Оглавление

Лекция 12: АВЛ-дерево, хэширование, B-деревья

29.11.2023

Глава 1

Информационный поиск и организация информации

1.1 АВЛ-дерево

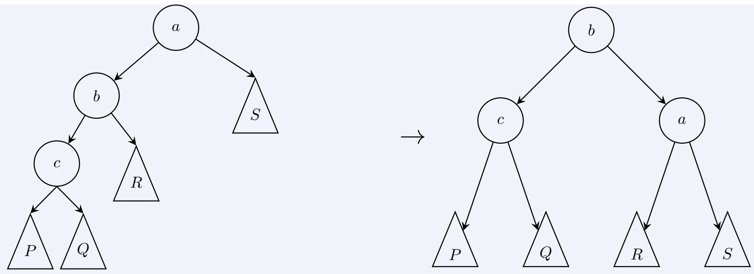
Определение 1. Высота дерева — длина пути (количество ребер) от корня до листьев.
Вершина дерева называется сбалансированной, если высоты ее правого и левого поддеревьев различаются не более чем на 1.
Двоичное дерево называется сбалансированным, если каждая его вершина сбалансирована.
Чтобы при добавлении элементов в дерево время поиска росло не слишком быстро, его нужно балансировать.
АВЛ-дерево (самобалансирующееся дерево) позволяет поддерживать приблизительное равенство двух ветвей двоичного дерева во всех его узлах, затрагивая за раз не более трех из них.

Алгоритм. (Балансировка)

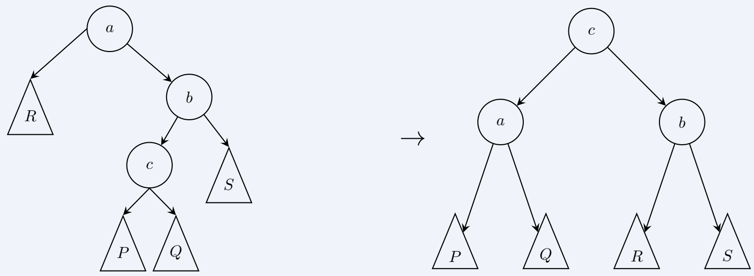
1. добавляем вершину d
2. проверяем вершины на пути от d к корню
3. обозначим через a, b, c (b, c – потомки a) первые несбалансированные вершины на пути от d к корню (баланс может нарушится только в трех или в двух вершинах)
4. выполняем перебалансировку (поворот):

2 вида перебалансировки:

1. узел b "вытягиваем"вверх на место a, так, что a становится потомком b (b соотвественно "прикрепляется"к узлу выше, если он был). Поддерево на b становится поддеревом a.



2. узел c "вытягиваем"вверх, a – левый ребенок, b – правый ребенок. Левое поддерево c прикрепляется к a, правое – к b.



Теорема 1. После операции поворота полученное дерево окажется сбалансированным.

Доказательство. (да, я просто списал с Романовского :c) Обозначим за $H(i)$ максимальную длину пути от добавленной вершины до вершины i, за h_k - максимальную высоту поддерева K.

По предположению:

$$H(a) = H(b) + 1 = H(c) + 2,$$

$$h_s = H(b) - 2 = H(c) - 1$$

Рассмотрим отдельно 2 случая:

1. $h_R < H(c)$. Тогда после первой балансировки дерева $H(a) = h_s + 1 = H(c)$ и высота верхней вершины стала меньше

2. в этом случае высота так же уменьшается на 1, так как определяющие максимальный путь поддеревья R и Q поднялись на одну ступень выше.

□

1.2 Хэширование

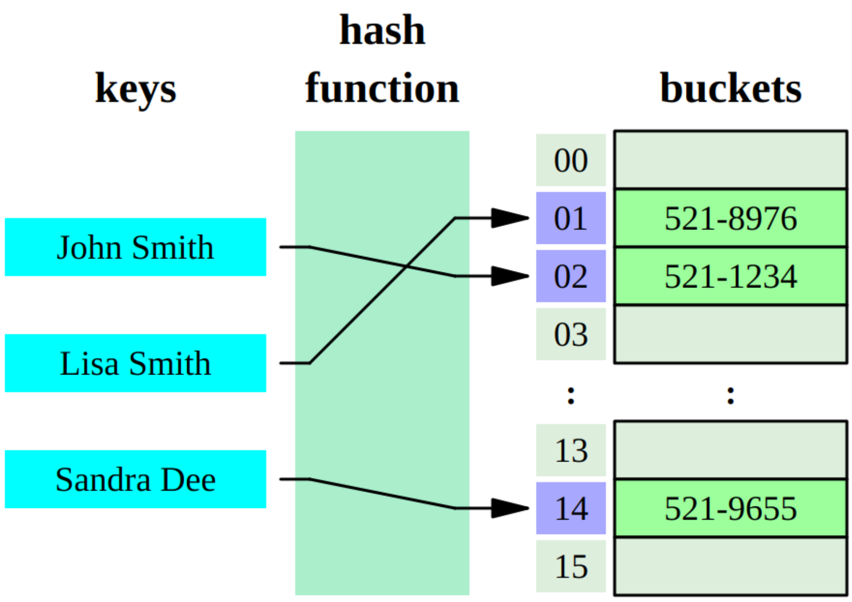
Определение 2. Хеш — это функция, сопоставляющая объектам какого-то множества числовые значения из ограниченного промежутка.

«Хорошая» хеш-функция:

- Быстро считается — за линейное от размера объекта время;
- Имеет не очень большие значения — влезающие в 64 бита;
- «Детерминировано-случайная» — если хеш может принимать n различных значений, то вероятность того, что хеши от двух случайных объектов совпадут, равна примерно $\frac{1}{n}$.

Обычно хеш-функция не является взаимно однозначной: одному хешу может соответствовать много объектов, т.е. она сюръективная.

Пример:



1.3 В-деревья

Определение 3. В-дерево — сильноветвящееся сбалансированное дерево поиска, в котором каждый узел содержит множество ключей и имеет более двух потомков.

Элементы находятся в листьях, остальные уровни представляют собой иерархию индексов, они указывают путь, по какой ветке двигаться, чтобы прийти в тот лист, где находится нужная запись.

Количество ключей в узле и количество потомков зависит от порядка В-дерева. Для каждого дерева фиксируется какое-то число t.

В-дерево обладает следующими свойствами:

1. Все листья находятся на одном и том же уровне, т.е. имеют оди-

наковую глубину (B-дерево идеально сбалансировано)

2. Каждый узел, кроме корневого, должен иметь, как минимум $t-1$, и не более $2t-1$ ключей
3. Если узел не является листом, то он имеет детей (кол-во ключей в узле $+ 1$) штук
4. Все ключи в узле должны располагаться в порядке возрастания их значений

Замечание. Примеры для всех операций в B-дереве очень долго писать, можете потыкать тут (открывайте в pdf)

Алгоритм. (Поиск в B-дереве)

1. Считать элемент для поиска
2. Сравнить искомый элемент с первым значением ключа в корневом узле дерева.
3. Если они совпадают, вернуть значение.
4. Если они не совпадают, проверить больше или меньше значение элемента, чем текущее значение ключа.
5. Если искомый элемент меньше, продолжить поиск по левому поддереву.
6. Если искомый элемент больше, сравнить элемент со следующим значением ключа в узле и повторять Шаги 3, 4, 5 и 6 пока не будет найдено совпадение или пока искомый элемент не будет сравнен с последним значением ключа в узле-листе.
7. Если последнее значение ключа в узле-листе не совпало с искомым, вернуть $null$.

Алгоритм. (Добавление элемента) В B-дереве новый элемент может быть добавлен только в узел-лист. Вставка происходит следующим образом:

1. Проверить пустое ли дерево.
2. Если дерево пустое, создать новый узел с новым значением ключа и его принять за корневой узел.
3. Если дерево не пустое, найти подходящий узел-лист, к которому будет добавлено новое значение, используя логику дерева двоичного поиска.
4. Если в текущем узле-листе есть незанятая ячейка, добавить но-

вый ключ-значение к текущему узлу-листу, следуя возрастающему порядку значений ключей внутри узла.

5. Если текущий узел полон и не имеет свободных ячеек, разделите узел-лист, отправив среднее значение родительскому узлу. Повторяйте шаг, пока отправляемое значение не будет зафиксировано в узле.

6. Если разделение происходит с корнем дерева, тогда среднее значение становится новым корнем дерева и высота дерева увеличивается на единицу.

Алгоритм. (Удаление элемента)

1. Если корень является листом (в дереве только один узел), просто удаляем ключ из этого узла.

2. Находим узел x, содержащий ключ, запоминая путь к нему.

3. Если x — лист, удаляем ключ из x. Если в x осталось не меньше $t - 1$ ключей, процедура завершается.

4. Если соседний правый узел имеет не менее t ключей, переносим ключ-разделитель в x и первый ключ соседа на место разделителя. Если соседний правый узел не подходит, но соседний левый узел имеет не менее t ключей, аналогично переносим ключ-разделитель и последний ключ соседа. Если ни один из соседей не подходит, объединяем x с одним из соседей и перемещаем разделяющий ключ в объединенный узел.

5. Если после объединения в родительском узле остается $t-2$ ключей и это не корень, повторяем процедуру для родителя.

6. Если в результате в корне осталось от 1 до $t - 1$ ключей, дополнительных действий не требуется. Если в корне не осталось ключей, исключаем корневой узел и делаем его единственного потомка новым корнем дерева.

7. Если x не является листом, удаляем самый правый ключ из поддерева i-го потомка x или самый левый ключ из поддерева $(i+1)$-го потомка x. Заменяем удаленный ключ на место ключа K.