

# ТЕХНОПОЛИС |

{ Алгоритмы и структуры данных }  
Структуры данных: динамический массив, стек, очередь, дек, бинарная куча

Нечаев Михаил

- Структура данных «Динамический массив»
- Амортизированное время работы
  - Метод предоплаты
  - Метод потенциалов
- Однонаправленные, двунаправленные списки
  - Поиск, добавление элементов, слияние списков
- Абстрактные типы данных
  - «Стек»
    - Амортизированное время работы
  - «Очередь»
  - «Дек»
  - Способы реализации
- Структура данных «Двоичная куча»
  - АТД «Очередь с приоритетом»

## **Структура данных**

Структура данных (англ. data structure) —  
программная единица, позволяющая хранить и  
обрабатывать множество однотипных и/или  
логически связанных данных

## Абстрактный тип данных

Абстрактный тип данных (АТД) — это множество объектов, определяемое списком операций, применимых к этим объектам, и их свойств. Вся внутренняя структура такого типа инкапсулирована. Абстрактный тип данных определяет набор функций, независимых от конкретной реализации типа, для оперирования его значениями.

Конкретные реализации АТД называются структурами данных.



## Массив

Тип или структура данных в виде набора компонентов (элементов массива), расположенных в памяти непосредственно друг за другом.

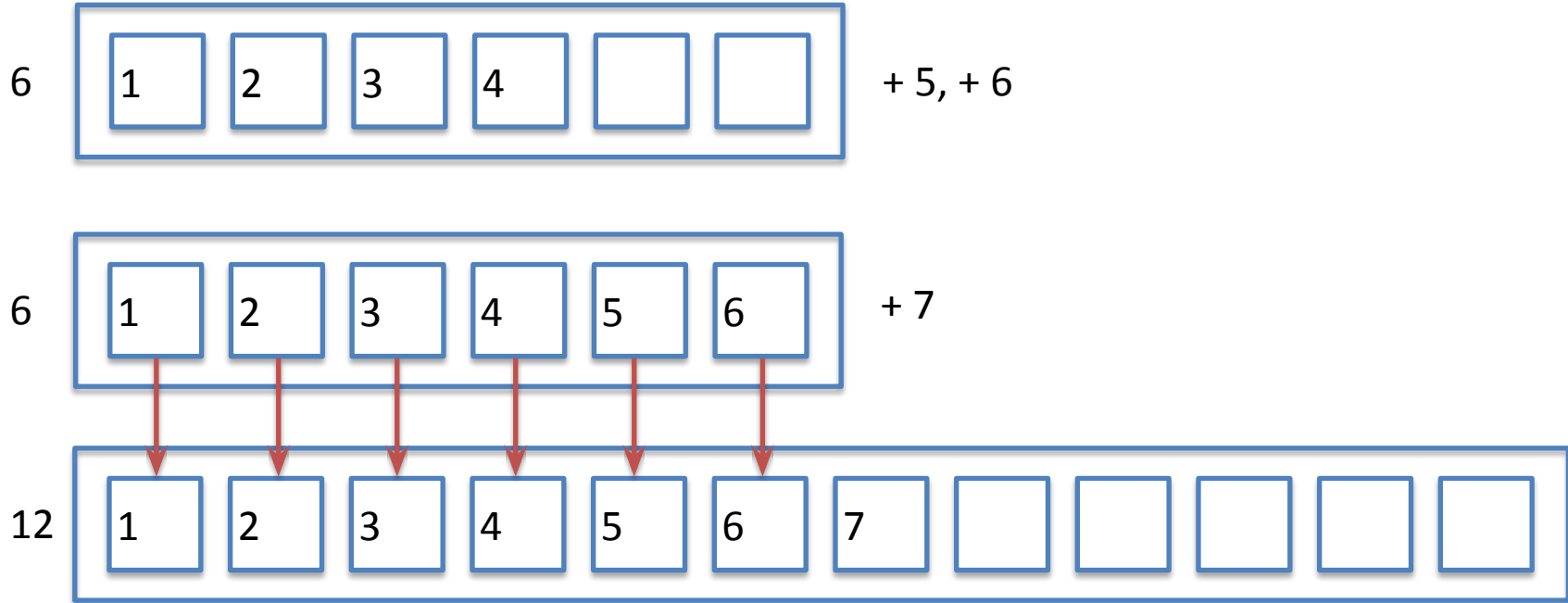
При этом доступ к отдельным элементам массива осуществляется с помощью индексации, то есть через ссылку на массив с указанием номера (индекса) нужного элемента. За счёт этого, в отличие от списка, массив является структурой данных, пригодной для осуществления произвольного доступа к её ячейкам



## **Динамический массив**

Массив — набор однотипных переменных с доступом по индексу.

Динамический массив — массив (буффер), изменяющий свой размер в зависимости от количества элементов.



## Операции

- Добавления элемента в конец
- Доступ к элементу по индексу
- Изменение элемента по индексу
- Удаление последнего элемента
- Получение размера



## Пример реализации

...

## **Время добавления/удаления элемента**

$O(1)$  — в лучшем случае

$O(n)$  — в худшем случае

А сколько в среднем случае?

Уменьшение в два раза, если в 4 раза больше элементов

## **Амортизационный анализ**

Метод подсчета времени, которое необходимо для последовательности операций над структурой данных.

Проводится анализ средней производительности в худшем случае, усредняя время по всем операциям.

## Зачем?

Например, чтобы показать, что хотя существуют «дорогие» операции, то после усреднения по всем возможным операциям, средняя стоимость может быть низкой, из-за редкого выполнения «дорогой» операции.

Оценка не учитывает вероятности

## Средняя амортизационная стоимость операций

$$S = \sum t_i / n$$

$t_1, t_2, \dots, t_n$  — время выполнения операций 1, 2, ... n, совершённых над структурой данных

Для подсчёта используются следующие методы

1. Метод усреднения (по формуле выше)
2. Метод потенциалов
3. Метод предоплаты

## Метод предоплаты

Использование  $X$  времени равносильно использованию  $X$  монет (плата за операцию)

У каждой операции своя стоимость (может быть больше или меньше реальной)

Для доказательства оценки строим учётные стоимости, что для каждой операции они составляли  $O(f(n,m))$

Тогда  $S = \sum t_i / n = n * O(f(n,m)) / n = O(f(n,m))$



## Метод потенциалов

$\Phi$  — потенциал текущего состояния структура данных

$\Phi_0, \Phi_1, \dots, \Phi_i$

$i$ -а операция стоит =  $s_i = t_i + \Phi_i - \Phi_{(i-1)}$

$n$  — количество операций,  $m$  — размер СД

$S = O(f(n,m))$ , если

1)  $\forall i : s_i = O(f(n,m))$

2)  $\forall i : \Phi_i = O(n * f(n,m))$

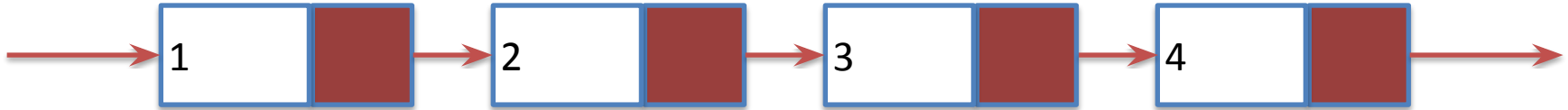
## **Время работы динамического массива**

- Метод предоплаты
- Метод потенциалов

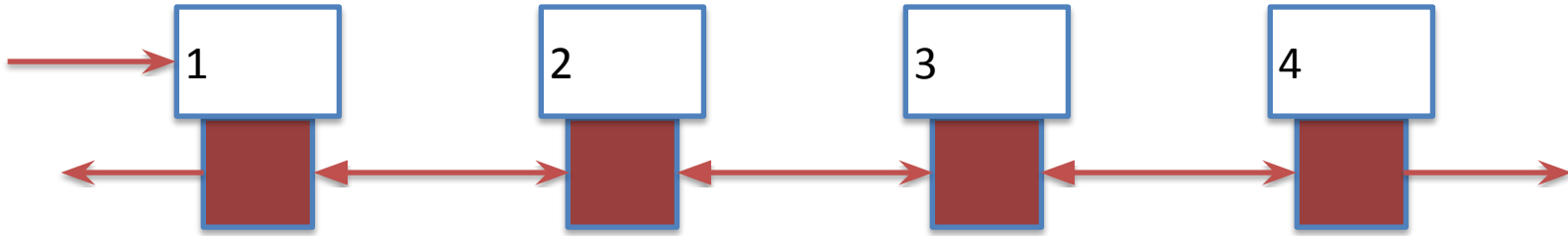
## **СВЯЗНЫЙ СПИСОК**

Динамическая структура данных, имеющая помимо своих собственных элементов, ссылки на следующий и/или предыдущий элемент списка

## Односвязный список



## Двусвязный список



## Операции

- Поиск элемента
- Вставка элемента
- Удаление элемента
- Объединение списков
- Получение размера



## Пример реализации

...

## Преимущества и недостатки

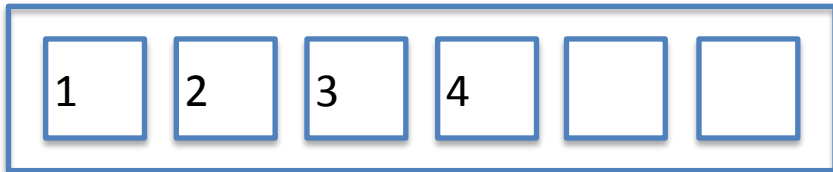
- Быстрая вставка элемента в любое место, при наличии указателя
- Быстрое удаление элемента, при наличии указателя
- Элементы в памяти находятся «разреженно»
  
- Долгий поиск нужного элемента
- Дополнительная память на ссылки
- Элементы в памяти находятся «разреженно»

## АТД Стек

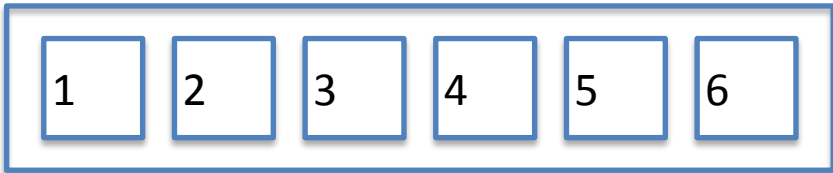
Абстрактный тип данных (или структура данных), работающий по принципу LIFO — Last In, First Out

## Операции

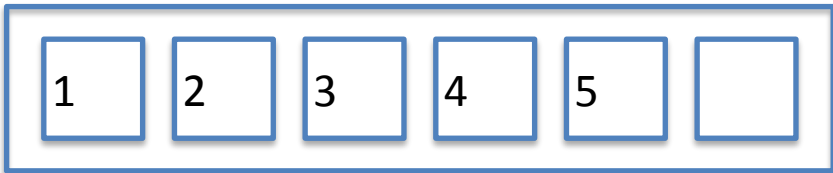
- Вставка (Push)
- Извлечение (Pop)



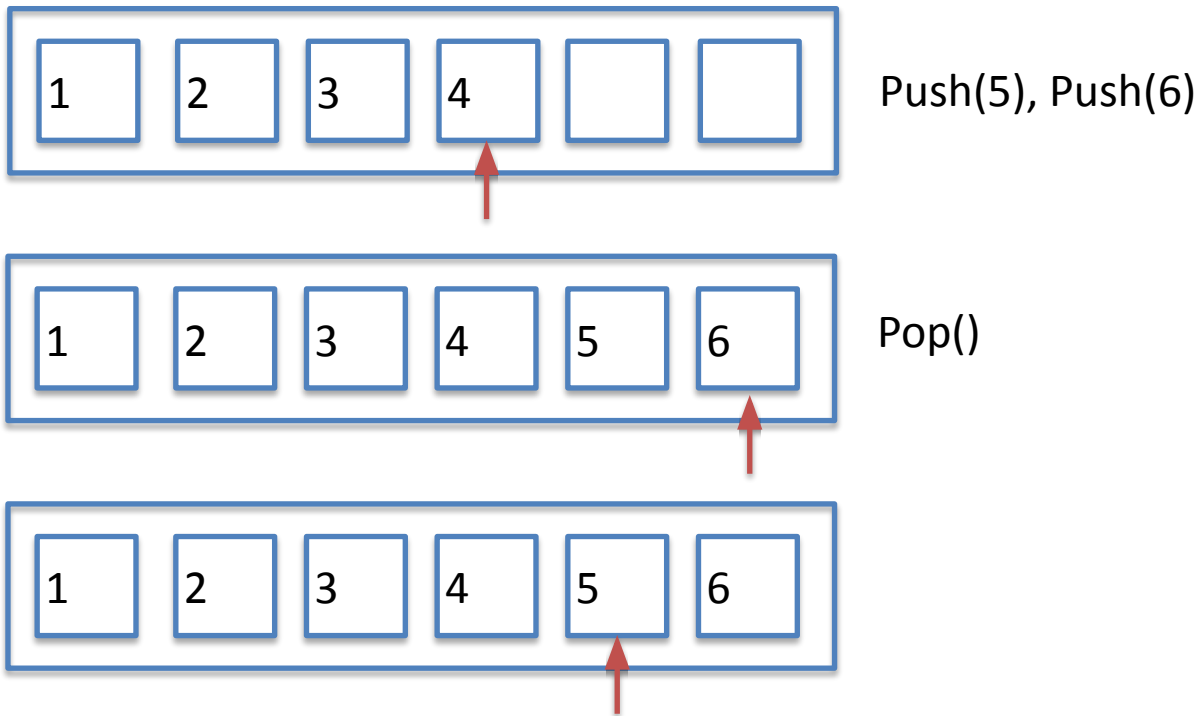
Push(5), Push(6)



Pop()

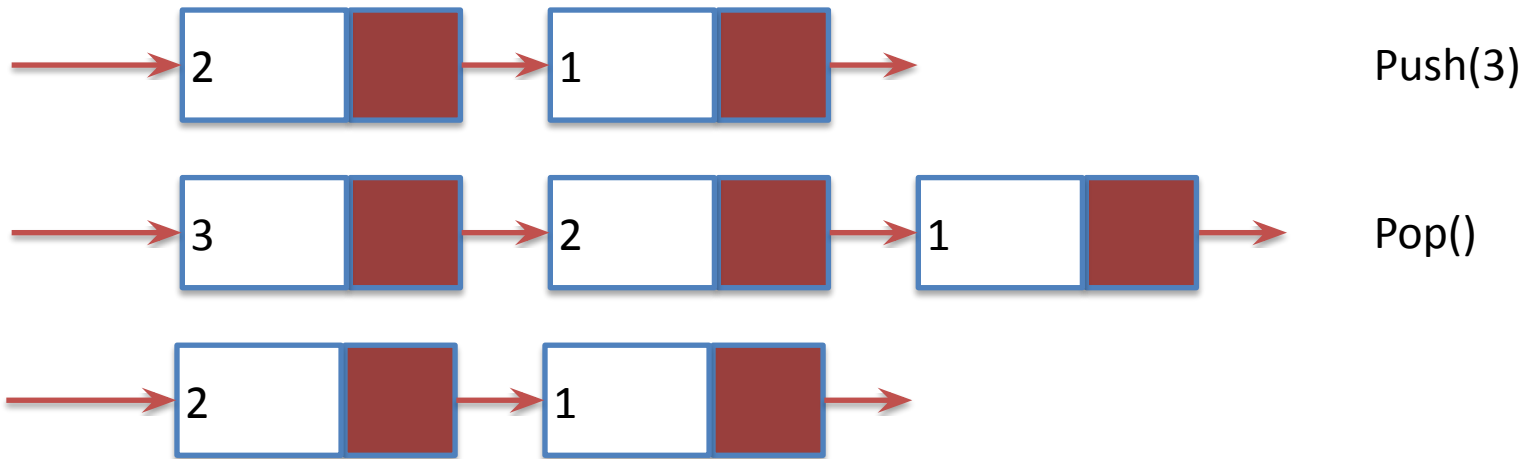


## На (динамическом) массиве





## На списке



## Пример реализации

...

## Время Push/Pop?

Push —  $O(1)$

Pop

$O(1)$  — в лучшем случае

$O(n)$  — в худшем случае

А сколько в среднем случае?

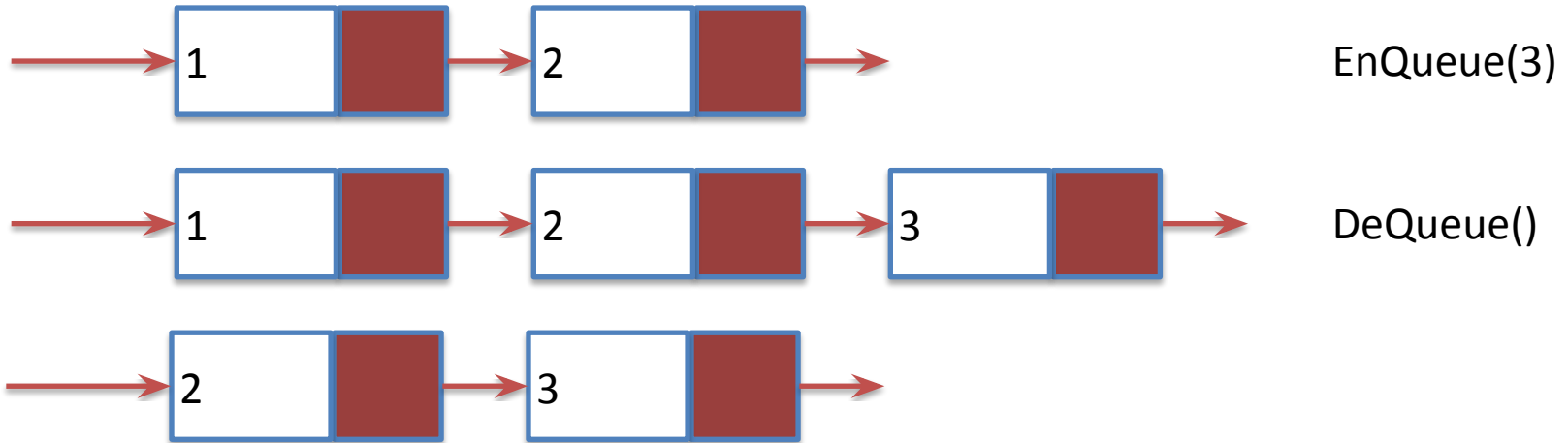
## **АТД Очередь**

Абстрактный тип данных (или структура данных), работающий по принципу FIFO — First In, First Out

## Операции

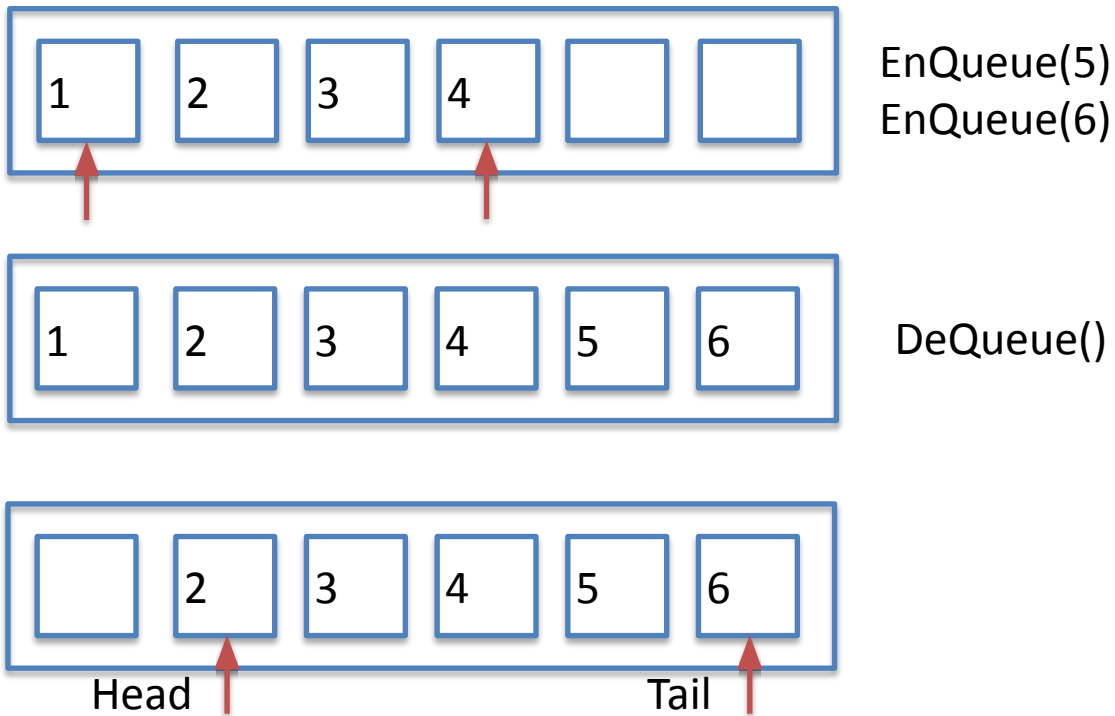
- Вставка (EnQueue)
- Извлечение (DeQueue)

## Очередь





## На (динамическом) массиве



## АТД Дэк

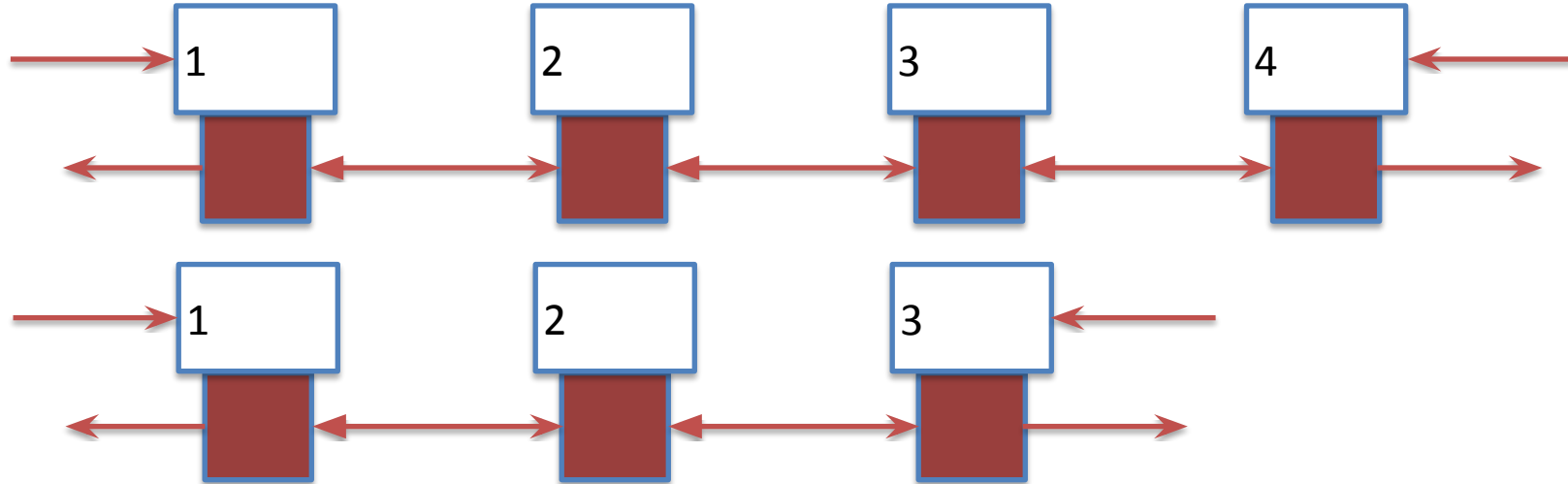
Абстрактный тип данных (или структура данных), работающий по принципу FIFO и LIFO

Можно добавлять и удалять элементы и в начале и в конце

# Операции

- Вставка в начало (PushFront)
- Вставка в конец (PushBack)
- Извлечение из начала (PopFront)
- Извлечение из конца (PopBack)

# Дэк



# Пример реализации

...

## Двоичная куча

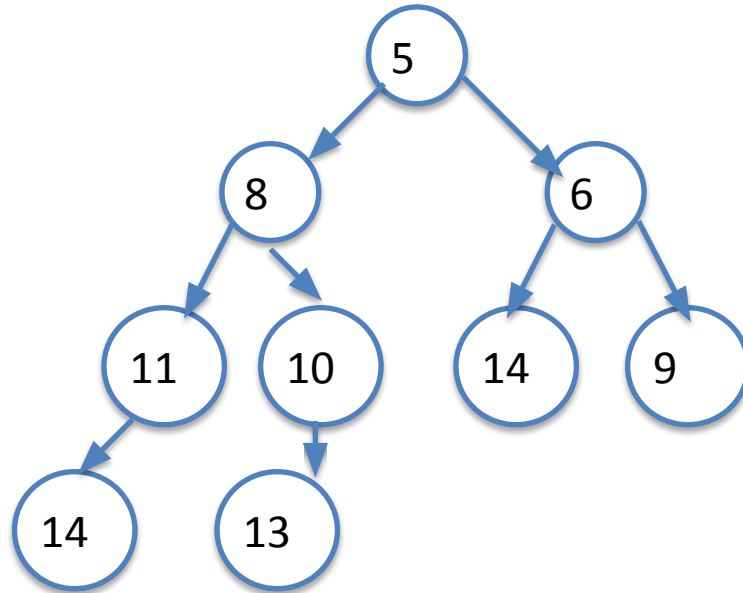
Двоичный подвешенный [связный ациклический граф — дерево],

для которого выполнены условия:

- 1 — Значение в любой вершине  $\leq (\geq)$  значений детей
- 2 — На  $i$ -ом слое, кроме последнего  $2^i$  вершин, где слои нумеруются с нуля
- 3 — Последний слой заполняется слева направо (может быть неполным)

## Двоичная куча

Глубина  $O(\log(n))$



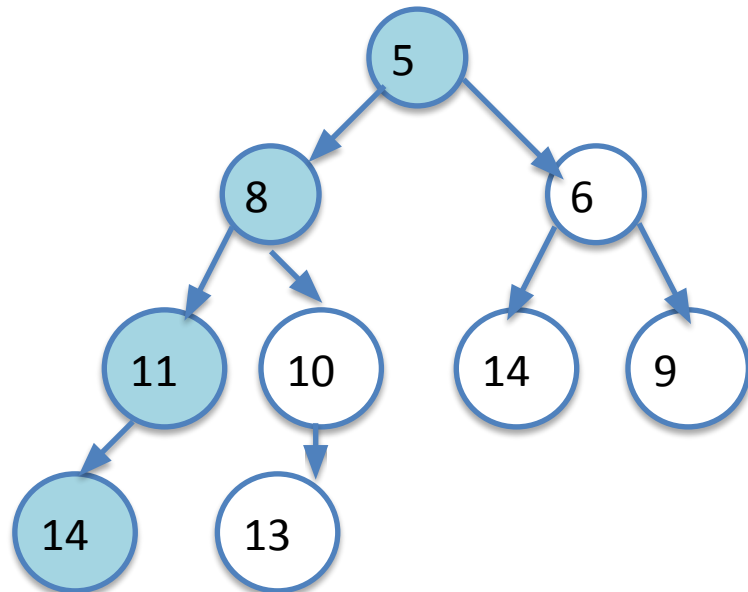
## Удобно хранить в массиве

$a[0]$  — корень

а дети  $a[i]$  -  $a[2i + 2]$  и  $a[2i + 1]$

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

5	8	6	11	10	14	9	14	13
---	---	---	----	----	----	---	----	----





## Восстановление свойств

После изменения элемента в куче, она может перестать удовлетворять условиям кучи.

Для поддержания свойств, есть:

- siftDown (просеивание вниз)

Спуск элемента, который меньше детей

- siftUp (просеивание вверх)

Подъём элемента, который больше родителей

## siftDown

Изменили элемент

Если его значение стало больше, то используем siftDown

Если элемент меньше детей — ОК

Иначе меняем элемент с наименьшим из его сыновей

+ siftDown для сына

Время работы  $O(\log n)$

## siftUp

Изменили элемент

Если его значение стало меньше, то используем siftUp

Если элемент больше родителя — ОК

Иначе меняем элемент с отцом

+ siftUp для сына

Время работы  $O(\log n)$

## Извлечение минимального элемента

Элемент в корне :)

> Получаем значение корня

+ Восстанавливаем кучу

1. Берём последний элемент
2. Ставим на место корня
3. `siftDown(0)`

Время работы  $O(\log n)$

## Добавление элемента

Вставляем элемент в конец

+ Восстанавливаем кучу  
`siftUp(elementIndex)`

Время работы  $O(\log n)$

## Построение кучи [1]

Вход — неупорядоченный массив данных

1. Первый элемент кладём в корень
2. Второй и последующие — в конец кучи
3. Запускаем siftUp для каждого добавленного

Время работы  $O(n * \log(n))$

## Построение кучи [2]

Вход — неупорядоченный массив данных

1. Представим что наш массив — это дерево
2. Запустим `siftDown` от всех вершин с детьми, начиная с предпоследнего слоя, так как листья уже «упорядочены»

До `siftDown` — поддереву упорядочено

После `siftDown` — поддереву и вершина упорядочены

Время работы  $O(n)$

## Построение кучи [2]

Доказательство времени работы

...



## **АТД Очередь с приоритетом**

Абстрактный тип данных (или структура данных),  
с операциями:

- 1 — Добавить элемент с приоритетом
- 2 — Достать элемент с наименьшим (наивысшим) приоритетом
- 3 — Посмотреть элемент на вершине

## На двоичной куче

1 — Добавить элемент с приоритетом

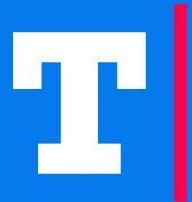
$O(\log n)$

2 — Достать элемент с наименьшим (наивысшим) приоритетом

$O(\log n)$

3 — Посмотреть элемент на вершине

$O(1)$



Спасибо за внимание!

[mikhail.nechaev@corp.mail.ru](mailto:mikhail.nechaev@corp.mail.ru)