Лекция 19 от 22.03.2016

Хорошая хеш-функция

Пусть H — семейство хеш-функций $h : U \to \{0, 1, \dots, m-1\}$. Назовём H *универсальной*, если выплолняется
$$\forall x \in U, y \in U \left|\{h(x) = h(y) \mid h \in H\}\right| = \frac{|H|}{m}$$

При случайном выборе $h \in H$ $\Pr[h(x) = h(y)] = \frac{1}{m}$

Будем считать, что пользуемся такой h.

Пусть $m = n^2$, где n — число хранимых ключей. Выберем $h \in H$; сколько будет коллизий (таких пар (x, y), что $h(x) = h(y)$)? Утверждается, что немного.

Давайте оценим матожидание коллизии:
$$E(\text{число коллизий}) = Pr(\text{коллизия}) * \binom{n}{2} = \frac{1}{m} \cdot \frac{n(n-1)}{2} = \frac{n-1}{2n} < \frac{1}{2}$$
$$Pr(\#\text{ коллизий} \geqslant 1) \leqslant \frac{E(\#\text{ коллизий})}{1} = \frac{1}{2}$$
$$Pr(\text{коллизий нет}) \geqslant \frac{1}{2}$$

Заметим, что тогда (мы опираемся на то, что данные сохраняются единожды) получается, что потратив в среднем две попытки мы можем найти такую h, что хеширование произойдёт без коллизий и поиск будет за константное время (это, кстати, называется *идеальным хешированием*).

Или давайте так:

Создадим таблицу из $m = n$ ячеек; но таблица будет не простой, а состоящей из хеш-таблиц; при этом внутри каждой такой таблицы коллизий не будет (мы об этом позаботимся).

Пусть n_j — число элементов таких, что $h(x) = j$. $\sum\limits_j n_j = n$, как несложно заметить. Введём также m_j — число ячеек в j-ой таблице второго уровня. Если мы хотим обеспечить отсутствие коллизий (и не потратить на это кучу времени), то m_j должно быть равно n_j^2. Вопрос — а чем такой способ лучше предыдущего? А давайте посмотрим на память: внешняя таблица имеет линейное количество ячеек, где каждая имеет константную память (там хранятся лишь ссылки на таблицы второго уровня). А память на таблицы второго уровня — $\sum\limits_{j=0}^{m-1} \Theta(n_j^2)$. Понятно, что если придумать худший случай, то все будут в одной ячейке. Давайте тогда считать матожидание:
$$E\left[\sum_{j=0}^{m-1} \Theta(n_j^2)\right]$$

Но сначала вот что:
$$n_j^2 = n_j + n_j^2 - n_j = n_j + n_j(n_j - 1) = n_j + 2\binom{n_j}{2}$$

$$E\left[\sum_{j=0}^{m-1} n_j^2\right] = E\left[\sum_{j=0}^{m-1} n_j + 2\binom{n_j}{2}\right] = E\left[\sum_{j=0}^{m-1} n_j\right] + 2E\left[\binom{n_j}{2}\right] =$$
$$= n + 2E\left[\#\text{ коллизий для } h\right] \leqslant n + 2 \cdot \frac{1}{m} \cdot \frac{n(n-1)}{2} = n + n - 1 = 2n - 1$$

Ой! Линейная память! Вот и искомое преимущество.

Но пока всё довольно абстрактно. Где брать такие универсальные хеш-функции? Давайте приведём пример:

Рассмотрим H_{pm}, где p — простое число, большее всех ключей. Пусть она состоит из функций $h(k) = ((ak+b) \mod p) \mod m;\ a \in [1;p],\ b \in [0;p]$

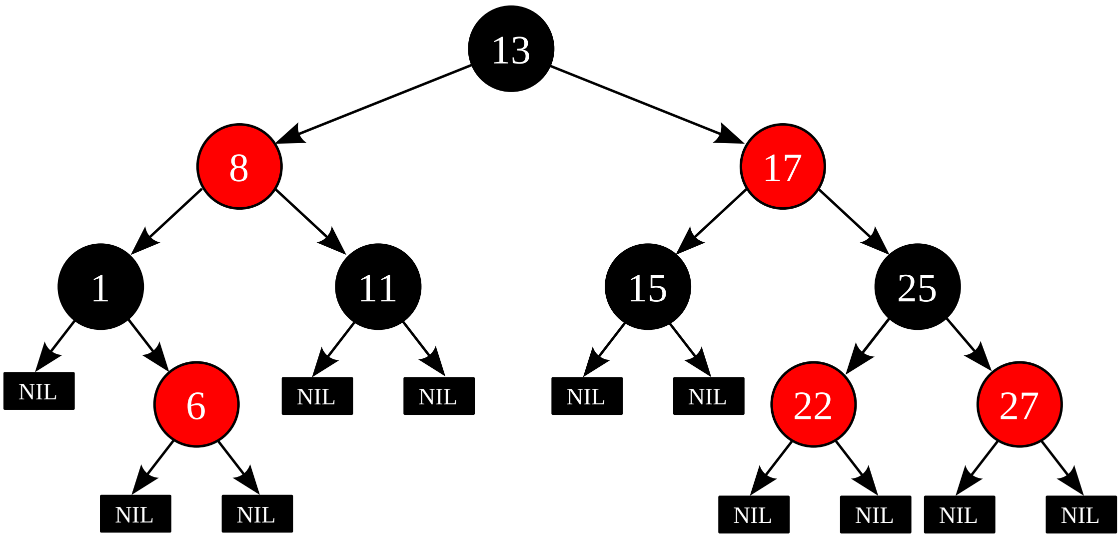
Красно-чёрное дерево

Что такое бинарное дерево поиска мы знаем; рассмотрим теперь *сбалансированное* бинарное дерево. Его ключевое свойство: высота такого дерева — $O(\log n)$. Разумеется, речь идёт о семействе деревьев; сказать, выполняется ли это для конкретного дерева нельзя.

Рассмотрим один класс таких деревьев — *красно-чёрные* деревья. Ключевые характеристики:

1. Любой узел — красный или чёрный.
2. Корень и листья — чёрные.
3. Родителем красного узла может быть только чёрный.
4. На всех простых путях из узла x до листьев-наследников одинаковое количество чёрных узлов.

Вот как может выглядеть такое дерево:



Высота RB-дерева с n ключами не больше $2\log_2(n+1)$. Докажем это не вполне формально:

"Подтянем" красные узлы к чёрным родителям. Легко заметить, что высота дерева уменьшилась не больше, чем вдвое: $n \leqslant 2h'$. Число листьев, получившихся в итоге — $n+1 \geqslant 2^{h'}$; $\log_2(n+1) \geqslant h' \geqslant \frac{h}{2}$. Значит, высота дерева — $O(\log n)$ и поиск в дереве займёт логарифмическое время.

```
rb_insert(t, x):
    tree_insert(t, x)
    x.color = 'red'
    while x != t.root and x.p.color = 'red' do
        if x.p = x.p.p.right then
            y:= x.p.p.left
            if y.color = 'red' then
                x.p.color:= 'black'
                y.color:= 'black'
```

```
    x.p.p.color = 'red'
else
    AAAAAAAAAAAA
```