Алгоритм Хаффмана

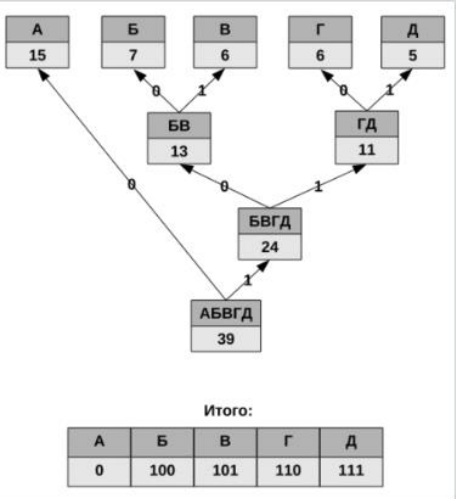
Алгоритм Хаффмана — жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью.

Этот метод кодирования состоит из двух основных этапов:

1. Построение оптимального кодового дерева.
2. Построение отображения код-символ на основе построенного дерева.

Классический алгоритм Хаффмана на входе получает таблицу частотностей символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево)

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.



Алгоритм Шеннона — Фано

Основные этапы:

1. Символы первичного алфавита m1 выписывают по убыванию вероятностей.
2. Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
3. В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
4. Полученные части рекурсивно делятся, и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Когда размер подалфавита становится равен нулю или единице, то дальнейшего удлинения префиксного кода для соответствующих ему символов первичного алфавита не происходит, таким образом, алгоритм присваивает различным символам префиксные коды разной длины. На шаге деления алфавита существует неоднозначность, так как разность суммарных вероятностей p{0}-p{1} может быть одинакова для двух вариантов разделения (учитывая, что все символы первичного алфавита имеют вероятность больше нуля).

a_i	$p(a_i)$	1	2	3	4	Итого
a_1	0.36	0	00			00
a_2	0.18		01			01
a_3	0.18	1	10			10
a_4	0.12		11	110		110
a_5	0.09			111	1110	1110
a_6	0.07				1111	1111

Алгоритм Шеннона

Первым шагом будет подсчёт вероятностей каждого символа. Затем считается число

l для каждой вероятности. Например, для a2 оно равно трём ({2^{-3} <= 0,18 <= 2^{-2}} — минимальная степень двойки −3, следовательно l равно трём). После этого считается сумма вероятностей от 0 до i-1 и переводится в двоичную форму. Потом дробная часть усекается слева до l{i} числа знаков.

a_i	$p(a_i)$	l_i	Сумма 0 до i-1	Сумма по $p(a_i)$ bin	Итоговый код
a_1	0.36	2	0.0	0.0000	00
a_2	0.18	3	0.36	0.0101	010
a_3	0.18	3	0.54	0.1000	100
a_4	0.12	4	0.72	0.1011	1011
a_5	0.09	4	0.84	0.1101	1101
a_6	0.07	4	0.93	0.1110	1110

Хеш-таблица

Хеш-табли́ца — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию удаления и операцию поиска пары по ключу.

Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение i=hash(key) играет роль индекса в массиве H. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива H[i].

Разрешение коллизий

Существует несколько способов разрешения коллизий:

- Метод списков

Каждая ячейка массива H является указателем на связный список пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются списки длиной более одного элемента.

- Открытая адресация

В массиве H хранятся сами пары ключ-значение. Алгоритм вставки элемента проверяет ячейки массива H в некотором порядке до тех пор, пока не будет найдена первая свободная ячейка, в которую и будет записан новый элемент. Этот порядок вычисляется на лету, что позволяет сэкономить на памяти для указателей, требующихся в хеш-таблицах с цепочками.

Последовательность, в которой просматриваются ячейки хеш-таблицы, называется **последовательностью проб**.

Ниже приведены некоторые распространенные типы последовательностей проб.

- Линейное пробирование: ячейки хеш-таблицы последовательно просматриваются с некоторым фиксированным интервалом k
- Квадратичное пробирование: интервал между ячейками с каждым шагом увеличивается на константу.
- Двойное хеширование: интервал между ячейками фиксирован, как при линейном пробировании, но, в отличие от него, размер интервала вычисляется второй, вспомогательной хеш-функцией, а значит, может быть различным для разных ключей.

Инверсия

Для заданных чисел c и m(c и m взаимно простые) число d ($0 < d < m$) называется инверсией числа c по модулю m, если выполняется условие $cd \ mod \ m = 1$ Для обозначения числа d, которое является инверсией числа c по модулю m используется обозначение $d = c^{-1} mod \ m$

Для нахождения инверсии можно применять обобщенный алгоритм Евклида. Приведем далее объяснение взаимосвязи задачи нахождения инверсии и задачи, описанной в назначении обобщенного алгоритма Евклида. Из определения инверсии следует, что $cd \ mod \ m = 1$. Следовательно, $cd \ = km + 1$, где k - некоторое целое число. С учетом того, что c и m взаимно простые запишем:

$$(-k)m + dc = \gcd(m, c)$$

Очевидно, что для данного равенства обобщенный алгоритм Евклида позволяет вычислить $-k$ и d, если на вход будут поданы m и c. Входные вектора принимают вид

$$U = (m, 1, 0)$$

$$V = (c, 0, 1)$$

Обратим внимание, что в контексте задачи вычисления инверсии вторые элементы обоих векторов могут быть исключены, так участвуют только в определении значения $-k$. Таким образом, входными векторами будут

$$U = (m, 0)$$

$$V = (c, 1)$$

Так как инверсия числа должны быть больше нуля, то если в результате вычисления получится значение $d < 0$, необходимо прибавить к d значение m (по свойству операции $mod \ \ d \ mod \ m = d + m \ mod \ m$).

Пример.

Вычислить значение $d = 7^{-1} mod \ 11$. Определим входные вектора:

$$U = (11, 0)$$

$$V = (7, 1)$$

Результаты выполнения цикла WHILE алгоритма представлены в таблице ниже.

№	q	T	U	V
1	11 div 7=1	(11 mod 7, 0-1*1)=(4,-1)	(7,1)	(**4**,-1)
2	7 div 4=1	(7 mod 4, 1-1*(-1))=(3,2)	(4,-1)	(**3**,2)
3	4 div 3=1	(4 mod 3, -1-1*2)=(1,-3)	(3,2)	(**1**,-3)
4	3 div 1=3	(3 mod 1, 2-3*(-3))=(0,11)	(1,-3)	**0**,11)

Таким образом, $U = (\gcd(11,7), d) = (1, -3)$. Так как $d = -3$, для получения инверсии необходимо прибавить к результату m, т.е. $d = -3 + 11 = 8$. Проверяем выполнение условия из определения инверсии:

Алгоритм быстрого возведения в степень

```
def exponentiation_modulo(a: int, x: int, p: int) -> int:
    if p == 0:
        raise ValueError("Модуль не может быть равен нулю")
    if x < 0:
        raise ValueError("Показатель не может быть отрицательным")
    result = 1
    a = a % p
    if a == 0:
        return 0
    while x > 0:
        if x & 1 == 1:
            result = (result * a) % p
        a = (a ** 2) % p
        x >>= 1
    return result
```

Перейдем теперь к описанию алгоритма быстрого возведения числа в степень по модулю. Как хорошо известно, в информатике любое десятичное число может быть представлено в виде полинома степеней двойки.

$x = x_0 2^0 + x_1 2^1 + \cdots + x_t 2^t,$

где $x_t \ldots x_0$– двоичное представление числа x, а t – нижняя граница $\log_a x$.

Тогда, по свойству показательной функции

$a^x \bmod p = \prod_{i=0}^{t} a^{x_i 2^t} \ mod\ p \qquad (1)$

В общем виде схема вычислений может быть описана следующим образом. Сначала вычисляются значения ряда

$a^1, a^2, a^4, \ldots, a^{2^t}\ (mod\ p),\ \ (2)$

где каждое новое значение ряда, начиная со второго, получается путем умножения текущего члена ряда на себя. Далее, с учетом значений битов двоичного представления числа x вычисляем итоговый результат по формуле (1).

Пример.

Определить значение $5^{20}\ mod\ 7$. Сначала вычисляем значения ряда (2):

$$5^1 \quad 5^2 \quad 5^4 \quad 5^8 \quad 5^{16}\ (mod\ 7)$$

$$5 \quad 4 \quad 2 \quad 4 \quad 2$$

$5^2\ mod\ 7 = [(5^1\ mod\ 7) * (5^1\ mod\ 7)\ ] mod\ 7$. Так как $(5^1\ mod\ 7)$ было вычислено на предыдущем шаге, то $5^2\ mod\ 7 = 5 * 5\ mod\ 7 = 4$.

$5^4\ mod\ 7 = [(5^2\ mod\ 7) * (5^2\ mod\ 7)\ ] mod\ 7$. Так как $(5^2\ mod\ 7)$ было вычислено на предыдущем шаге, то $5^4\ mod\ 7 = 4 * 4\ mod\ 7 = 2$.

$5^8\ mod\ 7 = [(5^4\ mod\ 7) * (5^4\ mod\ 7)\ ] mod\ 7$. Так как $(5^4\ mod\ 7)$ было вычислено на предыдущем шаге, то $5^8\ mod\ 7 = 2 * 2\ mod\ 7 = 4$.

$5^{16}\ mod\ 7 = [(5^8\ mod\ 7) * (5^8\ mod\ 7)\ ] mod\ 7$. Так как $(5^8\ mod\ 7)$ было вычислено на предыдущем шаге, то $5^{16}\ mod\ 7 = 4 * 4\ mod\ 7 = 2$.

Записываем показатель степени в двоичной форме:

$$20 = 10100$$

И вычисляем итоговое значение по формуле (1):

$$y = 2 * 1 * 2 * 1 * 1\ mod\ 7 = 4$$

i.e. $5^{117} \bmod 19$

117=1110101 in binary

$117=(2^0 + 2^2 + 2^4 + 2^5 + 2^6)$
$117= 1 + 4 + 16 + 32 + 64$

$5^{117} \bmod 19 = 5^{(1 + 4 + 16 + 32 + 64)} \bmod 19$

$5^{117} \bmod 19 = (5^1 * 5^4 * 5^{16} * 5^{32} * 5^{64}) \bmod 19$

5^117 mod 19 = (5^1 * 5^4 * 5^16 * 5^32 * 5^64) mod 19

5^117 mod 19 = (5^1 mod 19 * 5^4 mod 19 * 5^16 mod 19 * 5^32 mod 19 * 5^64 mod 19) mod 19

5^117 mod 19 = (5 * 17 * 16 * 9 * 5) mod 19

5^117 mod 19 = 61200 mod 19 = 1

5^117 mod 19 = 1

Шифр Шамира

Шифр Шамира представляет собой средство обеспечения конфиденциального обмена сообщениями между абонентами.

Первые действия абонентов направлены на инициализацию параметров шифра. Алиса генерирует большое просто число p и передает его Бобу по открытому каналу. Далее, Алиса формирует пару чисел C_A и D_A, так чтобы выполнялось равенство

$$C_A * D_A \; mod \; (p-1) = 1$$

Сформированная пара чисел держится Алисой в секрете.

Аналогично, Боб генерирует и держит в секрете пару чисел C_B и D_B, таких, что

$$C_B * D_B \; mod \; (p-1) = 1$$

На этом формирование параметров шифра закончено, и Алиса переходит к шифрованию сообщения m, представляющее собой некоторое число. Для использования данного шифра должно выполняться условие $m < p$. Процесс формирования криптограммы и ее последующей дешифровки включает несколько шагов.

Шаг 1. Алиса вычисляет значение

$$x_1 = m^{C_A} \; mod \; p$$

и отправляет его Бобу.

Шаг 2. Боб принимает отправленное ему число, вычисляет на его основе

$$x_2 = {x_1}^{C_B} \; mod \; p$$

и передает его Алисе.

Шаг 3. Алиса вычисляет

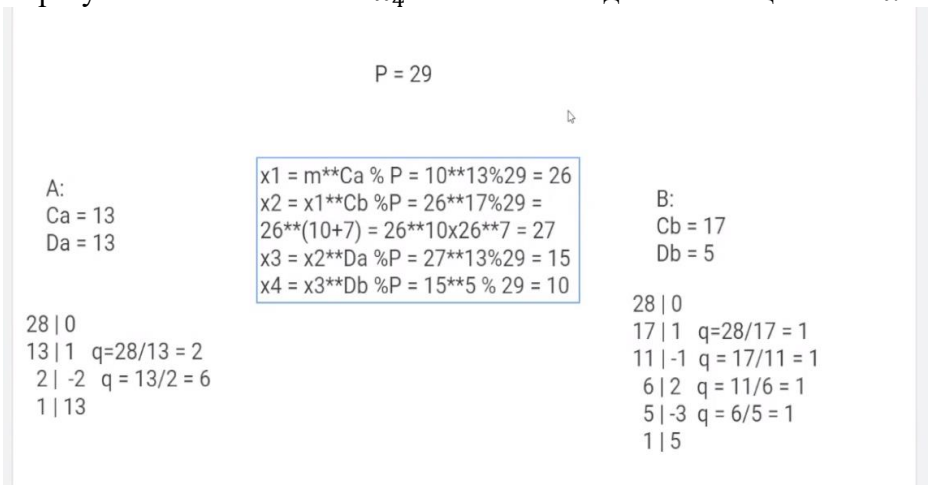
$$x_3 = {x_2}^{D_A} \; mod \; p$$

и отправляет его Бобу.

Шаг 4. Боб получает указанное число и вычисляет

$$x_4 = {x_3}^{D_B} \; mod \; p$$

Полученное в результате Бобом число x_4 является исходным сообщением m.



Шифр Эль-Гамаля

Шифрование Эль-Гамаля фактически базируется на системе Диффи-Хеллмана, позволяющей сформировать общий секретный ключ у двух абонентов. Построение криптограммы представляет собой умножение исходного сообщения на указанный ключ, а дешифрование происходит путем умножения криптограммы на обратное для ключа число.

Первоначально производится формирование общего большого простого числа p и числа $g, 1 < g < p - 1$, такого, что числа $g^1 mod\ p$, $g^2 mod\ p, \ldots, g^{p-1} mod\ p$ попарно различны между собой и образуют множество $\{1,2, \ldots, p - 1\}$. Более подробное описание способов выбора чисел p и g может быть найдено в разделе, посвященном системе Диффи-Хеллмана. Далее Боб генерирует целое число $x, 1 < x < p$ и производит вычисление $y = g^x\ mod\ p$. Число x хранится Бобом в секрете, а число y является открытым. Как и в шифре Шамира, преполагается, что необходимо передать сообщение $m < p$.

Шаг 1. Алиса формирует секретный сессионный ключ $k, 1 < k < p - 1$

Шаг 2. Алиса вычисляет числа

$$a = g^k mod\ p$$
$$b = m * y^k mod\ p$$

и передает пару (a, b) Бобу.

Шаг 3. Боб, получает (a, b) и вычисляет

$$m' = b * a^{p-1-x}\ mod\ p$$

Значение $m' = m$, таким образом Боб получил исходное сообщение.

Шифр RSA

Шифр RSA базируется на так называемой односторонней функции с «лазейкой» (trapdoor function)

$$y = x^d \ mod \ p$$

Система шифрования RSA базируется на двух фактах из теории чисел:

- Низкая трудоемкость решения задачи проверки числа на простоту
- Высокая трудоемкость решения задачи разложения чисел вида $n = pq$ (p и q – большие простые) на множители (задача факторизации)

Приведем далее описание процесса передачи шифрованного сообщения от абонента A (Алисы) абоненту B (Бобу) с помощью шифра RSA.

Вначале Боб инициализирует необходимые для получения сообщений от других абонентов параметры. Он генерирует большие простые числа P и Q, и вычисляет

$$N = PQ \quad (1)$$

Далее Боб вычисляет $\phi = (P-1)(Q-1)$, выбирает $d < \phi$, такое что $\gcd(d, \phi) = 1$, и по обобщенному алгоритму Евклида вычисляет c, такое что

$$cd \ mod \ \phi = 1 \quad (2)$$

На этом этап формирования параметров шифра считается законченным. Числа N и d являются открытыми ключами Боба, а число c – его закрытым ключом. Отметим здесь, что числа P, Q и ϕ хранятся Бобом в секрете и в дальнейшем не участвуют в процессе шифрования и дешифрования сообщений. Пусть Алиса планирует передать Бобу сообщение $m < N$. Рассмотрим действия абонентов.

Шаг 1. Алиса, используя открытые ключи Боба, формирует криптограмму

$$e = m^d \ mod \ N$$

и отправляет ее Бобу.

Шаг 2. Боб принимает криптограмму и производит ее дешифровку:

$$m' = e^c \ mod \ N$$

Таким образом, вычислив m', Боб получил исходное сообщение m. Докажем это утверждение, для этого рассмотрев вычисления, проводимые Бобом.

$$m' = e^c \ mod = m^{cd} \ mod \ N$$

На основании равенства (2) имеем $cd = k\phi + 1$, где k - некоторое целое число. Согласно Утверждению 2 раздела «Элементы теории чисел»

$$\varphi(N) = \varphi(PQ) = (P-1)(Q-1) = \phi$$

Следовательно,

$$m^{cd} \ mod \ N = m^{k\phi+1} \ mod \ N = m^{k\varphi(N)+1} \ mod \ N = m$$

Подпись RSA

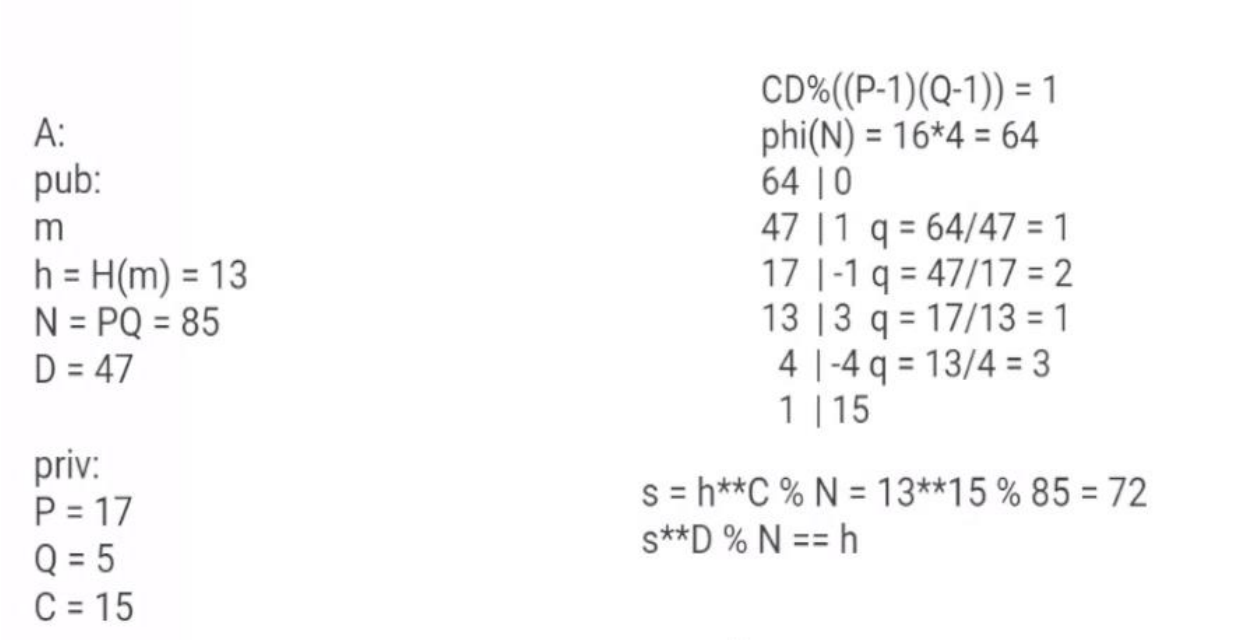
Допустим, Алиса планирует подписать документ m. Первым делом, она должна выбрать параметры RSA, аналогично тому, как это было описано в шифре. Выбираются большие простые числа $P, Q, P \neq Q$. Вычисляются числа $N = PQ$ и $\phi(N) = (P-1)(Q-1)$. Затем Алиса выбирает число d такое, что $\text{НОД}\big(d, \phi(N)\big) = 1$ и вычисляет число c такое, что $cd\ \big(mod\ \phi(N)\big) = 1$. Наконец, Алиса публикует в открытом доступе числа N, d, и хранит в секрете остальные значения. Числа d и c будем называть соответственно открытым и закрытым ключом.

Процесс подписи документа описывается следующими шагами:

1) *Алиса вычисляет значение хэш-функции документа* $h = H(m), h < N$. *Предполагается, что алгоритм вычисления всем известен.*
2) *Алиса вычисляет число* $s = h^c (mod\ N)$.
3) *Подписанный документ* $< m, s >$ *передаётся получателю*

Каждый, кто знает открытые параметры N, d, ассоциированные с Алисой, может легко проверить подлинность подписи следующим образом:

1) *Боб получает подписанный документ <m,s> и открытые параметры Алисы N,d.*
2) *Боб вычисляет значение хэш-функции документа* $h = H(m)$.
3) *Боб вычисляет значение* $e = s^d (mod\ N)$. *В случае правильности подписи, число* $e = h$.



Подпись Эль-Гамаля

Пусть, как и в предыдущем случае, Алиса планирует подписывать документы. Алиса выбирает числа P, g таким образом, что $P = 2Q + 1$, при этом P, Q – простые числа, а g – первообразный корень в поле по модулю P, т.е. такое число, что $\forall\, x, y \in \{1, P-1\}, x \neq y \Rightarrow g^x \neq g^y (mod\ P)$. Алиса выбирает случайное число $1 < x < P - 1$, которое в дальнейшем будем называть секретным ключом, и вычисляет открытый ключ $y = g^x (mod\ P)$. Числа P, g, y публикуются в открытом доступе и должны быть ассоциированы с Алисой. Теперь Алиса может подписывать документы при помощи следующих шагов:

1) *Вычисляется хэш-функцию документа* $h = H(m), 1 < h < P$.
2) *Выбирается случайное число* $1 < k < P - 1$ *такое, что* $\text{НОД}(k, P-1) = 1$, *и вычисляется число* $r = g^k (mod\ P)$.
3) *Вычисляются числа*
$$u = (h - xr)\ mod\ (P-1),$$
$$s = k^{-1} u\ mod\ (P-1),$$
где $kk^{-1}\ mod\ (P-1) = 1$.
4) *Формируется подписанный документ* <m,r,s>

Для проверки корректности подписи документа выполняются следующие шаги:

1) Вычисляется хэш-функция $h = H(m)$.
2) Проверяется следующее равенство:
$$y^r r^s = g^h (mod\ P)$$

Подпись ГОСТ Р34.10-94

Прежде всего, для всех пользователей системы, поддерживающей данный ГОСТ, формируются некоторые общие параметры. Выбираются два простых числа q размером 256 бит и p размером 1024 бита, которые связаны следующим соотношением:

$$p = bq + 1$$

для некоторого целого числа b. Требования к размерности чисел буквально означают, что старшие биты этих чисел должны быть равны единице. Затем выбирается число a такое, что

$$a^q\ mod\ p = 1$$

Для того, чтобы быстро получить такое число a, воспользуемся следующим приёмом. Выберём произвольное целое число $1 < g < p - 1$. Вычислим $a = g^b mod\ p$. Если $a > 1$, то мы нашли подходящий параметр, если нет, то выберем другое g. Полученное таким образом число будет гарантированно удовлетворять указанному выше требованию.

Получив три общих известных параметра p, q, a выполняется следующее. Каждый пользователь выбирает секретное число $0 < x < q$ и вычисляет:

$$y = a^x mod\ p$$

Число x – называется секретным ключом пользователя, а y – открытым ключом.

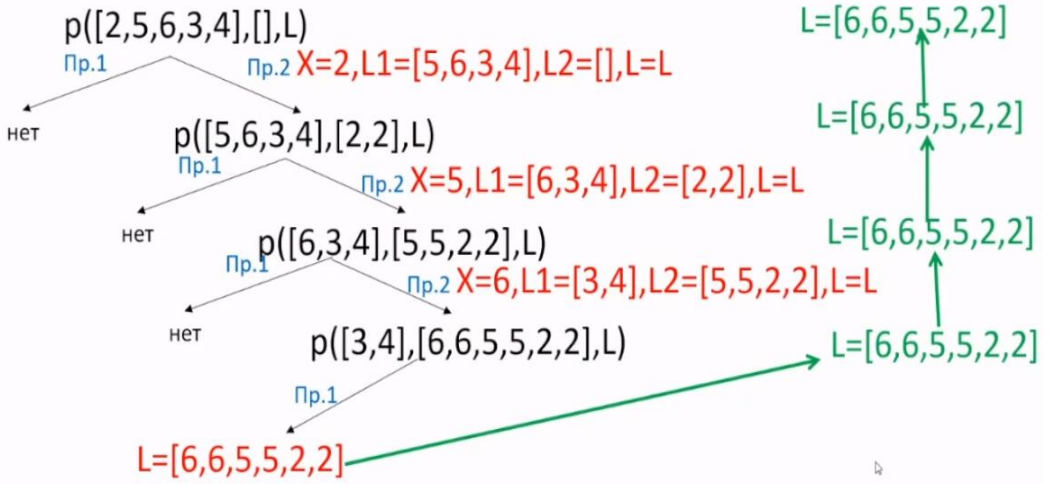
Перейдём к самому алгоритму:

1) *Абонент вычисляет хэш-функцию документа, который хочет подписать* $h = H(m)$. *Полученное значение должно лежать в пределах* $0 < h < q$.
2) *Формируется случайное число* $0 < k < q$.
3) *Вычисляем* $r = (a^k mod\ p)\ mod\ q$. *Если* $r = 0$, *то возвращаемся к шагу 2.*
4) *Вычисляем* $s = (kh + xr) mod\ q$. *Если* $s = 0$, *то возвращаемся к шагу 2.*
5) *Получаем подписанный документ* $< m, r, s >$

Для проверки подписи выполним следующие шаги:

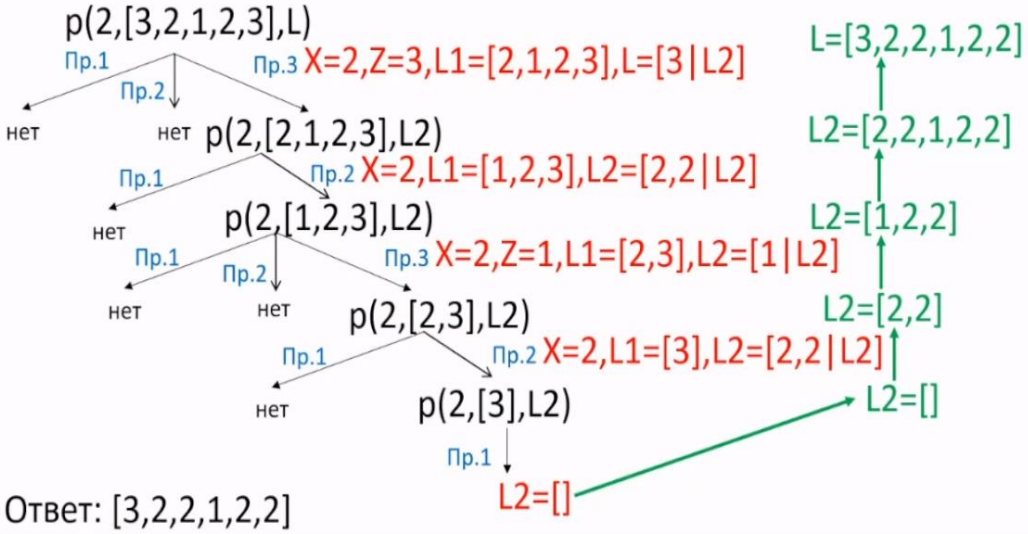
1) *Вычисляем хэш-функцию для сообщения* $h = H(m)$.
2) *Проверяем выполнение неравенств* $0 < r, s < q$.
3) *Вычисляем* $u_1 = sh^{-1} mod\ q, u_2 = -rh^{-1} mod\ q$.
4) *Вычисляем* $v = (a^{u_1} y^{u_2}\ mod\ p) mod\ q$.
5) *Проверяем выполнение равенства* $v = r$.

```
p([_,_],L,L):-!.
p([X|L1],L2,L):–p(L1,[X,X|L2],L).
```



Ответ: [6,6,5,5,2,2]

```
p(_,[_],[]):–!.
p(X,[X|L1],[X,X|L2]):–p(X,L1,L2),!.
p(X,[Z|L1],[Z|L2]):–p(X,L1,L2).
```



Ответ: [3,2,2,1,2,2]

fork

Процесс-потомок наследует следующие признаки родителя:

- сегменты кода, данных и стека программы;
- таблицу файлов, в которой находятся состояния флагов дескрипторов файла, указывающие, читается ли файл или пишется. Кроме того, в таблице файлов содержится текущая позиция указателя записи-чтения;
- рабочий и корневой каталоги;
- реальный и эффективный номер пользователя и номер группы;
- приоритеты процесса (администратор может изменить их через nice);
- контрольный терминал;
- маску сигналов;
- ограничения по ресурсам;
- сведения о среде выполнения;
- разделяемые сегменты памяти.

Потомок не наследует от родителя следующих признаков:

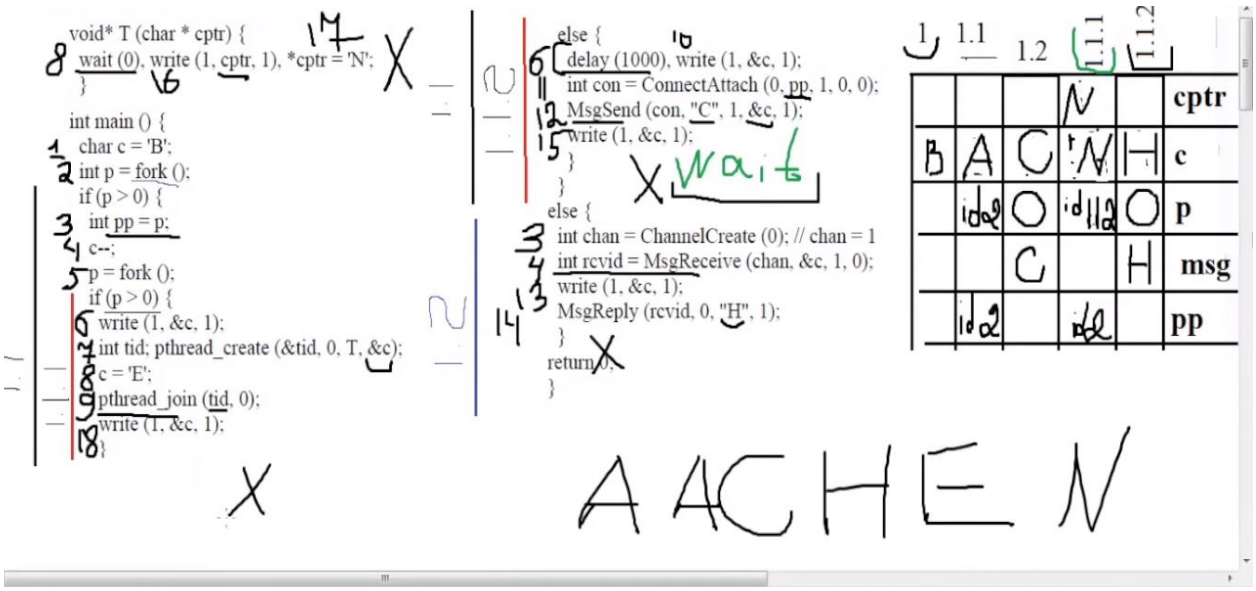
- идентификатора процесса (PID, PPID);
- израсходованного времени ЦП (оно обнуляется);
- сигналов процесса-родителя, требующих ответа;
- блокированных файлов (record locking).

При вызове fork() возникают два полностью идентичных процесса. Весь код после fork() выполняется дважды, как в процессе-потомке, так и в процессе-родителе.

Процесс-потомок и процесс-родитель получают разные коды возврата после вызова fork(). Процесс-родитель получает идентификатор (PID) потомка. Если это значение будет отрицательным, следовательно при порождении процесса произошла ошибка. Процесс-потомок получает в качестве кода возврата значение 0, если вызов fork() оказался успешным.

wait

Функция wait приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, ливбо требует вызвать функцию-обработчик. Если дочерний процесс к моменту вызова функции уже завершился (так называемый "зомби" ("zombie")), то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются.


void* T (char * cptr) {
wait (0), write (1, cptr, 1), *cptr = 'N';
}
int main () {
char c = 'B';
int p = fork ();
if (p > 0) {
int pp = p;
c--;
p = fork ();
if (p > 0) {
write (1, &c, 1);
int tid; pthread_create (&tid, 0, T, &c);
c = 'E';
pthread_join (tid, 0);
write (1, &c, 1);
}
else {
delay (1000), write (1, &c, 1);
int con = ConnectAttach (0, pp, 1, 0, 0);
MsgSend (con, "C", 1, &c, 1);
write (1, &c, 1);
}
}
else {
int chan = ChannelCreate (0); // chan = 1
int rcvid = MsgReceive (chan, &c, 1, 0);
write (1, &c, 1);
MsgReply (rcvid, 0, "H", 1);
}
return 0;
}
Wait
1
1.1
1.2
1.1.1
1.1.2
cptr
c
p
msg
pp
B A C N H
AACHEN