Практикум на ЭВМ
АВЛ-дерево

Баев А.Ж.

Казахстанский филиал МГУ

27 ноября 2018

Шаблон-функция

```
template <class myType>
myType GetMax (const myType & a, const myType & b) {
    if (b < a) {
        return a;
    }
    return b;
}
```

Использование

```
class Vector {
public:
    bool operator<(const Vector & another);
};
...
int result1 = GetMax<int>(5, 7);
double result2 = GetMax<double>(5.0, 7.0);
double result3 = GetMax<Vector>(Vector(5.0, 7.0),
                                Vector(1.0, 2.0));
```

Шаблон-класс

```
template <class ValueType>
struct Node {
    ValueType key;
    Node *left;
    Node *right;
    Node *parent;
    int depth;
};
```

Использование

Статическое

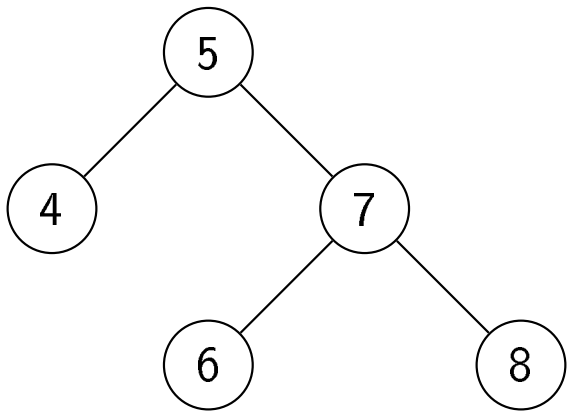
```
Node<ValueType> root;
root.key = 5;
```

Динамическое

```
root = new Node<ValueType>;
root->key = 5;
delete root;
```

Дерево поиска в STL

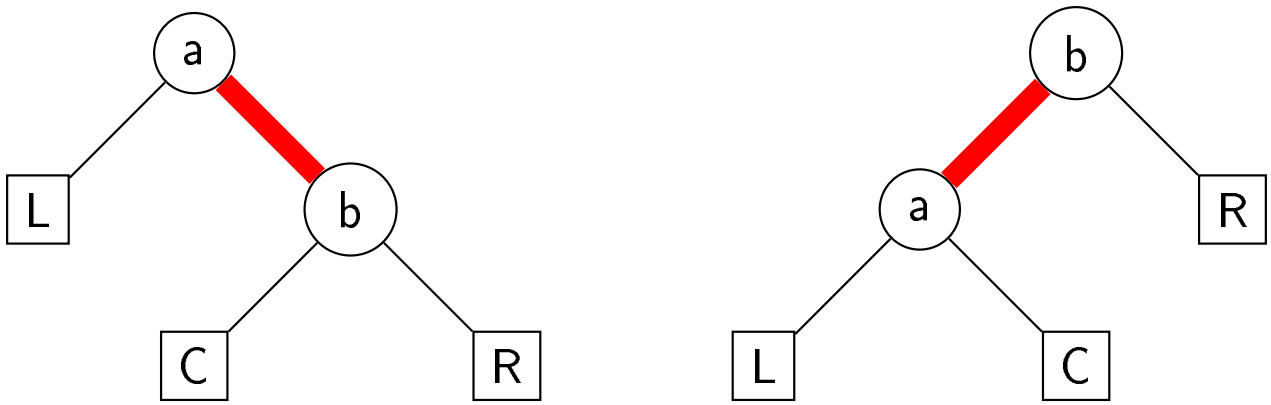
```
#include <set>

std::set<int> s;
s.insert(5);
s.insert(7);
s.insert(4);
s.insert(6);
s.insert(8);
for (const auto & elem : s)
    std::cout << elem << ' ';
```



Добавление в дерево

Добавляем элемент и начинаем рекурсивный подъем. Если в некоторый момент оказалось, что в узле a дизбаланс, то производим балансировку. Допустим $depth(L) + 1 < depth(b)$.

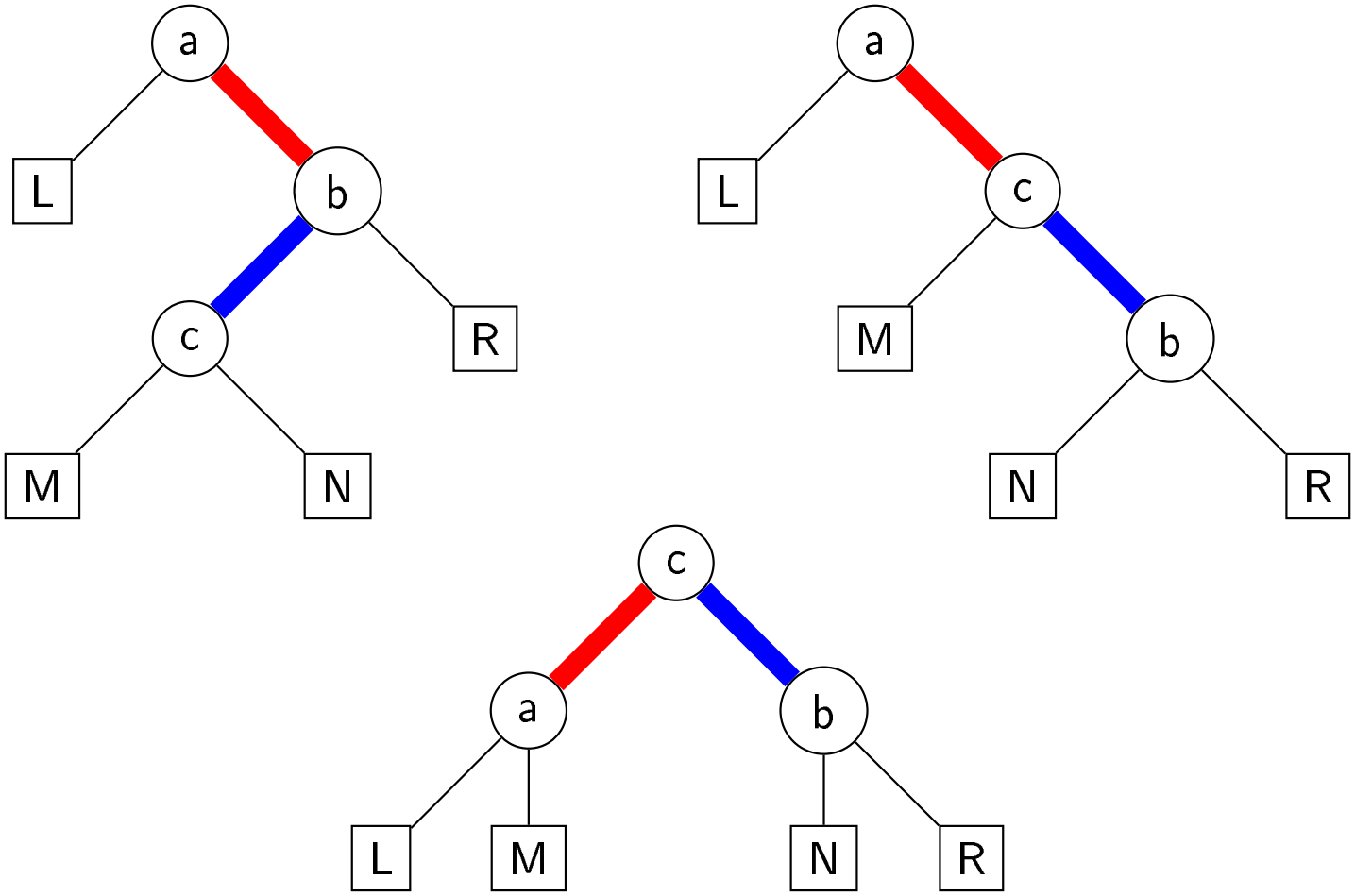
Первый случай $depth(C) < depth(R)$. Делаем малый поворот ребра ab влево.



Необходимо поменять ссылки на детей: $a.right$, $b.left$, $a.parent.child$.
Необходимо поменять ссылки на родителей: $a.parent$, $b.parent$, $c.parent$.

Добавление в дерево

Первый случай $depth(C) \geqslant depth(R)$. Делаем большой поворот тройки ab влево.



Необходимо сделать малый поворот вправо у вершины $a.right$ и малый поворот влево от a.

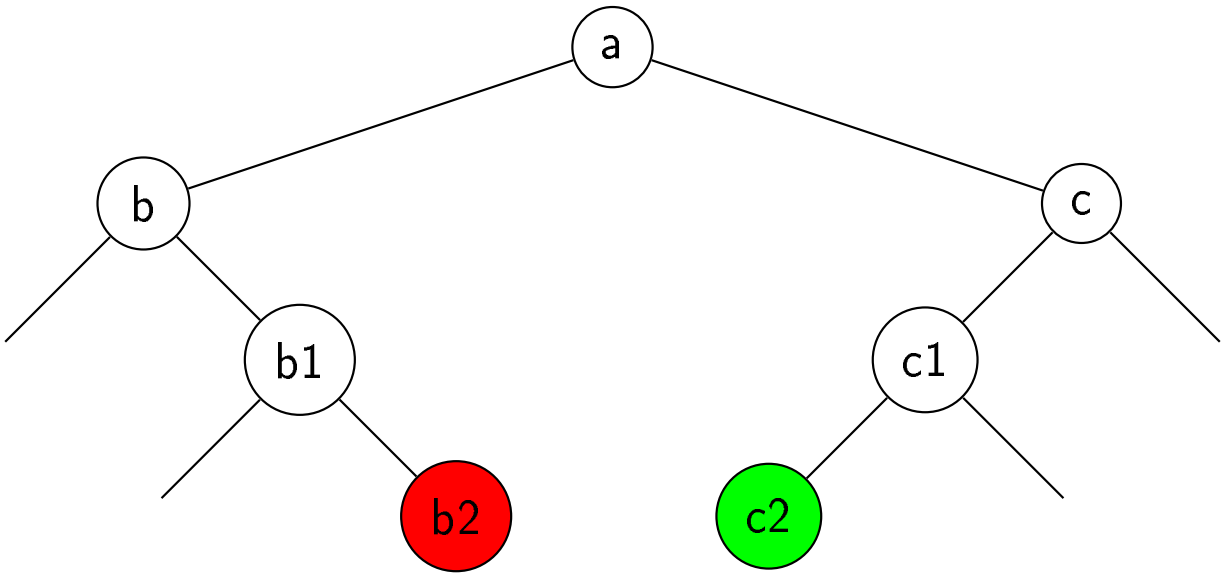
Удаление из дерева

Если элемент с данным ключом k — лист, удаляем и поднимаемся по рекурсии, восстанавливая балансировку при необходимости. Если данный элемент a не лист, то меняем его с ближайшим листом из

$$a- > left- > right- > right...- > right$$

или

$$a- > right- > left- > left...- > left$$

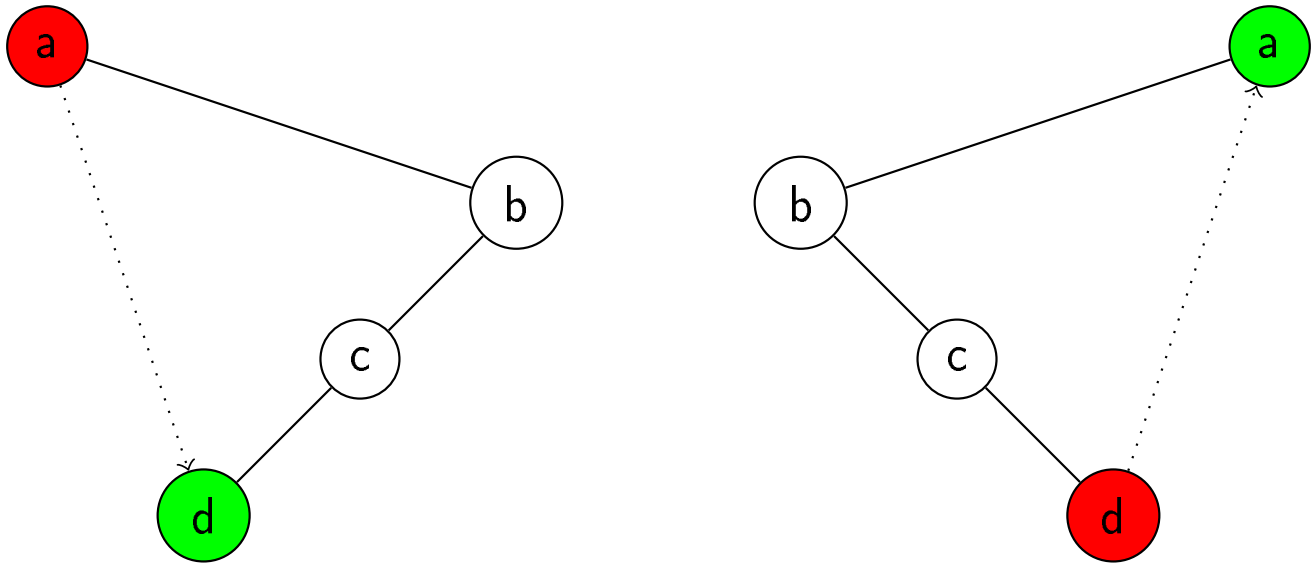


После обмена, например a и $b2$ достаточно вызвать процедуру удаления k из поддерева b. При подъеме из a и выше проводить процедуру балансировки.

Итерирование

Для данного узла необходимо переходить к узлу со следующим значением.

- если есть правый сын — переходим к начале минимуму правого сына;
- если правого сына нет — переходим к родителю, родителю родителя и так далее, пока не найдем родителя с ключом больше данного или не попадем в нулевой указатель (у корневого элемента рекомендуется хранить нулевой указатель в родительском поле).



Что нужно реализовать — работа с указателями

```
template <class ValueType>
class Set {
    size_t size_;
    Node<ValueType> *root;
    Node<ValueType> *begin(Node<ValueType> *ptr);
    Node<ValueType> *rbegin(Node<ValueType> *ptr);
    Node<ValueType> *next(Node<ValueType> *ptr);
    Node<ValueType> *prev(Node<ValueType> *ptr);
```

Что нужно реализовать — повороты и балансировка

```
void leftBig(Node<ValueType> **vertex);
void rightBig(Node<ValueType> **vertex);
void leftSmall(Node<ValueType> **vertex);
void rightSmall(Node<ValueType> **vertex);

int getDepth(Node<ValueType> *ptr);
void updateDepth(Node<ValueType> *ptr);

int getBalance(Node<ValueType> *ptr);
void makeBalancePlus(Node<ValueType> **ptr);
void makeBalanceMinus(Node<ValueType> **ptr);
```

Что нужно реализовать — интерфейсные функции и рекурсивные помощники

```
    void insert(const ValueType & key,
                Node<ValueType> **ptr,
                Node<ValueType> *parent = NULL);
    bool erase(const ValueType & key,
               Node<ValueType> **ptr);

public:
    Set();
    Set(const Set<ValueType> &source);
    size_t size() const;
    bool empty() const;
    void insert(const ValueType & key);
    void erase(const ValueType & key);
    Node<ValueType> *begin();
    Node<ValueType> *rbegin();
    void print();
    void clear(Node<ValueType> *ptr);
    ~Set();
};
```

Как сделать отладочный вывод?

```
std::vector<Node<ValueType>*> one, two;
int hmax = 6, width = 1 << hmax;
one.push_back(root);
for (int level = 0; level < hmax; ++level) {
    std::string filler(width - 1, ' ');
    for (auto elem : one) {
        if (elem) {
            std::cout << filler << elem->key << filler;
            two.push_back(elem->left);
            two.push_back(elem->right);
        } else {
            std::cout << filler << "--" << filler;
            two.push_back(nullptr);
            two.push_back(nullptr);
        }
    }
    std::cout << std::endl;
    one = two;
    two.clear();
    width /= 2;
}
```

Как сделать отладочный вывод?

1. Отладочный вывод.
2. Добавление элементов без балансировки.
3. Повороты и балансировка про вставке.
4. Удаление листьев без балансировки.
5. Итерирование по элементам дерева.
6. Удаление листьев с балансировкой.