# Содержание

# АТП Гос (ИВТ: Формалки + Оси)

**Disclaymer**: доверять этому конспекту или нет выбирайте сами

# 1. Алгоритмы и структуры данных

## 1.1. Динамический массив. Амортизационный анализ. Учётная оценка времени добавления элемента в динамический массив.

**Определение 1.1.1**: Пусть $f, g$ – произвольные функции, тогда

$$f(x) = O(g(x)) \Leftrightarrow \exists M > 0 : \exists x_0 : \forall x \geq x_0 : |f(x)| \leq M|g(x)|$$

**Определение 1.1.2**: Пусть время $n$ последовательных операций над некоторой структурой данных составит $t_1, ..., t_n$, тогда говорят, что **амортизированная стоимость** или **учётное время** операции составляет $t^* = \frac{1}{n} \sum_{i=1}^{n} t_i$.

**Замечание 1.1.1**: Далее, амортизированная сложность будет обозначаться со звездочкой, например, $O^*(1)$

**Замечание 1.1.2** (Динамический массив): Интерфейс динамического массива:

- Обращение по индексу за $O(1)$
- Добавить элемент в конец за $O^*(1)$

**Замечание 1.1.3** (Объяснение амортизационной сложности добавления): Очевидно, что при заполнении массива нам нужно делать реаллокацию. (Которая условно бесплатная, но копирование всех элементов на новое место – линейно).

Но, оказывается, если при каждой реаллокации мы будем увеличивать массив в два раза, то средняя стоимость добавления элемента в массив будет константной

Для доказательства будем использовать метод монеток.

- Пусть мы только что совершили реаллокацию – у нас есть $\frac{n}{2}$ свободного места и столько же уже добавленных элементов
- Пусть добавляем один из $\frac{n}{2}$ новых элементов – пусть его индекс $i \geq \frac{n}{2}$. Тогда, потратив монетку на добавление без реалокации, заложим по одной монетке на будущее копирование $i$-го и $i - \frac{n}{2}$-элемента.
- Таким образом, к заполнению буфера из $n$ элементов, мы заложим по монетке за каждый элемент и после новой реалокации сможем условно бесплатно скопировать их на новое место, потратив монетки.

## 1.2. Связные списки. Стек, очередь, дека и их реализации

**Замечание 1.2.1** (Список): Мы хотим от списка следующее:

| **Операция** | **Время** |
|---|---|
| Вставка в известное место | O(1) |
| Удаление из известного места | O(1) |
| Поиск | O(N) |
| Обращение по индексу | O(N) |

Наш список будет хранить цепочку из узлов, где каждый указывает на следующего за ним, а последний указывает в никуда.

Поиск и обращение по индексу требуют линейного прохода, но при этом вставка или удаление элемента – это создание узла и переприсвоение указателей.

Существует также двусвязный список – хранит в себе два указателя на узел после и позади нас. Благодаря дополнительному указателю, получаем возможность работы с обоими концами списка, не теряя в асимптотике.

**Замечание 1.2.2** (Стек): Мы хотим от стека следующее:

| **Операция** | **Время** |
|---|---|
| Вставка в начало | O(1) |
| Удаление из начала | O(1) |
| Узнать размер | O(1) |

Стек можно реализовать на односвязном списке, однако для быстрого получения размера следует завести счётчик, изменяемый при вставке/удалении.

**Замечание 1.2.3** (Очередь): Мы хотим от очереди следующее:

| **Операция** | **Время** |
|---|---|
| Вставка в начало | O(1) |
| Удаление из конца | O(1) |
| Узнать размер | O(1) |

Очередь тривиально реализуется на двухсвязном списке, однако есть способ реализовать её на двух стеках.

У нас будут два стека: входной и выходной. Вставка будет происходить в перывй, а удаление из второго. В случае удаление из пустого выходного стека требуется переложить все элементы из выходного стека в выходной (получим развёрнутый выходной стек).

Удаление из такой очереди уже будет $O^*(1)$, так как на каждый элемент хватит по 3 монетки – на его добавление, переброс в выходной стек и удаление.

**Замечание 1.2.4** (Дека): Деку иногда называют двусторонним стеком или двусторонней очередью:

| **Операция** | **Время** |
|---|---|
| Вставка в начало или в конец | O(1) |
| Удаление из начала или из конца | O(1) |
| Узнать размер | O(1) |

Тривиально реализуется на двухсвязном списке.