ITIVITI

LET IT TRADE

Алгоритмы и структуры данных

Косяков Михаил Сергеевич
к.т.н., доцент кафедры ВТ

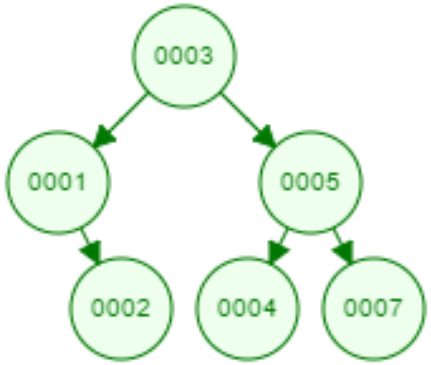
УНИВЕРСИТЕТ ИТМО

Содержание курса

- Введение в теорию алгоритмов
- Алгоритмы сортировок
- **Структуры данных**
  - Линейные структуры
  - **Бинарные деревья поиска**
  - Хеши и хеш-функции
- Алгоритмы на графах
  - Обходы графов в ширину и глубину
  - Минимальные остовные деревья
  - Поиск кратчайших путей в графе

Вставка в БДП

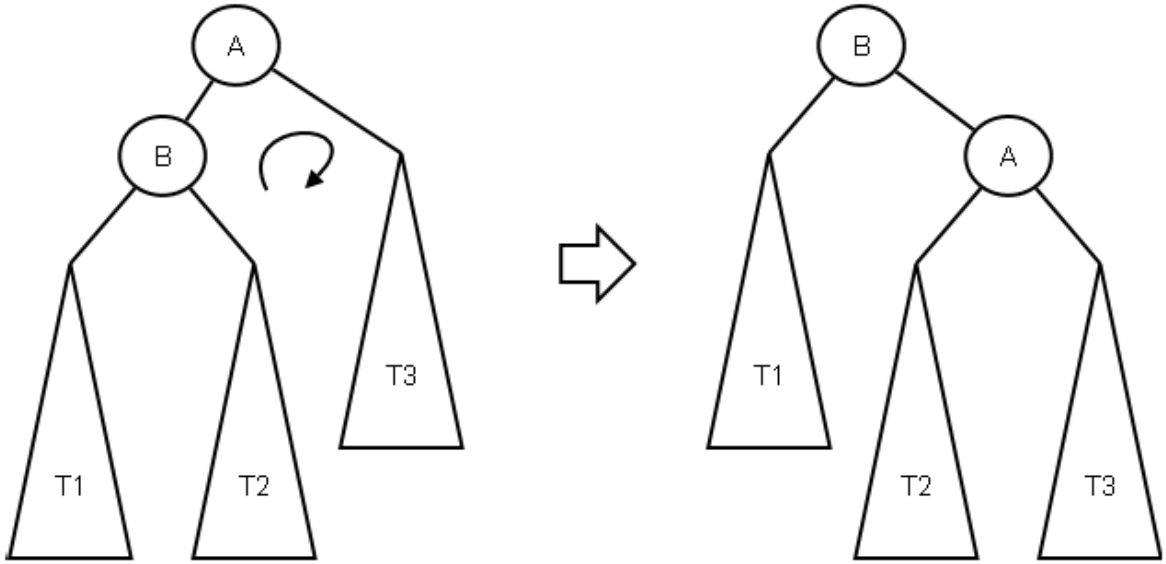
- Поиск отсутствующего ключа заканчивается на пустом поддереве
- Заменим пустое поддерево вставляемым узлом
- Вставка в корень дерева?
  - ключа 6 …



https://www.cs.usfca.edu/~galles/visualization/BST.html

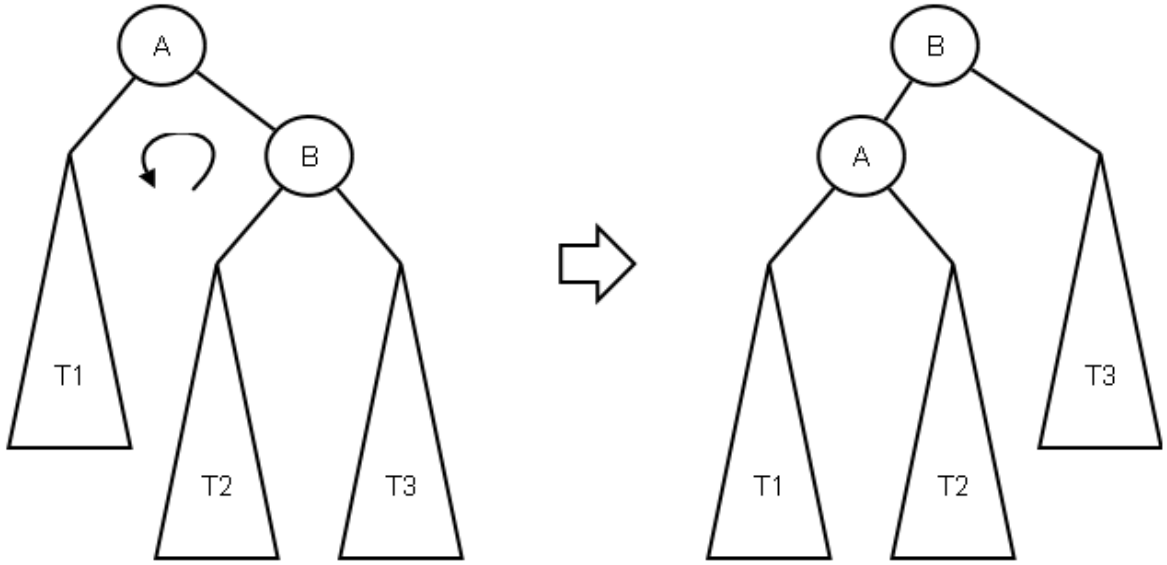
Повороты в БДП

- Меняем местами корень и один из его дочерних узлов
  - Дочерний узел поднимается, корень опускается
- Сохраняем порядок ключей в узлах БДП
- Поворот вправо:



Повороты в БДП

- Меняем местами корень и один из его дочерних узлов
  - Дочерний узел поднимается, корень опускается
- Сохраняем порядок ключей в узлах БДП
- Поворот влево:



Правый поворот в БДП

- Локальное изменение: затрагивает два узла и три связи
- Не является операцией АТД

```
1. void do_rot_right(Node* & node) {
2.     Node* tmp = node->left;
3.     node->left = tmp->right;
4.     tmp->right = node;
5.
6.     // корректируем count
7.     tmp->count = node->count;
8.     node->count = 1+do_count(node->left)+\
9.     do_count(node->right);
10.
11.    // меняем корень
12.    node = tmp;
13.}
```

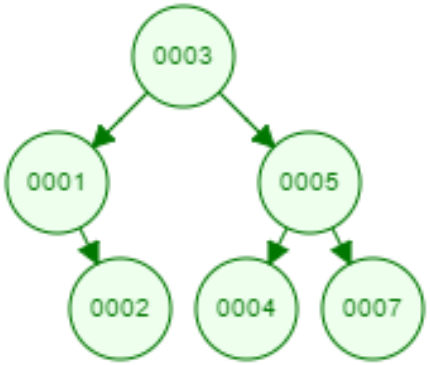
Левый поворот в БДП

- Локальное изменение: затрагивает два узла и три связи
- Не является операцией АТД

```
1. void do_rot_left(Node* & node) {
2.     Node* tmp = node->right;
3.     node->right = tmp->left;
4.     tmp->left = node;
5.
6.     // корректируем count
7.     tmp->count = node->count;
8.     node->count = 1+do_count(node->left)+\
9.     do_count(node->right);
10.
11.    // меняем корень
12.    node = tmp;
13.}
```

Вставка в корень БДП

- Вставляем ключ как обычно и поднимаем его до корня с помощью поворотов
- Рекурсивно:
  - вставляем ключ в корень соот. поддерева
  - делаем его корнем дерева с помощью соот. поворота



https://www.cs.usfca.edu/~galles/visualization/BST.html

Вставка в корень БДП

```
1. void do_insert_root(Node* & node, Item x) {
2.     if (node == NULL) { node = new Node(x); return; }
3.     Key t = node->item.key();
4.     if (x.key() == t) return;
5.     if (x.key() < t) {
6.         do_insert_root(node->left, x);
7.         node->count = 1+do_count(node->left)+\
8.         do_count(node->right);
9.         do_rot_right(node);
10.     }
11.     else { ... }
12.}
```

- Полезно сравнить с `do_insert()`
- Можно аналогично изменить `do_search()`

Разбиение БДП

- Перемещаем *i*-ый по порядку ключ в корень

```
1. void do_partition(Node* & node, size_t i) {
2.     // смотрим число узлов слева
3.     size_t c = 0;
4.     if (node->left != NULL) c = node->left->count;
5.
6.     if (c == i) return; // нашли интересующий элемент
7.     if (c > i) {
8.         do_partition(node->left, i); do_rot_right(node);
9.     } else {
10.        do_partition(node->right, i-c-1); do_rot_left(node);
11.    }
12.}
```

- Полезно сравнить с `do_select()`

Балансировка БДП

ITIVITI

LET IT TRADE

- Время выполнения операций поиска / вставки $O(height)$
- В наихудшем случае (когда?):
  - линейное время поиска
  - квадратичное время построения
- Идеальный случай – полностью сбалансированное дерево
  - тяжело поддерживать для динамической структуры
- Периодически балансировать дерево?

УНИВЕРСИТЕТ ИТМО

Балансировка БДП

- Полная балансировка всего дерева

```
1. void do_balance(Node* & node) {
2.     if ((node == NULL) || (node->count == 1)) return;
3.
4.     do_partition(node, node->count/2);
5.
6.     do_balance(node->left);
7.     do_balance(node->right);
8.}
```

- Частая полная балансировка больших деревьев?
- Давайте производить локальную балансировку дерева при операциях вставки / удаления / (поиска)

Балансировка БДП

ITIVITI

LET IT TRADE

- Цель - избежать наихудшего случая
- Рандомизированный подход
  - хорошее мат. ожидание времени работы
- Амортизационный подход
  - гарантированный верхний предел усредненной стоимости одной операции (общая стоимость всех операций, разделенная на число операций)
- Оптимизационный подход
  - гарантированное время каждой операции
  - необходимо обеспечить, чтобы высота БДП всегда оставалась ~ $\log N$

УНИВЕРСИТЕТ ИТМО

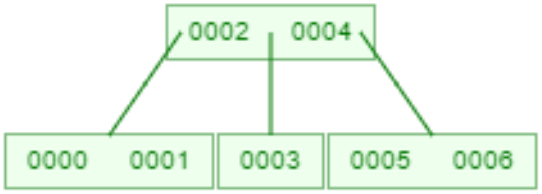
2-3-4 деревья поиска

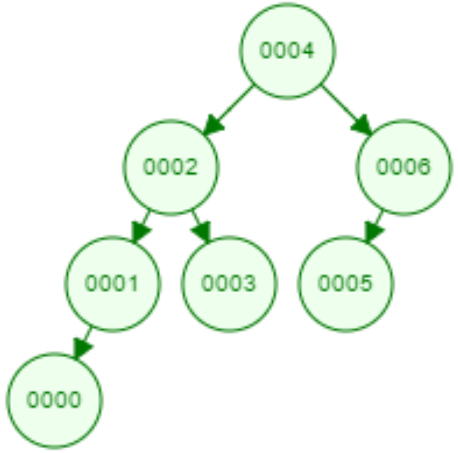
ITIVITI

- Позволим узлам содержать более одного ключа
- 2-3-4-дерево поиска – это либо пустое дерево, либо дерево, содержащее три типа узлов:
- 2-узлы с одним ключом и двумя связями
  - левая – к поддереву с меньшими ключами
  - правая – к поддереву с большими ключами
- 3-узлы с двумя ключами и тремя связями
  - левая – к поддереву с меньшими ключами
  - средняя – к поддереву с ключами между ключами узла
  - правая – к поддереву с большими ключами

2-3-4 деревья поиска

- 4-узлы с тремя ключами и четырьмя связями к поддеревьям, значения ключей которых определены диапазонами, образованными ключами узла
- Сбалансированное 2-3-4-дерево поиска: когда все NULL связи расположены на одинаковом расстоянии от корня

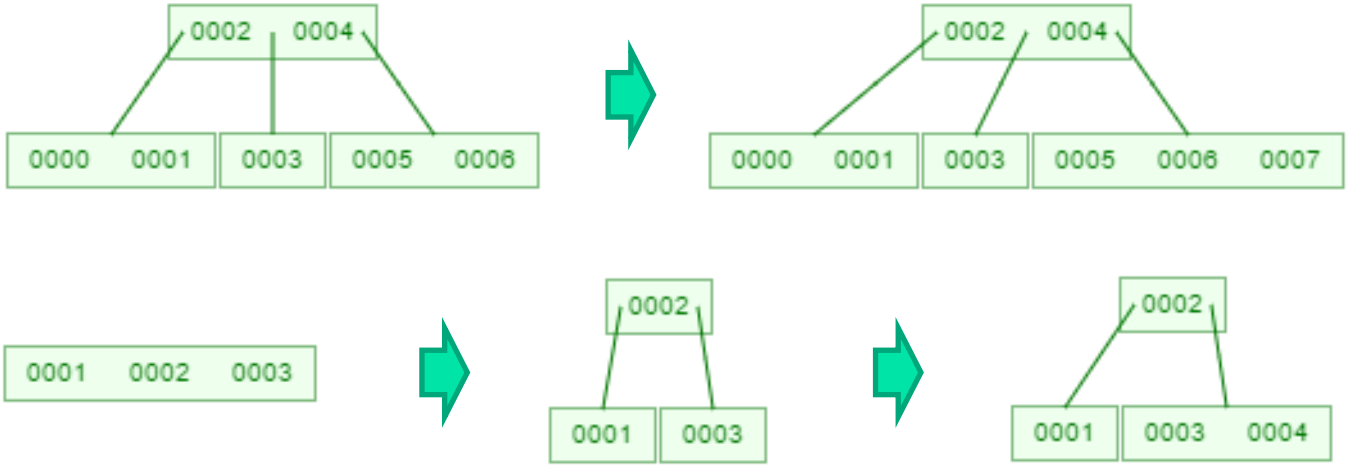




https://www.cs.usfca.edu/~galles/visualization/BTree.html

2-3-4 деревья поиска

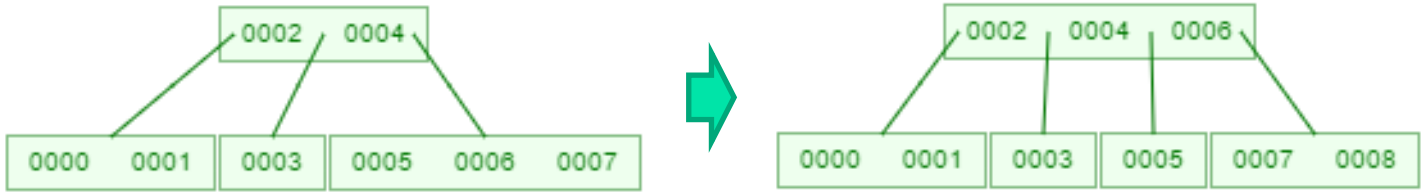
- Поиск?
- Вставка в 2-узел с сохранением баланса? … в 3-узел?
- Вставка в дерево из единственного 4-узла?



https://www.cs.usfca.edu/~galles/visualization/BTree.html

2-3-4 деревья поиска

- Вставка в 4-узел с сохранением баланса?



- Вставка в 4-узел, родителем которого является 4-узел?
  - либо решаем возникшую проблему
  - либо не допускаем такой ситуации

https://www.cs.usfca.edu/~galles/visualization/BTree.html

Вставка в 2-3-4 деревья поиска

ITIVITI

- Расщепляем 4-узлы снизу вверх (bottom-up)
  - Ничего не делаем спускаясь вниз по пути вставки
  - После вставки поднимаемся по дереву, расщепляя 4-узлы, пока не встретим 2- или 3-узел, или пока не дойдем до корня
- Расщепляем 4-узлы сверху вниз (top down)
  - Гарантируем, что путь поиска завершится не в 4-узле
  - При спуске вниз поддерживаем инвариант, что текущий узел не является 4-узлом
  - Если корень стал 4-узлом, то расщепляем его сразу, не дожидаясь следующей вставки
- Только расщепление корня приводит к увеличению высоты дерева

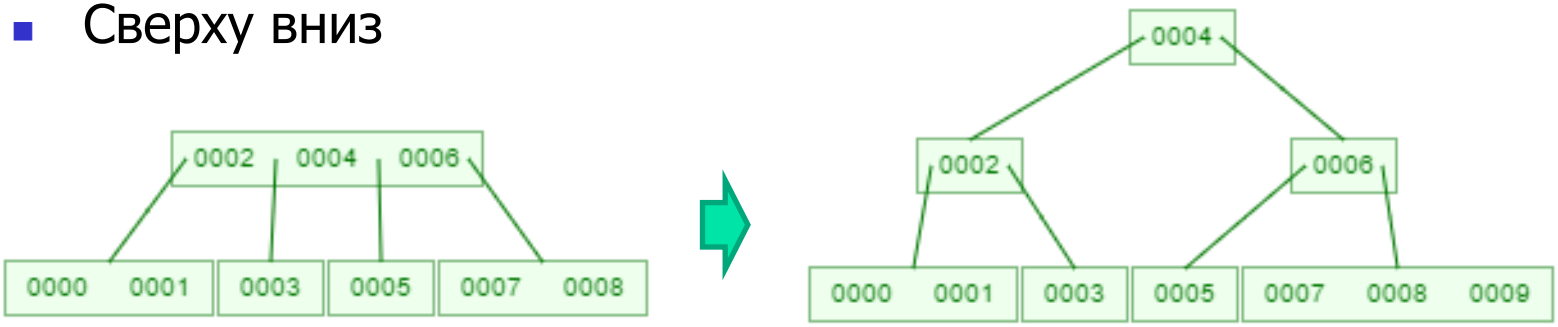
Вставка в 2-3-4 деревья поиска

ITIVITI

LET IT TRADE

- Снизу вверх

- Сверху вниз



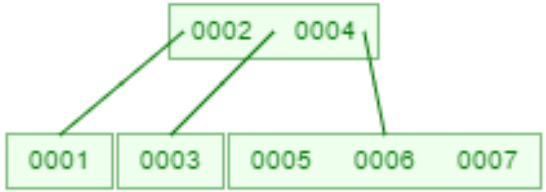
- В отличие от БДП 2-3-4 деревья поиска растут вверх!

https://www.cs.usfca.edu/~galles/visualization/BTree.html

УНИВЕРСИТЕТ ИТМО

Вставка в 2-3-4 деревья поиска

- Расщепление 4-узла
  - локальное преобразование (константное число операций)
  - сохраняет свойства упорядоченности и идеального баланса
- Высота дерева *height* ≤ $\log_2 N$ (наихудший случай – в дереве только 2-узлы)
  - время поиска / вставки $O(\log N)$
- Семь последовательно вставленных ключей:



https://www.cs.usfca.edu/~galles/visualization/BTree.html

2-3-4 деревья поиска

ITIVITI

LET IT TRADE

- Прямая реализация сложна и неэффективна:
  - поддержка нескольких типов узлов
  - преобразование узлов из одного типа в другой
  - копирование ключей и из одного узла в другой
  - сравнение искомого ключа с каждым ключом в узле
  - и т.д.
- Есть решение лучше – красно-черные деревья!

УНИВЕРСИТЕТ ИТМО

Красно-черные деревья

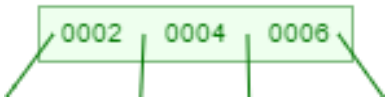
ITIVITI

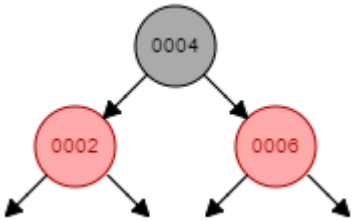
- Два типа связей:
  - вертикальные (черные) – отделяют разные узлы друг от друга
  - горизонтальные (красные) – соединяют элементы, хранящиеся в одном узле 2-3-4 дерева
- На каждый узел указывает одна связь, поэтому покраска связей эквивалентна покраске узлов
  - каждый узел хранит цвет связи, указывающей **на** этот узел
  - только один из элементов узла 2-3-4 дерева красится в черный цвет, остальные – в красный

Красно-черные деревья

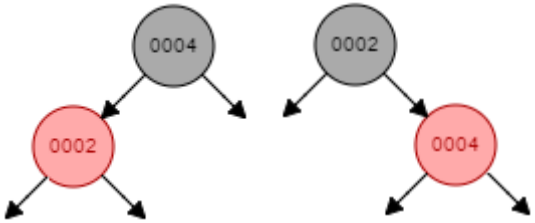
- Представление 2-узла

- Представление 4-узла





- Представления 3-узла

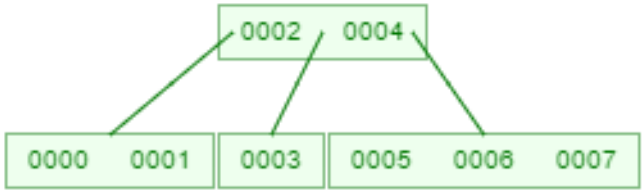


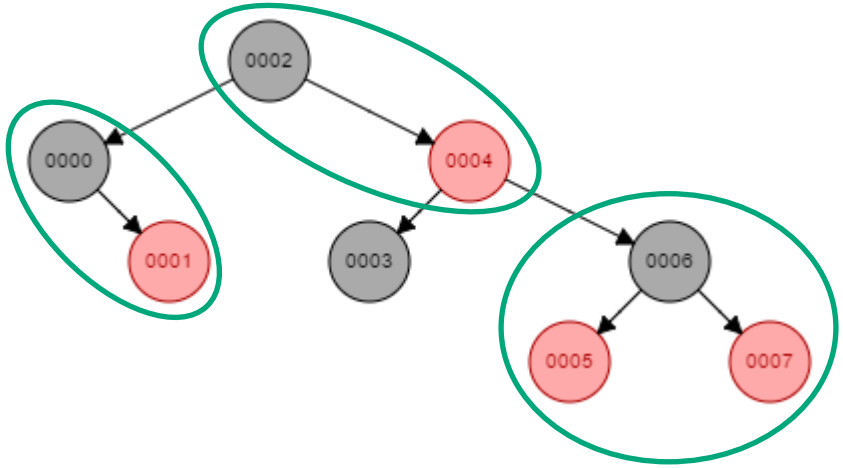
- Горизонтальные связи ведут из черного узла в красный
- Вертикальные – из любого узла в черный

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Красно-черные деревья

- Соответствие между 2-3-4 и красно-черными деревьями



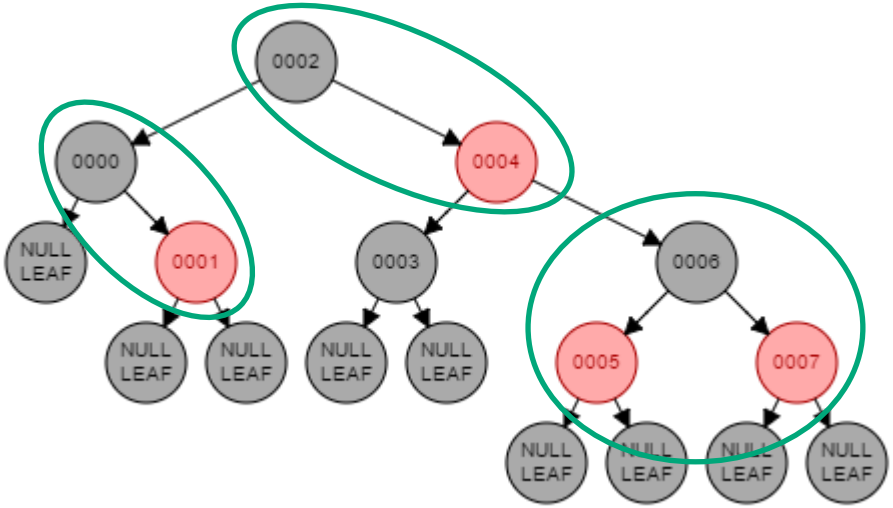


- Простой метод поиска, характерный для обычных БДП
- Простой метод вставки-балансировки, характерный для 2-3-4 деревьев

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Красно-черные деревья

- Альтернативное определение:



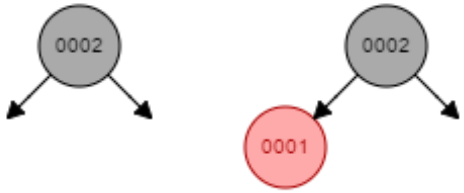
- Каждый узел красный или чёрный
- Корень и листья дерева – чёрные
- Если узел красный, то оба его дочерних узла – чёрные
- Все простые пути из любого узла до листьев содержат одинаковое количество чёрных узлов

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

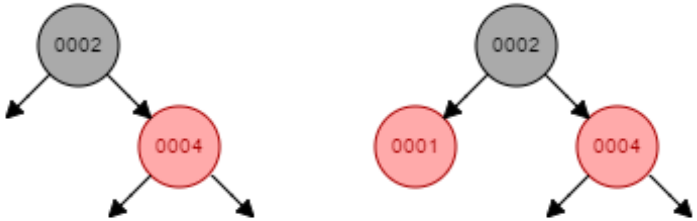
Вставка в красно-черные деревья

ITIVITI

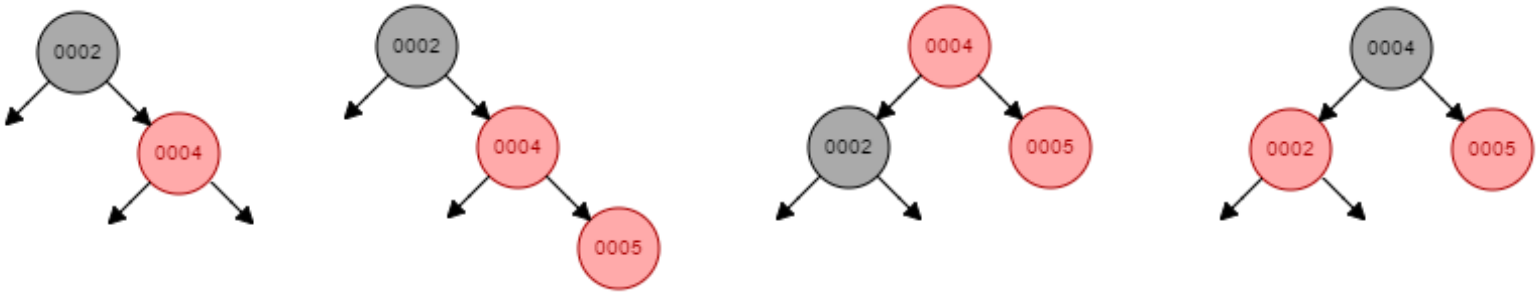
- Вставка в 2-узел



- Вставка в 3-узел (вариант №1)



- Вставка в 3-узел (вариант №2): поворачиваем и перекрашиваем 3-узел => вариант №1

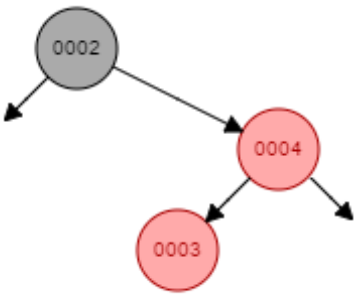


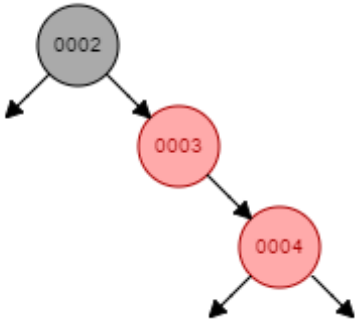
LET IT TRADE

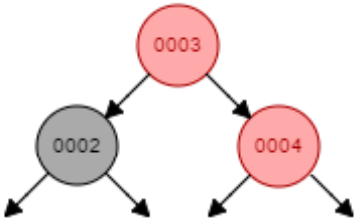
Вставка в красно-черные деревья

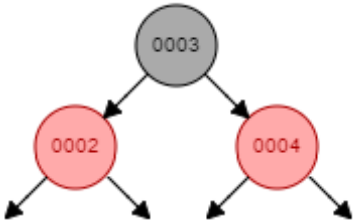
ITIVITI

- Вставка в 3-узел (вариант №3):
  меняем местами родительский
  и вставляемый узел
  (поворот в родителе) => вариант №2



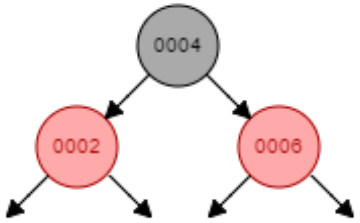


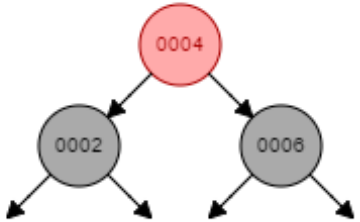




- Расщепление 4-узла: перекраска узлов
  (передаем средний элемент наверх)

  если корень покраснел,
  то красим в чёрный –
  увеличение высоты на единицу





LET IT TRADE

Вставка в красно-черные деревья

ITIVITI

LET IT TRADE

- Расщепляем 4-узлы сверху вниз (top down)
  - Гарантируем, что путь поиска завершится не в 4-узле
  - При спуске вниз поддерживаем инвариант, что текущий узел не является 4-узлом
  - Связей 4-узел – 4-узел быть не может
- Реализация в два этапа (не совсем top down):
  - При спуске вниз расщепляем 4-узлы и передаем средние элементы их родителям
  - При подъеме наверх завершаем вставку средних элементов в 3-узлы

Вставка в красно-черные деревья


```
1. private:
2. // направление по которому пришли в данный node
3. enum direction { dir_left, dir_right };
4.
5. bool is_red(Node* node) {
6.     if (node == NULL) return false; // листья всегда чёрные
7.     return node->red;
8. }
9.   ...
10. public:
11. void insert_rb(Item x) {
12.     do_insert_rb(m_root, x, dir_left);
13.     m_root->red = false; // корень всегда чёрный
14.}
```

Вставка в красно-черные деревья


```
1. void do_insert_rb(Node* & node, Item x, direction dir) {
2.     // добавляемый узел - красный
3.     if (node == NULL) { node = new Node(x); return; }
4.     // если 4-узел, то расщепляем
5.     if (is_red(node->left) && is_red(node->right)) {
6.         node->red = true;
7.         node->left->red = false;
8.         node->right->red = false;
9.     }
10.    Key t = node->item.key();
11.    if (x.key() == t) return; // ключ уже существует
12.    if (x.key() < t) {
13.        do_insert_rb(node->left, x, dir_left);
14.        node->count = 1+do_count(node->left)+\
15.        do_count(node->right);
```

УНИВЕРСИТЕТ ИТМО

Вставка в красно-черные деревья

ITIVITI

LET IT TRADE

```
16.         // вставка в 3-узел (вариант №3)
17.         if (is_red(node) && is_red(node->left) && \
18.         (dir == dir_right)) // в node пришли справа
19.             do_rot_right(node);
20.
21.         // вставка в 3-узел (вариант №2)
22.         if (is_red(node->left) && \
23.         is_red(node->left->left)) {
24.             do_rot_right(node);
25.             node->red = false;
26.             node->right->red = true;
27.         }
28.     }
29.     else { ... }
30.}
```


УНИВЕРСИТЕТ ИТМО

Удаление элемента

ITIVITI

- Удаляем всегда из листа
  - Если удаляемый элемент не в листе, то меняем его местами со следующим / предыдущим
  - Следующий / предыдущий элемент находим как min / max из соот. поддерева
- Нельзя просто взять и удалить элемент из листового 2-узла
  - Нарушится идеальный баланс
- Гарантируем, что путь поиска завершится не в 2-узле
  - При спуске вниз поддерживаем инвариант, что текущий узел не является 2-узлом

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

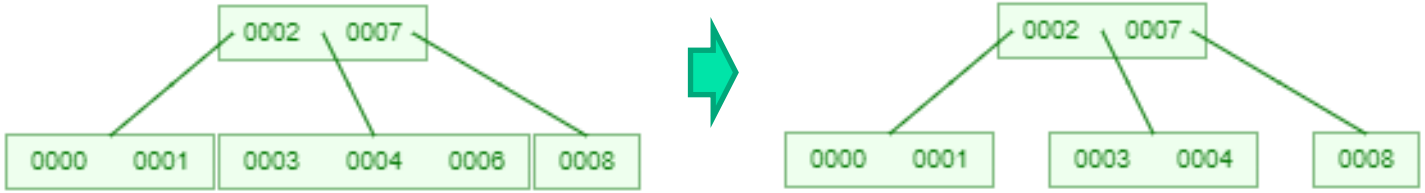
Удаление элемента

- Вариант №1 (Transfer): если сосед нашего 2-узла – 3- или 4-узел, то используем элемент соседа
  - Для правого соседа: наибольший элемент родителя и наименьший элемент соседа
  - Для левого соседа: наоборот

https://www.cs.usfca.edu/~galles/visualization/BTree.html

Удаление элемента

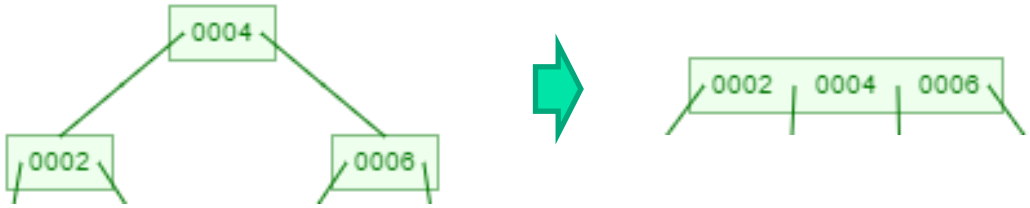
- Вариант №2 (Fusion): если оба соседа нашего 2-узла – 2-узлы, и родитель 3- или 4-узел
  - Берем элемент от родителя и создаем 4-узел



https://www.cs.usfca.edu/~galles/visualization/BTree.html

Удаление элемента

- Вариант №3 (Root removal): если оба соседа нашего 2-узла – 2-узлы, и родитель 2-узел (т.е. корень)
  - Преобразуем корень и его дочерние узлы в 4-узел



https://www.cs.usfca.edu/~galles/visualization/BTree.html

ITIVITI

LET IT TRADE

Спасибо за внимание!

УНИВЕРСИТЕТ ИТМО