

# Конспекты по проге 1 семестр

## Содержание

<b>1</b>	<b>Введение</b>	<b>2</b>
1.1	Алгоритм . . . . .	2
1.2	Асимптотики . . . . .	3
1.3	Структура данных, абстрактный тип данных (интерфейс) . . . . .	3
1.4	Массив . . . . .	4
<b>2</b>	<b>Базовые структуры данных</b>	<b>5</b>
2.1	Динамический массив . . . . .	5
2.2	Односвязный и двусвязный список . . . . .	5
2.3	Стек . . . . .	5
2.4	Очередь . . . . .	5
2.5	Дек . . . . .	6
2.6	Двоичная куча. АТД Очередь с приоритетом . . . . .	7
2.7	Амортизационный анализ . . . . .	7
2.8	Персистентный стек . . . . .	8

# 1 Введение

## 1.1 Алгоритм

**Определение** Алгоритм — это формально описанная вычислительная процедура, получающая исходные данные (input), называемые также входом алгоритма или его аргументом, и выдающая результат вычисления на выход (output). Алгоритм определяет функцию (отображение):

$$F: X \rightarrow Y \quad (1)$$

$X$  - входные данные,  $Y$  - выходные

### Примеры

Вычисление числа Фибоначчи

```
int fib(int n) {
    int x = 1;
    int y = 0;

    for (int i = 0; i < n; i++) {
        x += y;
        y = x - y;
    }
    return y;
}
```

Или же через перемножение матриц за  $O(\log(n))$

$$\begin{pmatrix} F_0 & F_1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} F_n & F_{n+1} \end{pmatrix}$$

Проверка числа на простоту

Перебор до  $\sqrt{n}$   
Решето Эратосфена

Быстрое возведение в степень

```
int power (int a, int n) {
    if (n == 0) return 1;
    if (n % 2 == 0) {
        int b = power (a, n/2);
        return b*b;
    } else {
        return power (a, n - 1)*a;
    }
}
```

## 1.2 Асимптотики

$f$  ограничена сверху функцией  $g$  асимптотически

$$f(x) \in O(g(x)) \Leftrightarrow \exists(C > 0)(\forall x): \|f(x)\| \leq C\|g(x)\|$$

$f$  ограничена снизу функцией  $g$  асимптотически

$$f(x) \in \Omega(g(x)) \Leftrightarrow \exists(C > 0)(\forall x): \|f(x)\| \geq C\|g(x)\|$$

$f$  ограничена сверху и снизу функцией  $g$  асимптотически

$$f(x) \in \Theta(g(x)) \Leftrightarrow \exists(C > 0), (C' > 0)(\forall x): C\|g(x)\| \leq \|f(x)\| \leq C'\|g(x)\|$$

$g$  доминирует над  $f$  асимптотически

$$f(x) \in o(g(x)) \Leftrightarrow \forall(C > 0)(\forall x): \|f(x)\| < C\|g(x)\|$$

$f$  доминирует над  $g$  асимптотически

$$f(x) \in \omega(g(x)) \Leftrightarrow \forall(C > 0)(\forall x): \|f(x)\| > C\|g(x)\|$$

$f$  эквивалентна  $g$  асимптотически при  $n \rightarrow n_0$

$$f(n) \sim g(n) \Leftrightarrow \lim_{n \rightarrow n_0} \frac{f(n)}{g(n)} = 1$$

Примеры

$$O(1) = O(2)$$

$$O(f(n))O(g(n)) = O(f(n)g(n))$$

$$f(n) \subset g(n) \Rightarrow O(f(n) + g(n)) = O(g(n))$$

## 1.3 Структура данных, абстрактный тип данных (интерфейс)

**Определение** Структура данных - программная единица, позволяющая хранить и обрабатывать данные. Для взаимодействия предоставляет интерфейс.

Примеры

`int a[n];`

`std::pair<int, int>`

Абстрактный тип данных (АТД) - тип данных, который скрывает внутреннюю реализацию, но предоставляет весь интерфейс для работы с данными, а также возможность создавать элементы этого типа. Абстрактный тип данных реализуется с помощью структуры данных.

Пример

Стек, реализованный через массив

Стек - АТД, массив - структуры данных

## 1.4 Массив

**Определение** Массив - структура данных, хранящая набор значений (элементов), которые идентифицируются по индексу или набору индексов.

Динамический массив - массив, размер которого может изменяться во время выполнения программы.

Линейный поиск - поиск по массиву за  $O(n)$

В отсортированном массиве поиск возможен за  $O(\log(n))$  с помощью бинарного поиска

```
int binarySearch(const int *a, int n, int item) {
    int leftPtr = -1, rightPtr = n;

    while (rightPtr - 1 > leftPtr) {
        int middlePtr = (leftPtr + rightPtr)/2;
        if (item > a[middlePtr]) {
            leftPtr = middlePtr;
        } else if (item < a[middlePtr]) {
            rightPtr = middlePtr;
        } else {
            return middlePtr;
        }
    }
    return rightPtr;
}
```

## 2 Базовые структуры данных

### 2.1 Динамический массив

+ : random access operator

– : нельзя удалить/вставить в середину за  $O(1)$

### 2.2 Односвязный и двусвязный список

+ : удаление и вставка в любое место

– : поиск/последовательный доступ дорог

Односвязный список имеет ссылку только на следующий узел.

Двусвязный список имеет ссылку на следующий и предыдущий узлы.

Операции со списками:

Поиск элемента ( $O(n)$ )

Вставка - смена указателей ( $O(1)$ )

Удаление - смена указателей ( $O(1)$ )

Объединение (при условии, что храним указатель на последний элемент) - смена указателей ( $O(1)$ )

### 2.3 Стек

Сертификат - Last In First Out

Операции:

Добавление в конец ( $O(1)$ )

Удаление с конца ( $O(1)$ )

Можно реализовать с помощью:

Динамического массива

Списка

Для хранения в массиве можно использовать указатель на последний элемент и сдвигать его в зависимости от операции.

Чтобы поддерживать минимум в стеке, достаточно хранить не элемент, а пару вида: значение элемента, минимальный элемент в стеке на момент вставки нашего элемента.

### 2.4 Очередь

Сертификат - First In First Out

Операции:

Добавление в конец

Удаление из начала

Можно реализовать с помощью:

Динамического массива

Списка

Двух стеков

### Циклическая очередь в массиве

Реализация циклической очереди в массиве основана на хранении двух указателей: на первый элемент в очереди и на последний. Тогда удаление/вставка элементов будет связана со сдвигом указателя на первый/последний элемент по формуле  $\text{newPtr} = (\text{ptr} + 1) \% \text{size}$

### Хранение очереди в списке

Для реализации на односвязном списке достаточно хранить указатель на первый и последний элемент, удаление/вставка производятся сменой указателей.

### Представление очереди в виде двух стеков

Для реализации очереди на двух стеках достаточно вставлять элементы в один стек, а забирать из другого. В случае, если второй стек пуст, перемещать все элементы из первого во второй.

### Время извлечения элемента

Для операции push возьмём 3 монеты: для самого push'a, резерв на pop из первого стека и резерв на pop из второго. Тогда учётная стоимость pop'a из второго стека будет равна 0, т.к. можно использовать оставшиеся монеты. Таким образом, каждая операция за  $O(1)$

### Поддержка минимума в очереди

Для поддержки минимума необходимо поддерживать минимум в каждом из стеков, тогда минимум в очереди - меньший минимум стеков.

## 2.5 Дек

Список элементов, в котором можно вставлять и удалять элементы с конца и начала.

Операции:

Вставка в конец

Извлечение из конца

Вставка в начало

Извлечение из начала

Можно реализовать с помощью:

Динамический массива

Двусвязного списка

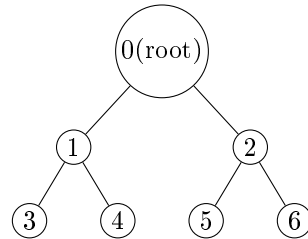
Хранение дека в массиве

Хранение аналогично очереди, только указатели могут сдвигаться как вперёд, так и назад ( $\text{newPtr} = (\text{ptr} - 1) \% \text{size}$ )

Хранение дека в списке

Храним аналогично очереди, но используем двусвязный список.

## 2.6 Двоичная куча. АТД Очередь с приоритетом



**Определение** Двоичное подвешенное дерево, которое удовлетворяет свойствам:

Значение в вершине  $\leq$  ( $\geq$  для максимума в корне) значению в потомке  
На  $i$ -ом слое  $2^i$  вершин (кроме, возможно, последнего). Глубина кучи  $\log n$   
Последний слой заполняется слева направо

Удобно хранить бинарную кучу в массиве:

$a[0]$  - корень

$a[i] \rightarrow (a[2i + 1], a[2i + 2])$  - потомки

**Операции** Вставка за  $O(\log n)$ :

Вставляем в свободное место

Рекурсивно поднимаем элемент, если он меньше (больше) родителя (siftUp)

Удаление за  $O(\log n)$ :

Удаляем минимум

Вставляем последний элемент в корень

Рекурсивно меняем элемент с минимальным (максимальным) потомком (siftDown)

**Очередь с приоритетом** - абстрактный тип данных, который поддерживает следующие операции:

push - добавить в очередь элемент с определенным приоритетом

pop - удалить из очереди элемент с наивысшим приоритетом

top - посмотреть элемент с наивысшим приоритетом (необязательная операция)

## 2.7 Амортизационный анализ

**Определение** - метод подсчёта времени, требуемого для выполнения последовательности операций над структурой данных. При этом время усредняется по всем выполняемым операциям, и анализируется средняя производительность операций в худшем случае.

**Средняя амортизационная стоимость**

$$x = \frac{\sum_{i=0}^n t_i}{n}$$

$t_i$  - время выполнения  $i$ -ой операции

**Амортизированное (учетное) время добавления элемента в динамический массив**  
Стоимость операции  $\text{add}(\text{item})$

Для каждой операции  $\text{add}(\text{item})$ , для которой не требуется расширение массива, будем использовать три монетки: одна для самой вставки, две в резерв. Одну из резерва положим к вставленному элементу с индексом  $i$ , а вторую к элементу с индексом  $i - \frac{n}{2}$ , где  $n$  - размер массива

Когда массив заполнится, у каждого элемента будет одна монетка в резерве, которую мы сможем использовать для копирования в новый массив размером  $2n$

**Метод потенциалов** Пусть  $\Phi$  - потенциал

$\Phi_0$  - изначальное значение

$\Phi_i$  - состояние после  $i$ -ой операции

Тогда стоимость  $i$ -й операции  $a_i = t_i + \Phi_i - \Phi_{i-1}$

Пусть  $n$  - количество операций,  $m$  - размер структуры данных, тогда  $a = O(f(n, m))$ , если:

$$\begin{aligned}\forall i: a_i &= O(f(n, m)) \\ \forall i: \Phi_i &= O(n(f(n, m)))\end{aligned}$$

Доказательство

$$a = \frac{\sum_{i=1}^n t_i}{n} = \frac{\sum_{i=1}^n a_i + \sum_{i=0}^{n-1} \Phi_i - \sum_{i=1}^n \Phi_i}{n} = \frac{n \cdot O(f(n, m)) + \Phi_0 - \Phi_n}{n} = O(f(n, m))$$

**Метод бух. учёта** Пусть операция стоит некоторое число монет.

Тогда для каждой операции мы можем взять монет с запасом, чтобы хватило на следующие возможные операции (пример с динамическим массивом)

Если монет хватило на все операции, то наше предположение о стоимости каждой операции (то, что мы взяли с запасом) верно

(Излагаю в неформальном стиле)

## 2.8 Персистентный стек

Стек, который хранит все свои состояния

Операции:

Вставка:  $\text{push}(i, x) \rightarrow j$ , где  $i$  - конкретное состояние,  $x$  - элемент, который нужно вставить,  $j$  - новое состояние после вставки. При вставке создаётся новое состояние стека, где хранится вставленный элемент и ссылка на состояние, в которое его вставили.

Удаление:  $\text{pop}(i) \rightarrow j$ . При удалении  $i$ -ого состояния идём с "родителем"  $i$ -ого состояния и подцепляем его копию к "деду"  $i$ -ого состояния.

В итоге имеем доступ к любой версии стека за  $O(1)$  времени и  $O(n)$  памяти.

Можно реализовать с помощью:

Массива

Списка