Р.В. Шамин

Лекции по информатике

Lector.ru

Грин Принт

Москва – 2019

УДК 004
ББК 21.2
Ш19

Рецензенты:
Заведующий кафедрой «Прикладная математика» Нижегородского государственного технического университета им. Р.Е. Алексеева, профессор, доктор физико-математических наук *А.А. Куркин*

Старший эксперт по разработке программного обеспечения в области искусственного интеллекта компании Mircosoft, доцент, кандидат физико-математических наук *Д.В. Сошников*

Шамин Роман Вячеславович
Ш19 Лекции по информатике. – М.: «Грин Принт», 2019. – 116 с.
ISBN 978-5-60434404-1-7

Курс лекций, читаемый автором в МИРЭА – Российском технологическом университете. Курс включает в себя различные разделы теоретической информатики и прикладные вопросы информатики.

Для студентов первых курсов университетов.

УДК 004
ББК 21.2

ISBN 978-5-60434404-1-7

Посвящается памяти замечательного математика
Евгения Михайловича Ландиса

Оглавление

Лекция № 1. Теория информации и структуры данных

1. Предмет информатики

Согласно «Большой российской энциклопедии», 2008: «Информатика – это наука о методах и процессах сбора, хранения, обработки, передачи, анализа и оценки информации с применением компьютерных технологий, обеспечивающих возможность ее использования для принятия решений».

Основным объектом информатики является понятие информации. Информация – это комплексная междисциплинарная категория, которая возникает в различных задачах и может нести различный смысл в зависимости от контекста. Однако согласно действующему ГОСТ 7.0-99:

«Информация – это сведения, независимо от формы их представления, воспринимаемые человеком или специальными устройствами как отражение фактов материального мира в процессе коммуникации».

Свойства информации:

- полезность
- актуальность
- достоверность
- объективность
- полнота
- понятность

Информация разделяется на аналоговую и цифровую. При этом информатика относится в

основном к цифровой информации, с которой работают компьютеры.

В XXI веке информация играет самую главную роль в нашей цивилизации и имеет самое большое значение для развития Человечества. Вспомним слова Натана Ротшильда, сказанные в 1815 году после битвы при Ватерлоо: «Кто владеет информацией, тот владеет миром.»

В настоящее время информатика проникла во все области нашей жизни, важнейшую роль информатика играет в следующих областях:

- телекоммуникация
- математическое моделирование
- компьютерная графика
- электронные банки данных
- шифрование
- машинное обучение и искусственный интеллект
- цифровая экономика

2. Теория информации

Мы будем изучать только цифровую информацию, поэтому важнейшим понятием является единица измерения информации – бит. Бит – это минимальная единица измерения информации. Бит может принимать только два значения 0 или 1. Это соответствует каком-либо физическому устройству, которое может быть только в двух положениях «0» или «1». Такие устройства легко конструируются.

Бит можно рассматривать как два варианта какого-либо высказывания, которое может быть истинным или ложным, хотя можно себе представить и вопрос, на который можно ответить «да», «нет» или «не знаю». Такая логика называется троичной логикой.

Бит – это двоичное число. Игра слов: ***bi****nary digi****t***, англ. *bit* – это кусочек, частица. С помощью битов формируются сообщения, при этом биты являются дискретными.

В современных компьютерах используют следующие единицы измерения:

1 Б (байт) = 8 бит
1 Кбайт (килобайт) = 1024 Б
1 Мбайт (мегабайт) = 1024 Кбайт
1 Гбайт (гигабайт) = 1024 Мбайт
1 Тбайт (терабайт) = 1024 Гбайт

Задача:
Сколько битов в 3 Мбайт?
Решение:
3 Мбайт = 3 * 1024 Кбайт = 3* 1024 * 1024 Б =
= 3* 1024 * 1024 * 8 бит = 25165824 бит

Цифровое сообщение – это упорядоченный конечный набор битов. Длиной сообщения мы будем называть количество бит, которое содержит данное сообщение и писать $|S| = N$, если сообщение состоит из N символов $S = (s_1, s_2, \ldots, s_N)$.

Сколько разных сообщений длины N? Ответ: 2^N различных сообщений.

Рассмотрим теперь систему, которая может находиться в одном из K состояний.

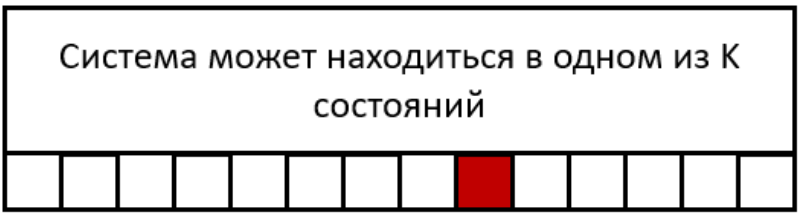

Рис. 1. Система из K состояний.

Рассмотрим задачу: какое минимальное количество бит нужно для описания всех K состояний системы?

Для этого нужно найти такое минимальное N, которое будет удовлетворять уравнению $K = 2^N$. Логарифмируя по основанию 2, получаем $N = \log_2 K$. Если в этой формуле N не будет четной, то его нужно округлить (вверх!) до целого.

Формула $N = \log_2 K$, где K – количество возможных состояний, а N – минимальное количество информации в битах, необходимое для описания состояний системы называется формулой Хартли.

Например, буквы русского языка (без различия по регистру) можно описать 5-ю битами, если не различать Е и Ё, а если различать, то 6-ю битами с избытком:

$\log_2 32 = 5$;

$\log_2 33 = 5{,}044$.

Пример:

Сколько нужно бит, чтобы описать состояния системы из 5 состояний? Нужно:

$\log_2 5 = 2{,}322 \Rightarrow N = 3$ бита.

Заметим, что тремя битами можно описать и системы и из 6, 7 и 8 состояний, но не из 9 состояний. Почему? Потому что

$\log_2 9 = 3{,}17 > 3$.

Пусть теперь система может находиться в одном из K состояний с разными вероятностями. В состоянии 1 с вероятностью p_1, в состоянии 2 с вероятностью p_2 и т.д. в состоянии K с вероятностью p_k, где $p_k \geq 0$. Тогда ценность знания, что система находится в состоянии p_k зависит от распределения вероятностей.

Фундаментальное понятие теории информации – энтропия информации. Под энтропией понимается мера неопределенности системы. Общей энтропией по Шеннону называется число H, определяемое по формуле

$$H = -\sum_{i=1}^{K} p_i \cdot \log_2 p_i$$

Частная энтропия по Шеннону h определяется по формуле

$$h = -p_i \cdot \log_2 p_i.$$

Согласно теории информации: прирост информации – это уменьшение энтропии.

Рассмотрим пример:

Если двоечник не поступил в РТУ МИРЭА, то тут мало информации, потому что «мы это и так знали», а вот если он поступил, то это «новость»! Полагая, что двоечник не поступает с вероятностью 0,9, а поступает с вероятностью 0,1 то, общая энтропия равна: H = 0.469.

А частная энтропия для не поступления равна: $-0,9 * \log_2 0,9 = 0.137$, а для поступления равна: $-0,1 * \log_2 0,1 = 0,332$.

3. Системы счисления и представление чисел в компьютере

Система счисления – это символический метод записи чисел.

Использование битов в качестве алфавита {0, 1} наводит на мысль, что компьютерам лучше использовать двоичную систему исчисления.

Натуральное число в десятичной системе исчисления представимо в виде:

$$x = a_N \ldots a_2 a_1 a_0,$$

где каждое a_n – это цифра 0..9. При этом число равно

$$x = a_0 10^0 + a_1 10^1 + \ldots + a_N 10^N.$$

В двоичной системе счисления число имеет вид:

$x = a_N \ldots a_2 a_1 a_0$,

где каждое a_n – это цифра 0 или 1. При этом число равно

$x = a_0 2^0 + a_1 2^1 + \ldots + a_N 2^N$.

В k-ичной системе счисления число имеет вид:

$x = a_N \ldots a_2 a_1 a_0$,

где каждое a_n – это цифра от 0 до k-1. А число равно

$x = a_0 k^0 + a_1 k^1 + \ldots + a_N k^N$.

Если k > 10, то в качестве цифр используются заглавные буквы латинского языка, например в шестнадцатеричной системе исчисления цифры: 0, 1, …, 9, A, B, C, D, E, F.

Если число x записано в k-ичной системе счисления, то пишут x_k. Наиболее распространены в информатике кроме десятичной системы исчисления еще двоичная и шестнадцатеричная.

Как перевести число из одной системы счисления в другую?

Чтобы любое число в k-ичной системе счисления перевести в десятичную систему нужно воспользоваться формулой:

$x = a_0k^0 + a_1k^1 + \ldots + a_Nk^N$.

Чтобы число X из десятичной системы перевести в k-ичную, нужно:

1. Разделить X на k: пусть X_1 – это целая часть отношения, а a_0 – остаток от деления.

2. Если X_1 не равно нулю, то делим X_1 на k, обозначаем через X_2 целую часть, через a_1 – остаток.

3. Повторяем эту процедуру до тех пор, пока целая часть от деления не станет равной нулю.

В результате

$X = a_Na_{(N-1)}\ldots a_1a_0$

будет представлением числа X в k-ичной системе счисления.

Примеры:

Перевести 17 в двоичную систему:

17 / 2 = 8 ост. 1;

8 / 2 = 4 ост. 0;

4 / 2 = 2 ост. 0;

2 / 2 = 1 ост. 0;

1 / 2 = 0 ост. 1,

следовательно, $17_{10} = 10001_2$.

Перевести 234 в шестнадцатеричную систему:

234 / 16 = 14 ост. A;

14 / 16 = 0 ост. E,

следовательно, $234_{10} = EA_{16}$.

Заметим, что при записи остатка мы пишем его в шестнадцатеричной записи: $10_{10}=A_{16}$, $14_{10}=E_{16}$.

В компьютере могут быть представлены числа только с конечным числом цифр после запятой. В этом случае любое число может быть представлено в виде a,b, где a – это целое число, a b – целое положительное число. Например: -648,512, где a = -648, b = 512.

Более того, такое число может быть представлено в виде

$$a,b * 10^p,$$

где a – целое, b – целое положительное, p – целое. Число a,b в этой записи называется мантиссой, а p – порядком.

Нормализованной называется запись, когда

$$1 \leq |a| < 10.$$

Например: -648,512 в нормализованной записи -6,48512*10^2.

Во многих языках программирования пишут

$$a,b*10^p = a.bEp,$$

используя точку. Например: -6,48512*10^2 = -6,48512E2.

Следует иметь в виду, что вещественные числа часто представляются приближенно, поэтому всегда возможны ошибки машинного округления чисел.

Число $\varepsilon > 0$ называется машинным эпсилон, если $a + \varepsilon = a$, где равенство понимается как сравнение

чисел на компьютере. Значение ε зависит от точности представления чисел.

4. Структуры данных

Основной задачей компьютера является обработка данных. Под обработкой данных понимается как вычислительные процедуры, так и процедуры, преобразующие одни данные в другие данные.

Данные – это форма представление информации, доступная для обработки. Формат представления данных – это последовательность бит (байт). Физически данные хранятся в виде файлов или потоков данных на физических носителях информации. Поток данных – это абстракция для доступа к данным из файлов, периферийных устройств и т.д.

Данные организованы в определенном формате, который определяется различными структурами данных.

Строительным элементом данных является переменная. Переменная – это именованная область памяти определенного типа. Тип переменной определяет и формат хранения данных.

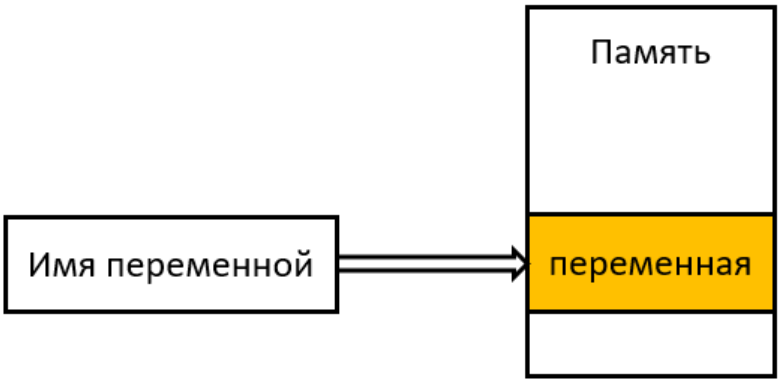

Рис. 2. Переменная и ее имя.

Переменная – это понятие языка программирования. В языках программирования каждая переменная иметь уникальное имя, которое, как правило, содержит буквы, цифры и подчеркивания, при этом имя переменной не может начинаться с цифры. Большинство языков программирования различают большие и малые буквы.

Вот примеры имен переменных:

Abc, ABC, _abc, a20, abc_20.

Неправильные имена: 20abc, _20.

Каждая переменная должна иметь определенный тип данных. В некоторых языка программирования (C++, Java, C#) каждая переменная перед использованием объявляется и получает фиксированный тип данных, который потом не может быть изменен. Такие языки называются строго типизированными. В других языках (Python, PHP,

JavaScript) тип данных не фиксирован и меняется в зависимости от контекста.

Простейшие типы данных:

- целочисленный (int)
- вещественный (double, real, float)
- строковый (string)
- логический (bool, boolean)

```
int param = 3.2;
string name = "Outlook";
double eps = 1.5E-12;
char sep = '\t';
bool Flag = false;
```

Рис. 3. Примеры объявления переменных.

При программировании серьезных проектов необходимо работать не только с переменными, но и со составными структурами данных. Простейшей структурой данных является массив.

Массив – структура данных, хранящая набор однотипных переменных, доступных по индексу.

Как правило, массивы индексируются, начиная с нуля. Т.е. первый элемент массива имеет индекс = 0. Каждый массив имеет определенный размер (количество элементов). Размер массива определяется либо в момент объявления или инициализации.

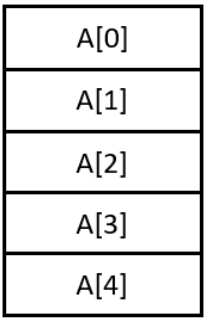

Рис. 4. Схема массива.

Массив – это тоже переменная с именем, например, A. Доступ к первому элементу A[0], а к последнему A[Count – 1], если Count – это количество элементов в массиве.

Массивы могут быть многомерными, когда каждый элемент индексируется не одним индексом, а несколькими. Например, двумерный массив A[i, j], а трехмерный A[i, j, k], где i, j, k – это индексы массива.

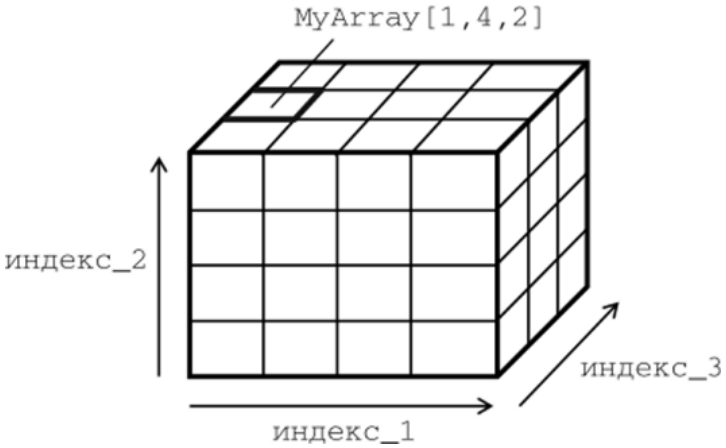

Рис. 5. Трехмерный массив.

Основная проблема массивов состоит в том, что массив имеет фиксированный размер. А как быть, если мы заранее не знаем количества элементов? Для этого необходимо использовать списки. Список – структура данных, состоящая из узлов, хранящих как данные, так и ссылку на следующий элемент.

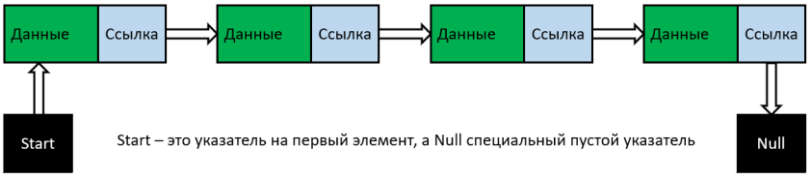

Рис. 6. Схема динамического списка.

При добавлении нового элемента ссылка последнего элемента заменяется на ссылку на новый элемент, а новый элемент получает ссылку на Null. Таким образом, количество элементов ограничивается только памятью.

Современные языки имеют встроенные списки, когда добавление нового элемента делается автоматически: List.append(X) – добавление X к списку List в Python.

Часто нам нужно организовать очередь, когда мы сохраняем и имеем доступ к элементам не по индексу, а по принципу «первый вошел, первый вышел» FIFO – «*first in, first out*».

Рис. 7. Схема очереди.

Важной особенностью очереди является то, что новые данные вставляются только в конец очереди, а извлекать можно только первый элемент очереди. Размер очереди может быть фиксированным или динамическим как в случае списков.

Еще одной важной структурой данных является стек.

Стек представляет собой структуру данных, в которой принцип: «последний вошел, первый вышел» LIFO – «*last in, first out*».

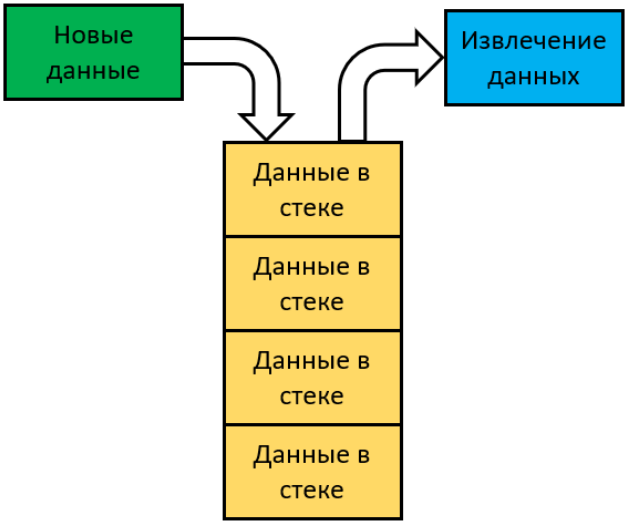

Рис. 8. Схема стека.

Стек – это очень эффективное средство организации данных, которая применяется в рекурсивных обходах дерева, организации вызовов подпрограмм и др. Обычно стек поддерживает три операции:

1. push(X) – поместить в стек элемент X
2. pop() – получить верхний элемент, удалив его из стека
3. peek() – получить верхний элемент, без удаления из стека

Стек можно сравнить со стопкой книг или магазином с патронами.

Последнюю структуру данных, которую мы рассмотрим, является хеш-таблица или словарь.

Классический массив представляет собой набор элементов, доступ к которым осуществляется с помощью индекса – целого числа 0, 1, …

А почему бы не сделать массив, доступ к элементам которого будет не с помощью чисел, а произвольного ключа-имени? Такой массив называется хеш-таблицей или словарем. Хеш-таблица позволяет эффективно организовать доступ по принципу «ключ – значение».

Например, для описания студентов можно использовать следующую структуру Students:

[key="фамилия имя отчество", data = "адрес"]

Например: адрес Лобанова Андрея Николаевича можно найти по запросу:

Students["Лобанов Андрей Николаевич"]

Лекция № 2. Алгебра логики и теория алгоритмов

1. Исчисление высказываний

Основой логики являются высказывания. Высказывание – это некоторое утверждение, которое может быть истинным или ложным. Высказывания мы будем обозначать заглавными латинскими буквами: A, B, C, …

Если высказывание A истинно, то мы будем писать A = 1, если A ложно, то пишем A = 0.

Примерами высказываний являются: A = «Волга впадает в Каспийское море», B = «Киты – это рыбы», C = «Плутон – девятая планета Солнечной системы», D = «На Марсе есть жизнь». При этом A = 1, B = 0, третье высказывание раньше было истинным, а сейчас нет. А последнее высказывание в настоящий момент неизвестно истинно или нет.

Математическая логика не занимается вопросами истинности или ложности конкретных высказываний! Для нас высказывания X, Y, Z – это просто переменные, которые могут принимать значения 0 или 1.

Операции с переменными, принимающими значения 0 и 1, называются булевой алгеброй.

Пусть A и B – высказывания. С этими высказываниями можно выполнять следующие основные логические операции:

- отрицание «НЕ»: ¬A
- конъюнкция «И»: A&B
- дизъюнкция «ИЛИ»: A∨B
- импликация «следует»: A → B

- эквивалентность: A ~ B.

Эти операции определяются

¬A = 0, если A = 1;
¬A = 1, если A = 0

A&B = 1, если A = 1 и B = 1
A&B = 0, если A = 0 или B = 0

A∨B = 1, если A = 1 или B = 1
A∨B = 0, если A = 0 и B = 0

(A → B) = 0, если A = 1 и B = 0
(A → B) = 1, в любом другом случае

A ~ B = 1, если A = B;
A ~ B = 0, если A≠B

С помощью приведенных операций и скобок можно конструировать произвольно сложные логические формулы, например:

((A&B) → (A∨C))∨(D~A)&(¬A∨C).

Каждая такая формула представляет собой новое высказывание, которое верно или ложно в зависимости от значений высказываний, входящих в формулу высказывания.

Формальное определение формулы:

1. Любое высказывание A и (A) являются формулами.

2. Если A и B – формулы, то формулами являются: (A)&(B), (A)∨(B), ¬(A), (A)→(B), (A)~(B).

Заметим, что логические операции могут быть выражены друг через друга:

- A & B = ¬(¬A∨¬B)
- A ∨ B = ¬(¬A&¬B)
- A → B = (¬A∨B)
- A ~ B = (A→B)&(B→A)

Вообще говоря, все логические операции можно выразить через две операции: ∨ и ¬ или через & и ¬.

Несложно установить различные равенства формул:

¬(¬A) = A
(A&B)&C = A&(B&C)
A&B = B&A
A&(B∨C) = (A&B)∨(A&C)
A∨B=B∨A
A∨(A&B) = A
(A∨B)∨C= A∨(B∨C)
A&(A∨B) = A

Поскольку значение каждой логической формулы зависит от значений высказываний, в нее входящих, то можно сказать, что эти формулы суть функции от высказываний. Для проверки истинности логической формулы при конкретных значениях входящих в нее переменных используются таблицы истинности. Таблица истинности – это перебор всех

возможных различных вариантов переменных и указания значения формулы.

A	B	C	$A \vee (B\&C)$
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

Рис. 1. Таблица истинности.

Пример таблицы истинности для формулы: A∨(B&C) приведен на рис.1. Если формула содержит N различных переменных, то таблица истинности содержит 2^N строк.

Любую логическую формулу можно привести к эквивалентной формуле, которая является дизъюнкцией элементарных конъюнкций. Такая форма формулы называется дизъюнктивной нормальной формой. Аналогично конъюнктивной нормальной формой называется вид формулы, когда она представляет собой конъюнкцию элементарных дизъюнкций.

Например, формула (A&B&¬C)∨A∨(A&¬B) находится в дизъюнктивной форме, а формула (A∨¬B)&(B∨C)&A находится в конъюнктивной форме. Использование нормальных форм позволяет в ряде случаев сразу сделать вывод о значении формулы.

Например, сразу видно, что последняя формула имеет ложное значение, если ложным является переменная A.

Поскольку любую формулу можно привести к виду, содержащему только операции &, ∨ и ¬, то будем считать, что формула U содержит только эти операции. Формула U∗ называется двойственной формулой, если она получается из формулы U путем замены всех операций & на операцию ∨, и операций ∨ на операцию &.

Например, для формулы

$U = (A\&B) \vee \neg C$

двойственной будет формула

$U^* = (A \vee B)\&\neg C.$

Очевидно, что $(U^*)^* = U$.

Если формула U содержит переменные $A_1, A_2, \ldots, A_n$, то мы будем писать это следующим образом $U = U(A_1, A_2, \ldots, A_n)$.

Заметим, что $\neg(A \vee B) = \neg A\&\neg B$, а также $\neg(A\&B) = \neg A \vee \neg B$, то из этого следует равенство:

$$\neg U(A_1, A_2, \ldots, A_n) = U^*(\neg A_1, \neg A_2, \ldots, \neg A_n).$$

Закон двойственности: если формулы U и V равносильны, то U* и V* также равносильны.

2. Исчисление предикатов.

Некоторые высказывания могут ссылаться на другие объекты, в зависимости от которых эти высказывания могут быть истинными или ложными. Например, «число n является простым числом». Это высказывание является истинным, если n = 937 и является ложным, если n = 1024.

Предикатом называется высказывание с параметрами, которые становятся обычными высказываниями при подстановке значений параметров. Рассмотрим произвольное множество M, которое называется множеством предметов или объектов. Предикатом на множестве M называется однозначная функция

$$P: M \to \{0, 1\},$$

которая каждому объекту $x \in M$ ставит в соответствие значение 0 или 1.

Пусть M – множество всех людей. Предикат: P(x) = «x – женщина». Предикаты могут зависеть от нескольких переменных. Например, пусть x – люди, n – натуральное число. Тогда предикат: P(x, n) = «Человек x имеет возраст n лет».

Из предикатов можно конструировать новые предикаты, используя логические операции. Например, из двух предикатов P(m, x) = «Машина m стоит x рублей» и Q(x) = «У меня есть x рублей» можно построить новый предикат R(m, x) = P(m, x) & Q(x), который имеет смысл, что «Я могу купить машину m».

Однако предикаты дают возможность создавать принципиально новые высказывания, в которых используются квантор существования и квантор всеобщности.

Квантор существования $\exists$ читается как «существует». Например, формула $\exists P(x)$ означает «существует хотя бы один $x' \in M$, для которого

$$P(x') = 1.$$

Квантор всеобщности $\forall$ читается как «для всех». При этом формула $\forall P(x)$ означает «для всех $x \in M$ $P(x) = 1$».

Говорят, что квантор связывает переменную, поскольку выражение $\forall P(x)$ уже является высказыванием, а не предикатом. Если предикат $Q(x, y)$ зависит от двух переменных, то предикат $R(y) = \exists Q(x, y)$ уже зависит от одной переменной.

Рассмотрим предикат, заданный на двух натуральных числах по формуле

$$P(n, m) = \text{“}n \le m\text{”}.$$

Образуем новый предикат

$$Q(n) = \forall m(P(n, m)).$$

Предикат Q имеет следующие значения: $Q(0) = 1$ и $Q(n) = 0$, если $n > 0$.

Для предиката $P(x) = \text{“}|\sin x| > 1\text{”}$. Высказывание $\exists P(x) = 0$, если x – вещественное число.

Можно комбинировать кванторы. Например, для предиката

$P(n, m) = “n \le m”$

построим высказывания:

$A = \exists n\ (\forall m(P(n, m)))$

и

$B = \forall n\ (\exists m(P(n, m)))$.

При этом A = 1, а B = 0.

Кванторы являются двойственными друг к другу:

$\neg(\forall x P(x)) = \exists x(\neg P(x))$ и $\neg(\exists x P(x)) = \forall x(\neg P(x))$

С помощью предиката P, заданного на множестве M, можно определить подмножество $A \subset M$ следующим образом: $A = \{x \in M: P(x) = 1\}$. Если на множестве M задан также другой предикат Q, который определяет множество $B = \{x \in M: Q(x) = 1\}$, то предикат $R(x) = P(x)\ \&\ Q(x)$ будет определять пересечение множеств $A \cap B$.

3. Конечные автоматы.

Перейдем к формальному определению вычислений и алгоритмов. Для этого используется понятие абстрактных вычислительных машин. Важнейшей абстракцией вычислительной машины является конечный автомат.

Рис. 2. Схема конечного автомата.

На вход конечному автомату подаются символы входного алфавита A, а на выходе – символы выходного алфавита B. При этом в каждый момент конечный автомат находится в одном из состояний их алфавита внутренних состояний. Конечный автомат работает в дискретное время t_1, t_2, …. Алфавиты A, B, Q – это конечные множества.

Выход КА зависит от входа и внутреннего состояния, при этом меняется внутреннее состояния в зависимости от входа и внутреннего состояния в предыдущий момент.

Поскольку конечный автомат работает дискретно, то входная последовательность обозначается:

a(1), a(2), …;

выходная последовательность:

$b(1), b(2), \ldots;$

последовательность внутренних состояний:

$q(0), q(1), q(2), \ldots .$

При этом $q(0)$ – это начальное состояние автомата.

Функционал конечного автомата описывается следующими рекуррентными уравнениями

$$b_k = f(a_k, q_{k-1}),$$
$$q_k = g(a_k, q_{k-1}).$$

Формально любой компьютер и программа представляет собой конечный автомат. Важно, что конечный автомат может иметь память, но только конечную!

Таким образом, формальное определение конечного автомата. Конечным автоматом называется пятерка:

$$K = \langle A, B, Q, q_0, f, g \rangle,$$

где $q_0 \in Q$ – начальное состояние.

Конечный автомат часто представляют в виде диаграммы, где кружки – это состояния, а стрелки – входящие символы.

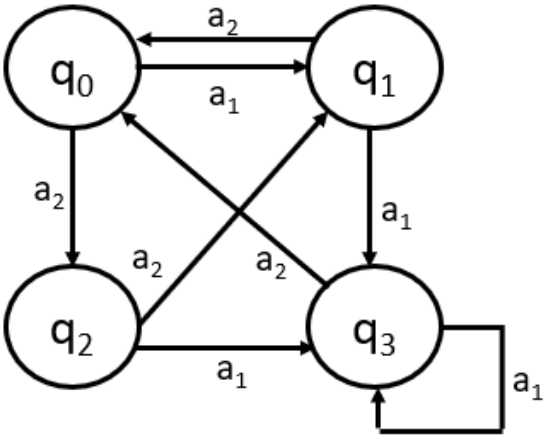

Рис. 3. Диаграмма конечного автомата.

Рассмотрим пример конечного автомата. Для этого нам необходимо задать множества A, B, Q, выделить начальное состояние и определить две функции:

$f: A \times Q \to B$

и

$g: A \times Q \to Q$.

Пусть множества

A = {♥, ♦, ♣, ♠},
B = {0, 1},
Q = {0, ♥, ♦, ♣, ♠},

Начальное состояние: $q_0 = 0$. Функция f(a(k), q(k – 1)) задается следующим образом:

если a(k) = q(k – 1), то f (a(k), q(k – 1)) = 1,
если a(k) ≠ q(k – 1), то f (a(k), q(k – 1)) = 0.

Функция g(a(k), q(k – 1)) задается так:

g(a(k), q(k – 1)) = a(k), если a(k) ≠ q(k – 1),
если a(k) = q(k – 1), то g(a(k), q(k – 1)) = 0.

Пусть на входе: ♥ ♥ ♥ ♦ ♦ ♣ ♣ ♣, т.е.:

a(1) = ♥, a(2) = ♥, a(3) = ♥, a(4) = ♦, a(5) = ♦, a(6) = ♣, a(7) = ♣, a(8) = ♣.

Внутренние состояния: 0, ♥, 0, ♥, ♦, 0, ♣, 0, ♣.
На выходе: 0, 1, 0, 0, 1, 0, 0

4. Понятие алгоритма и машина Тьюринга

Важнейшим понятием информатики (и не только) является понятие алгоритма. Бытовое определение алгоритма: алгоритм – это конечная совокупность точно заданных правил решения произвольного класса задач или набор инструкций, описывающих порядок действий исполнителя для решения некоторой задачи.

Приведенное выше определение не в полной мере точно, поэтому не может быть использовано в математических доказательствах, в частности в

вопросах о разрешимости. Поэтому в математике рассматривают различные так называемые уточнения понятия алгоритма, которые уже являются полностью формальными.

Заметим, что исполнителем алгоритма может быть машина, человек, ребенок и любой субъект, который может полностью понимать инструкции алгоритма и в точности их выполнять, без привлечения каких-либо иных сведений (интуиции, опыта, эмоций и т.д.).

Важно, что алгоритмы: дискретные и детерминированные. Т.е. они работают только с дискретными объектами и без использования каких-либо случайностей!

Машина Тьюринга – это абстрактное гипотетическое устройство, описывающее исполнителя алгоритма. С помощью машины Тьюринга можно дать точное определение вычислимы и невычислимых процедур.

МТ состоит из бесконечной в обе стороны ленты, которая разделена на ячейки с номерами …, – 2, – 1, 0, 1, 2, … и из головки машины, которая находится над ячейкой и может за один такт оставаться на месте или перемещаться на одну ячейку вправо или влево. В каждой ячейке может быть один из символов из конечного алфавита A, либо ячейка может быть пустой, но только конечное число ячеек непустые. В каждый момент времени ячейка может находиться в одном из состояний из конечного множества Q.

…	-3	-2	-1	0	1	2	3	…
		×		×	×	×		

Рис. 4. Схема машины Тьюринга.

Функционирование МТ определяется программой – набором инструкций:

$$aq \rightarrow a'q'D,$$

где $a,a' \in A \cup [\_]$, $q,q' \in Q$, $D \in \{L, S, R\}$. Инструкция применяется, если головка находится в состоянии q и над ячейкой a, тогда головка переходит в состояние q′, записывает в ячейку a′ или пустой символ и перемещается L – влево, R – вправо, S – на месте.

У машины Тьюринга есть начальное состояние $q_0 \in Q$ и финальное состояние $q_F \in Q$. Если МТ попадает в финальное состояние, то работа машины завершается, и мы будем говорить, что МТ применима к конфигурации начальных символов на ленте, если же головка оказывается в состоянии q над символом a, но в ее программе нет инструкции вида $aq \rightarrow a'q'D$ или МТ никогда не останавливается, то говорят, что МТ не применима к начальной конфигурации.

Приведем пример машины Тьюринга: A = {|}, Q = {W, F}. Начальное состояние S, финальное – F. Программа машины состоит из следующих инструкций:

$$|W \rightarrow |WR,$$
$$\_W \rightarrow |FS.$$

Начальная конфигурация состоит из N символов “|”, начиная с нулевой ячейки. Головка находится над нулевой ячейкой.

Эта машина вычисляет функцию:

$$f(N) = N + 1.$$

Пока головка машины находится над символом |, она сдвигается вправо, как только она оказывается над пустой клеткой, она записывает в эту клетку символ | и останавливается. В результате, на ленте оказывается N + 1 символов.

Любой алгоритм представляет собой функцию

$$f: N \to N,$$

где N – натуральные числа, поскольку алгоритм оперирует только дискретными, конечными объектами, которые можно занумеровать. Если функцию f(n) можно реализовать с помощью какой-либо машины Тьюринга, то мы будем говорить, что функция f(n) является вычислимой.

Тезис Тьюринга-Черча: «любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга».

На основании этого тезиса мы будем говорит, что алгоритм – это процедура, которая может быть заданной с помощью машины Тьюринга.

Заметим, что алгоритм, как и вычислимая функция не обязаны быть применимы или заданы на

всех входных значениях – алгоритм может работать бесконечно долго (зациклится) на некоторых или даже всех входных значениях.

Алгоритм может быть реализован любой программой с потенциально бесконечной памятью. Программирование на МТ эквивалентно программированию на любом другом языке программирования. Просто МТ – это модель простейшего вычислителя.

При конструировании алгоритмов необходимо особое внимание уделять оценки сложности алгоритмов. Сложность алгоритмов оценивают по количеству операций (шагов), необходимых для выполнения алгоритма в зависимости от длины (объема) исходных данных.

Рассмотрим алгоритм, который суммирует n чисел. Для этого нужно произвести порядка n операций, поэтому говорят, что сложность этого алгоритма O(n). Для суммирования элементов квадратной матрицы n×n нужно порядка n^2 операций, поэтому сложность этого алгоритма – $O(n^2)$.

Если сложность алгоритма равна $O(n^k)$, то говорят, что этот алгоритм имеет полиномиальную сложность или относится к классу P.

Существуют алгоритмы, которые имеют не полиномиальную, а экспоненциальную сложность: $O(2^n)$, O(n!), $O(n^n)$. Это различные алгоритмы полного перебора и т.д. Говорят, что алгоритм, имеющий экспоненциальную сложность, относится к классу NP.

Нерешенная проблема: P = NP? Не известно, можно или нельзя любой алгоритм NP решить за

полиномиальное время. За решение этой проблемы – миллион долларов!

5. Архитектура вычислительных систем.

Исторически первой архитектурой вычислительных систем является принцип фон Неймана, который заключается в следующем:

- однородность памяти – данные и программный код хранятся в одной памяти;
- адресность – память состоит из пронумерованных ячеек, к каждой имеется прямой доступ;
- программное управление – процессор исполняет команды, из которых состоит программа.

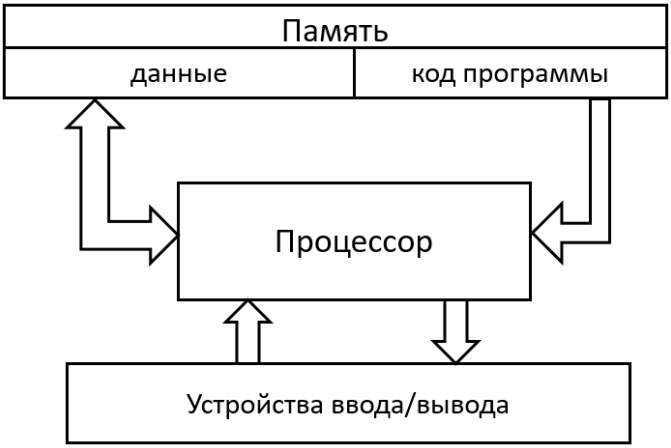

Рис. 5. Архитектура машины фон Неймана.

Архитектура Джон фон Неймана сыграла выдающуюся роль в создании компьютеров, но она имела ряд существенных недостатков: наличие узких мест – невозможность одновременного доступа к данным и коду, а также отсутствие защиты данных. Поэтому в современных компьютерах используются другие архитектуры компьютеров.

- CISC (англ. complex Instruction Set Computer — «компьютер с полным набором команд») — архитектура процессора с большим количеством команд.
- RISC (англ. reduced instruction set computing) - архитектура с сокращенным набором команд.
- MISC (англ. minimal instruction set computing) - архитектура с минимальным набором команд.
- VLIW (англ. very long instruction word - «очень длинная машинная команда») - архитектура с длинной машинной командой, в которой указывается параллельность выполнения вычислений.

Многие современные процессоры, например, Intel и AMD являются гибридными (CISC+RISC). Кроме того, современные процессоры состоят из нескольких ядер и поддерживают параллельные вычисления. Важным является энергопотребление процессоров.

Лекция № 3. Графы и деревья

1. Понятия теории графов

Графы описывают структуры «кружочков со стрелочками», которые возникают в очень многих областях: в естественных и общественных науках и инженерных приложениях. Дадим формальное определение ориентированного графа. Пусть V – непустое множество, элементы которого называются вершинами графа. Обычно мы будем рассматривать конечное множество вершин, но в ряде случаев можно рассматривать и бесконечное множество вершин. Далее рассмотрим множество упорядоченных пар элементов множества V, которое будем обозначать E и называть множеством дуг графа. Таким образом, пара [V, E] называется ориентированным графом.

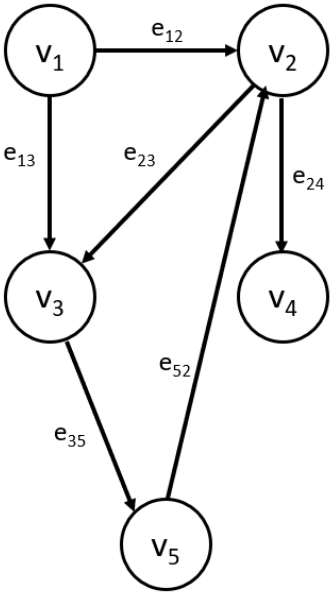

Рис. 1. Пример графа.

Мы обозначили через e_{ij} – дугу из вершины i в вершину j. Мы будем говорить, что дуга e_{ij} исходит из вершины i и входит в вершину j.

Если для каждой дуги $(v_i, v_j) \in E$ существует дуга $(v_j, v_i) \in E$, т.е. для каждой стрелочки существует обратная стрелочка, то граф называется неориентированным и пара взаимно обратных дуг обозначается линией.

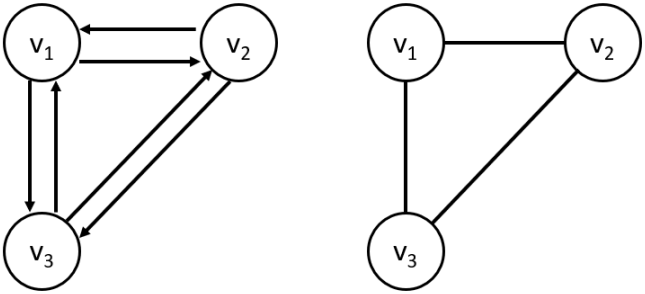

Рис. 2. Неориентированный граф.

В неориентированном графе дуги называются ребрами. Две вершины называются смежными, если между ними есть ребро. Вершина v и ребро $\{v_i, v_j\}$ называются смежными, если $v = v_i$. Граф называется полным, если любые две вершины этого графа смежные.

В ориентированном графе последовательность вершин такая, что из каждой вершины исходит дуга в следующую вершину, называется путем. Причем, первая вершина в последовательности называется началом пути, а последняя – концом пути. В неориентированном графе путь называется маршрутом.

Путь (маршрут) называется замкнутым, если конечная вершина совпадает с начальной. В противном случае путь называется незамкнутым. Незамкнутый путь, в котором все вершины различны называется цепью. Замкнутый путь (маршрут) называется контуром (циклом). Вершина v называется достижимой из вершины u, если существует путь из u в v.

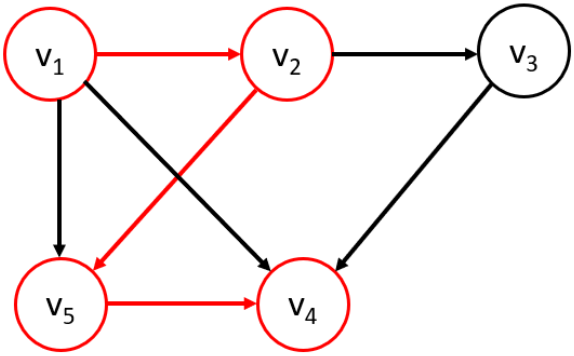

Рис. 2. Путь в графе.

Рассмотрим вопрос задания графа. Для машинной обработки графов недостаточно «нарисовать» граф. Нужно задать его формально. Для этого используют матрицу смежности. Пусть граф содержит конечное число вершин V = {1, 2, …, N}. Матрица смежности представляет собой квадратную матрицу N x N, состоящая из элементов a_{ij}.

$$a_{ij} = \begin{cases} 1 & \text{если } (v_i, v_j) \in E \\ 0 & \text{если } (v_i, v_j) \notin E \end{cases}$$

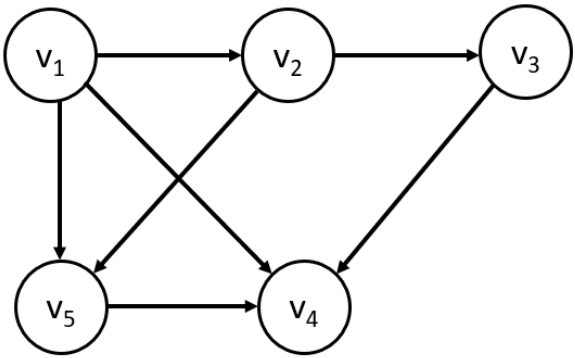

Рис. 3. Пример графа.

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	1
v_2	0	0	1	0	1
v_3	0	0	0	1	0
v_4	0	0	0	0	0
v_5	0	0	0	1	0

Рис. 4. Матрица смежности.

Пусть G – некоторый граф. Тогда граф H называется подграфом графа G, если выполнены условия $V_H \subset V_G$, $E_H \subset E_G$. Пишут $H \subset G$. Если H – подграф графа G и $V_H \subset V_G$, то подграф H называется основным графом или фактором.

Пусть G – граф, содержащий более одной вершины. Тогда для любой вершины $v \in V_G$ можно ввести операцию удаления этой вершины из графа.

Будем говорить, что H = G – v получается в результате удаления вершины v из графа G, если $V_H = V_G \setminus v$ и из множества дуг исходного графа удаляются все дуги, которые начинаются или заканчиваются в вершине v.

Операция добавления вершин и/или дуг к графу заключается в добавлении соответствующих вершин и/или дуг.

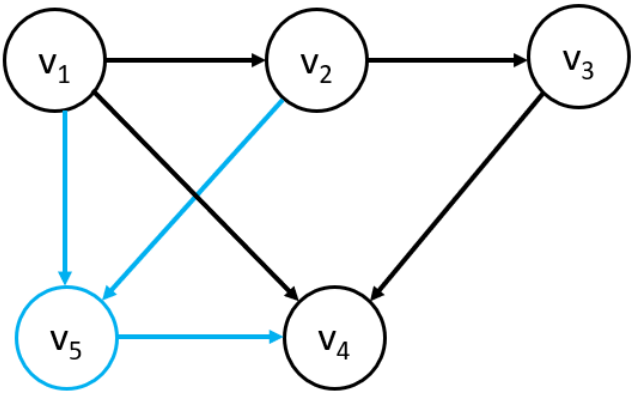

Рис. 5. Удаление вершины в графе.

В графе на рисунке удаляется v_5 – синим показаны вершины и дуги, которые нужно удалить.

Если каждой вершине и/или дуге графа приписать определенное число, то такой граф называется нагруженным графом. Число, приписанное вершине или дуге, будем обозначать L(v) и M(e).

Примером нагруженного графа является граф, в котором вершины обозначают города, а дуги –

дороги между городами. При этом каждой дуге можно приписать расстояние данной дороги.

Рассмотрим неориентированный нагруженный граф такой, что M(e) > 0 для любого ребра. Будем считать, что граф не имеет петель. Тогда между двумя вершинами графа u и v можно определить расстояние d(u, v):

$$d(u,v) = \min_{\pi(u,v)} \sum_{n=1}^{k} M(e^n)$$

Здесь минимум берется по всем путям, соединяющим u и v.

Функция d обладает всеми свойствами расстояния:

1) d(u, v) ≥ 0, причем d(u, v) = 0 только, если u = v,
2) d(u, v) = d(v, u),
3) d(u, v) ≤ d(u, w) + d(w, v).

Рассмотрим нагруженный ориентированный граф, дугам которого приписаны неотрицательные значения. Через w(u, v) мы обозначим вес дуги (u, v) графа. Выберем две различные вершины графа s и t, и рассмотрим кратчайший путь из s в t.

Для нахождения кратчайшего пути в графе воспользуемся алгоритмом Дейкстры. На каждой итерации алгоритма мы вершинам будем приписывать временные или постоянные метки m(u). Если метка постоянная, то ее значение – это

кратчайший путь из s в u. Через p(u) обозначим номер вершины, предшествующей u в кратчайшем пути из s.

1. Вершине s назначаем постоянную метку m(s) = 0, метки всех остальных вершин временные и равные ∞. Положим r = s.

2. Для всех не имеющих постоянные метки вершин, в которые есть дуги из вершины r:

если m(v) > m(p) + w(r, v), то положим m(r) = m(r) + w(r, v) и p(v) = r.

3. Пусть V′ – множество вершин с временными метками. Найдем v*, имеющую минимальную метку из V′. Тогда метку m(v*) считаем постоянной.

4. Положим r = v*. Если r ≠ t, то перейти к шагу 2.

5. Конец. Значение m(t) – минимальный путь. А сам путь s,…, $p^3(t)$, $p^2(t)$, p(t), t.

Здесь $p^k(t) = p(p^{k-1}(t))$.

2. Деревья.

Ориентированным деревом называется ориентированный граф, который является связным и в нем существует единственная вершина $v^0 \in V$, в которую не входят никакие дуги, и в каждую вершину $v \in V \setminus v^0$ входит ровно одна вершина.

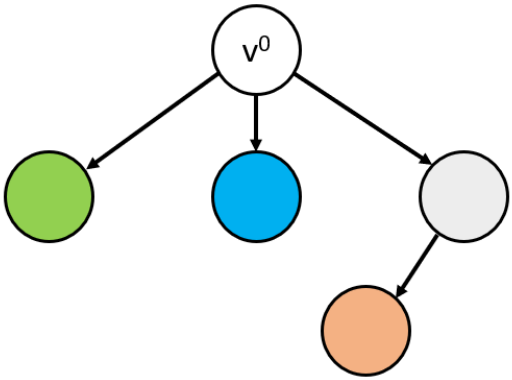

Рис. 6. Пример дерева.

Неориентированным деревом называется неориентированный граф, который не имеет циклов. Дерево называется бинарным, если из каждой вершины или не исходит ни одна дуга или исходит ровно две дуги.

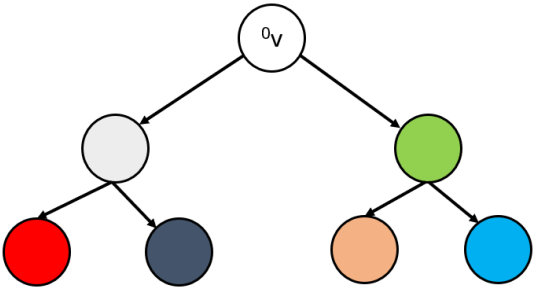

Рис. 7. Пример дерева.

Рассмотрим неориентированное дерево G = [V, E].

Тогда для этого дерева имеют место следующие утверждения:

1. В дереве для любых двух вершин существует единственный маршрут, соединяющих эти вершины.

2. Для количества ребер |E| и количества вершин |V| выполняется следующее соотношение: |E| = |V| + 1.

3. Если у дерева удалить любое ребро, то граф перестанет быть связным.

4. При добавлении нового ребра в дерево, это дерево станет графом, имеющим ровно один цикл.

Характерными примерами деревьев являются:

1. Дерево папок в файловой системе.

2. Дерево решений.

3. Классификация животных.

4. Иерархия подчинения в организации.

Важная характеристика дерева – это глубина дерева, которая определяется как максимальная длина пути в графе, где длина определяется количеством ребер в пути.

Функция называется рекурсивной, если она в своем теле вызывает саму себя (обычно с другими аргументами).

Пример функция вычисления факториала:

```
int Fact(int n) {
   if (n == 1)
   {
      return 1;
   } else {
      return n * Fact(n – 1);
   }
}
```

Рассмотрим рекурсивную процедуру обхода ориентированного дерева:

1. Вход алгоритма – вершина дерева v

2. Если из вершины v нет исходящих дуг, то выход.

3. Для каждой вершины, в которую входят дуги, исходящие из вершины v, выполнить алгоритм обхода дерева.

4. Выход.

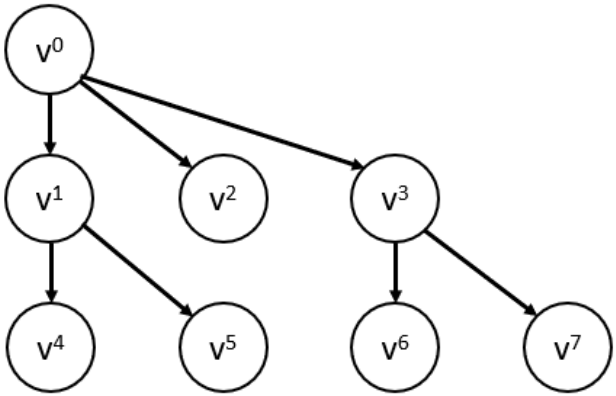

Рис. 8. Рекурсивный обход дерева.

Результат обхода:
v^0, v^1, v^4, v^5, v^2, v^3, v^6, v^7.

Лекция № 4 Синтез схем из функциональных элементов

1. Дискретные преобразователи.

Многие технические устройства могут быть представлены в виде дискретных преобразователей без памяти. Такие устройства преобразуют набор n входных переменных в набор выходных переменных. Пусть заданы алфавиты входных переменных X и выходных переменных Z.

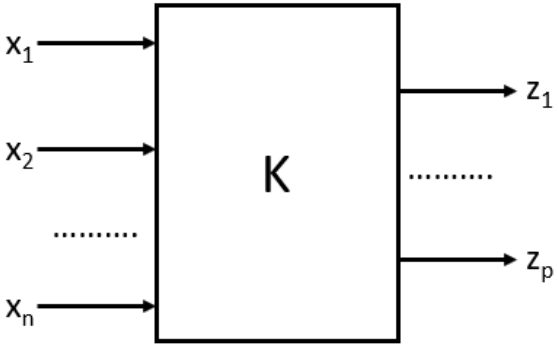

Рис. 1. Дискретные преобразователи.

Мы будем рассматривать в качестве дискретных преобразователей логические устройства.

В этом случае алфавиты входных и выходных переменных имеют вид: X = Z = {0, 1}. При этом дискретные преобразователи могут быть представлены в виде p функций:

$z_1 = f_1(x_1, x_2, \ldots, x_n)$,

$z_2 = f_2(x_1, x_2, \ldots, x_n)$,
……..
$z_p = f_p(x_1, x_2, \ldots, x_n)$.

Будем предполагать, что у нас есть конечное множество F, состоящее из объектов F_i, i = 1,2,…, r. Эти объекты мы будем называть элементам.

Каждый элемент F_i имеет n_i входов и один выход. Соответственно логической сетью мы будем называть объект, имеющий n входов и p выходов.

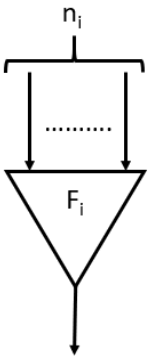

Рис. 2. Функциональный элемент.

Каждый элемент F_i имеет n_i входов и один выход. Соответственно логической сетью мы будем называть объект, имеющий n входов и p выходов.

Чтобы определить логическую сеть, построенную из функциональных элементов, воспользуется индуктивным методом. Базисом индукции будет одна изолированная вершина, которая по определению является логической сетью, называемой тривиальной логической сетью. Далее

будем использовать несколько операций, применение которых к любой логической сети, снова даст логическую сеть.

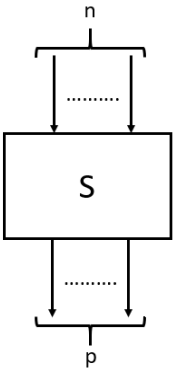

Рис. 3. Логическая сеть.

Операция объединения непересекающихся логических сетей. Пусть мы имеем две непересекающиеся логические сети S′ и S′′. Непересекающиеся это означает, что у них нет общих элементов, входов и выходов. При этом S′ имеет n входов и p выходов, а сеть S′′ имеет m входов и q выходов. Логическая сеть S есть результат объединения сети S′ и S′′, если S представляет собой теоретико-множественное объединение и имеет n + m входов и p + q выходов.

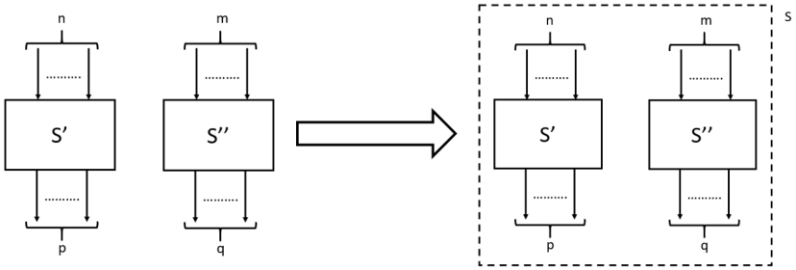

Рис. 4. Операция объединения сетей.

Операция присоединения функционального элемента F_i, имеющего n_i входов, к логической сети S'. Пусть у нас есть какая-либо логическая сеть S', которая имеет p выходов и функциональный элемент F_i. Если $n_i \leq p$, то в S' можно выбрать n_i различных выходов. Логическая сеть S представляет собой сеть S' с добавленным функциональным элементом F_i у которого входами являются выходы в сети S'.

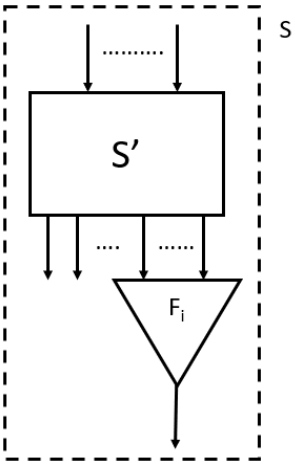

Рис. 5. Операция присоединения элемента.

Операция расщепления выхода. Пусть у нас есть логическая сеть S′ (возможно, тривиальная). Выделим в этой сети какой-либо выход, например, с номером j. Тогда логическая сеть S называется сетью, полученной путем расщепления выхода j. Входами этой сети являются входы сети S′, а выходы с номерами 1, …, j – 1, j + 1, …, p совпадают с выходами S′ и еще два выхода, совпадающих с выходом j сети S′.

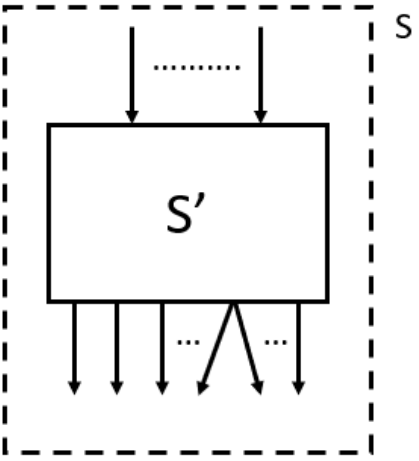

Рис. 6. Операция расщепления выхода.

2. Синтез логических схем.

В качестве функциональных элементов будем использовать три логических элемента, которые реализуют основные логические операции: ¬, &, ∨.

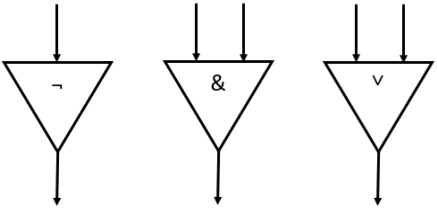

Рис. 7. Функциональные элементы.

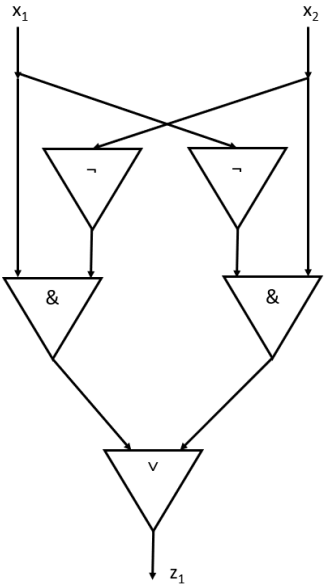

Рис. 8. Пример логической схемы.

Если $x_1 = 0$, $x_2 = 0$, то $z_1 = 0$; Если $x_1 = 1$, $x_2 = 0$, то $z_1 = 1$;

Если $x_1 = 0$, $x_2 = 1$, то $z_1 = 1$; Если $x_1 = 1$, $x_2 = 1$, то $z_1 = 0$.

Таким образом данная схема является сумматором по модулю 2: $S(x_1, x_2) = x_1 + x_2 \bmod 2$.

Проблема синтеза логических схем из функциональных элементов состоит в следующем. Пусть нам задано множество функциональных элементов F и дана произвольная система логических уравнений:

$$z_1 = f_1(x_1, x_2, \ldots, x_n),$$
$$z_2 = f_2(x_1, x_2, \ldots, x_n),$$
$$\ldots\ldots\ldots$$
$$z_p = f_p(x_1, x_2, \ldots, x_n).$$

Требуется построить логическую схему $S(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_p)$, которая реализует эту систему уравнений. Если запас функциональных элементов обладает полнотой, то это всегда можно сделать. Однако возникает важный практический вопрос о построении минимальных логических схем.

Через $L(S)$ обозначим количество функциональных элементов и назовем это число сложностью схемы S. Поскольку одни и те же логические уравнения можно реализовывать с помощью различных схем, то возникает задача построение такой схемы S^*, что $L(S^*) = \min L(S)$, где минимум берется по всем схемам, реализующим уравнения.

Лекция № 5. Парадигмы и методы программирования

1. Что такое программирование?

Основой компьютерных и информационных технологий является программирование. Почти с самого начала программное обеспечение стоит дороже компьютеров.

Программирование – это процесс создания программ, который включает в себя следующие этапы:

1. Проектирование ПО
2. Разработка алгоритмов
3. Написание кода
4. Отладка кода
5. Тестирование

Эти процессы часто выполняются непоследовательно, а циклически, например: 1-2-3-4-3-2-3-4-5-3-4-5.

Очень важно понимать, что программирование – это не только и не сколько написание кода программы, а проектирование архитектуры программы. Код пишут – кодеры, а программисты проектируют программы.

Серьезные программные комплексы пишутся большими коллективами в течение длительного времени. Поэтому очень важно КАК писать программы. Они должны быть понятными и

модернизируемыми. Принципиальное значение имеет именно технология написания программ.

Технологии написания программ разделяются на различные парадигмы программирования. В настоящее время эти парадигмы, как правило, используются гибридно. Многие современные языки программирования поддерживают несколько парадигм программирования.

Парадигмы программирования:

- структурное программирование
- объектно-ориентированное
- функциональное программирование
- логическое программирование

Зачем нужны парадигмы программирования?

- При написании больших программ нужно обеспечить возможность одновременной разработки большим коллективом.
- Возможность дописывания и переписывания программ.
- Возможность масштабирования программ.
- Надежность программ

Профессиональное программирование – это обязательно работа по профессиональным правилам, когда любой код пишется не «как короче», а «как правильно».

2. Структурное программирование.

Принципы структурного программирования:

- Программа представляется в виде иерархической структуры блоков.
- Тело программы содержит управляющие структуры: последовательность, ветвление, цикл и подпрограммы.
- Разработка программы ведется методом «сверху вниз».

Целью структурного программирования является повышение производительности труда программистов при создании больших программ и сократить количество ошибок, а также упростить отладку и сопровождения программ. Структурное программирование возникло в 1970-х года, когда программирование стало новой отраслью, при этом сложность и стоимость разработки программ резко выросло.

Структурное программирование возникло как борьба с оператором безусловного перехода – GOTO, использование которого приводило к «спагетти-коду».

Большинство языков программирования поддерживают структурное программирование.

Программирование начинается с проектирования программы, которое состоит в том, чтобы представить в иерархическом виде архитектуру программы.

Головная программа разбивается на различные блоки, которые в свою очередь также разбиваются на другие блоки. Этот процесс продолжается до элементарных блоков.

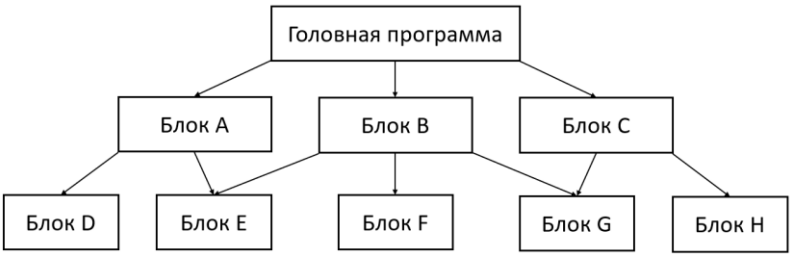

Рис. 1. Разработка «сверху вниз».

Каждый блок должен решать элементарную задачу, при этом обычно код каждого блока должен быть небольшим (условно – одна страница), чтобы его код легко читался. Если блок оказывается большим – его нужно снова разделить на подблоки.

Каждый блок должен иметь четкое описание входных параметров, с минимальным использованием глобальных переменных, а также четкий результат его работы.

При программировании блоков используют технику «заглушек», когда блоки только обозначаются, но не реализуются.

Подпрограмма (функция) – это отдельный код, который выполняется при вызове, а после окончания выполнения подпрограммы, управление возвращается в точку вызова. Подпрограмма может получать входные параметры и возвращать результат.

Подпрограмма, возвращающая результат, называется функцией.

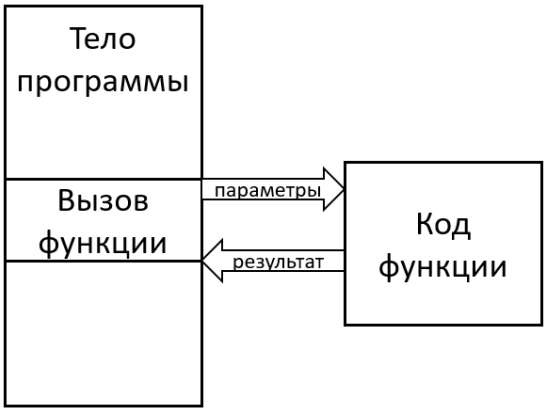

Рис.2. Вызов подпрограмм.

```
const int N = 10;
int S = 0;
for (int i = 0; i < N; i++)
{
        S += f(i);
}
std::cout << S;

int f(int a)
{
        return a * a;
}
```

Преимущества функций – код используется многократно, что существенно сокращает размер программы. При изменении алгоритма функции

нужно внести исправление в одном месте. Функции позволяют реализовывать рекурсии.

2. Объектно-ориентированное программирование

Объективно-ориентированное программирование – это парадигма программирования, где элементарным блоком является не функция, а объект. При этом объект объединяет в себе не только данные, но и код. Это создает более высокий уровень абстракции, что позволяет более эффективно проектировать и реализовывать сложные программы.

Объект – это переменная типа класса, а класс – это пользовательский тип переменной.

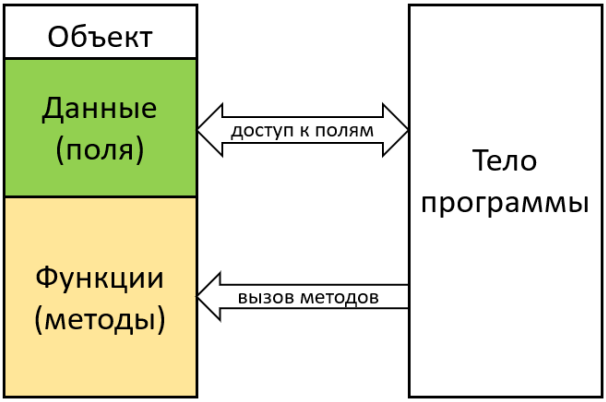

Рис. 3. Схема работы с объектом.

В ООП каждый реальный объект, с которым работает программа оформляется в виде класса,

который содержит данные, описывающие этот объект, а также функции, которые работают с объектом.

В результате мы можем мыслить программируемую ситуацию, как набор объектов со своими данными и функционалом.

ООП поддерживают многие современные языки. Основные: C++, Java, C#, Python, PHP.

Объективно-ориентированное программирование описывается тремя принципами:

1. инкапсуляция
2. наследование
3. полиморфизм

Инкапсуляция – это основной принцип ООП, который состоит в том, что данные и методы их обработки объединены в одну переменную – объект. При этом методы каждого объекта видят данные своего объекта. Инкапсуляция подразумевает, что из вне объекта доступны не все поля и методы.

Пример. Пусть мы моделируем поведение машины и светофора. В нашей модели будет два класса: класс описывающий светофор и класс, описывающий машину.

```
// класс для описания светофора
class Tlihgts {
  int Light; // горящий цвет
public:
  int GetLight(); // вернуть текущий цвет
  void Set(int light); // установить цвет
  void Next(); // следующий цвет
```

```
};

// класс для описания машин
class TCar {
  int State; // состояние машины
public:
  int GetState(); // вернуть состояние
  void Stop(); // остановиться
  void Start(); // начать движение
};

TLights Lights; // создать объект светофор
TCar Car1, Car2; // создать объекты машина1 и машина2
Lights.Set(1); // установить красный цвет
if(Light.GetLight() != 3) // если цвет не зеленый
{
  Car1.Stop(); // остановить машины
  Car2.Stop();
}
else
{
  Car1.Start(); // ехать
  Car2.Start();
}
```

Инкапсуляция – мощное средство ООП, которое помогает проектировать сложные системы естественным образом.

Наследование в ООП – механизм, позволяющий не только сократить повторное использование кода,

но и сделать естественным проектирование родственных классов.

Если у нас есть класс TClass1, то мы можем объявить (создать) новый класс TClass2, который автоматически получит все поля и методы класса TClass1. При этом новый класс может быть дополнен новыми полями и методами, а также он может перекрыть любой метод класса-родителя. Класс TClass2 в свою очередь тоже может быть наследован и т.д.

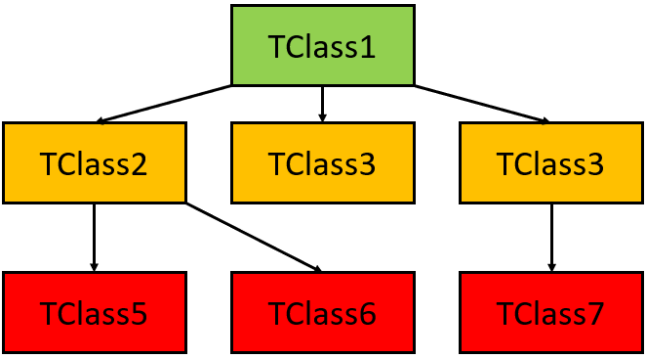

Рис. 4. Иерархия классов.

В результате наследования возникает иерархия классов, которая может быть представлена в виде графа, в котором стрелки – наследование.

В некоторых языка программирования, например, C++ допускается множественное наследование, но этот механизм считается тяжеловесным и небезопасным.

Наследование классов позволяет экономно проектировать классы объектов для представления сложных объектов.

Рассмотрим классы для представления геометрических фигур: точки, прямоугольник, окружности и эллипса.

```
// класс точки
class TPoint {
public:
   double X1, Y1; // координаты точки
   void Draw(); // нарисовать
}
// класс прямоугольника
class TRect : public TPoint {
public: double X2, Y2; // добавить новую точку
}

// класс окружности
class TCircle : public TPoint {
public:
   double R; // радиус окружности
}
// класс эллипса
class TEllipse : public TCircle {
public:
   double R2; // второй радиус
}
```

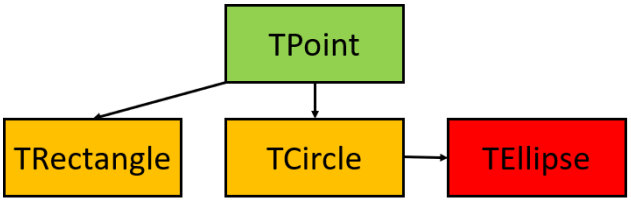

Рис. 5. Пример иерархии классов.

При наследовании классов, все поля и методы сохраняются. При этом добавляются новые поля и методы. При этом наследование позволяет повторно использовать код.

Полиморфизм в объектно-ориентированном программировании – это удивительные возможности по модернизации кода программы, написанного ранее без перекомпиляции текста старой программы.

Полиморфизм – это свойство наследования классов, при котором некоторые методы наследуемого класса могут быть переопределены в наследнике.

Методы, которые можно изменять с помощью полиморфизма, называются виртуальными.

При наследовании мы не меняем (и можем не знать) код Func1, однако после наследования и изменения Func2 функция Func1 будет уже ссылаться на новую функцию Func2.

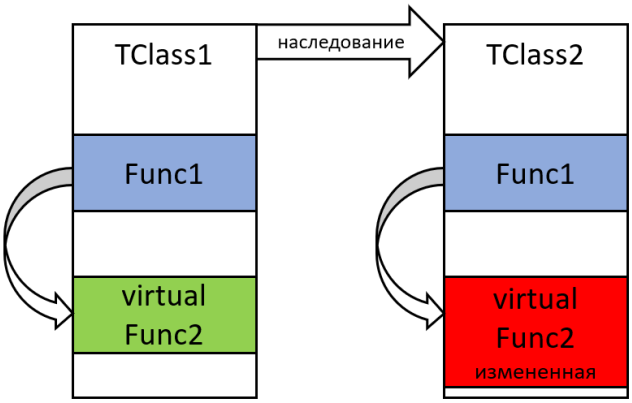

Рис. 6. Пример полиморфизма.

Полиморфизм – это очень интересный и мощный инструмент объектно-ориентированного программирования.

3. Виды программирования.

В настоящее время программирование – это огромная отрасль, которая включает в себя различные виды программирования:

- профессиональное программирование
- научное программирование
- учебное программирование

Также программирование можно разделить по платформам программирования:

- десктопное программирование
- мобильное программирование
- веб-программирование
- программирование встраиваемых систем
- программирование баз данных

Профессиональное программирование или промышленное программирование может быть нацелено либо на создание коммерческих приложений, либо на создание приложений, используемых на предприятиях или частными лицами.

Научное программирование – это создание программ для решения научных задач.

Современные технологии программирования ориентированы на эффективное создание приложений большим коллективом программистов.

Работа программистом – это высокооплачиваемая профессия с большим числом вакансий, но, чтобы стать программистом нужно учиться этому, работая над проектами.

Десктопные приложения – это программы, запускаемые на компьютерах, ноутбуках. В настоящее время рынок десктопных приложений резко сокращается – пользователи все меньше устанавливают приложения, предпочитая работать в веб-интерфейсе или с мобильными приложениями.

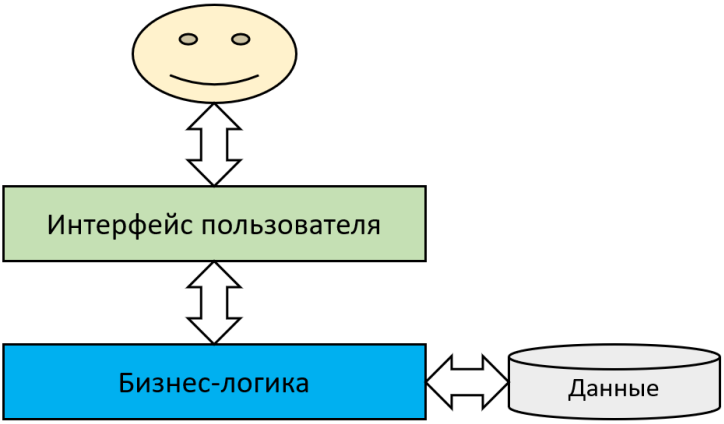

Рис. 7. Схема десктопных приложений.

Разработка десктопных приложений требует создания интерфейса пользователя и бизнес-логики приложения. При этом технологии программирования подразумевают, что разработка интерфейса и логики должны быть максимально

разделены с тем, чтобы изменения интерфейса требовали минимального изменения логики программы.

Основные языки: Java, C#, C++, Python.

Мобильные приложения – это приложения, запускаемые на смартфонах, планшетах. Поскольку сейчас смартфоны есть у каждого (иногда и по несколько штук), то рынок мобильных приложений очень большой и быстро развивается.

Создание мобильных приложений несколько сложнее, чем программирование десктопных программ из-за специфики работы мобильных операционных систем и функционирования смартфонов.

В настоящий момент основные платформы для мобильной разработки:

- Android
- iOS

Возможно, появятся и другие.

Спецификой мобильных приложений является установленный способ их дистрибуции через специальные магазины (Google Play, AppStore).

С одной стороны, это дает широкие возможности для распространения своих приложений, но с другой вынуждает следовать строгим правилам этих магазинов.

Известны случаи, когда мобильные приложения приносили быстро огромные деньги.

Основные языки: Java, Kotlin, C#, JavaScript.

Веб-программирование – это самое распространенное программирование, которое

состоит в создании веб-сайтов, а также мобильных веб-приложений.

Сайт представляет собой набор файлов в формате HTML, картинок, различных данных, а также базы данных, в которой хранится информация о сайте, а также данные пользователей сайта и т.д.

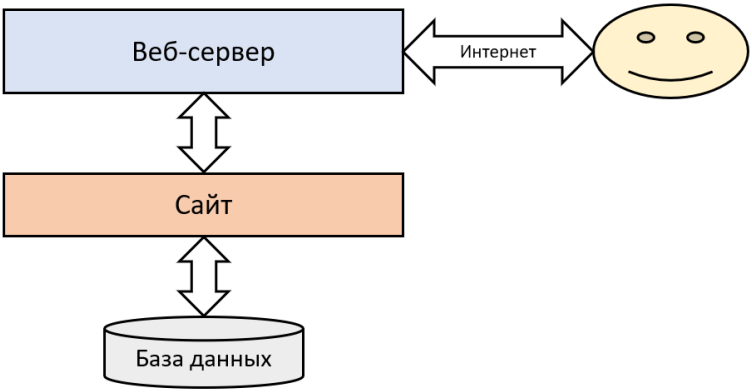

Рис. 8. Схема веб-программ.

Для работы сайта необходим хостинг, на котором установлен веб-сервер. Хостинг – это услуга, предоставляющая хранение файлов сайта, базы данных и работу веб-сервера. Веб-сервер – это сервер, принимающий запросы HTTP и выдающий ответы HTML, картинки и т.д. При этом веб-сервер обеспечивает работу скриптов (PHP и т.п.).

Основные языки разметки: HTML5, языки бэкенд: PHP, Python, C#, фронтенд: JavaScript, TypeScript.

Программирование баз данных – это проектирование и реализация баз данных.

Современные базы данных реализуются СУБД (системы управления базами данных).

Для работы с базами данных обычно используется декларативный язык запросов SQL. С помощь SQL можно создавать, модифицировать и управлять реляционными базами данных. Также с помощью SQL осуществляется работа с данными базы данных.

Наиболее распространенными являются реляционные базы данных, в которых информация хранится в таблицах данных. Также существуют иерархические, сетевые, объектно-ориентированные базы данных.

Наиболее известные СУБД: MySQL, Oracle Database, Microsoft SQL-Server, PostgreSQL и многие другие.

Программирование баз данных – это не только разработка структуры БД и работа с данными, но и программирование триггеров, хранимых процедур на стороне сервера базы данных.

Лекция № 6. Кодирование и криптография

1. Кодирование

Кодирование – это процесс преобразования данных из одной формы, удобной для использования, в форму удобную для хранения, передачи и обработки.

В информатике для компьютерной обработки любые объекты кодируются двоичным кодом. Двоичный код – это кодирование каждого объекта последовательность бит или байт. В компьютере применяются раз личные таблицы кодов, например, ASCII и Unicode.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Рис. 1. Таблица ASCII

Таблица ASCII включает всего 120 символов, расширенная ASCII содержит 256 символов (1 байт). Расширенная таблица содержит национальный алфавит (например, буквы русского языка).

Для кодирования символов всех алфавитов (например, китайские иероглифы) одного байта (256 символов) уже недостаточно. Поэтому большое

распространение получил код Unicode, который состоит из 2 байтов – количество символов $2^{16} = 65536$.

2. Коды, обнаруживающие и исправляющие ошибки.

Важная задача кодирования – это возможность обнаружения ошибок, которые возникают в процессе хранения и/или передачи информации.

Простейший код, который может распознать однократную ошибку – это использование бита четности. Как мы знаем каждый байт состоит из 8-ми битов, например: 01101101. Идея бита четности состоит в том, что к каждому байту добавляем дополнительный бит (бит четности), который должен обеспечить условие, что сумма бит, включая бит четности, должна быть четной. Например, для байта 01101101 бит четности равен 1, а для байта 01010101 бит четности равен 0. Для проверки целостности байта нужно найти сумму битов, включая бит четности, и проверить выполнено ли условие четности суммы.

Можно ли создать код, который мог бы обнаруживать несколько ошибок? Да, для того, чтобы обеспечить обнаружение одной или двух ошибок в коде, можно использовать код с избытком – для этого байт 01101101 будем кодировать (000)(111)(111)(000)(111)(111)(000)(111), где каждый бит мы троировали. В результате любая однократная или двукратная ошибка будет обнаружена, поскольку в этом случае обязательно нарушится тройное повторение в коде.

Код, в котором каждый бит повторяется три раза, способен не только обнаруживать ошибки, но и исправлять. Однократная ошибка в этом коде приведет к тому, что в одной из троек будет (001), (110) и т.д., т.е. будут совпадать только два бита. Поэтому для исправления ошибок нужно выбирать бит голосованием: (010) = 0, (110) = 1 и т.д.

Как нужно строить код, чтобы обнаруживать заданное количество ошибок? Для этого необходимо, чтобы символы кода были достаточно «далеки» друг от друга.

Определим расстояние между символами кода. Пусть каждый символ кодируется последовательностью из N битов $x = (x_1, x_2, \dots x_N)$, $y = (y_1, y_2, \dots, y_N)$. Расстояние ϱ определим следующей формулой:

$$\varrho(x, y) = \sum_{i=1}^{N} |x_i - y_i|$$

где x_i и y_i принимают значения 0 или 1. Если все символы находятся на расстоянии не меньшем, чем d, то код сможет корректировать n ошибок при условии:

$$d \geq 2n + 1.$$

Расстояния между символами должны быть не меньше заданной величины.

3. Основы криптографии.

Криптография – это прикладная наука о методах обеспечения конфиденциальности, целостности и аутентичности данных при хранении и передаче информации.

Задачи криптографии:

1. обеспечение конфиденциальности – это шифрование данных с целью невозможности несанкционированного прочтения информации
2. обеспечение целостности – исключения возможности незаметного изменения информации
3. обеспечение аутентичности – проверка подлинности авторства информации

Значение криптографии переоценить сложно: в наш век все определяется информацией и ее передачей. Поэтому криптография используется не только в военном деле и в спецслужбах, но и всеми – начиная от электронной почты и мобильных телефонов.

Основным методом криптографии является шифрование данных. Простейшая схема шифрования подразумевает, что два абонента имеют возможность договориться о ключе, который будет известен только им, но не злоумышленнику.

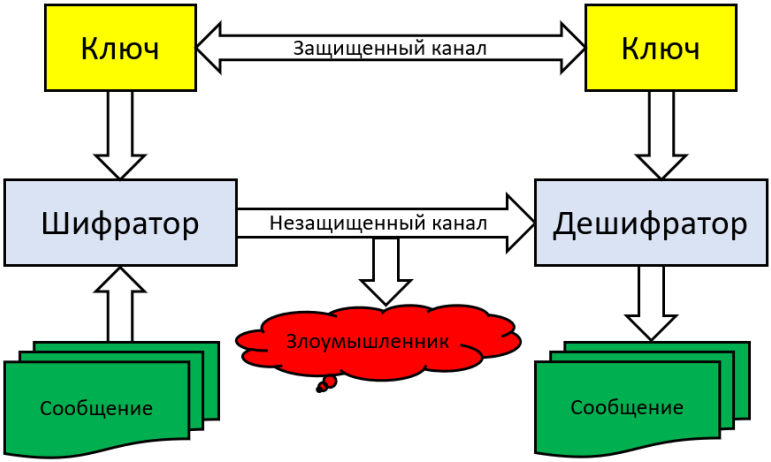

Рис. 1. Шифрование с ключом.

Шифрование с закрытым ключом подразумевает, что у двух абонентов есть возможность договориться о ключе заранее так, чтобы гарантировать, что этот ключ не будет известен. Это условие не всегда возможно. А кроме того, подвержено человеческому фактору.

Простейшее шифрование с ключом состоит в том, мы договариваемся, что букву «А» заменяем на «К», «Б» на «У» и т.д. Эта замена должна быть обратимой. В этой схеме шифром является схема «А»-«К», «Б»-«У», … Это шифрование совершенно не эффективно: мало того, что оно легко взламывается, так оно еще и требует большого ключа.

Шифр Цезаря состоит в том, что каждая буква заменяется на другую со сдвигом на секретное число: «А» на «Г», «Б» на «Д» и т.д. «Я» на «В». Этот шифр

имеет историческое значение и очень легко взламывается.

Математически формула для шифра с ключом имеет следующий вид. Пусть T – это сообщение, K – это ключ, а C – это зашифрованное сообщение. Тогда должны существовать две функции E – шифрующая и D – дешифрующая такие, что:

C = E(T, K) и T = D(C, K)

Такие системы шифрования называются симметричными криптосистемами, поскольку для зашифрования и расшифрования используются один и тот же ключ. Примеры: AES, DES, ГОСТ 28147-89, RC2, IDEA и много других.

Шифр Вернама является абсолютно криптостойким, но при этом очень простым. проблема только в том, что нужны одноразовые шифры, длина которых равна шифруемому сообщению. Поэтому этот шифр используется только в редких случаях.

Операция XOR – «Исключающего ИЛИ»:

0 XOR 0 = 0 1 XOR 0 = 1
0 XOR 1 = 1 1 XOR 1 = 0

T = 011010111 – сообщение
K = 101010101 – ключ

C = T XOR K и T = C XOR K – шифрование/расшифрование

C = 110000010 – шифрованное сообщение

Таким образом, если ключ сгенерировать случайным образом и использовать для каждого шифрования один раз, то взломать такой шифр невозможно.

Вариантом этого шифра является использования одноразовых шифроблокнотов. Шифроблокнот – указание замены букв (с учетом номера буквы).

Шифр Штирлица: указываем номер страницы и номер буквы на этой странице по секретной книге.

4. Элементы теории чисел.

Будем рассматривать натуральные числа N = {0, 1, 2…}. Число p > 1 называется простым, если p делится только на 1 и на p. Простые числа: 2, 3, 5, 7, … Простых чисел бесконечное число, но они встречаются очень редко. Не существует быстрого алгоритма для нахождения больших простых чисел, а также для проверки является ли число простым.

Если P и Q – большие простые числа и X = PQ, то не зная чисел P и Q, практически не возможно их найти по X, но зная X и P, легко найти Q = X / P.

Пусть a и b > 0 натуральные числа, тогда число c = a mod b – это остаток от деления a на b. Очевидно, что c ∈ {0, 1, …, b – 1}. Например, 17 mod 3 = 2, 15 mod 5 = 0.

При создании шифров с открытым ключом принципиальное значение имеет теорема Ферма:

пусть p – простое число и $0 < a < p$, тогда имеет место $a^{p-1} \bmod p = 1$.

5. Хранение паролей.

Проблема хранения паролей в открытом доступе является крайне актуальной. Если мы на сайте пишем проверку пароля, но злоумышленник имеет доступ к нашему коду, то мы не можем сравнивать вводимый пароль с хранимым. Для этого используют различные односторонние функции.

Функция $y = f(x)$ называется односторонней, если она однозначна, легко вычисляется, но ее обратная функция $x = f^{-1}(y)$ является трудно вычислимой.

Пример односторонней функции $y = a^x \bmod p$, где p – простое число, а x от 0 до p – 1.

Пусть x – пароль. Тогда мы вычисляем $y = f(x)$. И храним только y. Когда пользователь вводит пароль x' мы вычисляем $y' = f(x')$ и сравниваем y с y'. Если они равны, то пользователь ввел правильный пароль. Противник может знать y и вид функции f(x), но этой информации ему не хватит, чтобы подобрать такое x, чтобы $f(x) = y$, поскольку для этого нужно вычислить

$$x = f^{-1}(y),$$

что сделать для односторонних функций очень сложно, практически невозможно. Такие функции называются хеш-функциями.

Широко распространена функция MD5. Неизвестно о фактах ее взлома.

6. Система «свой-чужой».

Проблема «свой-чужой» состоит в том, чтобы с самолета в открытом доступе передать пароль ПВО, чтобы отличить свой самолет от чужого. Если просто каждому самолету присвоить пароль, то противник может узнать этот пароль и потом повторить.

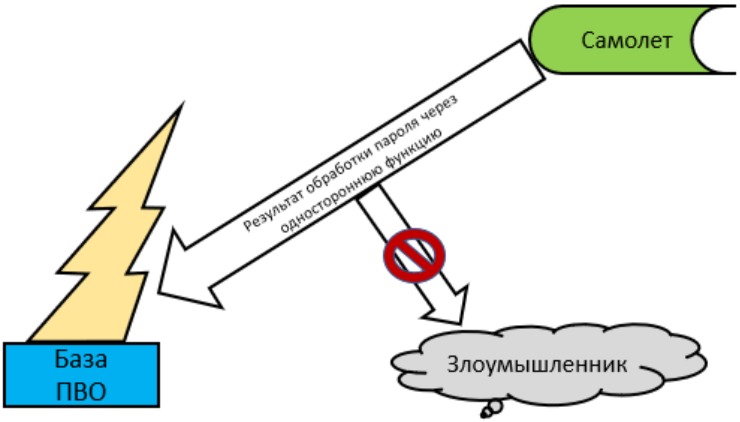

Рис. 2. Система «свой-чужой».

Поэтому самолет передает по открытому каналу не пароль, а пароль, прогнанный через одностороннюю функцию. Но чтобы эта информация не повторялась самолет должен шифровать не только свой позывной (пароль), но и текущие дату и время:

y = f(“позывной” + дата + время)

Передаем y по открытому каналу. На базе ПВО также вычисляют y, используя предполагаемый позывной и дату/время.

7. Шифрование с открытым ключом.

Крупнейшее достижение криптографии – это шифрование с открытым ключом. Исходная задача такова: как организовать шифрованное сообщение между абонентами, если у них нет секретного канала связи, а только открытый? Симметричное шифрование с ключом не годится, поскольку нет возможности передать секретный пароль.

Рассмотрим систему Диффи-Хеллмана. Выбираем большое простое число p и число g такое, что $1 < g < p - 1$ и все числа $\{1, 2, \ldots, p - 1\}$ могут быть представлены как различные степени ($g \bmod p$). Эти числа известны всем.

1. Каждый k-й абонент загадывает секретное число X_k, которые хранятся в тайне.

2. Каждый k-й абонент вычисляет число $Y_k = g^{Xk} \bmod p$, которое обнародует

3. Если m-й абонент хочет послать сообщение k-му абоненту сообщение, то он вычисляет $Z_{mk} = (Y_k)^{Xm} \bmod p$. k-й абонент аналогично вычисляет число Z_{km}. Можно доказать, что $Z_{mk} = Z_{km} = Z$. Таким образом, два абонента получили одинаковое число, которое не передавалось по открытому каналу и которое никто другой вычислить не может, поскольку никто не знает Y_k и Y_m.

4. Число Z используется в качестве ключа для симметричного шифра.

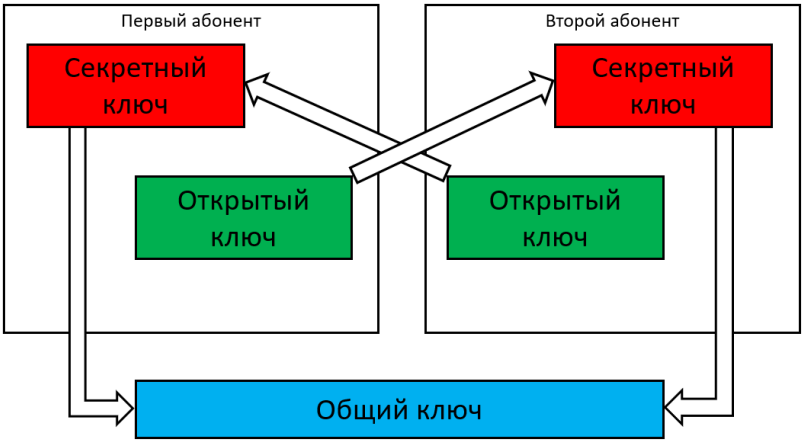

Рис. 3. Система с открытым ключом.

Алгоритм Диффи-Хеллмана был открыт в середине 70-х годов и привел к революции в криптографии. В эпоху интернета, когда абоненты в принципе не имеют закрытого канала, открытые ключи играют выдающуюся роль.

Рассмотрим пример использования системы Диффи-Хеллмана.

Выберем p = 23, g = 5.

Пусть первый абонент выбирал $X_1 = 7$, а второй $X_2 = 13$.

Вычисляем:

$Y_1 = 5^7 \bmod 23 = 17,$

$Y_2 = 5^{13} \bmod 23 = 21.$

$Z_{12} = 21^7 \bmod 23 = 10,$

$Z_{21} = 17^{13} \bmod 23 = 10$.

Таким образом, Z = 10 это их взаимный ключ.

Наиболее распространенным шифром с открытым ключом является алгоритм RSA (от имен Rivest, Shamir, Adleman). Этот шифр основан на том, что если есть два больших простых числа P и Q, то задача разложения N = PQ на простые сомножители очень сложная.

1. Каждый абонент выбирает по два больших простых числа P и Q, N = PQ.

2. После этого каждый абонент вычисляет r = (P – 1)(Q – 1) и выбирает число d < r.

3. Находит число c такое, что cd mod r = 1.

4. Каждый абонент обнародует числа d и N.

При передачи сообщения m от первого абонента второму (считаем, что $m < N_2$):

1. Шифруем $e = m^{d2} \bmod N_2$

2. По открытому каналу пересылаем число e.

3. Второй абонент расшифровывает сообщение по формуле $m = e^{c2} \bmod N_2$.

8. Электронная подпись.

Электронная подпись документа решает задачу гарантии, что подписанный документ принадлежит автору и не был изменен. Причем, важно, что эту проверку может сделать любой, кто имеет текст и электронную подпись.

Автор:

1. Выбирает два больших простых числа P и Q.

2. Вычисляет числа N = PQ и r = (P – 1)(Q – 1).

3. Выбирает число d, которое является взаимно простым с r.

4. Вычисляет число $c = d^{-1} \bmod r$.

5. Автор публикует числа N и d, а число c хранит в тайне.

Далее, автор текста m:

1. Вычисляет хеш-функцию от m, например, y = MD5(m).

2. Вычисляет число $s = y^c \bmod N$, которое является подписью.

Для проверки текста m вычисляем $w = s^d \bmod N$. Если w = MD5(m), то текст m – был подписан цифровой подписью s.

9. Электронные деньги.

Цифровые деньги предназначены для обеспечения анонимности покупок:

1. Покупатель снимает сумму со своего счета в банке.

2. Покупатель пересылает деньги в магазин.

3. Магазин сообщает об этом банк, сумма зачисляется на счет магазина

4. Покупатель забирает товар.

Задача электронных денег состоит в том, чтобы описанная выше схема была надежна и обеспечивала анонимность.

1. Банк выбирает секретную информацию: большие простые числа P и Q и число c.

2. Вычисляет числа $N = PQ$, $d = c^{-1} \bmod (P - 1)(Q - 1)$.

3. Числа N и d публикуются.

Рассмотрим одностороннюю функцию $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ – взаимно однозначную, но с трудно вычислимой f^{-1}. Эта функция несекретная. Электронная банкнота – это пара чисел <n, s>, где $s = (f(n))^c \bmod N$. Банкнота верная, если $s^d \bmod N = f(n)$ – это может проверить любой!

Лекция № 7. Реляционные базы данных

1. Системы управления базами данных.

СУБД – системы управления базами данных играют принципиальную роль в информационных технологиях, поскольку представляют собой программные средства для управления базами данных, которые хранят информацию.

Наиболее распространенные системы управления баз данных являются клиент-серверные реляционные СУБД.

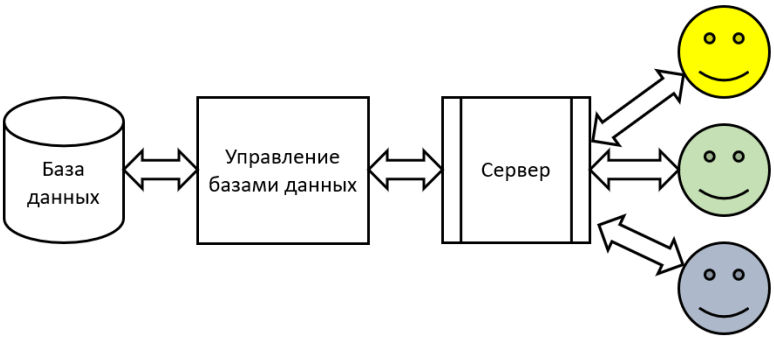

Рис. 1. Системы управления базами данных.

Системы управления базами данных на основе технологий «клиент-сервер» позволяют одновременно работать с базами данных большому количеству пользователей. В частности, эти технологии используются для работы баз данных, используемых сайтами.

2. Реляционные базы данных.

Реляционные базы данных представляют собой набор таблиц, связанных между собой внешними ключами. Основным элементом реляционной БД являются таблицы, которые приставляют собой динамический набор записей, где каждая запись описывает какой-либо объект. При этом строками в таблице являются записи, а столбцами поля в записи.

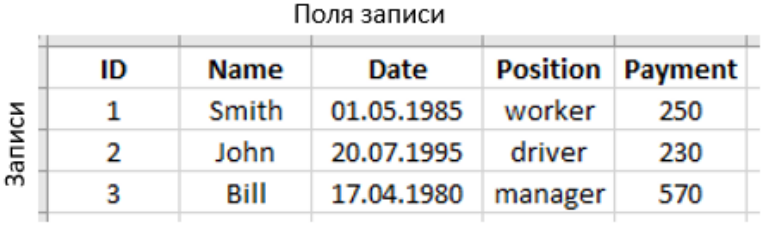

Рис. 2. Пример таблицы.

Каждая запись представляет собой информацию об определенном объекте, а каждое поле может иметь свой тип: целое, дробное, строка, дата, текст.

Для идентификации различных записей обычно используют определенное поле, которое называется идентификатором ID. Для этого нужно обеспечить, чтобы разные объекты (записи) имели различные идентификаторы. Например, имя не может быть идентификатором сотрудников, поскольку бывают однофамильцы. Иногда используются идентификаторы, присущие объектам (номер паспорта, гос. номер машины и т.д.), либо эти

идентификаторы присваиваются объекту при создании новой записи в таблице.

3. Язык SQL.

Для работы с реляционными базами данных, как правило, используется декларативный язык программирования SQL (structured query language) – «язык структурированных запросов». Различные СУБД имеют различные диалекты SQL. Мы будем рассматривать на примере СУБД MySQL, которая наиболее часто применяется в веб-программировании.

Создадим базу данных:

```
CREATE DATABASE Personal;
```

Для удаления базы данных:

```
DROP DATABASE Personal;
```

Язык SQL является регистронезависимым, но обычно принято команды SQL писать заглавными буквами.

Попытка создать существующую БД или удалить не существующую приведет к ошибке. Поэтому:

```
CREATE DATADASE IF NOT EXIST Personal;
DROP DATABASE IF EXIST Personal;
```

Для подключения базы данных команда: USE Personal;

При создании таблиц базы данных в SQL необходимо указать какие у таблицы будут поля и какого типа будут эти поля. Рассмотрим пример.

```
CREATE TABLE Person
(
   ID INT,
   Name VARCHAR(50),
   Birthday DATETIME,
   Organization VARCHAR(250),
   Note TEXT
);
```

После создания таблицы ее можно изменять. Например, добавить поле:

```
ALTER TABLE Person ADD Email VARCHAR(50);
```

При определении полей в таблице кроме типа можно указывать различные атрибуты, которые позволяют указать дополнительную информацию об этих полях.

Уникальный идентификатор записи называется первичным ключом. Первичный ключ может состоять из одного поля или нескольких. Поля, являющиеся первичными ключами, должны быть всегда заполненными и уникальными.

```
CREATE TABLE Person
(
   ID INT PRIMARY KEY,
…….
);

CREATE TABLE Person
(
   ID INT,
…….
   PRIMARY KEY(ID)
);

CREATE TABLE Person
(
   Name VARCHAR(50),
   Surname VARCHAR(50),
   PRIMARY KEY(Name, Surname)
);
```

Если таблица описывает какой-либо объект, то желательно всегда явно указывать первичный ключ. Но использование первичного ключа налагает определенные ограничения на вводимую информацию.

Рассмотрим еще несколько атрибутов, которые часто используются при создании таблиц.

ID INT PRIMARY KEY AUTO_INCREMENT, - это означает, что данное поле не нужно заполнять, оно будет само заполняться, увеличиваясь на 1.

Email VARCHAR(50) UNIQUE, - атрибут уникальности поля. БД не даст создать запись, если такое значение поля уже есть в таблице.

Name VARCHAR(50) NOT NULL, - это значит, что данное поле не может иметь значение NULL, т.е. быть незаполненным.

Age INT DEFAULT 20, - атрибут DEFAULT задает значение по умолчанию.

CONSTRAINT My_Age CHECK(Age > 18 AND Age < 65), - задаются автоматические ограничения. Идентификатор My_Age – это идентификатор ограничения.

Важнейшим достижением реляционных баз данных является использования внешних ключей у различных таблиц. Рассмотрим пример:

```
CREATE TABLE Organization
(
   ID INT PRIMARY KEY,
   Name VARCHAR(250)
)
CREATE TABLE Person
(
……
   Organization VARCHAR(250),
……
);
```

Проблема состоит в том, что одна и тоже организация может быть у разных персон и в ссылаться на ее имя плохо – в случае переименовании

нужно будет менять у всех. Для решения этой задачи используем внешний ключ, который задаст связь между таблицами.

```
CREATE TABLE Person
(
……
  ID_ Organization INT,
……
  FOREIGN KEY (ID_ Organization) REFERENCES Organization (ID)
);
```

Для добавления записей в таблицу используется следующая команда

INSERT Person (ID, Name, Birthday) values (120, ‘Smith’, 1993-05-17 14:10:00);

Если в этой команде будут пропущены какие-либо поля, то они будут заполнены значением NULL, если какие-то из этих полей будут с атрибутом NOT NULL, то возникнет ошибка. Если поле имеет атрибут DEFAULT, то будет значение, указанное в атрибуте. Поля, имеющие атрибут AUTO_INCREMENT, не нужно заполнять. Если поле связано внешним ключом, то вставляемое значение должно ссылаться на существующее значение.

Можно вставлять и несколько строк сразу:

```
INSERT Person (ID, Name, Birthday) values
(120, ‘Smith’, 1993-05-17 14:10:00),
(140, ‘John’, 1995-03-20 15:30:10);
```

Для чтения данных из таблицы используется мощный оператор SELECT

```
SELECT * FROM Person;
```

Этот оператор вернет выборку, содержащую все записи, включающие все поля.

Если нужно вернуть не все поля, то можно указать какие нужны:

```
SELECT Age, Birthday FROM Person;
```

Если нужны не все записи, то можно указать условие:

```
SELECT Age, Birthday FROM Person WHERE Age > 20 AND Age <= 30;
```

Для сортировки используется следующая команда:

```
SELECT * FROM Person ORDER BY Age;
```

Допустим, что мы хотим получить выборку всех персон с указанием названия организации, но по условию таблица Person ссылается только на ID организации. Для решения этой задачи используем:

SELECT Person.Name, Person.Age, Organization.Name FROM Person, Organization WHERE Person.ID_Organization = Organization.ID;

Аналогично можно делать выборки из любого количества таблиц.

Для удаления строк используется оператор DELETE:

DELETE FROM Person; - удалить все записи из таблицы Person;

DELETE FROM Person WHERE Age < 18; - удалить все записи, у которых Age < 18.

Чтобы изменить некоторые поля в записях в таблице используется оператор UPDATE:

UPDATE Person SET Age = Age + 1;

увеличить во всех записях поле Age на 1.

Чтобы изменить только в некоторых записях используем конструкцию WHERE:

UPDATE Person SET Name = 'Paul', Age = 23 WHERE ID = 120;

С операторов UPDATE также, как и с оператором DELETE нужно быть осторожным, поскольку действия этих операторов отменить будет нельзя!

3. О методах проектирования баз данных.

При создании баз данных принципиальное значение имеет проектирование базы данных с учетом логической структурой предметной области. Для этого используются различные подходы:

- Метод моделирования "сущность-связь" (ER modeling) дает абстрактную модель предметной области, используя следующие основные понятия: сущности (entities), взаимосвязи (relationships) между сущностями и атрибуты (attributes) для представления свойств сущностей и взаимосвязей.
- Методы моделирования временных данных (Temporal data modeling) дают абстрактную модель фрагмента предметной области, представляющего временные ряды данных, и используют следующие основные понятия: временные метки (timestamps), временной ряд (time series), дата, диапазон дат, классы.
- Метод моделирования "свод данных" (Data Vault) дает абстрактную модель фрагмента предметной области, основываясь на математических принципах нормализации отношений, и использует следующие основные понятия: сущности-концентраторы (Hub Entities), связывающие сущности (Link Entities), сущности-сателлиты (Satellite Entities).

Лекция № 8. Основы машинного обучения

В последнее время машинное обучение является мейнстримом современных компьютерных технологий. Можно уверенно сказать, что кроме умения программировать именно знание основ и практического применения методов машинного обучения становится стандартом для IT специалиста.

Сейчас более модным является словосочетание «машинное обучение», нежели слова «искусственный интеллект», поскольку искусственный интеллект звучит громко, напоминая о роботах всемогуторах, которые имеют мышление и могут быть аналогами человека... Сейчас принято разделять сильный искусственный интеллект и слабый искусственный интеллект. Под слабым искусственным интеллектом принято понимать те, обучающиеся системы, которые могут хорошо решать отдельные задачи. Все основные достижения в области искусственного интеллекта относятся именно к слабому искусственному интеллекту. Как правило, говоря о машинном обучении, говорят о слабом искусственном интеллекте.

1. Генетические алгоритмы.

Генетические алгоритмы представляют собой алгоритмы коллективной оптимизации многомерных функций. Особенностью генетических алгоритмов является разбиение искомого решения на части. При этом приближения к решению называются генами, а составные части – хромосомами.

Допустим, мы ищем минимум функции

$y=f(x_1, x_2, x_3)$

трех переменных. В этом случае генами будут вектора $x=(x_1, x_2, x_3)$, а хромасомами – числа x_1, x_2 и x_3.

Следующая идея генетических алгоритмов состоит в том, что мы будем одновременно оптимизировать не один ген, а целую популяцию генов. При этом в процессе оптимизации используются следующие основные процедуры:

1. Порождение новых генов случайным образом.
2. Скрещивание генов.
3. Мутация генов.

Операция скрещивания генов является основной в генетических алгоритмах. Под скрещиванием мы будем понимать операцию создания нового гена, у которого часть хромосом от одного гена, а часть хромосом от другого гена. При этом те гены, которые являются лучшими относительно оптимизируемой функции, чаще участвуют в скрещивании.

Под мутацией мы будем понимать операцию, когда у гена случайным образом изменяются одна или несколько хромосом.

Исходной задачей является минимизация функции. В начале мы рассмотрим вариант функции нескольких переменных. Пусть задана функция $F:R^n \rightarrow R^n$, которую для простоты мы будем считать неотрицательной $F(x) \geq 0$. Множество, на котором мы

будем оптимизировать функцию, обозначим через D. Для простоты мы будем считать, что множество D представляет собой некоторый прямоугольник

$$D = \{x \in R^n: a_i \le x_i \le b_i, i=1,2, \ldots, n\}.$$

Задача состоит в том, чтобы найти такой вектор $x^* \in D$, что

$$\min_{x \in D} F(x) = F(x^*).$$

В генетическом алгоритме нам нужно будет ранжировать особи (гены) по их качеству, т.е. близости к оптимальному решению. В генетических алгоритмах для оценки этой близости используют термин – приспособленность особи. При этом приспособленность оценивается с помощью функции приспособленности. В качестве функции приспособленности можно использовать и целевую функцию, но лучше использовать безразмерный вариант. Мы предлагаем использовать функцию приспособленности для особи x, заданную по следующей формуле

$$Fit(x) = \frac{1}{1 + F(x)}.$$

Ясно, что для любого $x \in D$ и любой неотрицательной функции мы имеем

$0 < Fit(x) \leq 1$.

Мы будем говорить, что особь x' более приспособленная или лучше, чем особь x'', если

$Fit(x') > Fit(x'')$.

Если мы нашли такую особь, что Fit(x)=1, то эта особь будет идеальной.

Генетический алгоритм предполагает, что что мы одновременно имеем пул различных особей, которые будут эволюционировать по приведенным выше правилам. Покажем, как эти правила будут реализованы в нашем случае.

Порождение случайной особи $x \in D$. Случайная особь генерируется по следующей формуле

$$x = (\xi_1, \xi_2, \ldots, \xi_n), \quad \xi_i \sim R(a_i, b_i),\ i = 1, \ldots, n.$$

Здесь ξ_i – это случайные величины, равномерно распределенные на отрезке $[a_i, b_i]$.

Скрещивание генов. Пусть мы имеем две особи $x, y \in D$. Вычислим вероятности

$$p = \frac{Fit(x)}{Fit(x) + Fit(y)}, \quad q = \frac{Fit(y)}{Fit(x) + Fit(y)}.$$

Очевидно, что $p + q = 1$.

В результате их скрещивания мы получаем новую особь

$z=(z_1, z_2, \ldots, z_n)$,

где

$$z_i = \begin{cases} x_i, & \xi \le p \\ y_i, & \xi > p \end{cases}$$

где $\xi \sim R(0,1)$ - равномерно распределенная случайная величина. Другими словами, z_i равна x_i с вероятностью p и y_i с вероятностью q. При этом видно, что чем более приспособлена особь, тем больше ее генов войдут в потомка.

Случайная мутация. Мутацией особи $x \in D$ мы назовем особь

$$\tilde{x} \in D,$$

которая строится по следующему правилу. Во-первых, выбираем случайный номер $1 \le j \le n$. Во-вторых, вычисляем случайную

величину

$$\delta_j = \delta_j(x_j)$$

такую, что $j + \delta_j \in [a_j, b_j]$.

После этого особь $\tilde{x}$ состоит из компонент

$$\tilde{x}_i = \begin{cases} x_i, & i \neq j \\ x_i + \delta_i, & i = j \end{cases}$$

Есть различные варианты мутации. Например, можно изменять не одну хромосому, а несколько. Кроме того, можно изменять выбранную хромосому на близкую хромосому, а можно на случайную величину.

Рассмотрим некоторые примеры использования генетических алгоритмов. Первый пример относится к традиционной задаче нахождения минимума функции. Рассмотрим функцию

$$F(x) = \sum_{k=1}^{n} x_k^2 \cdot (1 + |\sin(100 \cdot x_k)|),$$

заданную на R^n. Мы будем рассматривать n = 5 . Эта функция имеет большое количество локальных минимумов. Легко, однако видеть, что глобальным минимумом этой функции является точка x=0.

Применяя к задаче нахождения минимума этой функции генетический алгоритм, мы будем использовать следующие параметры:

Количество особей пуле: M=1000.

Количество убиваемых наихудших особей: M_C=200.

Количество итераций: L=1000.

На рис. 1 мы приведем значения целевой функции для первых 10 итераций.

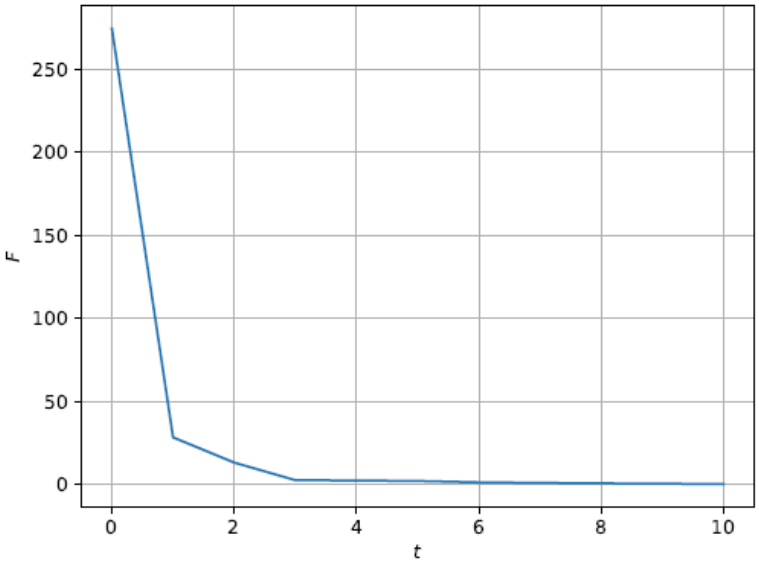

Рис. 1. Значения целевой функции.

Мы видим, что за считанное количество итераций значение целевой функции быстро приближается к собственному минимуму.

Рассмотрим другую задачу. Пусть у нас есть конечное множество натуральных чисел A. Необходимо разбить данное множество на два подмножества

$$A = K_1 \cup K_2, \;\; K_1 \cap K_2 = \varnothing$$

таким образом, чтобы величина

$$H(K_1, K_2) = \left| \sum_{a_k \in K_1} a_k - \sum_{a_m \in K_2} a_m \right|$$

достигала минимального значения.

Хорошо известно, что эта задача является NP-полной. Применим для ее решения генетический алгоритм. Сначала введем кодировку в данной задаче. Пусть мощность множества A равна N. В качестве хромосомы мы введем N-мерный вектор

$$x = (x_1, x_2, \ldots, x_N),$$

каждая компонента которого принимает только два значения 0 или 1. При этом если $x_k = 1$, то мы будем считать, что элемент $a_k \in A$ принадлежит множеству K_1, в противном случае этот элемент принадлежит множеству K_2.

Рассмотрим применение генетического алгоритма в случае, когда N = 5000, а множество A состоит из следующих элементов

$$A = \{1, 2, \ldots, 5000\}.$$

Для генетического алгоритма мы будем использовать следующие параметры:

Количество особей пуле: M=1000.

Количество убиваемых наихудших особей: M_C = 200.

Количество итераций: L = 100.

Результаты вычислений приведены на рис. 2

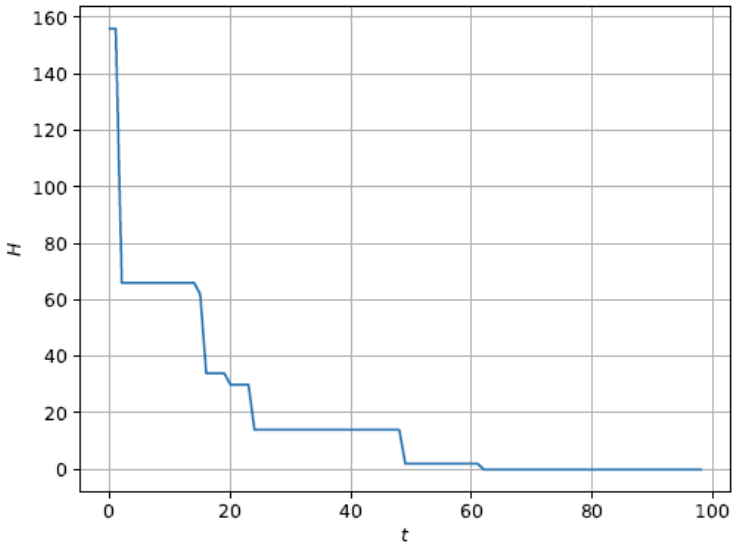

Рис 2. Результаты вычислений.

2. Обучение персептрона

Персептрон представляет собой математическую модель нейронов, которые являются составной частью нейронных сетей. При этом и единичный нейрон (персептрон) позволяет решать задачу линейной классификации многомерных объектов.

Пусть мы имеем множество S (конечное или бесконечное) n-мерных векторов

$$x = (x_1, x_2, \ldots, x_n).$$

При этом будем считать, то это множество разбивается на два класса, которые мы будем обозначать через +1 и -1.

Таким образом, персептрон является функцией

$$y : \mathbb{R}^n \to \{-1, +1\}.$$

Эта функция должна разделять множество S на два класса. Функция персептрона устроена следующим образом

$$y = \text{sign}(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n),$$

где w = (w_0, w_1, w_2, …, w_n) – вектор весов персептрона, а sign(t) – функция знака

$$\text{sign}(t) = \begin{cases} -1, & t < 0 \\ +1, & t \geq 0 \end{cases}$$

Весовой коэффициент w_0 называется порогом активации. Можно считать, что у персептрона есть еще одни вход x_0, который всегда равен $x_0 = 1$.

Персептрон полностью задается вектором своих весов w. Если задать какой-либо вектор w, то персептрон будет разделять пространство R^n на два множества, причем граница этого раздела будет – гиперплоскость, которая определяется уравнением

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = -w_0.$$

Основной вопрос обучения персептрона состоит в определении весовых коэффициентов. Процедура машинного обучения персептрона основана на имеющейся обучающей выборке.

Пусть мы имеем набор обучающих данных

$$(x^m, d^m),\ m = 1, 2, \ldots, M,$$

где $x^m \in S$, а $d^m \in \{-1, 1\}$.

Приведем алгоритм обучения персептрона. В этом алгоритме используется параметр скорости обучения $0 < \alpha < 1$.

1. Положить вектор весов равным нулю

w=(0, 0, …, 0).

2. Повторять N раз следующие шаги.
3. Для каждого тестового набора (x^m, d^m):
4. Вычислить

y = sign[(w, x^m)]

5. Если $yd^m < 0$, то скорректировать веса

$$w_0 = w_0 + \alpha d^m,$$

$$w_i = w_i + \alpha d^m x_i^m,\ i = 1, 2, \ldots, n.$$

Успех обучения персептрона обеспечивается линейной разделимостью множества, на котором мы обучаем персептрон. Можно доказать, что если множество линейно разделимо, то процедура обучения, при адекватно выбранной константе обучения, всегда приведет к тому, что обученный персептрон будет разделять это множество.

Если же множество не является линейно разделимым, то есть не существует гиперплоскости, которая разделит два класса, то, очевидно, что невозможно обучить персептрон никаким образом.

В частности, невозможно обучить персептрон, так, чтобы он смог вычислить операцию XOR. В свое время осознание этого факта привело к тому, что популярность персептронов и нейронных сетей резко упала. Однако рассмотренное ограничение на линейную разделимость классифицируемого множества имеет место только для однослойных нейронных сетей. Для нейронных сетей, содержащих более одного слоя, этих ограничений уже нет.

Рассмотрим пример обучения персептрона. Для наглядности мы будем использовать размерность n = 2. В этом случае гиперплоскость, определяемая обученным персептроном, будет представлять собой линию.

Обучим персептрон таким образом, чтобы он смог определить по двум числам, какое из больше. Обучающую выборку мы сформируем из пары случайных чисел $x = (x_1, x_2)$. Вектор x мы отнесем к классу -1, если

$$x_1 \leq x_2$$

и к классу 1 в противном случае.

На рис. 3 мы приведем пример обучающей выборки.

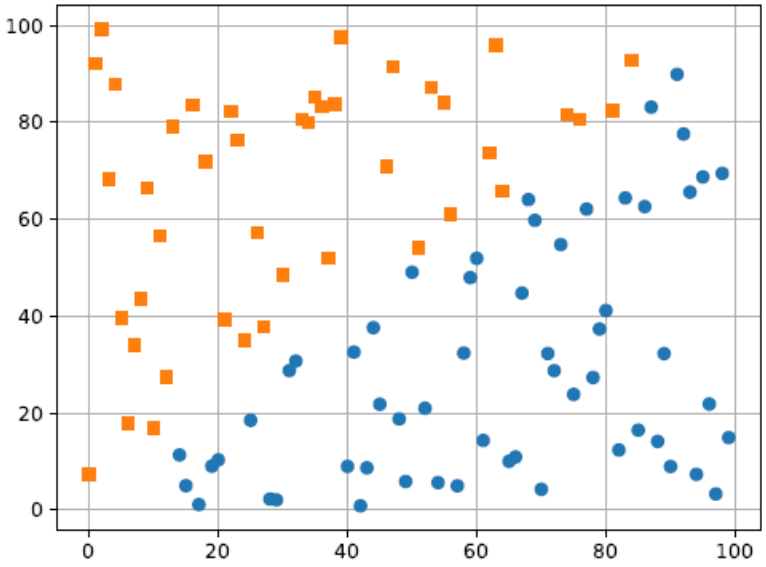

Рис. 3. Обучающее множество для персептрона.

Перейдем к обучению нашего персептрона. Как мы уже говорили, обученный персептрон представляет собой линию, которая разделяет два множества, см. рис. 4.

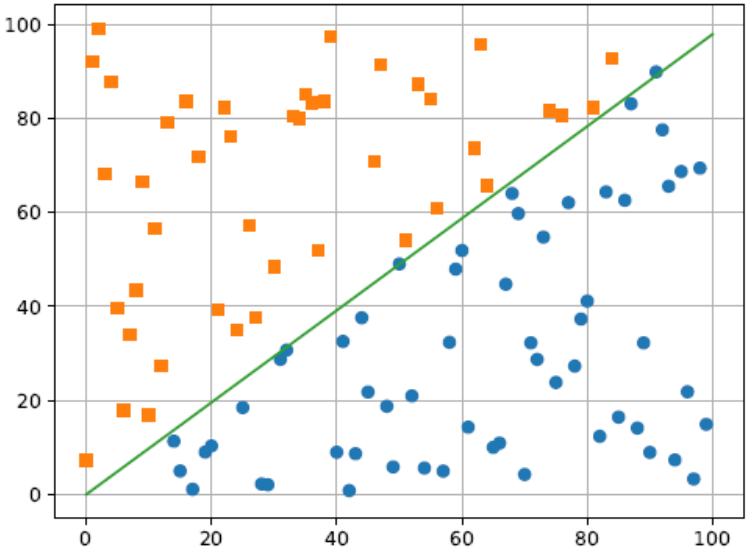

Рис. 4. Разделение множества линией персептрона.

Теперь рассмотрим пример, когда обучающее множество не является линейно разделимым. Обучающая выборка состоит из четырех элементов

((0, 0), -1), ((0, 1), 1), ((1, 0), 1), ((1, 1), -1).

Это соответствует логической операции исключающего ИЛИ.

Это обучающее множество можно видеть на рис. 5.

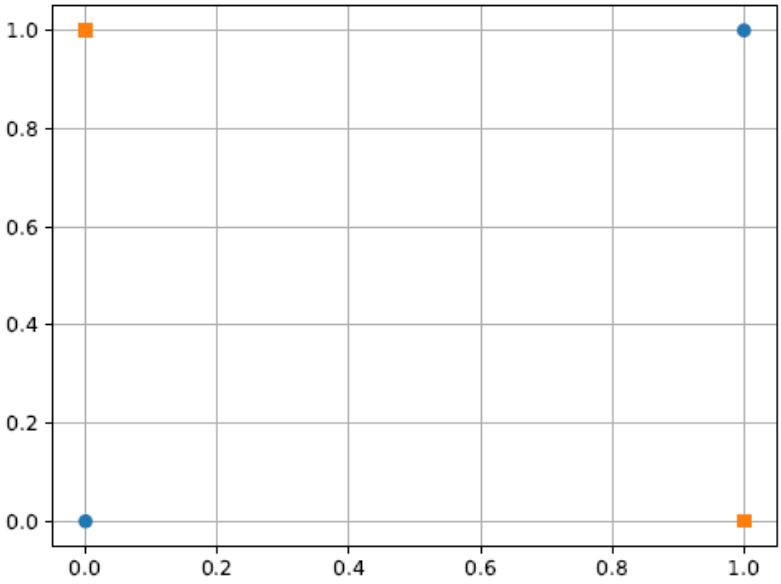

Рис. 5. Обучающее множество.

Ясно, что это множество не является линейно разделимым, поэтому обучение персептрона будет неэффективным. Взяв константу обучения alpha = 0.1, мы получим после обучения следующие веса

$w_0 = 0, w_1 = 0.1, w_2 = 0.$

Уравнению

$0.1\ x = 0$

удовлетворяет вертикальная линия, см. рис. 6.

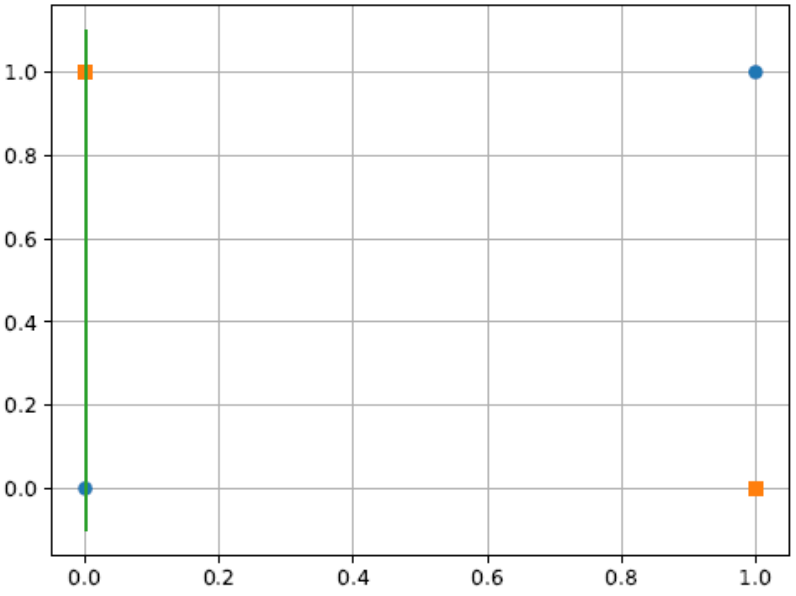

Рис. 6. Обученный персептрон.

Видно, что в данном случае персептрон не смог разделить данное множество.

Литература

[1] Мотвани Р., Ульман Д.Д., Хопфорт Д.Э. Введение в теорию автоматов, языков и вычислений. – М.: ООО «И.Д. Вильямс», 2015.

[2] Громкович Ю. Теоретическая информатика. Введение в теорию автоматов, теорию вычислимости, теорию сложности, теорию алгоритмов, рандомизацию, теорию связи и криптографии. – СПб.: БХВ-Петербург, 2010.

[3] Духин А.А. Теория информации. – М.: «Гелиос АРВ», 2007.

[4] Рябко Б.Я., Фионов А.Н. Основы современной криптографии и стеганографии. – М.: Горячая линия – Телеком, 2016.

[5] Шамин Р.В. Машинное обучение в задачах экономики. – М.: «Грин Принт», 2019.

[6] Яблонский С.В. Введение в дискретную математику. – М.: Наука, 1986.

Учебное издание

Шамин Роман Вячеславович

Лекции по информатике

Художник *Михаил Шамин*

Формат 60 x 90 1/16. Бумага офсетная. Печать цифровая

Усл. печ. л. 7,25. Зак. № 30447. Тираж 600 экз.

ООО «Грин Принт». 105318, г. Москва, Измайловское ш., д. 28

Тел.: +7(495)118-09-26