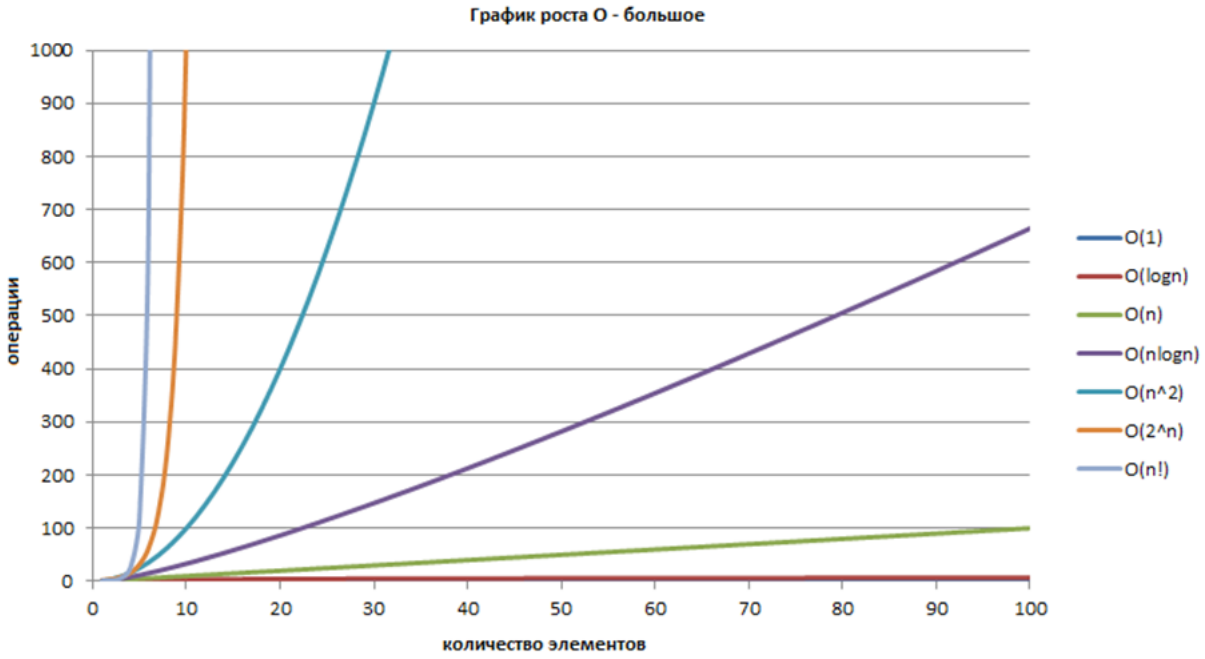
Подготовка к экзамену по дисциплине "Алгоритмы и структуры данных"

Автор: Минец Максим

Ассимтотика



O - оценка сверху
Наихудшее время работы алгоритма.
$f(n) = O(g(n))$, где $f(n)-$ функция времени нашего алгоритма.
Существуют положительные константы c и n_0, такие, что $0 \leq f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$

Θ - средняя оценка
Среднее время работы алгоритма.
$f(n) = \Theta(g(n))$
Существуют положительные константы c_1, c_2 и n_0, такие, что $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$,
$\forall n \geq n_0$

Ω - оценка снизу
Наилучшее время работы алгоритма.
$f(n) = \Omega(g(n))$, где $f(n)$
Существуют положительные константы c и n_0, такие, что $0 \leq c \cdot g(n) \leq f(n)$, $\forall n \geq n_0$

$\forall$ двух функций $f(n)$ и $g(n)$ равносильно следующее:
$f(n) = O(g(n))$
$g(n) = \Omega(f(n))$

o - нестрогая оценка сверху
$f(n) = o(g(n))$
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$g(n)$ растет быстрее (быстрее стремится к бесконечности) чем $g(n) \Rightarrow$ предел равен нулю.

ω - нестрогая оценка снизу
$f(n) = \omega(g(n))$
$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$
$f(n)$ растет быстрее (быстрее стремится к бесконечности) чем $g(n) \Rightarrow$ предел равен нулю.

Некоторые свойства асимптотики

Транзитивность
$f(n) = \Theta(g(n))$ и $g(n) = \Theta(h(n))$ влечет $f(n) = \Theta(h(n))$
$f(n) = O(g(n))$ и $g(n) = O(h(n))$ влечет $f(n) = O(h(n))$
$f(n) = \Omega(g(n))$ и $g(n) = \Omega(h(n))$ влечет $f(n) = \Omega(h(n))$
$f(n) = o(g(n))$ и $g(n) = o(h(n))$ влечет $f(n) = o(h(n))$
$f(n) = \omega(g(n))$ и $g(n) = \omega(h(n))$ влечет $f(n) = \omega(h(n))$

Рефлексивность
$f(n) = \Theta(f(n))$
$f(n) = O(f(n))$
$f(n) = \Omega(f(n))$

Симметричность
$f(n) = \Theta(g(n))$ если и только если $g(n) = \Theta(f(n))$

Обращение
$f(n) = O(g(n))$ если и только если $g(n) = \Omega(f(n))$
$f(n) = o(g(n))$ если и только если $g(n) = \omega(f(n))$

Теорема
Для любых двух функций $f(n)$ и $g(n)$ свойство $f(n) = \Theta(g(n))$ выполнено тогда и только тогда, когда $f(n) = O(g(n))$ и $f(n) = \Omega(g(n))$.
Для любых двух функций свойства $f(n) = O(g(n))$ и $g(n) = \Omega(f(n))$ равносильны.

1 задача
Доказать, что $2^{n+10} = O(2^n)$ и найти коэффициент c и n_0.
Решение
$f(n) = 2^{n+10}; g(n) = 2^n$
$2^{n+10} \leq c \cdot 2^n$
$1024 \cdot 2^n \leq c \cdot 2^n$
$c = 1024$ и $n_0 = 1$, то для любого $n \geq 1$ верно

2 задача
Доказать, что $2^{10n} \neq O(2^n)$.
Решение
$f(n) = 2^{10n}; g(n) = 2^n$
$2^{10n} \leq c \cdot 2^n \ |:2^n$
$2^{9n} \leq c$, мы не можем подобрать такое c и изначальное n_0, при которых это неравенство всегда будет выполняться

3 задача
Доказать, что $2n = o(n^2)$.
Решение
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2n}{n^2} = \lim_{n\to\infty} \frac{2}{n} = 0$, ч.т.д.

4 задача
Доказать, что $\frac{1}{2}n^2 - 3n = \Theta(n^2)$.
Решение
$0 \le c_1 \cdot n^2 \le \frac{1}{2}n^2 - 3n \le c_2 \cdot n^2 \mid : n^2$
$0 \le c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$
Нужно подобрать c_1 и $c_2 \ge 0$ $\forall n \ge n_0$
Берем n_0 такое, чтобы c_1, что оно было больше нуля $\Rightarrow$ минимальное $n_0 = 7 \Rightarrow c_1 = \frac{1}{14}$
$\lim_{n\to\infty} \frac{3}{n} = 0 \Rightarrow c_2 = \frac{1}{2}$
Ответ: $n_0 = 7; c_1 = \frac{1}{14}; c_2 = \frac{1}{2}$

5 задача
Поясните, почему утверждение "время работы алгоритма A равно как минимум $O(n^2)$"лишено смысла
Решение
$O(g(n))$ - это оценка сверху, некий максимум для нашей функции, поэтому утверждение можно переформулировать как "максимум функции как минимум равен n^2 то есть наше утверждение не является по сути оценкой функции

6 задача
Для любых двух функций $f(n)$ и $g(n)$ мы имеем $f(n) = \Theta(g(n))$ тогда и только тогда, когда $f(n) = O(g(n))$ и $f(n) = \Omega(g(n))$
Решение
По определению:
$f(n) = O(g(n)) \Rightarrow \exists c_1, n_1 > 0$ такие, что $0 \le f(n) \le c_1 \cdot g(n)$, $\forall n \ge n_1$
$f(n) = \Omega(g(n)) \Rightarrow \exists c_2, n_2 > 0$ такие, что $0 \le c_2 \cdot g(n) \le f(n)$, $\forall n \ge n_2$
Пусть $n_0 = max(n_1, n_2)$, тогда $0 \le c_2 \cdot g(n) \le f(n) \le c_1 \cdot g(n)$, $\forall n \ge n_0$
Таким образом, по определению $f(n) = \Theta(g(n))$

7 задача
Справедливы ли соотношения $2^{n+1} = O(2^n)$ и $2^{2n} = O(2^n)$
Решение
1) Необходимо найти такие $c, n_0 > 0$, что $\forall n \ge n_0$
$0 \le 2^{n+1} \le c \cdot 2^n \mid : 2^n$
$0 \le 2 \le c$
$\exists c = 3, n_0 = 1$: $0 \le 2^{n+1} \le 3 \cdot 2^n$, $\forall n \ge n_0 \Rightarrow 2^{n+1} = O(2^n)$ по определению
2) Покажем, что невозможно найти такие $c, n_0 > 0$, что $\forall n \ge n_0$
$0 \le 2^{2n} \le c \cdot 2^n \mid : 2^n$
$0 \le 2^n \le c$
Не существует таких $c, n_0 > 0$, что $0 \le 2^{2n} \le 2^n$, $\forall n \ge n_0 \Rightarrow 2^{2n} \ne O(2^n)$ по определению

8 задача
Докажите, что множество $o(g(n)) \cap \omega(g(n))$ является пустым
Решение
Пусть $\exists f(n)\ f(n) = o(g(n))$ и $f(n) = \omega(g(n)) \Rightarrow \forall c_1, c_2 > 0\ \exists n_0$:
$0 \le f(n) \le c_1 \cdot g(n)$ и $0 \le c_2 \cdot g(n) \le f(n)\ \forall n \ge n_0$
Таким образом, $\forall c_1, c_2 > 0\ \exists n_0$: $0 \le c_2 \cdot g(n) \le f(n) \le c_1 \cdot g(n)$
Если $g(n)$ асимптотически положительна, то пусть $c_2 > c_1$:
$0 \le c_2 \cdot g(n) \le c_1 \cdot g(n)$, что неверно.
Не существует такой $f(n)$, что $f(n) = o(g(n))$ и $f(n) = \omega(g(n))$
Примечание: грубо говоря, $f(n)$ не может быть одновременно строго больше и строго меньше $g(n)$

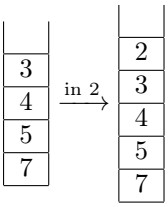
9 задача
Пусть $f(n)$ и $g(n)$ - асимптотически положительные функции. Докажите или опровергните справедливость каждого из приведенных ниже утверждений:
a. Из $f(n) = O(g(n))$ вытекает $g(n) = O(f(n))$
b. $f(n) + g(n) = \Theta(min(f(n), g(n)))$
Решение
a. $f(n) = O(g(n)) \Rightarrow \exists c_1, n_0 > 0$:
$0 \le f(n) \le c_1 \cdot g(n)$, $\forall n \ge n_0$ по определению
Положим, что $c_2 = \dfrac{1}{c_1}$, тогда получим $0 \le c_2 \cdot f(n) \le g(n)$, $\forall n \ge n_0 \Rightarrow g(n) = \Omega(f(n))$
b. Для докозательства утверждения необходимо найти c_1, c_2, n_0 большие нуля:
$c_1 \cdot min(f(n), g(n)) \le f(n) + g(n) \le c_2 \cdot min(f(n), g(n))$, $\forall n \ge n_0$
Левая часть очевидна
$f(n), g(n) \ge min(f(n), g(n)) \Rightarrow f(n) + g(n) \ge min(f(n), g(n)) \cdot c_1 = 1$
Рассмотрим правую часть. Пусть $f(n) \le g(n)$, тогда
$f(n) + g(n) \le c_2 \cdot f(n) \Rightarrow g(n) \le (c_2 - 1) \cdot f(n)$
Но в общем случае для функций $f(n), g(n)$ не существует такого c_2
Например, положим $f(n) = n$, $g(n) = 2^n$

10 задача
Пусть $f(n)$ и $g(n)$ - асимптотически положительные функции. Докажите или опровергните справедливость каждого из приведенных ниже утверждений:
a. Из $f(n) = O(g(n))$ вытекает $2^{f(n)} = O(2^{g(n)})$
b. $f(n) = O((f(n))^2)$
Решение
a. Рассмотрим $f(n) = 2n$ и $g(n) = n$
$2n = O(n)$, так как $\exists c = 2$: $0 \le f(n) \le c \cdot g(n)$
Покажем, что невозможно найти такие $c, n_0 > 0$ для любых $n \ge n_0$
$0 \le 2^{2n} \le c \cdot 2^n\ |{:}2^n$
$0 \le 2^n \le c$
Не существует таких $c, n_0 > 0$, что $0 \le 2^{2n} \le c \cdot 2^n$, $\forall n \ge n_0 \Rightarrow 2^{2n} \ne O(2^n)$ по определению
Таким образом, исходное утверждение неверно
b. Рассмотрим $f(n) = \dfrac{1}{n}$
Покажем, что невозможно найти такие $c, n_0 > 0$ для любых $n \ge n_0$
$0 \le \dfrac{1}{n} \le \dfrac{c}{n^2}\ |{\cdot}n^2$
$0 \le n \le c$
Не существует таких $c, n_0 > 0$, что $0 \le \dfrac{1}{n} \le \dfrac{c}{n^2}$, $\forall n \ge n_0 \Rightarrow \dfrac{1}{n} \ne O((\dfrac{1}{n})^2)$ по определению
Таким образом, исходное утверждение неверно

11 задача
Пусть $f(n)$ и $g(n)$ - асиптотически неотрицательные функции. Докажите с помощью базового определения Θ-обозначения, что $max(f(n), g(n)) = \Theta(f(n) + g(n))$
Решение
Мы хотим доказать, что
$0 \leq c_1 \cdot (f(n) + g(n)) \leq max(f(n), g(n)) \leq c_2 \cdot (f(n) + g(n))$, для $n \geq n_0$
Найти n_0, c_1 и c_2:
1) $f(n) + g(n) \geq max(f(n), g(n))$, так как функция асимптотически отрицательно
2) $f(n) \leq max(f(n), g(n))$ и $g(n) \leq max(f(n), g(n))$, значит:
$f(n) + g(n) \leq 2 \cdot max(f(n), g(n)) \Rightarrow 0,5 \cdot (f(n) + g(n)) \leq max(f(n), g(n))$
3) $0 \leq 0,5 \cdot (f(n) + g(n)) \leq max(f(n), g(n)) \leq 1 \cdot (f(n) + g(n)) \Rightarrow$
$\Rightarrow max(f(n), g(n)) = \Theta(f(n) + g(n))$, так как существует $c_1 = 0,5$ и $c_2 = 1$

12 задача
Покажите, что для любых действительных констант a и b, где $b > 0$, выполняется соотношение
$(n + a)^b = \Theta(n^b)$
Решение
Хотим доказать, что: $0 \leq c_1 \cdot n^b \leq (n + a)^b \leq c_2 \cdot n^b$ для $n \geq n_0$
Найти n_0, c_1 и c_2
1) $n + a \leq 2n$, при $|a| \leq n$
2) $\frac{n}{2} \leq n + a$, при $|a| \leq \frac{n}{2}$ $(2|a| \leq n)$
3) При $2|a| \leq n$: $0 \leq \frac{n}{2} \leq n + a \leq 2n$
4) Возведем в степень b: $0 \leq \frac{1}{2^b} n^b \leq (n + a)^b \leq 2^b n^b$
$(n + a)^b = \Theta(n^b)$ верно при $c_1 = (\frac{1}{2})^b, c_2 = 2^b, n_0 = 2|a|$

Стек



Битмапы

Способ представления множества какого-то чисел в виде массива из нулей и единиц.

5 задача
{1, 5, 7} - множество
Решение
Самое большое число = размер битмапа, в нашем случае, 7
Использовано всего 7 бит для представления 3-х чисел

1	0	0	0	1	0	1

- битмап нашего множества

Способ сжатия битмапов - WAH

n - размер слова
$\frac{n}{w-1}$ - количество слов

Имеем изначальный битмап: | 0 | 1 | 0 | 1 | 0 | $\cdots$ | 1 | 1 | 0 | 0 | и необходимо разбить этот битмап на слова (на небольшие битмапы, которые мы сможем сжать)

Принято брать 31 бит на размер нашего маленького битмапа, то есть мы берем изначальный битмап и делим его на слова по 31 биту. Если у последнего слова не хватает битов до 31, то мы просто дописываем нули ⇒ получаем кучу битмапов, и нам нужно их как-то сжать, чтобы они занимали меньше места.

6 задача

0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0

Решение
Сжимаем ряды, которые полностью состоят из нулей и единиц можно сжать следующим образом:
1) Первый бит: слово несжатое ('0') или сжатое слово ('1');
2) Какие ряды сжали: ряды нулей ('0') или ряды единиц ('1');
3) Остальные 30 бит - это запись количества рядом, которые мы сжали

В нашем случае ряд нулей сжали так:

1	0	0	0	0	$\cdots$	0	0	1	0

- '10' - двоичная запись двойки

А ряд единиц так:

1	1	0	0	0	$\cdots$	0	0	0	1

Минус такого способа в том, что ряд, где будет находиться только один ноль или только одна единица - мы сжать не сможем.

Способ сжатия битмапов - Concise

$p = log_2 w$ - количество бит, выделяемое на запись индекса единственного нуля или единицы в слове
$w - 2 - log_2 w$ - счёт слов
$w = 32$ - количество бит, которые выделяем под слово
Дальше считаем, сколько мы будем выделять бит на сжатие слова:
$w - 2 - log_2 w = 32 - 2 - 5 = 25$

7 задача
Сжать следующий битмап:

0	0	0	0	$\cdots$	0	1	0	0	0
0	0	0	0	$\cdots$	0	0	0	0	0

Решение
1) Первый бит = '1', так как это сжатое слово
2) Второй бит = '0', так как сжимаем слова с нулями
3) Следующие 5 бит = '00011', так как позиция единственной единицы ("лишнего"символа) равен 3
4) Оставшиеся 25 бит = двоичное представление двойки, так как сжимаем еще и следующий ряд
Получаем битмап:

1	0	0	0	0	1	1	0	0	$\cdots$	0	0	0	1	0

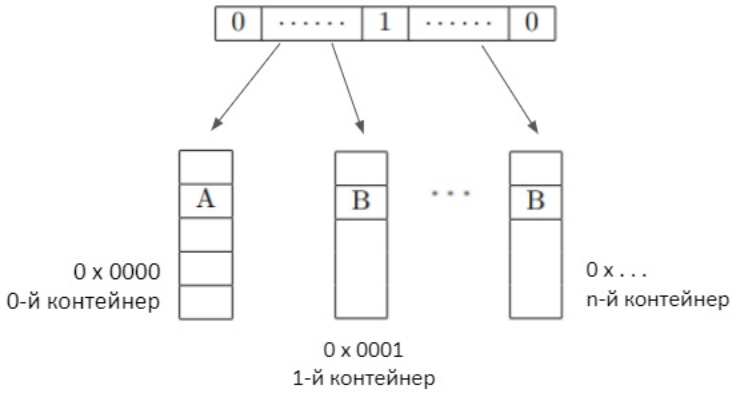
В список наших "хороших"(состоящий только из нулей или только из единиц) битмапов мы можем внести только один "плохой"(который содержит один ноль или одну единицу)

Roaring bitmap

Разбиваем наш изначальный битмап на слова, так называемые чанки
Размер чанка = 2^{16}
Мы должны правильно отсортировать эти чанки, то есть представить каждый чанк в одном из двух варинатнов:
1) Array - массив int'ов.
Сохраняем массив чисел, которые у нас лежат в множестве размера 2^{16}
Размер Array равен количеству единиц
2) Bitmap - массив нулей и единиц Размер Bitmap равен 2^{16}

Мы смотрим насколько наш чанк разряжен. Разряженные чанки сохраняем в Array. Если наш чанк содержит меньше чем 4096 единиц, то мы считаем этот чанк достаточно разряженным, чтобы сохранить его в Array. Если же единиц ≤ 4096, то сохраняем его в виде обычного битмапа размера 2^{16}

В итоге наш изначальный битмап представляется в виде списка контейнеров, который и представляет сжатый битмап.



8 задача

Числа от 0 до 100: {0, 100}

Решение

Всего 100 значений. Так как единиц $<$ 4096, то засовываем данное множество в Array-контейнер. Получаем чанк:

1	1	1	$\cdots$	1	1	0	0	0	0	$\cdots$	0	0

- Array-контейнер

0 1 $\cdots$ $\cdots$ 100 $\cdots$ $\cdots$ 2^{16} - индексы

9 задача

Множество $\{2^{16} + 2, \cdots, 2^{16} + 2i, \cdots, 2 \cdot 2^{16} - 2\}$

Решение

Данное множество записываем в Bitmap

Когда мы будем расшифровывать, нам нужно будет перевести их именно в то место битмапа, где они у нас были, поэтому будем добавлять к значению $2^{16} \cdot$ индекс (например, 1 x 0001), то есть мы получим смещение значений относительно нуля.

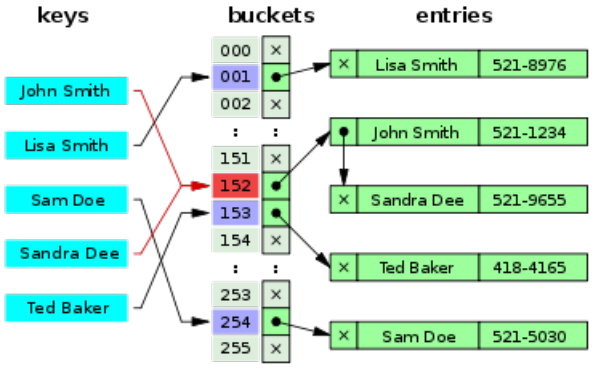
Hash и Hash-table

1
⋮
m
- Hash-таблица

$i = hash(key)$
Вставка, поиск и удаление работают за $O(1)$

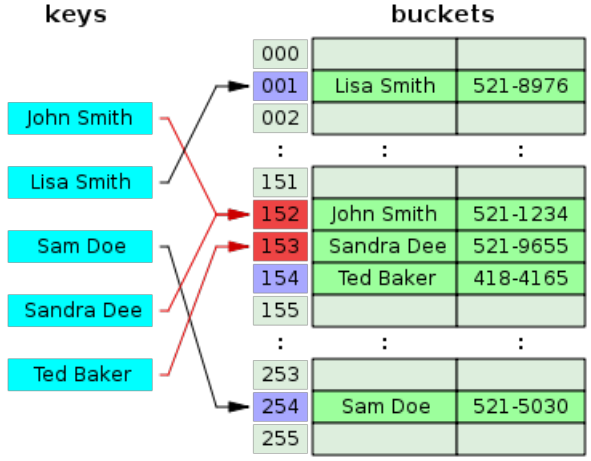
Необходимо вставить ключ key. Если key - переменная типа int, то её индекс = key % m (m - размер hash-таблицы). Если же на месте уже есть ключ, то вставляем это значение на то же место ⇒ образуется коллизия (например, ключи 1 и m+1)

Способы разрешения коллизий:
1) Взять другую хэш-функцию, которая будет лучше определять позицию в зависимости от значения
2) Способ разрешения коллизией с помощью цепочек



Если наши два элемента указывают на один индекс нашей таблицы, то эти ключи складываются в список. При поиске придётся проходить еще и по списку ⇒ увелечение времени работы алгоритма

3) Используя линейный/квадратичный



При поиске индекса мы не сразу вставляем key на позицию, а сначала смотрим занята ли ячейка хэш-таблицы. Если она занята, то мы каким-либо способом идем дальше по таблице и ищем свободные места линейным (проверяем каждый индекс) или квадратичным (увеличиваем каждый шаг на const) способом поиска

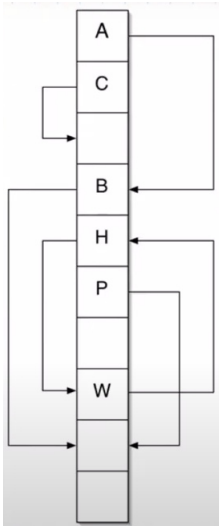
4) Использование второй хэш-функции
$h_1 = hash_1(key)$
$h_2 = hash_2(key)$
Вставляем элемент в одну из позиций, если там свободно.

Значение не вставляется, если:
1) когда доходим до конца таблицы
2) мы зациклились, то есть вернулись в изначальный индекс, полученный хэш-функцией

Кукушкин хэш

Изначально при вставке мы используем две функции хэширования:
$h_1 = hash_1(key)$ и $h_2 = hash_2(key)$

Если h_1 или h_2 свободны - вставляем в свободный индекс.
Если h_1 или h_2 заняты (то есть возникает коллизия), то выбираем рандомно h_1 или h_2, вставляем наш элемент на это место, при этом запоминаем элемент, стоявший на позиции до этого. Для того элемента применяем вторую хэш-функцию и перемещаем на новую позицию.



Пример:
$\mathrm{h}_1(B) = 2$
$\mathrm{h}_2(B) = 7$
Если в дальнейшем вытолкнут B, то B снова встанет на индекс, найденный первым хэшем

Вероятность коллизий в кукушкином фильтре сильно уменьшается

Минус: все может зациклиться $\Rightarrow$ это исправляется только изменением изначальных хэш-функций

Фильтр Блума

		Верная гипотеза	
		H_0	H_1
Результат применения критерия	H_0	H_0 верно принята	H_0 неверно принята (Ошибка *второго* рода)
	H_1	H_0 неверно отвергнута (Ошибка *первого* рода)	H_0 верно отвергнута

H_0 – нулевая гипотеза
H_1 – альтернативная гипотеза

0
1
1
0
1
1
0

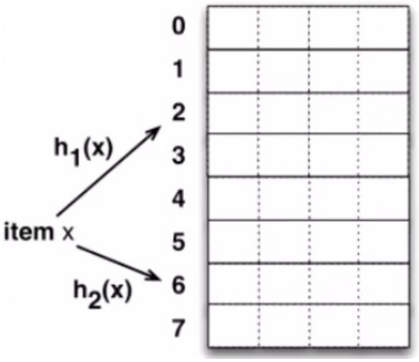
$hash(key)$ - указывает на 1 позицию
k - оптимальное количество хэш-функций
m - размер хэш-таблицы
n - количество данных, которые мы хотим вставить в фильтр
$k = \frac{m}{n} \cdot \ln 2$

При вставке мы применяем все наши хэш-функции и на все позиции (которые мы получили из хэш-функций) ставим единицы

Когда проверяем есть ли элемент в фильтре: точно можем сказать, если значения нет (если его нет - на одной из позиций будет стоять ноль)
На каком-то месте может стоять единица не этого элемента, то есть на этот индекс может указывать хэш-функция какого-то другого значения, - это называется ошибкой второго рода (false positive) $\rightarrow$ возможно, там есть это значение, но мы не можем этого утверждать
Ошибка первого рода - если мы ошибочно отвергаем верную гипотезу

Кукушкин фильтр

Фильтр представляет из себя список ячеек, куда мы можем класть значения. В эти ячейки кладем значение не полностью, а только часть значение - отпечаток значения (например, 7 бит от ключа)



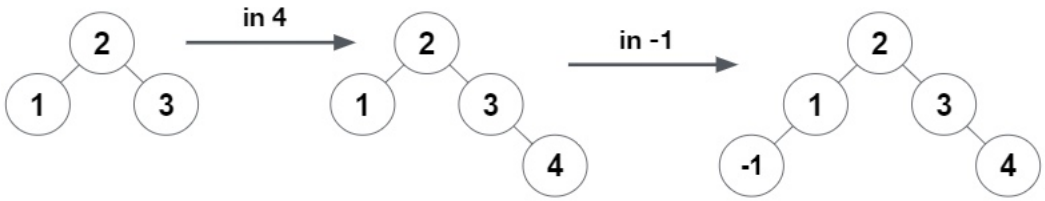
Используем только две хэш-функции:
$i_1 = hash(x)$
$i_2 = i_1 \oplus hash(f)$
f - 7 бит ключа $\rightarrow$ отпечаток значения

Вставка элемента. Получаем две позиции:
1) если место есть $\rightarrow$ просто вставляем
2) если же места нет $\rightarrow$ вытесняем какое-то значение, вставляем элемент на его место и ищем позицию для вытесненного значения

В каждом индексе может лежать несколько элементов, то есть используем контейнеры

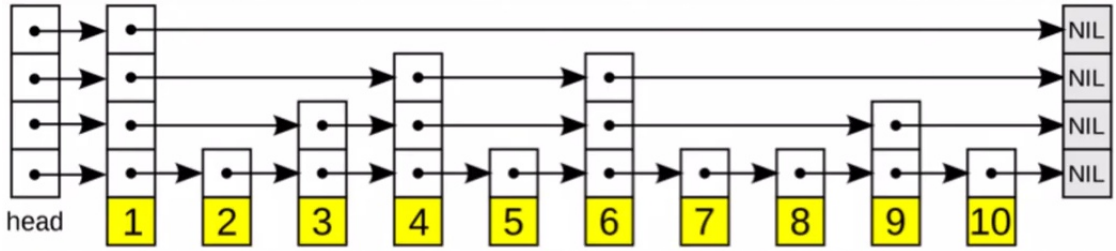
Бинарные деревья

Двоичное дерево: у каждого узла либо нет, либо 1-2 ребенка и при этом: справа всегда значение больше чем корень, слева - значение меньшее корня



Skip-list

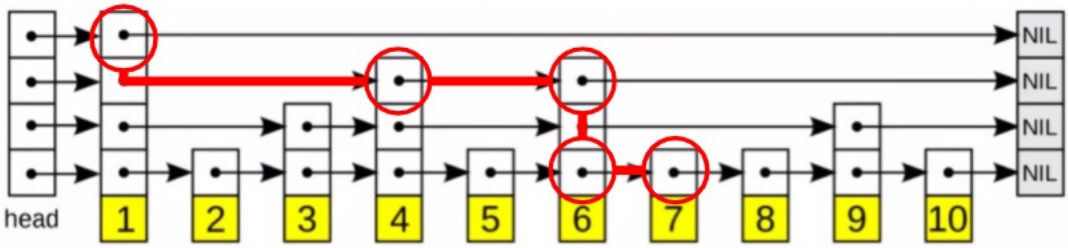
Skip-list - лист, состоящий из нескольких уровней



Самый нижний уровень - полный список, в котором содержатся все элементы
На следующих уровнях: элемент вставляется с какой-то вероятностью - p (вероятность обычно задается изначально)

Когда мы вставляем новый элемент в список, то он вставляется в нижний уровень (полный список), и дальше на каждый следующий уровень вычисляется по вероятности, будет ли элемент там присутствовать или нет

Поиск: идем, начиная с верхнего уровня.
Пример: поиск элемента '7'

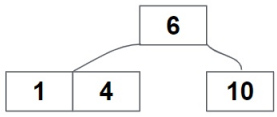


Что хорошего: при константном p - поиск, вставка и удаление элемента проходят за $log_2 n$

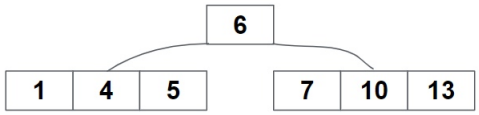
B-дерево

Свойства B-дерева:
1) Изначальный параметр - t (минимальная степень B-дерева), где $t \geq 2$
2) Корень может содержать от 1 до ($2t$ - 1) ключей и от 0 до $2t$ детей
3) Обычные узлы содержат $[t-1; 2t-1]$ ключей и $[2; 2t]$ детей
4) Узел содержит ключи от k_1 до k_n. От каждого ключа отходит два ребенка, если это не лист. $i-$ ребенок содержит ключи из отрезка $[k_{i-1}; k_i]$
5) Сбалансированное дерево $\Rightarrow$ все листья лежат на одном уровне
6) Все ключи в узлах отсортированы по неубыванию

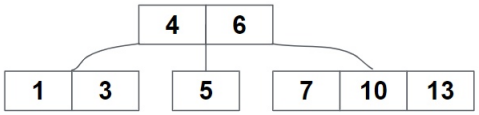
10 задача
Вставить элементы 4, 6, 10, 1, 5, 13, 7, 3, 2, 8, 9 в B-дерево, где минимальная степень $t = 2$
Решение
Вставить 4, 6, 10 (будет корнем нашего дерева): | 4 | 6 | 10 |
Вставить 1 (середина, то есть 6, поднимаетсяя выше и становится корнем дерева, а узел разбивается на 2 ребенка)



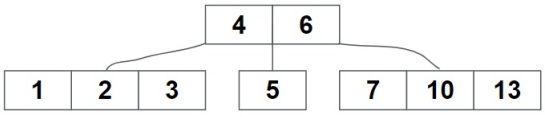
Вставить 5, 13, 7 (просто вставляем, так как лист)



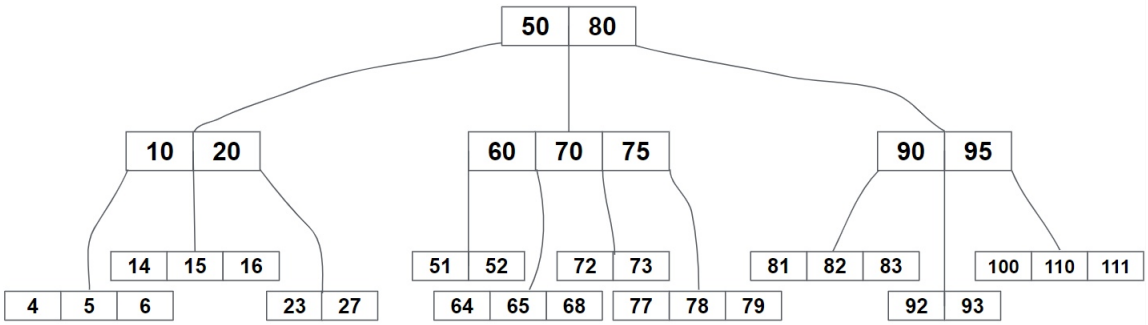
Вставить 3 (середину (4) поднимаем и разделяем детей)



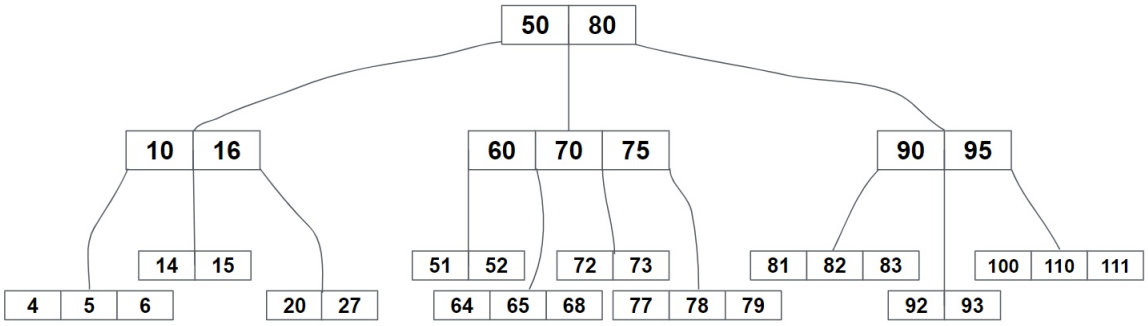
Вставить 2 (просто вставляем, так как лист)



11 задача
Удаление в следующем B-дереве ($t = 3$):

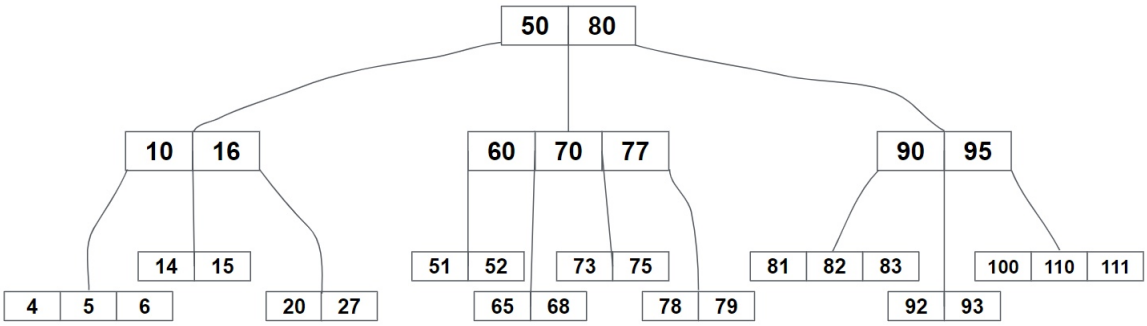


Решение
Удалить 23: не можем просто так удалить ключ. Нам нужно "попросить"ключ, чтобы не нарушалась балансировка.
Рассмотрим братьев. Если это наш левый брат, то мы берем его наибольший ключ. Двигаем его вверх по дереву. Затем двигаем родителя на позицию, откуда удаляем ключ.



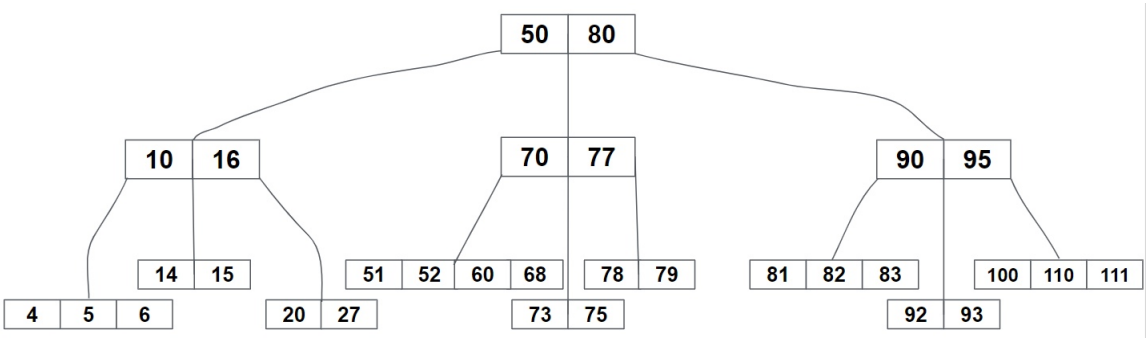
Удалить 64: если мы хотим удалить ключ и он находится в листе, в котором больше чем $(t-1)$ ключей, то просто удаляем.

Удалить 72: аналогично удалению 23, но берем минимальное значение правого брата.

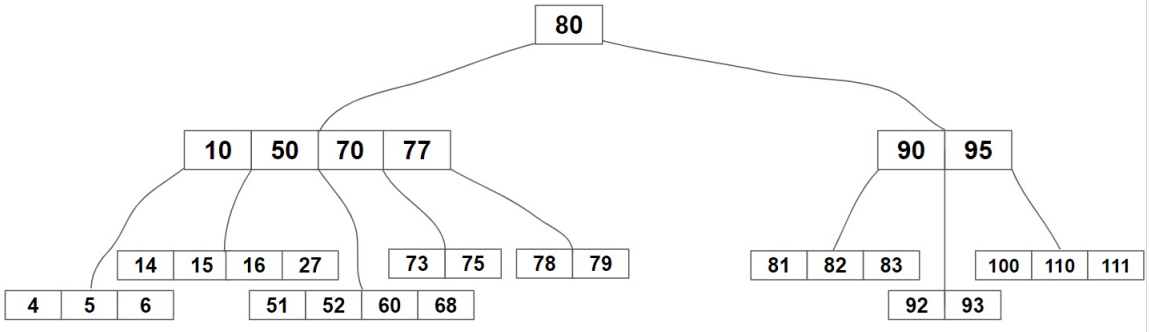


Удалить 65: если у обоих братьев минимальное количество ключей (то есть у обоих $t-1$). Выбираем любого из братьев, спускаем их родителя и склеиваем братьев $\rightarrow$ из этого узла

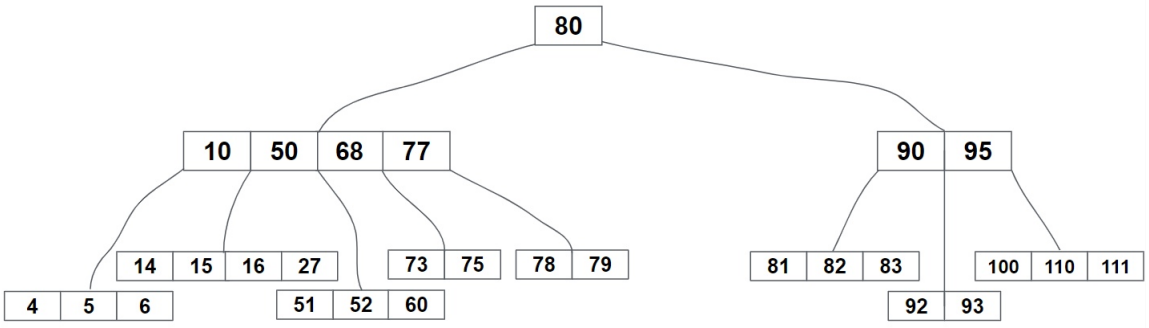
просто удаляем 65.



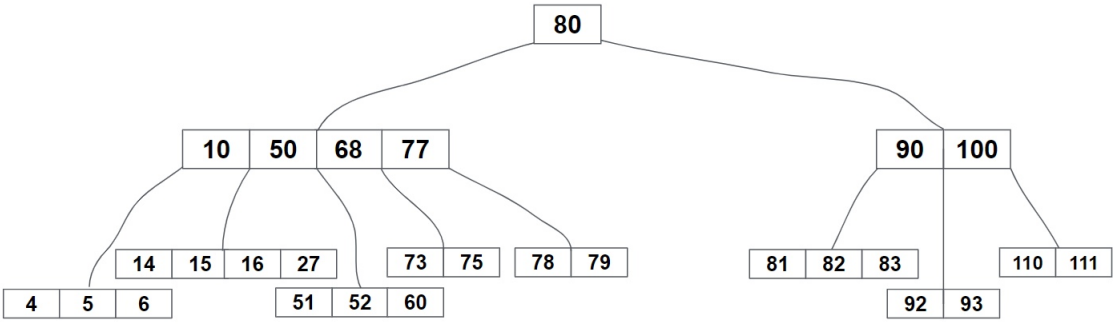
Удалить 20: у братьев и родителей значение забрать не можем. Тогда склеиваем двух братьев и удаляем в склееном ребенке 20. Затем запрашиваем ключ у брата нашего родителя - не можем $\Rightarrow$ снова склеиваем и забираем значение у корня.



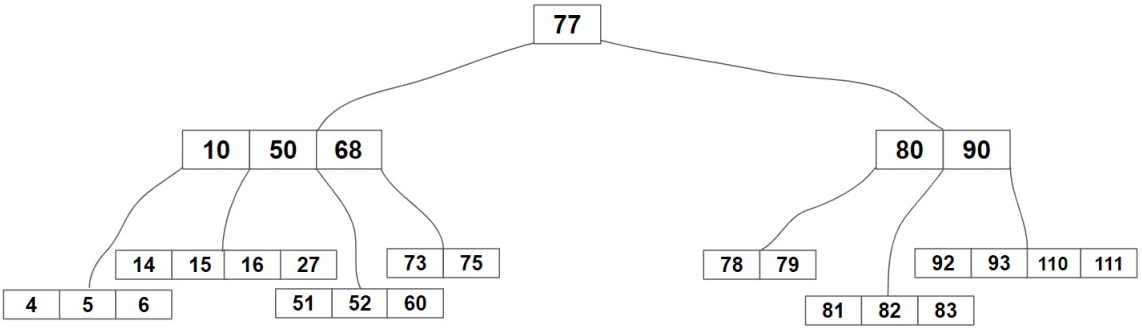
Удалить 70: необходимо найти такой ключ, который встанет на наше место. Проходимся по левому и правому поддереву. Если идем по левому поддереву, то ищем там наибольший ключ (68). Если идем по правому поддереву, то берем наименьший ключ (75). Так как из правого поддерева мы не можем взять ключ (там минимально $t - 1$ значений), то берем максимальное из левого поддерева.



Удалить 95: аналогично удалению 70.



Удалить 100: у левого и правого поддеревьев $t-1$ количество ключей $\Rightarrow$ не можем взять у них ключ, поэтому возвращаемся к операции склейки. Спускаем 100 в склеенного ребенка и там просто удаляем. После этого берем ключ у брата родителя (90), перетащив его через корень.



Удалить 77: 77 - наш корень. У правого и левого поддеревьев максимальный и минимальный элементы лежат в узлах с минимальным $t-1$ количеством значений, то склеиваем два дочерних узла корня.
Корень имеет максимальное количество значений $t-1=5$, но так можно $\Rightarrow$ все супер!