Практическое занятие №5

Густов Владимир Владимирович
gutstuf@gmail.com

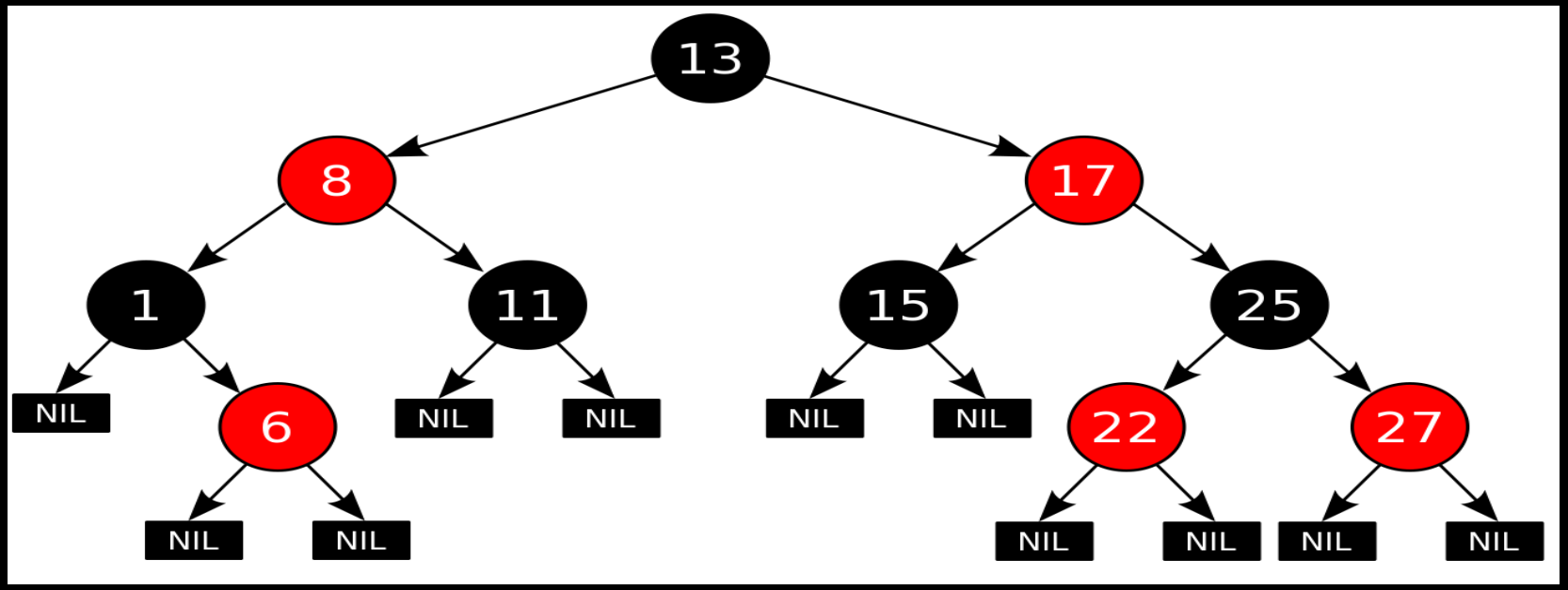
Цитата дня: Преждевременная оптимизация — корень всех зол (с) Дональд Кнут

Красно-чёрное дерево

КЧД – сбалансированное бинарное дерево поиска, использующее для балансировки такой атрибут, как «цвет» (красный и чёрный).

Свойства:

- вставка/удаление элемента — O(log n) (даже в худшем случае);
- поиск элемента — O(log n);



Свойства

- каждый узел окрашен либо в красный, либо в чёрный цвет (цвет – условность, по сути это дополнительное информационное поле в структуре, а-ля `bool color`);
- корень всегда окрашен в чёрный цвет;
- листья (`null/nil` узлы) всегда окрашены в чёрный цвет;
- каждый красный узел имеет только чёрные дочерние узлы и только чёрного родителя;
- чёрные узлы могут иметь в качестве дочерних чёрные;
- пути от узла к листьям должны содержать одинаковое количество чёрных узлов (так называемая – чёрная высота, **bh**).

Алгоритм вставки в КЧД

- вставка производится по правилам БДП (слева строго меньше, справа - больше или равно);
- после вставки необходимо производить балансировку дерева (посредством поворотов и перекраски узлов) **с учётом свойств КЧД**;
- возможны 4 ситуации (случая) при вставки (для узла X):

1) X — корень;
2) дядя узла X — красный;
3) дядя узла X — чёрный (вставка образует угол);
4) дядя узла X — чёрный (вставка образует линию);

Резюме

1. красно-чёрное дерево обладает теми же свойствами, что и обычное бинарное дерево поиска;
2. имеет дополнительный бит информации (цвет);
3. имеет фантомный чёрный лист;
4. балансировка производится посредством поворотов и перекраски;
5. после балансировки должны выполняться/сохраняться все свойства красно-чёрного дерева;

Алгоритм удаления из КЧД

- удаление производится по правилам БДП (нет детей — удаляем; 1 дитё — заменяем; два и более — берём либо максимальный из левого поддерева, либо минимальный из правого);
- после удаления необходимо производить балансировку дерева (посредством поворотов и перекраски узлов) **с учётом свойств КЧД**;
- при нарушении черной высоты, производится балансировка (т.е. балансировка потребуется только при удалении/смещении чёрного узла);
- **замещаемый узел (дочерний узел, которым заменяем удалённый) первоначально сохраняет цвет удалённого узла;**

Алгоритм удаления из КЧД

nil – (пустой!) листовой узел, всегда чёрный. **Участвует** в алгоритме удаления (и в родственных связях) **как** и **все остальные узлы**. Не удаляем.

При удалении узла Z (с замещением на узел X) возможны следующие **исходы**:

1) Z – красный, X – красный или *nil* -> балансировка не нужна;
2) Z – красный, X – чёрный -> необходима балансировка;
3) Z – чёрный, X – красный -> балансировка не нужна;
4) Z – чёрный, X – чёрный или *nil* -> необходима балансировка;

Для балансировки используются 4 **случая** (описаны после примеров):

1) узел X чёрный и его брат красный;
2) узел X чёрный и его брат чёрный, и оба племянника чёрные;
3) узел X чёрный и его брат чёрный, один из племянников красный **И** второй чёрный;
4) узел X чёрный и его брат чёрный, а один из племянников красный.

Note: как выглядит родство и сами алгоритмы вставки/удаления, можно найти в презентации №4

Резюме

1. при добавлении оперируем цветом дяди;
2. при удалении оперируем цветом племянников и брата;
3. при удалении красной вершины (зачастую) нет необходимости в балансировке.

Попробуйте построить сами

1) 24, 79, 14, 24, -66, 35, 53, 4
2) 80, 40, 62, -95, -72, -78, 37, 3, 91

Проверить себя можете здесь (плохо работает с отрицательными числами):
https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Ассоциативный массив

Абстрактная структура данных основанная на взаимоотношениях «ключ-значение».
Ассоциативный массив (читай - массив ассоциаций) содержит в себе пары (ключ, значение), где:
-ключ – уникальная идентификационная информация, с помощью которой можно «описать»/распознать значение;
- значение – какая-то информация, данные;
Так же считается, что ассоциативный массив не может хранить в себе две пары с одинаковыми ключами.

Примеры из жизни:
- таблицы в базах данных;
- журнал успеваемости/посещаемости студентов (ключом является ФИО студента, а значением – его оценки/Н-ки);
- турнирный зачёт (ключом будет считаться имя команды, а значением – её баллы/медали);
- телефонный справочник (ключом является номер телефона, а значением имя компании, т.к. одна и та же компания может иметь несколько разных номеров).

В общем, практически в любой области, где данные можно ассоциировать с каким-то (уникальным/не повторяющимся) значением, можно реализовать ассоциативный массив.

Словарь (таблица)

Абстрактная структура данных, является ассоциативным массивом реализующимся с помощью сбалансированного бинарного дерева поиска.

Свойства (как в сбалансированных БДП, далее СБДП):
- вставка/удаление элемента — O(log n);
- поиск элемента — O(log n);
- все пары (далее – элементы) отсортированы по ключу;
- не имеет пар с одинаковыми ключами.

В силу того, что в основном словари реализуются на основе СБДП, мы не станем отдельно рассматривать алгоритмы вставки и удаления, т.к. прошли их с вами на предыдущем занятии (см. КЧД).

Ключ	Значение
Васин И.О.*	180
Дугина А. В.	180
Петров Р.Д.	175

Словарь

Предметная область: журнал учёта роста

* - для краткости, имя и отчество сокращается до инициалов

Словарь (в общем)

- представляет из себя ассоциативный массив (ключ + значение);
- реализуется с помощью сбалансированного бинарного дерева поиска (AVL, КЧД, etc);
не содержит дубликаты ключей (для этого есть мультисловарь);
- значения могут дублироваться, а ключи – нет.

Пример:

```
map<string, string> m; // STL-контейнер std::map (#include <map>)
m.emplace(std::make_pair(“Это ключ”, “это значение”));
m.emplace(std::make_pair(“Новый ключ”, “это значение”));
m.emplace(std::make_pair(“Это ключ”, “а это новое значение”));

// последняя вставка не будет исполнена, т.к. ключ дублируется
for (auto& el: m)
    cout << el.first << “ “ << el.second << endl;
```

Вывод:

```
Новый ключ это значение
Это ключ это значение
```

Словарь (STL)

Обращаться к элементам словаря можно как через оператор квадратные скобки (`MyMap[“Key”]`), так и через метод at (`MyMap.at(“Key”)`).
Но стоит учесть, что с помощью оператора [] также можно и добавлять новые элементы в контейнер: `MyMap[“New”] = “Val”`. Если значение не указано, то оно конструируется по умолчанию:

```
std:::map<int, int> obj;
obj[5]; // значение инициализируется 0ом
Cout << obj.size() << “ “ << obj[5]; // 1 0
```

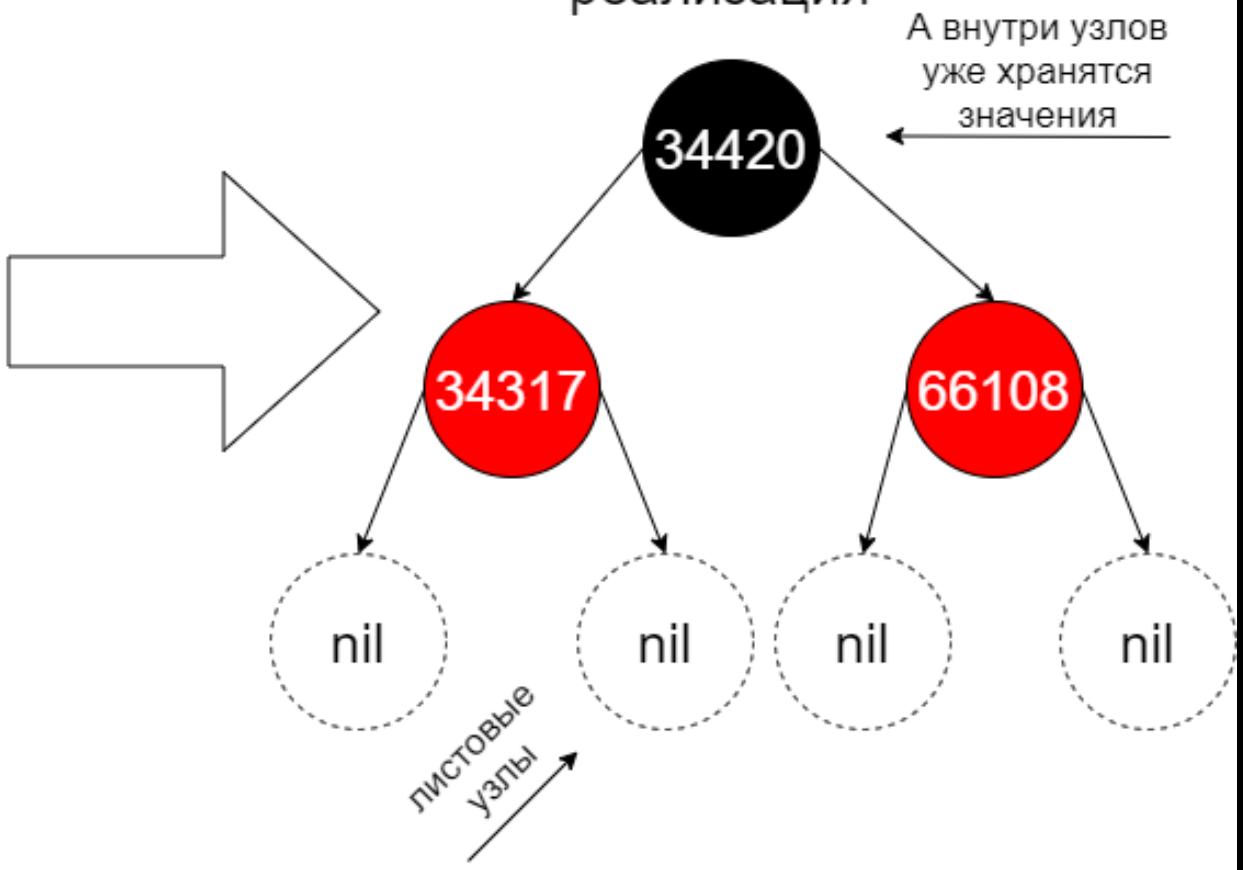
Note: в качестве ключа для словаря вы можете использовать любой тип данных (в том числе и свои классы/структуры) для которых реализованы операции сравнения на (меньше) < и (равенство) ==.

Словарь

Что видит пользователь структуры

Ключ	Значение
3-43-17	Дугин О.Б.
3-44-20	Воробьёв В.В.
6-61-08	Грудинин П.Н.

Что из себя представляет реализация



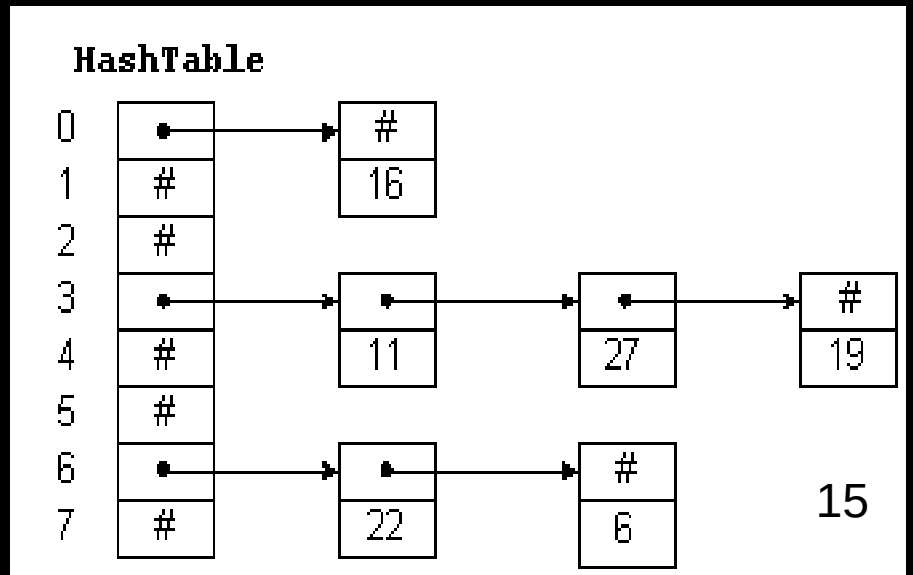
Хеш-таблица

Ассоциативный массив, чьи ключи преобразуются некоторой математической функцией (хеш-функция).

Свойства:

- вставка/удаление элемента — O(1) (в худшем – O(n));
- поиск элемента — O(1) (в худшем – O(n));
- все пары (далее – элементы) хранятся **не отсортированными**;
- может содержать элементы с одинаковым хешем (коллизия).

С помощью хеш-таблиц можно реализовать словарь с операциями за O(1), но стоит учесть, что канонический словарь всегда отсортирован.



Хеш-таблица

В основе реализации хеш-таблиц лежит массив указателей, либо массив встроенных структур (т.е. других структур, чаще всего – списков). Использование хеширования для вычисления индекса элемента (хеш) позволяет добиваться сложности O(1) при работе с таблицей, т.к. нет необходимости в проходе по всей структуре в поиске нужных элементов.

Но при этом не гарантируется, что время выполнения отдельной операции будет O(1).
Это связано с тем, что при достижении некоторого значения *коэффициента заполнения* необходимо перестроить хеш-таблицу (обновить индексы).
В таком случае требуется увеличить размер массива и заново добавить в пустую хеш-таблицу все пары.

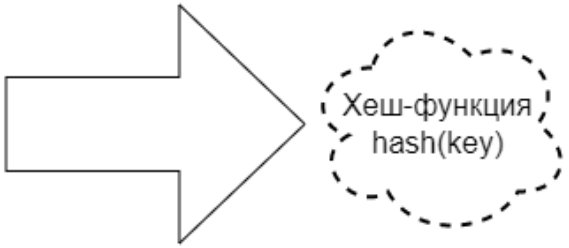
Коэффициент заполнения хеш-таблицы – число хранимых элементов, делённое на размер массива. От этого коэффициента зависит среднее время выполнения операций добавления/удаления/поиска.

STL реализация – *std::unordered_map.*

Хеш-таблица

Что видит пользователь структуры

Ключ	Значение
3-43-17	Дугин О.Б.
3-44-20	Воробьёв В.В.
6-61-08	Грудинин П.Н.



Что из себя представляет реализация

Индекс	Содержимое
0	указатель*
1	указатель
2	указатель

Ключ	Значение
3-43-17	Дугин О.Б.

Ключ	Значение
3-44-20	Воробьёв В.В.

Ключ	Значение
6-61-08	Грудинин П.Н.

* - либо вместо указателя лежит сам объект-пара, содержащий в себе и ключ и значение

Хеш и хеширование

Хеш-функция – это функция преобразующая ключ ***key*** в некоторый индекс ***i*** равный ***h(key)***, где ***h(key)*** – хеш (хеш-сумма) ***key***.

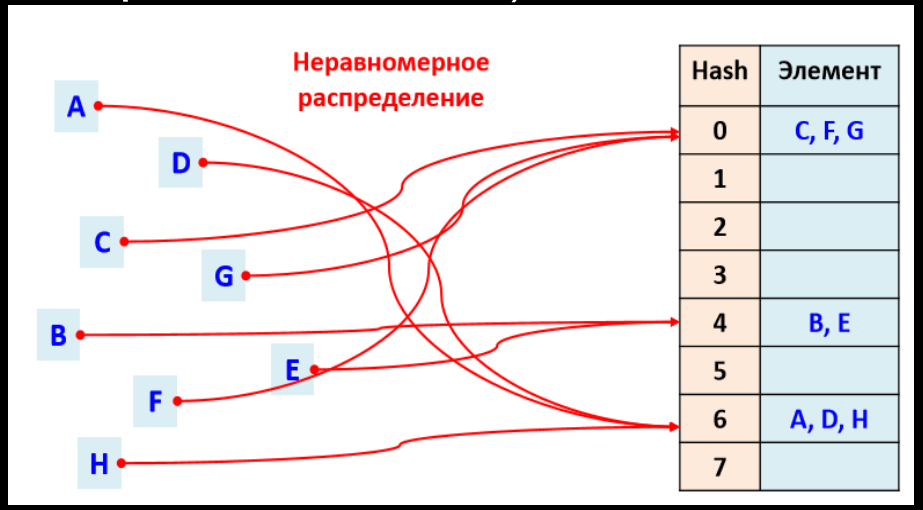
Хеширование – процесс преобразования ключей в индексы хеш-таблицы.

Хеш (хеш-сумма) – результат обработки данных (ключей) хеш-функцией.

Цель хэш-функции - превратить объект, используя его содержимое, в целочисленное значение - это и будет индекс для массива.
Не стоит путать с криптографическими хеш-функциями, цель которых заключается в шифровании данных и проверки целостности/подлинности информации.

Пример простейшей (но плохой, с точки зрения коллизий) хеш-функции:

```
int calc_hash(string key) {
    int hash = 0;
    for(auto el: key)
        hash += el ^ hash;
    return hash;
}
```



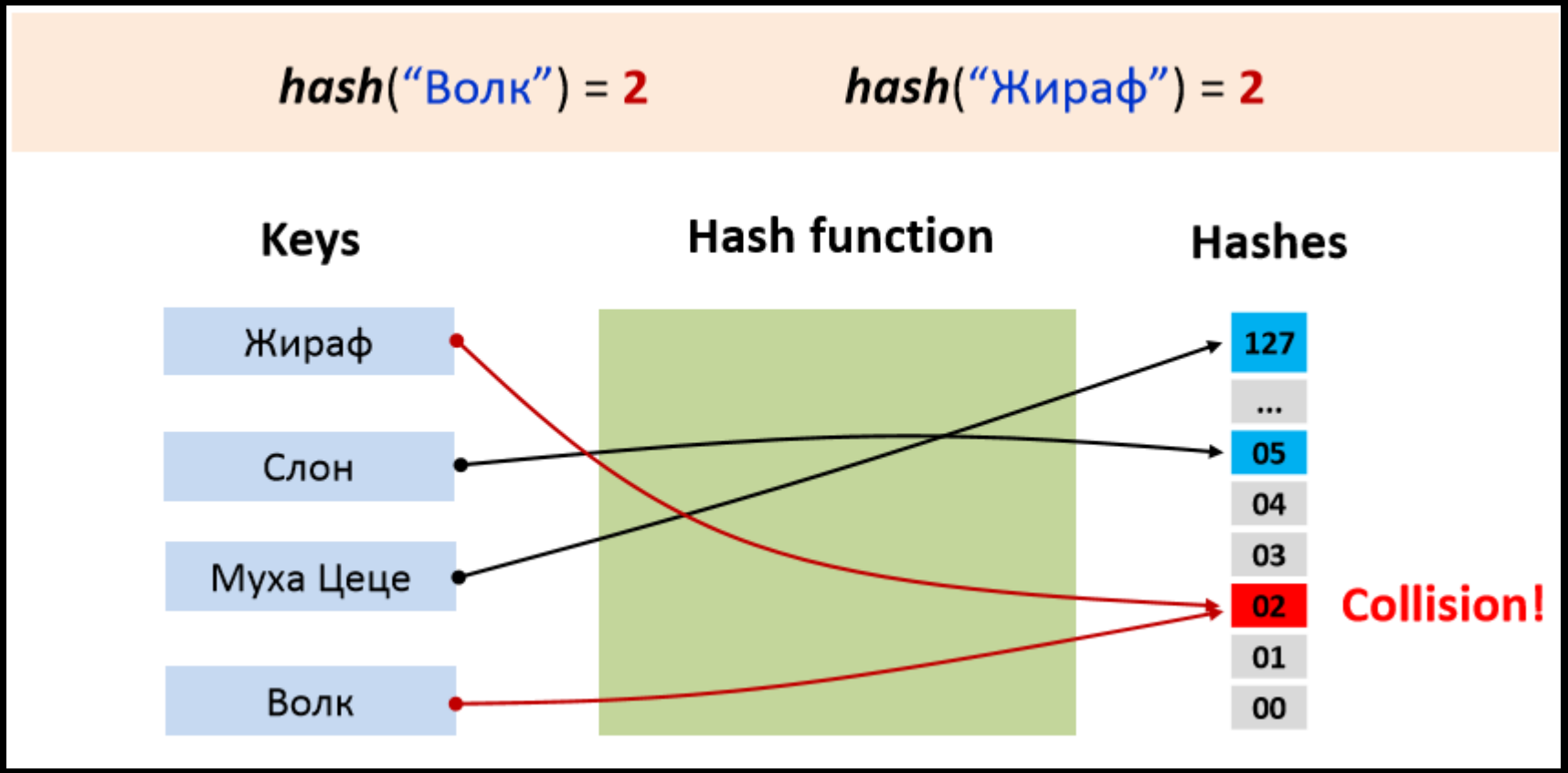
Свойства (хорошей) хеш-функции:
- должна быстро вычисляться – дабы выполнение хеширования не стоило дороже, чем худшая операция за O(n);
- является равномерной – равномерно заполняет ячейки массива;
- не должна быть непрерывной – для близких значений должны получаться сильно различающиеся результаты;
- значения функции не должны образовывать кластеры – множество близко стоящих точек (значений);

Примеры хеш-функций: со схемой Горнера, алгоритм Седжвика

Так или иначе, практически любая хеш-функция для двух (и более) ключей может выдать один и тот же хеш – такая ситуация называется ***коллизией***.

Коллизии и способы их решения

Коллизия – случай, при котором хеш-функция выдаёт одинаковый хеш для разных входных данных. Вероятность возникновения коллизий используется для оценки качества хеш-функций.

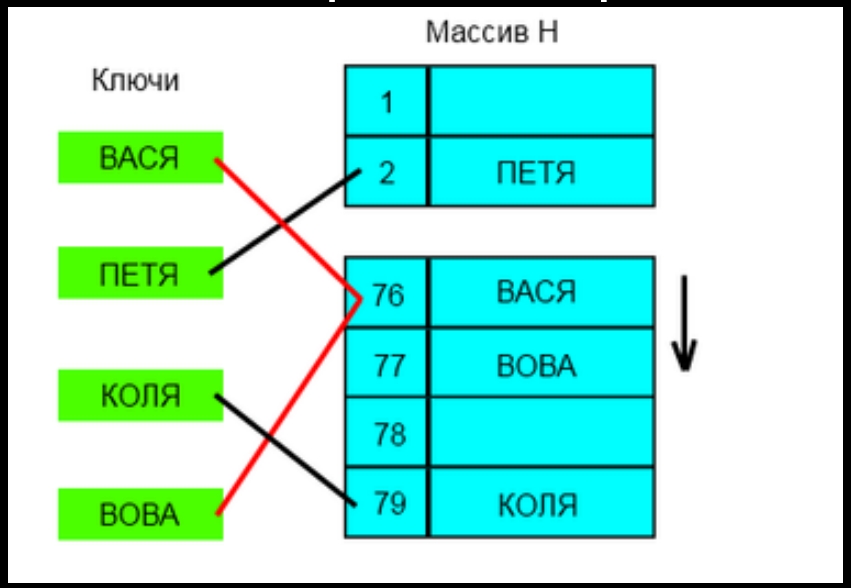


Коллизии и способы их решения

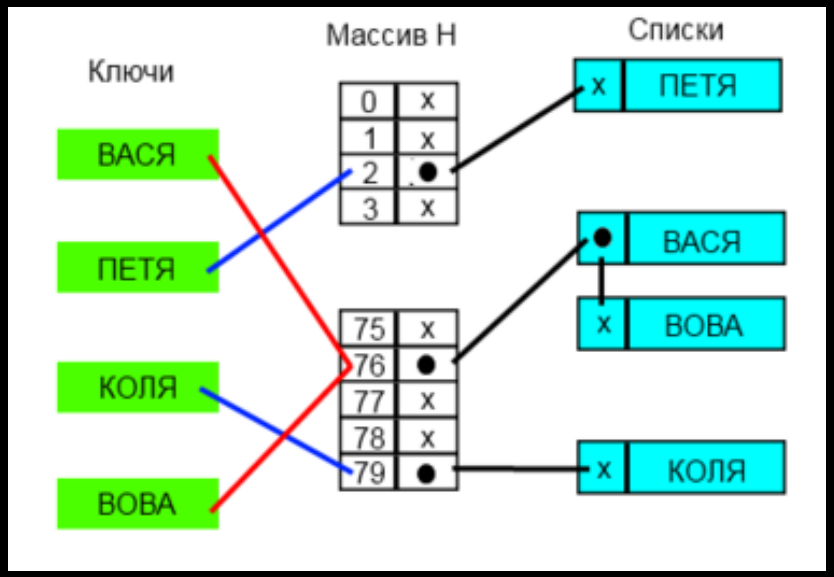
Хеш-функции не вызывающие коллизий называются – совершенными (perfect hash function).

Для решения коллизий используется метод открытой адресации, метод цепочек (закрытой адресации), либо метод рехеширования.

Метод открытой адресации:



Метод цепочек:



Метод открытой адресации

Реализует массив указателей/объектов, в котором каждая ячейка массива содержит пару ключ-значение.

1. вычисляется хеш-функция;
2. если по индексу ничего нет (пустая ячейка), то нет и элемента;
3. если по индексу элемент с нашим ключом – элемент найден;
4. если по индексу элемент с другим ключом или элемент помечен удалённым, индекс увеличивается на единицу, переходим к пункту 2;

Следующий индекс вычисляется по формуле:

`(index + 1) mod H`, где *H* – размер хеш-таблицы.

При вставке:

5. если ячейка свободна, то производится вставка пары;
6. если ячейка занята, то производится итерация индекса до свободной ячейки, и при её наличии – вставка.

При удалении:

5. если ячейка свободна, то нет и элемента;
6. если ячейка занята, то производится итерация индекса до нужной ячейки;
7. элемент помечается удалённым.

Метод открытой адресации

Преимущества:
+ под таблицу требуется меньше памяти, нежели чем при методе цепочек;
+ предоставляет быстрый доступ к элементам таблицы.

Недостатки:
- крайне нетривиальное удаление элемента из таблицы и образование кластеров — последовательностей занятых ячеек;
- кластеризация замедляет все операции с хеш-таблицей. При добавлении требуется перебирать всё больше элементов, при проверке тоже. Чем больше в таблице элементов, тем больше в ней кластеры и тем выше вероятность того, что добавляемый элемент попадёт в кластер;
- чувствительная к качеству хеш-функции.

Метод цепочек (закрытая адресация)

Реализует массив односвязных списков, в котором каждый элемент списка представляет из себя пару ключ-значение.

1. вычисляется хеш-функция;
2. если элемент уже существует (ячейка занята), но ключ не совпадает, то производится поиск по элементам списка вглубь.

При вставке:

3. если ячейка пуста, то производится вставка пары;
4. если ячейка занята, то производится вставка в конец списка.

При удалении:

3. если ячейка пуста, то нет и элемента;
4. если ячейка занята, то производится поиск элемента в списке и его удаление.
5. если список пуст, то удаляется «точка входа» (список).

Метод цепочек

Плюсы:
+ простота реализации
+ требования к хеш-функции ниже, чем при открытой адресации

Недостатки:
- более длительное обращение к элементам таблицы (в сравнении с открытой адресацией);
- больше занимаемой памяти (списки хранят указатели на следующие узлы);

Метод рехеширования

Реализация такая же, как при открытой адресации. Работает почти как метод открытой адресации, только вместо итерации индекса на 1, при попадании на занятую ячейку, используется вычисление второй (и так далее) хеш-функции. До тех пор, пока не будет найдена свободная ячейка.

При этом вторая (и последующие) хеш-функция могут качественно отличаться от первой хеш-функции.

Метод рехеширования

Плюсы:
+ требования к хеш-функции ниже, чем при линейной открытой адресации;
+ таблица занимает меньше памяти, чем при методе цепочек.

Недостатки:
- те же, что и при открытой адресации;

Резюме

Хеш-таблицы плохи также тем, что на их основе нельзя реализовать быстро работающие дополнительные операции поиска MIN, MAX и алгоритм обхода всех хранимых пар в порядке возрастания или убывания ключей.

Так же, помимо хеш-таблиц существуют и другие структуры данных основанных на использование хешей, например: хеш-деревья и хеш-списки.

В общем:
- в словаре элементы отсортированы (исходя из свойств БДП);
- у хеш-таблицы элементы не отсортированы (исходя из хеш-функций);
- вместо перебора ключей (для поиска элемента), предпринимается попытка обращения к элементам непосредственно, за счёт выполнения арифметического преобразования ключей в адреса таблицы.
- если коэффициент заполнения превосходит 0.5 - 0.7 – таблицу расширяют: создают новый массив указателей с нужным размером, и в порядке увеличения индексов извлекают элементы из старого массива и применив хеширование, вставляют в новый;
- у открытой адресации есть различные модификации: квадратичная, линейная, последовательная (рассмотренная) и т.д;
- словарь – операции за O(log n), хеш-таблица – за O(1) (и O(n) в худшем случае).

Ссылки

-пример реализации словаря с помощью связного списка;
- ещё несколько вариантов хеш-функций (имхо, не самых надёжных);
- стратегии решения коллизий;
- небольшое пояснение по хеш-таблице;
- вики, про работу с хеш-таблицей с линейным пробированием при открытой адресации;
- теория об идеальном хешировании.