



Алгоритмы и структуры данных

Лекция 14. Хеш-таблицы

Антон Штанюк (к.т.н, доцент)

19 мая 2022 г.

Нижегородский государственный технический университет им. Р.Е. Алексеева
Институт радиоэлектроники информационных технологий
Кафедра "Компьютерные технологии в проектировании и производстве"

Основные определения

Хеш-функция

Разрешение коллизий

Программная реализация

Список литературы

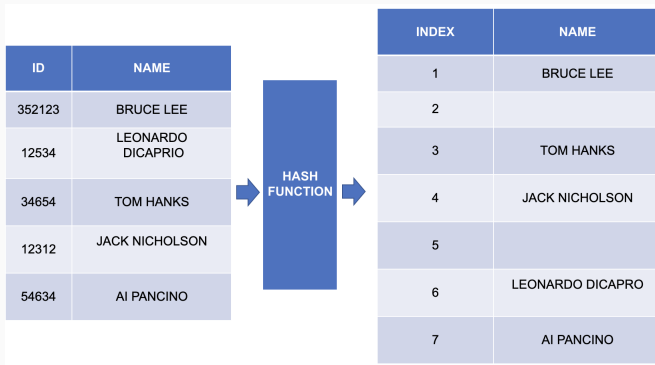
Основные определения

Хэш-таблица - это структура данных для хранения пар (ключ, значение).
Доступ к элементам осуществляется по ключу.

Поддержка трех операций:

1. добавление пары
2. удаление пары
3. поиск

Понятие хеш-таблицы

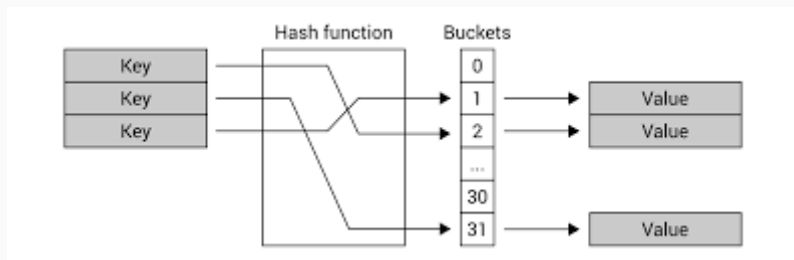


Ключи могут быть строками, числами, указателями (и др.)

Хеш-таблицы позволяют в среднем за время (1) выполнять добавление, поиски и удаление элементов

Имеется две основных архитектурных разновидности:

1. с цепочками
2. с открытой адресацией

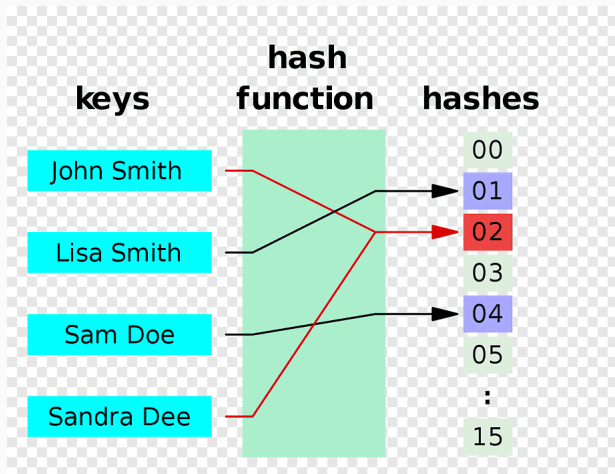


При обращении к таблице по ключу **key** выполняется вычисление **хеш-функции** $H()$

$$i = H(key)$$

i - играет роль индекса в обычном массиве

Коллизия - когда разным ключам соответствуют одинаковые i , то есть $H(x) = H(y)$



Хеш-функция

Хеш-функция (Hash function) – это функция преобразующая значения ключа (например: строки, числа, файла) в целое число

Хорошая хеш-функция должна удовлетворять следующим условиям:

- быстро вычисляться
- иметь минимальное число коллизий
- равномерно заполнять диапазон значений на выходе

Более полный набор свойств:

- **Необратимость.** Из хеша нельзя получить исходные данные даже теоретически. Слишком много информации отбрасывается в процессе; это не зашифровка информации.
- **Детерминированность.** Если подать хеш-функции одинаковые данные, то и хеш у них будет одинаковым. Именно это свойство позволяет использовать хеши для проверки подлинности информации.
- **Уникальность.** Идеальная хеш-функция выдает стопроцентно уникальный результат для каждого возможного набора данных. В реальности такое невозможно, и иногда случаются коллизии — одинаковые хеши для разных сведений. Но существующие хеш-функции достаточно сложны, поэтому вероятность коллизии сводится к минимуму.

- **Разнообразие.** Даже если два набора информации различаются одним-двумя символами, их хеши будут кардинально разными. У них не будет общих блоков, по ним невозможно будет понять, что исходные данные схожи.
- **Высокая скорость генерации.** Это в целом свойство любых хешей: в отличие от зашифрованных версий файлов, они генерируются быстро, даже если входной массив данных большой.

Рассмотрим общий вид простейшей ХФ:

$$H(key) = key \bmod M$$

Для M выбирают простое число, далекое от степени двойки.

следует выбирать M , опираясь на значения входящих ключей. Так, например если все или большинство $key = 10^m$ (m – натуральное число), то неудачным выбором будет $M = 10 * m$ и $M = 10^m$.

Пример плохой ХФ:

Плохая ХФ

$H(\text{key}) = [\text{последние 3 цифры key}] = \text{key} \% 1000$

Функция порождает много коллизий, если ключи - цены 000,500,999

Общий вид такой функции:

Мультипликативная ХФ

$$h(key) = [M * (key * A)]$$

Здесь A – рациональное число, по модулю меньше единицы ($0 < A < 1$), а key и M обозначают то же, что и в предыдущем методе: ключ и размер хеш-таблицы. Также правая часть функции содержит три пары скобок:

() – скобки приоритета;

[] – скобки взятия целой части;

{ } – скобки взятия дробной части.

Аргумент хеш-функции key ($k \geq 0$) в результате даст значение хеш-кода $h(key) = x$, лежащие в диапазоне $0 \dots M - 1$. Для работы с отрицательными числами можно число x взять по модулю.

От выбора A и M зависит то, насколько оптимальным окажется хеширование умножением на определенной последовательности. Не имея сведений о входящих ключах, в качестве M следует выбрать одну из степеней двойки, т. к. умножение на 2^m равносильно сдвигу на m разрядов, что компьютером производится быстрее. Неплохим значением для A (в общем случае) будет $(\sqrt{5} - 1)/2 \approx 0,6180339887$. Оно основано на свойствах золотого сечения.

$$h(25) = [13 * (\{25 * 0,618033\})] = [13 * \{15,450825\}] = [13 * 0,450825] = [5,860725] = 5$$

$$h(44) = [13 * (\{44 * 0,618033\})] = [13 * \{27,193452\}] = [13 * 0,193452] = [2,514876] = 2$$

$$h(97) = [13 * (\{97 * 0,618033\})] = [13 * \{59,949201\}] = [13 * 0,949201] = [12,339613] = 12$$

Для ХФ могут использоваться функции вычисления контрольных сумм:

- **CRC** - циклически избыточный код
- **MD5** - 128 битный алгоритм
- **SHA** - криптографические ХФ

Один из лучших способов определить хэш-функцию от строки S следующий:

Вычисление ХФ для строки

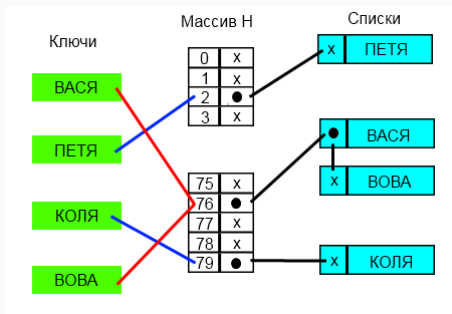
$$h(S) = S[0] + S[1] * P + S[2] * P^2 + S[3] * P^3 + \dots + S[N] * P^N$$

Хеш-функция строк

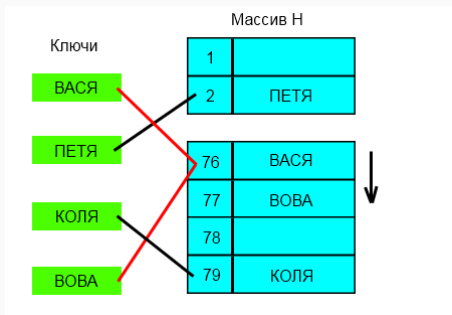
```
#define HASH_MUL 31
#define HASH_SIZE 128
unsigned int hash(char *s)
{
    unsigned int h = 0;
    char *p;
    for (p = s; *p != '\0'; p++) {
        h = h * HASH_MUL + (unsigned int)*p;
    }
    return h % HASH_SIZE;
}
```

Разрешение коллизий

Метод цепочек:



Открытая адресация:



Программная реализация

```
#ifndef _HASH_TABLE_  
#define _HASH_TABLE_  
  
#include <iostream>  
#include <vector>  
#include <list>  
#include <string>  
using namespace std;  
  
typedef int TKey;  
typedef string TValue;  
typedef list< pair< TKey,TValue> > slist;  
...
```

```
class Hash_Table {
private:
    unsigned static const defaultCapacity=17;
    vector< slist> table;
    unsigned capacity;
    unsigned count;
public:
    Hash_Table();
    Hash_Table(unsigned divisor);
    ~Hash_Table();
    unsigned HashIntKey(int key);
    unsigned DataToKey(TValue Data) ;
    TValue Search(TKey key) ;
    void Insert(TKey key_val, TValue el_value);
    void Output();
    void Delete(TValue element);
    TValue operator[](int key);
};
```

```
unsigned Hash_Table::HashIntKey(int key) {  
    return key%capacity;  
}
```

```
Hash_Table::Hash_Table() {  
    count=0;  
    capacity=defaultCapacity;  
    table.resize(defaultCapacity, slist());  
}
```

```
Hash_Table::Hash_Table(unsigned divisor){  
    count=0;  
    capacity=divisor;  
    table.resize(capacity, slist());  
}
```

```
Hash_Table::~~Hash_Table() {  
    table.clear();  
}
```

```
TValue Hash_Table::Search(TKey key) {
    unsigned pos=HashIntKey(key);
    slist lst = table.at(pos);
    if(lst.size()< 1) //if list hasnt elements
        return (TValue)0;
    for(slist::iterator it = lst.begin(); it != lst.end(); ++it) {
        if(it->first == key) {
            return (it->second);
        }
    }
    Insert(key,TValue()); //insert element with default data
    return operator[](key);
}
```






```
void Hash_Table::Insert(TKey key_val, TValue el_value) {
    unsigned pos=HashIntKey((int)key_val);
    pair< TKey,TValue> toInsert(key_val, el_value);
    slist& lst = table.at(pos);
    if(!lst.empty()) {
        for(slist::iterator it = lst.begin(); it != lst.end(); ++it) {
            if(it->first == key_val) {
                lst.push_front(toInsert);
                return;
            }
        }
    }
    table[pos].push_front(toInsert);
    count++;
}
```




```
void Hash_Table::Output() const {
    int NumOfCollision = 0;
    cout<< "Hash_table_capacity";
    NumOfCollision += table[i].size() - 1;
    for(slist::iterator it = table[i].begin(); it != table[i].end();it++) {
        cout<< "HashTable["<< i<< "]:" << (it->second)<< endl;
    }
    cout<< "N.Hash_collisions:" << NumOfCollision<< endl;
}
```

```
TValue Hash_Table::operator[](int key) {  
    unsigned pos=HashIntKey(key);  
    slist& lst = table.at(pos);  
  
    if(lst.size()< 1)  
        return (TValue)0;  
    for(slist::iterator it = lst.begin(); it != lst.end(); ++it) {  
        if(it->first == key){  
            return (it->second);  
        }  
    }  
    Insert(key,TValue());  
    return operator[](key);  
}  
#endif
```


Список литературы






-  SkillFactory.Блог
Хеширование
<https://blog.skillfactory.ru/glossary/heshirovanie/>
-  Кормен Т., Лейзерсон Ч., Ривест Р.
Алгоритмы: построение и анализ
МЦНМО, Москва, 2000
-  Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы:
построение и анализ.
2-е изд. — М.: «Вильямс», 2006
-  Википедия
Алгоритм
<http://ru.wikipedia.org/wiki/Алгоритм>
-  Википедия
Список алгоритмов
http://ru.wikipedia.org/wiki/Список_алгоритмов

 Традиция
Задача коммивояжёра
<http://tradio.ru/wiki/Задача>

 Википедия
NP-полная задача
<http://ru.wikipedia.org/wiki/NP-полная>

 Серджвик Р.
Фундаментальные алгоритмы на C++. Части 1-4
Diasoft, 2001

 Седжвик Р.
Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск
СПб.: ДиаСофтЮП, 2003

-  Седжвик Р.
Фундаментальные алгоритмы на С. Алгоритмы на графах
СПб.: ДиаСофтЮП, 2003
-  Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.
Издательский дом «Вильямс», 2000
-  Кнут Д.
Искусство программирования, том 1. Основные алгоритмы
3-е изд. — М.: «Вильямс», 2006
-  Кнут Д.
Искусство программирования, том 2. Получисленные методы
3-е изд. — М.: «Вильямс», 2007
-  Кнут Д.
Искусство программирования, том 3. Сортировка и поиск
2-е изд. — М.: «Вильямс», 2007



Кнут Д.

Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений

М.: «Вильямс», 2007



Кнут Д.

Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации

М.: «Вильямс», 2007