Урок 1

# Общие сведения об алгоритмах и структурах данных

## Введение в алгоритмы и структуры данных

# Введение

Структуры данных и алгоритмы являются неотъемлемыми компонентами любой программы. Поэтому знание основных структур данных и алгоритмов помогают программисту выбрать оптимальный путь для решения конкретной задачи. В данном курсе будут рассмотрены основные типы структур данных и алгоритмы для работы с ними. Для работы со структурами данных будут разработаны алгоритмы, связанные с поиском, удалением, вставкой, выводом, прохождением и сортировкой данных. Рассмотрим, как осуществляется расчет эффективности алгоритма. Узнаем что такое О-большое.

Программируя на Java, не стоит слишком сильно задумываться о реализации структур данных. Java – это не просто язык программирования, а огромная платформа, которая включает в себя большое количество библиотек, в том числе и реализации тех структур данных, которые будут рассмотрены в этих уроках. Но все же понимание работы этих структур необходимо для правильного их выбора в ходе решения конкретной задачи.

Коротко о том, что будем изучать. В первом уроке рассказываются общие сведения об алгоритмах и структурах данных. Приводятся краткие сведения об ООП, т.к. этот подход к программированию мы будем использовать. Рассмотрим примитивные и ссылочные типы данных. Научимся вычислять эффективность алгоритма с помощью О-синтаксиса.

Второй урок посвящен такой структуре данных как массив. Будут рассмотрены алгоритмы поиска, вставки и удаления элемента. Узнаем про сдвиги в массивах. Также рассмотрим примитивные сортировки методом пузырька, вставок, выбора.

В третьем уроке перейдем к ADT, так называемым абстрактным структурам данных, о которых поговорим чуть позже. Начнем со стека и очереди. Посмотрим, где они могут использоваться и как организуются.

В четвертом уроке поговорим о связанных списках. Рассмотрим различия между списками и массивами для хранения элементов данных. Реализуем очередь и стек на основе связанного списка.

Пятый урок посвящен интересному подходу в программировании, который называется рекурсией. В этом уроке не будет рассматриваться никакая новая структура данных. Попробуем изменить некоторые алгоритмы, которые раньше решались через циклы. Также рекурсия часто используется в деревьях, но о них чуть позже.

Шестой урок посвящен деревьям, а точнее двоичным деревьям. В этом уроке будет рассмотрена простейшая древовидная структура – несбалансированное дерево двоичного поиска. Рассмотрим, вставку, удаление и обход таких деревьев.

Седьмой урок посвящен графам. Рассмотрены задачи поиска в глубину и ширину.

В восьмом уроке рассматриваются хеш-таблицы. Очень интересная структура, в которой вставка, поиск и удаление происходит за время О(1). Что это за время узнаем немного позже.

# Понятие о структурах данных и алгоритмах

Под структурой данных понимается способ хранения данных в памяти компьютера. Это могут быть массивы, связанные списки, деревья, стеки, очереди и т.д.

Перед тем как говорить о структурах данных, давайте посмотрим, как работает память компьютера. Рассмотрим это на примере маленьких коробочек, в которые мы перекладываем вещи на время переезда.

Представим, что в коробку можно положить только одну вещь. Перевезти необходимо восемь вещей, так что для этого понадобится восемь коробок. В принципе, можно сказать, что так работает память компьютера, только вместо коробок у памяти есть ячейки. Но чтобы понимать в какую коробку мы положили конкретную вещь, ее можно подписать, также и у памяти компьютера, у каждой ячейки есть свой адрес.

Книги

адрес:fe001aab

Зубная паста

адрес:ee4543aa

Каждый раз, когда мы хотим что-то сохранить в памяти компьютера, мы запрашиваем у него место в памяти, а он выдает адрес, где можно сохранить данные. Так какие же данные можно сохранить в памяти компьютера?

## Ссылочные и примитивные типы данных

В Java типы данных можно разделить на два вида – это примитивные и ссылочные. Разница между этими типами в способе хранения данных. Рассмотрим простой пример, в котором объявляются две переменные.

```
int a;
Person p1;
```

В первой строке объявляется примитивная переменная типа int. Когда этой переменной будет передано какое-либо значение, то оно будет храниться в области памяти, которая связана с именем a.

Во второй строке переменная p1 является ссылочной и область памяти, которая ассоциируется с p1 и не содержит данных, а хранит только адрес памяти, в которой хранится объект типа Person.

Пока области памяти p1 не будет присвоен объект, в ней будет храниться ссылка на специальный объект null. Таким же образом область памяти «a» не содержит значения, пока оно не присвоено. Если, в программе обратиться к переменной «a» до присвоения ей значения, то компилятор выдаст ошибку.

```
public static void main(String[] args) {
        int a;
        Person p1;

        System.out.println(a);


}



Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - variable a might not
have been initialized

        at com.structalgorithm.lesson1.Lesson1App.main(Lesson1App.java:21)
```

Компилятор выдает ошибку о том, что переменная «a» не инициализирована.

Как мы могли заметить, в отличие от C++ в Java для работы со ссылочными типами данных не используются указатели, что делает код намного проще для понимания. Но это только визуально, указатели не используются явно, они скрыты от программиста, но они есть и все ссылочные типы их используют.

Что касается оператора присваивания, когда речь идет о ссылочных типах, то копируется только адрес.

```
p1 = p2;
```

Теперь имена p1 и p2, относятся к одному и тому же объекту, то есть ссылаются на него.

Так как ссылочные типы данных – это объекты, то они должны быть созданы с помощью специального оператора «new». Оператор «new» возвращает ссылку на объект.

```
Person p1 = new Person();
```

Возвращает ссылку на адрес в памяти. Не указатель, как в C++, который содержит адрес в памяти, где хранится объект, а ссылку на адрес. Поэтому программист не знает фактический адрес «p1». А как же освободить память, которая была выделена под объект, если мы не знаем, где расположены данные. А в Java это и не нужно, так как Java периодически просматривает все блоки памяти, которые были выделены с помощью оператора «new» и проверяет, есть ли на них использующиеся ссылки. Если ссылок нет, то память чистится. Это называется уборкой мусора. В Java реализован специальный помощник, занимающийся уборкой мусора автоматически, который называется сборщик мусора.

Рассмотрим, как передаются аргументы ссылочных и примитивных типов данных в методы.

```
public static void main(String[] args) {
        int a = 5;
        changeValue(a);
        System.out.println(a);
}

public static void changeValue(int a){
        a = 10;

}
```

В этом примере создается переменная примитивного типа данных int. Когда аргументом передается переменная примитивного типа, то создается ее копия. Поэтому переменная «a», объявленная в методе main и переменная a, переданная в метод changeValue – это две разные переменные. Выполнив листинг выше, получим следующий ответ.

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ lesson2 ---

5

------------------------------------------------------------------------

BUILD SUCCESS

------------------------------------------------------------------------

Total time: 0.582s

Finished at: Sun Nov 12 12:19:48 MSK 2017

Final Memory: 5M/119M
```

Сделаем тоже самое для переменной ссылочного типа данных.

```
public static void main(String[] args) {
        Person p1 = new Person("Artem");
        changeName(p1);
        System.out.println(p1.name);
}

public static void changeName(Person p1){
        p1.name = "Dmitriy";
}
```

Создадим объект person, в котором есть одно поле типа String и присвоим ему значение «Artem». Передадим переменную p1 в метод changeName. Запустим программу и получим следующий результат.

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ lesson2 ---

Dmitriy

------------------------------------------------------------------------

BUILD SUCCESS

------------------------------------------------------------------------

Total time: 0.582s

Finished at: Sun Nov 12 12:19:48 MSK 2017

Final Memory: 5M/119M
```

В результате выполнения программы имя объекта p1 изменилось. Так как в качестве аргумента было передано не значение p1, а ссылка на адрес памяти, где хранятся данные p1.

Немного поговорим о равенстве. Для примитивных типов, где сравниваются значения, все выглядит очень просто.

```
public static void main(String[] args) {

	int a = 5;
	int b = 5;

	if (a == b){
		System.out.println("a и b равны");
	} else {
		System.out.println("a и b не равны");
	}


}
```

Когда сравниваются два примитивных типа, то сравниваются их значения. Поэтому, сравнивая значения 5 и 5, будет выведена надпись, что «a и b равны».

Когда речь идет о ссылках на объекты, то такое сравнение вернет ложный результат.

```
public static void main(String[] args) {

	Person a = new Person("Artem");
	Person b = new Person("Artem");

	if (a == b){
		System.out.println("a и b равны");
	} else {
		System.out.println("a и b не равны");
	}


}
```

Почему так происходит. Ранее было сказано, что оператор new возвращает ссылку на адрес в памяти, где физически будет расположен объект. Так как мы дважды вызываем оператор new, то вернутся две разные ссылки. Поэтому, сравнивая ссылки, мы и получаем в качестве результата false.

Для сравнения ссылочных типов данных в Java используют метод equals.

```
public static void main(String[] args) {

	Person a = new Person("Artem");
	Person b = new Person("Artem");

	if (a.equals(b)){
		System.out.println("a и b равны");
	} else {
		System.out.println("a и b не равны");
	}


}
```

Метод equals необходимо переопределить. В данном случае, в классе Person.

```
@Override
public boolean equals(Object obj) {
	if (obj == this) return true;
	if (!(obj instanceof Person)){
		return false;
	}
	Person p = (Person) obj;

	return p.name == this.name;

}
```

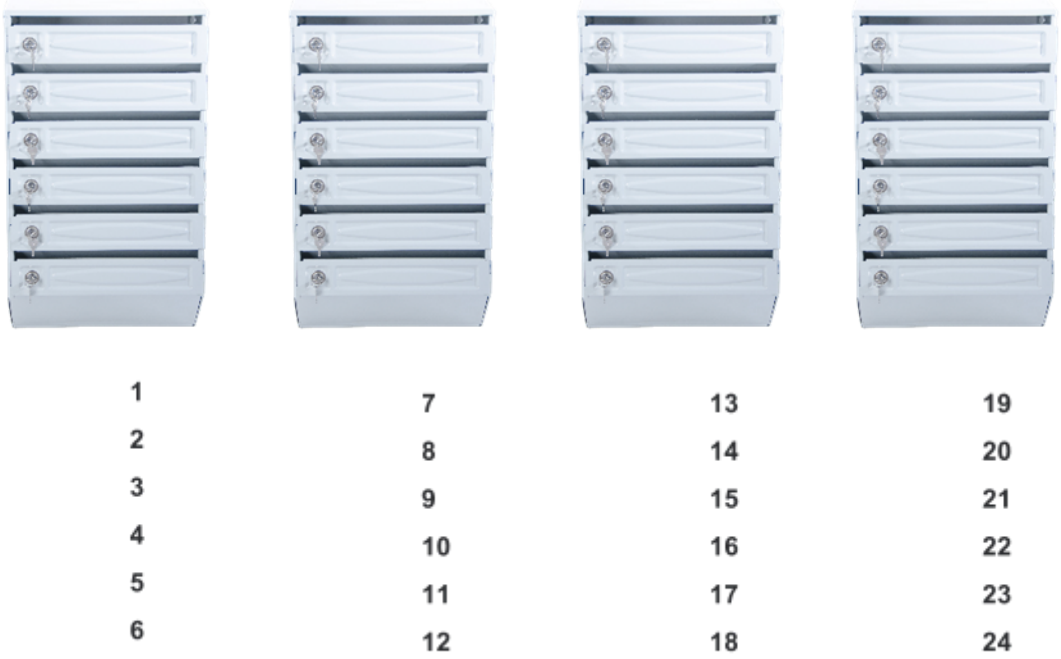
В таблице перечислены все примитивные типы данных, которые есть в Java. Те типы, которые отсутствуют в таблице, являются ссылочными.

| Наименование | Размер в битах | Диапазон значений |
|---|---|---|
| int | 32 | от -2 147 483 648 до 2 147 483 647 |
| Float | 32 | от $10^{-38}$ до $10^{38}$ с 7 значащими цифрами |
| double | 64 | от $10^{-308}$ до $10^{308}$ с 15 значащими цифрами |
| Byte | 8 | от -128 до 127 |
| Short | 16 | от -32 768 до 32 767 |
| Long | 64 | от -9 223 372 036 854 775 808 до +9 223 372 036 854 775 807 |
| Char | 16 | от `u0000` до `uFFFF` |
| Boolean | 1 | True, false |

## Немного про массивы

Помимо обычных данных, которые хранят одно значение, может возникнуть ситуация, когда необходимо хранить множество данных. Для этого можно использовать массивы.

Допустим, почтальон приносит почту в многоквартирный дом. У него много писем и корреспонденции для жильцов. Если он положит всю корреспонденцию в одну коробку, то жильцам будет сложно найти свою почту. То ли дело использовать для каждой квартиры почтовые ящики и разложить письма и газеты в них.




Почтовые ящики все пронумерованы номерами квартир. Теперь жильцам просто найти свою почту. Для этого необходимо открыть ящик со своим номером. Так и работают массивы. Для них в памяти

выделяется место для хранения определенного количества элементов и к каждому из этих элементов можно обратиться по индексу. Подробно массивы будут рассмотрены в следующем уроке.

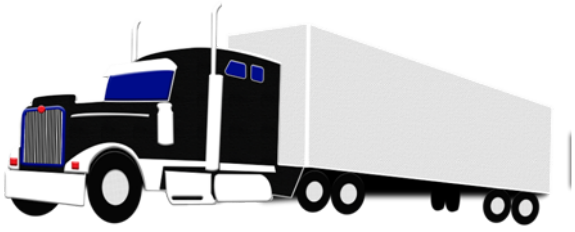
Все остальные структуры, такие как связанные списки, стеки, очереди, деревья, графы, хеш-таблицы и т.д, относятся к абстрактным типам данных.

## Абстрактный тип данных

Что такое типы данных? Это элементы данных, которые обладают определенными характеристиками и операциями, которые можно совершать над этими данными. Например, примитивный тип данных int. Если говорить про элемент данных, то это целое число, которое находится в диапазоне от -2 147 483 648 до 2 174 483 647. Над этим числом можно проводить операции сложения, вычитания, деления, умножения и т.д. Можно сказать, что операции, которые над ним проводятся, являются неотъемлемой частью его смыслового содержания. Чтобы понять смысл типа данных, нужно посмотреть какие операции над ним можно выполнять.

С появлением ООП стало возможным создавать свои типы данных. В качестве каркаса типа данных стали выступать классы, которые содержат поля, являющиеся характеристиками, и методы, которые являются операциями над данными. Например, класс для представления даты и времени, или дробных чисел.

Хорошо, с типом данных разобрались. Что же такое абстракция? Абстракция – это отделение логики работы какой-либо структуры от конкретной реализации. Можно сказать, что абстракция – это некая модель поведения какого-либо объекта. Например, мы производим автомобили: грузовые, легковые, автобусы. Какие общие характеристики можно выделить у этих машин?



К общим характеристикам машин можно отнести:

- ✓ количество колес;
- ✓ объем двигателя;
- ✓ массу;
- ✓ количество посадочных мест;

- ✓ и т.д.

К общим операциям можно отнести:

- ✓ завести двигатель;
- ✓ остановить двигатель;
- ✓ нажать на педаль газа;
- ✓ нажать на педаль тормоза;

и т.д.

Описав основные характеристики для машин, можно сказать, что создана некая абстракция, которая характеризует машины. Теперь на основе этой абстракции можно создать конкретные классы с реализацией для автобусов, грузовых и легковых машин.

Таким образом, можно сказать, что в объектно-ориентированном программировании абстрактным типом данных называется структура, которая не зависит от конкретной реализации, а содержит характеристики (поля класса) и операции над ними (методы).

В данном курсе кроме массивов мы будем создавать структуры данных, которые относятся к абстрактным типам данных. Для простоты мы не будем использовать интерфейсы и модификаторы доступа private.

Перед тем как переходить к алгоритмам, хотелось бы сказать несколько общих слов о структурах данных. Мы рассматривали пример с массивом, когда почтальон разносил почту. Можно ли как-то заменить массив для хранения набора данных?

Давайте рассмотрим следующий пример. Представьте, что на работе есть список дел для целого отдела. Сотруднику Иванову дали несколько заданий на месяц, которые он должен выполнить в определенном порядке.

Январь 2018

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 Ремонт компьютера | 2 Установка антивируса | 3 Почистить компьютер | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |

Представим, что этот календарь – это память компьютера. Дни, которые закрашены градиентом, заняты другими специалистами. Что будет, если хранить эти данные в массиве? Перед сотрудником Ивановым поставили новую задачу. Он должен установить операционную систему на компьютер пользователя. Как добавить эту задачу в календарь, если для массива выделяется конкретная область, а каждый элемент вставляется справа от предыдущего? В нашем случае четвертая ячейка занята, и записать новую задачу не представляется возможным.

Давайте внимательно посмотрим на наш календарь. Его можно представить не виде таблицы, а в виде списка дел.


Отлично. Теперь мы можем добавлять сколько угодно дел, если будем использовать список. Но как отделить в памяти компьютера дела Иванова от дел Петрова? Давайте сделаем список связанным и для каждого элемента сделаем ссылку на следующий элемент. Тогда они могут располагаться в памяти где угодно, так как связаны друг с другом ссылкой.

Структура такого типа данных будет обладать следующими характеристиками:

- ✓ Данные (в нашем случае задача);
- ✓ Ссылка в памяти компьютера на следующий элемент;

Теперь память нашего компьютера будет выглядеть следующим образом.


Отлично, теперь мы знаем две структуры, которые могут хранить наборы данных. Чем они отличаются, кроме того, что по-разному хранят свои значения, мы узнаем в уроке 2 и 3.

Давайте рассмотрим еще несколько структур данных, о которых подробно будет рассказано в следующих уроках.

Представим, что мы разработали свою социальную сеть. Эта сеть растет с каждым днем и у нас уже зарегистрировано более десяти миллионов человек. С каждым днем количество зарегистрированных пользователей становится больше и больше. При прохождении процедуры регистрации в нашей сети, есть одно условие, каждый новый пользователь должен иметь уникальный логин. Т.е. наша программа должна получить логин каждого зарегистрированного пользователя и сопоставить с логином регистрирующегося пользователя и если логин совпадает вывести сообщение об ошибке.

Попробуем использовать массив для хранения информации о логинах.

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 Viktor | 2 Main | 3 Worker | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 Builder | 13 | 14 |
| 15 Boiler | 16 | 17 | 18 | 19 Warrior | 20 | 21 |
| 22 | 23 | 24 Zeus | 25 | 26 | … | 10000000 philip |

Новый пользователь заходит на сайт и хочет зарегистрироваться под логином Philip.

Наша программа начинает перебирать все элементы массива:

Первый элемент – Viktor != Philip

Второй элемент – Main != Philip

Третий элемент – Worker != Philip

….

Элемент десять миллионов – Philip == Philip

Таким образом, проделав десять миллионов операций, система выдает ошибку, т.к. пользователь с таким логином существует.

Как же упростить поиск и уменьшить количество операций. Можно использовать абстрактный тип данных, который называется двоичное дерево, о котором мы поговорим в 7 уроке.

Про структуры данных немного поговорили, а что же касается алгоритмов?

Алгоритмы обеспечивают выполнения различных задач над структурами данных, такие как поиск, вставка, удаление и т.д.

Как правило, конкретная структура данных выбирается в зависимости от решаемой задачи. Посмотрим в таблице основные характеристики структур данных, которые будут рассмотрены в дальнейших уроках.

| Структура | Достоинства | Недостатки |
|---|---|---|
| Массив | Быстрая вставка, быстрый доступ | Медленный поиск, медленное удаление, фиксированный размер |
| Упорядоченный массив | Поиск выполняется быстрее, чем в обычном массиве | Медленная вставка и удаление, фиксированный размер |
| Стек | Быстрый доступ в порядке «Последним пришел, первым вышел» | Медленный доступ к другим элементам |
| Очередь | Быстрый доступ в порядке «Первым пришел, первым вышел» | Медленный доступ к другим элементам |
| Связанный список | Быстрая вставка, быстрое удаление | Медленный поиск |
| Двоичное дерево | Быстрый поиск, вставка, удаление | Сложный алгоритм удаления |
| Граф | Моделирование реальных ситуаций. | Некоторые алгоритмы сложны и медленны |
| Хеш-таблица | Очень быстрый доступ (если ключ известен) | Медленный доступ(если ключ неизвестен), память используется неэффективно. |

# Как определить эффективность структуры данных

Определить точное время выполнения конкретного алгоритма и структуры данных очень сложно. В настоящее время существует множество различных вычислительных устройств, которые обладают разной вычислительной мощностью. Например, говоря об алгоритмах поиска, их можно выполнять как на мобильных устройствах, так и на огромных суперкомпьютерах, которые используют различные методы параллельной обработки данных. Таким образом, время, затраченное на поиск, будет разное. Как же тогда оценивать эффективность?

Вернемся к примеру, в котором происходил поиск и сравнения логинов пользователя для регистрации в социальной сети.

В первом случае мы использовали линейный поиск, во втором случае, который касается деревьев, используется бинарный поиск. О линейном и бинарном поиске поговорим во втором уроке. А сейчас поверим на слово и посмотрим на рисунок ниже.


Если наш список пользователей состоит из 100 элементов, то на поиск может потребоваться 100 операций сравнения. Если список будет состоять из 4000000000 элементов, то на поиск может потребоваться 4000000000 операций сравнения. Такое время называется линейным.

Если использовать бинарный поиск, то на поиск пользователя в списке из 100 элементов, может потребоваться 7 операций сравнения, а на поиск логина в списке из 4000000000 элементов, всего 32 операции сравнения. Такое время называется логарифмическим.

На рисунке линейное и логарифмическое время представлено в специальной нотации, которая называется О-большое.

Используя О-большое, мы абстрагируемся от аппаратного обеспечения, будь то суперкомпьютеры или слабые домашние ПК. Мы не учитываем время, затрачиваемое на выполнение одной операции, а считаем количество операций, которые необходимо произвести для получения результата.

Записывая эффективность через О-большое, можно сказать, что «О» - это некоторая функция, которая учитывает время выполнения того или иного алгоритма на разных аппаратных платформах, а «n» - это количество операций.

Но почему мы говорим, что линейный поиск выполняется за O(n)? Ведь, наш алгоритм, который ищет уже зарегистрированных пользователей, может найти необходимое значение в первом элементе. Тогда время выполнения будет O(1). Потому что О-большое определяет время выполнения в худшем случае. Т.е. в случае, когда необходимое нам значение находится в конце списка.

Ниже приведены пять самых распространенных разновидностей О-большого, которые часто будут встречаться нам во время прохождения курса.

O(log n) – бинарный поиск.

O(n) – линейный поиск.

O(n* log n) – быстрая сортировка.

$O(n^2)$ – медленный алгоритм сортировки(пузырьковая).

O(!n) – очень медленный алгоритм, примером можно привести задачу о коммивояжере.

# Домашнее задание

1. Данный урок содержит только теоретическую информацию. И конкретного практического задания не предусмотрено.

# Дополнительные материалы

1. Тут и тут можно ознакомится с абстрактными типами данных.
2. В книге «Grokking Algorithms: An illustrated guide for programmers and other curious people» можно прочитать про алгоритмы и О-большое.

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Лафорте Р. Структуры данных и алгоритмы в Java. Классика Computers Science. 2-е изд. —СПб.: Питер, 2013. — 121-178 сс.