Тема 5. Быстрые деревья поиска

Красно-черное дерево

Еще один вариант построения быстрого дерева.

Определение: Красно-черное дерево — это бинарное дерево поиска, удовлетворяющее следующим условиям:

1. Каждая вершина имеет свой "цвет" — красный или черный (корень дерева всегда черный);

2. Красная вершина может иметь только черных потомков;

3. В любой ветке от корня до концевой вершины содержится одинаковое число черных вершин (это число называется черной длиной дерева);

Третье условие собственно и дает основания надеяться на необходимую логарифмическую сложность поиска, поскольку красно-черное дерево является "идеально сбалансированным по черной длине". Более строго этот факт устанавливает следующее

Утверждение. Глубина красно-черного бинарного дерева с N вершинами не превосходит $2\log_2(N+1)$.

Доказательство. Пусть красно-черное дерево имеет глубину k. Заметим, что в силу условия 2 длина любой ветви от корня не меньше, чем $k/2$. Таким образом, для количества вершин дерева N имеем

$$N \geq 2^{k/2} - 1$$

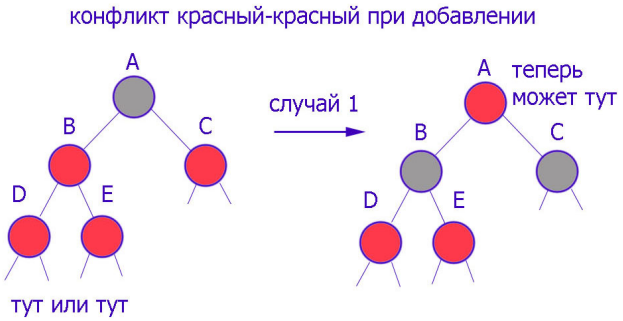
Красно-черные деревья удобно реализовывать со ссылкой на родительский элемент, т.е.

```
template <class T>
struct TreeNode
{
   T value;
   TreeNode *left, *right, *parent;
   int color;    // 0 - red or 1 - black
   TreeNode() { left = right = parent = nullptr; }
}
```

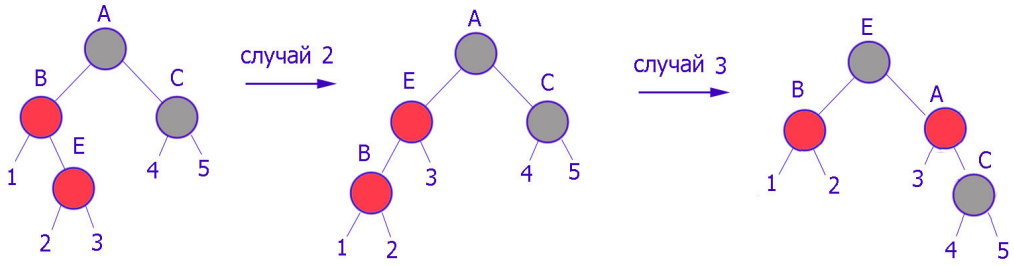
Добавление в красно-черное дерево.

Добавляем как обычно и красим добавленный элемент красным. Черная длина не нарушилась, вожможен конфликт красный-красный (п.2)

Иллюстрация: возможные случаи устранения конфликтов при добавлении.



конфликт красный-красный при добавлении



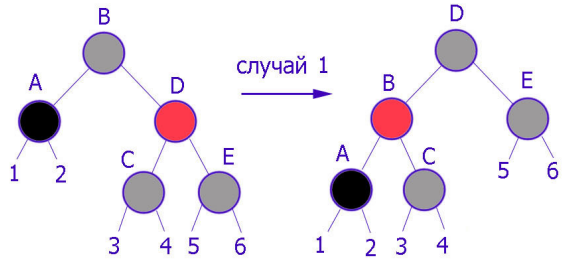
Сложность добавления $O(\log_2 N)$.

Удаление.

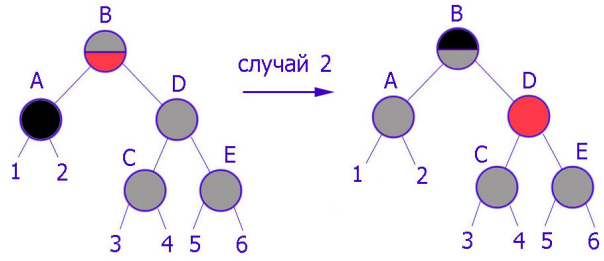
Удаляем как обычно. Если физически удалился красный элемент, то все. Если удалился черный, то черный цвет оставляем в дереве, перенося его на потомка. Для унификации с случаем удаления листа вводим фиктивные листы черного цвета, которые смогут принять на себя лишний черный цвет. Таким образом, в дереве появились "дважды черные" вершины, и теперь этот двойной черный цвет надо "размазать" по дереву.

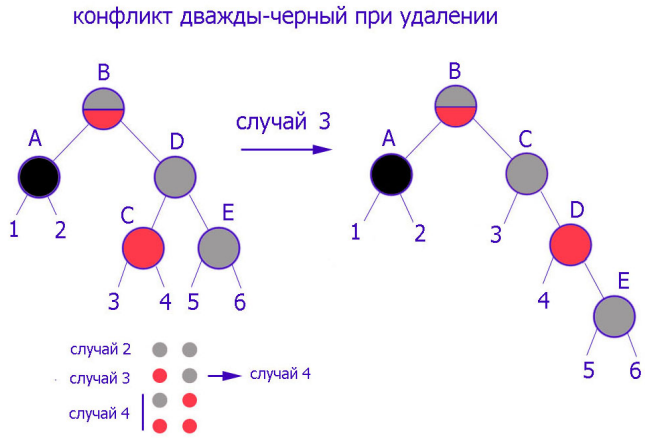
Иллюстрация: возможные случаи устранения дважды черного цвета при удалении.

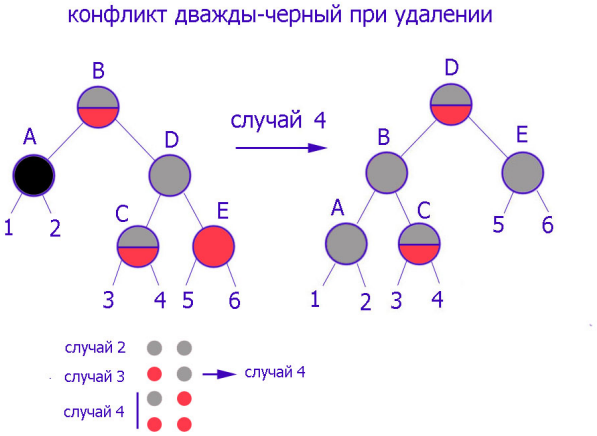
конфликт дважды-черный при удалении



конфликт дважды-черный при удалении







Несмотря на кажущуюся сложность описанных схем, алгоритм удаления элемента выписывается достаточно просто. Вот его набросок.

```
1. Пусть A - текущая дважды черная вершина.
2. Если A - корень, то убрать лишний цвет и закончить.
3. Если A лежит слева от своего родителя, то
   если цвет D красный, то
       выполнить преобразование 1 и перейти к п.1.
   иначе // цвет D черный
      если цвет E черный, то
         если цвет C черный, то
            выполнить преобразование 2
            если B дважды черная, то
               сделать B текущей и перейти к п.1.
            иначе закончить.
         иначе // цвет C красный
            выполнить преобразование 3
         конец если
      конец если
      выполнить преобразование 4
      закончить
   конец если
4. Если A лежит справа от своего родителя, то
   симметричный фрагмент кода
```

Красно-черные деревья используются, например, в стандартной реализации STL set и map. По эффективности эти деревья чуть лучше (не используют рекурсию), чем AVL деревья, но, конечно, по порядку они обеспечивают одну и ту же трудоемкость.