ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

*«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

КУРСОВАЯ РАБОТА

**Алгоритм «B\* – Дерево»**

по дисциплине «Объектно-ориентированное программирование»

|  |  |  |
| --- | --- | --- |
| Выполнил  Студент  гр. 3331506/90401 | *(подпись)* | Назаренко И.И. |
| Работу принял | *(подпись)* | Ананьевский М.С. |

Санкт-Петербург

2022 г.

**Введение**

В деревьях поиска, таких как двоичное дерево поиска, AVL дерево, красно-чёрное дерево и т.п. каждый узел содержит только одно значение (ключ) и максимум двое потомков. Однако есть особый тип дерева поиска, который называется B-дерево. В нем узел содержит более одного значения (ключа) и более двух потомков. B-дерево считается сбалансированным по высоте. Для B-дерева всегда задается порядок m, который определяет его ветвистость. Обычно от 50 до 2000.

B-деревья имеют существенные преимущества по сравнению с альтернативными реализациями, когда время доступа к данным узла значительно превышает время, затрачиваемое на обработку этих данных, потому что тогда стоимость доступа к узлу может быть амортизирована по нескольким операциям внутри узла. Обычно это происходит, когда данные узла находятся во вторичном хранилище, например на дисках. Благодаря максимизации количества ключей в каждом внутреннем узле высота дерева уменьшается, а количество обращений к дорогим узлам уменьшается.

**B\*-дерево** — разновидность [B-дерева](https://ru.wikipedia.org/wiki/B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), в которой каждый узел дерева заполнен не менее чем на ⅔ (в отличие от B-дерева, где этот показатель составляет 1/2). Для выполнения требования «заполненность узла не менее 2/3», приходится отказываться от простой процедуры разделения переполненного узла, как в обычном B-дереве. Вместо этого происходит «переливание» в соседний узел. Если же и соседний узел заполнен, то ключи приблизительно поровну разделяются на 3 новых узла. Самой сложной операцией для B\*-дерева является удаление узлов, поэтому такой вид деревьев предпочтителен, когда основные операции, совершаемые над ним – поиск и добавление новых узлов. Основные свойства B\*-дерева:

*Свойство 1:* Глубина всех листьев одинакова.  
 *Свойство 2:* Каждый узел, за исключением корневого, имеет не более m потомков.  
 *Свойство 3:* Каждый узел, за исключением корневого и листьев, имеет не менее потомков.  
 *Свойство 4:* Корневой узел имеет не менее 2 и не более потомков  
 *Свойство 5:* Все ключи в узле должны располагаться в порядке возрастания их значений.

*Свойство 6*: Узлы, не являющиеся листьями и имеющие k потомков, содержать k-1 ключ.

На рисунке 1 изображено B\*-дерево с порядком, равным 4. При таком порядке корневой узел должен иметь от 1 до 5ти ключей, остальные узлы – от 2х до 3х ключей.

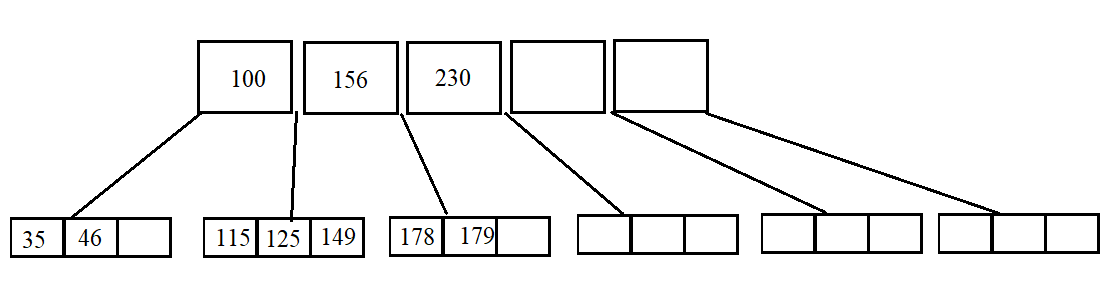


Рисунок 1 – Пример структуры B\*-дерева

**Описание алгоритма**  
Над B\*-деревом можно проводить следующие операции:

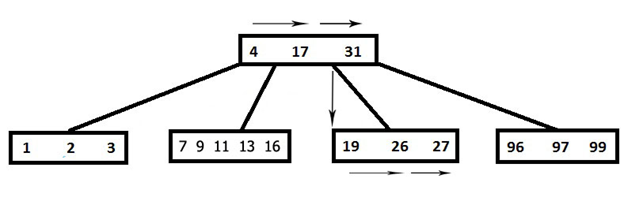
1. Поиск
2. Вставка
3. Удаление

***Алгоритм поиска***

Операция поиска осуществляется довольно быстро благодаря сбалансированности дерева

Поиск в B-дереве очень схож с поиском в бинарном дереве, только здесь мы должны сделать выбор пути к потомку не из 2 вариантов, а из нескольких. В остальном — никаких отличий. На рисунке ниже показан поиск ключа 27. Поясним иллюстрацию (и соответственно стандартный алгоритм поиска):

* Идем по ключам корня, пока меньше необходимого. В данном случае дошли до 31.
* Спускаемся к ребенку, который находится левее этого ключа.
* Идем по ключам нового узла, пока меньше 27. В данном случае – нашли 27 и остановились.

  
  
 Операция поиска выполняется за время O(t logt n), где t – минимальная степень. Важно здесь, что дисковых операций мы совершаем всего лишь O(logt n)!

***Алгоритм добавления***

  В этом алгоритме при переполнении листа обязательно используется переливание на одну из соседних страниц. Когда соседние листы заполнены, выполняется расщепление не одной страницы, а двух соседних страниц. На новый лист переносится треть ключей с каждого из двух заполненных листов, так что каждый лист B\*-дерева будет заполнен по крайней мере на 2/3, а не на 1/2, как для обычного B-дерева. Корневой лист B\*-дерева должен при этом быть большего размера, чем остальные, и он может быть заполнена не более чем на 4/3 от объема обычного листа. При расщеплении корня он дает два обычных листа, заполненных на 2/3, а новый корень, как и для B-деревьев, содержит только один ключ.

Рассмотрим примеры добавления узлов в B\*-дерево. Будем рассматривать пример для порядка m=4. В таком случае в корне должно быть от 1 до 4 ключей, в остальных узлах от 2х до 3х ключей.

**1 сценарий**

У нас имеется узел, который является листом и в нем еще есть свободные места. Тогда просто идем по порядку, сравнивая значения и вставляем наш ключ в подходящее место.



Рисунок 2 – до добавления



Рисунок 3 – после добавления

**2 сценарий**

У нас имеется корневой узел, он является листом и он полностью заполнен. Хотим добавить еще ключ с номером 5.

**

Рисунок 4 – до добавления

Выделяем среднее значение из имеющихся и выносим этот ключ наверх, создавая новый узел. Оставшиеся значения раскидываем в 2 узла, которые являются потомками узла, образованного ранее.

**

Рисунок 5 – после добавления

**3 сценарий**

У нас имеется полностью заполненный узел, в который нам нужно добавить еще ключ 3.

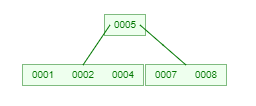


Рисунок 6 – до добавления

Вместо разделения узла, как это делается с обычным B-деревом, мы смотрим, есть ли место в правом соседнем узле, и если место есть, то перераспределяем ключи между этими узлами. Если в правом узле нету места, то проверяем таким же образом левый.

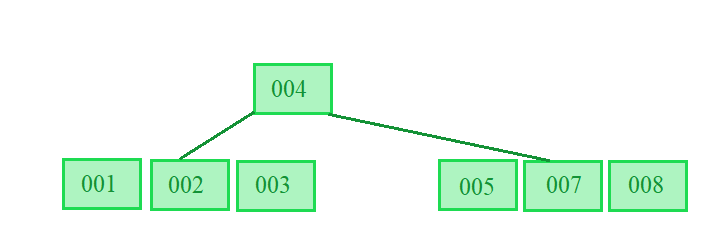


Рисунок 7 – после добавления

Если же нам нужно добавить ключ, а узел и его соседи заполнены, а в родительском узле еще есть свободное место, тогда происходит разделение 2х этих узлов одновременно, с добавлением ключей к родительскому узлу.

Принцип такой: мы выстраиваем в порядке возрастания все имеющиеся ключи в 2х узлах, родительский ключ, и ключ, который мы хотим добавить. Добавляем в первый дочерний узел первые ключей. Следующий ключ будет первым родительским ключом. Во второй дочерний узел заливаем следующие ключей. Следующий ключ будет вторым родительским ключом. Добавляем оставшиеся ключи в третий дочерний узел. Готово, мы разделили 2 заполненных узла на 3 узла, наполненность которых не меньше 2/3. Добавим для примера ключ 6.

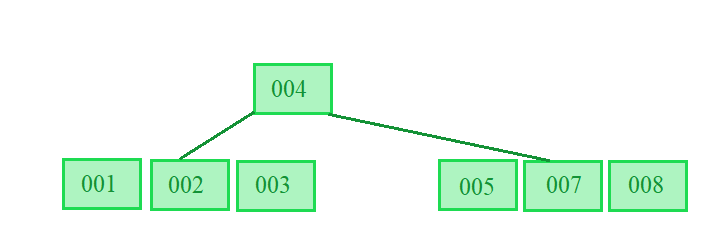


Рисунок 8 – до добавления

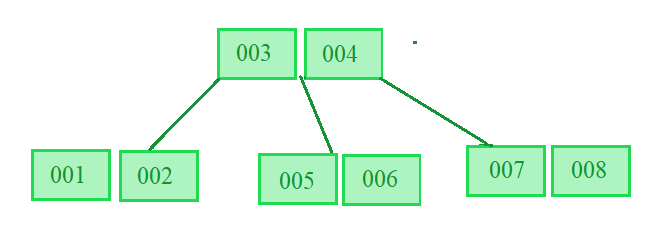


Рисунок 9 - после добавления

**4 сценарий**

Если нам нужно добавить ключ, а узел, его сосед и родительский узел заполнены, то происходит разделение родительского узла. Добавим для примера ключ 7.

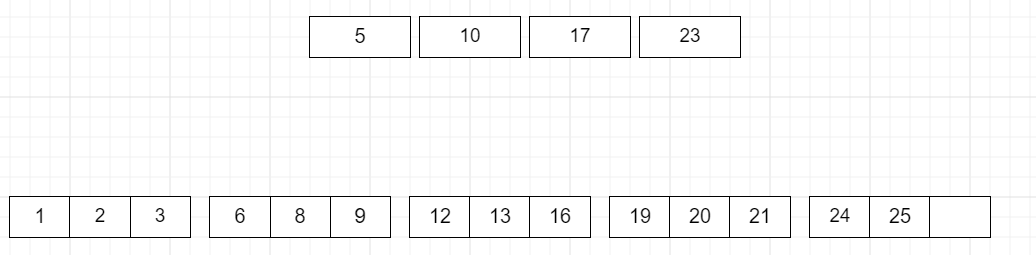


Рисунок 10 – до добавления

Рассмотрим выделенную область. Проделываем условно все операции из предыдущего сценария. В итоге у нас появился лишний ключ в родительском узле.

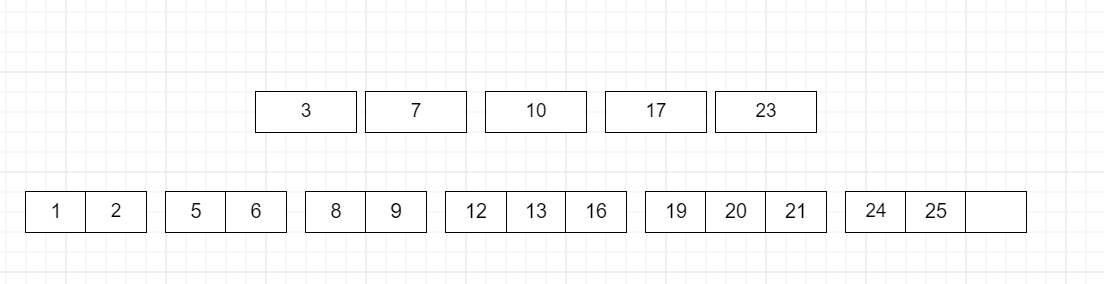


Рисунок 11 - Выделился лишний ключ

После этого перестраиваем дерево, выделяя средний по значению узел наверх. Меняем указатели.

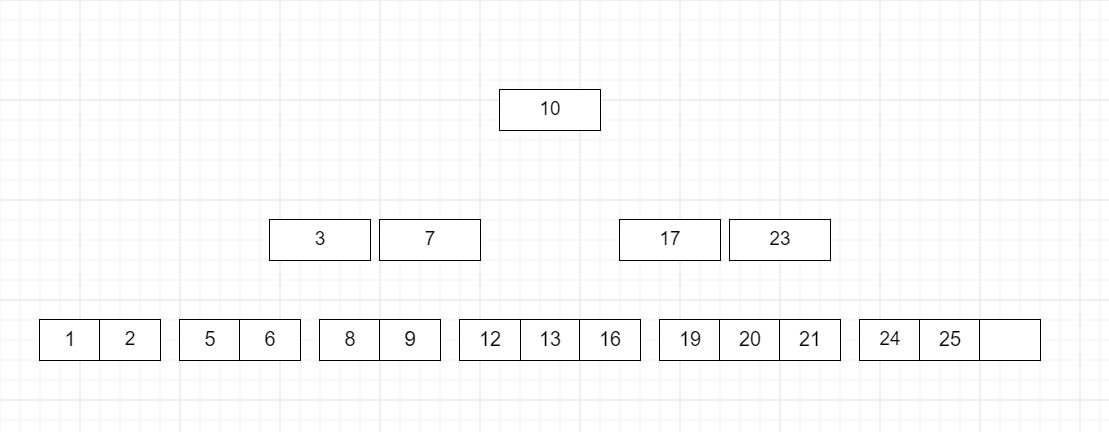


Рисунок 11 – Перестроенное дерево

***Алгоритм удаления***

**1 сценарий**

Ключ, который нам нужно удалить, находится в листе и при удалении ключа лист останется заполненным на 2/3. Тогда мы можем просто удалить ключ. Например, удалим значение 4.

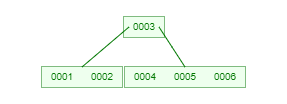


Рисунок 12 – Дерево до удаления

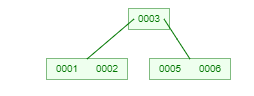


Рисунок 13 – Дерево после удаления

**2 сценарий**

Значение надо удалять из листа, а он останется заполнен меньше, чем на 2/3 после простого удаления. В таком случае смотрим на соседей слева и справа. Если в них заполнение более плотное, тогда можем удалить нужный ключ, а после выполнить переливание между узлами. Допустим, хотим удалить значение 4.

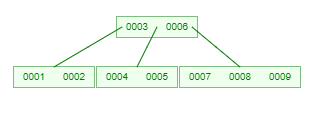


Рисунок 14 – Дерево до удаления

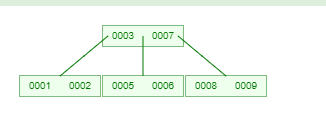


Рисунок 15 – Дерево после удаления

**3 сценарий**

У узла минимальное количество ключей, тогда можем, удаляя, объединить 2 соседних узла, выполняя переливание с родительским узлом. Удалим, например, значение 5.

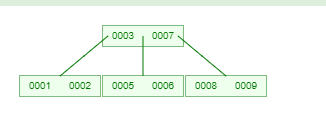


Рисунок 14 – Дерево до удаления

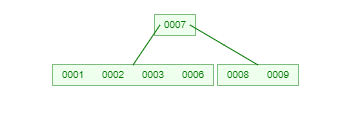


Рисунок 15 – Дерево после удаления

**4 сценарий**

Если удаляем значение из узла, который не является листом, тогда нам нужно найти дочерний узел, в котором допустимое количество ключей, чтобы один можно было удалить. Тогда выполняем переливание.

**5 сценарий**

Если узел не является листом, у его дочерних узлов не имеется лишних мест. Если кол-во дочерних узлов нельзя уменьшить, то обращаемся к родительскому узлу, выполняем переливание между ним и его другими дочерними узлами. Удалим, например, узел 18.

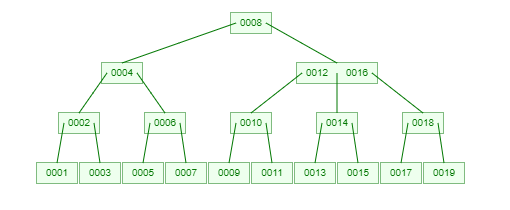


Рисунок 16 – Дерево до удаления

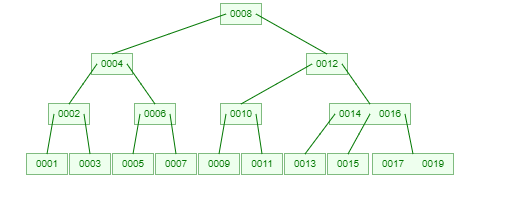


Рисунок 17 – Дерево после удаления

**7 сценарий**

У родительского или дочерних узлов уже нечего занимать, тогда придется уменьшать высоту дерева и полностью его перестраивать. Например, удалим ключ 6.

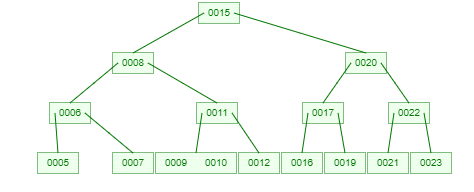


Рисунок 16 – Дерево до удаления

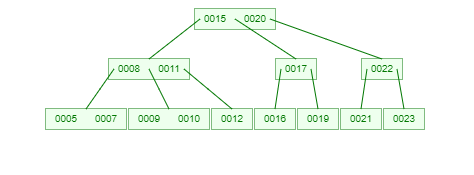


Рисунок 17 – Дерево после удаления

**Временная сложность алгоритма**

Временная сложность заявлена, как , где n – количество узлов, t – степень ветвления.

Построим графики для добавления, удаления элементов и поиска по дереву.

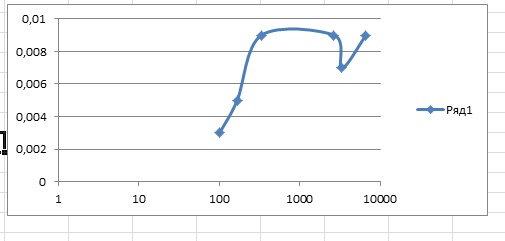


Рисунок 18 – Временная сложность добавления

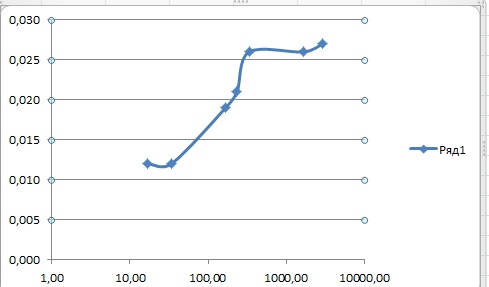


Рисунок 19 – Временная сложность поиска

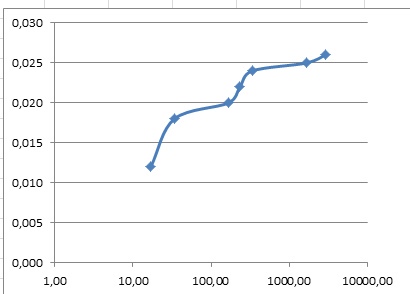


Рисунок 20 – Временная сложность удаления

Графики пока не успеваю выйти на логарифмический уровень, скорее всего для этого нужно большие объемы данных, но в принципе уже прослеживается некая зависимость: графики находятся примерно на одном уровне, потом скачком переходят на следующий, связано это с увеличением высоты дерева.

**Список литературы**

1. Дональд Кнут. 3. Сортировка и поиск // Искусство программирования

2. B-дерево [электронный ресурс] // <https://ru.wikipedia.org/wiki/B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>

3. B-Tree Vizualization [Электронный ресурс] // https://www.cs.usfca.edu/~galles/visualization/BTree.html

4. B\*-Trees implementation in C++ // <https://www.geeksforgeeks.org/b-trees-implementation-in-c/>

5. Сбалансированное дерево поиска B-tree [электронный ресурс] // <https://habr.com/ru/post/337594/>

6. B-Tree [электронный ресурс] // <https://habr.com/ru/post/114154/>