Санкт-Петербургский Государственный Университет
Математико-механический факультет

Материалы доклада по теме:

"STL(C++, set, multiset, map, multimap)"

Выполнил: студент группы 21.Б12-мм Зернов С. Н.

Санкт-Петербург
2022

1 Особенности array и vector

Выпишем временные оценки работы методов STD::VECTOR:

Контейнер	push_back	insert	pop_front	pop_back	erase	find	count
std::vector	O(1)	O(n)	O(1)	O(1)	O(n)	O(n)	O(n)
?	n/a	O(log n)	n/a	n/a	O(log n)	O(log n)	O(log n)

Легко заметить, что важнейшие методы работают за линейное время. Задачей данной работы является описание контейнера, позволяющего значительно ускорить операции вставки, удаления и поиска элементов.
Во второй строке таблицы отражена временная сложность методов искомого контейнера.

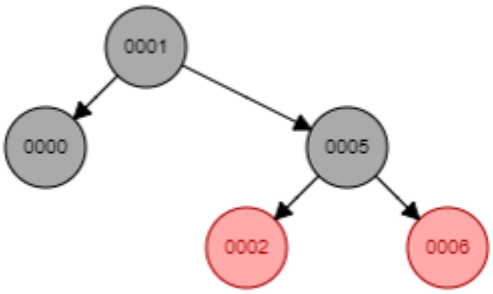
Поиск за O(n) удручает

2 Красно-чёрные деревья

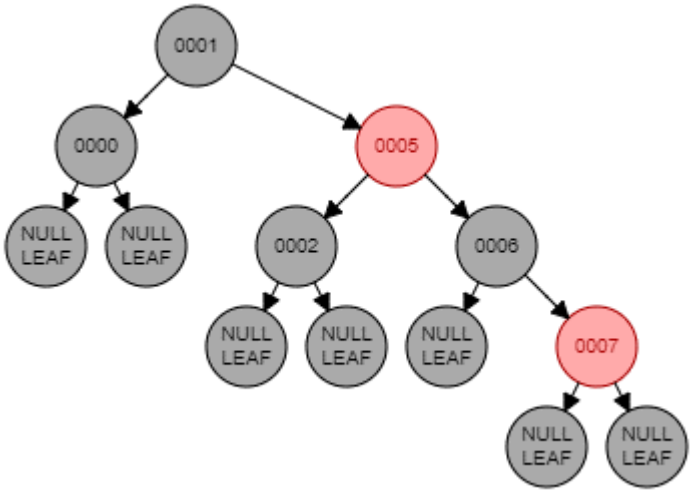
Разберёмся с внутренней структурой нашего контейнера. Из-за требуемой логарифмической сложности методов, естественным решением задачи является построение бинарного дерева.
Разработчики STL выбрали особый подвид бинарного дерева поиска — красно-чёрное дерево. Данный параграф будет посвящён его свойствам.

Свойства:

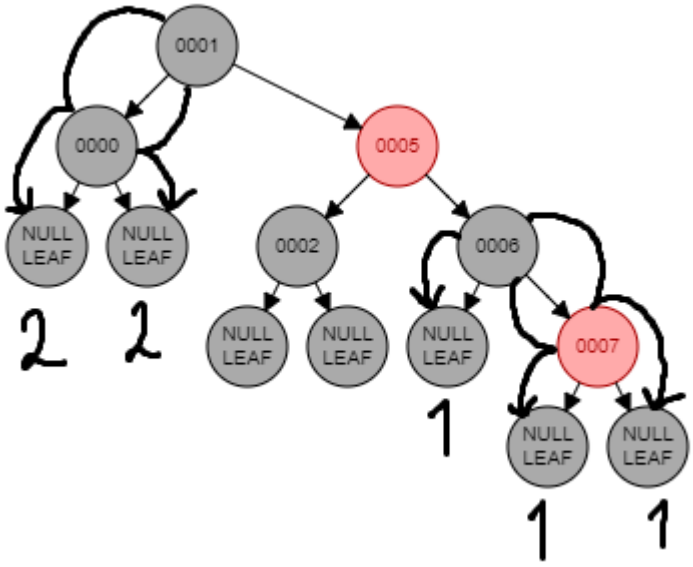
- Каждый узел окрашен в красный или чёрный цвет
- Корень — чёрный



- Листья — чёрные NULL-узлы
- У каждого красного узла два чёрных потомка



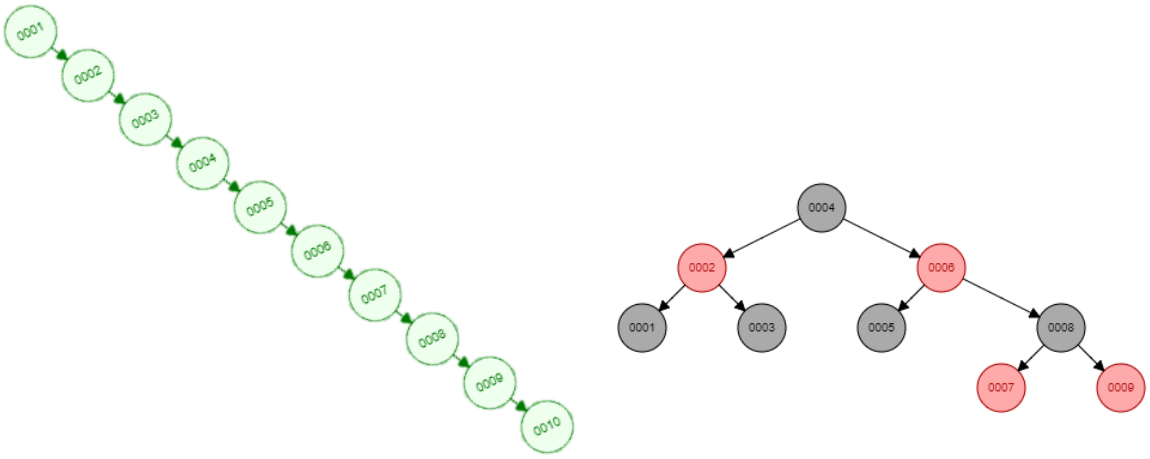
- Пути от узла к его листьям должны содержать одинаковое количество черных узлов



Что это даёт?

Расположим числа от 1 до 10 в бинарном и красно-чёрном дереве. Представим, что нам надо найти элемент 10: в первом случае на это уйдёт O(n) операций (дерево превратится в связный список), во втором случае мы уложимся в O(log n).

Почему разработчики STD выбрали именно красно-чёрное дерево?

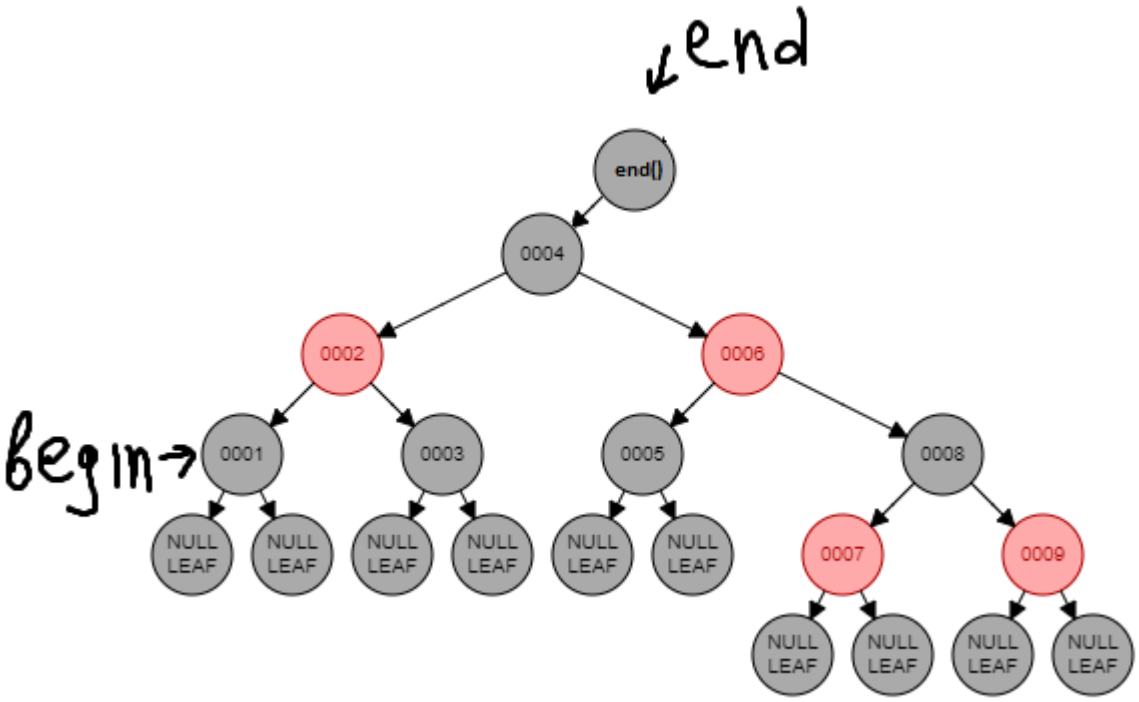


(a) Бинарное дерево

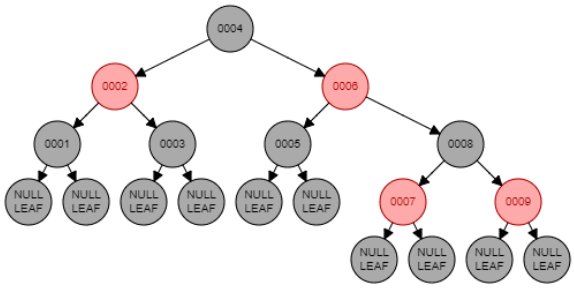
(b) Красно-чёрное дерево

Ответ: Операции insert, erase и find выполняются за «железный логарифм»: без вероятностей и усреднений.

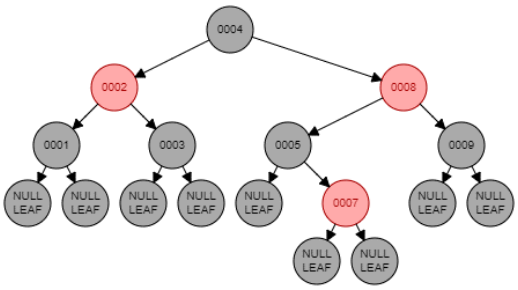
Введём дополнительные обозначения: будем называть begin() самый левый узел, а end() — специальный новый узел. Если дерево пустое, то они совпадают.



Условимся, что у нас есть механизмы вставки (insert) и удаления вершины (erase) из дерева: зачастую они требуют перестройки, а потому непросты в реализации (много случаев)



(a) До удаления вершины с ключём 8



(b) После

О вставке и удалении можно прочитать отдельно.

3 Схематическая реализация std::map

Поля и основные методы

```
template <typename Key, typename Value, typename Compare = std::less<Key>>

class MyMap {
    using value_type = std::pair<const Key, Value>;
private:
    BaseNode* leftmost;
    BaseNode* root;
    Compare comp;

    struct BaseNode {
        Node* left;
        Node* right;
        Node* parent;
        bool red;
    };

    struct Node : BaseNode {
        value_type kv;
    };

public:

    struct iterator {
        BaseNode* node;
        //increment complexity            O(log n)
        //                    O(n)      m.begin()      m.end(),    .   .
        value_type& operator *() const {
            return (static_cast<Node*>(node))->kv;
        }
    };

    Value& operator[](const Key& key);
    Value& at(const Key& key);
    const Value& at(const Key& key) const;
    iterator find(const Key& key);
    size_t count(const Key& key) const;
};
```

Опишем полученный класс:

- В шаблон передаются типы ключа, значения и компаратор (принцип сравнения ключей). По умолчанию, Compare = std::less<const Key>
- Существует четвёртое поле шаблона — Allocator (для кастомного аллокатора). По умолчанию, Allocator = std::allocator<std::pair<const Key, T>
- В приватных полях класса хранится внутренняя структура красно-чёрного дерева (цвет вершины и указатели на детей и родителя). Так же указатель на корень и самую левую вершину
- в типе Node хранятся пары const Key, Value
- Итератор возвращает it=std::pair<const Key, Value>

4 Методы std::map

Перечислим основные методы:

- operator[] — обращается по ключу к вершине и возвращает ссылку на значение. Если нет такого ключа, то создаёт вершину с ним и присваивает значение по умолчанию
- at — похож на operator[], однако не создаёт вершину при отсутсвии ключа
- insert — возвращает std::pair<iterator, bool>. Итератор — куда вставилось, bool — произошла ли вставка (если значение с таким ключом есть, то вставка бы не произошла)
- erase — по итератору или ключу удаляет вершину дерева
- find — по ключу возвращает итератор на вершину
- count — считает количество элементов с данным ключом. Для map мало смысла, для multimap актуальна.

5 std::set

std::set — красно-чёрное дерево, в котором в вершинах хранятся уникальные ключи.

- В каком-то смысле std::map без value
- Аналог множества в математике

6 Методы std::set

Методы аналогичны std::map, однако:

- Возвращают уже не std::pair<Key, Value>, а просто Key
- Нет оператора []

7 std::multimap и std::multiset

От стандартных std::map и std::set отличаются лишь тем, что ключи могут повторяться. Поэтому:

- Методы доступа к элементам бессмысленны (at, [])
- find возвращает по ключу любой из элементов дерева
- По существу применимы лишь методы upper_bound, lower_bound, equal_range

8 Методы std::multimap и std::multiset

- upper_bound — по ключу возвращает первый итератор с не меньшим значением ключа (если его нет, то end())
- lower_bound — по ключу возвращает итератор с не большим значением ключа (аналогично, может вернуть end())
- equal_range — по ключу возвращает промежуток [lower_bound; upper_bound)