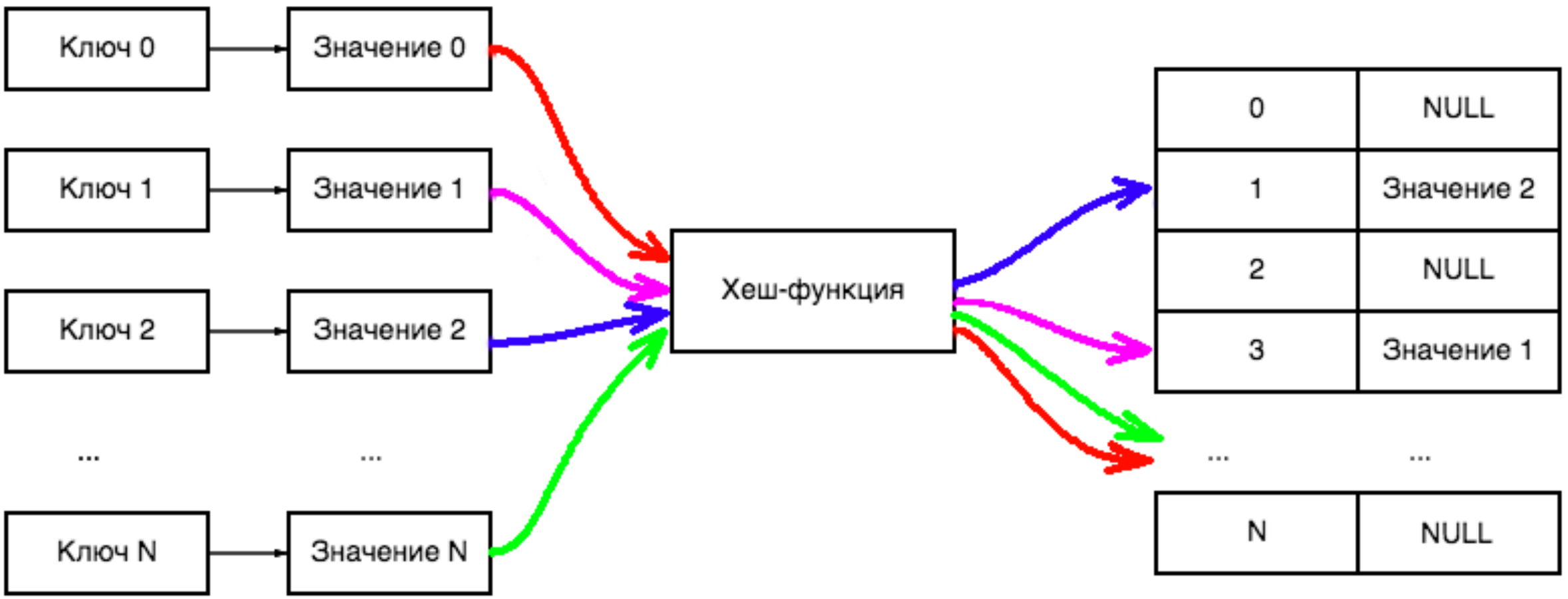
Хеш-таблицы

Хеш-таблица (hash-table) — структура данных, реализующая интерфейс ассоциативного массива. Такая структура данных способна эффективно хранить ключи и пары ключей и их значений.
Для хеш-таблиц характерно быстрое выполнение таких операции, как вставки/удаления/поиск. В лучшем случае для всех трех операций сложность будет составлять константное время O(1), однако в некоторых случаях сложность может быть выше.

В языке C++ хеш-таблицы представлены `unoredered_set` и `unordered_map`.

Работа хеш-таблицы связана с функцией хеширования. Функция хеширования принимает в себя ключ (в некотором, заранее оговорённом виде) а на выходе отдает некоторое положительное натуральное число от 0 до N.



К хеш-функции возникают определенные требования, такие как:

1) Детерминированность. Для одного и того же ключа хеш-функция должна возвращать одно и тоже значение.
2) Равномерность. То есть функция должна быть подобрана таким образом, чтобы минимизировать коллизии.
3) Эффективность. Вычисление хеш-функции должно происходить быстро.
4) Ограниченность. В итоге хеш-функция должна вернуть натуральное число, которое может служить индексом массива, в котором будут храниться значения и которое не будет превышать размер этого массива.

Пример реализации хеш-функции на языке Си для строки.

```
unsigned long hash(unsigned char *str){
    unsigned long hash = 5381;
    int c;
    while (c = *str++)
        hash = ((hash << 5) + hash) + c;
    return hash;
}
```

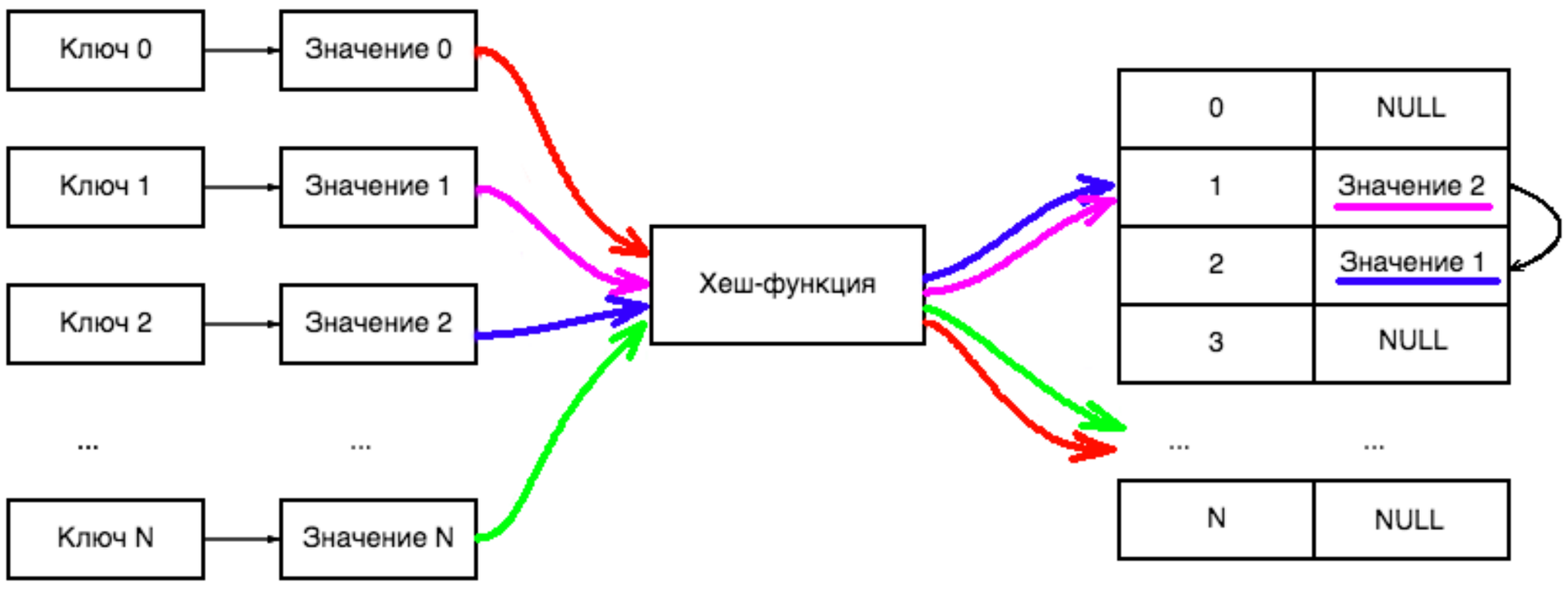
В примере представлена простая хеш-функция, которая называется `djb2`. Она использует начальное значение 5381 и делает операцию побитового сдвига на 5 бит влево, а затем добавляет символ строки. Этот процесс повторяется для каждого символа строки, позволяя сгенерировать уникальный хеш-код для каждой строки.

Пример реализации хеш-функции на языке Си для структуры содержащей целое, строку и число с плавающей точкой.

```
typedef struct {
    int id;
    char name[50];
    float salary;
} Employee;

unsigned int employee_hash(const Employee *e) {
    unsigned int hash = 5381;
    unsigned int id_hash = e->id;
    float salary_hash = e->salary;
    // хеш для id
    hash = ((hash << 5) + hash) + id_hash;

    // хеш для имени
    for (int i = 0; e->name[i] != '\0'; ++i) {
        hash = ((hash << 5) + hash) + e->name[i];
    }

    // хеш для salary
    unsigned char *salary_bytes = (unsigned char *)&salary_hash;
    for (int i = 0; i < sizeof(float); ++i) {
        hash = ((hash << 5) + hash) + salary_bytes[i];
    }

    return hash;
}
```

Если несколько значений, после преобразования с помощью хеш-функции получают одинаковые индексы, такое явление называется коллизией (столкновением). Существует два основных вида хеш-таблиц. По сути они отличаются поведением в случае возникновения коллизий в хеш-функции - хеш таблицы с цепочками и открытой адресацией.

Идея метода открытой адресации состоит в том, что все элементы будут хранятся в хеш-таблице без дополнительной структуры данных, ячейка хранит либо значение элемента, либо NULL. Для того, чтобы вставить данные в ячейку, можно двигаться по списку пока не будет найдена пустая ячейка. Такой метод решения коллизий называется линейное хеширование.



Линейное хеширование реализуется просто, однако с ним связаны существенные сложности – нетривиальное удаление элемента таблицы и кластеризация. Кластеризация это явление создания длинных последовательностей занятых ячеек, которое увеличивает среднее время поиска в таблице. Для снижения эффекта кластеризации используется другая стратегия разрешения коллизий – двойное хеширование. Основная идея заключается в том, что для определения шага смещения исследований при коллизии в ячейке используется другая хеш-функция, вместо линейного смещения на одну позицию.

Хеш-таблицы с цепочками реализуют дополнительную структуру данных, которая позволяет сохранить значение, для ключа которого возникла коллизия в хеш-функции. Это например может быть список, дерево и т.д.