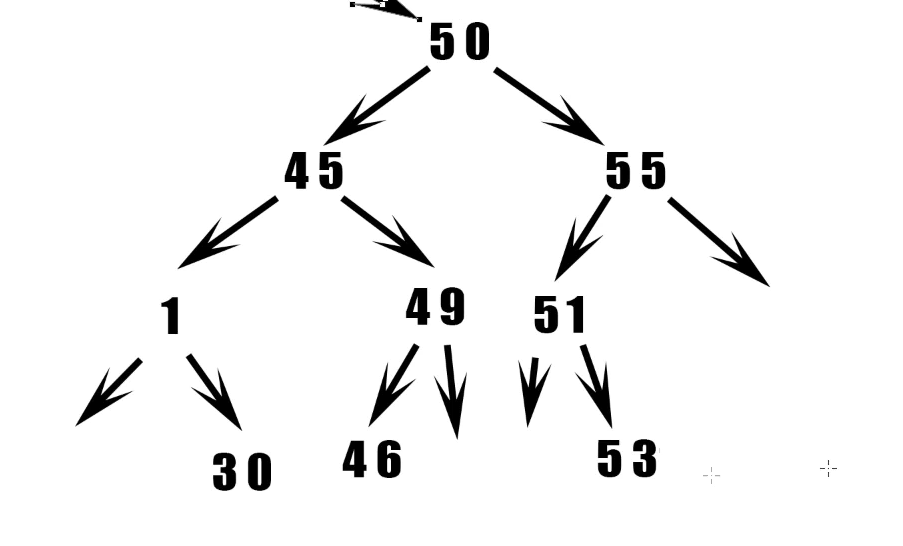


Бинарное дерево – упорядоченная структура данных (всегда отсортированная)

Слева хранится меньшее число, справа – большее.

Добавление нового элемента идет от корня.

**Очень быстрый поиск** за счет того, что часть данных отбрасывается. Например мы имеет следующее дерево



* 30 больше 50 ? Нет. Идем налево. Правую ветку отбрасываем
* 30 больше 45 ? Нет. Идем налево. Правую ветку отбрасываем
* 30 больше 1 ? Да. Идем направо. Левую ветку отбрасываем

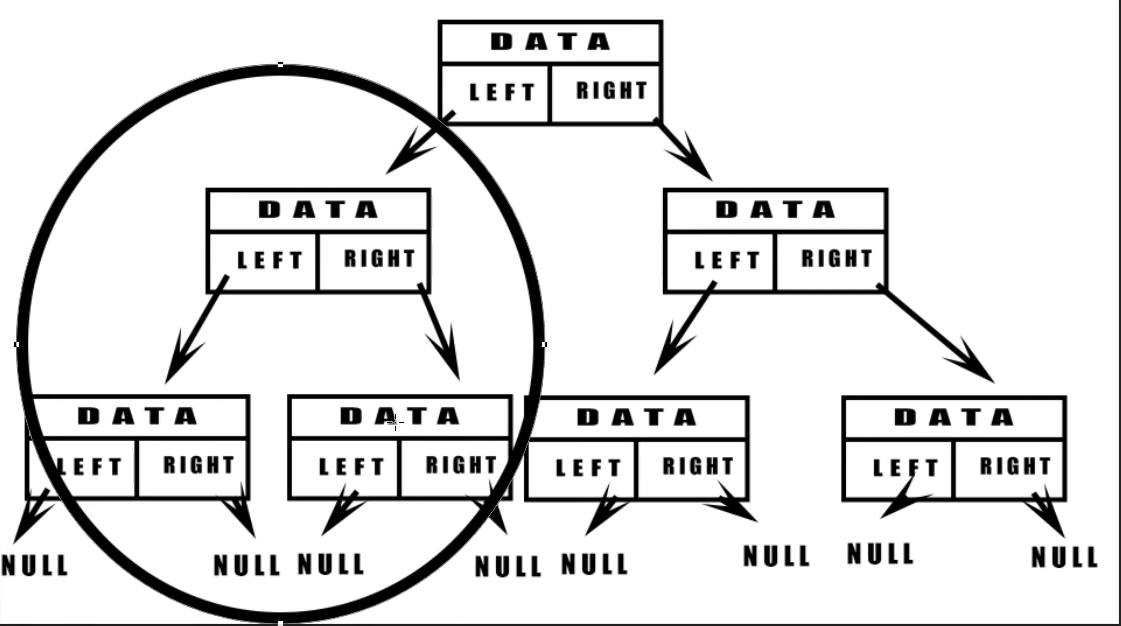
Итог: нашли число 30 за 3 шага.

Глубина бинарного дерева = количество шагов надо для того, чтобы найти число

Добавление тоже идет быстрее, чем в обычном массиве.

Используется в структуре **map** и для **реализации ассоциативных массивов (**в таком случае в ячейках будут записываться ключи (каждое значение уникально и может бысть сравнено с другими значениями))

Бинарное дерево является рекурсивной структурой, ибо:

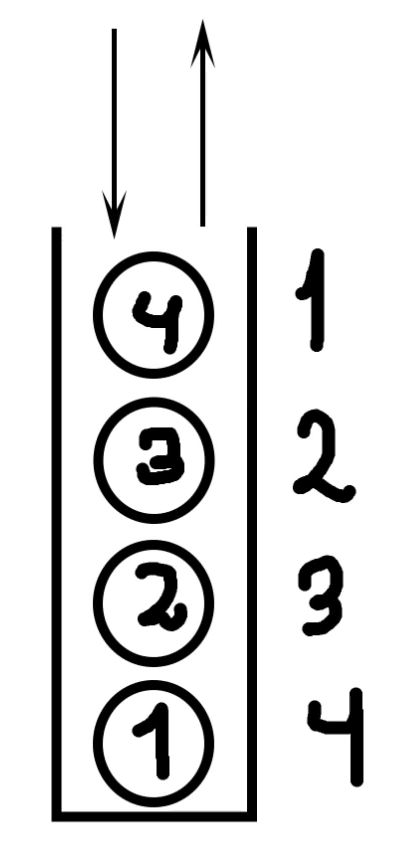


…часть дерева, что взята в кружочек – тоже полноценное бинарное дерево.

Для манипуляции с данными в бинарном дереве удобно использовать рекурсивные алгоритмы

# Стек

Работает по принципу LIFO (Last in – first out)



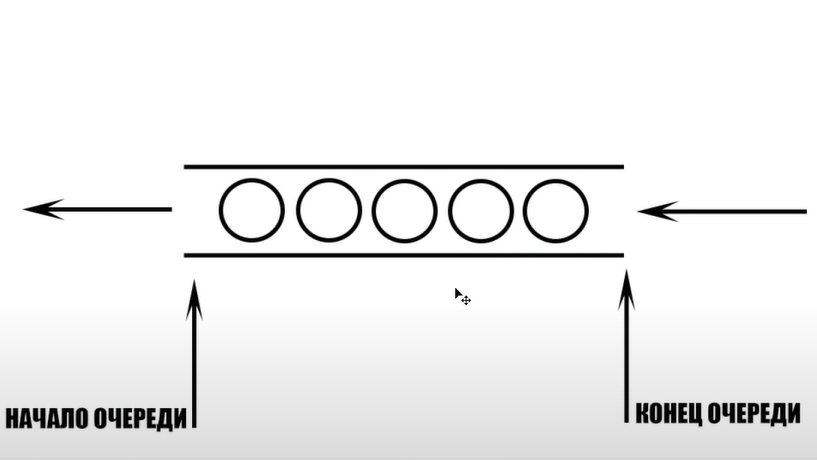
По своей сути напоминает магазин пистолета.

Не важна скорость перебора элементов (ибо мы не перебираем)

Однако критически важно добавлять или удалять элементы (добавление и удаление происходит с одной стороны) – поэтому лучше использовать **урезанный односвязный список**

# Очередь

Подчиняется правилу FIFO: first input – first output



Очереди нужны для того, чтобы обрабатывать команды и программы в том порядке, в котором они поступают

Нельзя вставить/убрать элемент из середины очереди

Можно взаимодействовать последний и первый элемент (добавить элемент в начало очереди нельзя – только в конец)

Добавляем в конец, извлекаем из начала

Идеально подходил реализация через **двусвязный список** (пару методов убрать, пару методов прибавить)

## Кольцевая очередь

Элемент из начала очереди не исключается, а мы получаем данные из него и переносим в конец.

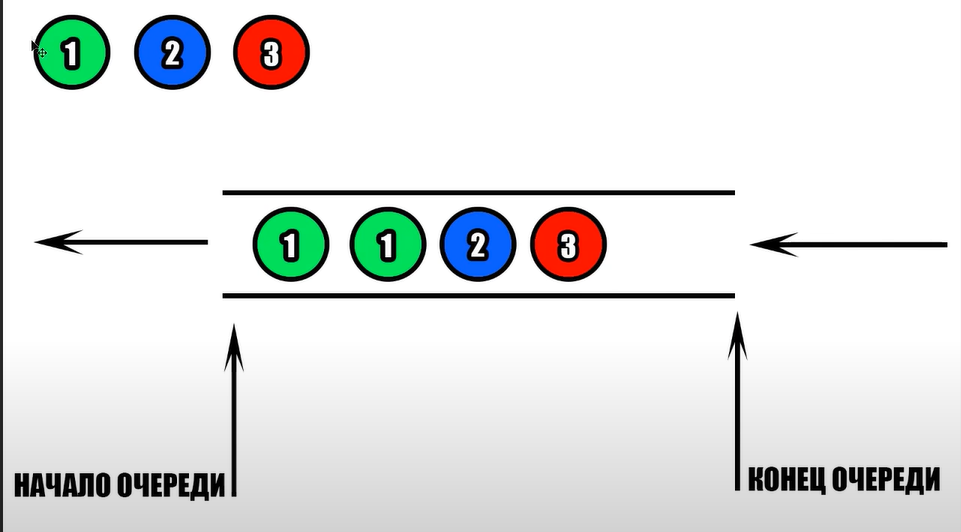
## Очередь с приоритетом

Порядок извлечение зависит от приоритета обработки элемента в очереди

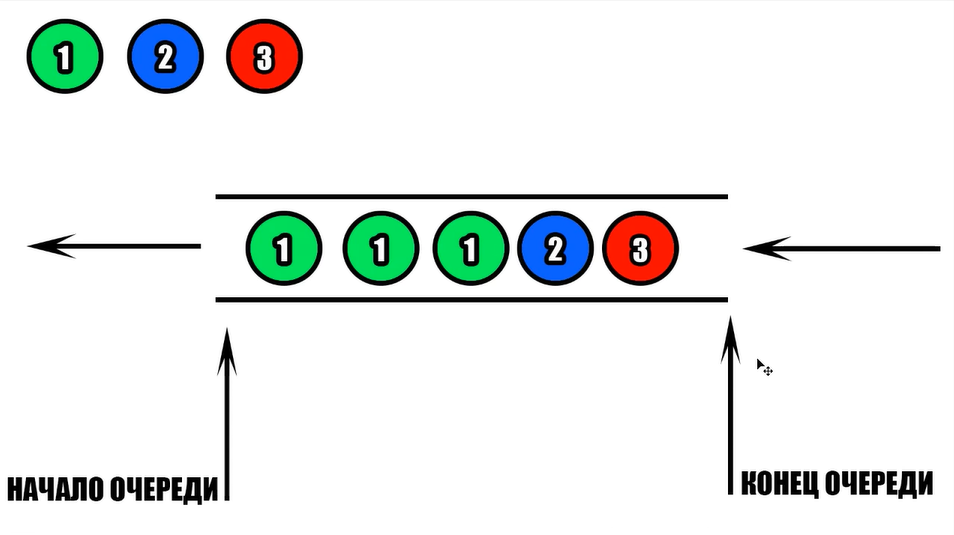
Элементы в очереди должны быть упорядочены по приоритетам в момент поступления в очередь (сразу становится в нужный порядок) **– приоритетное включение**

Если мы говорим про извлечение, то существует точно такой же, как и предыдущий, только с извлечением - **приоритетное исключение** (данные поступают в обычном порядке, но извлекаются согласно приоритетам)

**Приритетное включение**

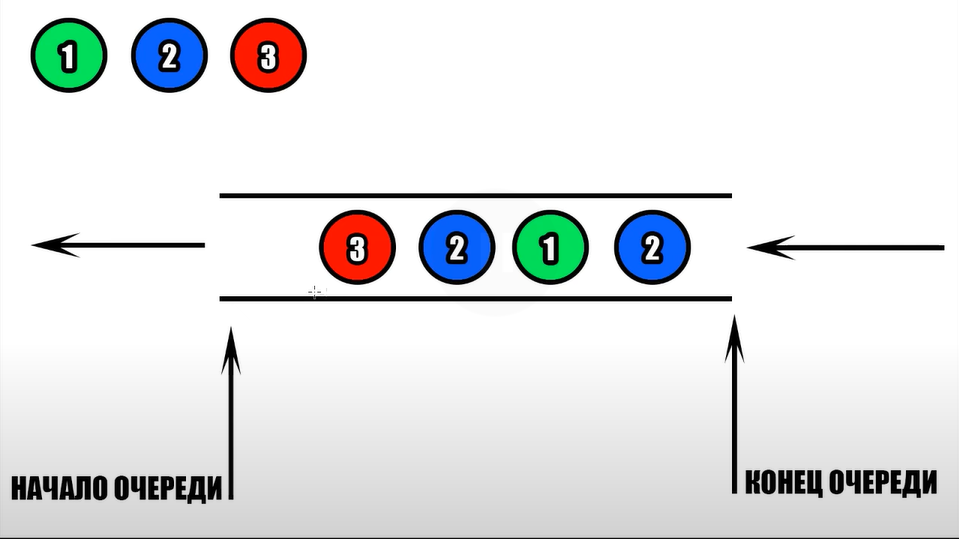


Если добавим еще один элемент с приоритетом 1

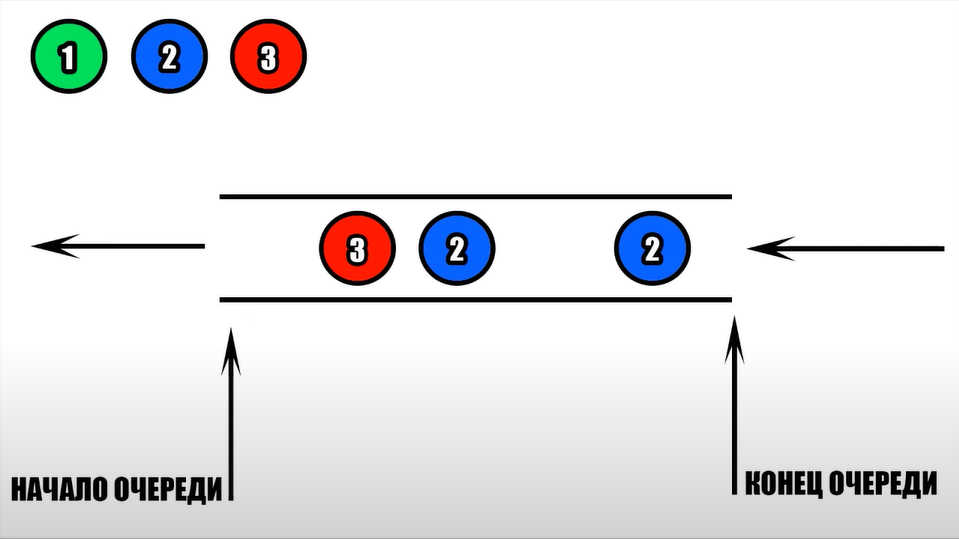


**Приоритетное исключение**

Поступают элементы вне приоритетов



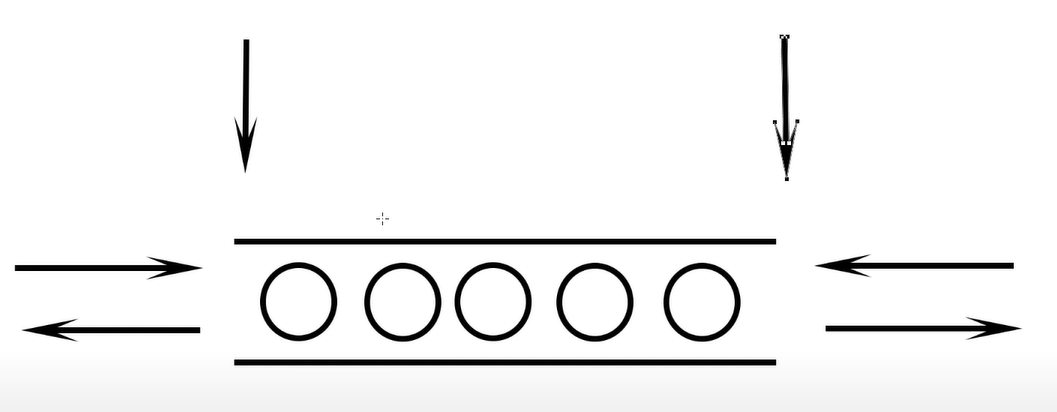
Исключаем следующий по приоритету элемент



# Deque (Дек)

Deque – double ended queue – двустороняя очередь

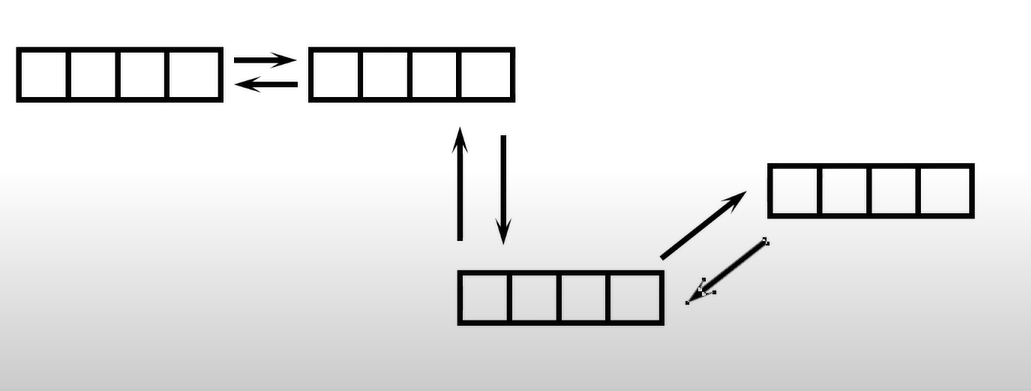
Добавить и извлечь элемент можно с двух концов, а концы очереди называют «левый конец дека» и «правый конец дека»



Также существуют «дек с ограниченым входом» или «дек с ограниченым выходом», когда с одной из сторон запрещен, соответственно, вход или выход (добавление или извлечение)

Можно **итерироваться** по элементам и **вставлять** элемент внутрь

В С++ она имеет следующий вид:



Совокупность нескольких массивов, которые связаны между собой по принципу двусвязного списка (группа раздробленых массивов). **Гибрид массива и двусвязного списка**

Когда кончается место в нашем деке, то создается еще один маленький подмасив, что позволяет нам не пересоздавать массив. Быстрая скорость доступа к элементам.