



ПОИСК

*Если ты не будешь искать –
другие найдут.*

Р. Оппенгеймер

Содержание

2

- Постановка задачи
- Линейный и бинарный поиск в массиве
- Таблицы
 - ▣ Прямая адресация в массиве
 - ▣ Хеширование, методы построения хеш-функций
 - ▣ Методы разрешения коллизий

Задача поиска

3

- *Исходные данные* – последовательность элементов данных $(a_0, a_2, \dots, a_{m-1})$, для которых введено отношение порядка
- *Элемент последовательности*

```
type TElement=record
  key: Integer; { Ключ }
  OtherData: ...; { Другие данные }
end;
```
- *Результат поиска* – номер элемента последовательности с заданным значением ключа или спец. значение, указывающее отсутствие элемента в последовательности (`const NULL=-1`).
- *Упрощение*: элемент \equiv ключ.

```
type Table=array [0..M-1] of Integer;
```
- *Задача*: организовать последовательность таким образом, чтобы поиск, вставка и удаление элементов осуществлялись за наименьшее время.

Линейный поиск

4

□ *Неупорядоченный массив*

□ Поиск элемента

```
for i:=1 to N do
  if A[i]=X then begin
    Result:=i;
    break;
  end;
```

□ Вставка элемента

```
N:=N+1;
A[N]:=X;
```

□ Удаление элемента

```
for i:=d+1 to N do
  A[i-1]:=A[i];
A[N]:=NULL;
N:=N-1;
```

Бинарный поиск

5

□ *Упорядоченный массив*

□ Вставка элемента

```
N:=N+1;  
A[N]:=X;  
{ И далее нужна  
сортировка массива! }
```

□ Удаление элемента

```
for i:=d+1 to N do  
  A[i-1]:=A[i];  
A[N]:=NULL;  
N:=N-1;
```

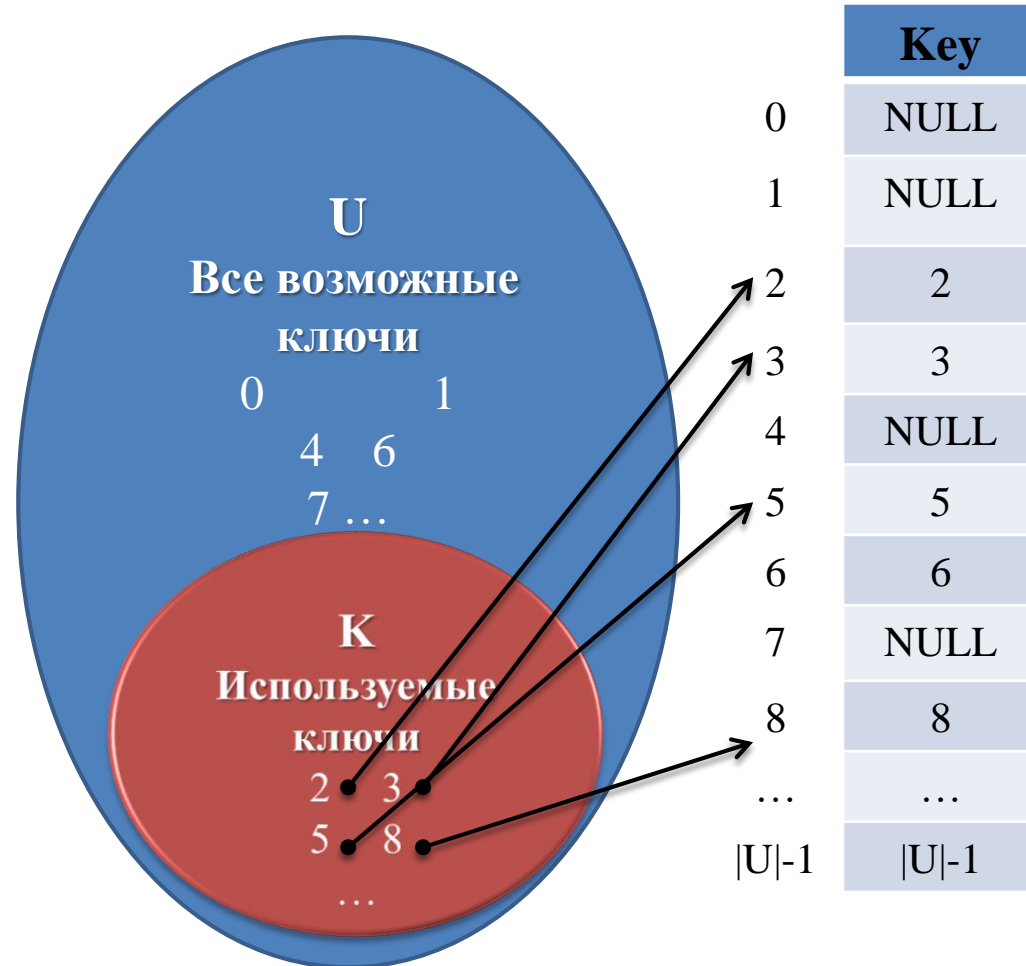
□ Поиск элемента

```
L:=1; R:=N;  
Result:=NULL;  
while L<=R do begin  
  m:=L+(R-L) div 2;  
  if X>A[m] then  
    L:=m+1  
  else  
    if X<A[m] then  
      R:=m-1  
    else begin  
      Result:=m;  
      break;  
    end;  
end;  
end;
```

Прямая адресация

6

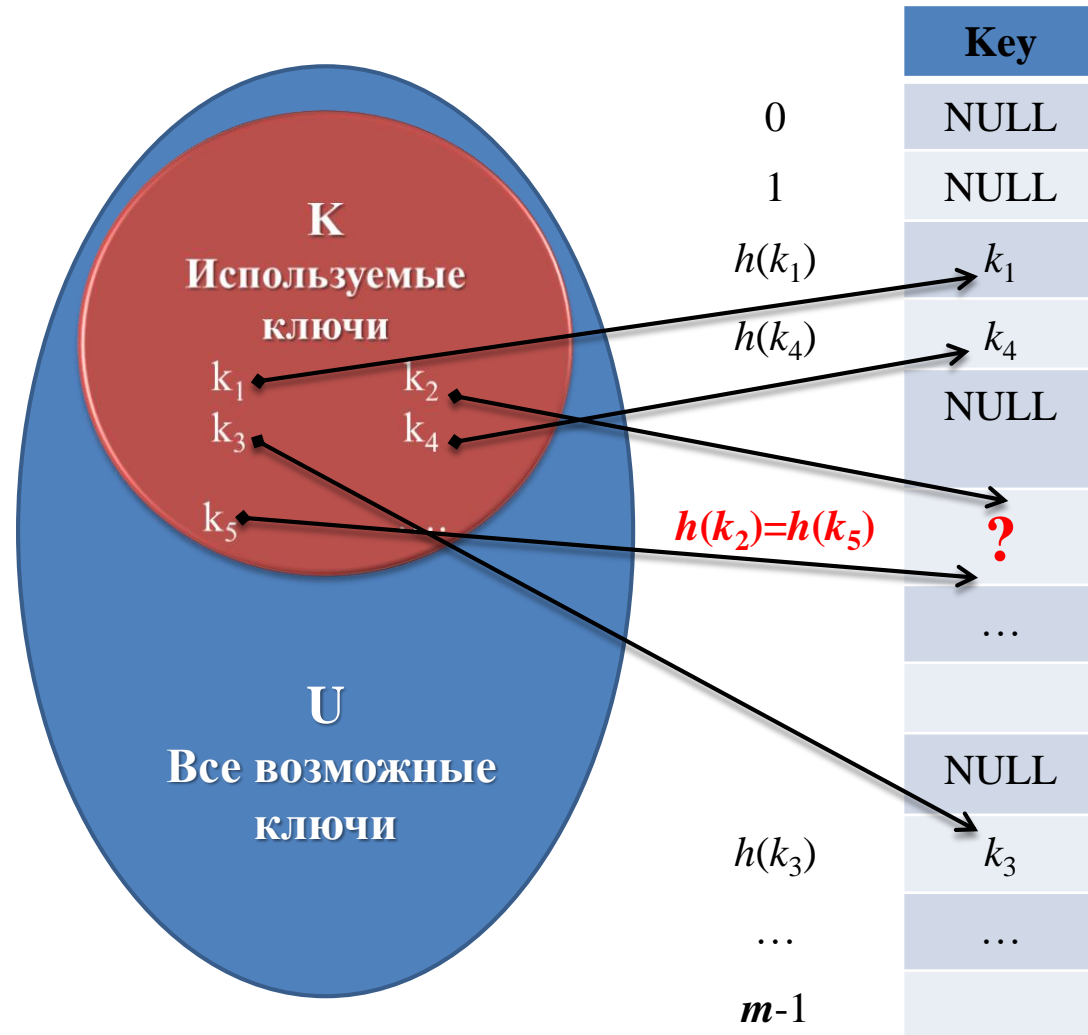
- Поиск элемента
 $Result := Tab[k]$
- Вставка элемента
 $T[k] := k$
- Удаление элемента
 $T[k] := NULL$
- Недостатки
 - Применима, если только количество возможных ключей не очень велико.
 - Неэкономное использование памяти.



Хеш-таблицы

7

- *Хеш-функция*
 $h: U \rightarrow \{0, \dots, m-1\}$.
- *Хеш-значение*
ключа k —
число $h(k)$.
- *Коллизия* —
совпадение хеш-
значений разных
ключей.



Разрешение коллизий: метод цепочек

8

- Элементы множества, которым соответствует одно и то же хеш-значение, представляются в виде связного списка ("цепочки").
 - ▣ Каждый элемент цепочки хранит ключ и указатель на следующий элемент цепочки.
- Операции
 - ▣ Вставка
 - Добавить элемент X в список, голова которого находится в элементе $\text{Tab}[h(X)]$ (в голову, хвост или в порядке возрастания/убывания).
 - ▣ Поиск
 - Найти элемент с ключом k в списке, голова которого в элементе $\text{Tab}[h(k)]$
 - ▣ Удаление
 - Удалить элемент X из списка, голова которого находится в элементе $\text{Tab}[h(X)]$.

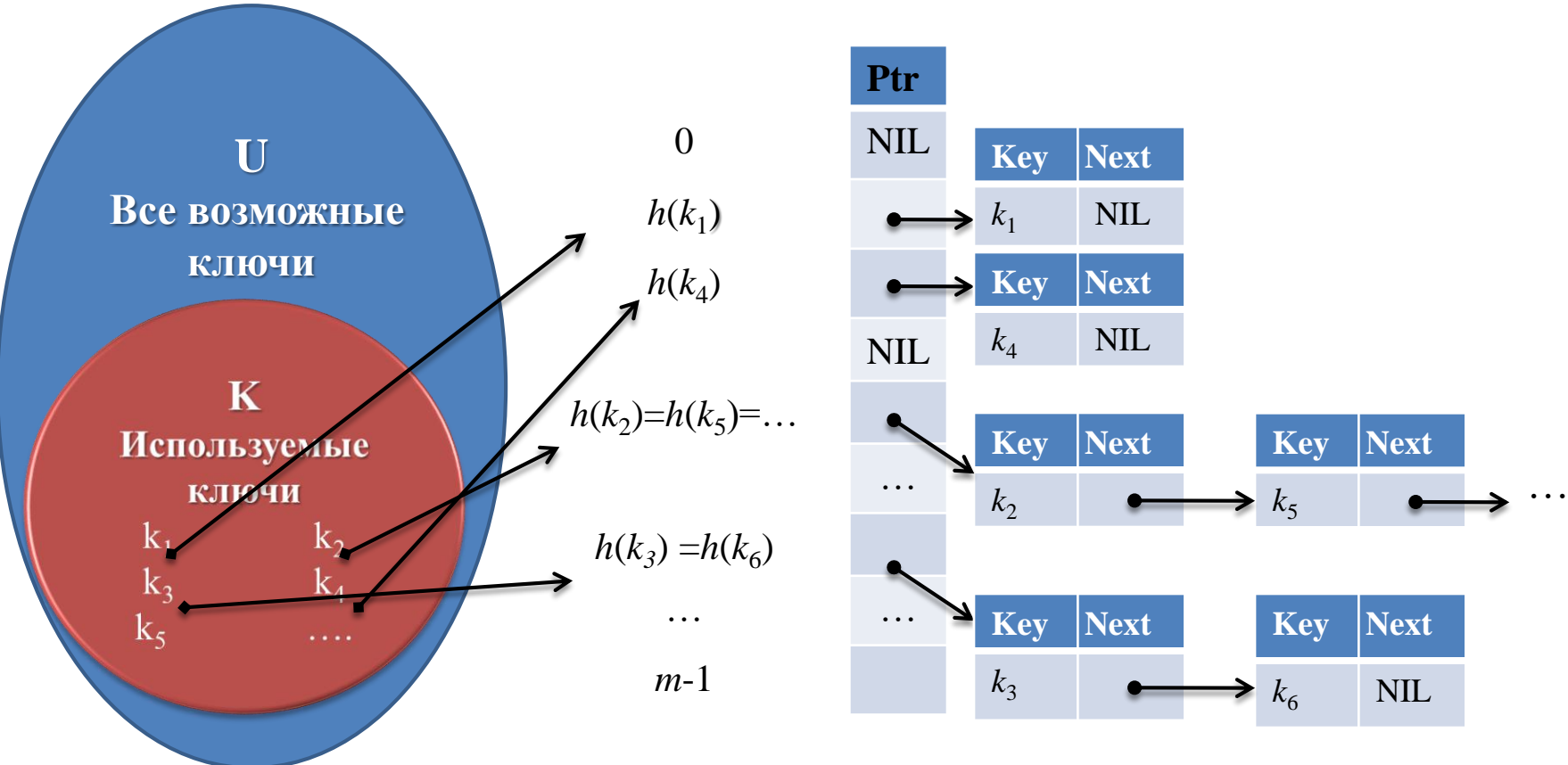
Реализация метода цепочек

9

- Возможные структуры данных, реализующие цепочки
 - Хеш-таблица и связанные списки на основе ссылочных типов
 - Хеш-таблица и связанные списки на основе индексов
 - Таблица элементов и хеш-таблица

Реализация метода цепочек

- Хеш-таблица хранит указатели на цепочки.



Реализация метода цепочек

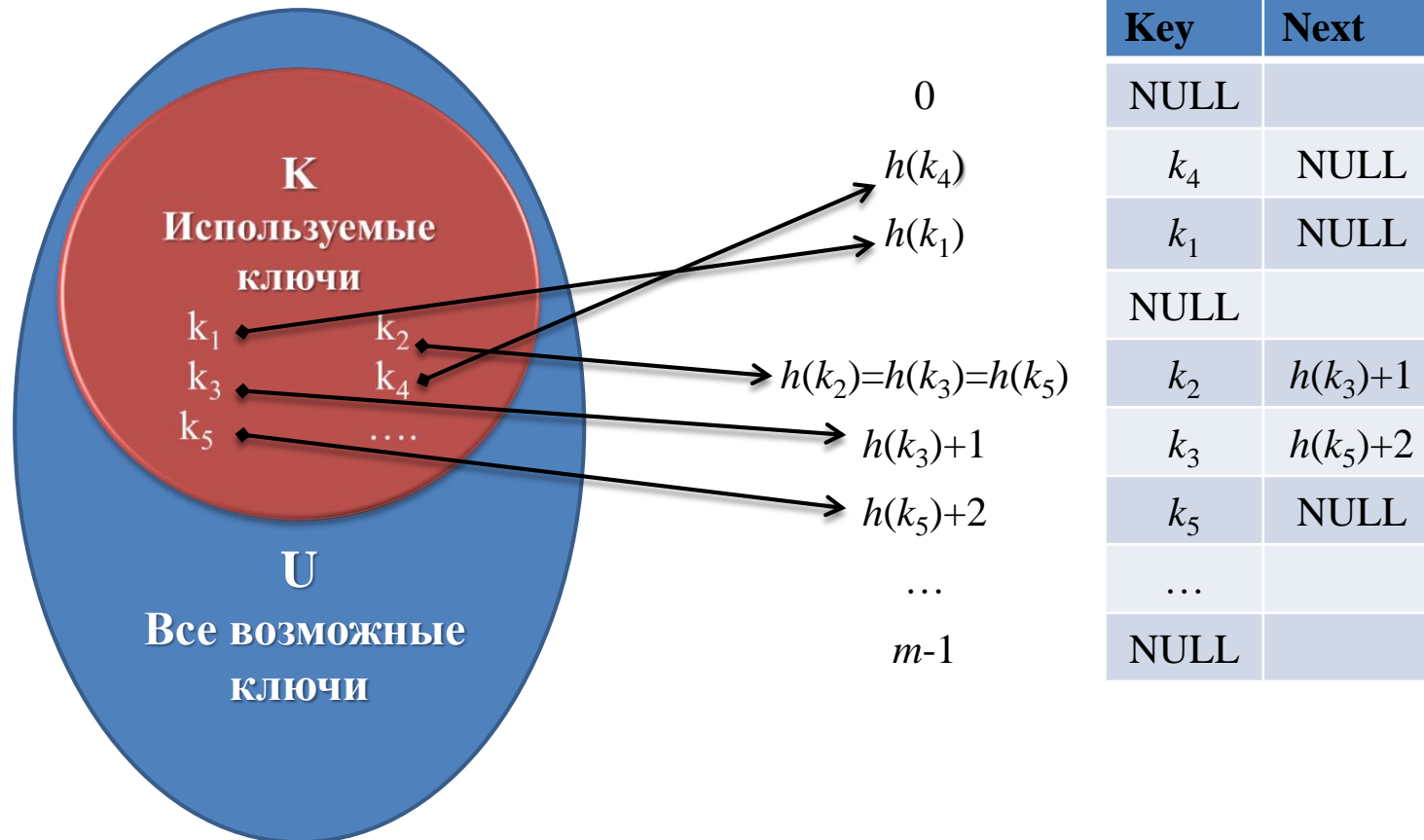
11

```
const
  NULL=-1;
  M=...;
type
  PElement=^TElement;
  TElement=record
    { Info: TInfo; }
    key: Integer;
    Next: PElement;
end;
THashTab=array [0..M-1] of PElement;
```

Реализация метода цепочек

12

- Хеш-таблица хранит ключ и индекс следующего элемента цепочки.



Реализация метода цепочек

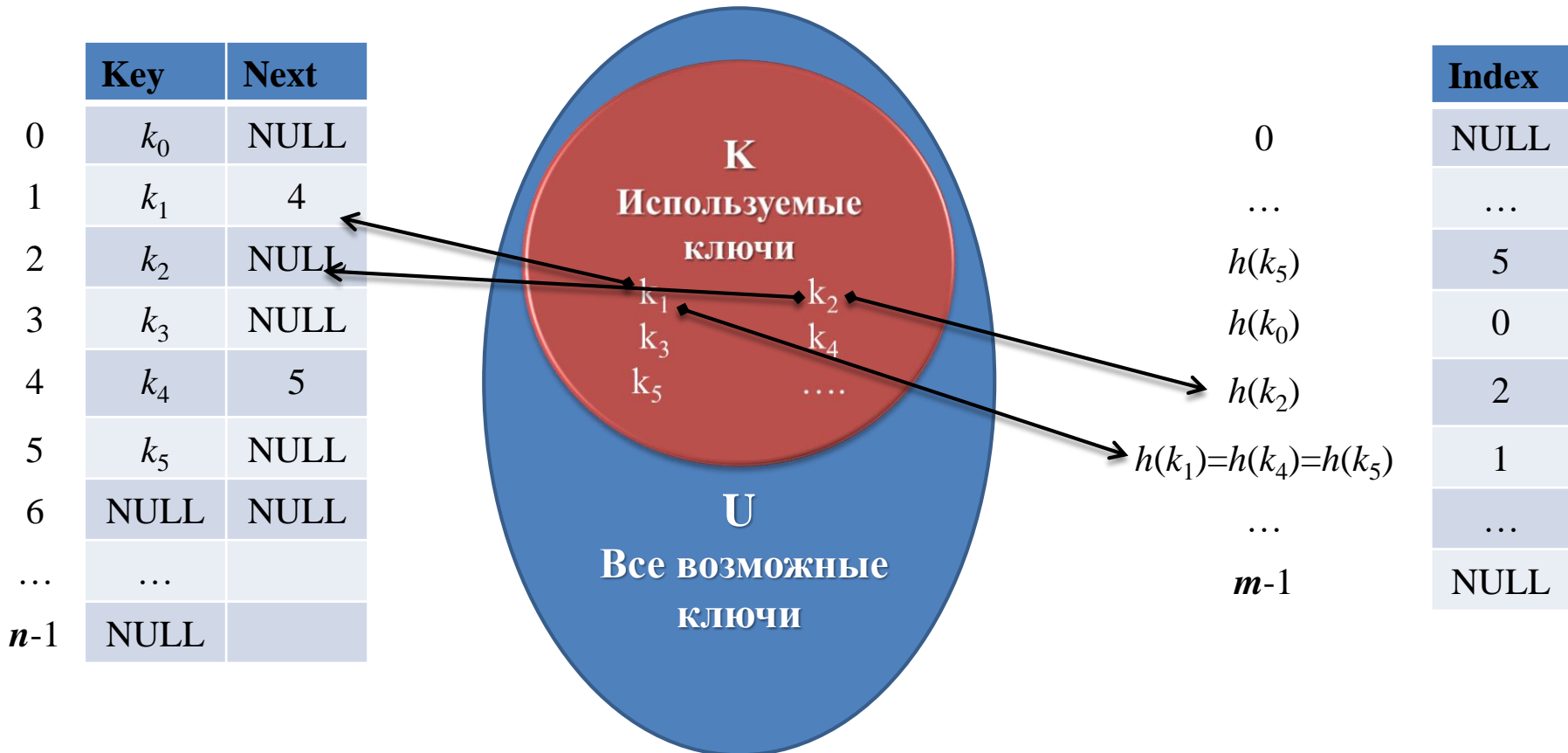
13

```
const
  NULL=-1;
  M=...;
type
  TElement=record
    { Info: TInfo; }
    key: Integer;
    Next: Integer;
end;
THashTab=array [0..M-1] of TElement;
```

Реализация метода цепочек

14

- Таблица элементов хранит элементы (ключ и индекс следующего элемента цепочки) в порядке поступления, хеш-таблица хранит индекс элемента в таблице элементов.



Реализация метода цепочек

15

```
const
  NULL=-1;
  M=...;
  N=...;
type
  TElement=record
    { Info: TInfo; }
    key: Integer;
    Next: Integer;
end;
TElementTab=array [0..N-1] of TElement;
THashTab=array [0..M-1] of Integer;
```

Выбор хеш-функции

16

- Хорошая хеш-функция должна (приблизительно) удовлетворять предположению равномерного хеширования: для очередного ключа все хеш-значения должны быть равновероятны.

$$\sum_{k: h(k)=j} P(k) = \frac{1}{m} \text{ для } j = 0, 1, \dots, m - 1$$

- Например, если ключи – случайные действительные числа, независимо и равномерно распределенные на $[0; 1[$, то хеш-функция $h(k) = \text{Int}(k * m)$ удовлетворяет этому условию.

Выбор хеш-функции

17

- На практике при выборе хеш-функции пользуются различными эвристиками, основанными на специфике задачи.
 - Хеш-функцию подбирают так, чтобы ее значения не были связаны с различными закономерностями, которые могут встретиться в хешируемых данных.
 - Пример: таблица символов компилятора, в которой хранятся идентификаторы программы. Хорошая хеш-функция выдаст различные значения у схожих идентификаторов (например, `CurrentIndex` и `CurrentPtr`).
- Подходы к построению хеш-функций
 - Деление с остатком
 - Умножение

Парадокс дней рождения

18

- Какова вероятность, что у двух человек в группе из 23 человек дни рождения совпадут?

Парадокс дней рождения

19

- Какова вероятность, что у двух человек в группе из 23 человек дни рождения совпадут?
- $> 50\%$

Парадокс дней рождения

20

- Какова вероятность, что у двух человек в группе из 23 человек дни рождения совпадут?
- **Парадóкс дней рождéния** — утверждение, гласящее, что если дана группа из 23 или более человек, то вероятность того, что хотя бы у двух из них дни рождения (число и месяц) совпадут, превышает 50 %.
- Для 60 и более человек вероятность такого совпадения превышает 99 %

Хеш-функции на базе деления

21

- $h(k) = k \bmod m$
- Выбор значения m
 - Если $m = 2^p$, то $h(k)$ – это p младших битов числа k . Если нет уверенности, что комбинации младших битов числа будут встречаться с одинаковой частотой, то степень двойки в качестве m – плохой выбор.
 - Если ключи – числа в системе счисления с основанием 2^p , то $m = 2^p - 1$ – плохой выбор.
 - Хорошие результаты, как правило, получаются, если m – простое число, далеко отстоящее от степеней двойки.
 - Пусть в хеш-таблицу с цепочками надо поместить 2000 элементов, и мы допускаем перебор до 3 вариантов при поиске отсутствующего в таблице элемента. Тогда простое число $m = 701$ – хороший выбор ($701 \approx 2000/3$).

Хеш-функции на базе умножения

22

- $h(k) = \text{Int}(m * \text{Frac}(k * A))$, где $0 < A < 1$
- Выбор значения m
 - Качество хеш-функции не зависит от выбора m .
 - Обычно в качестве m выбирают степень двойки, т.к. в этом случае умножение реализуется с помощью более быстрой операции сдвига.
 - Некоторые значения A дают лучшие результаты относительно других. Например, $A = \frac{\sqrt{5} - 1}{2}$
 - Оптимальный выбор зависит от хешируемых данных.

Разрешение коллизий: метод открытой адресации

23

- Цепочки отсутствуют, все записи хранятся в хеш-таблице.
- При поиске хеш-адрес вычисляется несколько раз (и должен давать различные значения).
 - ▣ Хеш-функция имеет вид
$$h: U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$$
 - ▣ Последовательность проб для данного ключа k :
$$(h_i) = \{ h_0(k, 0), h_1(k, 1), \dots, h_{m-1}(k, m-1) \}$$

Линейное апробирование

24

- В случае коллизии рассмотреть следующую по порядку ячейку таблицы.
- $h_i(k) = (h_0(k) + i) \bmod m$.
- Пример: вставка записей с ключами 89, 18, 49, 58, 69.
- Проблемы:
 - ▣ кластеризация
 - ▣ реализация удаление записи
 - ▣ высокая стоимость вставки записи: $1/(1-p)$, где p – степень заполнения таблицы.

	Key
0	49
1	58
2	69
3	
4	
5	
6	
7	
8	18
9	89

Реализация метода открытой адресации

25

```
const
  NULL=-1;
  M=...;
type
  TElement=record
    { Info: TInfo; }
    Key: Integer;
  end;
  THashTab=array [0..M-1] of TElement;
```

Реализация метода открытой адресации: вставка

26

```
function Insert(k: Integer):  
Integer;  
var i, j: Integer;  
begin  
    i:=0;  
    Result:=NULL; { переполнение }  
    repeat  
        j:=h(k, i);  
        if HT[j]=NULL then begin  
            HT[j]:=k;  
            Result:=j;  
            break;  
        end  
        else  
            i:=i+1;  
    until i=m;  
end;
```

Реализация метода открытой адресации: ПОИСК

27

```
function Search(k: Integer): Integer;
var i, j: Integer;
begin
  i:=0;
  Result:=NULL; { отсутствует }
  repeat
    j:=h(k, i);
    if HT[j]=k then begin
      Result:=j;
      break;
    end;
    i:=i+1;
  until (HT[j]=NULL) or (i=m);
end;
```

Реализация метода открытой адресации: удаление

28

```
function Remove(k: Integer): Integer;
var i, j: Integer;
begin
  i:=0;
  Result:=NULL; { отсутствует }
  repeat
    j:=h(k, i);
    if HT[j]=k then begin
      Result:=j;
      HT[j]:=NULL;
      break;
    end;
    i:=i+1;
  until (HT[j]=NULL) or (i=m);
end;
```

Квадратичное апробирование

29

- $h_i(k) = (h_0(k) + i^2) \bmod m$.
- Размер таблицы должен быть в половину больше по сравнению с линейным апробированием, m должно быть простым числом.
- Пример: вставка записей с ключами 89, 18, 49, 58, 69.
 - $89 \rightarrow 9$
 - $18 \rightarrow 8$
 - $49 \rightarrow 9, 0$
 - $58 \rightarrow 8, 9, 2$
 - $69 \rightarrow 9, 0, 3$

	Key
0	49
1	
2	58
3	69
4	
5	
6	
7	
8	18
9	89

Квадратичное апробирование

30

- Пример: вставка записей с ключами 10, 20, 30, 40, 50, 60, 70.
 - ▣ $10 \rightarrow 0$
 - ▣ $20 \rightarrow 0, 1$
 - ▣ $30 \rightarrow 0, 1, 4$
 - ▣ $40 \rightarrow 0, 1, 4, 9$
 - ▣ $50 \rightarrow 0, 1, 4, 9, 6 (16)$
 - ▣ $60 \rightarrow 0, 1, 4, 9, 6 (16), 5 (25)$
 - ▣ $70 \rightarrow 0, 1, 4, 9, 6 (16), 5 (25), 6 (36), 9 (49), 4 (64), 1 (81), 0 (100), 1 (121), 4 (144), 9 (169), 6 (196), \dots$

	Key
0	10
1	20
2	
3	
4	30
5	60
6	50
7	
8	
9	40

Двойное хеширование

31

- $h_i(k) = (h(k) + i * h_2(k)) \bmod m$.
- Например, $h_2(k) = R - (k \bmod R)$, где простое число $R < m$.
- Пример: вставка записей с ключами 89, 18, 49, 58, 69, 23 и $R=7$
 - $89 \rightarrow 9$
 - $18 \rightarrow 8$
 - $49 \rightarrow 9, 6$
 - $58 \rightarrow 8, 3$
 - $69 \rightarrow 9, 0$
 - $23 \rightarrow 3, 8, 3, 8, \dots$

	Key
0	69
1	
2	
3	58
4	
5	
6	49
7	
8	18
9	89

Рехеширование

32

- Если таблица почти заполнена, создать новую таблицу с размером, по крайней мере в 2 раза большим, чем у предыдущей таблицы. Вычислить новый хеш-адрес каждой записи, и вставить ее в новую таблицу.
- Рехеширование осуществляется, когда таблица заполнена наполовину, или, не удастся вставить запись, или достигнуто определенное значение заполнения таблицы.

Рехеширование

33

	Key
0	6
1	15
2	23
3	24
4	
5	
6	13



	Key
0	
...	
6	6
7	23
8	24
...	
13	13
14	
15	15
16	

Заключение

34

- Линейный и бинарный поиск в массиве
- Прямая адресация в массиве
- Хеширование
 - ▣ хеш-адрес
 - ▣ хеш-функция
 - ▣ коллизия
- Методы построения хеш-функций
 - ▣ деление с остатком
 - ▣ умножение
- Методы разрешения коллизий
 - ▣ метод цепочек
 - ▣ открытая адресация