МГТУ им. Н. Э. Баумана

ТЕХНОПАРК

Mail.Ru Group

Лекция 3. Хеш-таблицы

Алгоритмы и структуры данных

Корепанов Д. С.

План лекции «Хеш-таблицы»

- Хеш-функции.
- Хеш-таблицы.
  - Разрешение коллизий методом цепочек.
  - Разрешение коллизий методом открытой адресации.
    - Линейное пробирование
    - Квадратичное пробирование
    - Двойное хеширование
- Другие сценарии использования хеш-функций.

Быстрый контейнер. Постановка задачи.

Задача. Хранить ключи в контейнере:

- быстро добавлять, Add
- быстро удалять, Delete
- быстро проверять наличие. Has

Решение 1.

Неупорядоченный массив:

- быстрое добавление – O(1),
- длительное удаление – O(n),
- длительный поиск – O(n).

Решение 2.

Упорядоченный массив:

- длительное добавление – O(n),
- длительное удаление – O(n),
- быстрый поиск – O(log n).

Быстрый контейнер. Постановка задачи.

Частное решение 3.
Пусть ключи – неотрицательные целые числа в диапазоне $[0, \ldots, n-1]$.
Будем хранить A – массив bool.

$A[i] = true \Leftrightarrow$ i содержится:

- мгновенное добавление – O(1),
- мгновенное удаление – O(1),
- мгновенный поиск – O(1).

Быстрый контейнер. Хеш-таблица.

Хеширование – преобразование ключей к числам.

Хеш-таблица – массив ключей с особой логикой, состоящей из:

1. Вычисления хеш-функции, которая преобразует ключ поиска в индекс.
2. Разрешения конфликтов, т.к. два и более различных ключа могут преобразовываться в один и тот же индекс массива.

Отношение порядка над ключами не требуется.

Хеш-функции

Определение. **Хеш-функция —** преобразование по детерминированному алгоритму входного массива данных произвольной длины (один ключ) в выходную битовую строку фиксированной длины (значение).

Результат вычисления хеш-фукнции называют «хешем».

Определение. **Коллизией** хеш-функции H называется два различных входных блока данных X и Y таких, что
$h(X) = h(Y)$.

Хеш-функции

Количество возможных значений хеш-функции не больше M и для любого ключа k:

$$0 \le h(k) < M$$

Важно! Хорошая хеш-функция должна:

1. Быстро вычисляться.
2. Минимизировать количество коллизий.

HASH = рубить, перемешивать.

Хеш-функции

Качество хеш-функции зависит от задачи и предметной области.

Пример плохой хеш-функции.

h(k) = [последние [три] цифры k] = k % 1000.

Такая хеш-функция порождает много коллизий, если множество ключей – цены.

Частые значения:
000, 500, 999, 998, 990, 900.

Хеш-функции. Метод деления.

$h(k) = k \bmod M$.

M определяет размер диапазона значений:
[0, .. , M–1].

Как выбрать M?

- Если $M = 2^K$, то значение хеш-функции не зависит от старших байтов.
- Если $M = 2^8 - 1$, то значение хеш-функции не зависит от перестановки байт.

Хорошо в качестве M брать простое число, далекое от степеней двойки.

Хеш-функции. Метод деления.

Сумма Флетчера - это остаток от деления интерпретируемого как длинное число потока данных на 255.

Пусть G - длинное число потока данных, $B=2^8=256$, $D = B - 1$

$$G \% D = (x_n*B^n + ... + x_1*B + x_0) \% D =$$

$$= (x_n*(...)*D + x_n + ... + x_1*D + x_1 + x_0) \% D =$$

$$= ((...)*D \% D + (x_n + ... + x_1 + x_0) \% D) \% D =$$

$$= (x_n + ... + x_1 + x_0) \% D$$

- $(D+1)^n = D^n + ... + D + 1 = (...)*D + 1$
- $(a + b) \% d = (a \% d + b \% d) \% d$

Хеш-функции. Метод умножения.

$\underline{h(k) = [M \cdot \{k \cdot A\}]}$,

где {} – дробная часть,
где [] – целая часть,
A – действительное число, $0 < A < 1$,
M определяет диапазон значений: [0, .. , M–1].

Кнут предложил в качестве A использовать число, обратное к золотому сечению:

$$A = \phi^{-1} = \left(\frac{\sqrt{5} - 1}{2}\right) = 0.6180339887 \ldots$$

Такой выбор A дает хорошие результаты хеширования.

Хеш-функции. Метод умножения.

Хеш-функцию $h(k) = [M \cdot \{k \cdot A\}]$ вычисляют без использования операций с числами с плавающими точками.

Пусть M – степень двойки. $M = 2^p, p \leq 32$.

Вместо действительного числа A берут близкое к нему $A = \frac{s}{2^{32}} = \frac{2654435769}{2^{32}}$. То есть $s = 2654435769$.

Тогда $h(k) = \left[2^p \cdot \left\{k \cdot \frac{s}{2^{32}}\right\}\right] = \left[2^p \cdot \left\{\frac{r_1 2^{32} + r_0}{2^{32}}\right\}\right] = \left[2^p \cdot \frac{r_0}{2^{32}}\right]$

$= \left[\frac{r_0}{2^{32-p}}\right] = \left[\frac{r_{01} 2^{32-p} + r_{00}}{2^{32-p}}\right] = r_{01} =$ Старшие p бит r_0.

Итого, $\boldsymbol{h(k) = \left(k \cdot s \bmod 2^{32}\right) \gg (32 - p)}$.

Хеш-функции строки.

Строка $s = s_0, s_1, \dots, s_{n-1}$.

Вариант 1. $h_1(s) = (s_0 + s_1 a + s_2 a^2 + \cdots + s_{n-1} a^{n-1}) \bmod M$.

Вариант 2. $h_2(s) = (s_0 a^{n-1} + s_1 a^{n-2} + \cdots + s_{n-2} a + s_{n-1}) \bmod M$.

Число M – степень двойки.

Важно правильно выбрать константу a.

Хотим, чтобы при изменении одного символа, хеш-функция изменялась.
То есть, чтобы все значения $s \cdot a \bmod M$, $0 \leq s < M$ были различны.
Для этого достаточно, чтобы a и M были взаимно простыми.

Хеш-функции строки

$h_2(s)$ вычисляется эффективнее, если использовать метод Горнера:

$$h_2(s) = \Big(\big((s_0 a + s_1)a + s_2\big)a + \cdots + s_{n-2}\Big)a + s_{n-1}.$$

$h_1(s)$ можно вычислять аналогично, но начиная с конца строки.

Но в с-строках известен только указатель на начало строки, а размер строки не известен.

Поэтому удобнее вычислять $h_2(s)$.

Хеш-функция строки

```
// Хеш-функция строки.
int Hash( const char* str, int m )
{
    int hash = 0;
    for( ; *str != 0; ++str )
        hash = ( hash * a + *str ) % m;
    return hash;
}
```

Парадокс дней рождений.

Сколько необходимо взять человек, что бы вероятность совпадения дней рождения (число и месяц) хотя бы у двух людей превышает 50 %?

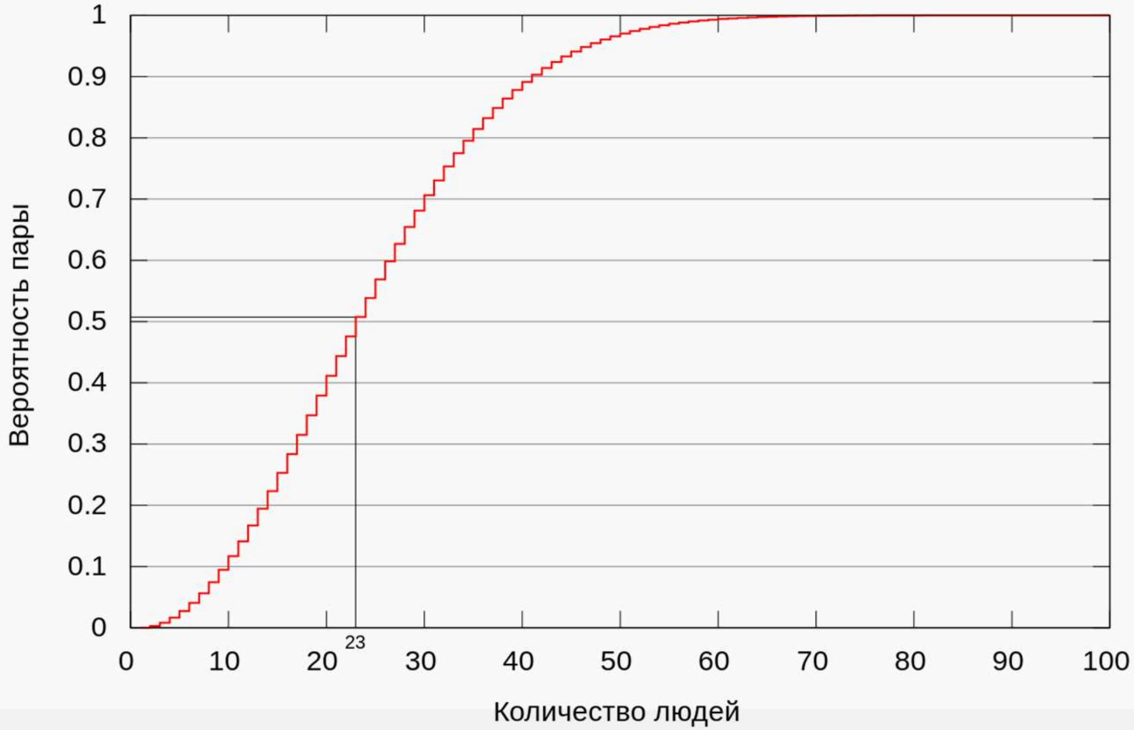
Хеш-функции

Вычислим вероятность того, что все дни рождения в группе будут различными:

$$\bar{p}(n) = 1\cdot\left(1-\frac{1}{365}\right)\cdot\left(1-\frac{2}{365}\right)\cdots\left(1-\frac{n-1}{365}\right) = \frac{365\cdot 364\cdots(365-n+1)}{365^n} = \frac{365!}{365^n(365-n)!},$$

Тогда вероятность того, что хотя бы у двух человек из *n* дни рождения совпадут:

$$p(n) = 1 - \bar{p}(n).$$



Ответ: 23.

Хеш-таблицы

При вставке в хеш-таблицу размером 365 ячеек всего лишь 23-х элементов вероятность коллизии уже превысит 50 %, при вставке 50 элементов вероятность превысит 97% (если каждый элемент может равновероятно попасть в любую ячейку).

Хеш-таблицы различаются по методу разрешения коллизий.

Основные **методы** разрешения коллизий:

1. Метод цепочек.
2. Метод открытой адресации.

Хеш-таблицы

Определение. Хеш-таблица – структура данных, хранящая ключи в таблице. Индекс ключа вычисляется с помощью хеш-функции. Операции: добавление, удаление, поиск.

Пусть хеш-таблица имеет размер M, количество элементов в хеш-таблице – N.

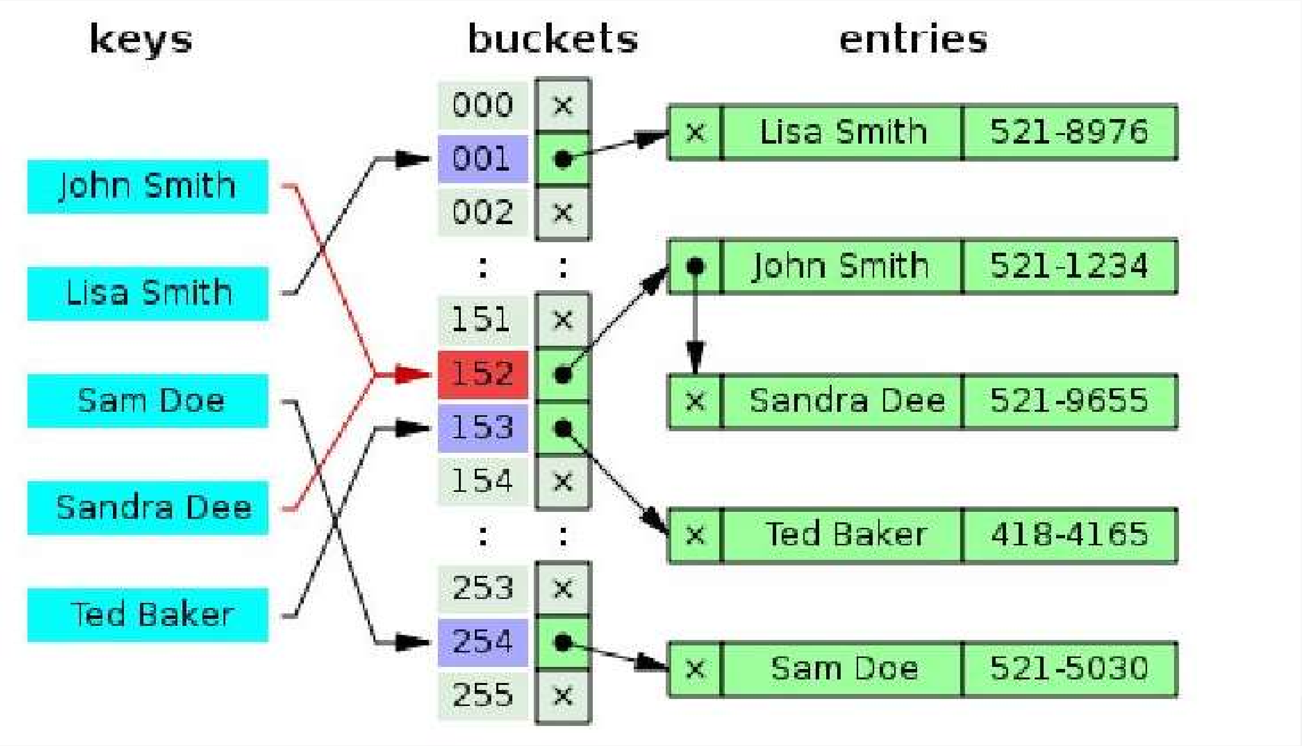
Определение. Число хранимых элементов, делённое на размер массива (число возможных значений хеш-функции), называется **коэффициентом заполнения хеш-таблицы** (load factor). Обозначим его $\alpha = \frac{N}{M}$.

Этот коэффициент является важным параметром, от которого зависит среднее время выполнения операций.

Хеш-таблицы. Метод цепочек.

Каждая ячейка массива является указателем на связный список (цепочку).

Коллизии приводят к тому, что появляются цепочки длиной более одного элемента.



Хеш-таблицы. Метод цепочек.

Добавление ключа.

1. Вычисляем значение хеш-функции добавляемого ключа – h.
2. Находим A[h] – указатель на список ключей.
3. Вставляем в начало списка (в конец списка дольше). Если запрещено дублировать ключи, то придется просмотреть весь список.

Время работы:
В лучшем случае – **O(1)**.
В худшем случае
– если не требуется проверять наличие дубля, то **O(1)**,
– иначе – **O(N)**.

Хеш-таблицы. Метод цепочек.

Удаление ключа.

1. Вычисляем значение хеш-функции удаляемого ключа – h.
2. Находим A[h] – указатель на список ключей.
3. Ищем в списке удаляемый ключ и удаляем его.

Время работы:
В лучшем случае – **O(1)**.
В худшем случае – **O(N)**.

Хеш-таблицы. Метод цепочек.

Поиск ключа.

1. Вычисляем значение хеш-функции ключа – h.
2. Находим A[h] – указатель на список ключей.
3. Ищем его в списке.

Время работы:
В лучшем случае – **O(1)**.
В худшем случае – **O(N)**.

Хеш-таблицы. Метод цепочек.

Среднее время работы.

Теорема. Среднее время работы операций поиска, вставки (с проверкой на дубликаты) и удаления в хеш-таблице, реализованной методом цепочек – $O(1+\alpha)$, где α – коэффициент заполнения таблицы.

Доказательство. Среднее время работы – математическое ожидание времени работы в зависимости от исходного ключа.

Время работы для обработки одного ключа $T(k)$ зависит от длины цепочки и равно $\mathrm{O}(1+N_{h(k)})$, где N_i – длина i-ой цепочки. Предполагаем, что хеш-функция равномерна, а ключи равновероятны.

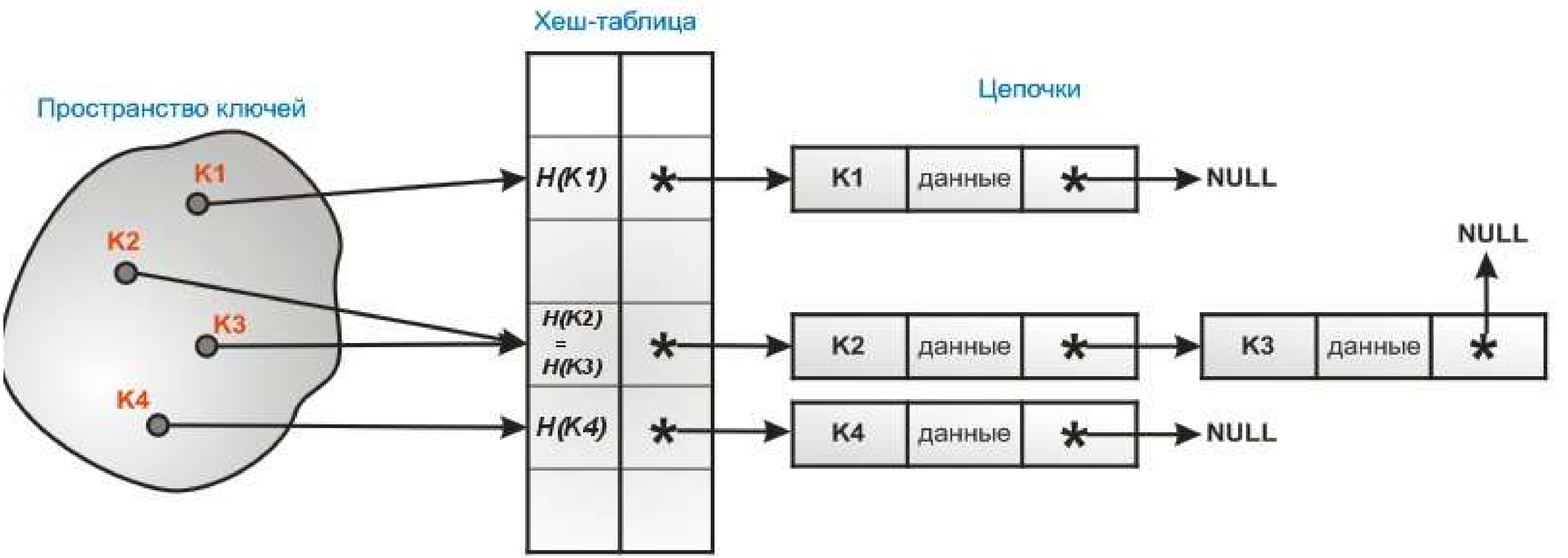
Среднее время работы

$$T_{\text{ср}}(M,N) = M\big(T(k)\big) = \sum_{i=0}^{M-1} \frac{1}{M}(1+N_i) = \frac{1}{M}\sum_{i=0}^{M-1}(1+N_i) = \frac{M+N}{M}$$
$$= 1+\alpha$$

Хеш-таблицы. Метод цепочек.

```
// Хеш-функция.
template<class T>
int Hash( T& data );

// Элемент цепочки в хеш-таблице.
template<class T>
struct HashTableNode {
   T Data;
   HashTableNode<T>* Next;
};

// Хеш-таблица.
template<class T>
class HashTable {
public:
   HashTable( int initialSize );
   bool Has( const T& key ) const;
   void Add( const T& key );
   bool Delete( const T& key );
private:
   vector<HashTableNode<T>*> table;
};
```

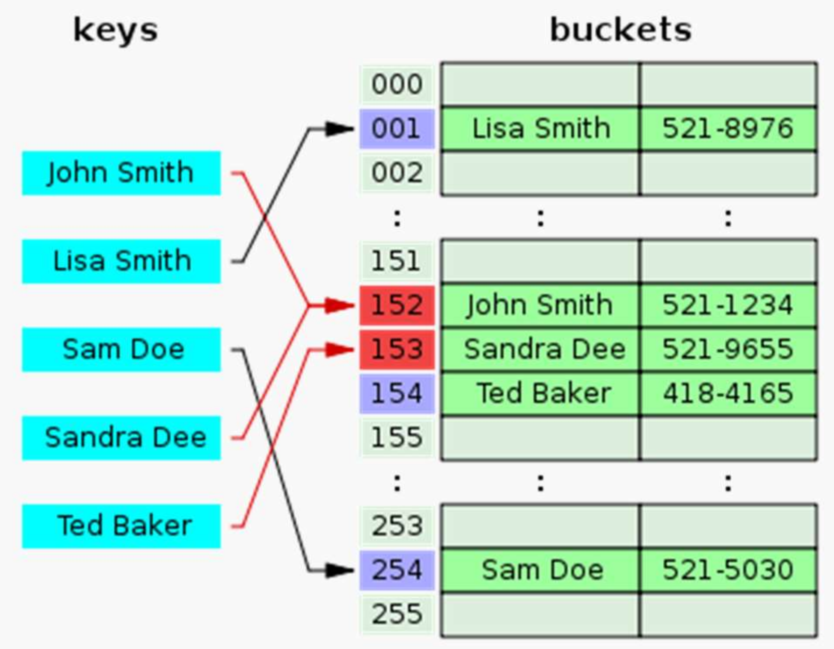
Хеш-таблицы. Метод цепочек.



Хеш-таблицы. Открытая адресация.

Все элементы хранятся непосредственно в массиве.
Каждая запись в массиве содержит либо элемент, либо NIL.

При поиске элемента систематически проверяем ячейки до тех пор, пока не найдем искомый элемент или не убедимся в его отсутствии.



Хеш-таблицы. Открытая адресация.

Вставка ключа.

1. Вычисляем значение хеш-функции ключа – h.
2. Систематически проверяем ячейки, начиная от A[h], до тех пор, пока не находим пустую ячейку.
3. Помещаем вставляемый ключ в найденную ячейку.

В п.2 поиск пустой ячейки выполняется в некоторой последовательности. Такая последовательность называется **«последовательностью проб».**

Хеш-таблицы. Открытая адресация.

Последовательность проб зависит от вставляемого в таблицу ключа. Для определения исследуемых ячеек расширим хеш-функцию, включив в нее номер пробы (от 0).

$$h : U \times \{0, 1, \dots, M-1\} \to \{0, 1, \dots, M-1\}.$$

Важно, чтобы для каждого ключа k последовательность проб

$$\langle h(k,0), h(k,1), \dots, h(k,M-1) \rangle$$

представляла собой перестановку множества $\langle 0,1,\dots,M-1 \rangle$, чтобы могли быть просмотрены все ячейки таблицы.

Хеш-таблицы. Открытая адресация.

```
// Вставка ключа в хеш-таблицу (без учета удаленных
элементов).
void HashTable::Add( const T& k )
{
    for( int i = 0; i < tableSize; ++i ) {
        int j = h( k, i );
        if( IsNil( table[j] ) ) {
            table[j] = k;
            return;
        }
    }
    throw HashTableException( “Overflow” );
}
```

Хеш-таблицы. Открытая адресация.

Поиск ключа.

Исследуется та же последовательность, что и в алгоритме вставки ключа.

Если при поиске встречается пустая ячейка, поиск завершается неуспешно, поскольку искомый ключ должен был бы быть вставлен в эту ячейку в последовательности проб, и никак не позже нее.

Хеш-таблицы. Открытая адресация.

Удаление ключа.

Алгоритм удаления достаточен сложен.

Нельзя при удалении ключа из ячейки i просто пометить ее значением NIL. Иначе в последовательности проб для некоторого ключа (или некоторых) возникнет пустая ячейка, что приведет к неправильной работе алгоритма поиска.

Решение. Помечать удаляемые ячейки спец. значением «Deleted».

Нужно изменить методы поиска и вставки.
В методе вставки проверять «Deleted», вставлять на его место.
В методе поиска продолжать поиск при обнаружении «Deleted».

Хеш-таблицы. Открытая адресация.

Вычисление последовательности проб.

Желательно, чтобы для различных ключей k последовательность проб $\langle h(k,0), h(k,1), \dots, h(k,M-1)\rangle$ давала большое количество последовательностей-перестановок множества $\langle 0,1,\dots,M-1\rangle$.

Обычно используются три метода построения h(k, i):

1. Линейное пробирование.
2. Квадратичное пробирование.
3. Двойное хеширование.

Хеш-таблицы. Открытая адресация.

Линейное пробирование.

$$h(k, i) = (h'(k) + c\ i) \bmod M.$$

Основная проблема – кластеризация.

Последовательность подряд идущих занятых элементов таблицы быстро увеличивается, образуя кластер.

Попадание в элемент кластера при добавлении гарантирует «одинаковую прогулку» для различных ключей и проб. Новый элемент будет добавлен в конец кластера, увеличивая его.

Если $h(k_1, i) = h(k_2, j)$, то $h(k_1, i + r) = h(k_2, j + r)$ для всех r.

Хеш-таблицы. Открытая адресация.

Теорема. 1) Если a и M не являются взаимно простыми, то $\{s \cdot a \bmod M, 0 \le s < M\} \ne \{0, \dots, M-1\}$.

2) Если a и M взаимно просты, то $\{s \cdot a \bmod M, 0 \le s < M\} = \{0, \dots, M-1\}$.

Доказательство. 1) Пусть a и M не являются взаимно простыми. Тогда a и M имеют общий делитель d.
$a = d \cdot x, M = d \cdot y$.
Для любого s остаток от деления $s \cdot a$ на M также делится на d:
$s \cdot a = M \cdot k + r, r = s \cdot d \cdot x - d \cdot y \cdot k = d(sx - yk)$.

2) От противного. Пусть множество $\{s \cdot a \bmod M, 0 \le s < M\}$ имеет меньше M различных элементов. Тогда существуют i и j, что $ia \equiv ja \pmod M$, $i < j < M$. Следовательно, $(j - i)a = M \cdot u$. Из этого следует, что $j - i$ делится на M, т.к. a и M – взаимно простые. Но $0 < j - i < M$. Противоречие.

Хеш-таблицы. Открытая адресация.

Квадратичное пробирование.

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod M.$$

Требуется, чтобы последовательность проб содержала все индексы 0, .. , M–1. Требуется подбирать c_1 и c_2.

При $c_1 = c_2 = 1/2$, то проба вычисляется рекуррентно:

$$h(k, i+1) = h(k, i) + i + 1 \ (mod\ M).$$

Возникает **вторичная кластеризация**. Проявляется на ключах с одинаковым хеш-значением $h'(\cdot)$.
Если $h(k_1, 0) = h(k_2, 0)$, то $h(k_1, i) = h(k_2, i)$ для всех i.
Соответствует цепочкам в методе цепочек. Разница лишь в том, что в методе открытой адресации эти цепочки могут еще пересекаться.

Хеш-таблицы. Открытая адресация.

Квадратичное пробирование.

Утверждение. Если $c_1 = c_2 = 1/2$, а $M = 2^p$, то квадратичное пробирование дает перестановку {0, 1, 2, 3, …, M-1}.

Доказательство. От противного. Пусть существуют i и j, $0 \leq i, j \leq M - 1$, для которых

$$\frac{i(i+1)}{2} \equiv \frac{j(j+1)}{2} (mod\ 2^p).$$

Тогда

$$i^2 + i - j^2 - j = 2^{p+1}D,$$
$$(i-j)(i+j+1) = 2^{p+1}D,$$

Если i и j одинаковой четности, то $i + j + 1$ нечетна, но $i - j$ не может делиться на 2^{p+1}.

Если i и j разной четности, то $i - j$ нечетна, но $i + j + 1$ не может делиться на 2^{p+1}, т.к. $0 < i + j + 1 < 2^{p+1}$. Противоречие.

Хеш-таблицы. Открытая адресация.

Двойное хеширование.

$$h(k, i) = \left(h_1(k) + i h_2(k)\right) \bmod M.$$

Требуется, чтобы последовательность проб содержала все индексы 0, .. , M–1. Для этого все значения $h_2(k)$ должны быть взаимно простыми с M.

- M может быть степенью двойки, а $h_2(k)$ всегда возвращать нечетные числа.
- M простое, а $h_2(k)$ меньше M.

Общее количество последовательностей проб = $O(M^2)$.

Хеш-таблицы. Открытая адресация.

Анализ хеш-таблиц с открытой адресацией.

Теорема. Математическое ожидание количества проб при неуспешном поиске в хеш-таблице с открытой адресацией и коэффициентом заполнения $\alpha = \frac{n}{m} < 1$ в предположении равномерного хеширования не превышает $\frac{1}{1-\alpha}$.

Без доказательства.

Время работы методов поиска, добавления и удаления:
В лучшем случае – $O(1)$.
В худшем случае – $O(N)$.
В среднем – $O\left(\frac{1}{1-\alpha}\right)$.

Хеш-таблицы. Открытая адресация.

Плюсы.

+ Основное преимущество метода открытой адресации – не тратится память на хранение указателей списка.
+ Нет элементов, хранящихся вне таблицы.

Минусы.

– Хеш-таблица может оказаться заполненной. Коэффициент заполнения α не может быть больше 1.
– При приближении коэффициента заполнения α к 1 среднее время работы поиска, добавления и удаления стремится к N.
– Сложное удаление.

Динамическая хеш-таблица.

Изначально может быть неизвестно количество хранимых ключей. Коэффициент заполнения α может приближаться к 1, а в реализации методом цепочек может быть больше 1.

Среднее время работы для метода цепочек: $O(1+\alpha)$, для открытой адресации $O(1/(1-\alpha))$.

Требуется динамически увеличивать размер таблицы. Аналогично динамическому массиву.

Процесс увеличения размера хеш-таблицы называется «**перехешированием**».

Динамическая хеш-таблица.

Перехеширование.

1. Создать новую пустую таблицу. Размер новой таблицы $\widetilde{M}$ может быть равен $2 \cdot M$, где M – размер старой таблицы. Если размер таблицы должен быть простым, то следует использовать простое число близкое к $2 \cdot M$.
2. Проитерировать старую таблицу. Каждый ключ старой таблицы перенести в новую. Для добавления в новую таблицу надо использовать другую хеш-функцию, возвращающую значения от 0 до $\widetilde{M} - 1$.

Динамическая хеш-таблица.

Когда выполнять перехеширование?

Для разных хеш-таблиц следует использовать разные стратегии.

Для хеш-таблиц, реализованных методом цепочек:

Например, когда коэффициент заполнения α достиг 2-3.

Для хеш-таблиц, реализованных методом открытой адресации:

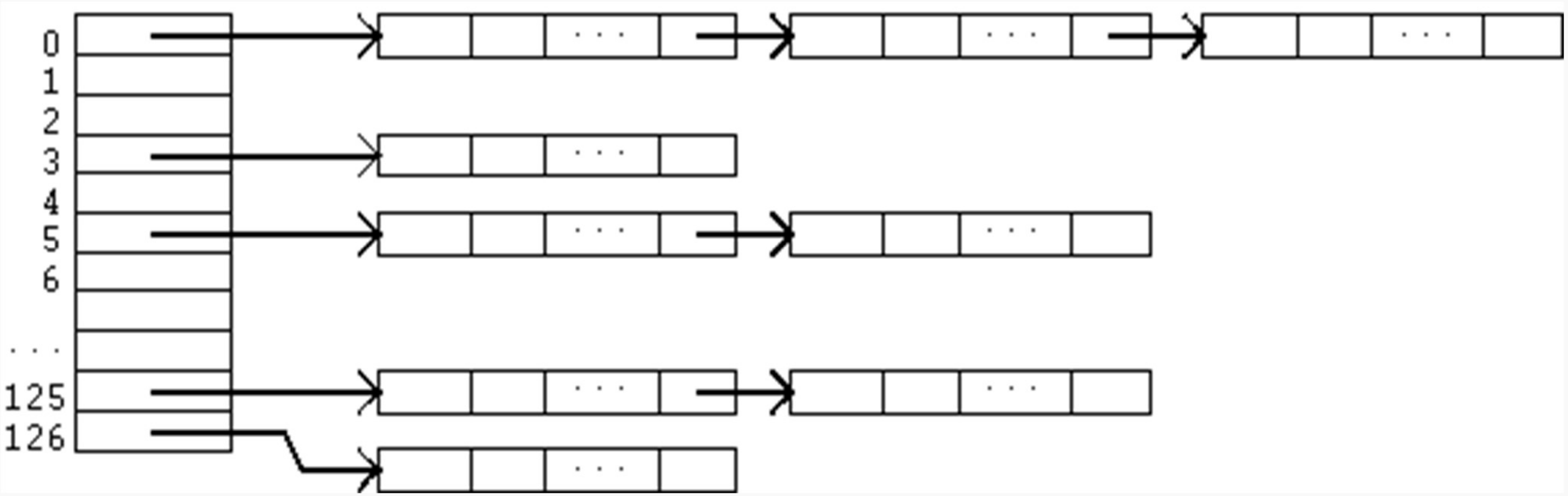
Например, когда α достиг значения ⅔ или ¾.

Хеш-таблицы. Время работы.

	Лучший случай.	В среднем. Метод цепочек.	В среднем. Метод открытой адресации.	Худший случай.
Поиск	$O(1)$	$O(1+\alpha)$	$O\left(\frac{1}{1-\alpha}\right)$	$O(N)$
Вставка	$O(1)$	$O(1+\alpha)$	$O\left(\frac{1}{1-\alpha}\right)$	$O(N)$
Удаление	$O(1)$	$O(1+\alpha)$	$O\left(\frac{1}{1-\alpha}\right)$	$O(N)$

Хеш-таблицы. Более экзотические варианты.

- Рандомное пробирование:
  $h(k, i) = h(k) + r_i$, где r_i - элемент псевдо-рандомной последовательности, сгенерированный для заданного seed
- Цепочка bucket-ов



- https://habr.com/ru/company/mailru/blog/323242/

Контрольная сумма (checksum).

Функции вычисления контрольной суммы также являются хеш-функциями.

Предназначена для проверка целостности данных

Простейший вариант – простое суммирование значений байт:

Сообщение: 12 34 56
Контрольная сумма: 12+34+56=102

Пример – валидация штрих-кода EAN-13.

4 600123 456782

Сложи цифры на четных позициях, умножь сумму на три и добавь цифры на нечетных позициях. Должно получиться число, которое делится на десять:

(6 + 0 + 2 + 4 + 6 + 8) * 3 + 4 + 0 + 1 + 3 + 5 + 7 + 2 = 78 + 22 = 100

Вариант посложнее – брать по 4 байта из файла, и делаем с ними логическую операцию XOR

CRC.

CRC — циклически избыточный код, Cyclic redundancy check.

CRC = сообщение % полином

Возьмём исходное сообщение: $K(x) = k_0 + k_1 x + \cdots + k_n x^n$,

И порождающий многочлен: $P(x) = p_0 + p_1 x + \cdots + p_m x^m$.

Поделим исходное сообщение на многочлен с остатком:

$$K(x) = A(x) \cdot P(x) + R(x),$$

$A(x)$ – частное, $R(x)$ – остаток, $\deg R < \deg P = m$.

$$R(x) = r_0 + r_1 x + \cdots + r_{m-1} x^{m-1}$$

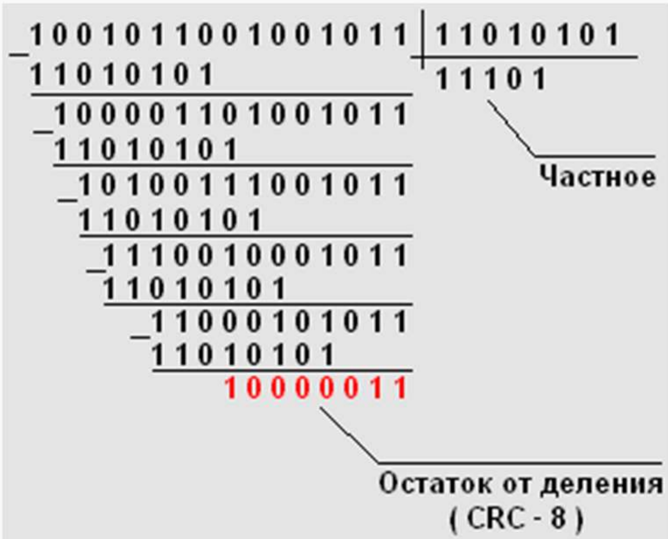
Все коэффициенты в поле Z_2.

CRC.

Пример расчёта CRC-8:

Исходный массив данных: 1001 0110 0100 1011.

Порождающий многочлен: 1101 0101.



Пример расчета контрольной суммы CRC - 8

CRC.

Обычно при вычислении CRC исходное сообщение умножается на x^m:

$$H_P(K)(x) = K(x) \cdot x^m \bmod P(x)$$

Для разных стандартов CRC используются многочлены разных степеней, с разными коэффициентами.

CRC стандарт	Многочлен
CRC-1	$x + 1$
CRC-5-USB	$x^5 + x^2 + 1$
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$
CRC-16(-IBM)	$x^{16} + x^{15} + x^2 + 1$
CRC-32-IEEE 802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
CRC-64-ISO	$x^{64} + x^4 + x^3 + x + 1$

CRC.

Типичная эффективная реализация:

```
crc = CRCINIT
while( bufflen-- )
      crc = t[(crc ^ *buff++) & 0xFF] ^ (crc >> 8);
```

CRC32 используется в:

- **TCP/IP**
- **Zip/RAR**

Криптографические хеш-функции.

Хеш-функции удовлетворяющие требованиям:

1. **Стойкость к поиску первого прообраза** — отсутствие эффективного полиномиального алгоритма вычисления обратной функции, т.е. нельзя восстановить текст m по известному хеш-значению H(m) за реальное время (необратимость).

2. **Стойкость к поиску второго прообраза** — вычислительно невозможно, зная сообщение m и его хеш-значение H(m), найти такое другое сообщение $m' \neq m$, чтобы $H(m) = H(m')$.

3. **Стойкость к коллизиям** — нет эффективного полиномиального алгоритма, позволяющего найти два разных сообщения с одинаковыми хеш-значениями.

Криптографические хеш-функции.

MD1, MD2, MD3, MD4, MD5, MD6, SHA-1, SHA-2 , SHA-3 — известные криптографические хеш-функции/семейство хеш-функций.

MD = Message Digest. Один из самых популярных – MD5 – 128-битный алгоритм хеширования. Разработан Рональдом Л. Ривестом в 1991г. Использует битовые операции с блоками длины 128.

SHA = Secure Hash Code. Один из самых популярных – SHA-256.

Важная особенность криптографической хеш-функции – лавинный эффект. Замена одного символа приводит к полному изменению значения хеша:

MD5("md5") = 1BC29B36F623BA82AAF6724FD3B16718.

MD5("md4") = C93D3BF7A7C4AFE94B64E30C2CE39F4F

Криптографические хеш-функции.

Практическое использование:

- Проверка целостности
- Поиск дублей
- Проверка парольной фразы
- Цифровая подпись
- Блокчейн

MD5

Шаг 1. Выравнивание потока

Сначала к концу данных дописывают единичный бит. Затем добавляют некоторое число нулевых байт, чтобы длина данных стала сравнима с 448 по модулю 512

Шаг 2. Добавление длины сообщения

В конец данных дописывают 64-битное представление длины данных

Шаг 3. Инициализация буфера

Для вычислений инициализируются 4 переменных размером по 32 бита:

```
A = 01 23 45 67; // 67452301h
B = 89 AB CD EF; // EFCDAB89h
C = FE DC BA 98; // 98BADCFEh
D = 76 54 32 10. // 10325476h
```

MD5

Шаг 4. Вычисление в цикле

Определяем 4 вспомогательные функции:

- F(x, y, z) = (x & y) | (~x & z)
- G(x, y, z) = (x & z) | (y & ~z)
- H(x, y, z) = x ^ y ^ z
- I(x, y, z) = y ^ (x | ~z)

Инициализируем таблицу констант T[1..64]:

T[i] = int(4294967296 * abs(sin(i)))

Каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов. Для этого блок представляется в виде массива X из 16 слов по 32 бита.

MD5

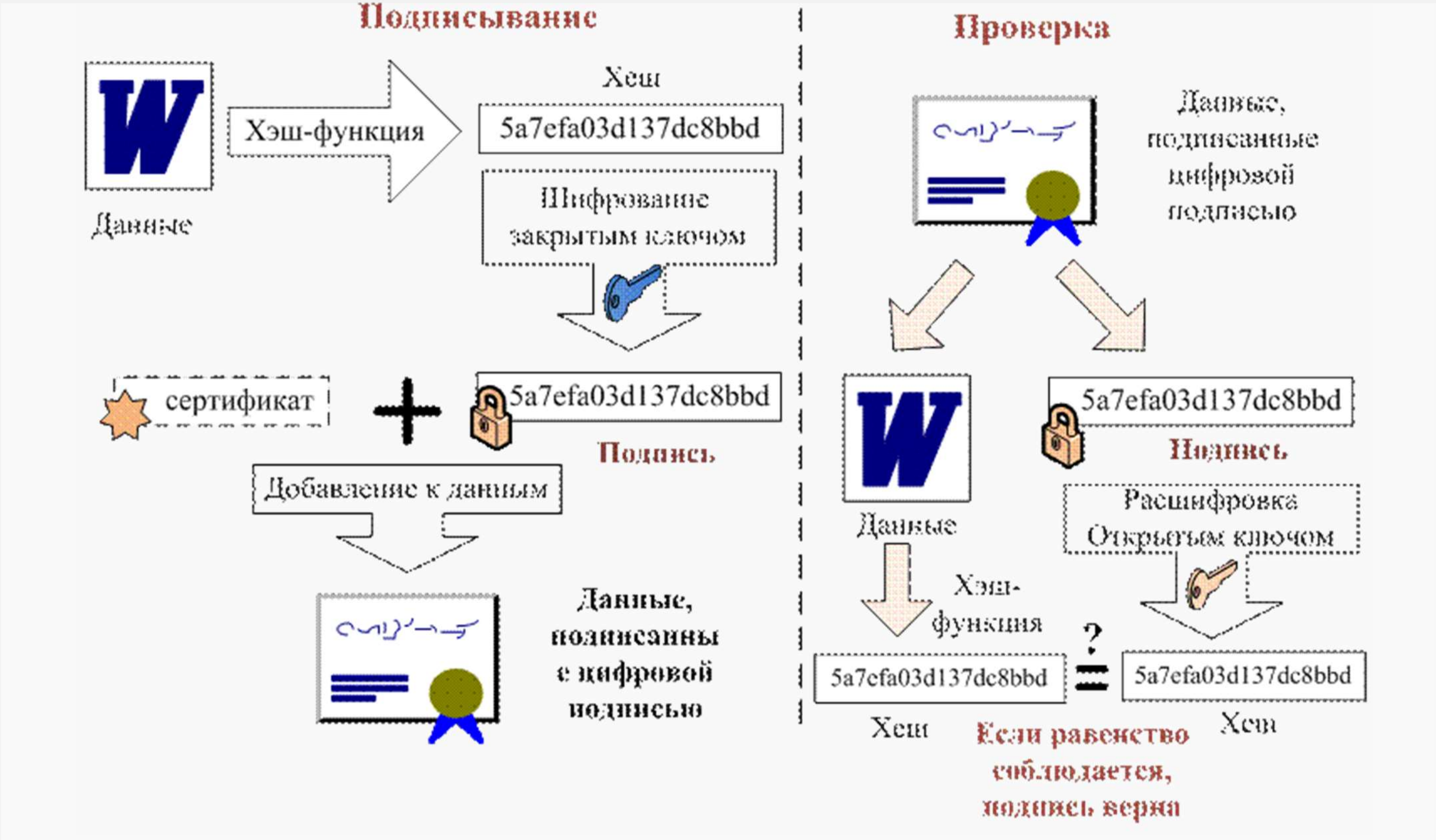
```
tempA = A
tempB = B
tempC = C
tempD = D

/* [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  0 7  1][DABC  1 12  2][CDAB  2 17  3][BCDA  3 22  4]
[ABCD  4 7  5][DABC  5 12  6][CDAB  6 17  7][BCDA  7 22  8]
[ABCD  8 7  9][DABC  9 12 10][CDAB 10 17 11][BCDA 11 22 12]
[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

/* ещё 3 аналогичных этапа с вызовом функций G, H, I */

A += tempA
B += tempB
C += tempC
D += tempD
```

MD5

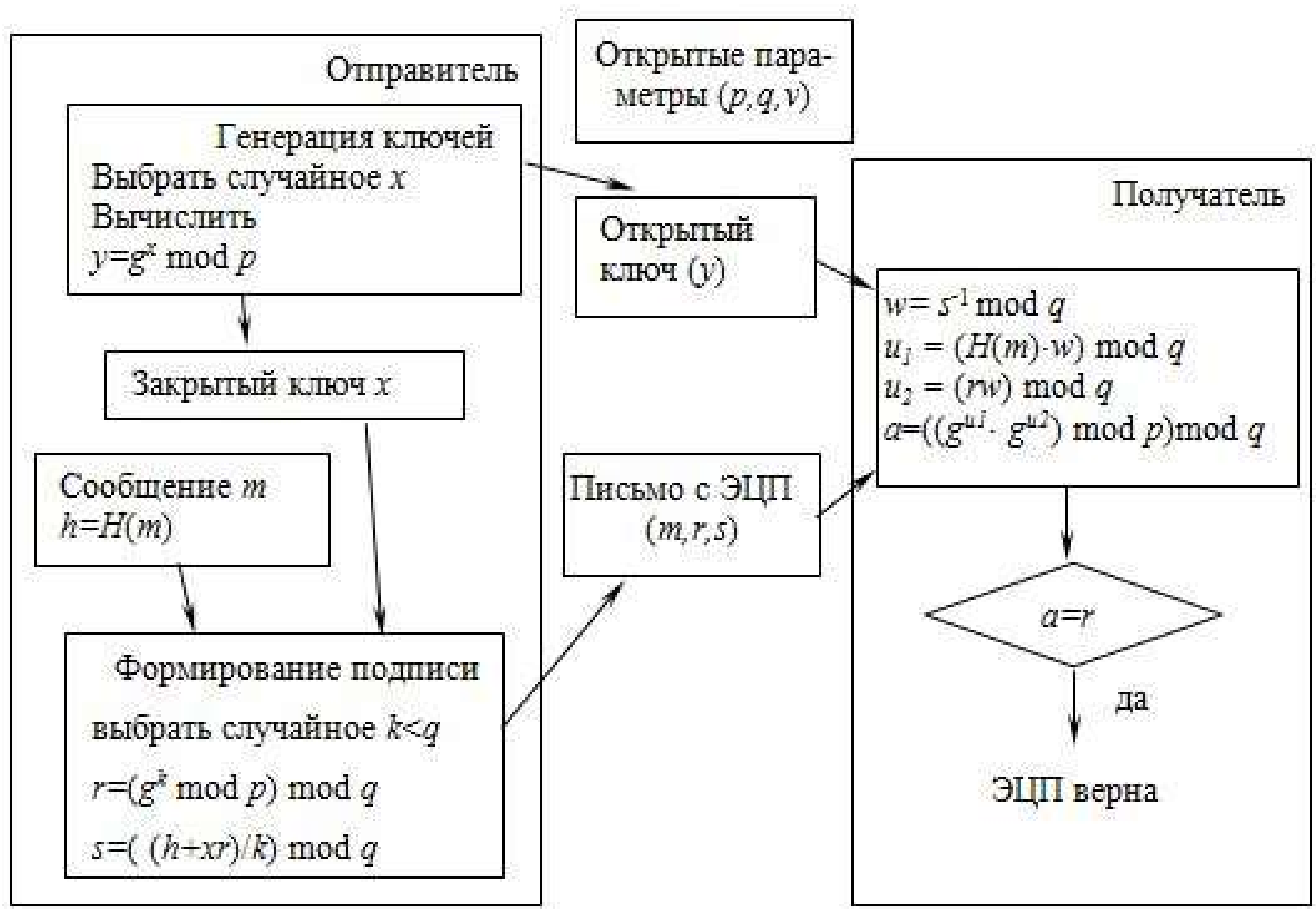
История взлома MD5:

- В 1996 году Ганс Доббертин нашёл псевдоколлизии в MD5, используя определённый инициализирующий буффер (ABCD).
- В 2004 году китайские исследователи Ван Сяоюнь, Фен Дэнгуо, Лай Сюэцзя и Юй Хунбо объявили об обнаруженной ими уязвимости в алгоритме, позволяющей за небольшое время (1 час на кластере IBM p690) находить коллизии.
- В 2005 году Ван Сяоюнь и Юй Хунбо из университета Шаньдуна в Китае опубликовали алгоритм, который может найти две различные последовательности в 128 байт, которые дают одинаковый MD5-хеш.
- В 2006 году чешский исследователь Властимил Клима опубликовал алгоритм, позволяющий находить коллизии на обычном компьютере с любым начальным вектором (A,B,C,D) при помощи метода, названного им «туннелирование».

Цифровая подпись DSA.



Цифровая подпись DSA.



Цифровая подпись DSA.

Подпись сообщения

- Выбор случайного числа k из (0; q), где q – большое простое по размерности h(m)
- Вычисление r = (g^k mod p) mod q, где p просто такое что (p-1) делится на q, а g такое, что его мультипликативный порядок по модулю p равен q
- Вычисление s = (k^{-1}(h(m) + x r)) mod q, где x – закрытый ключ
- Выбор другого k, если оказалось, что r = 0 или s = 0

Подписью является пара чисел (r, s)

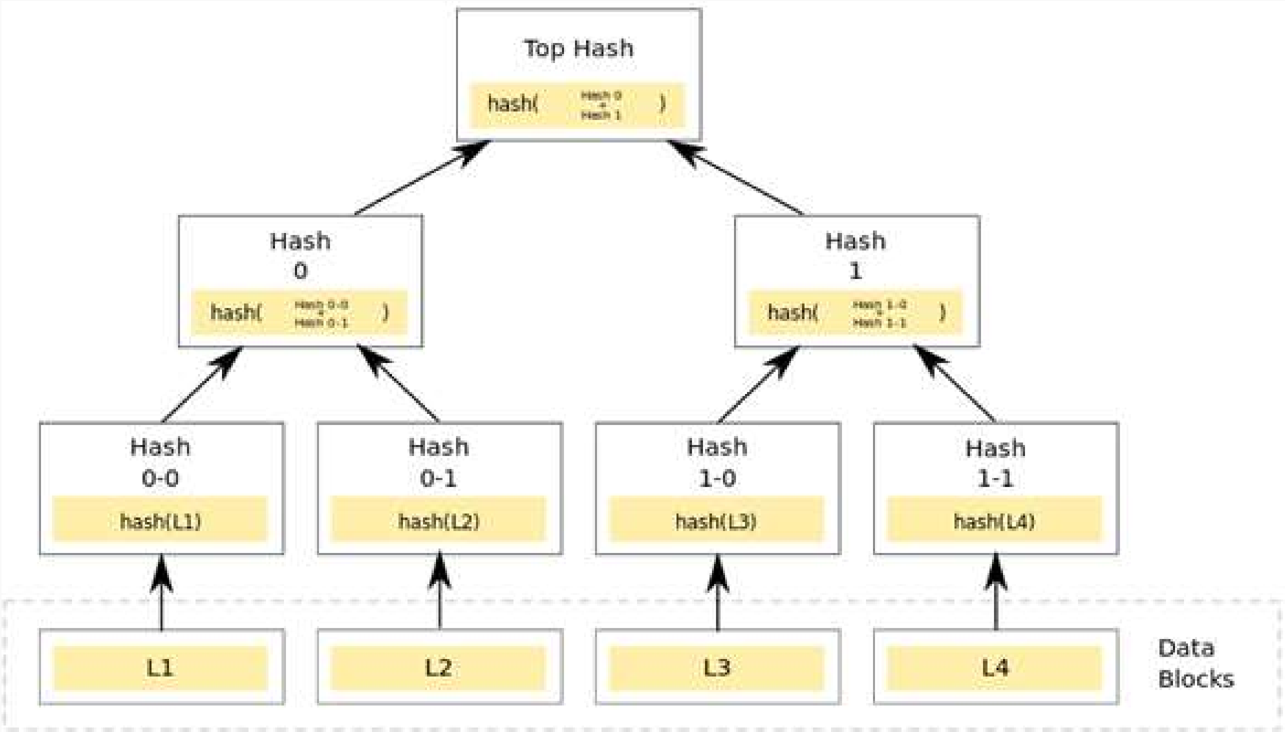
Проверка подписи

- Вычисление w = s^{-1} mod q
- Вычисление u_1 = (h(m) w) mod q
- Вычисление u_2 = (r w) mod q
- Вычисление v = ((g^{u_1} y^{u_2}) mod p) mod q, где y = g^x mod p – открытый ключ

Подпись верна, если v = r

Открытыми параметрами являются числа (p, q, g, y). Закрытый параметр только один — число x. При этом числа (p, q, g) могут быть общими для группы пользователей, а числа x и y являются соответственно закрытым и открытым ключами конкретного пользователя.

Дерево Меркла.

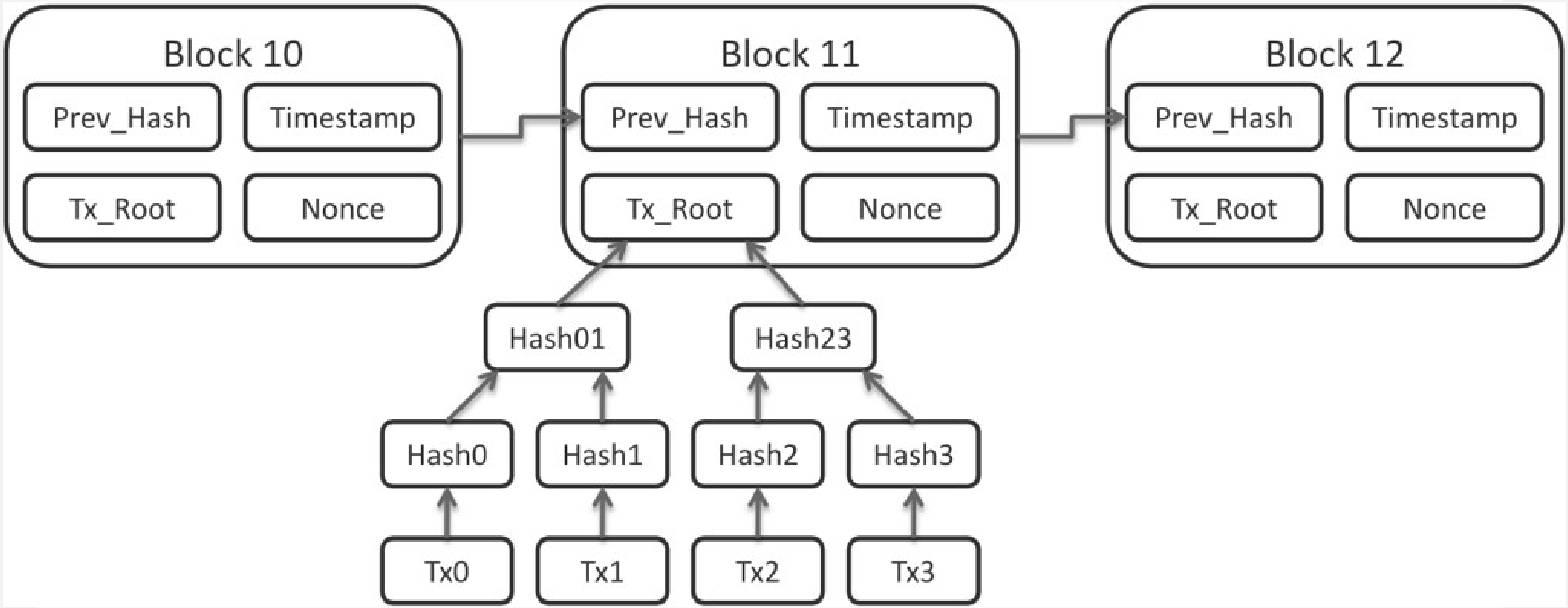


Деревья Меркла используются в:

- Файловых системах для проверки целостности файлов
- Распределённых БД для быстрой синхронизации копий
- Блокчейнах для упрощенной верификации платежей в «легких клиентах»

Блокчейн.

Блокчейн — цепочка криптографически связанных блоков.



Блокчейн.

Пример современного хеша блока BTC –

0000000000000000000526273c1abbdb82cc3f7964b5c287193eeaf0f86d14b3

Хешируются с помощью SHA-256 данные:

- Хеш списка добавленных транзакций до 1Мб
- Хеш предыдущего блока
- Timestamp
- Nonce (соль, подбирается)

Если значение меньше порогового значения, то блок может быть добавлен в цепочку.

Порог = Максимум / Сложность.

Вычисляется на основе истории так, чтобы очередные блоки находились в среднем раз в 10 минут.

Блокчейн биткоина.

В блокчейне биткоина по состоянию на 19.11.21:

- находится 710 423 блок
- средний размер блока 1980 транзакций
- общий размер блокчейна 435.71 Гб.
- за последние сутки сгенерировано 149 блока (~1 блок в 9.5 минут)
- за последние сутки выполнено 295 100 транзакции

https://bitinfocharts.com/ru/bitcoin/

МГТУ им. Н. Э. Баумана

ТЕХНОПАРК

Mail.Ru Group

???????????

Спасибо за внимание!