Лекция 8.
Хеш-таблицы

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2021 г.

АТД «Словарь»

- **Словарь** (ассоциативный массив, associative array, map, dictionary) — это структура данных (контейнер) для хранения пар вида «ключ» — «значение» (key — value)

- Реализации словарей отличаются вычислительной сложностью операций добавления (Add), поиска (Lookup) и удаления (Delete) элементов

- Наибольшее распространение получили следующие реализации:
  - Деревья поиска (search trees)
  - **Хеш-таблицы** (hash tables)
  - Списки с пропусками (skip lists)
  - Связные списки, массивы

Хеш-таблицы (hash tables)

- **Хеш-таблица** (hash table) — это структура данных для хранения пар «ключ» — «значение»
- Доступ к элементам осуществляется по ключу (key)
- Ключи могут быть строками, числами, указателями, ...
- Хеш-таблицы позволяют в среднем за время ***O*(1)** выполнять добавление, поиск и удаление узлов

Основная идея

- Чем хороши статические массивы (`int a[400]`)?
- Быстрый доступ ***O*(1)** к элементу массива по его ключу (индексу): `a[241] = 31`
- **Ограничение** — ключи (индексы) могут быть только целыми неотрицательными числами

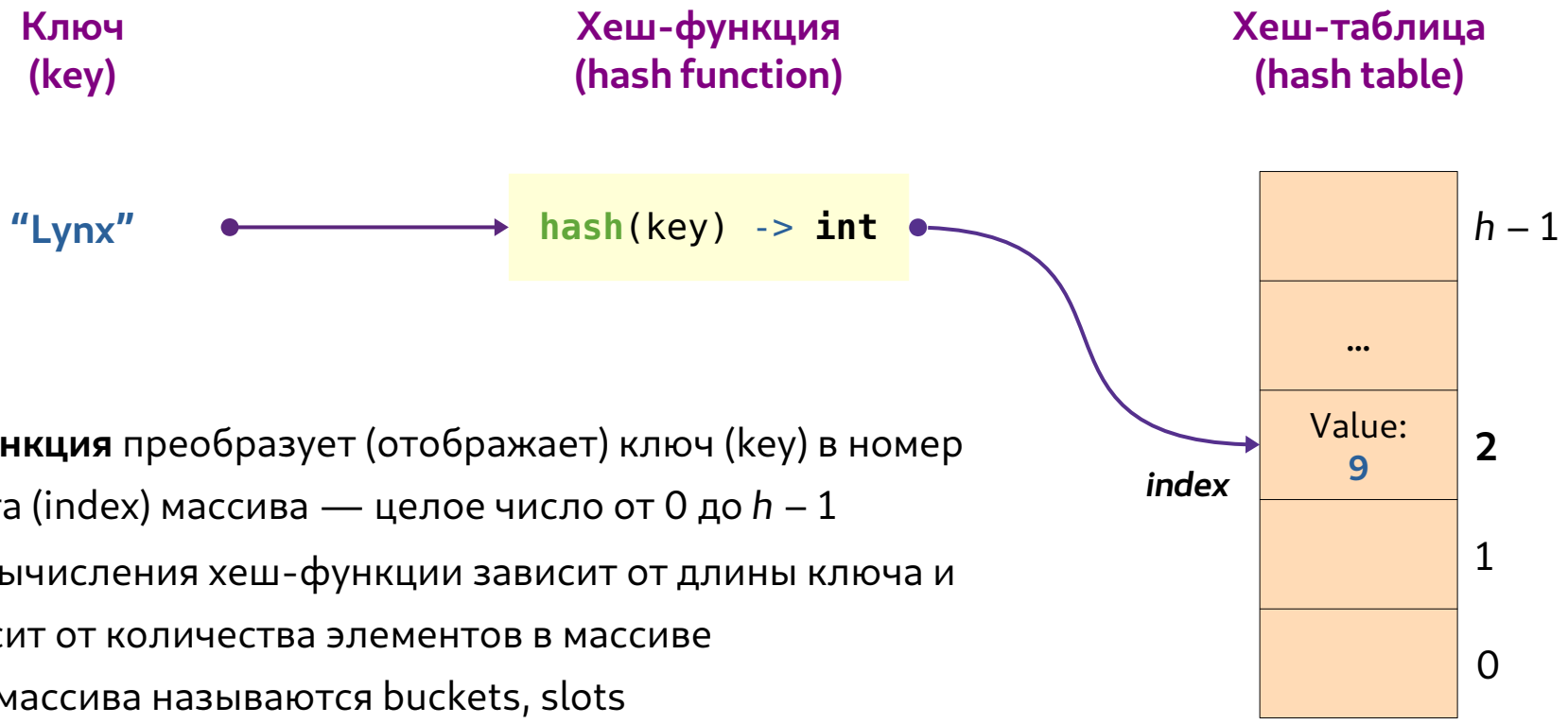
Основная идея

- Чем хороши статические массивы (`int a[400]`)?
- Быстрый доступ ***O*(1)** к элементу массива по его ключу (индексу): `a[241] = 31`
- **Ограничение** — ключи (индексы) могут быть только целыми неотрицательными числами

- **Можно ли как-то использовать типы float, double, string (char[]) в качестве индексов в массиве?**
- **Пример:** массив профилей пользователей Reddit: словарь, где ключ — имя пользователя, а значение — профиль с данными пользователя

- Массив структур:
  `struct reddit_user users[MAX_USERS];`

Хеш-таблицы (hash tables)



- **Хеш-функция** преобразует (отображает) ключ (key) в номер элемента (index) массива — целое число от 0 до *h* – 1
- Время вычисления хеш-функции зависит от длины ключа и не зависит от количества элементов в массиве
- Ячейки массива называются buckets, slots

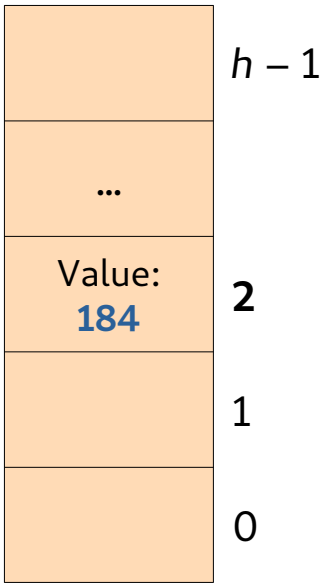
Хеш-таблицы (hash tables)

- На практике обычно известна информация о диапазоне значений ключей
- На основе этого выбирается размер h таблицы и выбирается хеш-функция
- **Коэффициент α заполнения хеш-таблицы** (load factor, fill factor) — отношение числа n хранимых в хеш-таблице элементов к размеру h массива (среднее число элементов на одну ячейку)

$$\alpha = \frac{n}{h}$$

- Пример: h = 128, в хеш-таблицу добавили 50 элементов, тогда α = 50 / 128 ≈ 0.39
- От этого коэффициента зависит среднее время операций добавления, поиска и удаления элементов

Хеш-таблица (hash table)



Хеш-функции (hash function)

- **Хеш-функция** (hash function) — это функция, преобразующая значение ключа (например, строки, числа, файла) в целое число
- Значение, возвращаемое хеш-функцией называется хеш-кодом (hash code), контрольной суммой (hash sum) или просто хешем (hash)

```
/* Хеш-функция для строк [Керниган-Ричи, “Практика программирования”] */
unsigned int KRHash(char *s)
{
    unsigned int h = 0, hash_mul = 31;

    while (*s)
        h = h * hash_mul + (unsigned int)*s++;
    return h % HASH_SIZE;
}
```

$T_{Hash} = O(|s|)$

Хеш-функции (hash function)

- **Хеш-функция** (hash function) — это функция, преобразующая значение ключа (например, строки, числа, файла) в целое число
- Значение, возвращаемое хеш-функцией называется хеш-кодом (hash code), контрольной суммой (hash sum) или просто хешем (hash)

```
#define HASH_SIZE 128     /* Размер хеш-таблицы */

int main()
{
    unsigned int h = KRHash("ocelot");

    printf("Hash sum: %d\n", h);
    return 0;
}
```

o	c	e	l	o	t
111	99	101	108	111	116

Хеш-функции (hash function)

```c
#define HASH_SIZE 128

unsigned int KRHash(char *s)
{
    unsigned int h = 0, hash_mul = 31;

    while (*s)
        h = h * hash_mul + (unsigned int)*s++;
    return h % HASH_SIZE;
}
```

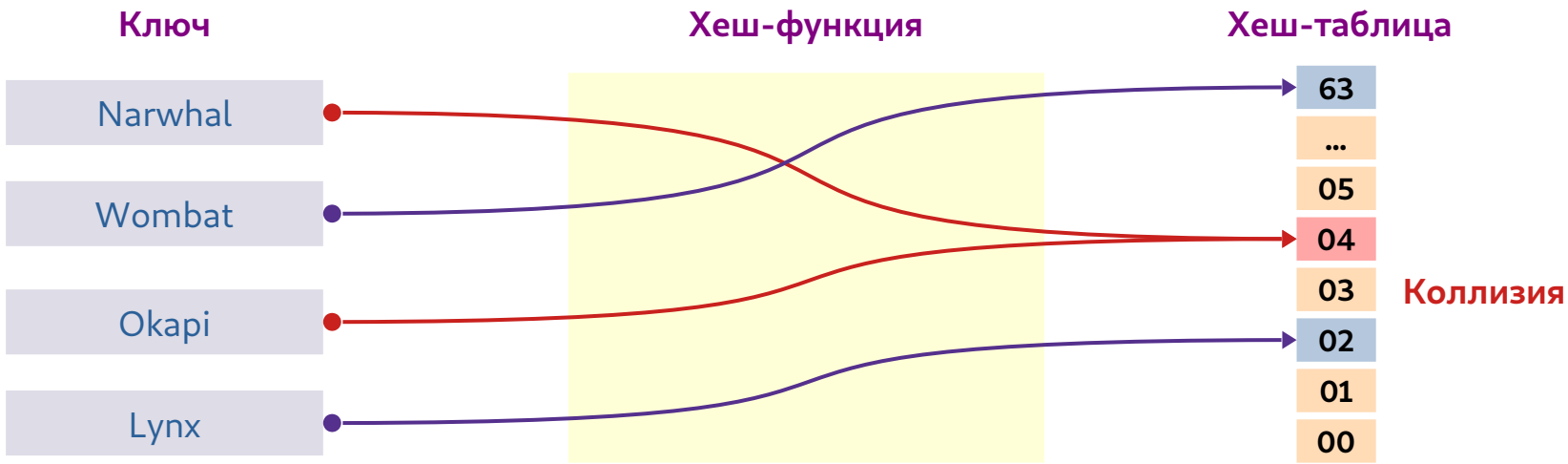
```
h = 0 * hash_mul + 111
h = 111 * hash_mul + 99
h = 3540 * hash_mul + 101
h = 109841 * hash_mul + 108
h = 3405179 * hash_mul + 111
h = 105560660 * hash_mul + 116
return 3272380576 % HASH_SIZE    // KRHash("ocelot") = 32
```

o	c	e	l	o	t
111	99	101	108	111	116

Коллизии (collisions)

- **Коллизия** (collision) — это совпадение значений хеш-функции для двух разных ключей

hash(“Narwhal”) = **4** ***hash***(“Okapi”) = **4**

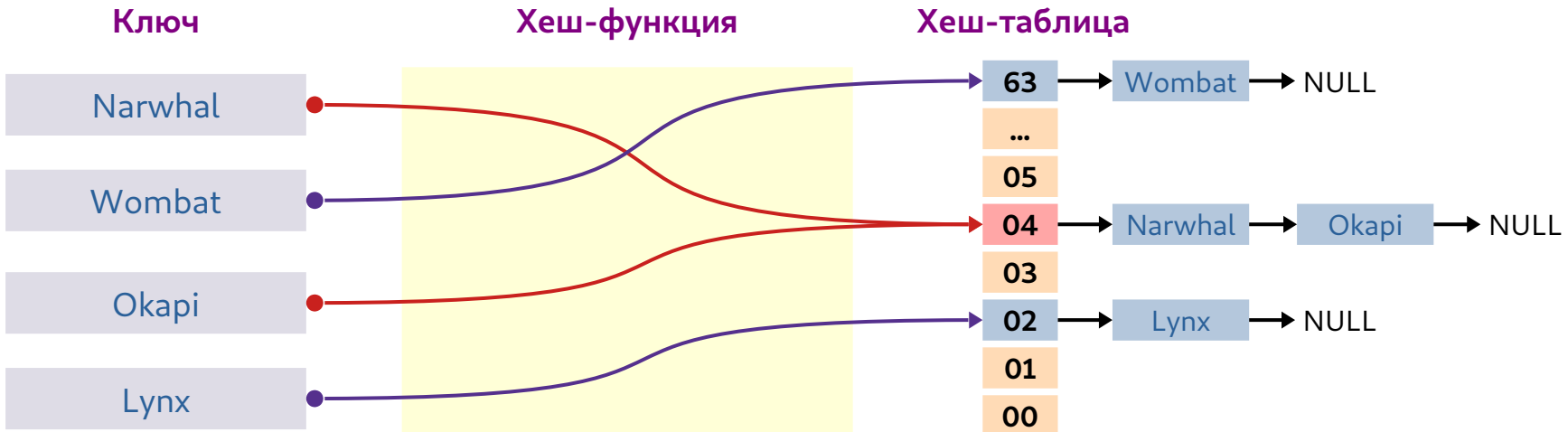


Разрешение коллизий (collision resolution)

Метод цепочек (chaining) — закрытая адресация

Элементы с одинаковым значением хеш-функции объединяются в связный список. Указатель на список хранится в соответствующей ячейке хеш-таблицы

- При коллизии элемент добавляется в начало списка
- Поиск и удаление в худшем случае требуют просмотра всего списка



Разрешение коллизий (collision resolution)

Открытая адресация (open addressing)

- В каждой ячейке хеш-таблицы хранится не указатель на связный список, а один элемент (ключ, значение)
- Если ячейка с индексом ***hash***(*key*) занята, то осуществляется поиск свободной ячейки в следующих позициях таблицы
- **Линейное хеширование** (linear probing) — проверяются позиции:

$$\textbf{hash}(key) + 1, \textbf{hash}(key) + 2, \ldots, \textbf{hash}(key + i) \textbf{ mod } h, \ldots$$

- Если свободных ячеек нет, таблица заполнена

Пример:

- ***hash***(D) = 2, но ячейка с индексом 2 занята
- Обходим ячейки: 3 — занята, 4 — свободна

Хеш	Элемент
0	B
1	
2	A
3	C
4	D
5	

Требования к хеш-функциям

- **Быстрое вычисление хеш-кода** по значению ключа

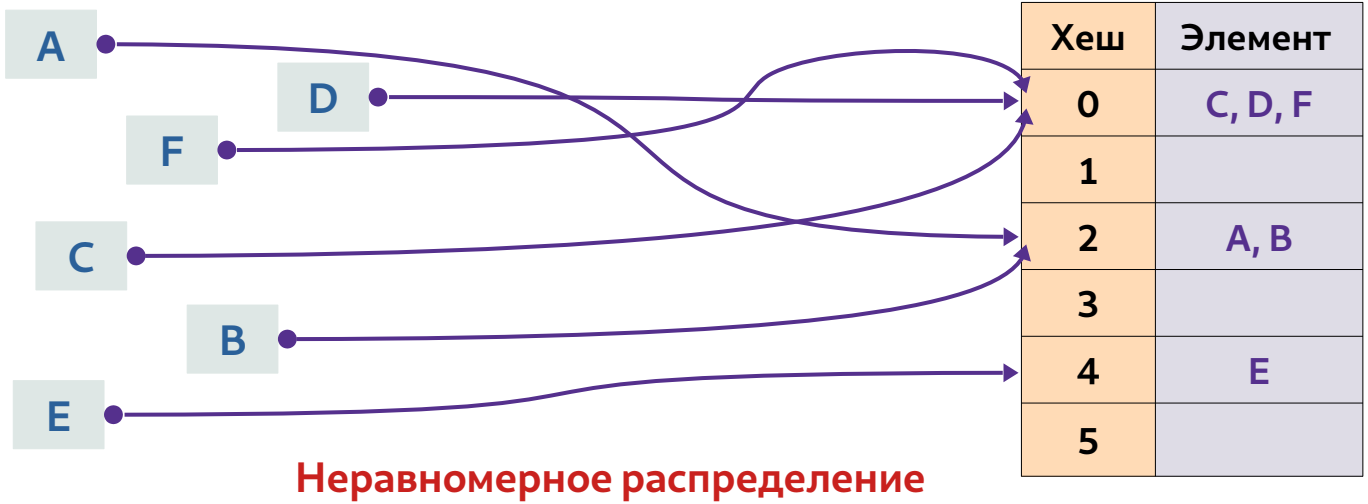
  Сложность вычисления хеш-кода не должна зависеть от количества *n* элементов в таблице
- **Детерминированность** — для заданного значения ключа хеш-функция всегда должна возвращать одно и то же значение

```
unsigned int KRHash(char *s)
{
    unsigned int h = 0, hash_mul = 31;

    while (*s)
        h = h * hash_mul + (unsigned int)*s++;
    return h % HASH_SIZE;
}
```

Требования к хеш-функциям

- **Равномерность** (uniform distribution) — хеш-функция должна равномерно заполнять индексы массива возвращаемыми номерами
- Желательно, чтобы все хеш-коды формировались с одинаковой равномерной распределённой вероятностью



Требования к хеш-функциям

- **Равномерность** (uniform distribution) — хеш-функция должна равномерно заполнять индексы массива возвращаемыми номерами
- Желательно, чтобы все хеш-коды формировались с одинаковой равномерной распределённой вероятностью

Эффективность хеш-таблиц

- Хеш-таблица требует предварительной инициализации ячеек значениями NULL — трудоёмкость O(h)
- Ключ — это строка из m символов

Операция	Вычислительная сложность в среднем случае	Вычислительная сложность в худшем случае
Add(*key*, *value*)	$O(m)$	$O(m)$
Lookup(*key*)	$O(m + mn / h)$	$O(m + nm)$
Delete(*key*)	$O(m + mn / h)$	$O(m + nm)$
Min()	$O(m(n + h))$	$O(m(n + h))$
Max()	$O(m(n + h))$	$O(m(n + h))$

Пример хеш-функции для строк (ELF)

```
unsigned int ELFHash(char *s)
{
    unsigned int h = 0, g;

    while (*s) {
        h = (h << 4) + (unsigned int)*s++;
        g = h & 0xF0000000L;
        if (g)
            h ^= g >> 24;
        h &= ~g;
    }
    return h % HASH_SIZE;
}
```

- Применяется в файлах формата ELF в UNIX-подобных операционных системах
- В функции используются только поразрядные операции (для эффективности)

Пример хеш-функции для строк (Jenkins one-at-a-time hash)

```
unsigned int JenkinsHash(char *s)
{
    unsigned int h = 0;

    while (*s) {
        h += (unsigned int)*s++;
        h += (h << 10);
        h ^= (h >> 6);
    }
    h += (h << 3);
    h ^= (h >> 11);
    h += (h << 15);
    return h % HASH_SIZE;
}
```

- Функция one-at-a-time из семейства хеш-функций Дженкинса (Bob Jenkins)
- Основана на *лавинном эффекте* (avalanche effect)

Пример хеш-функции для чисел

- **Ключи** — размер файла (int)
- **Значение**, хранимое в словаре — название файла
- Требуется разработать хеш-функцию

hash(*filesize*) → [0..1023]

```
function hash(int filesize)
    return filesize mod 1024
end function
```

Пример хеш-функции для строк

$$\text{hash}(s) = \sum_{i=0}^{L-1} h^{L-(i+1)} \cdot s[i] =$$
$$= h^{L-1} \cdot s[0] + h^{L-2} \cdot s[1] + \ldots + h \cdot s[L-2] + s[L-1],$$

где s — ключ (строка), L — длина строки, $s[i]$ — код символа i

Хеш-таблицы (hash table)

- Длину *h* хеш-таблицы выбирают как простое число
- Для такой таблицы модульная хеш-функция даёт равномерное распределение значений ключей

$$\mathbf{hash}(key) = index \ \mathbf{mod}\ h$$

Хеш-таблицы vs. бинарные деревья поиска

- Эффективность реализации словаря хеш-таблицей (метод цепочек) и бинарным деревом поиска
- Ключ — это строка из m символов
- Оценка сложности для **худшего случая** (worst case):

Операция	Хеш-таблица (неупорядоченный словарь)	Бинарное дерево поиска (упорядоченный словарь)
Add(*key*, *value*)	$O(m)$	$O(nm)$
Lookup(*key*)	$O(m + nm)$	$O(nm)$
Delete(*key*)	$O(m + nm)$	$O(nm)$
Min()	$O(m(n + h))$	$O(n)$
Max()	$O(m(n + h))$	$O(n)$

Хеш-таблицы vs. бинарные деревья поиска

- Эффективность реализации словаря хеш-таблицей (метод цепочек) и бинарным деревом поиска
- Ключ — это строка из m символов
- Оценка сложности для **среднего случая** (average case):

Операция	Хеш-таблица (неупорядоченный словарь)	Бинарное дерево поиска (упорядоченный словарь)
Add(*key*, *value*)	$O(m)$	$O(m \log n)$
Lookup(*key*)	$O(m + mn / h)$	$O(m \log n)$
Delete(*key*)	$O(m + mn / h)$	$O(m \log n)$
Min()	$O(m(n + h))$	$O(\log n)$
Max()	$O(m(n + h))$	$O(\log n)$

Реализация хеш-таблицы

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define HASHTAB_SIZE 5051

struct listnode {
    char *key;
    int value;

    struct listnode *next;
};

struct listnode *hashtab[HASHTAB_SIZE];
```

Хеш-функция

```
unsigned int hashtab_hash(char *key)
{
    unsigned int h = 0, hash_mul = 31;

    while (*key)
        h = h * hash_mul + (unsigned int)*key++;
    return h % HASHTAB_SIZE;
}
```

$T_{Hash} = O(|key|)$

Инициализация хеш-таблицы

```
void hashtab_init(struct listnode **hashtab)
{
    int i;

    for (i = 0; i < HASHTAB_SIZE; i++)
        hashtab[i] = NULL;
}
```

$T_{Init} = O(h)$

Добавление элемента в хеш-таблицу

```
void hashtab_add(struct listnode **hashtab, char *key, int value)
{
    struct listnode *node;

    int index = hashtab_hash(key);
    node = malloc(sizeof(*node));
    if (node != NULL) {
        node->key = key;
        node->value = value;
        node->next = hashtab[index];
        hashtab[index] = node;
    }
}
```

$$T_{Add} = T_{Hash} + O(1) = O(|key|)$$

Поиск элемента в хеш-таблице

```
struct listnode *hashtab_lookup(struct listnode **hashtab, char *key)
{
    struct listnode *node;

    int index = hashtab_hash(key);
    for (node = hashtab[index]; node != NULL; node = node->next) {
        if (0 == strcmp(node->key, key))
            return node;
    }
    return NULL;
}
```

$$T_{Lookup} = O(|key| + |key| \cdot n)$$

Пример работы с хеш-таблицей

```
int main()
{
    struct listnode *node;

    hashtab_init(hashtab);
    hashtab_add(hashtab, “Ocelot”, 17);
    hashtab_add(hashtab, “Flamingo”, 4);
    hashtab_add(hashtab, “Fox”, 14);

    node = hashtab_lookup(hashtab, “Flamingo”);
    if (node != NULL)
        printf(“Node: %s, %d\n”, node->key, node->value);
    return 0;
}
```

Удаление элемента из хеш-таблицы

```
void hashtab_delete(struct listnode **hashtab, char *key)
{
    struct listnode *node, *prev = NULL;
    int index = hashtab_hash(key);
    for (node = hashtab[index]; node != NULL; node = node->next) {
        if (0 == strcmp(node->key, key)) {
            if (prev == NULL)
                hashtab[index] = node->next;
            else
                prev->next = node->next;
            free(node);
            return;
        }
        prev = node;
    }
}
```

$$T_{Delete} = O(|key| + |key| \cdot n)$$

Удаление элемента из хеш-таблицы

```
int main()
{
    struct listnode *node;

    /* ... */

    hashtab_delete(hashtab, “Ocelot”);
    node = hashtab_lookup(hashtab, “Ocelot”);
    if (node != NULL)
        printf(“Node: %s, %d\n”, node->key, node->value);
    else
        printf(“Node not found\n”);
    return 0;
}
```

Домашнее чтение

- **[DSABook] Глава 13. «Хеш-таблицы»**
- Прочитать в [Sedgewick, С. 575] о хеш-функциях для вещественных чисел
- Прочитать в «Практике программирования» [Kernighan] раздел о хеш-таблицах
- [CLRS, С. 282] Глава 11. «Хеш-таблицы»

ご清聴ありがとうございました!

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2021 г.