Алгоритмы и структуры данных

Хеш-функции, универсальное хеширование, кукушкино хеширование

Хеш-множества

Операции:

1. find(value)   -> O(1)

----

7. insert(value)   -> O(1)
8. remove(value)   -> O(1)

Спойлер: хеш-таблицы лучше здесь (при правильной реализации дадут O(1))

Хеш-таблицы: метод цепочек

Давайте хранить в массиве не элементы, а головы связных списков

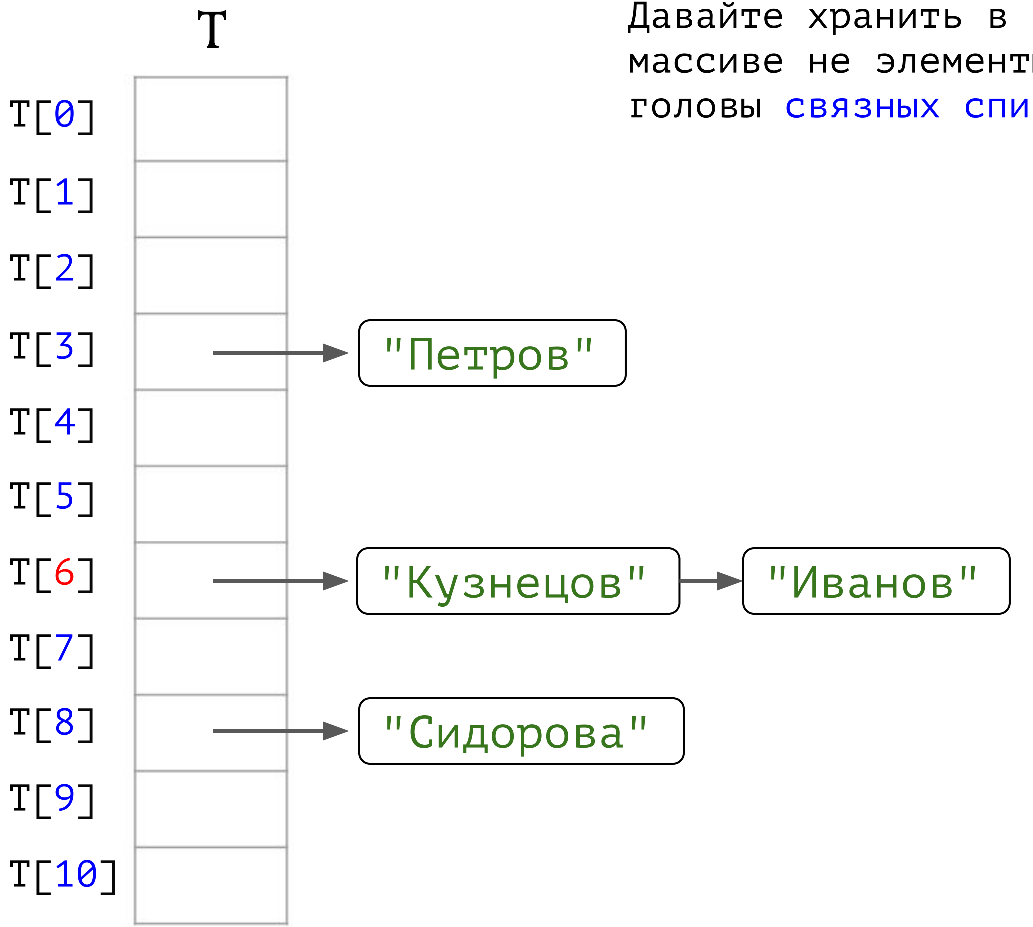
U

U - строки, фамилии

insert("Кузнецов")
h("Кузнецов") = 6

Произошла коллизия!

Добавляем в голову связного списка из соответствующей ячейки.



Хеш-таблицы: метод цепочек

Пусть хеш-функция h распределяет ключи по ячейкам равномерно и независимо. Назовем свойство простым равномерным хешированием.

Утверждение #1: мат. ожидание количества операций при неуспешном поиске в хеш-таблице с методом цепочек и коэффициентом заполненности $\alpha = \frac{n}{m}$ в предположении равномерного хеширования равно $O(1 + \alpha)$

Утверждение #2: мат. ожидание количества операций при успешном поиске в хеш-таблице с методом цепочек и коэффициентом заполненности $\alpha = \frac{n}{m}$ в предположении равномерного хеширования равно $O(1 + \alpha)$

Хеш-таблицы: метод открытой адресации

	T
T[0]	
T[1]	
T[2]	
T[3]	"Петров"
T[4]	
T[5]	
T[6]	"Иванов"
T[7]	"Кузнецов"
T[8]	"Сидорова"
T[9]	"Ким"
T[10]	

Как при этом меняются остальные операции?

Поиск:

1. Ищем с помощью исследования,
2. Проверяем по ключу, нашли или нет

```
find("Сергеев")
h("Сергеев") = 7
```

Это был пример успешного поиска,
а когда поиск пора признавать неуспешным?

Дошли до пустоты, значит не нашли.

Хеш-таблицы: метод открытой адресации

А что по поводу сложности операций?

Худший случай любой операции понятен: O(N)

Утверждение #1: мат. ожидание количества исследований при неуспешном поиске в хеш-таблице с открытой адресацией и коэффициентом заполненности $\alpha = \frac{n}{m} < 1$ в предположении равномерного хеширования не превышает $\frac{1}{1-a}$

А значит вставка в среднем работает за $O(1 + \frac{1}{1-a})$

Утверждение #2: мат. ожидание … при успешном поиске … не превышает $\frac{1}{a} ln(\frac{1}{1-a})$

Хеш-таблицы: осталось обсудить

1. Примеры хороших, плохих, ~~злых~~ хеш-функций
2. Как приблизиться к равномерному хешированию?
3. Сколько бакетов иметь и добавлять при рехеше?
4. Как выработать устойчивость к pathological dataset?

Хеш-функции

…	1	2	3	…	…	…	13	…	…	…	…	42	…	…

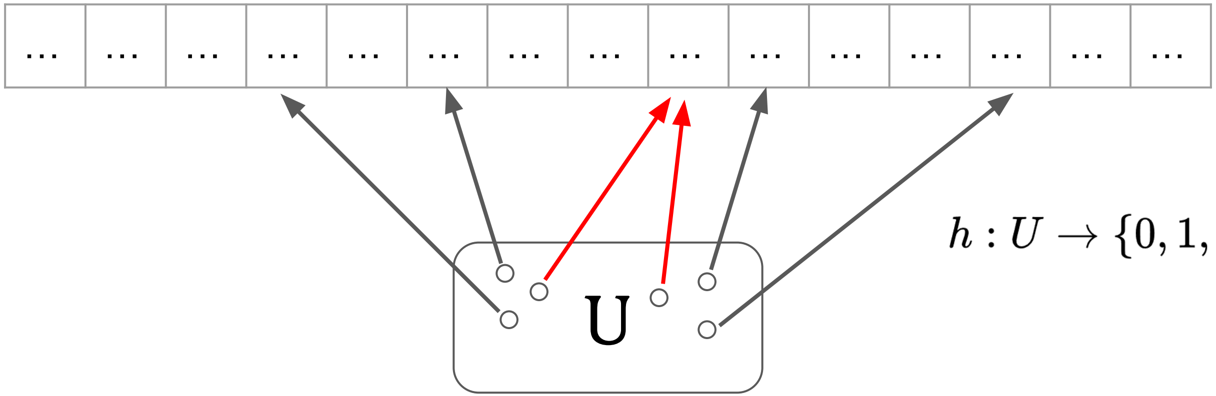
Пусть у нас есть массив размера m.
И пусть U - множество ключей, |U| ⩾ m.

Тогда функцию $h : U \to \{0, 1, 2, \ldots, m-1\}$ назовем хеш-функцией.

Обычно:

$$\begin{tikzcd} \mathrm{U} \arrow[r, "\text{hashCode}"'] & \mathbb{Z} \arrow[r, "\text{функция сжатия}"'] & \{0, 1, 2, \ldots, m-1\} \end{tikzcd}$$

Хеш-функции



$h : U \to \{0, 1, 2, \ldots, m-1\}$

Пусть у нас есть массив размера m.
И пусть U - множество ключей, |U| ⩾ m.

Тогда функцию $h : U \to \{0, 1, 2, \ldots, m-1\}$ назовем хеш-функцией.
Ситуацию: $x, y \in U : h(x) = h(y)$ будем называть коллизией.

Хеш-функции

Соображения (про эффективность операций):

1. Все сильно зависит от хеш-функции h(x)

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$: $\forall x \in U \Rightarrow h(x) = 0$

   Тогда шанс коллизии - 100% => таблица вырождается в связный список, а сложность операций - O(N)

Хеш-функции

Соображения (про эффективность операций):

1. Все сильно зависит от хеш-функции h(x)

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$: $\forall x \in U \Rightarrow h(x) = 0$

   Тогда шанс коллизии - 100% => таблица вырождается в связный список, а сложность операций - O(N)

   Понятно, что такая хеш-функция - плохая и брать ее не стоит.

Хеш-функции

Еще примеры плохих хеш-функций:

1. Пусть U - множество телефонных номеров 8-913-568-23-51, etc

Хеш-функции

Еще примеры плохих хеш-функций:

1. Пусть U - множество телефонных номеров 8-913-568-23-51, etc

   И пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$ функция, которая по каждому номеру берет его верхние 4 цифры (допустим, что m - 10000)

   Насколько это плохо?

Хеш-функции

Еще примеры плохих хеш-функций:

1. Пусть U - множество телефонных номеров 8-913-568-23-51, etc

   И пусть $h: U \to \{0, 1, 2, ..., m-1\}$ функция, которая по каждому номеру берет его верхние 4 цифры (допустим, что m - 10000)

   Насколько это плохо? Очень плохо. Коллизия будет происходить для всех номеров из одного региона.

Хеш-функции

Еще примеры плохих хеш-функций:

2. Пусть U - множество телефонных номеров 8-913-568-23-51, etc

   И пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$ функция, которая по каждому номеру берет его нижние 4 цифры (допустим, что m - 10000)

Хеш-функции

Еще примеры плохих хеш-функций:

2. Пусть U - множество телефонных номеров 8-913-568-23-51, etc

   И пусть $h: U \to \{0, 1, 2, \ldots, m-1\}$ функция, которая по каждому номеру берет его нижние 4 цифры (допустим, что m - 10000)

   Это уже чуть лучше, хотя коллизий все еще очень много.

Хеш-функции

Еще примеры плохих хеш-функций:

2. Пусть U - множество телефонных номеров 8-913-568-23-51, etc

   И пусть $h: U \to \{0, 1, 2, \ldots, m-1\}$ функция, которая по каждому номеру берет его нижние 4 цифры (допустим, что m - 10000)

   Это уже чуть лучше, хотя коллизий все еще очень много.

   Тем не менее, мы избавились от искусственного распределения данных, можно считать, что последние 4 цифры распределены равномерно.

Хеш-функции

Еще примеры плохих хеш-функций:

2. Пусть U - множество телефонных номеров 8-913-568-23-51, etc

   И пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$ функция, которая по каждому номеру берет его нижние 4 цифры (допустим, что m - 10000)

   Это уже чуть лучше, хотя коллизий все еще очень много.

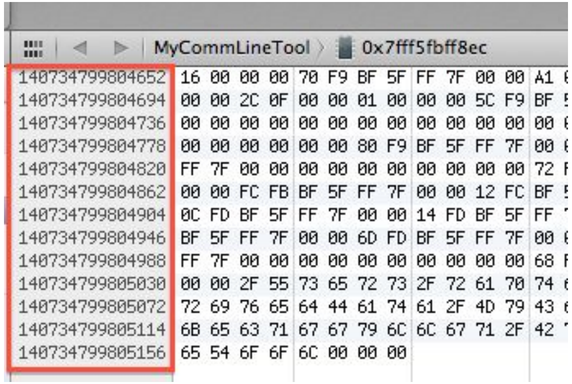
   Тем не менее, мы избавились от искусственного распределения данных, можно считать, что последние 4 цифры распределены равномерно.

   Мысль: было бы хорошо использовать в хешах всю исходную информацию, а не только суффикс или префикс

Хеш-функции

Еще примеры плохих хеш-функций:

3. Пусть U - множество адресов объектов в памяти.

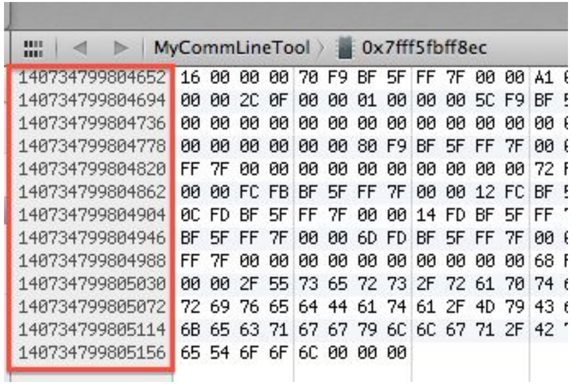


Хеш-функции

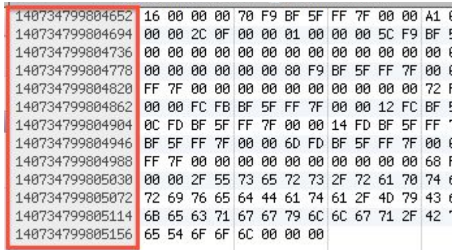
Еще примеры плохих хеш-функций:

3. Пусть U - множество адресов объектов в памяти.

   Адреса в памяти обычно выровнены на размер машинного слова. Т.е. кратны 4 или 8.



Хеш-функции



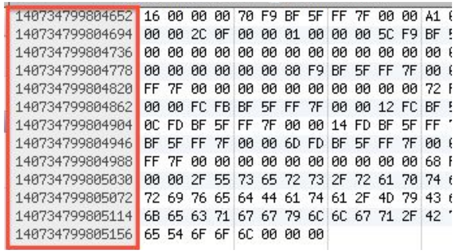
Еще примеры плохих хеш-функций:

3. Пусть U - множество адресов объектов в памяти.

   Адреса в памяти обычно выровнены на размер машинного слова. Т.е. кратны 4 или 8.

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$ снова берет нижние 4 цифры.

Хеш-функции



Еще примеры плохих хеш-функций:

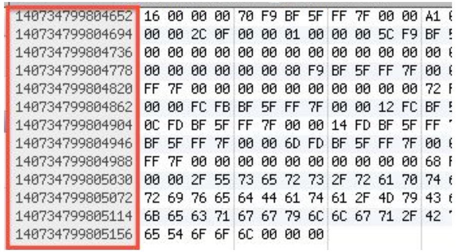
3. Пусть U - множество адресов объектов в памяти.

   Адреса в памяти обычно выровнены на размер машинного слова. Т.е. кратны 4 или 8.

   Пусть $h : U \to \{0, 1, 2, \dots, m-1\}$ снова берет нижние 4 цифры.

   Т.е. $h(x) = x \: mod \: 1000;$ Есть ли здесь проблема?

Хеш-функции



Еще примеры плохих хеш-функций:

3. Пусть U - множество адресов объектов в памяти.

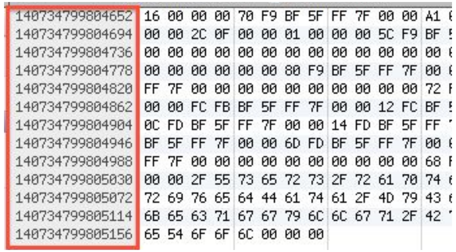
   Адреса в памяти обычно выровнены на размер машинного слова. Т.е. кратны 4 или 8.

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$ снова берет нижние 4 цифры.

   Т.е. $h(x) = x \; mod \, 1000;$ Есть ли здесь проблема?

   Все адреса изначально четные => и остаток от деления на 1000 (которая тоже четная) будет четным!

Хеш-функции



Еще примеры плохих хеш-функций:

3. Пусть U - множество адресов объектов в памяти.

   Адреса в памяти обычно выровнены на размер машинного слова. Т.е. кратны 4 или 8.

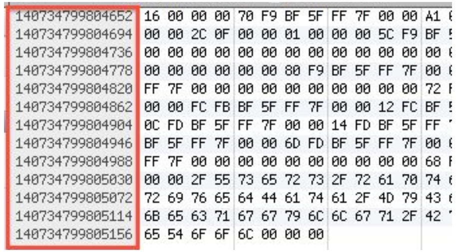
   Пусть $h : U \to \{0, 1, 2, \dots, m-1\}$ снова берет нижние 4 цифры.

   Т.е. $h(x) = x \; mod \, 1000;$ Есть ли здесь проблема?

   Все адреса изначально четные => и остаток от деления на 1000 (которая тоже четная) будет четным!

   А значит, что все нечетные buckets останутся пустыми.

Хеш-функции



140734799804652	16	00	00	00	70	F9	BF	5F	FF	7F	00	00	A1
140734799804694	00	00	2C	0F	00	00	01	00	00	00	5C	F9	BF
140734799804736	00	00	00	00	00	00	00	00	00	00	00	00	00
140734799804778	00	00	00	00	00	00	80	F9	BF	5F	FF	7F	00
140734799804820	FF	7F	00	00	00	00	00	00	00	00	00	00	72
140734799804862	00	00	FC	FB	BF	5F	FF	7F	00	00	12	FC	BF
140734799804904	0C	FD	BF	5F	FF	7F	00	00	14	FD	BF	5F	FF
140734799804946	BF	5F	FF	7F	00	00	6D	FD	BF	5F	FF	7F	00
140734799804988	FF	7F	00	00	00	00	00	00	00	00	00	00	68
140734799805030	00	00	2F	55	73	65	72	73	2F	72	61	70	74
140734799805072	72	69	76	65	64	44	61	74	61	2F	4D	79	43
140734799805114	6B	65	63	71	67	67	79	6C	6C	67	71	2F	42
140734799805156	65	54	6F	6F	6C	00	00	00					

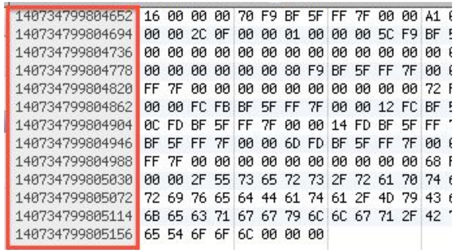
Еще примеры плохих хеш-функций:

4. Пусть U - множество адресов объектов в памяти.

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$; $h(x) = x \; mod \; 2^k$

   Насколько это плохо?

Хеш-функции



140734799804652	16	00	00	00	70	F9	BF	5F	FF	7F	00	00	A1
140734799804694	00	00	2C	0F	00	00	01	00	00	00	5C	F9	BF
140734799804736	00	00	00	00	00	00	00	00	00	00	00	00	00
140734799804778	00	00	00	00	00	00	80	F9	BF	5F	FF	7F	00
140734799804820	FF	7F	00	00	00	00	00	00	00	00	00	00	72
140734799804862	00	00	FC	FB	BF	5F	FF	7F	00	00	12	FC	BF
140734799804904	0C	FD	BF	5F	FF	7F	00	00	14	FD	BF	5F	FF
140734799804946	BF	5F	FF	7F	00	00	6D	FD	BF	5F	FF	7F	00
140734799804988	FF	7F	00	00	00	00	00	00	00	00	00	00	68
140734799805030	00	00	2F	55	73	65	72	73	2F	72	61	70	74
140734799805072	72	69	76	65	64	44	61	74	61	2F	4D	79	43
140734799805114	6B	65	63	71	67	67	79	6C	6C	67	71	2F	42
140734799805156	65	54	6F	6F	6C	00	00	00					

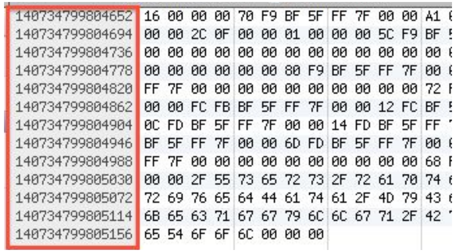
Еще примеры плохих хеш-функций:

4. Пусть U - множество адресов объектов в памяти.

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$; $h(x) = x \, mod \, 2^k$

   Насколько это плохо? Еще хуже.

Хеш-функции



140734799804652	16	00	00	00	70	F9	BF	5F	FF	7F	00	00	A1
140734799804694	00	00	2C	0F	00	00	01	00	00	00	5C	F9	BF
140734799804736	00	00	00	00	00	00	00	00	00	00	00	00	00
140734799804778	00	00	00	00	00	00	80	F9	BF	5F	FF	7F	00
140734799804820	FF	7F	00	00	00	00	00	00	00	00	00	00	72
140734799804862	00	00	FC	FB	BF	5F	FF	7F	00	00	12	FC	BF
140734799804904	0C	FD	BF	5F	FF	7F	00	00	14	FD	BF	5F	FF
140734799804946	BF	5F	FF	7F	00	00	6D	FD	BF	5F	FF	7F	00
140734799804988	FF	7F	00	00	00	00	00	00	00	00	00	00	68
140734799805030	00	00	2F	55	73	65	72	73	2F	72	61	70	74
140734799805072	72	69	76	65	64	44	61	74	61	2F	4D	79	43
140734799805114	6B	65	63	71	67	67	79	6C	6C	67	71	2F	42
140734799805156	65	54	6F	6F	6C	00	00	00					

Еще примеры плохих хеш-функций:

4. Пусть U - множество адресов объектов в памяти.

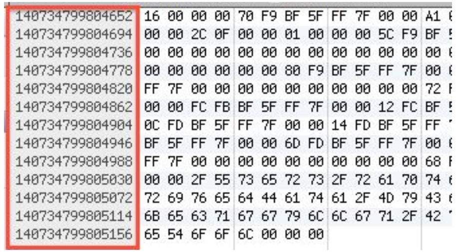
   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$; $h(x) = x \mod 2^k$

   Насколько это плохо? Еще хуже.

   Во-первых, проблема с четностью никуда не делать.

   Во-вторых, $x \mod 2^k$ - это же просто взятие нижних k битов!
   а мы про них как минимум знаем, что нижние 3 равны нулю (из-за выравнивания на 8).

Хеш-функции



140734799804652	16	00	00	00	70	F9	BF	5F	FF	7F	00	00	A1
140734799804694	00	00	2C	0F	00	00	01	00	00	00	5C	F9	BF
140734799804736	00	00	00	00	00	00	00	00	00	00	00	00	00
140734799804778	00	00	00	00	00	00	80	F9	BF	5F	FF	7F	00
140734799804820	FF	7F	00	00	00	00	00	00	00	00	00	00	72
140734799804862	00	00	FC	FB	BF	5F	FF	7F	00	00	12	FC	BF
140734799804904	0C	FD	BF	5F	FF	7F	00	00	14	FD	BF	5F	FF
140734799804946	BF	5F	FF	7F	00	00	6D	FD	BF	5F	FF	7F	00
140734799804988	FF	7F	00	00	00	00	00	00	00	00	00	00	68
140734799805030	00	00	2F	55	73	65	72	73	2F	72	61	70	74
140734799805072	72	69	76	65	64	44	61	74	61	2F	4D	79	43
140734799805114	6B	65	63	71	67	67	79	6C	6C	67	71	2F	42
140734799805156	65	54	6F	6F	6C	00	00	00					

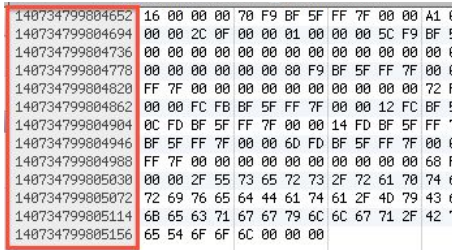
Еще примеры плохих хеш-функций:

5. Бонус: реальный пример из жизни.

   Пусть U - множество адресов объектов в памяти.

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\};\;\; h(x) = (x >> log_2(sizeof(void*)) \; mod \; 2^k$

Хеш-функции



Еще примеры плохих хеш-функций:

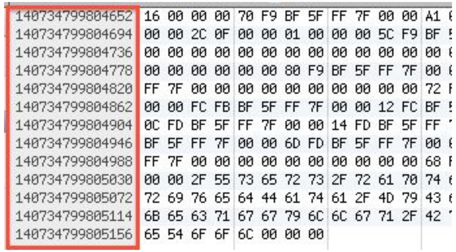
5. Бонус: реальный пример из жизни.

   Пусть U - множество адресов объектов в памяти.

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$; $h(x) = (x >> log_2(sizeof(void*)) \; mod \; 2^k$

   Сразу про нижние биты подумали, поэтому добавили сдвиг.

Хеш-функции



140734799804652	16	00	00	00	70	F9	BF	5F	FF	7F	00	00	A1
140734799804694	00	00	2C	0F	00	00	01	00	00	00	5C	F9	BF
140734799804736	00	00	00	00	00	00	00	00	00	00	00	00	00
140734799804778	00	00	00	00	00	00	80	F9	BF	5F	FF	7F	00
140734799804820	FF	7F	00	00	00	00	00	00	00	00	00	00	72
140734799804862	00	00	FC	FB	BF	5F	FF	7F	00	00	12	FC	BF
140734799804904	0C	FD	BF	5F	FF	7F	00	00	14	FD	BF	5F	FF
140734799804946	BF	5F	FF	7F	00	00	6D	FD	BF	5F	FF	7F	00
140734799804988	FF	7F	00	00	00	00	00	00	00	00	00	00	68
140734799805030	00	00	2F	55	73	65	72	73	2F	72	61	70	74
140734799805072	72	69	76	65	64	44	61	74	61	2F	4D	79	43
140734799805114	6B	65	63	71	67	67	79	6C	6C	67	71	2F	42
140734799805156	65	54	6F	6F	6C	00	00	00					

Еще примеры плохих хеш-функций:

5. Бонус: реальный пример из жизни.

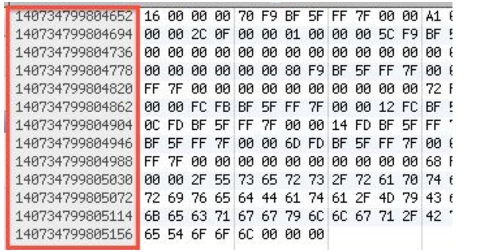
   Пусть U - множество адресов объектов в памяти.

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$; $h(x) = (x >> log_2(sizeof(\texttt{void}*)) \; mod \; 2^k$

   Сразу про нижние биты подумали, поэтому добавили сдвиг.

   Вот только не учли, что эта хеш-таблица использовалась для специальных объектов фиксированного размера - 128 байт, которые аллоцировались подряд.

Хеш-функции



Еще примеры плохих хеш-функций:

5. Бонус: реальный пример из жизни.

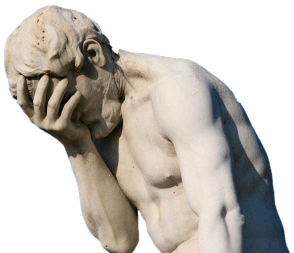
   Пусть U - множество адресов объектов в памяти.

   Пусть $h : U \to \{0, 1, 2, \ldots, m-1\}$; $h(x) = (x >> log_2(sizeof(\texttt{void}*)) \; mod \; 2^k$

   Сразу про нижние биты подумали, поэтому добавили сдвиг.

   Вот только не учли, что эта хеш-таблица использовалась для специальных объектов фиксированного размера - 128 байт, которые аллоцировались подряд.

   Угадайте, в какой момент начались проблемы с производительностью.



Хеш-функции

Практические советы:

1. Нужно стараться задействовать все биты/цифры из ключа, а не только их часть

Хеш-функции

Практические советы:

1. Нужно стараться задействовать все биты/цифры из ключа, а не только их часть

2. Нужно учитывать специфику исходных данных, хеширование числовых значений и адресов - может и должно отличаться.

Хеш-функции

Практические советы:

1. Нужно стараться задействовать все биты/цифры из ключа, а не только их часть

2. Нужно учитывать специфику исходных данных, хеширование числовых значений и адресов - может и должно отличаться.

Практические советы по количеству buckets (на которое вы зачастую делите при сжатии хеша):

Хеш-функции

Практические советы:

1. Нужно стараться задействовать все биты/цифры из ключа, а не только их часть

2. Нужно учитывать специфику исходных данных, хеширование числовых значений и адресов - может и должно отличаться.

Практические советы по количеству buckets (на которое вы зачастую делите при сжатии хеша):

1. #(buckets) = простое число

Хеш-функции

Практические советы:

1. Нужно стараться задействовать все биты/цифры из ключа, а не только их часть

2. Нужно учитывать специфику исходных данных, хеширование числовых значений и адресов - может и должно отличаться.

Практические советы по количеству buckets (на которое вы зачастую делите при сжатии хеша):

1. #(buckets) = простое число
(чтобы было меньше общих делителей с данными)

Хеш-функции

Практические советы:

1. Нужно стараться задействовать все биты/цифры из ключа, а не только их часть

2. Нужно учитывать специфику исходных данных, хеширование числовых значений и адресов - может и должно отличаться.

Практические советы по количеству buckets (на которое вы зачастую делите при сжатии хеша):

1. #(buckets) = простое число
(чтобы было меньше общих делителей с данными)

2. не нужно брать степень двойки (это просто нижние биты)

Хеш-функции

Практические советы:

1. Нужно стараться задействовать все биты/цифры из ключа, а не только их часть

2. Нужно учитывать специфику исходных данных, хеширование числовых значений и адресов - может и должно отличаться.

Практические советы по количеству buckets (на которое вы зачастую делите при сжатии хеша):

1. #(buckets) = простое число
(чтобы было меньше общих делителей с данными)

2. не нужно брать степень двойки (это просто нижние биты)

3. не нужно брать степень десятки (это просто нижние цифры)

Хеш-функции: метод деления

Пусть: $h(x) = t(x) \, mod \, m$

Где $t(x)$ - функция, преобразующая ключи к целым неотрицательным числам

m - количествоа бакетов, выбираемое по описанным ранее принципам: простое, не степень двойки, обычно близкое к количеству элементов домноженному на константу

Хеш-функции: метод деления

Пусть: $h(x) = t(x) \, mod \, m$

Где $t(x)$ - функция, преобразующая ключи к целым неотрицательным числам

m - количествоа бакетов, выбираемое по описанным ранее принципам: простое, не степень двойки, обычно близкое к количеству элементов домноженному на константу

Пример: допустим у вас 350 элементов в хеш-таблице и пора делать рехеш, какое новое m выбрать?

Хеш-функции: метод деления

Пусть: $h(x) = t(x)\, mod\, m$

Где $t(x)$ - функция, преобразующая ключи к целым неотрицательным числам

m - количествоа бакетов, выбираемое по описанным ранее принципам: простое, не степень двойки, обычно близкое к количеству элементов домноженному на константу

Пример: допустим у вас 350 элементов в хеш-таблице и пора делать рехеш, какое новое m выбрать?

берем близкое к 700 простое число: 701

Хеш-функции: метод деления

Пусть: $h(x) = t(x)\, mod\, m$

Где $t(x)$ - функция, преобразующая ключи к целым неотрицательным числам

m - количествоа бакетов, выбираемое по описанным ранее принципам: простое, не степень двойки, обычно близкое к количеству элементов домноженному на константу

Пример: допустим у вас 350 элементов в хеш-таблице и пора делать рехеш, какое новое m выбрать?

берем близкое к 700 простое число: 701

простые числа можем брать из таблиц или вычислять не самыми наивными алгоритмами.

Хеш-функции: метод деления

Пусть: $h(x) = t(x) \, mod \, m$

Где $t(x)$ - функция, преобразующая ключи к целым неотрицательным числам

На самом деле от t тоже много что зависит: допустим, вы хотите преобразовать строку к числу, как это сделать быстро, эффективно и чтобы минимизировать количество коллизий?

Хеш-функции: метод деления

Пусть: $h(x) = t(x) \, mod \, m$

Где $t(x)$ - функция, преобразующая ключи к целым неотрицательным числам

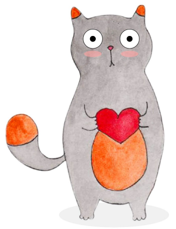
На самом деле от t тоже много что зависит: допустим, вы хотите преобразовать строку к числу, как это сделать быстро, эффективно и чтобы минимизировать количество коллизий?

Например, можно использовать MurmurHash2

```c
static inline uint32_t murmur_32_scramble(uint32_t k) {
    k *= 0xcc9e2d51;
    k = (k << 15) | (k >> 17);
    k *= 0x1b873593;
    return k;
}
uint32_t murmur3_32(const uint8_t* key, size_t len, uint32_t seed)
{
    uint32_t h = seed;
    uint32_t k;
    /* Read in groups of 4. */
    for (size_t i = len >> 2; i; i--) {
        // Here is a source of differing results across endiannesses.
        // A swap here has no effects on hash properties though.
        memcpy(&k, key, sizeof(uint32_t));
        key += sizeof(uint32_t);
        h ^= murmur_32_scramble(k);
        h = (h << 13) | (h >> 19);
        h = h * 5 + 0xe6546b64;
    }
    /* Read the rest. */
    k = 0;
    for (size_t i = len & 3; i; i--) {
        k <<= 8;
        k |= key[i - 1];
    }
    // A swap is *not* necessary here because the preceding loop already
    // places the low bytes in the low places according to whatever endianness
    // we use. Swaps only apply when the memory is copied in a chunk.
    h ^= murmur_32_scramble(k);
    /* Finalize. */
    h ^= len;
    h ^= h >> 16;
    h *= 0x85ebca6b;
    h ^= h >> 13;
    h *= 0xc2b2ae35;
    h ^= h >> 16;
    return h;
}
```

Это уже Murmur3

Mu - multiply
R - rotate



Хеш-функции: метод умножения

Пусть: $0 < A < 1$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor$

Хеш-функции: метод умножения

Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor$

Хеш-функции: метод умножения

Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor$

Сначала "нормируем" ключ, домножая его на A и беря дробную часть; затем "нормируем" уже m (количество бакетов), гарантированно попадая в промежуток [0; m)

Хеш-функции: метод умножения

Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor$

Сначала "нормируем" ключ, домножая его на A и беря дробную часть; затем "нормируем" уже m (количество бакетов), гарантированно попадая в промежуток [0; m)

Что хорошо в таком подходе - больше не нужно думать об ограничениях на m. Обычно в таких реализациях наоборот специально берут $m = 2^p$

Хеш-функции: метод умножения

Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor$

Сначала "нормируем" ключ, домножая его на A и беря дробную часть; затем "нормируем" уже m (количество бакетов), гарантированно попадая в промежуток [0; m)

Что хорошо в таком подходе - больше не нужно думать об ограничениях на m. Обычно в таких реализациях наоборот специально берут $m = 2^p$

Пусть ω - количество бит в машинном слове, тогда пусть $A = s/2^w$, где s - некоторое целое ω-битовое. И пусть x - тоже ω-битовое.

Хеш-функции: метод умножения

Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor$

Сначала "нормируем" ключ, домножая его на A и беря дробную часть; затем "нормируем" уже m (количество бакетов), гарантированно попадая в промежуток [0; m)

Что хорошо в таком подходе - больше не нужно думать об ограничениях на m. Обычно в таких реализациях наоборот специально берут $m = 2^p$

Пусть ω - количество бит в машинном слове, тогда пусть $A = s/2^w$, где s - некоторое целое ω-битовое. И пусть x - тоже ω-битовое.

Тогда сначала вычислим $x * s$, получим 2ω-битовое число: $r_1 2^\omega + r_0$

Хеш-функции: метод умножения

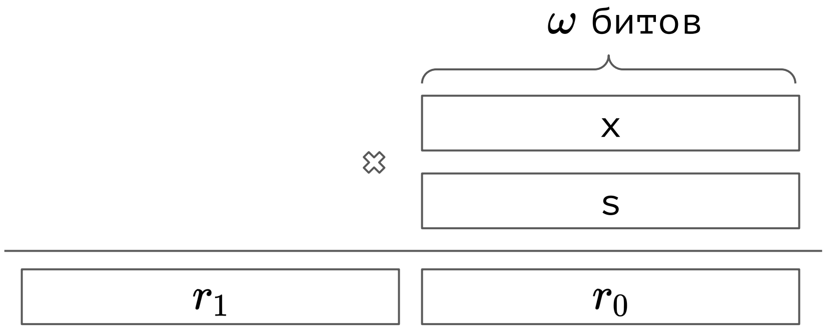
Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor, \ m = 2^p$

Пусть ω - количество бит в машинном слове, тогда пусть $A = s/2^w$, где s - некоторое целое ω-битовое. И пусть x - тоже ω-битовое.

Тогда сначала вычислим $x * s$, получим 2ω-битовое число: $r_1 2^\omega + r_0$



Хеш-функции: метод умножения

Пусть: $0 < A < 1$

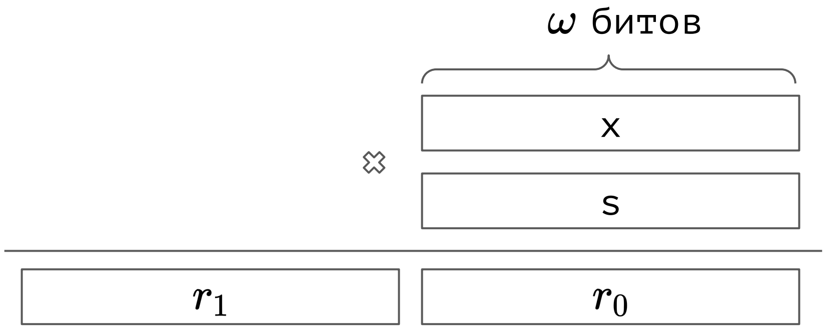
дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor, \ m = 2^p$

Пусть ω - количество бит в машинном слове, тогда пусть $A = s/2^w$, где s - некоторое целое ω-битовое. И пусть x - тоже ω-битовое.

Тогда сначала вычислим $x * s$, получим 2ω-битовое число: $r_1 2^\omega + r_0$

Тогда: $\{xA\} = r_0/2^\omega$



Хеш-функции: метод умножения

Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

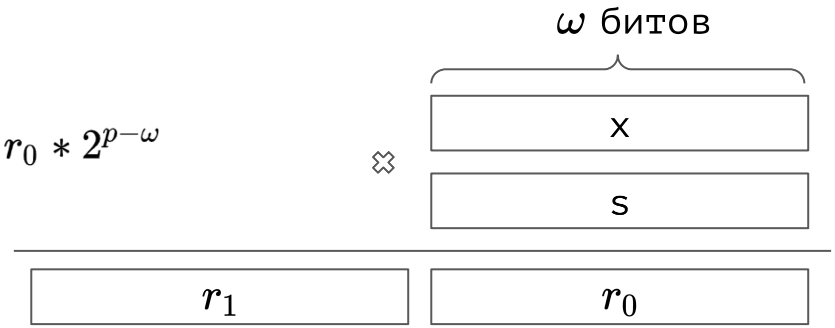
Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor, \; m = 2^p$

Пусть ω - количество бит в машинном слове, тогда пусть $A = s/2^w$, где s - некоторое целое ω-битовое. И пусть x - тоже ω-битовое.

Тогда сначала вычислим $x * s$, получим 2ω-битовое число: $r_1 2^\omega + r_0$

Тогда: $\{xA\} = r_0/2^\omega$

Получаем: $\lfloor m * \{xA\} \rfloor = 2^p * (r_0/2^\omega) = r_0 * 2^{p-\omega}$



Хеш-функции: метод умножения

Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor, \; m = 2^p$

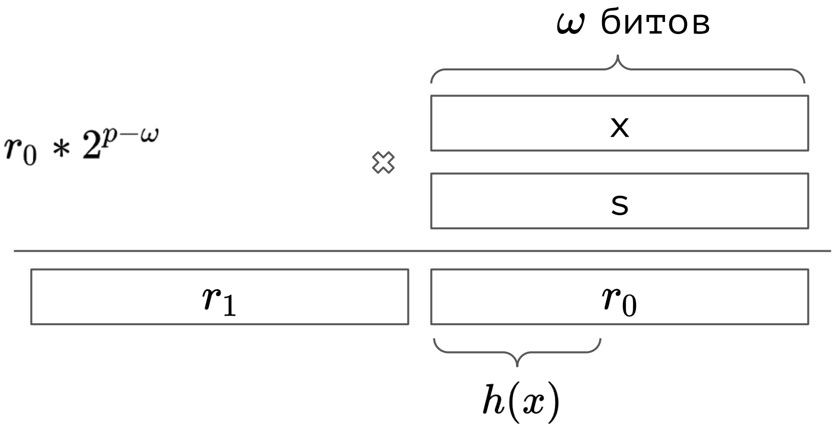
Пусть ω - количество бит в машинном слове, тогда пусть $A = s/2^w$,
где s - некоторое целое ω-битовое. И пусть x - тоже ω-битовое.

Тогда сначала вычислим $x * s$, получим 2ω-битовое число: $r_1 2^\omega + r_0$

Тогда: $\{xA\} = r_0/2^\omega$

Получаем: $\lfloor m * \{xA\} \rfloor = 2^p * (r_0/2^\omega) = r_0 * 2^{p-\omega}$

Т.е. нужно просто взять
верхние p битов от r_0!



Хеш-функции: метод умножения

Пусть: $0 < A < 1$

дробная часть: $xA - \lfloor xA \rfloor$

Тогда зададим: $h(x) = \lfloor m * \{xA\} \rfloor, \ m = 2^p$

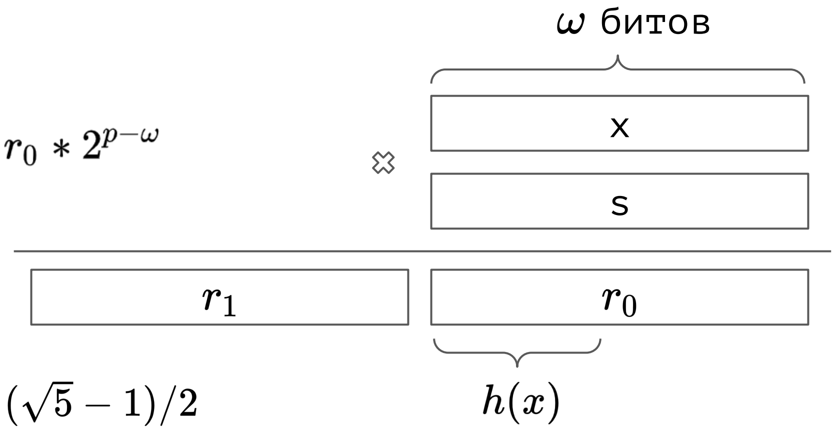
Пусть ω - количество бит в машинном слове, тогда пусть $A = s/2^w$, где s - некоторое целое ω-битовое. И пусть x - тоже ω-битовое.

Тогда сначала вычислим $x * s$, получим 2ω-битовое число: $r_1 2^\omega + r_0$

Тогда: $\{xA\} = r_0/2^\omega$

Получаем: $\lfloor m * \{xA\} \rfloor = 2^p * (r_0/2^\omega) = r_0 * 2^{p-\omega}$

Т.е. нужно просто взять верхние p битов от r_0!



На практике в качестве A берут $(\sqrt{5} - 1)/2$

Хеш-таблицы: осталось обсудить

1. Примеры хороших, плохих, ~~злых~~ хеш-функций
2. Как приблизиться к равномерному хешированию?
3. Сколько бакетов иметь и добавлять при рехеше?
4. **Как выработать устойчивость к pathological dataset?**

Хеш-таблицы: метод цепочек

Вопрос: а что со сложностью операций в такой структуре данных?

Соображения (про поиск и удаление):

1. Все сильно зависит от хеш-функции h(x)

2. Даже если хеш-функция минимизирует количество коллизий, не забываем, что $|U| \gg m$

   Это значит, что всегда есть хотя бы один bucket (элемент массива), в который хешируются как минимум $\frac{|U|}{m}$ элементов.

   Если мы начнем добавлять именно такие элементы, то как бы ни была хороша хеш-функция, мы получим множество коллизий => сложность в худшем O(N) (pathological dataset)

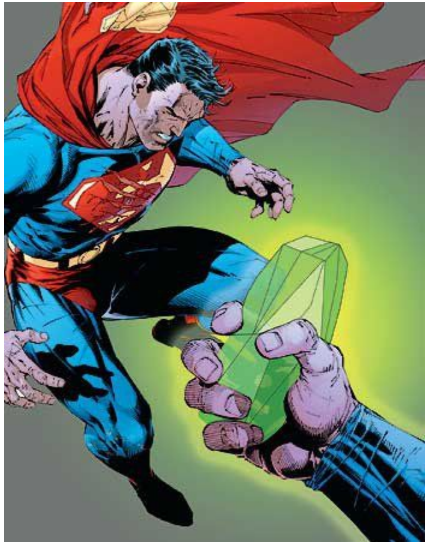
Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Такой набор всегда есть, но для каждой хеш-функции он свой.



Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Такой набор всегда есть, но для каждой хеш-функции он свой.

Почему об этом стоит беспокоиться?
(ведь сложность в среднем то отличная)

Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Такой набор всегда есть, но для каждой хеш-функции он свой.

Почему об этом стоит беспокоиться?
(ведь сложность в среднем то отличная)

Потому, что это может быть способом для атаки на ваш код.

- особенно в эпоху open source
- особенно, если хеш-функция у вас слабенькая

Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Такой набор всегда есть, но для каждой хеш-функции он свой.

Почему об этом стоит беспокоиться?
(ведь сложность в среднем то отличная)

Потому, что это может быть способом для атаки на ваш код.

- особенно в эпоху open source
- особенно, если хеш-функция у вас слабенькая

Статья, где описаны модельные атаки на некоторые DNS, прокси и даже коллекции языка Perl: Crosby, Wallach, 2003

Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Такой набор всегда есть, но для каждой хеш-функции он свой.

Почему об этом стоит беспокоиться?
(ведь сложность в среднем то отличная)

Потому, что это может быть способом для атаки на ваш код.

- особенно в эпоху open source
- особенно, если хеш-функция у вас слабенькая

Статья, где описаны модельные атаки на некоторые DNS, прокси и даже коллекции языка Perl: Crosby, Wallach, 2003

Очень успешные атаки, разница в производительности на порядки.

Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Такой набор всегда есть, но для каждой хеш-функции он свой.

Это может быть способом для атаки на ваш код.
- особенно в эпоху open source
- особенно, если хеш-функция у вас слабенькая

Что с этим делать:

1. Использовать криптографические хеш-функции (SHA-2)

Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Такой набор всегда есть, но для каждой хеш-функции он свой.

Это может быть способом для атаки на ваш код.
- особенно в эпоху open source
- особенно, если хеш-функция у вас слабенькая

Что с этим делать:

1. Использовать криптографические хеш-функции (SHA-2)
   a. Нельзя* подобрать pathological dataset
   b. Узнаете про них в курсе по криптографии

Хеш-таблицы: метод цепочек

Pathological dataset: набор данных, на которых производительность хеш-таблицы (с фиксированной хеш-функцией) падает до O(N)

Такой набор всегда есть, но для каждой хеш-функции он свой.

Это может быть способом для атаки на ваш код.

- особенно в эпоху open source
- особенно, если хеш-функция у вас слабенькая

Что с этим делать:

1. Использовать криптографические хеш-функции (SHA-2)

2. Использовать рандомизацию внутри алгоритма!
(уже делали так в QuickSort из этих же соображений)

Универсальное хеширование

Пусть $\mathcal{H}$ - семейство хеш-функций $h : U \to \{0, 1, 2, \ldots, m-1\}$

Будем называть $\mathcal{H}$ универсальным, если для любых x и y из U

$\underset{h \in \mathcal{H}}{Pr}[h(x) = h(y)] \leq \frac{1}{m}$, при условии, что h выбирается случайным образом равномерно и независимо.

Универсальное хеширование

Пусть $\mathcal{H}$ - семейство хеш-функций $h : U \to \{0, 1, 2, \ldots, m-1\}$

Будем называть $\mathcal{H}$ универсальным, если для любых x и y из U

$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \frac{1}{m}$, при условии, что h выбирается случайным образом равномерно и независимо.

Что здесь важно:

1. т.е. в среднем (в зависимости от выбора хеш-функции!) ведет себя, как равномерное хеширование

Универсальное хеширование

Пусть $\mathcal{H}$ - семейство хеш-функций $h : U \to \{0, 1, 2, \dots, m-1\}$

Будем называть $\mathcal{H}$ универсальным, если для любых x и y из U

$\underset{h \in \mathcal{H}}{Pr}[h(x) = h(y)] \leq \frac{1}{m}$, при условии, что h выбирается случайным образом равномерно и независимо.

Что здесь важно:

1. т.е. в среднем (в зависимости от выбора хеш-функции!) ведет себя, как равномерное хеширование

2. рандом теперь не по данным, а по выбору хеш-функции

Универсальное хеширование

Пусть $\mathcal{H}$ - семейство хеш-функций $h : U \to \{0, 1, 2, \ldots, m-1\}$

Будем называть $\mathcal{H}$ универсальным, если для любых x и y из U

$\underset{h \in \mathcal{H}}{Pr}[h(x) = h(y)] \leq \frac{1}{m}$, при условии, что h выбирается случайным образом равномерно и независимо.

Что здесь важно:

1. т.е. в среднем (в зависимости от выбора хеш-функции!) ведет себя, как равномерное хеширование

2. рандом теперь не по данным, а по выбору хеш-функции

3. могут встречаться и плохие хеш-функции!

Универсальное хеширование: пример

Пусть U - множество IP-адресов: $(x_1, x_2, x_3, x_4) : x_i \in \{0, 1, \ldots, 255\}$

И пусть **m** (количество бакетов) - некоторое простое число.

Универсальное хеширование: пример

Пусть U - множество IP-адресов: $(x_1, x_2, x_3, x_4) : x_i \in \{0, 1, \ldots, 255\}$

И пусть **m** (количество бакетов) - некоторое простое число.

Тогда для любого $a = (a_1, a_2, a_4, a_4) : a_i \in \{0, 1, \ldots, m-1\}$
введем $h_a(x) = (a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m$

Универсальное хеширование: пример

Пусть U - множество IP-адресов: $(x_1, x_2, x_3, x_4) : x_i \in \{0, 1, \ldots, 255\}$

И пусть **m** (количество бакетов) - некоторое простое число.

Тогда для любого $a = (a_1, a_2, a_4, a_4) : a_i \in \{0, 1, \ldots, m-1\}$
введем $h_a(x) = (a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m$

Получили семейство хеш-функций $h_a(x) : U \to \{0, 1, 2, \ldots, m-1\}$

Универсальное хеширование: пример

Пусть U - множество IP-адресов: $(x_1, x_2, x_3, x_4) : x_i \in \{0, 1, \ldots, 255\}$

И пусть **m** (количество бакетов) - некоторое простое число.

Тогда для любого $a = (a_1, a_2, a_4, a_4) : a_i \in \{0, 1, \ldots, m-1\}$
введем $h_a(x) = (a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4) \, mod \, m$

Получили семейство хеш-функций $h_a(x) : U \to \{0, 1, 2, \ldots, m-1\}$

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Универсальное хеширование: пример

Пусть U - множество IP-адресов: $(x_1, x_2, x_3, x_4) : x_i \in \{0, 1, \ldots, 255\}$

И пусть **m** (количество бакетов) - некоторое простое число.

Тогда для любого $a = (a_1, a_2, a_4, a_4) : a_i \in \{0, 1, \ldots, m-1\}$
введем $h_a(x) = (a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m$

Получили семейство хеш-функций $h_a(x) : U \to \{0, 1, 2, \ldots, m-1\}$

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$

Универсальное хеширование: пример

Пусть U - множество IP-адресов: $(x_1, x_2, x_3, x_4) : x_i \in \{0, 1, \dots, 255\}$

И пусть **m** (количество бакетов) - некоторое простое число.

Тогда для любого $a = (a_1, a_2, a_4, a_4) : a_i \in \{0, 1, \dots, m-1\}$
введем $h_a(x) = (a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m$

Получили семейство хеш-функций $h_a(x) : U \to \{0, 1, 2, \dots, m-1\}$

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr_{h \in \mathcal{H}} [h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$

Универсальное хеширование: пример

Пусть U - множество IP-адресов: $(x_1, x_2, x_3, x_4) : x_i \in \{0, 1, \dots, 255\}$

И пусть **m** (количество бакетов) - некоторое простое число.

Тогда для любого $a = (a_1, a_2, a_4, a_4) : a_i \in \{0, 1, \dots, m-1\}$
введем $h_a(x) = (a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4) \, mod \, m$

Получили семейство хеш-функций $h_a(x) : U \to \{0, 1, 2, \dots, m-1\}$

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4) \, mod \, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4) \, mod \, m$$

Универсальное хеширование: пример

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr\limits_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4)\, mod\, m$$

Универсальное хеширование: пример

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr\limits_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\ mod\ m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4)\ mod\ m$$

$$a_4(x_4 - y_4)\ mod\ m = \sum_{i=1}^{3} a_i(y_i - x_i)\ mod\ m$$

Универсальное хеширование: пример

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr\limits_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4)\, mod\, m$$

$$a_4(x_4 - y_4)\, mod\, m = \sum_{i=1}^{3} a_i(y_i - x_i)\, mod\, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Универсальное хеширование: пример

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4)\, mod\, m$$

$$a_4(x_4 - y_4)\, mod\, m = \sum_{i=1}^{3} a_i(y_i - x_i)\, mod\, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p)\, mod\, m = k$, где p, k - константы < **m**
($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Универсальное хеширование: пример

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr\limits_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4) \, mod \, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4) \, mod \, m$$

$$a_4(x_4 - y_4) \, mod \, m = \sum_{i=1}^{3} a_i(y_i - x_i) \, mod \, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p) \, mod \, m = k$, где p, k - константы < **m**
($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Если мы выбираем a_4 абсолютно случайно из [0; m), сколько может быть значений у выражения: $a_4 \, mod \, m$?

Универсальное хеширование: пример

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4) \, mod \, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4) \, mod \, m$$

$$a_4(x_4 - y_4) \, mod \, m = \sum_{i=1}^{3} a_i(y_i - x_i) \, mod \, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p) \, mod \, m = k$, где p, k - константы < **m**
($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Если мы выбираем a_4 абсолютно случайно из [0; m), сколько может быть значений у выражения: $a_4 \, mod \, m$? m - штук!

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4) \, mod \, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4) \, mod \, m$$

$$a_4(x_4 - y_4) \, mod \, m = \sum_{i=1}^{3} a_i(y_i - x_i) \, mod \, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p) \, mod \, m = k$, где p, k - константы < **m**
($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Если мы выбираем a_4 абсолютно случайно из [0; m), сколько может быть значений у выражения: $a_4 \, mod \, m$? m - штук!

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4)\, mod\, m$$

$$a_4(x_4 - y_4)\, mod\, m = \sum_{i=1}^{3} a_i(y_i - x_i)\, mod\, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p)\, mod\, m = k$, где p, k - константы < **m**
($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Если мы выбираем a_4 абсолютно случайно из [0; m), сколько может быть значений у выражения: $(a_4 * p)\, mod\, m$? С учетом того, что **p < m**?

Универсальное хеширование: пример

Пусть U - множество IP-адресов: $(x_1, x_2, x_3, x_4) : x_i \in \{0, 1, \ldots, 255\}$

И пусть **m** (количество бакетов) - некоторое простое число.

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr\limits_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4) \, mod \, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4) \, mod \, m$$

$$a_4(x_4 - y_4) \, mod \, m = \sum_{i=1}^{3} a_i(y_i - x_i) \, mod \, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p) \, mod \, m = k$, где p, k - константы < **m**
($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Если мы выбираем a_4 абсолютно случайно из [0; m), сколько может быть значений у выражения: $(a_4 * p) \, mod \, m$? С учетом того, что **p < m**?

Маленькая теорема из теории чисел: $gcd(a_4, m) = 1; gcd(p, m) = 1$
$\Rightarrow gcd(a_4 * p, m) = 1$

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr\limits_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4) \, mod \, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4) \, mod \, m$$

$$a_4(x_4 - y_4) \, mod \, m = \sum_{i=1}^{3} a_i(y_i - x_i) \, mod \, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p) \, mod \, m = k$, где p, k - константы < **m** ($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Если мы выбираем a_4 абсолютно случайно из [0; m), сколько может быть значений у выражения: $(a_4 * p) \, mod \, m$? С учетом того, что **p < m**? Тоже m штук!

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr\limits_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4)\, mod\, m$$

$$a_4(x_4 - y_4)\, mod\, m = \sum_{i=1}^{3} a_i(y_i - x_i)\, mod\, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p)\, mod\, m = k$, где p, k - константы < **m** ($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Если мы выбираем a_4 абсолютно случайно из [0; m), сколько может быть значений у выражения: $(a_4 * p)\, mod\, m$? С учетом того, что **p < m**? Тоже m штук!

Тогда $Pr[(a_4 * p)\, mod\, m = k] = \frac{1}{m}$

Утверждение: $\mathcal{H} = \{h_a | a_1, a_2, a_3, a_4 \in \{0, 1, .., m-1\}\}$ - универсально.

Доказательство: зафиксируем два разных ключа $(x_1, x_2, x_3, x_4) \neq (y_1, y_2, y_3, y_4)$
Пусть для определенности $x_4 \neq y_4$; Посчитаем вероятность коллизии:
$\Pr\limits_{h \in \mathcal{H}}[h(x_1, x_2, x_3, x_4) = h(y_1, y_2, y_3, y_4)]$; т.е.

$$(a_1 * x_1 + a_2 * x_2 + a_3 * x_3 + a_4 * x_4)\, mod\, m = (a_1 * y_1 + a_2 * y_2 + a_3 * y_3 + a_4 * y_4)\, mod\, m$$

$$a_4(x_4 - y_4)\, mod\, m = \sum_{i=1}^{3} a_i(y_i - x_i)\, mod\, m$$

Теперь фиксируем a_1, a_2, a_3 (считаем, что случайный выбор уже произошел, последняя случайная величина - a_4)

Тогда получаем: $(a_4 * p)\, mod\, m = k$, где p, k - константы < **m** ($-255 \leq x_4 - y_4 \leq 255$, возьмем m больше)

Если мы выбираем a_4 абсолютно случайно из [0; m), сколько может быть значений у выражения: $(a_4 * p)\, mod\, m$? С учетом того, что **p < m**? Тоже m штук!

Тогда $Pr[(a_4 * p)\, mod\, m = k] = \frac{1}{m}$ => вероятность коллизии - $\frac{1}{m}$ => $\mathcal{H}$ - универсально $\square$

Универсальное хеширование: сложность

Теорема: Пусть есть хеш-таблица, в которой коллизии разрешаются методом цепочек,

И пусть $\mathcal{H}$ - универсальное множество хеш-функций,

$n_i -$ длина i-ого списка в таблице, α - коэфф. заполненности.

Универсальное хеширование: сложность

Теорема: Пусть есть хеш-таблица, в которой коллизии разрешаются методом цепочек,

И пусть $\mathcal{H}$ - универсальное множество хеш-функций,

$n_i -$ длина i-ого списка в таблице, α - коэфф. заполненности.

Тогда: $E[n_{h(k)}] \le \alpha$, если ключа k в таблице нет и

$E[n_{h(k)}] \le 1 + \alpha$, если ключ k в таблице есть,

где h выбирается случайным образом из $\mathcal{H}$

Теорема: Пусть есть хеш-таблица, в которой коллизии разрешаются методом цепочек,

И пусть $\mathcal{H}$ - универсальное множество хеш-функций,

$n_i -$ длина i-ого списка в таблице, α - коэфф. заполненности.

Тогда: $E[n_{h(k)}] \leq \alpha$, если ключа k в таблице нет и

$E[n_{h(k)}] \leq 1 + \alpha$, если ключ k в таблице есть,

где h выбирается случайным образом из $\mathcal{H}$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

Теорема: Пусть есть хеш-таблица, в которой коллизии разрешаются методом цепочек,

И пусть $\mathcal{H}$ - универсальное множество хеш-функций,

$n_i -$ длина i-ого списка в таблице, α - коэфф. заполненности.

Тогда: $E[n_{h(k)}] \leq \alpha$, если ключа k в таблице нет и

$E[n_{h(k)}] \leq 1 + \alpha$, если ключ k в таблице есть,

где h выбирается случайным образом из $\mathcal{H}$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

$Pr[h(k) = h(l)] \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq \frac{1}{m}$

Теорема: Пусть есть хеш-таблица, в которой коллизии разрешаются методом цепочек,

И пусть $\mathcal{H}$ - универсальное множество хеш-функций,

$n_i -$ длина i-ого списка в таблице, α - коэфф. заполненности.

Тогда: $E[n_{h(k)}] \leq \alpha$, если ключа k в таблице нет и

$E[n_{h(k)}] \leq 1 + \alpha$, если ключ k в таблице есть,

где h выбирается случайным образом из $\mathcal{H}$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

$Pr[h(k) = h(l)] \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq \frac{1}{m}$

Теперь введем Y_k - количество ключей, отличных от k, но хешируемых в тот же бакет. Тогда $Y_k = \sum\limits_{l \neq k} X_{kl}$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

$Pr[h(k) = h(l)] \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq \frac{1}{m}$

Теперь введем Y_k - количество ключей, отличных от k, но хешируемых в тот же бакет. Тогда $Y_k = \sum_{l \neq k} X_{kl}$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

$Pr[h(k) = h(l)] \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq \frac{1}{m}$

Теперь введем Y_k - количество ключей, отличных от k, но хешируемых в тот же бакет. Тогда $Y_k = \sum_{l \neq k} X_{kl}$

$$E[Y_k] = E[\sum_{l \neq k} X_{kl}] = \sum_{l \neq k} E[X_{kl}] \leq \sum_{l \neq k} \frac{1}{m}$$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

$Pr[h(k) = h(l)] \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq \frac{1}{m}$

Теперь введем Y_k - количество ключей, отличных от k, но хешируемых в тот же бакет. Тогда $Y_k = \sum_{l \neq k} X_{kl}$

$$E[Y_k] = E[\sum_{l \neq k} X_{kl}] = \sum_{l \neq k} E[X_{kl}] \leq \sum_{l \neq k} \frac{1}{m}$$

Дальше два варианта:

1) Если элемента k еще не было в таблице, то: $n_{h(k)} = Y_k;$

$$|l : l \in U \text{ и } l \neq k| = n$$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

$Pr[h(k) = h(l)] \le \frac{1}{m} \Rightarrow E[X_{kl}] \le \frac{1}{m}$

Теперь введем Y_k - количество ключей, отличных от k, но хешируемых в тот же бакет. Тогда $Y_k = \sum\limits_{l \ne k} X_{kl}$

$$E[Y_k] = E[\sum_{l \ne k} X_{kl}] = \sum_{l \ne k} E[X_{kl}] \le \sum_{l \ne k} \frac{1}{m}$$

Дальше два варианта:

1) Если элемента k еще не было в таблице, то: $n_{h(k)} = Y_k;$

$|l : l \in U \text{ и } l \ne k| = n$

Тогда: $E[n_{h(k)}] = E[Y_k] \le n/m = \alpha$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

$Pr[h(k) = h(l)] \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq \frac{1}{m}$

Теперь введем Y_k - количество ключей, отличных от k, но хешируемых в тот же бакет. Тогда $Y_k = \sum\limits_{l \neq k} X_{kl}$

$$E[Y_k] = E[\sum_{l \neq k} X_{kl}] = \sum_{l \neq k} E[X_{kl}] \leq \sum_{l \neq k} \frac{1}{m}$$

Дальше два варианта:

1) Если элемента k еще не было в таблице, то: $n_{h(k)} = Y_k$;

   $|l : l \in U \text{ и } l \neq k| = n$

   Тогда: $E[n_{h(k)}] = E[Y_k] \leq n/m = \alpha$

2) Если элемент k уже был в таблице, то: $n_{h(k)} = Y_k + 1$

   $|l : l \in U \text{ и } l \neq k| = n - 1$

Доказательство: для каждой пары ключей k и l введем индикаторную случайную величину: $X_{kl} = \begin{cases} 1, \text{если } h(k) = h(l) \\ 0, \text{иначе} \end{cases}$

$Pr[h(k) = h(l)] \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq \frac{1}{m}$

Теперь введем Y_k - количество ключей, отличных от k, но хешируемых в тот же бакет. Тогда $Y_k = \sum_{l \neq k} X_{kl}$

$$E[Y_k] = E[\sum_{l \neq k} X_{kl}] = \sum_{l \neq k} E[X_{kl}] \leq \sum_{l \neq k} \frac{1}{m}$$

Дальше два варианта:

1) Если элемента k еще не было в таблице, то: $n_{h(k)} = Y_k;$

$|l : l \in U \text{ и } l \neq k| = n$

Тогда: $E[n_{h(k)}] = E[Y_k] \leq n/m = \alpha$

2) Если элемент k уже был в таблице, то: $n_{h(k)} = Y_k + 1$

$|l : l \in U \text{ и } l \neq k| = n - 1$

Тогда: $E[n_{h(k)}] = E[Y_k] + 1 \leq \frac{n-1}{m} + 1 =$

$1 + \alpha - 1/m < 1 + \alpha \quad \square$

Универсальное хеширование: сложность

Теорема: Пусть есть хеш-таблица, в которой коллизии разрешаются методом цепочек,

И пусть $\mathcal{H}$ - универсальное множество хеш-функций,

$n_i -$ длина i-ого списка в таблице, α - коэфф. заполненности.

Тогда: $E[n_{h(k)}] \leq \alpha$, если ключа k в таблице нет и

$E[n_{h(k)}] \leq 1 + \alpha$, если ключ k в таблице есть,

где h выбирается случайным образом из $\mathcal{H}$

Универсальное хеширование: сложность

Теорема: Пусть есть хеш-таблица, в которой коллизии разрешаются методом цепочек,

И пусть $\mathcal{H}$ - универсальное множество хеш-функций,

$n_i -$ длина i-ого списка в таблице, α - коэфф. заполненности.

Тогда: $E[n_{h(k)}] \leq \alpha$, если ключа k в таблице нет и

$E[n_{h(k)}] \leq 1 + \alpha$, если ключ k в таблице есть,

где h выбирается случайным образом из $\mathcal{H}$

Следствие: если при создании хеш-таблицы выбирать хеш-функцию случайным образом из универсального множества, время работы поиска и добавления в среднем* останется O(1).

Открытая адресация: типы исследования

Хеш-таблицы: метод открытой адресации

```
insert("Кузнецов")
h("Кузнецов") = 6
```

Коллизия!

Что делать?

В массиве занято -> ищем следующее свободное место по некоторому правилу.

T

T[0]	
T[1]	
T[2]	
T[3]	"Петров"
T[4]	
T[5]	
T[6]	"Иванов"
T[7]	"Кузнецов"
T[8]	"Сидорова"
T[9]	
T[10]	

Давайте хранить в массиве все-таки именно что элементы

Линейное исследование: ищем просто следующую пустую ячейку.

(есть и другие варианты)

Хеш-таблицы: метод открытой адресации

Чему при равномерном хешировании равна вероятность попасть в ячейку 9?

T

T[0]	
T[1]	
T[2]	
T[3]	"Петров"
T[4]	
T[5]	
T[6]	"Иванов"
T[7]	"Кузнецов"
T[8]	"Сидорова"
T[9]	
T[10]	

Давайте хранить в массиве все-таки именно что элементы

Линейное исследование: ищем просто следующую пустую ячейку.

(есть и другие варианты)

Хеш-таблицы: метод открытой адресации

Чему при равномерном хешировании равна вероятность попасть в ячейку 9?

4/11 (т.к. попадание в любой из трех занятых бакетов приведет к этому)

T

T[0]	
T[1]	
T[2]	
T[3]	"Петров"
T[4]	
T[5]	
T[6]	"Иванов"
T[7]	"Кузнецов"
T[8]	"Сидорова"
T[9]	
T[10]	

Давайте хранить в массиве все-таки именно что элементы

Линейное исследование: ищем просто следующую пустую ячейку.

(есть и другие варианты)

Хеш-таблицы: метод открытой адресации

Чему при равномерном хешировании равна вероятность попасть в ячейку 9?

4/11 (т.к. попадание в любой из трех занятых бакетов приведет к этому)

И чем больше цепочка, тем больше шанс.

Замкнутый круг.

	T
T[0]	
T[1]	
T[2]	
T[3]	"Петров"
T[4]	
T[5]	
T[6]	"Иванов"
T[7]	"Кузнецов"
T[8]	"Сидорова"
T[9]	
T[10]	

Давайте хранить в массиве все-таки именно что элементы

Линейное исследование: ищем просто следующую пустую ячейку.

(есть и другие варианты)

Хеш-таблицы: метод открытой адресации

Чему при равномерном хешировании равна вероятность попасть в ячейку 9?

4/11 (т.к. попадание в любой из трех занятых бакетов приведет к этому)

И чем больше цепочка, тем больше шанс.

Замкнутый круг.

T

T[0]	
T[1]	
T[2]	
T[3]	"Петров"
T[4]	
T[5]	
T[6]	"Иванов"
T[7]	"Кузнецов"
T[8]	"Сидорова"
T[9]	
T[10]	

Это приводит к "первичной кластеризации" - ситуации, когда элементы "сбились" в одну цепочку.

Хеш-таблицы: метод открытой адресации

Чему при равномерном хешировании равна вероятность попасть в ячейку 9?

4/11 (т.к. попадание в любой из трех занятых бакетов приведет к этому)

И чем больше цепочка, тем больше шанс.

Замкнутый круг.

T

T[0]	
T[1]	
T[2]	
T[3]	"Петров"
T[4]	
T[5]	
T[6]	"Иванов"
T[7]	"Кузнецов"
T[8]	"Сидорова"
T[9]	
T[10]	

Это приводит к "первичной кластеризации" - ситуации, когда элементы "сбились" в одну цепочку.

Нужны другие способы исследования, кроме линейного.

Хеш-таблицы: метод открытой адресации

Формальное определение хеш-функции для линейного исследования:

$h(k,i) = (h'(k) + i)\, mod\, m$, где $h'(k)$ - вспомогательная хеш-функция,
i - номер попытки исследования.

Хеш-таблицы: метод открытой адресации

Формальное определение хеш-функции для линейного исследования:

$h(k, i) = (h'(k) + i) \, mod \, m$, где $h'(k)$ - вспомогательная хеш-функция,
i - номер попытки исследования.

Альтернатива: квадратичное исследование:

$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \, mod \, m$,

Хеш-таблицы: метод открытой адресации

Формальное определение хеш-функции для линейного исследования:

$h(k,i) = (h'(k) + i)\, mod\, m$, где $h'(k)$ - вспомогательная хеш-функция,
i - номер попытки исследования.

Альтернатива: квадратичное исследование:

$h(k,i) = (h'(k) + c_1 i + c_2 i^2)\, mod\, m$, где c_1 и c_2 - некоторые константы, подобранные так, чтобы все элементы таблицы были охвачены

Хеш-таблицы: метод открытой адресации

Формальное определение хеш-функции для линейного исследования:

$h(k,i) = (h'(k) + i)\, mod\, m$, где $h'(k)$ - вспомогательная хеш-функция,
i - номер попытки исследования.

Альтернатива: квадратичное исследование:

$h(k,i) = (h'(k) + c_1 i + c_2 i^2)\, mod\, m$, где c_1 и c_2 - некоторые константы, подобранные так, чтобы все элементы таблицы были охвачены

Проблема кластеризации остается (т.к. для настоящих коллизий вся цепочка исследований будет одинаковой), но проявляется в гораздо более мягкой форме.

Хеш-таблицы: метод открытой адресации

Формальное определение хеш-функции для линейного исследования:

$h(k,i) = (h'(k) + i)\, mod\, m$, где $h'(k)$ - вспомогательная хеш-функция,
i - номер попытки исследования.

Альтернатива: квадратичное исследование:

$h(k,i) = (h'(k) + c_1 i + c_2 i^2)\, mod\, m$, где c_1 и c_2 - некоторые константы, подобранные так, чтобы все элементы таблицы были охвачены

Альтернатива: двойное хеширование: $h(k,i) = (h_1(k) + i h_2(k))\, mod\, m$,

Используем два хеша, получаем один из лучших вариантов открытой адресации: кластеризация больше не проблема.

Хеш-таблицы: хеширование кукушки

Хеш-таблицы: хеширование кукушки

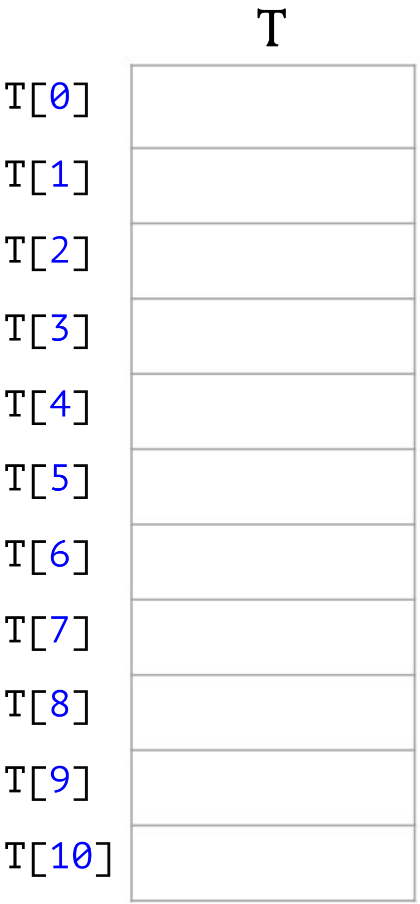
	T
T[0]	
T[1]	
T[2]	
T[3]	
T[4]	
T[5]	
T[6]	
T[7]	
T[8]	
T[9]	
T[10]	

Хеш-таблицы: хеширование кукушки

	T
T[0]	
T[1]	
T[2]	
T[3]	
T[4]	
T[5]	
T[6]	
T[7]	
T[8]	
T[9]	
T[10]	

Пусть h_1, h_2 - хеш-функции из универсального семейства.

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

Хеш-таблицы: хеширование кукушки

T	
T[0]	
T[1]	
T[2]	
T[3]	
T[4]	
T[5]	
T[6]	
T[7]	
T[8]	
T[9]	
T[10]	

Пусть h_1, h_2 - хеш-функции из универсального семейства.

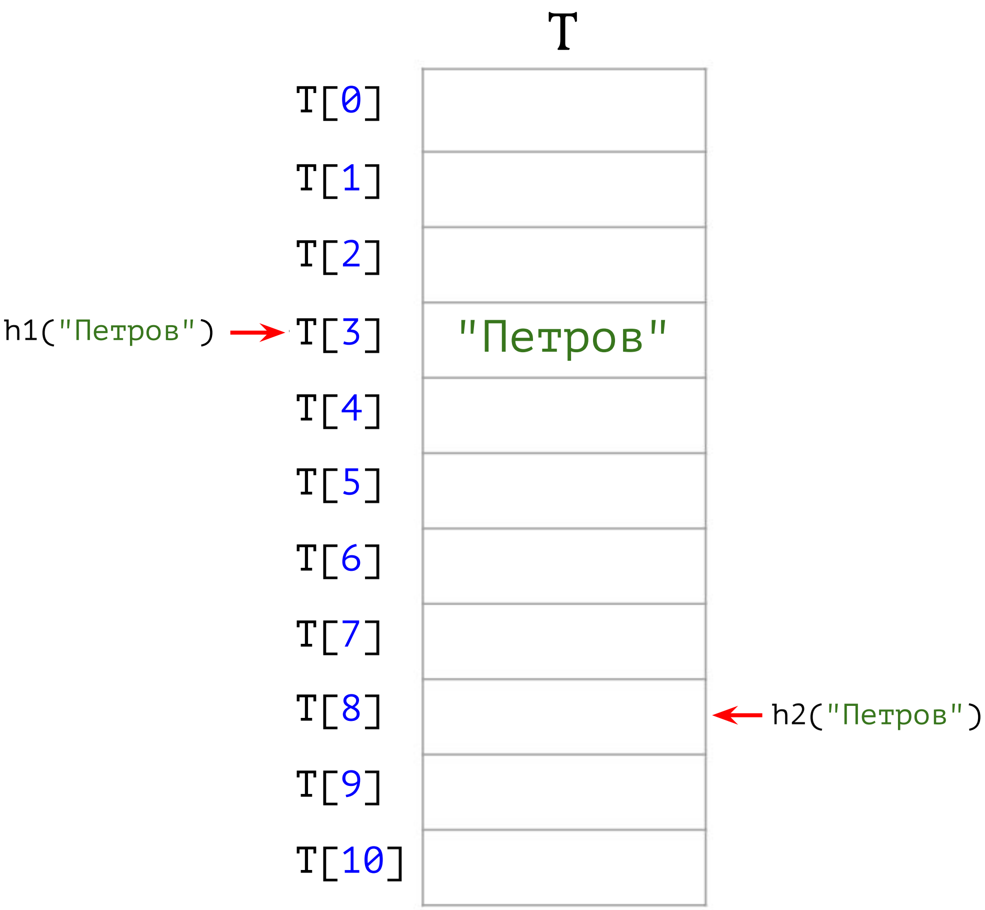
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

```
add("Петров"):

h1("Петров") = 3
h2("Петров") = 8
```

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

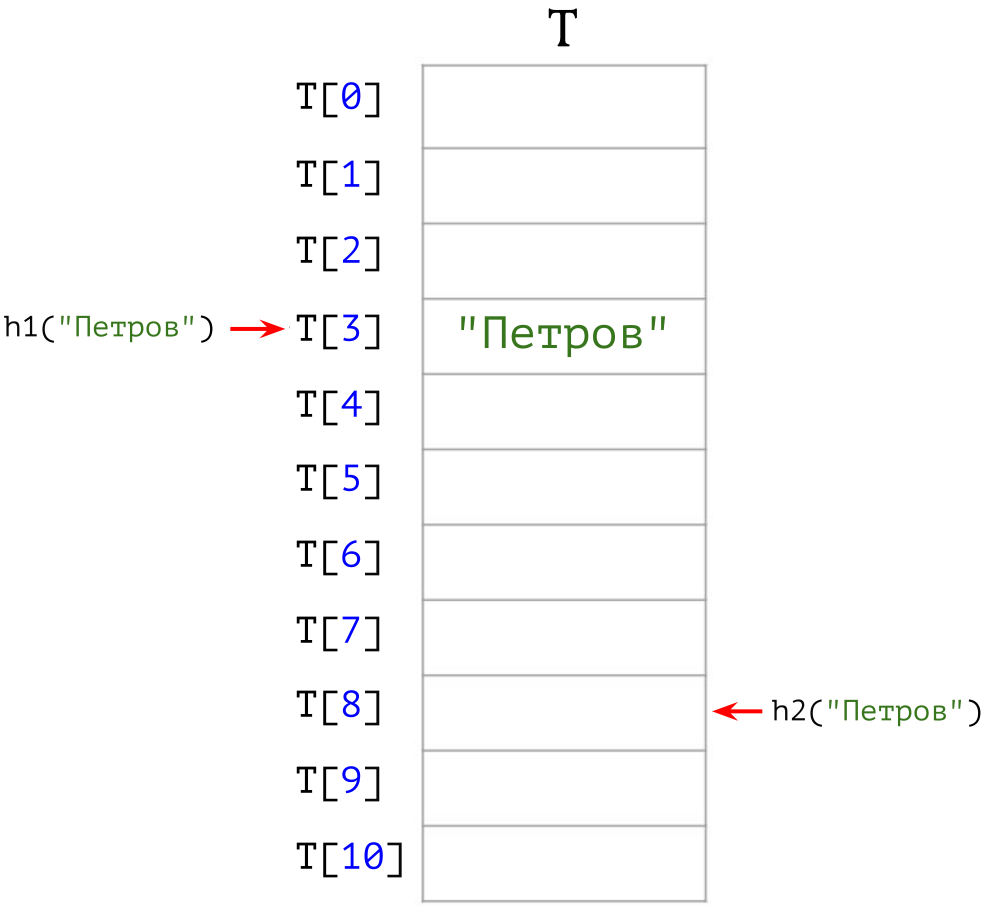
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

```
add("Петров"):

h1("Петров") = 3
h2("Петров") = 8
```

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

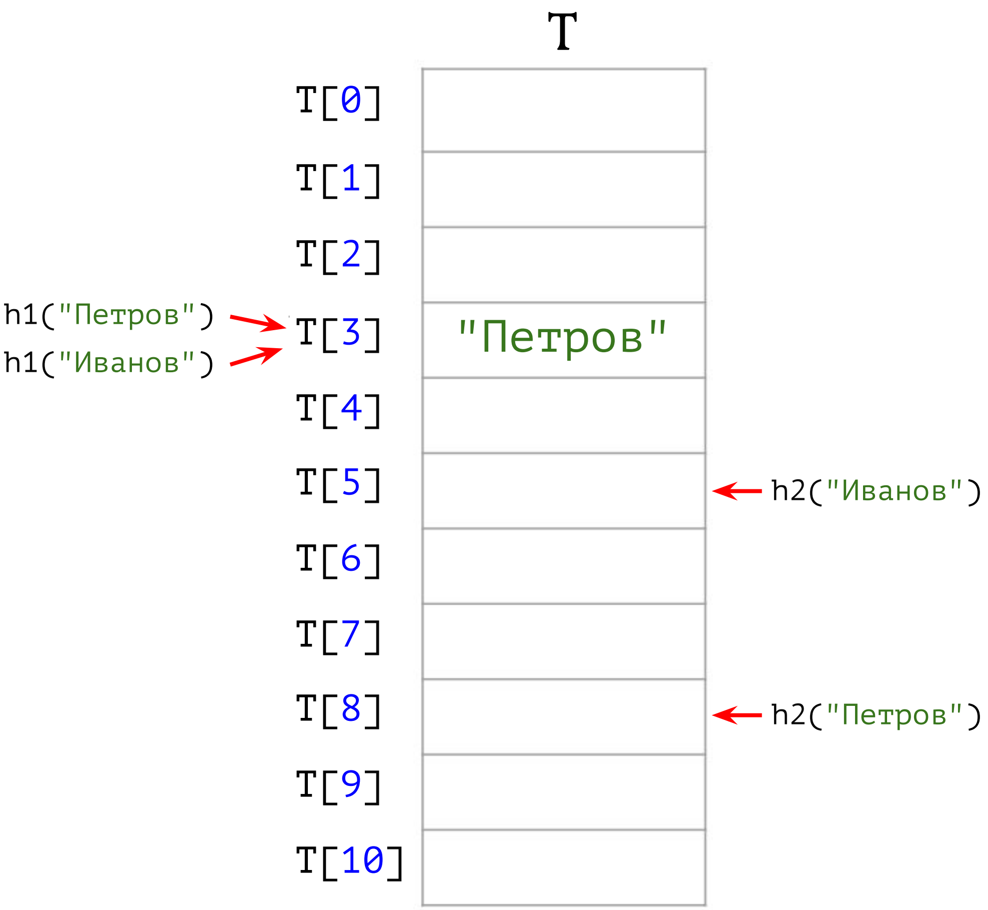
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

```
add("Иванов"):

h1("Иванов") = 3
h2("Иванов") = 5
```

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

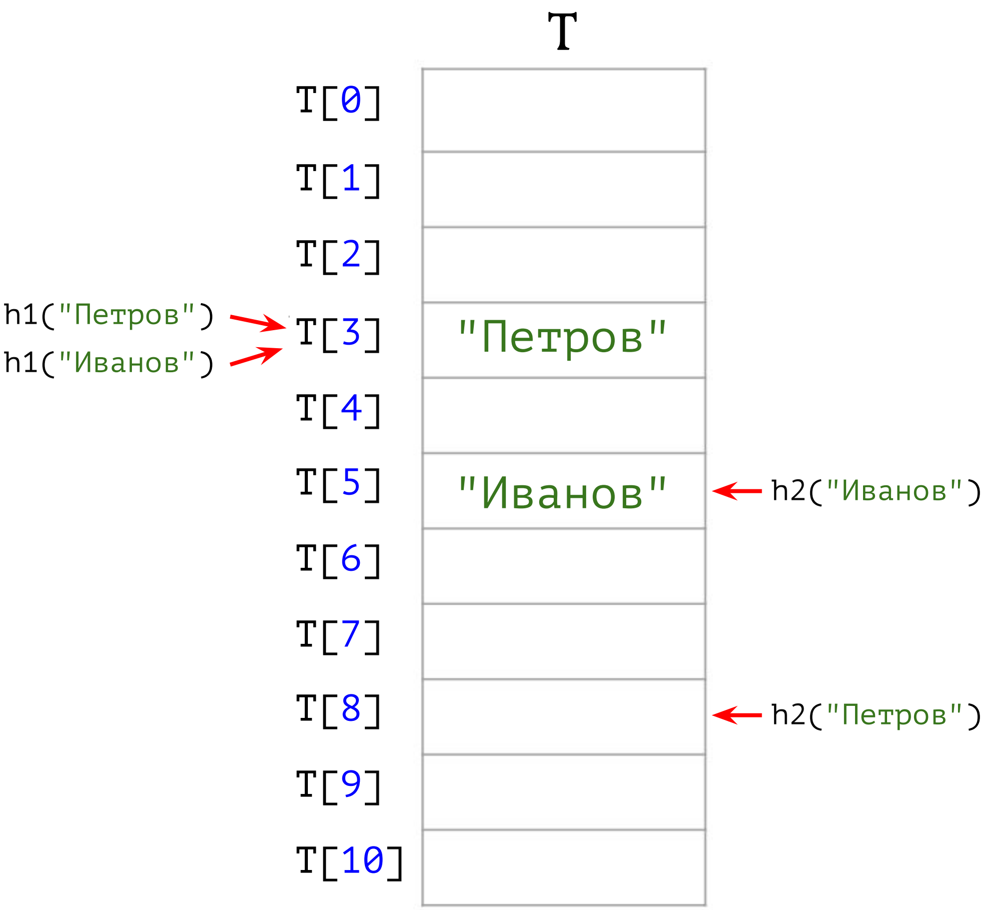
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

```
add("Иванов"):

h1("Иванов") = 3
h2("Иванов") = 5
```

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

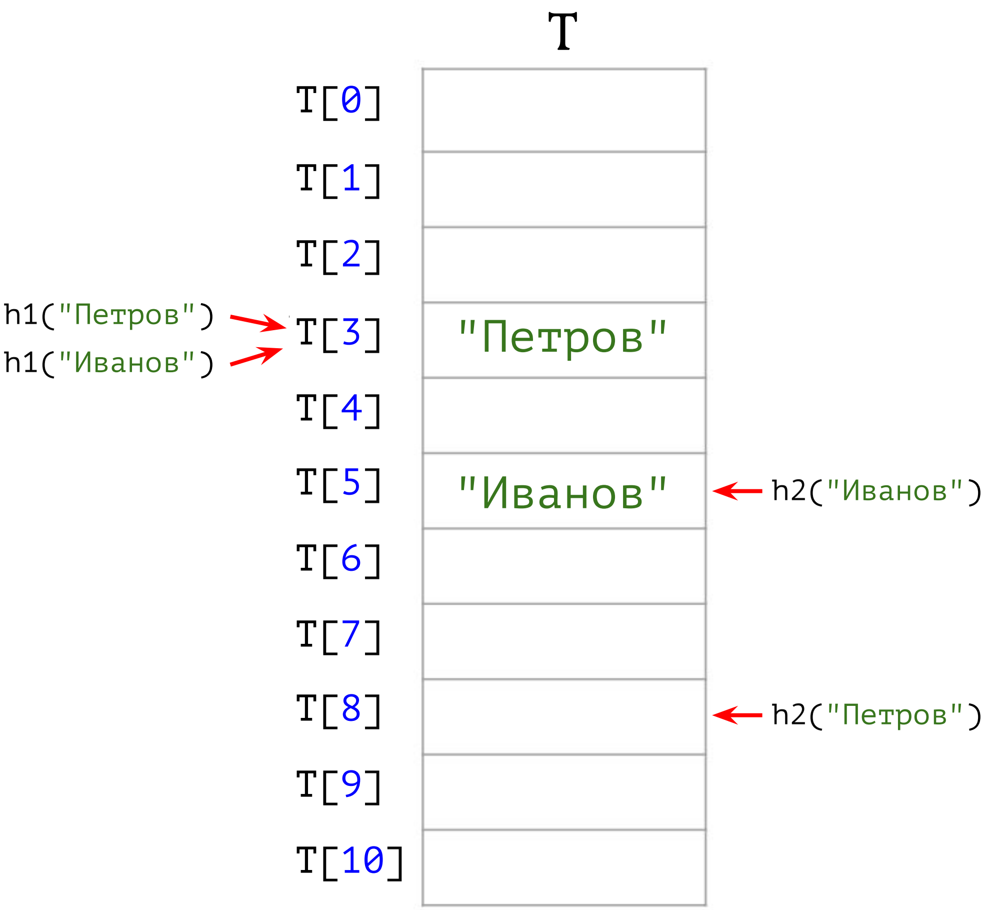
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

```
add("Иванов"):

h1("Иванов") = 3
h2("Иванов") = 5
```

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

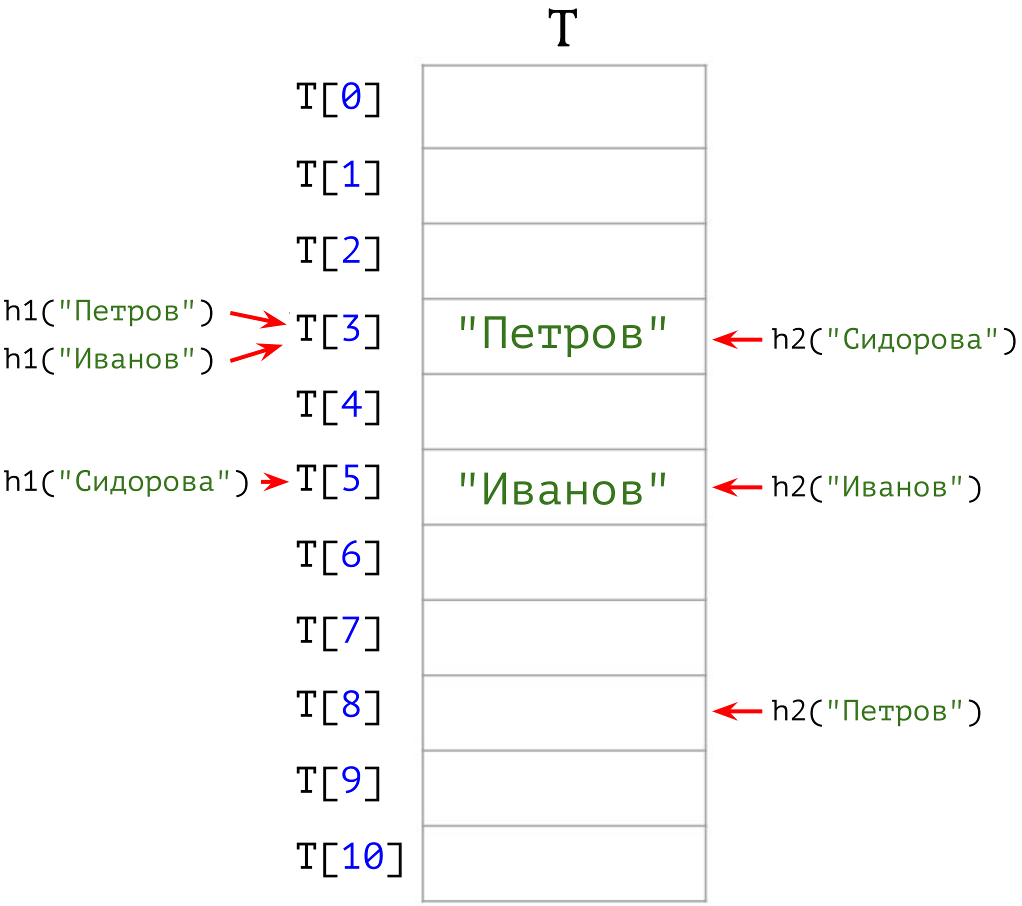
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

add("Сидорова"):

h1("Сидорова") = 5
h2("Сидорова") = 3

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

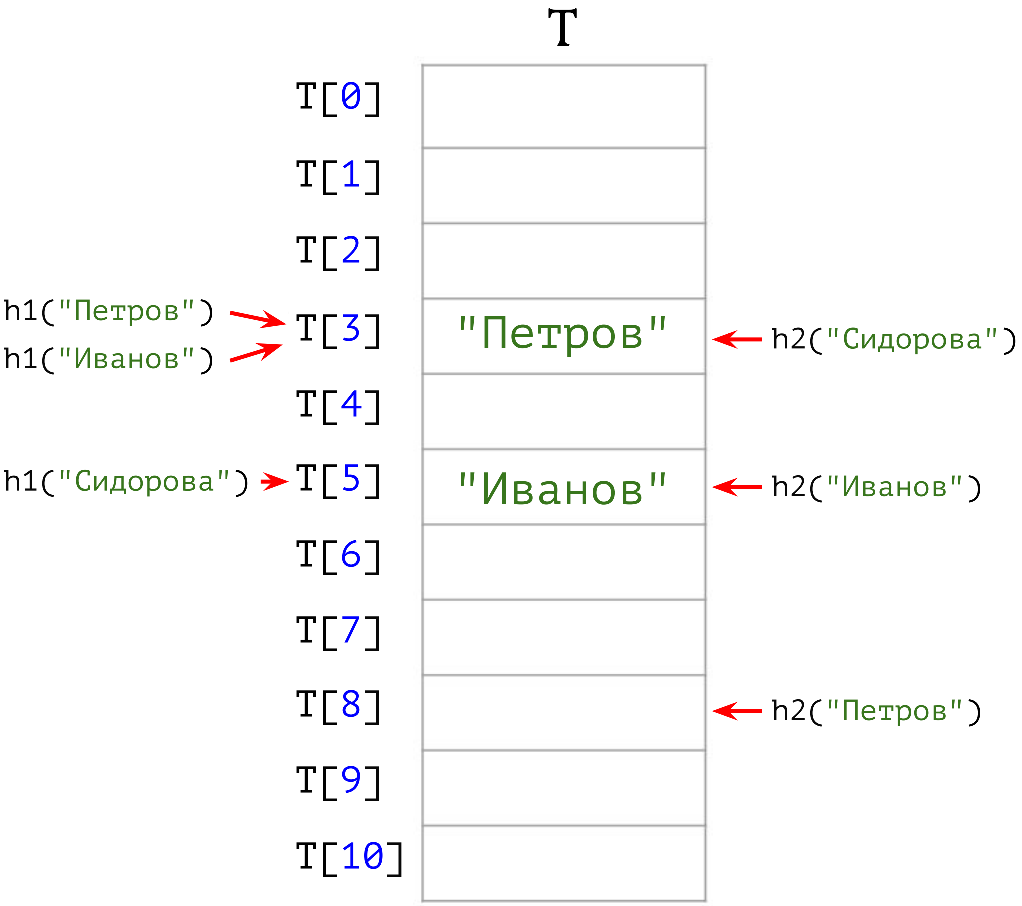
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

```
add("Сидорова"):

h1("Сидорова") = 5
h2("Сидорова") = 3
```

Хеш-таблицы: хеширование кукушки



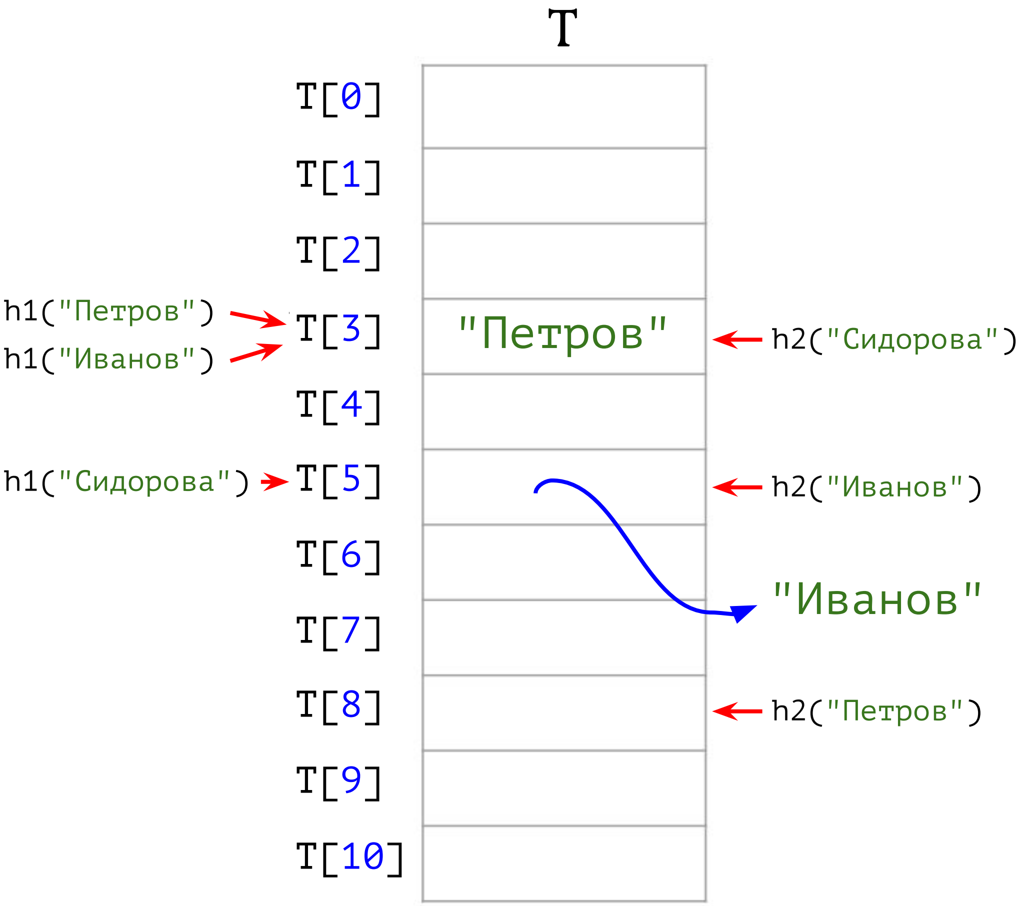
Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

2. Иначе: выбираем одну из позиций и "выталкиваем" того, кто ее занимает, из гнезда

Хеш-таблицы: хеширование кукушки



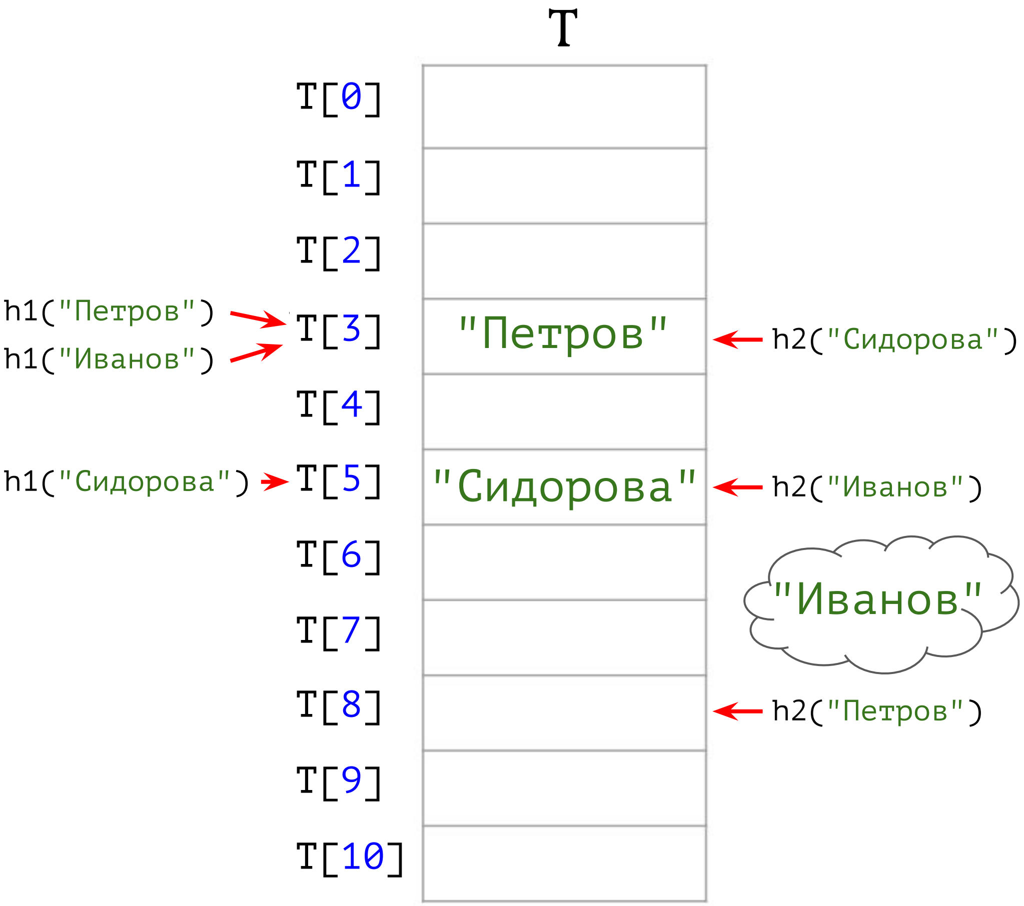
Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

2. Иначе: выбираем одну из позиций и "выталкиваем" того, кто ее занимает, из гнезда

Хеш-таблицы: хеширование кукушки



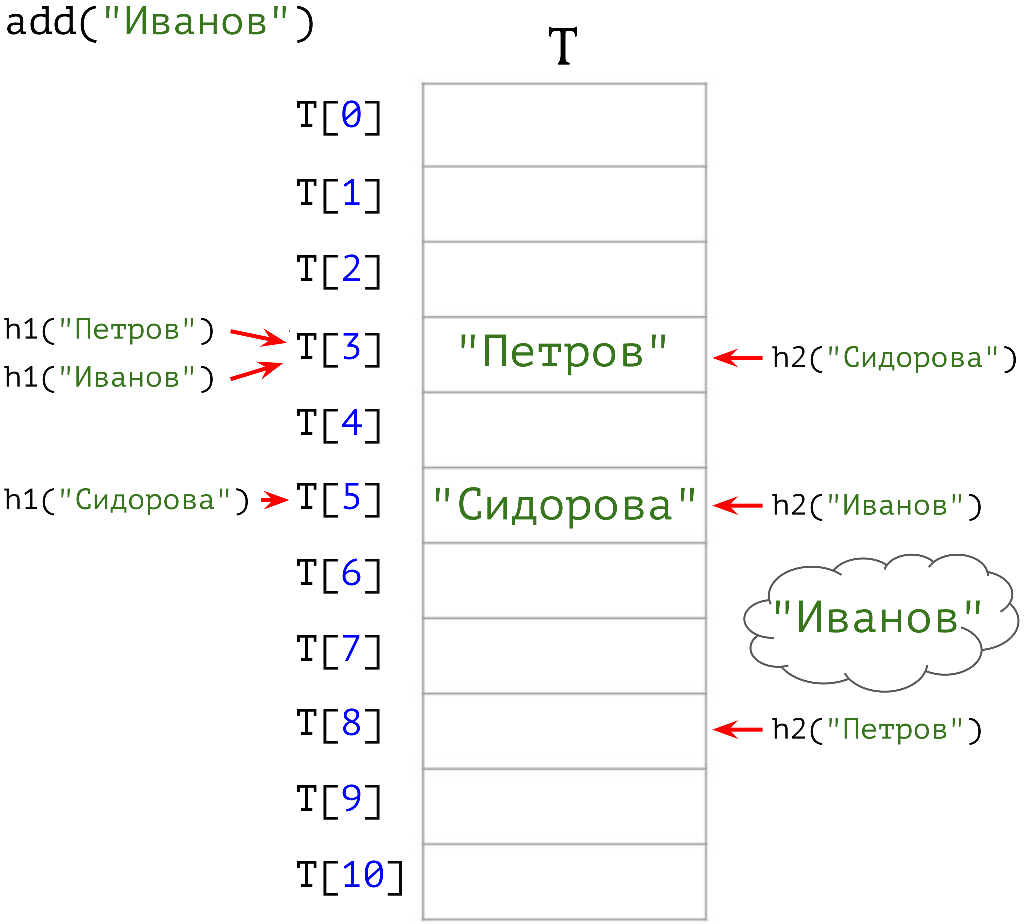
Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

2. Иначе: выбираем одну из позиций и "выталкиваем" того, кто ее занимает, из гнезда

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм добавления:

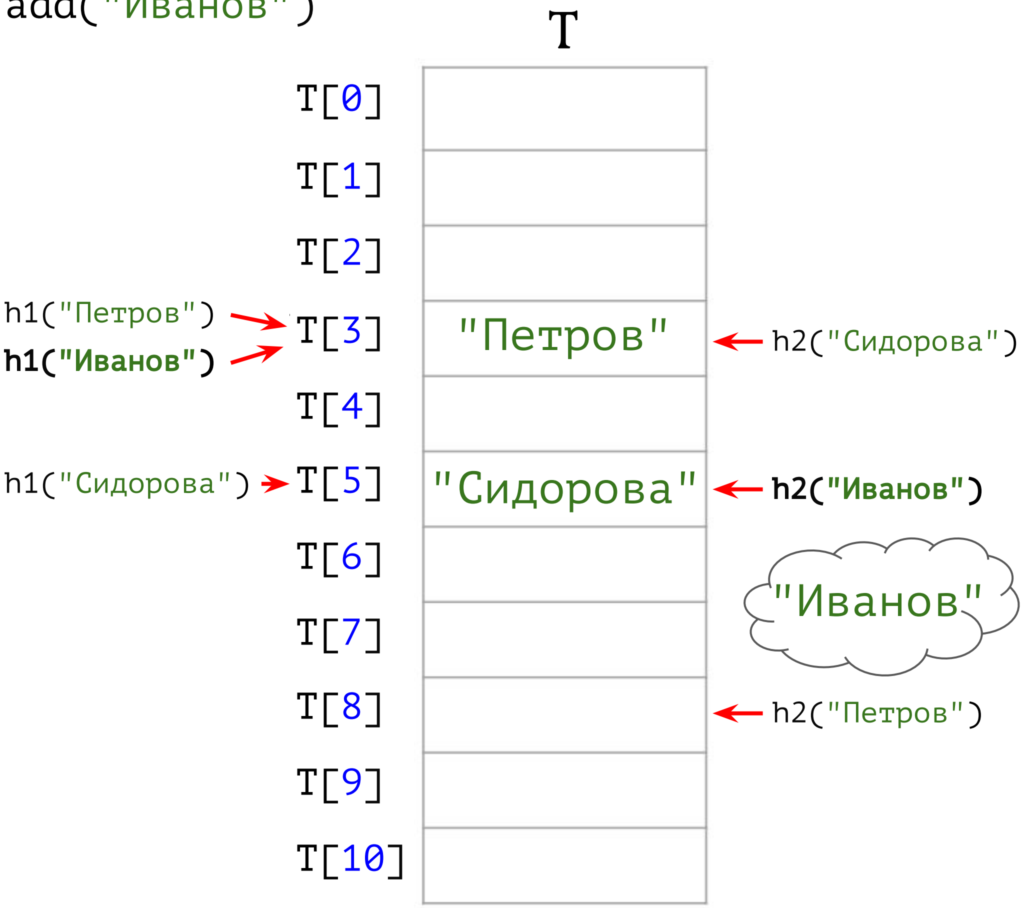
1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

2. Иначе: выбираем одну из позиций и "выталкиваем" того, кто ее занимает, из гнезда

3. Затем добавляем вытолкнутого птенца по той же процедуре

Хеш-таблицы: хеширование кукушки

add("Иванов")



Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм добавления:

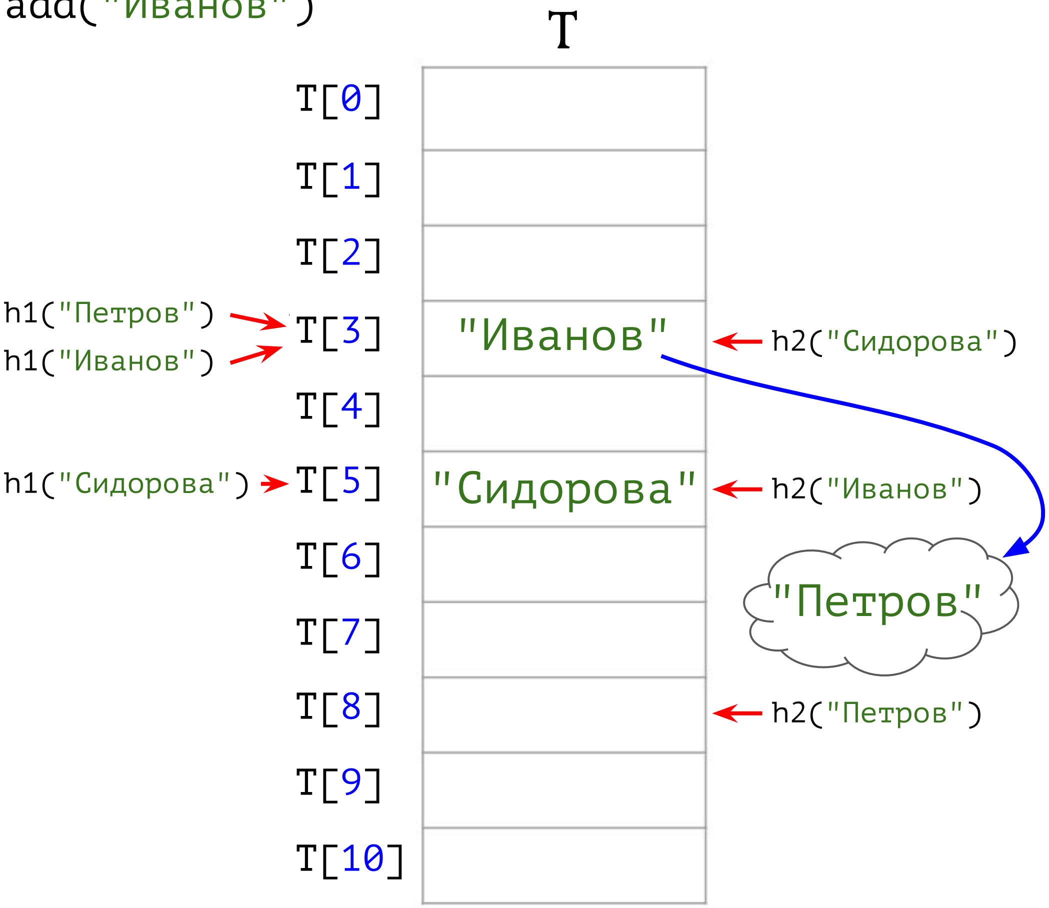
1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

2. Иначе: выбираем одну из позиций и "выталкиваем" того, кто ее занимает, из гнезда

3. Затем добавляем вытолкнутого птенца по той же процедуре

Хеш-таблицы: хеширование кукушки

add("Иванов")



Пусть h_1, h_2 - хеш-функции из универсального семейства.

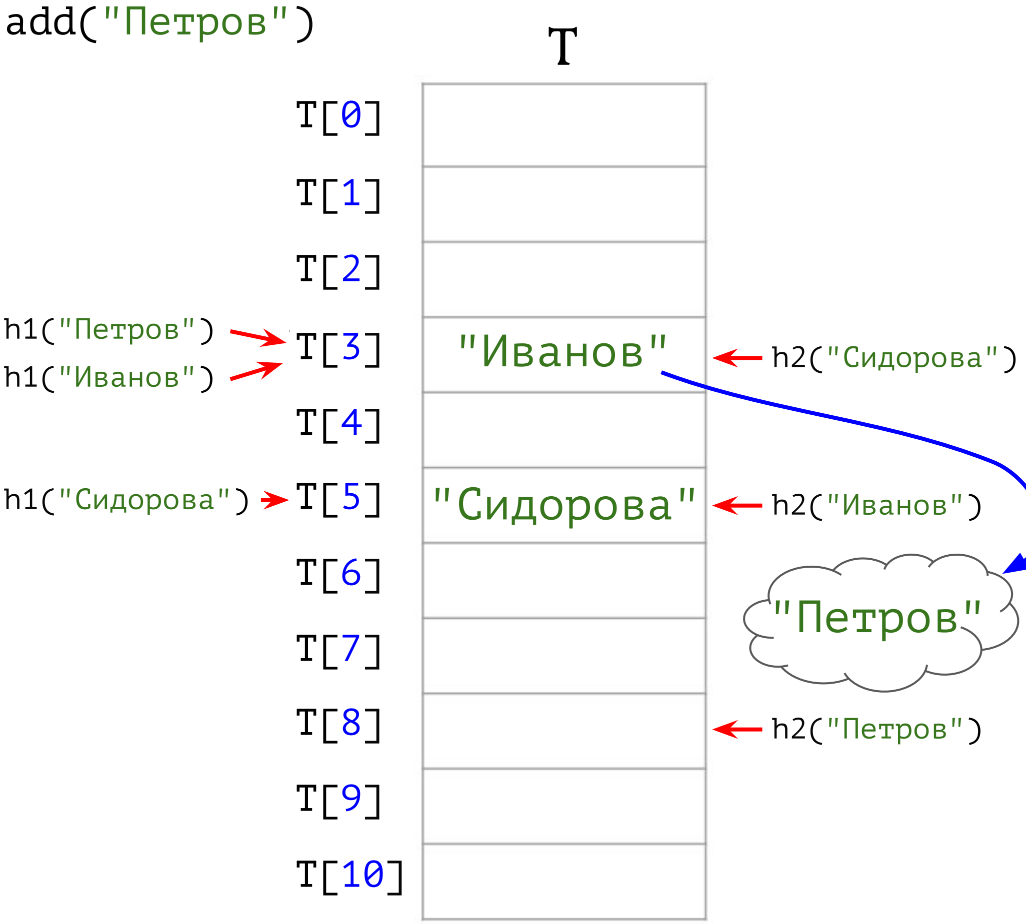
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

2. Иначе: выбираем одну из позиций и "выталкиваем" того, кто ее занимает, из гнезда

3. Затем добавляем вытолкнутого птенца по той же процедуре

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

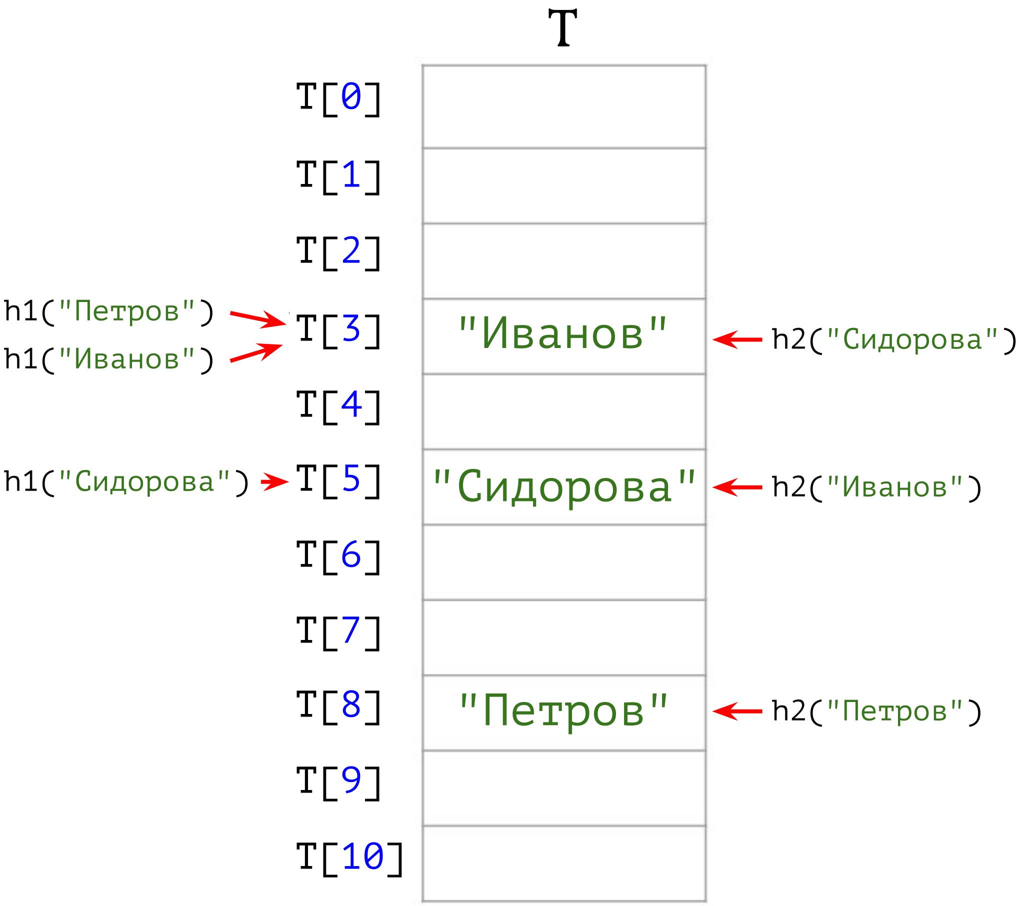
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

2. Иначе: выбираем одну из позиций и "выталкиваем" того, кто ее занимает, из гнезда

3. Затем добавляем вытолкнутого птенца по той же процедуре

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

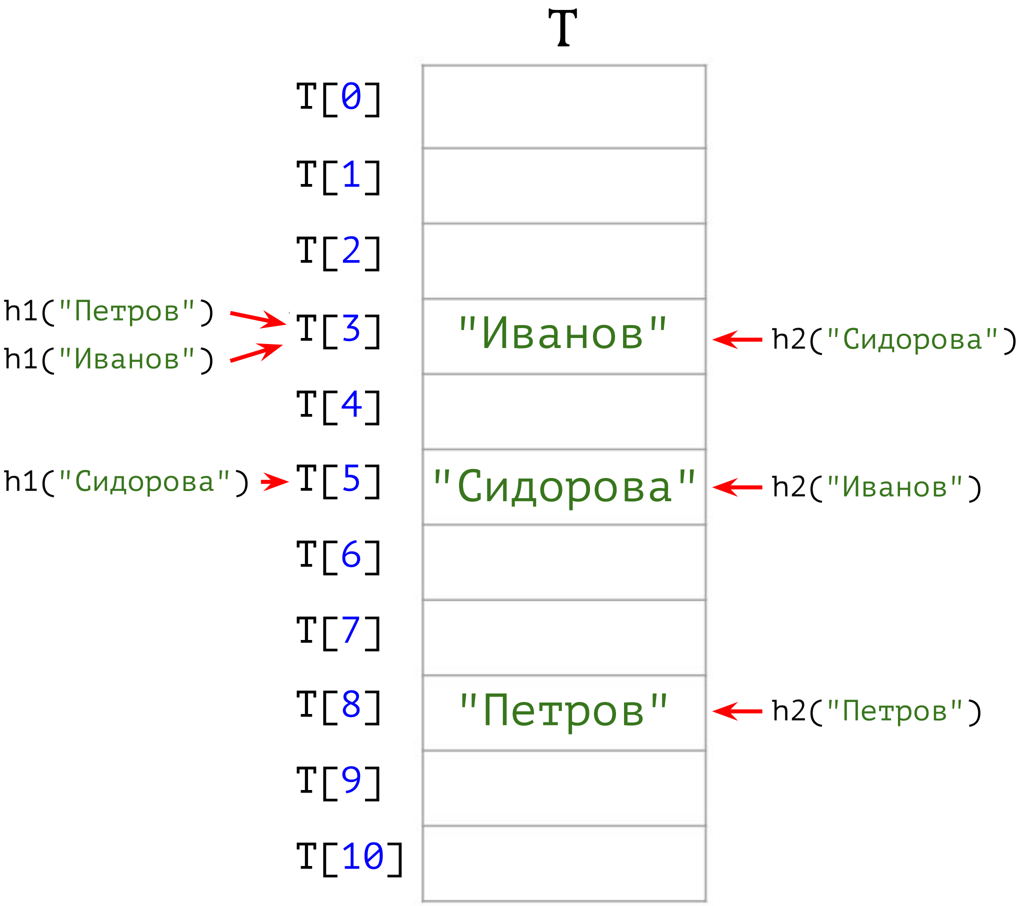
Алгоритм добавления:

1. Если h1(x) пусто, кладем элемент туда. Иначе, если h2(x) пусто - кладем туда.

2. Иначе: выбираем одну из позиций и "выталкиваем" того, кто ее занимает, из гнезда

3. Затем добавляем вытолкнутого птенца по той же процедуре

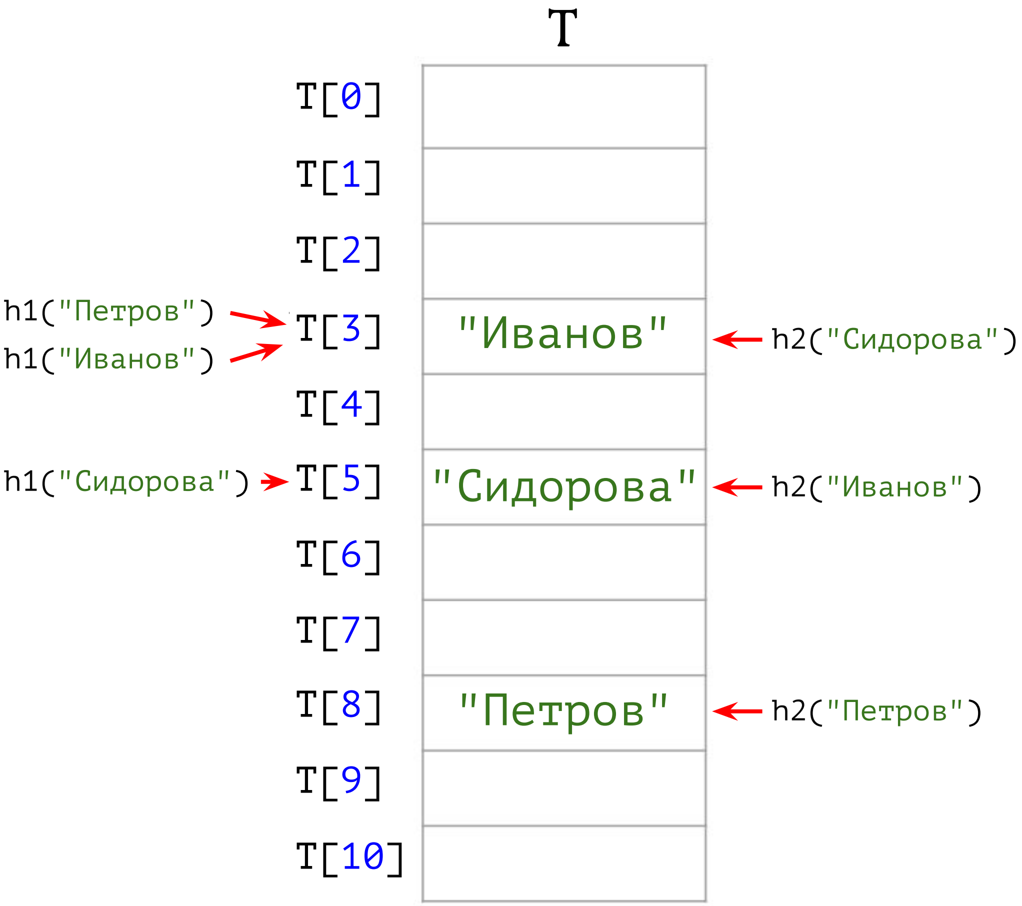
Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм поиска?

Хеш-таблицы: хеширование кукушки

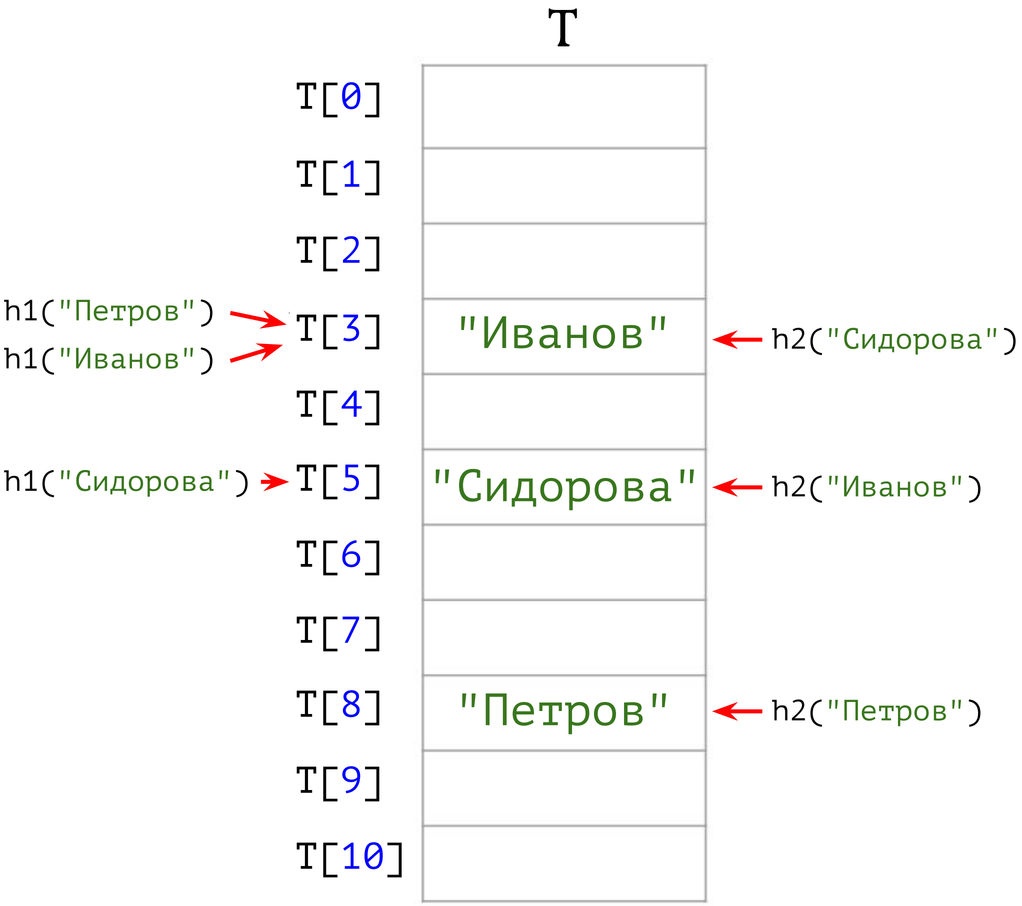


Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм поиска?

1. Проверяем h1(x). Если там наш элемент, то нашли

Хеш-таблицы: хеширование кукушки

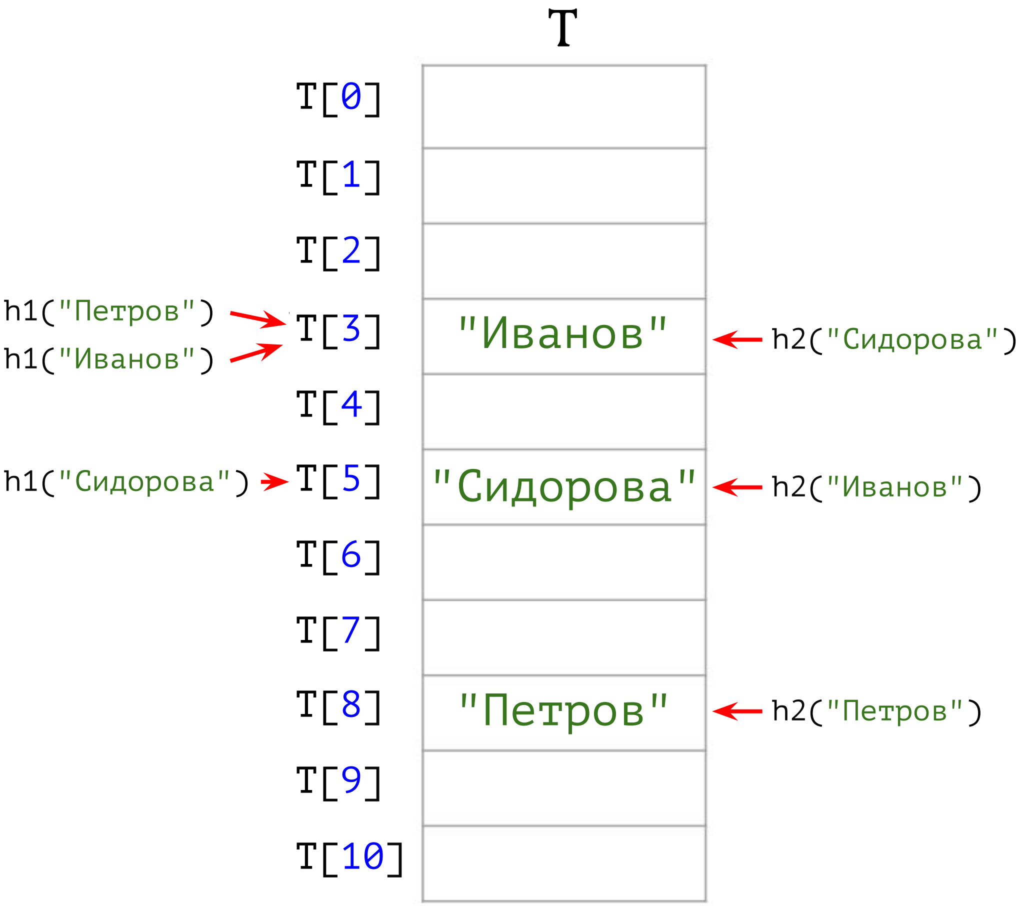


Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм поиска?

1. Проверяем h1(x). Если там наш элемент, то нашли
2. Проверяем h2(x). Если там наш элемент, то нашли.
3. Иначе, не нашли.

Хеш-таблицы: хеширование кукушки



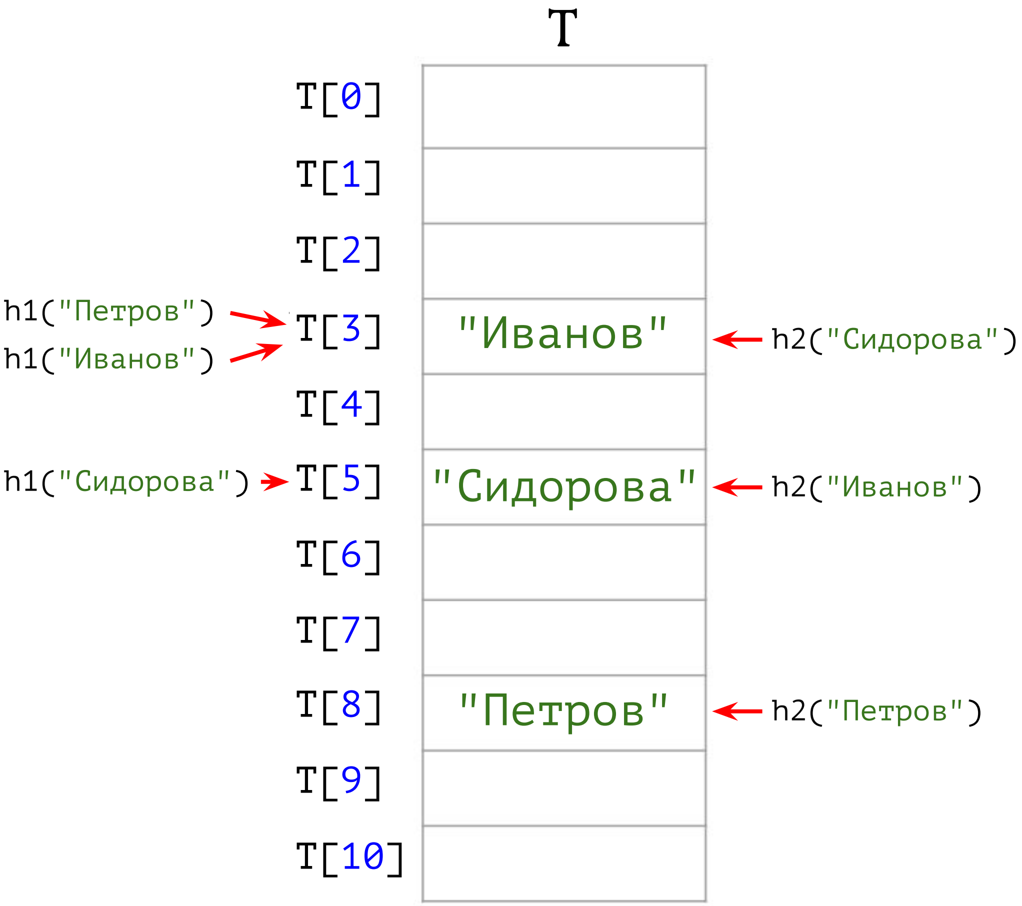
Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм поиска?

1. Проверяем h1(x). Если там наш элемент, то нашли
2. Проверяем h2(x). Если там наш элемент, то нашли.
3. Иначе, не нашли.

Сложность в худшем O(1)!

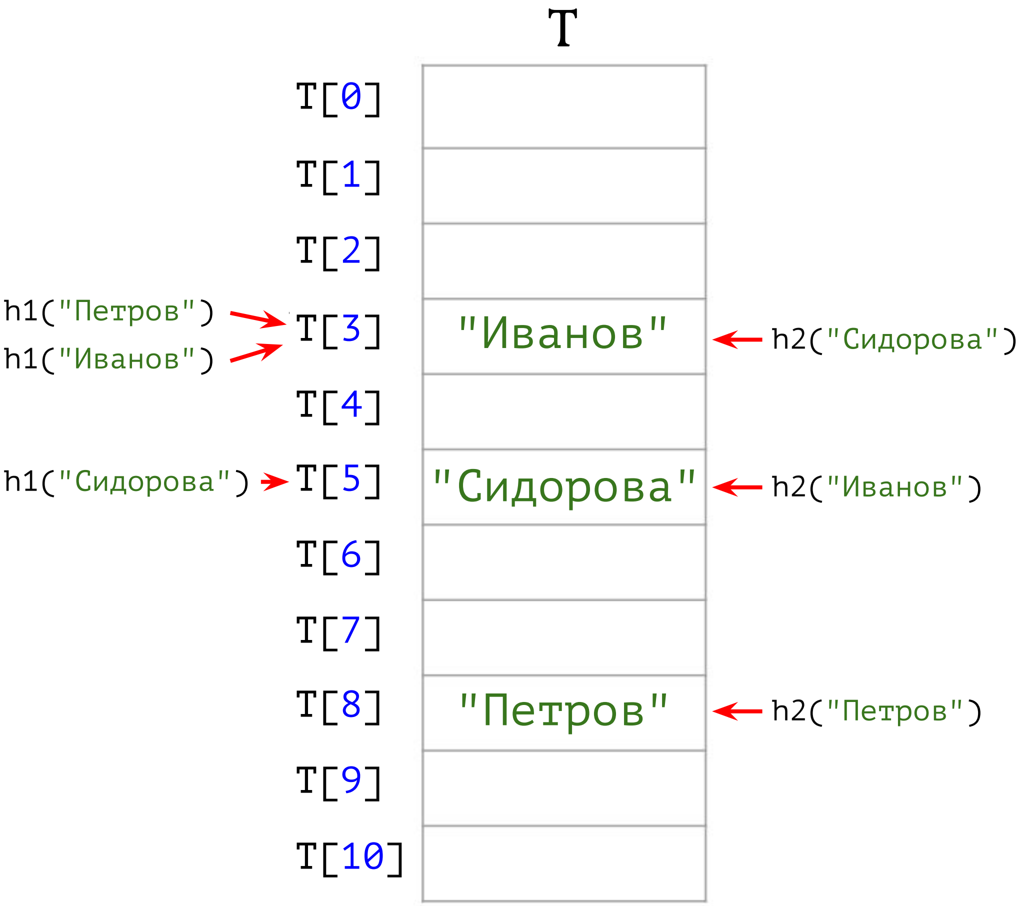
Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм удаления?

Хеш-таблицы: хеширование кукушки

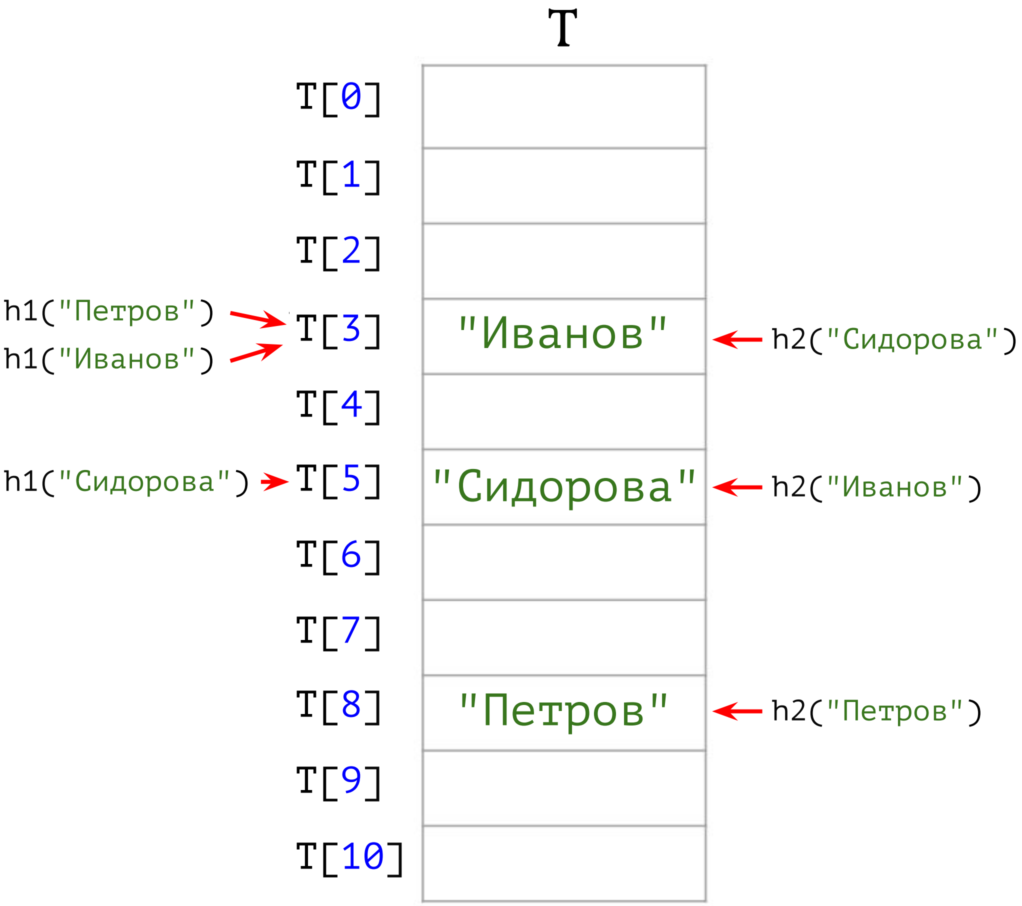


Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм удаления?

1. Ищем элемент по прошлой процедуре, если нашли, то просто удаляем.

Хеш-таблицы: хеширование кукушки



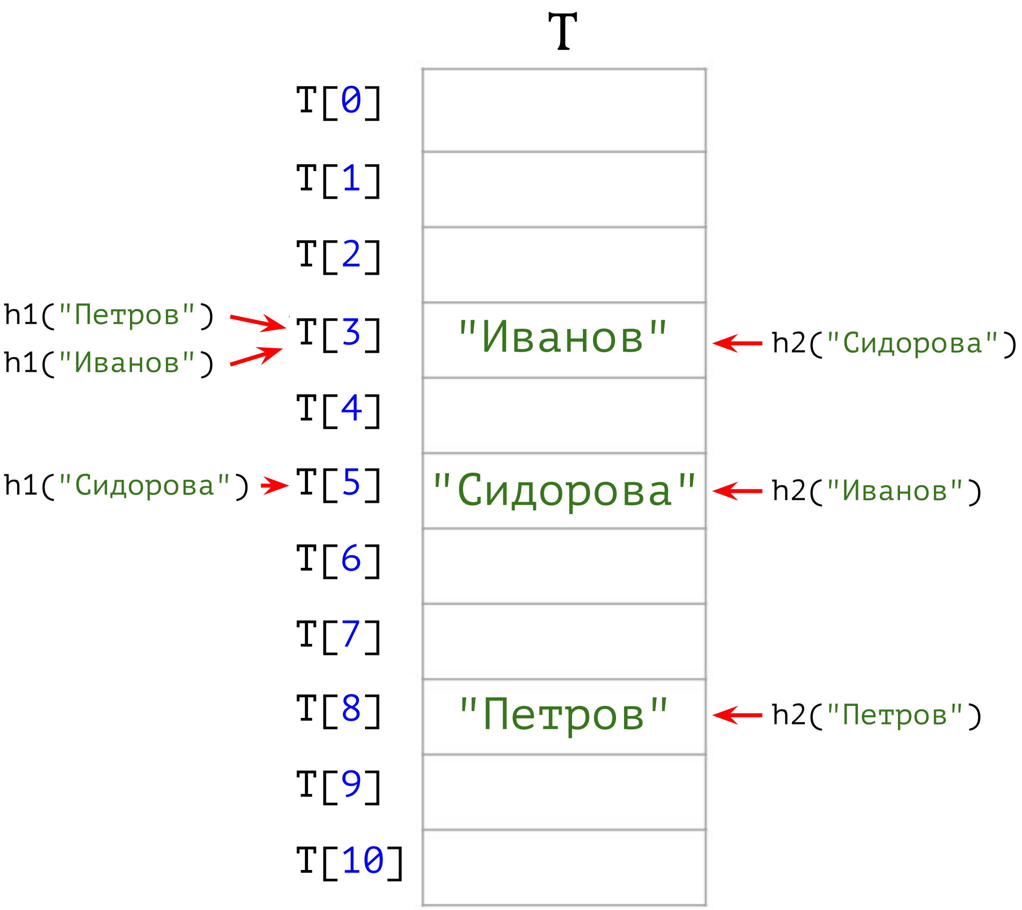
Пусть h_1, h_2 - хеш-функции из универсального семейства.

Алгоритм удаления?

1. Ищем элемент по прошлой процедуре, если нашли, то просто удаляем.

Никаких больше сложностей с дырками!

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

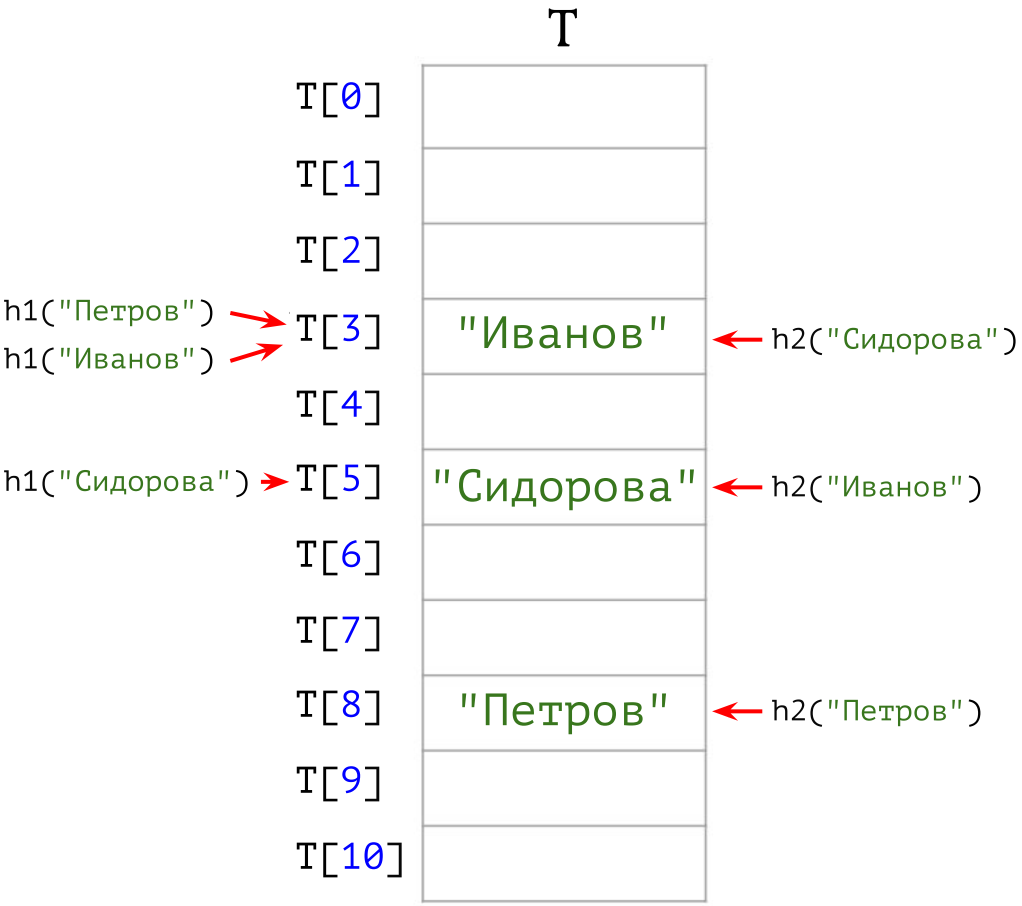
Алгоритм удаления?

1. Ищем элемент по прошлой процедуре, если нашли, то просто удаляем.

Никаких больше сложностей с дырками!

Сложность в худшем O(1)

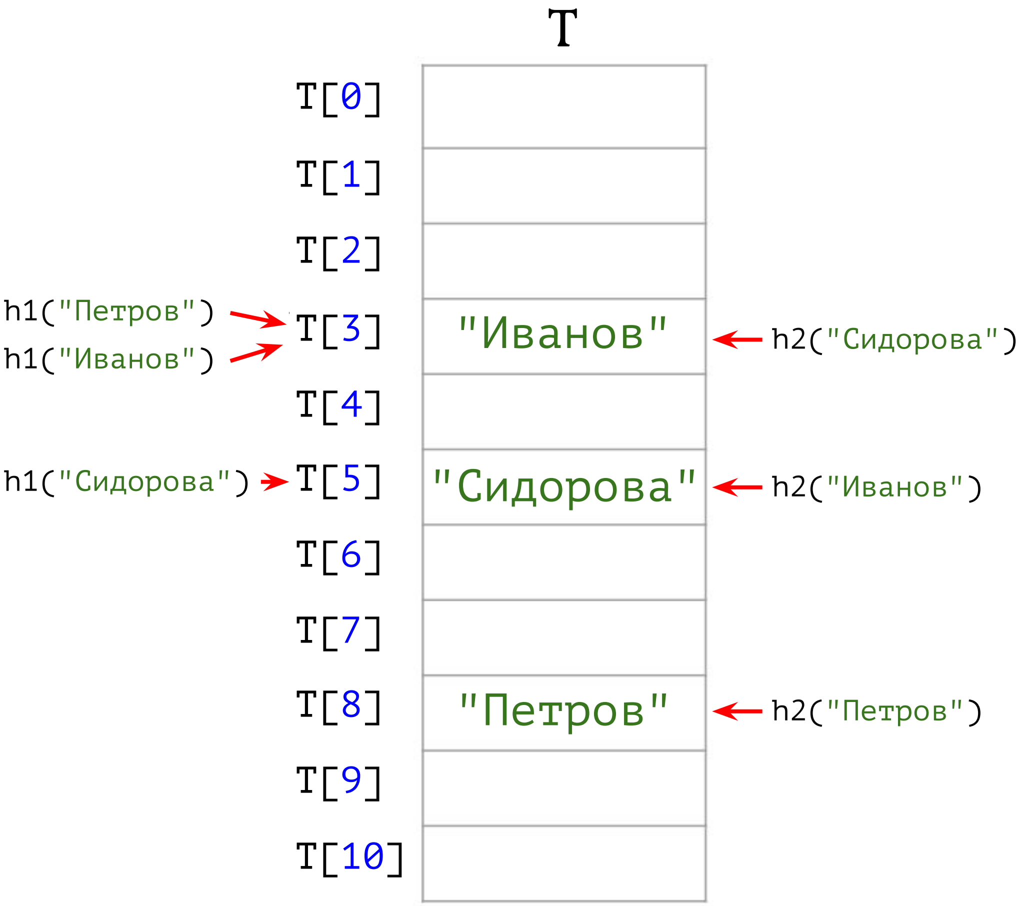
Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

В чем подвох?

Хеш-таблицы: хеширование кукушки



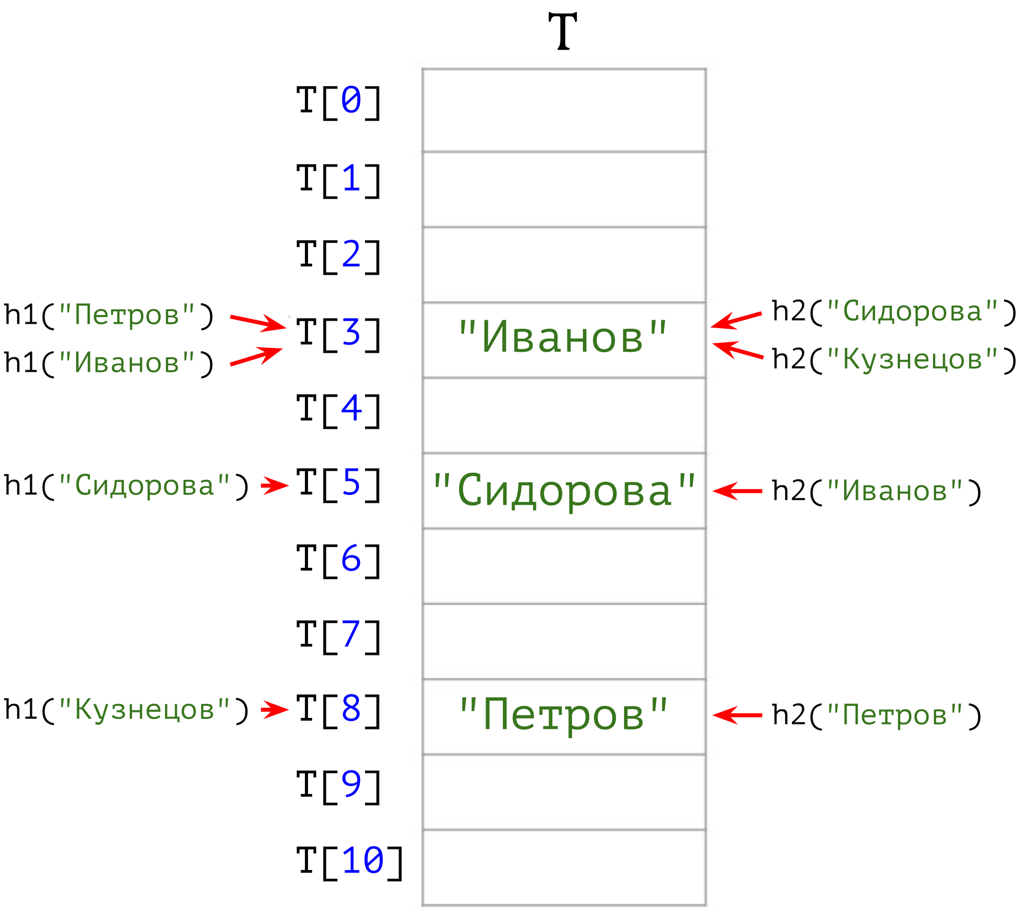
Пусть h_1, h_2 - хеш-функции из универсального семейства.

В чем подвох?

add("Кузнецов"):

h1("Кузнецов") = 8
h2("Кузнецов") = 3

Хеш-таблицы: хеширование кукушки



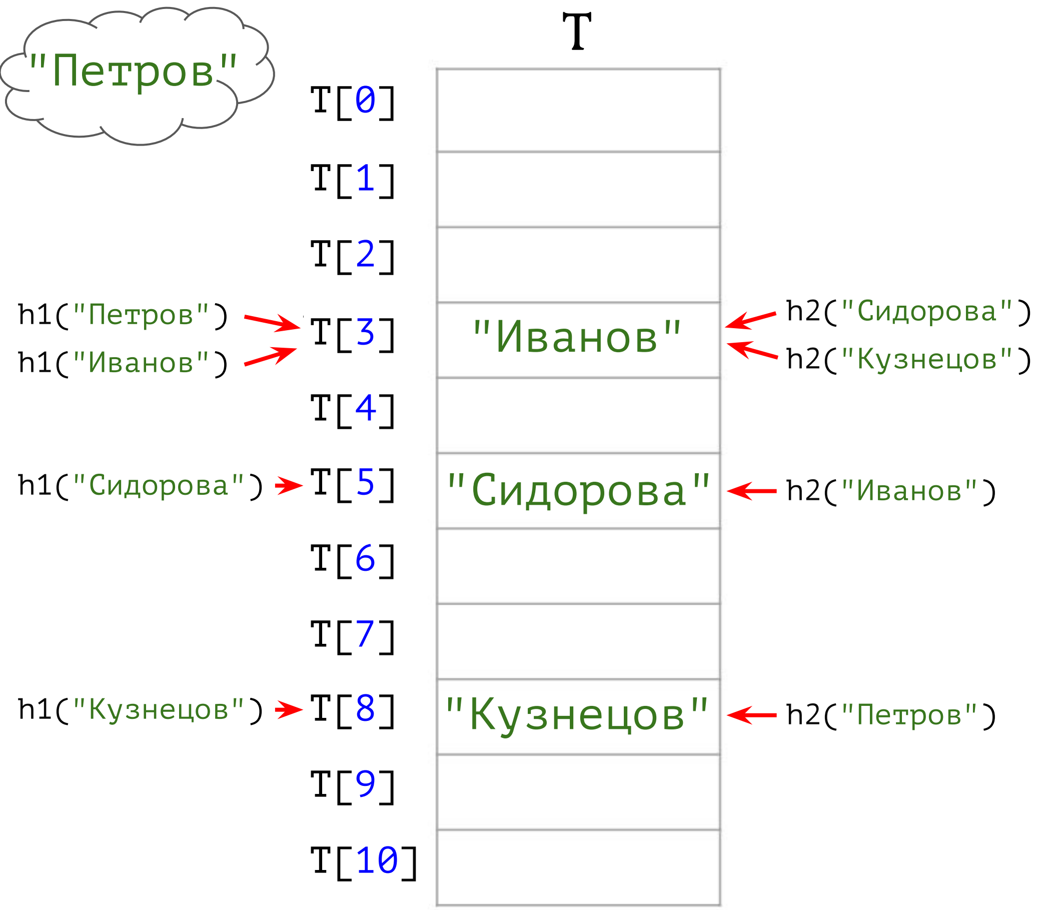
Пусть h_1, h_2 - хеш-функции из универсального семейства.

В чем подвох?

add("Кузнецов"):

h1("Кузнецов") = 8
h2("Кузнецов") = 3

Хеш-таблицы: хеширование кукушки



Пусть h_1, h_2 - хеш-функции из универсального семейства.

В чем подвох?

add("Кузнецов"):

h1("Кузнецов") = 8
h2("Кузнецов") = 3

Хеш-таблицы: хеширование кукушки

Пусть h_1, h_2 - хеш-функции из универсального семейства.

В чем подвох?

add("Кузнецов"):

h1("Кузнецов") = 8
h2("Кузнецов") = 3

Получили цикл! Никогда не найдем пустое место, куда все 4 элемента поместятся

Хеш-таблицы: хеширование кукушки

Пусть h_1, h_2 - хеш-функции из универсального семейства.

В чем подвох?

add("Кузнецов"):

h1("Кузнецов") = 8
h2("Кузнецов") = 3

Встретили цикл => делаем рехеш и выбираем новую пару функций

Хеш-таблицы: хеширование кукушки

Теорема: при использовании кукушкиного хеширования и коэффициенте заполненности таблицы меньшим 0.5 (т.е. при не более, чем 50% заполненности таблицы) среднее время добавления нового элемента константно.

Rasmus Pagh и Flemming Friche Rodler, 2001

Хеш-таблицы: хеширование кукушки

Теорема: при использовании кукушкиного хеширования и коэффициенте заполненности таблицы меньшим 0.5 (т.е. при не более, чем 50% заполненности таблицы) среднее время добавления нового элемента константно.

Rasmus Pagh и Flemming Friche Rodler, 2001

Хеш-таблицы: хеширование кукушки

Теорема: при использовании кукушкиного хеширования и коэффициенте заполненности таблицы меньшим 0.5 (т.е. при не более, чем 50% заполненности таблицы) среднее время добавления нового элемента константно.

Rasmus Pagh и Flemming Friche Rodler, 2001

Док-во:

https://www14.in.tum.de/lehre/2014WS/ea/split/sub-Hashing-handout.pdf

Хеш-таблицы: хеширование кукушки

Практическое замечание №1: нет смысла делать прямолинейный поиск циклов, достаточно ограничить максимальное количество пересылок за раз.

Хеш-таблицы: хеширование кукушки

Практическое замечание №1: нет смысла делать прямолинейный поиск циклов, достаточно ограничить максимальное количество пересылок за раз. Обычно число пересылок ограничивают logM, где M - текущий размер таблицы.

Хеш-таблицы: хеширование кукушки

Практическое замечание №1: нет смысла делать прямолинейный поиск циклов, достаточно ограничить максимальное количество пересылок за раз. Обычно число пересылок ограничивают logM, где M - текущий размер таблицы.

Практическое замечание №2: хотя в теории кукушкино хеширование может работать быстрее открытой адресации (речь про поиск и добавление), на практике оно на 20-30% медленнее из-за особенностей работы с кэшем.

Хеш-таблицы: хеширование кукушки

Практическое замечание №1: нет смысла делать прямолинейный поиск циклов, достаточно ограничить максимальное количество пересылок за раз. Обычно число пересылок ограничивают logM, где M - текущий размер таблицы.

Практическое замечание №2: хотя в теории кукушкино хеширование может работать быстрее открытой адресации (речь про поиск и добавление), на практике оно на 20-30% медленнее из-за особенностей работы с кэшем.

Практическое замечание №3: используется в кодовой базе TikTok

Takeaways

- Хеш-таблицы - как самое быстрое, что придумали для поиска/добавления/удаления

- Не забывайте, что это сложность в среднем

Takeaways

- Хеш-таблицы - как самое быстрое, что придумали для поиска/добавления/удаления
- Не забывайте, что это сложность в среднем
- Цепочки VS открытая адресация (думайте о памяти и об удалении)

Takeaways

- Хеш-таблицы - как самое быстрое, что придумали для поиска/добавления/удаления

- Не забывайте, что это сложность в среднем

- Цепочки VS открытая адресация (думайте о памяти и об удалении)

- Универсальное хеширование - как способ защиты от атак на вашу хеш-таблицу.