Лабораторная работа №1 Кодирование и шифрование

Предмет: Практическая линейная алгебра

Автор: Made by Polyakov Anton, the part of R3236, suir family

Преподаватель: Алексей Алексеевич Перегудин

Source matlab code

Шифр Хилла

Возьмем в качестве алфавита = русский алфавит и пробел:

А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й
1	2	3	4	5	6	7	8	9	10	11
К	Л	М	Н	О	П	Р	С	Т	У	Ф
12	13	14	15	16	17	18	19	20	21	22
Х	Ц	Ч	Ш	Щ	Ь	Ы	Ъ	Э	Ю	Я
23	24	25	26	27	28	29	30	31	32	33

```
abc = ['А', 'Б', 'В', 'Г', 'Д', 'Е', 'Ё', 'Ж', 'З', 'И', 'Й', 'К', 'Л', 'М', 'Н', 'О', 'П', ...
    'Р', 'С', 'Т', 'У', 'Ф', 'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ъ', 'Ы', 'Ь', 'Э', 'Ю', 'Я', ' '];
n = length(abc)
```

```
n = 34
```

```
msg = 'Я ЯБЛОКО СОК'
```

```
msg =
'Я ЯБЛОКО СОК'
```

```
m = length(msg)
```

```
m = 12
```

Генерируем ключи

Генерируем матрицу N*N со случайными целыми числами в диапазоне a...b

```
% key4= randi([1 100],4,4);
% det(key4);
```

Не иметь общих делителей <=> НОД == +-1

```
% if gcd(det(key4),length(abc)) == 1
%     key4
% end
```

Сделав пару итераций программы под разные размерности, выпишем матрицы...

```
key3 = [68 47 64;
        1 78 37;
        8 82 89];

key2 = [33, 95;
        73, 10];

key4 = [28 1 3 80;
        60 57 69 25;
        29 37 56 20;
        92 38 77 37];
```

Как работает алгоритм в случае ключей 3*3?

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

P - векторы-столбцы исходный текст (open)

C - векторы-столбцы зашифрованный текст (closed)

K - key - ключ-матрица шифрования

Операции выполняем по модулю n=3

Чтобы расшифровать, надо получить обратную матрицу ключа:

inv_K = K^-1 **(mod n)**

Тогда.....

Шифрование: C = K*P (mod n)

Расшифрование: P= inv_K * C(mod n)

Работа с матрицей 2x2

Шифрование

```
string_vector_1 = [33, 33, 13, 12, 34, 16;
                   34, 2, 16, 16, 19, 12];

% Посчитаем первый элемент
two_letters_vector = string_vector_1(:,1);
encrypted_string = '';
encryped_vector = mod(key2*two_letters_vector, n);
encryped_vector = transpose(encryped_vector);
encrypted_string= append(encrypted_string,abc(encryped_vector(1)));
encrypted_string= append(encrypted_string,abc(encryped_vector(2)));

for i=2:size(string_vector_1, 2)
 two_letters_vector = string_vector_1(:,i);
 encryped_vector = mod(key2*two_letters_vector, n);
 for j=1:size(encryped_vector,1)
     encrypted_string= append(encrypted_string,abc(encryped_vector(j)));
 end
end
encrypted_string
```

```
encrypted_string =
```

```
'АЫУНЙУКОВТБЬ'
```

Вмешательство from hackerman

Чтобы убедиться, что матрицы расшифровываются правильно, можно закомментировать блок с вмешательством

```
% Выбранные индексы могут стать одинаковыми, это небольшая беда -__-
chosen_index = randi([1,12],1,3)
```

```
chosen_index = 1×3
    10    11     2
```

```
chosen_symbol = randi([1,34],1,3)
```

```
chosen_symbol = 1×3
    32    22     4
```

```
for i=1:3
    encrypted_string(chosen_index(i)) = abc(chosen_symbol(i));
end
encrypted_string
```

```
encrypted_string = 
'АГУНЙУКОВЮФЬ'
```

Расшифрование

```
inv_key2 = inverse_module_matrix(key2, n);
decrypted_string = '';
c = 1;
while c < m
    letter_code1 = strfind(abc, encrypted_string(c));
    letter_code2 = strfind(abc, encrypted_string(c+1));
    new_encrypted_vector = [letter_code1;letter_code2];
    decrypted_vector = mod((inv_key2*new_encrypted_vector),n);

    for j=1:size(decrypted_vector,1)
        if decrypted_vector(j) == 0
            % Отдельная проверка для нулей потому что 0 --> 34 = ' ' в моем
            % алфавите, но матлаба индексы начинаются с 1 ...
            decrypted_string= append(decrypted_string,abc(n));
        else
            decrypted_string= append(decrypted_string,abc(int16(decrypted_vector(j))));
        end
    end
    c = c + 2;
end
decrypted_string
```

```
decrypted_string = 
'ШАЯБЛОКОБЗЦЖ'
```

Работа с матрицей 3x3

В этом и последующем блоке код копируется, увеличивается лишь размерность матриц. Также можно заметить, что при больших размерностях от оригинального сообщения остается (почти) ничего после хакера

Шифрование

```
string_vector_2 = [33,2,12,19;
                   34, 13,16,16;
                   33,16,34,12];

three_letters_vector = string_vector_2(:,1);
encrypted_string = '';
encryped_vector = mod(key3*three_letters_vector, n);
encryped_vector = transpose(encryped_vector);
encrypted_string= append(encrypted_string,abc(encryped_vector(1)));
encrypted_string= append(encrypted_string,abc(encryped_vector(2)));
encrypted_string= append(encrypted_string,abc(encryped_vector(3)));

for i=2:size(string_vector_2, 2)
 three_letters_vector = string_vector_2(:,i);
 encryped_vector = mod(key3*three_letters_vector, n);
 for j=1:size(encryped_vector,1)
     encrypted_string= append(encrypted_string,abc(encryped_vector(j)));
 end
end

encrypted_string
```

```
encrypted_string =
'ГЬДВИЦГБМЦЙО'
```

Вмешательство from hackerman

```
chosen_index = randi([1,12],1,3)
```

```
chosen_index = 1×3
     4     7    12
```

```
chosen_symbol = randi([1,34],1,3)
```

```
chosen_symbol = 1×3
    33     6    34
```

```
encrypted_string
```

```
encrypted_string =
'ГЬДВИЦГБМЦЙО'
```

```
for i=1:3
    encrypted_string(chosen_index(i)) = abc(chosen_symbol(i));
end
encrypted_string
```

```
encrypted_string =
'ГЬДЯИЦЕБМЦЙ '
```

```

```

Расшифрование

```
inv_key3 = inverse_module_matrix(key3,n);
decrypted_string = '';
c = 1;
while c < m
    new_encrypted_vector = [strfind(abc, encrypted_string(c));
                            strfind(abc, encrypted_string(c+1));
                            strfind(abc, encrypted_string(c+2))];
    decrypted_vector = round(mod((inv_key3*new_encrypted_vector),n));

    for j=1:size(decrypted_vector,1)
        if decrypted_vector(j) == 0
            decrypted_string= append(decrypted_string,abc(n));
        else
            decrypted_string= append(decrypted_string,abc(int16(decrypted_vector(j))));
        end
    end
    c = c + 3;
end
decrypted_string
```

```
decrypted_string =
'Я ЯФПЬБМИУЕФ'
```

Работа с матрицей 4x4

Шифрование

```
string_vector_3 = [33 13 34;
                   34 16 19;
                   33 12 16;
                   2 16 12];

four_letters_vector = string_vector_3(:,1);
encrypted_string = '';
encryped_vector = mod(key4*four_letters_vector, n);
encryped_vector = transpose(encryped_vector)
```

```
encryped_vector = 1×4
    27    23    23     7
```

```
encrypted_string= append(encrypted_string,abc(encryped_vector(1)));
encrypted_string= append(encrypted_string,abc(encryped_vector(2)));
encrypted_string= append(encrypted_string,abc(encryped_vector(3)));
encrypted_string= append(encrypted_string,abc(encryped_vector(4)));
```

```
for i=2:size(string_vector_3, 2)
 four_letters_vector = string_vector_3(:,i);
 encryped_vector = mod(key4*four_letters_vector, n);
 for j=1:size(encryped_vector,1)
     encrypted_string= append(encrypted_string,abc(encryped_vector(j)));
 end
end

encrypted_string
```

```
encrypted_string = 
'ЩХХЁЬЬХФЁДВР'
```

Вмешательство from hackerman

```
chosen_index = randi([1,12],1,3)
```

```
chosen_index = 1×3
    12     6    10
```

```
chosen_symbol = randi([1,34],1,3)
```

```
chosen_symbol = 1×3
     5    15    32
```

```
encrypted_string
```

```
encrypted_string = 
'ЩХХЁЬЬХФЁДВР'
```

```
for i=1:3
    encrypted_string(chosen_index(i)) = abc(chosen_symbol(i));
end
encrypted_string
```

```
encrypted_string = 
'ЩХХЁЬНХФЁЮВД'
```

Расшифрование

```
inv_key4 = inverse_module_matrix(key4,n);
decrypted_string = '';
c = 1;
while c < m
    new_encrypted_vector = [strfind(abc, encrypted_string(c));
                           strfind(abc, encrypted_string(c+1));
                           strfind(abc, encrypted_string(c+2));
                           strfind(abc, encrypted_string(c+3))];
    decrypted_vector = round(mod((inv_key4*new_encrypted_vector),n));

    for j=1:size(decrypted_vector,1)
        if decrypted_vector(j) == 0
            decrypted_string= append(decrypted_string,abc(n));
        else
            decrypted_string= append(decrypted_string,abc(int16(decrypted_vector(j))));
```

```
        end
    end
    c = c + 4;
end
decrypted_string
```

```
decrypted_string = 
'Я ЯБЦЛАЯРЙОЯ'
```

```

```

Взлом шифра Хилла

У нас на руках два зашифрованных сообщения с помощью шифра Хилла с одним и тем же ключом, но в добавок у нас есть оригинал одного из сообщений.

Надо расшифровать второе сообщение

***

В процессе решения я столкнулся с тем, что взломать не брутфорсом я смог только алфавит длиной = простое число, в моем случае при n=34 попросту не существовало обратного элемента по мультипликативной группе, следовательно обратную матрицу по модулю невозможно найти. Тогдая я добавил пару букв до n=37, и чтобы наверняка, добавил условие - определитель ключа обязательно > 0

***

```
abc = ['А', 'Б', 'В', 'Г', 'Д', 'Е', 'Ё', 'Ж', 'З', 'И', 'Й', 'К', 'Л', 'М', 'Н', 'О', 'П', ...
    'Р', 'С', 'Т', 'У', 'Ф', 'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ъ', 'Ы', 'Ь', 'Э', 'Ю', 'Я', ' ', '!','
n = length(abc)
```

```
n = 37
```

```
msg1 = 'Я ЯБЛОКО СОК';
msg2 = 'ФЛЕКСИМ ДИКО';
m = length(msg1)
```

```
m = 12
```

Генерируем ключ 2x2

Некоторые ключи могут быть плохими - для взлома

Именно нахождения этого ключа позволит нам расшифровать второе сообщение, просто закройте глаза на код ниже

```
% key= randi([1 100],2,2);
% while not (isinteger(det(key)) && gcd(det(key),n) == 1 && det(key) > 0)
%     key= randi([1 100],2,2)
% end
% gcd(det(key),n)
```

```
key = [40 58;
```

```
        7 74];
% key = [45 98; 61 21]; - nope
% key = [39 8; 57 6];
```

Шифруем два сообщения

Берем вектора-столбцы по два символа-элемента, из которых уже собираем прямоугольную матрицу:

```
string_vector_1 = [33 33 13 12 34 16;
                   34 2 16 16 19 12];
string_vector_2 =[22 6 19 14 5 12;
                  13 12 10 34 10 16];
```

Получаем зашифрованные сообщения в виде строк, матрицы, в векторах которых по две буквы

```
[encrypted_string1, encrypted_vectors1] = encrypt2(abc, key,n,string_vector_1)
```

```
encrypted_string1 =
'?ЗЬЗДПБИТОГА'
encrypted_vectors1 = 2×6
    36    30     5     2    20     4
     9     9    17    10    16     1
```

```
[encrypted_string2, encrypted_vectors2] = encrypt2(abc, key,n,string_vector_2)
```

```
encrypted_string2 =
'ЕЕЙДЖФОЦВ!БИ'
encrypted_vectors2 = 2×6
     6    11     8    16     3     2
     6     5    22    24    35    10
```

Как взломать?

Хорошее решение - хитрость

Шифрование в общем виде: C = K*P , где нам очень хотелось бы получить матрицу-ключ K, тогда...

K = C*P^-1 , но P - вектор-стобец, **так нельзя делать**

Но ведь можно сделать равносильный переход, *добавив еще по два символа* (т.е. по вектору справа) в P, C, тем самым сделав их матрицами 2x2:

*[c1 c3; c2 c4] = [k1 k2; k3 k4] * [p1 p3; p2 p4]*

Теперь обратную матрицу от P найти вполне возможно

```
% Параллельно считаем "истинную" обратную матрицу, чтобы видеть как все круто
inv_K_true = inverse_module_matrix(key,n);
c = 1;
while c < m/2
    % Собираем квадратные матрицы
    % Обозначение переменных - C2 = C(2*2) и P2 = P(2*2)
    P2 = [encrypted_vectors1(:,c) encrypted_vectors1(:,c+1)];
    C2 = [string_vector_1(:,c) string_vector_1(:,c+1)];
    inv_P2 = inverse_module_matrix(P2,n);
```

```
    possible_K = mod(C2*inv_P2,n);
    % Сравниваем расшифровки
    if strcmp( decrypt2(abc, possible_K, n, encrypted_string1)  ,  msg1)
        disp('Key matrix is found');
        possible_K
        break;
    end
    c = c+2;
end
```

```
Key matrix is found
possible_K = 2×2
   37.0000   16.0000
   30.0000    3.0000
```

```
% Теперь расшифруем исходник
decrypt2(abc, possible_K, n, encrypted_string2)
```

```
ans =
'ФЛЕКСИМ ДИКО'
```

Брутфорс матрицы-ключа - плохое

Предположим, что ключ выбирала бабушка, которая особо большие числа не любит, тогда асимптотика вида **O(n^4) для нас не страшна...**

Код ниже сделан на рандомах, но ничего не мешает сделать 4 вложенных цикла...

```
% syms a b c d
% while true
%         a = randi([1, 1000]);
%         b = randi([1, 1000]);
%         c = randi([1, 1000]);
%         d = randi([1, 1000]);
%         A = [a b ; c d];
%         % функция check_key недописана, но совершенно очевидно, что
%         % она в себе будет содержать расшифровку сообщения 1 по ключу A
%         % и последующее сравнение с оригиналом 1
%         if (det(A) > 0 && gcd(int32(det(A)), n) == 1 && strcmp(decrypt2(abc, A, n),msg1))
%             break;
%         end
% end
```

Код Хэмминга (7, 4)

Код Хэмминга - это метод для обнаружения и исправления ошибок в передаче данных. Он добавляет дополнительные проверочные биты к данным, чтобы их можно было проверить на наличие ошибок.

(7,4) - значит блок длиной 7 бит и сообщение длиной 4 бита, 3 бита на кодировку всех возможных ошибок

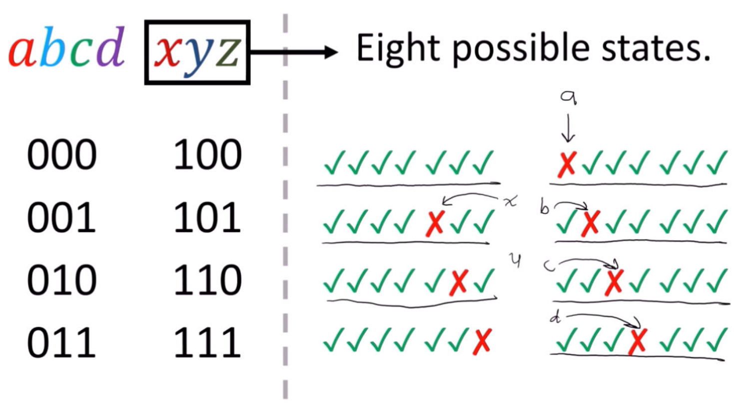
Значение каждого проверочного бита рассчитывается на основе позиции в блоке. Исходные данные и значения проверочных битов формируют код Хэмминга, который передается по каналу связи.

При получении данных, получатель вычисляет значения проверочных битов и сравнивает их с принятыми данными. Если значения не совпадают, то произошла ошибка. По значениям проверочных битов можно определить ошибочный бит и восстановить исходное сообщение. **Это перестает работать при двойных и более ошибках...**

Как это работает?

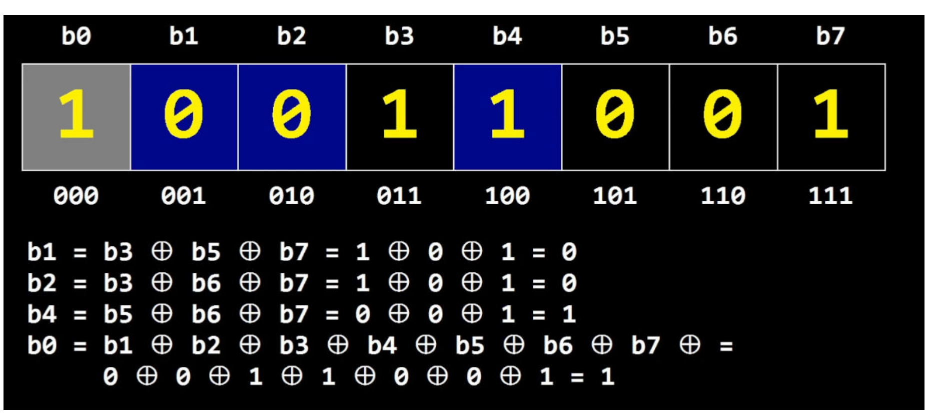
Добавляя дополнительных 3 бита(8 состояний) мы в них закладываем восемь примерно таких состояний...

- Отсутствуют ошибки?
- Есть ли ошибка в бите 1?
- Есть ли ошибка в бите 2?
- ...
- Есть ли ошибка в бите 7?

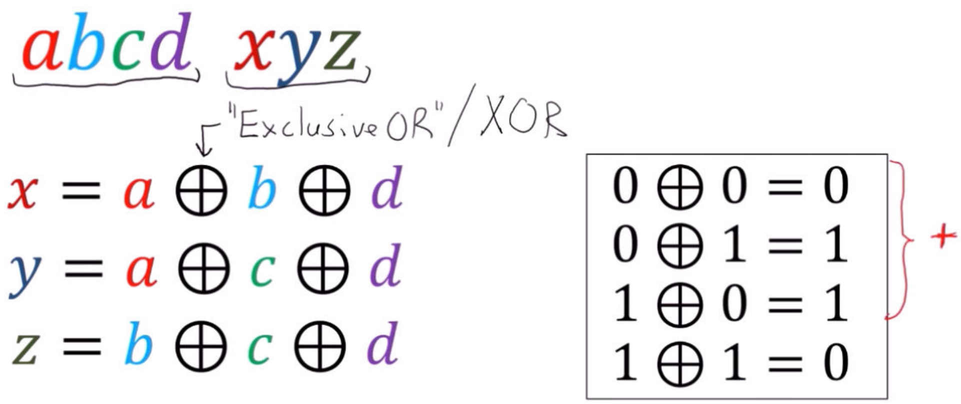


Почему линейный?

Потому что операции в численном варианте - это XOR, и их можно преобразовать в матричные операции



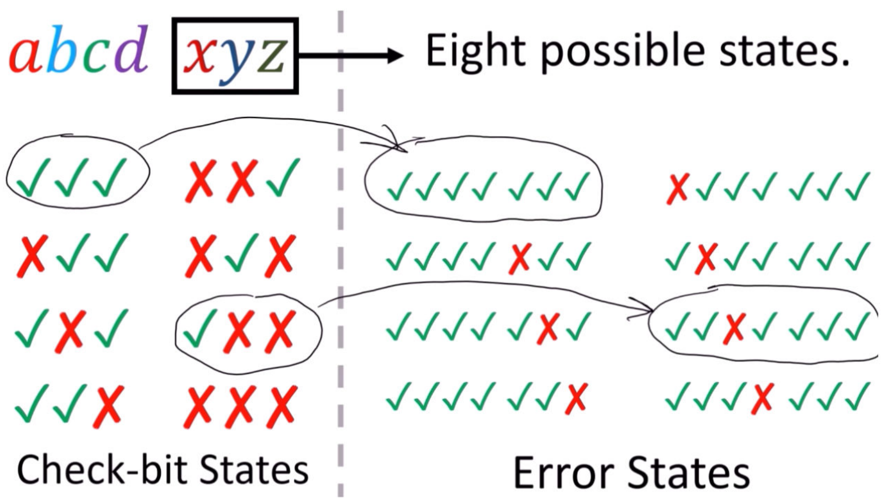
Про выбор порядка битов в коде



Пусть, что a,b,c,d - биты информации, x,y,z - биты проверки состояний. **Тогда [abcdxyz] - код Хэмминга в общем виде.**

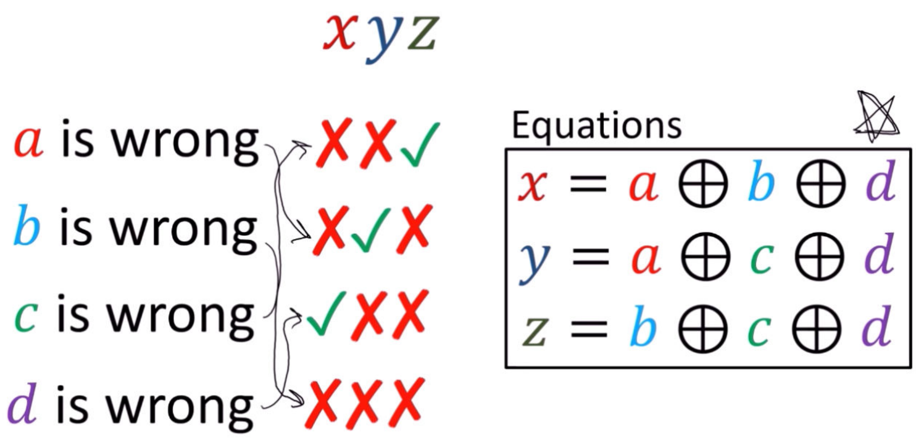
При смене порядка битов местами **сменятся лишь формулы** для битов состояний, операции XOR не сменятся на другие (свойства операции XOR)...

Начнем с того, что все проверочные состояния выбираются произвольно нами, допустим 100 будет отвечать за ошибку в третьем бите кода Хэмминга и.т.д, продолжим устанавливать соответствия



На картинке выше мы сделали биекцию между всеми возможными комбинациями битов состояний и допустимыми одиночными ошибками где-то в коде.

Давайте рассмотрим ошибки в битах ценной информации. Как мы можем отразить в комбинации, что ошибочен какой-то ценный бит? Вариант выбора не единственнен, глянем картинку с примером.

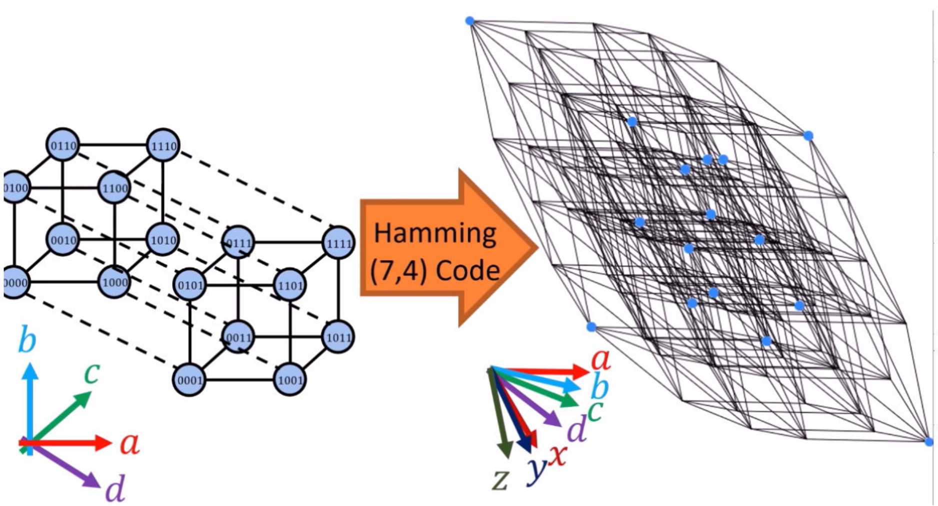


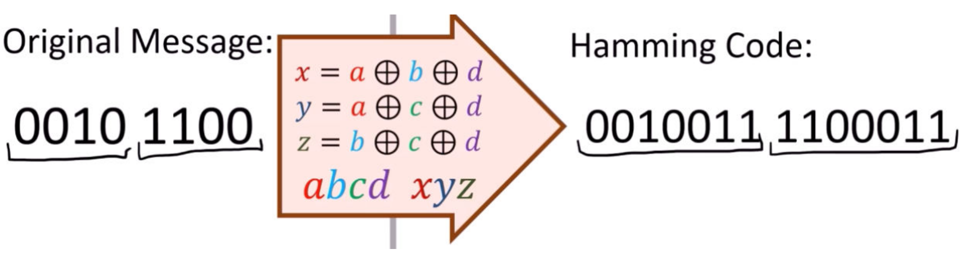
Но такие стрелочки мы можем расставить как захотим, и тогда, если посмотреть по столбцам, уравнения битов состояний просто сменит свой вид, а это правило мы лишь запишем себе на листок, чтобы потом кодировать и проверять правильно.

В итоге допустим будет даже такой вариант кода - [axbyczd] - и конечно это повлияет на вид матриц G, H

Как можно составить матрицу G?

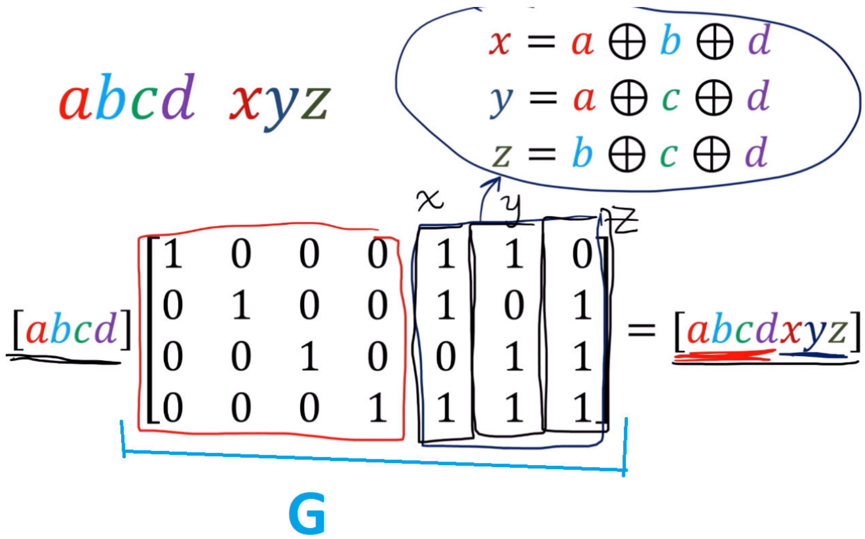
По сути "code generator matrix" или "образующая матрица G" это линейное преобразование: 4D --> 7D пространство, которое изобразить довольно трудно. Матрица этого преобразования позволяет нам из наших данных получить код Хэмминга





Поэтому обычно для кода Хэмминга используют численные методы или матричное представление. Рассмотрим второй вариант, так как мы в матлабе сейчас :)

На картинке ниже мы видим пример матрицы G на основе формул битов состояний, которые появились из-за выбора биекций



В этом случае происходит не простое матричное умножение, а еще и "взятие по модулю 2" или же XOR - кому как удобно.

Так как в итоговом коде первые 4 позиции занимает ценная информация, тогда первые 4 строки занимает матрица эквивалентного преобразования (= единичная матрица). Потом участие принимают формулы состояния, которые занимают три оставшихся места.

<u>Если мы захотим поменять в коде Хэмминга биты местами, нам следует поменять соответствующие им векторы-столбцы местами тоже</u>

В моем случае я взял *канонический базис* и сразу транспонировал матрицу для удобства перемножения

```
G = [1 1 0 1;
     1 0 1 1;
     1 0 0 0;
     0 1 1 1;
     0 1 0 0;
     0 0 1 0;
     0 0 0 1];
```

Как можно составить матрицу H?

"parity-check matrix" или "матрица управления H" нам понадобится уже в случае ошибок, ибо из кода Хэмминга без ошибок легко считать исходные данные.

Чтобы подружиться с примером ниже давайте на веру примем, что какая-то матрица H существует для любого G. Вспомним, что H*C = 0, где C - код Хэмминга без ошибок. Тогда добавим в C ошибку, повторим то же самое, но по свойству линейности разобьем новый код с ошибкой на две компоненты

= два вектора. Тогда один по свойству выше станет нулевым, **а второй станет проекцией столбца с матрицы H (вектор-синдром)**.

$$H\vec{c}^T = \vec{0} \Leftarrow$$

$$H(\vec{c}+\vec{e})^T = \cancel{H\vec{c}^T}^{\,0} + H\vec{e}^T = \underline{\underline{H\vec{e}^T}}$$

$$\begin{bmatrix} 1 & \boxed{1} & 0 & 1 & 1 & 0 & 0 \\ 1 & \boxed{0} & 1 & 1 & 0 & 1 & 0 \\ 0 & \boxed{1} & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \right) = \left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \boxed{\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}} \right)$$

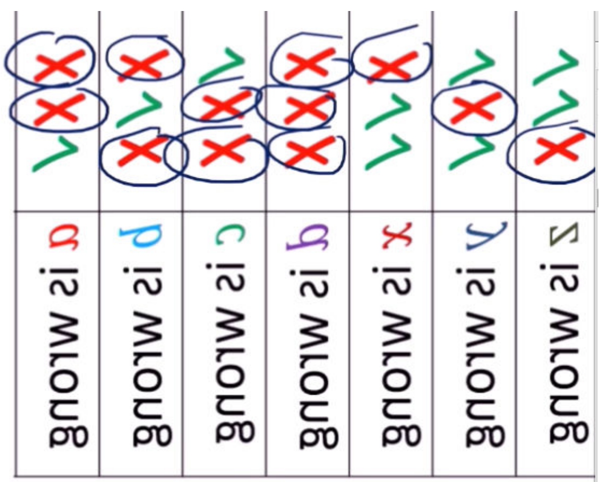
Нас это наводит на мысль, что столбцы матрицы H должны явно нам подсказывать в каком месте у вектора ошибки была единичка, т.е. какой бит ошибочный...

Кстати, если ошибок будет несколько,естественно они сложатся в единую, тогда подсказку мы не сможем получить **однозначно**

Так как нам выбрать подсказки?

Вспомним биекции, которые мы задали на этапе проектировки матрицы G, например такие:

✓✓✓	No errors
✗✗✓	a is wrong
✗✓✗	b is wrong
✓✗✗	c is wrong
✗✗✗	d is wrong
✗✓✓	x is wrong
✓✗✓	y is wrong
✓✓✗	z is wrong



$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$
\begin{array}{ccccccc}
abcdxyz & abcdxyz & abcdxyz & abcdxyz & abcdxyz & abcdxyz & abcdxyz \\
\begin{bmatrix}1\\1\\0\end{bmatrix} & \begin{bmatrix}1\\0\\1\end{bmatrix} & \begin{bmatrix}0\\1\\1\end{bmatrix} & \begin{bmatrix}1\\1\\1\end{bmatrix} & \begin{bmatrix}1\\0\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\end{bmatrix} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow
\end{array}
$$

$$
\begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1
\end{bmatrix}
$$

Магическое переворачивание таблицы делает всю работу за нас - нам лишь остается убедиться в коде ниже или на дополнительной картинке, что это все работает.

<u>Так как такой поворот уникален, то выбор матрицы H - единственный и зависит от G</u>

```
H = [0 0 0 1 1 1 1;
     0 1 1 0 0 1 1;
     1 0 1 0 1 0 1];
```

Как соотносятся образ одной матрицы с ядром другой матрицы?

переформулируем: Image(G) = Kernel(H) ?

*Они **равны** в случае, если в код Хэмминга не внесли ошибку*

Ядро H = все вектора V, для которых верно {H*v = 0} , с оговоркой выше под это попадают все вектора кода Хэмминга

$$
\mathbf{H.C} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 0\\1\\1\\0\\0\\1\\1 \end{pmatrix} = \begin{pmatrix} 0\\0\\0 \end{pmatrix}
$$

Образ G - это только что переведенные 4 бита в 7 бит кода, тоже по сути все вектора...

$$
\mathbf{D} = \begin{pmatrix} 1\\0\\1\\1 \end{pmatrix} \quad \mathbf{G.D} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1\\0\\1\\1 \end{pmatrix} = \begin{pmatrix} 0\\1\\1\\0\\0\\1\\1 \end{pmatrix} \quad et \quad \mathbf{C} = \mathbf{G.D} = \begin{pmatrix} 0\\1\\1\\0\\0\\1\\1 \end{pmatrix}
$$

Перевод слова в код с помощью матрицы Хэмминга

```
% from 00000 to 11111 - 32 letters in abc
abc = ['А', 'Б', 'В', 'Г', 'Д', 'Е', 'Ё', 'Ж', 'З', 'И', 'Й', 'К', 'Л', 'М', 'Н', 'О', 'П', ...
    'Р', 'С', 'Т', 'У', 'Ф', 'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ъ', 'Ы', 'Ь', 'Э', 'Ю'];
word = 'РАКИ';
bin_word = word_to_bin(abc, word)
```

```
10001
0
1011
1001
bin_word =
'10001000000101101001'
```

```
m = length(word)
```

```
m = 4
```

```

% cell2mat превращает cell array в нормальный тип данных - матрица
divided_word =strread(bin_word,'%4s');

% внутри cell индексы {} , внутри массива/матрицы ()
% word{1}(1)

hamming_msg_vectors = zeros([7,5]);

for i=1:5
    msg_vector = str2num(transpose(divided_word{i}));
    hamming_msg_vectors(:,i) = mod(G*msg_vector,2);
end
```

Вредоносное вмешательство from hackerman

Термин "умножения" в случае в моем случае не совсем корректно применять, потому что сразу после него мы ищем остаток по модулю 2 от матрицы <---> **" * " and XOR**

Не меняем ничего, расшифровываем

```
word0 = hamming_msg_vectors;

for j=1:5
    H_result = mod(H*word0(:,j),2)
end
```

```
H_result = 3×1
     0
     0
     0
```

```
H_result = 3×1
     0
     0
     0
H_result = 3×1
     0
     0
     0
H_result = 3×1
     0
     0
     0
H_result = 3×1
     0
     0
     0
```

```
dycrypted_string = hamming_matrix_to_word(word0, abc)
```

```
dycrypted_string =
'РАКИ'
```

Все перемножения = нулевые вектора, то ошибок не было , значит все буквы будут исходными, проверяем...

Так как мы меняем случайный бит, то это не всегда может быть бит информации и поэтому не всегда исходное сообщение нарушится

Меняем 1 бит

```
word1 = hamming_msg_vectors;
idx1 = randi([1 7]);
idx2 = randi([1 5]);
word1(idx1, idx2) = not(word1(idx1, idx2));

% Декодируем
hamming_matrix_to_word(word1, abc)
```

```
ans =
'РАКК'
```

Восстанавливаем исходное сообщение

```
%Пробегаемся по каждой букве, проверяя условие целостности передачи
% Значение вектора result - называется синдромом
problem_letter_index = -999;
problem_letter_bit_index = -999;
health = '0000000';
for j=1:5
    result = mod(H*word1(:,j),2);
    if not(isequal(mod(H*word1(:,j),2),zeros([3,1])))
        result
        % Ищем индексы виновника и инвертируем его
        problem_letter_index = j
        problem_letter_bit_index = bin2dec(transpose(int2str(result)))
```

```
        word1(problem_letter_bit_index,problem_letter_index) =  ...
        not(word1(problem_letter_bit_index,problem_letter_index));
    end
end
```

```
result = 3×1
    1
    1
    0
problem_letter_index = 5
problem_letter_bit_index = 6
```

```
% Декодируем снова... SUCCESS
hamming_matrix_to_word(word1, abc)
```

```
ans =
'РАКИ'
```

В остальных случаях, матрица H нам ничем не поможет, поэтому просто давайте полюбуемся на итог

Меняем 2 бита

```
word2 = hamming_msg_vectors;
for i=1:2
    idx1 = randi([1 7]);
    idx2 = randi([1 5]);
    word2(idx1, idx2) = not(word2(idx1, idx2));
end
% Декодируем
hamming_matrix_to_word(word2, abc)
```

```
ans =
'ШАКК'
```

Меняем 3 бита

```
word3 = hamming_msg_vectors;
for i=1:3
    idx1 = randi([1 7]);
    idx2 = randi([1 5]);
    word3(idx1, idx2) = not(word3(idx1, idx2));
end
% Декодируем
hamming_matrix_to_word(word3, abc)
```

```
ans =
'ШЗИИ'
```

Меняем 4 бита

```
word4 = hamming_msg_vectors;
for i=1:4
    idx1 = randi([1 7]);
    idx2 = randi([1 5]);
    word4(idx1, idx2) = not(word4(idx1, idx2));
```

```
end
% Декодируем
hamming_matrix_to_word(word4, abc)
```

```
ans =
'ШПКИ'
```

Загадка про заключенных на код Хэмминга

Что происходит ?

1. Охранник выкладывает монеты на доску
2. Он указывает на ключ только первому
3. Первый заключенный переворачивает одну монету
4. Доску передают второму заключенному

Так как доска 8*8, то для кодировки координат ключа достаточно будет (3+1)+(3+1) = 6 бит. 3 бита на само число от 1 до 8 и один логический бит: 0 --> X and 1 --> Y - для координаты

Договоримся, что монетка "T" --> 1 and "H" --> 0

Используя код Хэмминга, мы должны сделать так, чтобы наш друг смог найти **только** два подтвержденных кода.

Получается наш друг должен найти 7+7 бит информации на доске, которые он однозначно декодирует. Значит нам надо с ним договориться где он будет их искать...

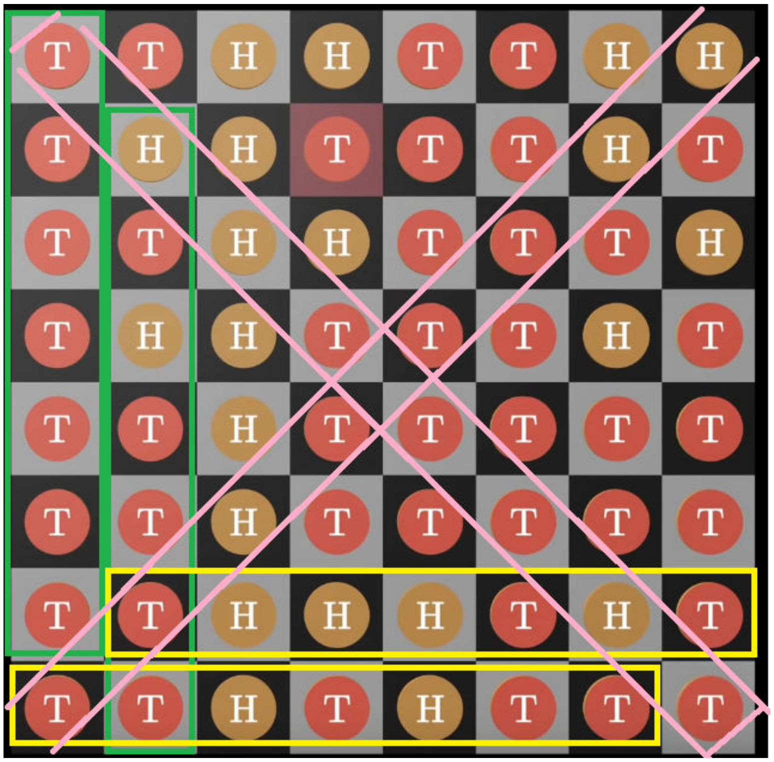
На картинке ниже я попытался посчитать все возможные коды, которые можно в ряд уместить на доске. Так как наш код длиной 7, а доска 8, то можно еще смещение на +- 1 вперед добавить.

По вертикали+горизонтали: (2*2*8) + (2*2*8)

По диагонали: 2+2

В добавок каждый код можно читать слева-направо или наоборот :) extra " *2 "

В итоге 136 комбинаций тривиальных



В теории нам нужно 2 из 2^7 комбинаций 7 бит всего, однако есть два преимущества..

Переворотом монетки мы уменьшим количество нужных комбинаций вдвое -> повышаем шансы

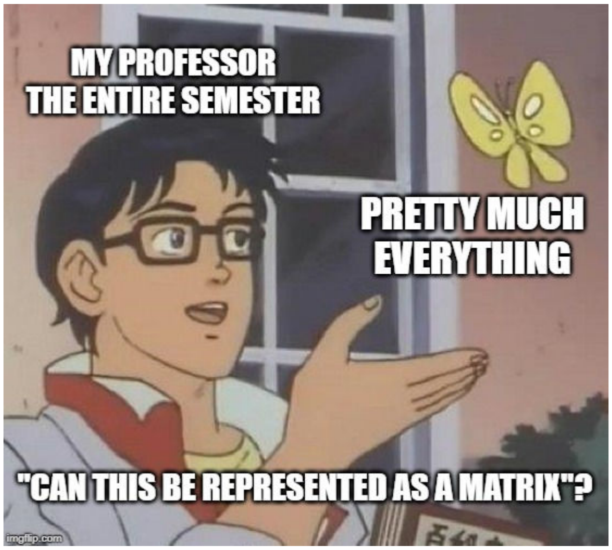
Кодом хэмминга мы **возможно** уменьшим количество нужных комбинаций вдвое -> повышаем шансы

Начало теории вероятности ни к чему хорошему не приведет, ибо оно не учитывает того, что код хэмминга может оказаться с куда большим числом ошибок, а также, что охранник так случайно разложить монетки и положить ключ, что на доске воспроизвести 14 бит, которые поймет наш друг - будет невозможным

В случае когда доска состоит полностью из монеток T или H, ответ очевиден: мы выбираем монетку с координатами ключа, ведь доска не переворачивается

Конец решения

Спасибо, если прочитал до этого момента или отвлекся на мем!



Матричные преобразования действительно крутые

Вспомогательные функции и их описание

Первое-Второе задание

Обратная матрица по модулю

```
function result = inverse_module_matrix(matrix, modulus)
    base = det(matrix);
    inv_element = 1;

    if gcd(base, modulus) ~= 1
        disp("Solution not found.")
    else
        for i = 1: modulus
            if mod(base * i, modulus) == 1
                inv_element = base*i;
                break;
            end
        end
    end
    result = mod((inv_element*(matrix^-1)),modulus);
end
```

Шифрование ключем 2x2

```
function [encrypted_string, number_vector] = encrypt2(abc, key, n,string_vector)
    encrypted_string = '';
    number_vector = zeros(2, 12);
    vector2 = string_vector(:,1);
    encrypted_number_vector = mod(key*vector2, n);
    number_vector = encrypted_number_vector;
    encrypted_number_vector = transpose(encrypted_number_vector);
    encrypted_string= append(encrypted_string,abc(encrypted_number_vector(1)));
```

```
    encrypted_string= append(encrypted_string,abc(encrypted_number_vector(2)));
    for i=2:6
         vector2 = string_vector(:,i);
         encrypted_number_vector = mod(key*vector2, n);
         number_vector(:,i) = encrypted_number_vector;

         for j=1:size(encrypted_number_vector,1)
             if encrypted_number_vector(j) == 0
                 encrypted_string= append(encrypted_string,abc(n));
            else
                encrypted_string= append(encrypted_string,abc(int16(encrypted_number_vector(j))
            end

         end
    end

end
```

Расшифровка ключем 2x2

```
function decrypted_string = decrypt2(abc, key, n, encrypted_string)
    decrypted_string = '';
    inverse_key = key;
    c = 1;
    while c < 12
        letter_code1 = strfind(abc, encrypted_string(c));
        letter_code2 = strfind(abc, encrypted_string(c+1));
        new_encrypted_vector = [letter_code1;letter_code2];
        decrypted_vector = mod((inverse_key*new_encrypted_vector),n);

        for j=1:size(decrypted_vector,1)
            if decrypted_vector(j) == 0
                decrypted_string= append(decrypted_string,abc(n));
            else
                decrypted_string= append(decrypted_string,abc(int16(decrypted_vector(j))));
            end
        end
        c = c + 2;
    end

end
```

Третье задание

Перевод слова в двоичное представление

```
function result = word_to_bin(abc,word)
    result = '';
    for i=1:length(word)
         index = strfind(abc, word(i))-1;
```

```
        disp(dec2bin(index))
        result= append(result,dec2bin(index,5));
    end

end
```

Перевод матрицы векторов(вектор=1 буква) в слово

```
function word = hamming_matrix_to_word(M,abc)
    word = '';
    bytes = '';
    % собираем с кода Хэмминга биты информации в строку
    for i=1:5
        bytes = append(bytes, strcat(int2str(M(3,i)), int2str(M(5,i)), int2str(M(6,i)), int2str
    end
    divided_bytes = strread(bytes,'%5s');
    % Теперь эту строку делим по 5 байт, чтобы перевести в символ и делаем
    % из этого слово
    for j=1:4
        word = append(word, abc(bin2dec(divided_bytes{j})+1));
    end

end
```

Источники

Убрать кашу из головы по кодам Хэмминга мне больше всего помогли эти источники

1. Вики на русском - понял основы, но не хватило обощения теории
2. Крутой чувак на английском - дал обобщение с примерами: 1,2,4 видео из цикла