Красно-черные и splay деревья

Java Developer Level 2

Progwards - Академия компьютерного мастерства

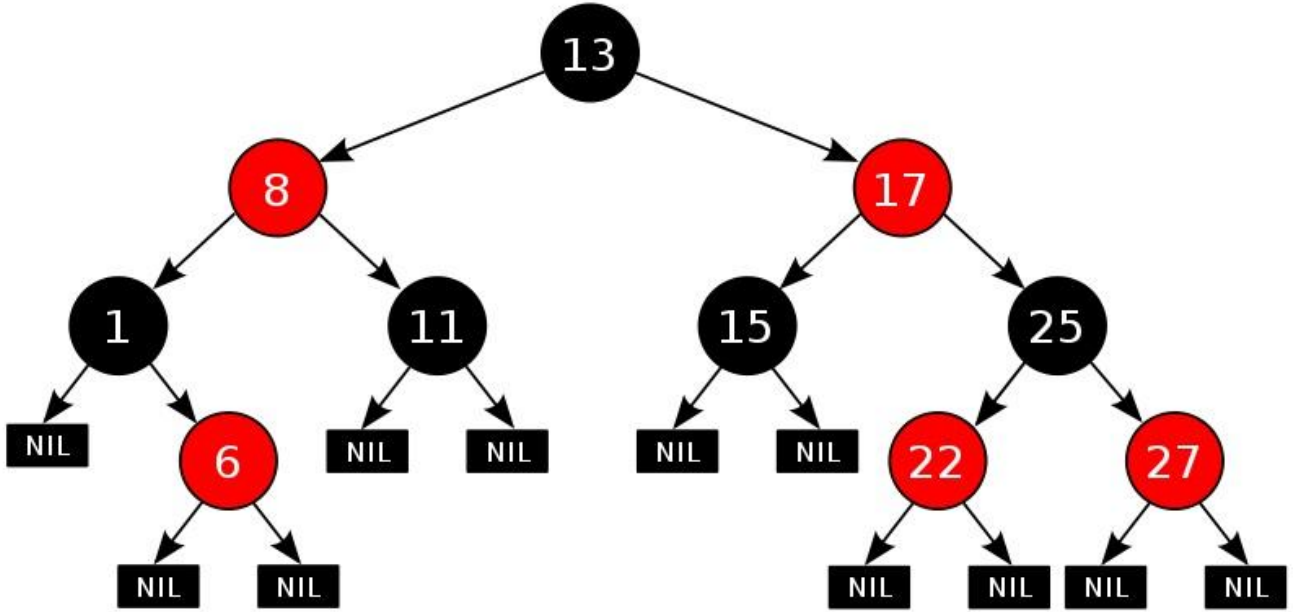
Содержание лекции

- Красно-черные деревья
- Splay-деревья
- Сравнения деревьев
- Рандомизированные деревья

Красно-черные деревья

PROGWARDS
Академия компьютерного мастерства

Это вид двоичного дерева поиска

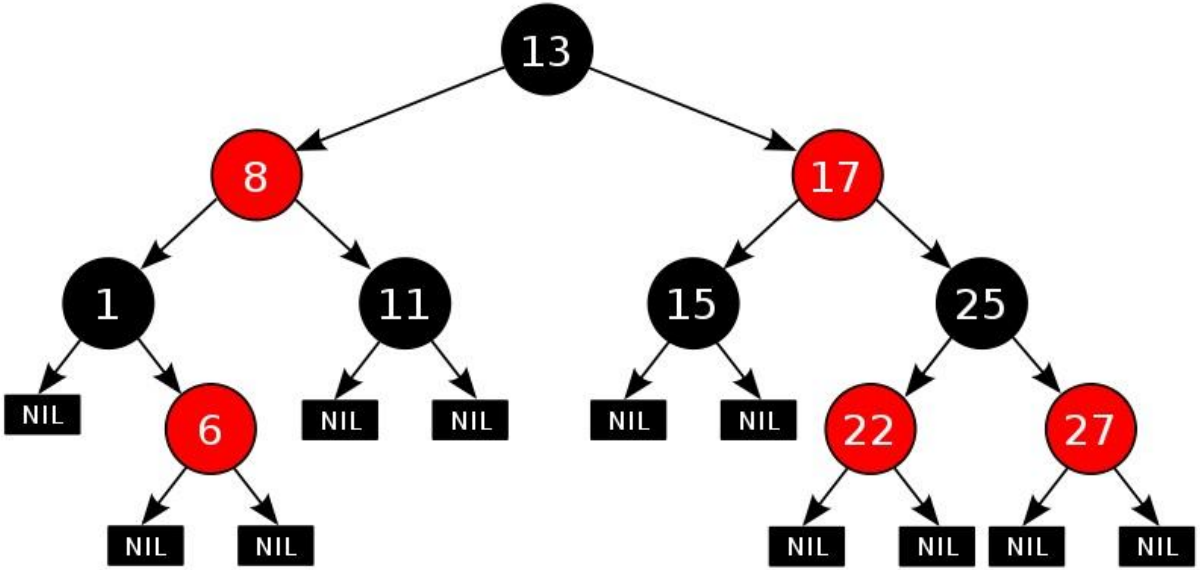


Свойства красно-черного дерева

- Узел либо красный, либо черный
- Корень всегда черный
- Все листья **null** - черные
- Оба потомка каждого красного узла — черные
- Всякий простой путь от данного узла до любого листового узла, являющегося его потомком, содержит одинаковое число черных узлов

Красно-черные деревья

- Примерно сбалансированное дерево
- Рудольф Байер, 1978 г.



Вставка

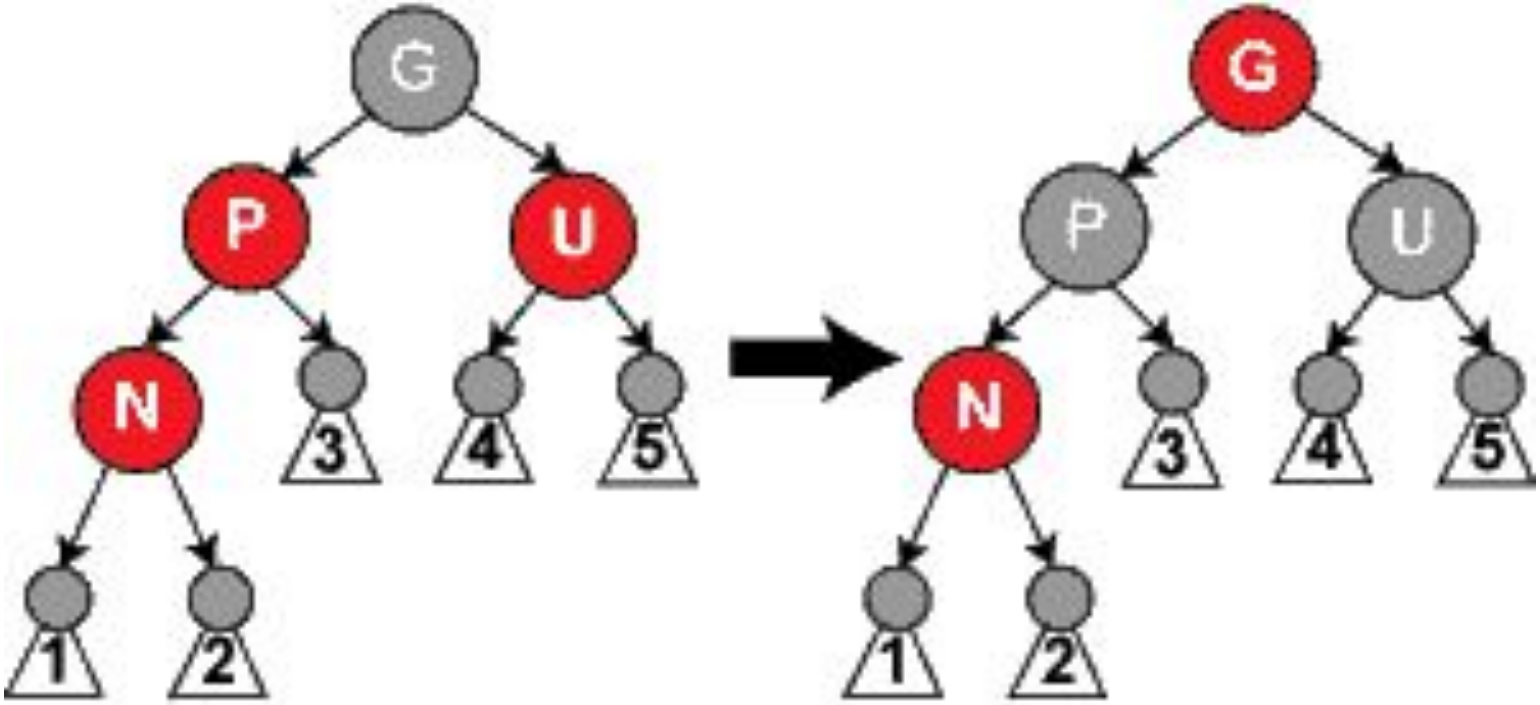
- Как и в обычном дереве поиска
- Вставляемый узел всегда красный
- Проверяем, нарушились ли свойства дерева, и если да - то проводим балансировку

Темнинология

- N - новый узел
- P - родитель
- G - дед
- U - дядя

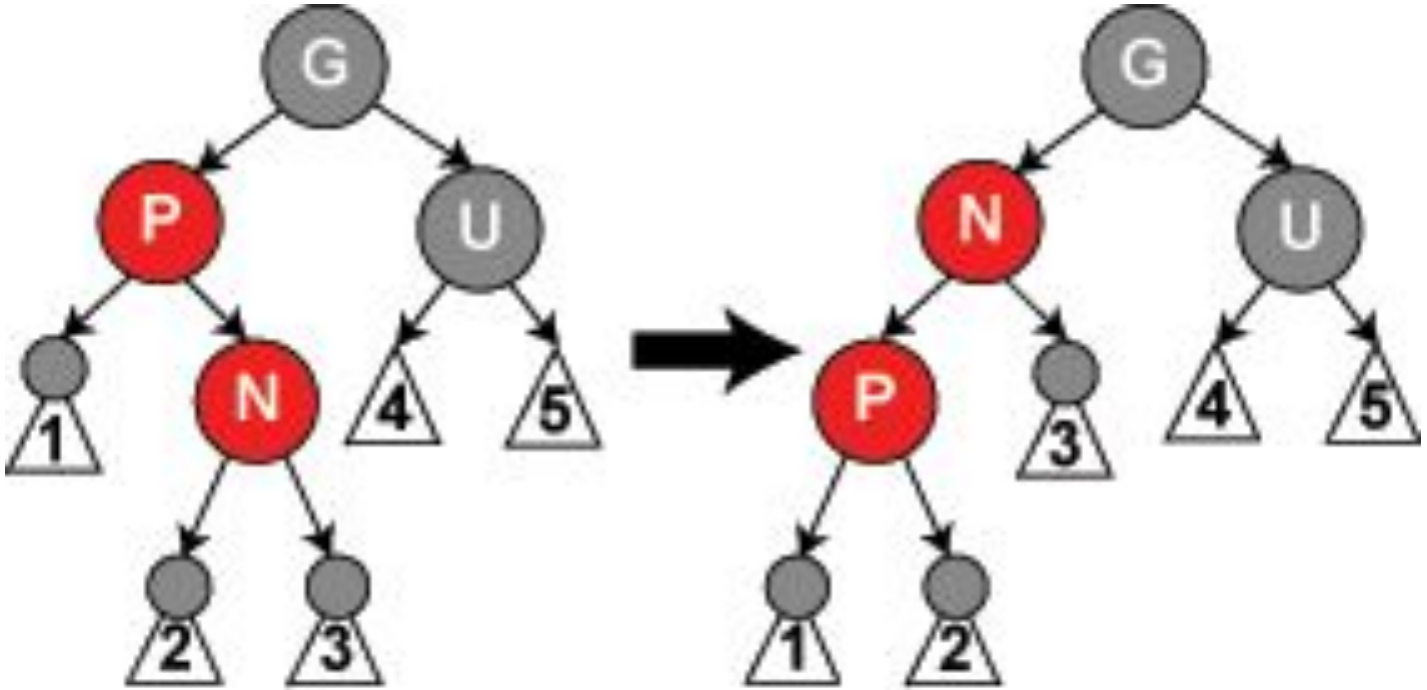
Проверка/балансировка

- Если отец и дядя красные



Родитель - красный, дядя — чёрный.

- Если мы левый, а родитель правый или наоборот

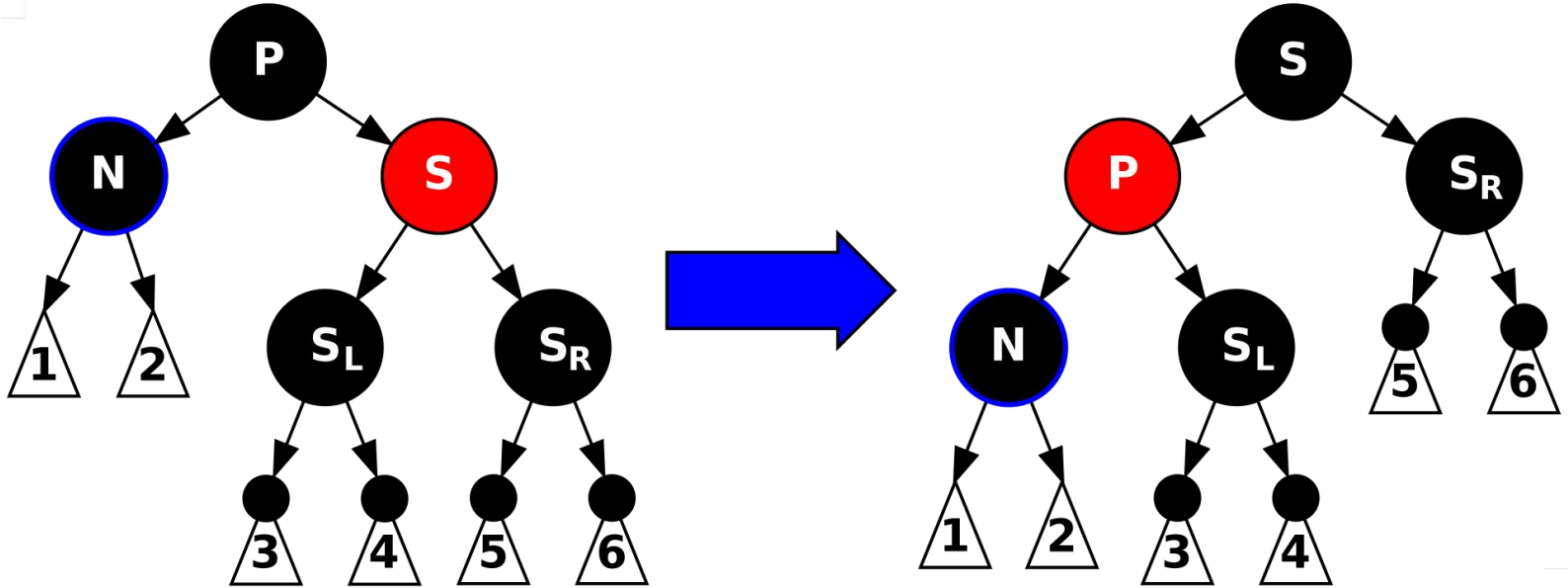


Темнинология

- M - удаляемый узел
- C - потомок, которого нашли на замену
- N - новое положение этого потомка
- S - его брат
- Sl - левый потомок S
- Sr - правый потомок S
- P - родитель
- G - дед
- U - дядя

СбалансироватьПослеУдаления

- S - красный



Сравнение АВЛ и красно-черных деревьев

PROGWARDS
Академия компьютерного мастерства

N - количество вершин, h - высота дерева

Для АВЛ дерева

$$N(h) = \Theta(\lambda^h), \text{ где } \lambda = (\sqrt{5}+1)/2 \approx 1,62$$

Для красно-черного

$$N(h) \geq 2^{(h-1)/2} = \Theta(\sqrt{2}^h)$$

Итого

$$\log \lambda / \log \sqrt{2} \approx 1,388$$

Тест производительности

test set	representation	BST	AVL	RB
normal	plain	5.22	4.62	4.49
	parents	4.97	4.47	4.33
	threads	5.00	4.63	4.51
	right threads	5.12	4.66	4.57
	linked list	5.05	4.79	4.59
sorted	plain	*	4.21	4.90
	parents	*	4.04	4.70
	threads	*	4.25	5.02
	right threads	*	4.24	5.02
	linked list	*	4.31	4.98
shuffled	plain	5.98	5.80	5.69
	parents	5.80	5.68	5.51
	threads	5.80	5.77	5.65
	right threads	5.89	5.80	5.71
	linked list	5.88	6.06	5.84

Table 6: Times, in seconds, for 5 runs of the unsorted and sorted versions of the cross-reference collator for each kind of tree. *Pathological case not measured.

test set	representation	BST	AVL	RB
normal	plain	3.24	3.09	3.01
	parents	3.10	2.94	2.86
	threads	3.12	3.04	2.98
	right threads	3.21	3.06	3.02
	linked list	3.19	3.21	3.08
shuffled	plain	3.23	3.28	3.24
	parents	3.13	3.12	3.05
	threads	3.15	3.20	3.15
	right threads	3.22	3.22	3.19
	linked list	3.24	3.43	3.32

Table 7: Times, in seconds, for 50 runs of a reduced version of the cross-reference collator for each kind of tree designed to fit within the processor cache.

P6 performance counters [24] directly confirms that reducing the test set size reduces additional cache misses in the shuffled case from 248% to 33%.

Расширяющиеся деревья

- Это вид двоичного дерева поиска
- Splay - скошенный
- Не хранит дополнительной информации для поддержания структуры
- Роберт Тарьян и Даниель Слейтор в 1983 году

Расширяющиеся деревья

- Подъем текущей вершины в корень при любой операции с деревом

- Вставка
- Удаление
- Поиск

Splay-деревья

- Зачем?

Теорема о текущем множестве

Пусть qi - число раз, которое запрошен элемент i

Тогда m запросов поиска выполняется

$$O\left(m + \sum_i q_i \log \frac{m}{q_i}\right)$$

Splay-дерево будет амортизационно работать не хуже, чем самое оптимальное фиксированное дерево

Теорема о статической оптимальности

Пусть tj - число запросов, которое совершили к элементу j с момента x. Тогда m запросов поиска выполняется

$$O\left(m + n\log n + \sum_j \log(t_j + 1)\right)$$

В среднем недавно запрошенный элемент не уплывает далеко от корня

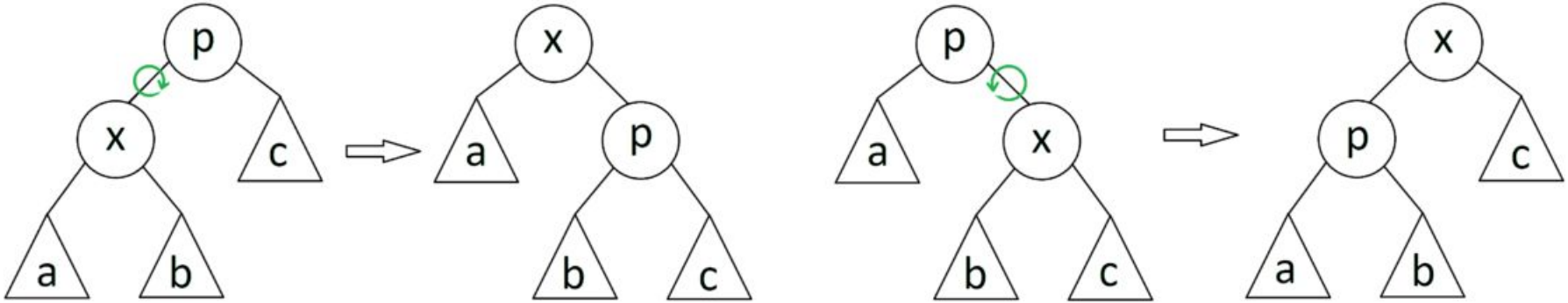
Splay

```
Перекосить():
  пока родитель не пусто то
     если родитель.левый == текущий то
        ВращатьНаправо()
     иначе
        ВращатьНалево()
```

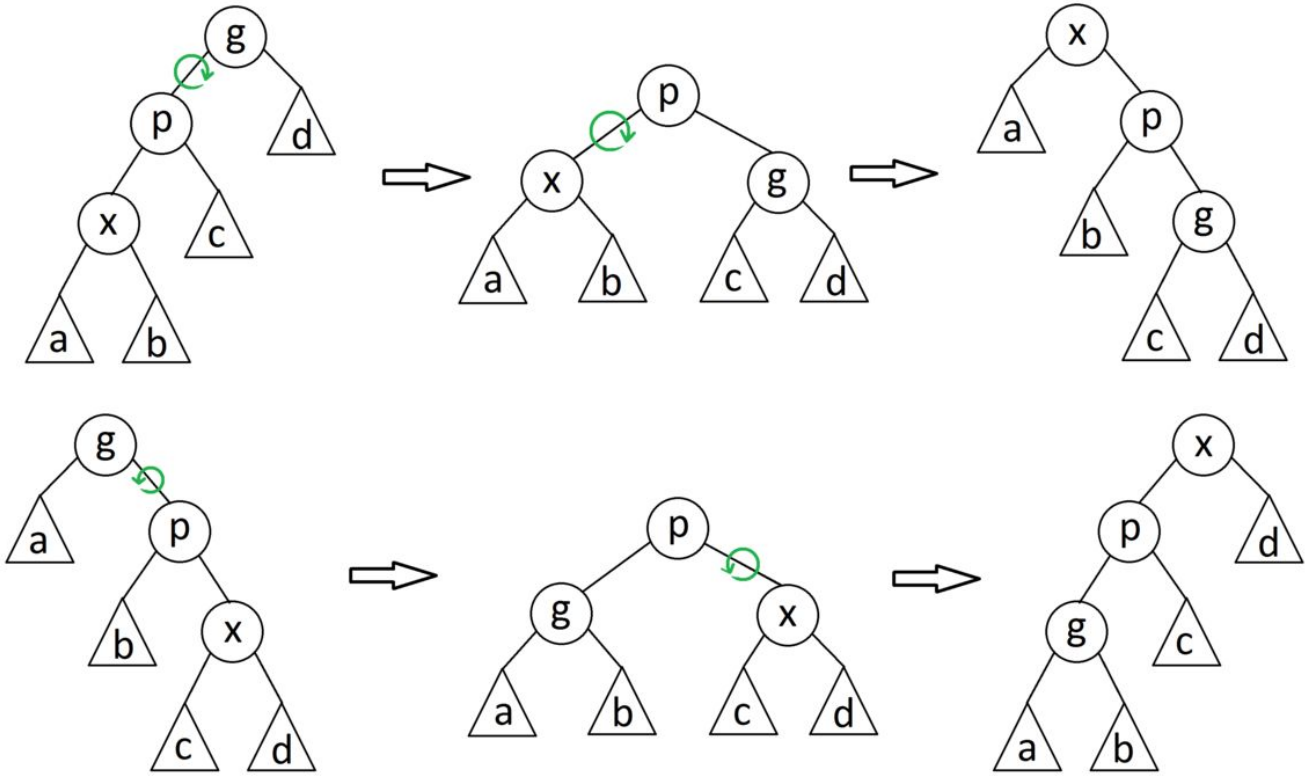
Splay - деревья

- Zig поворот
- Zig-Zig поворот
- Zig-Zag поворот

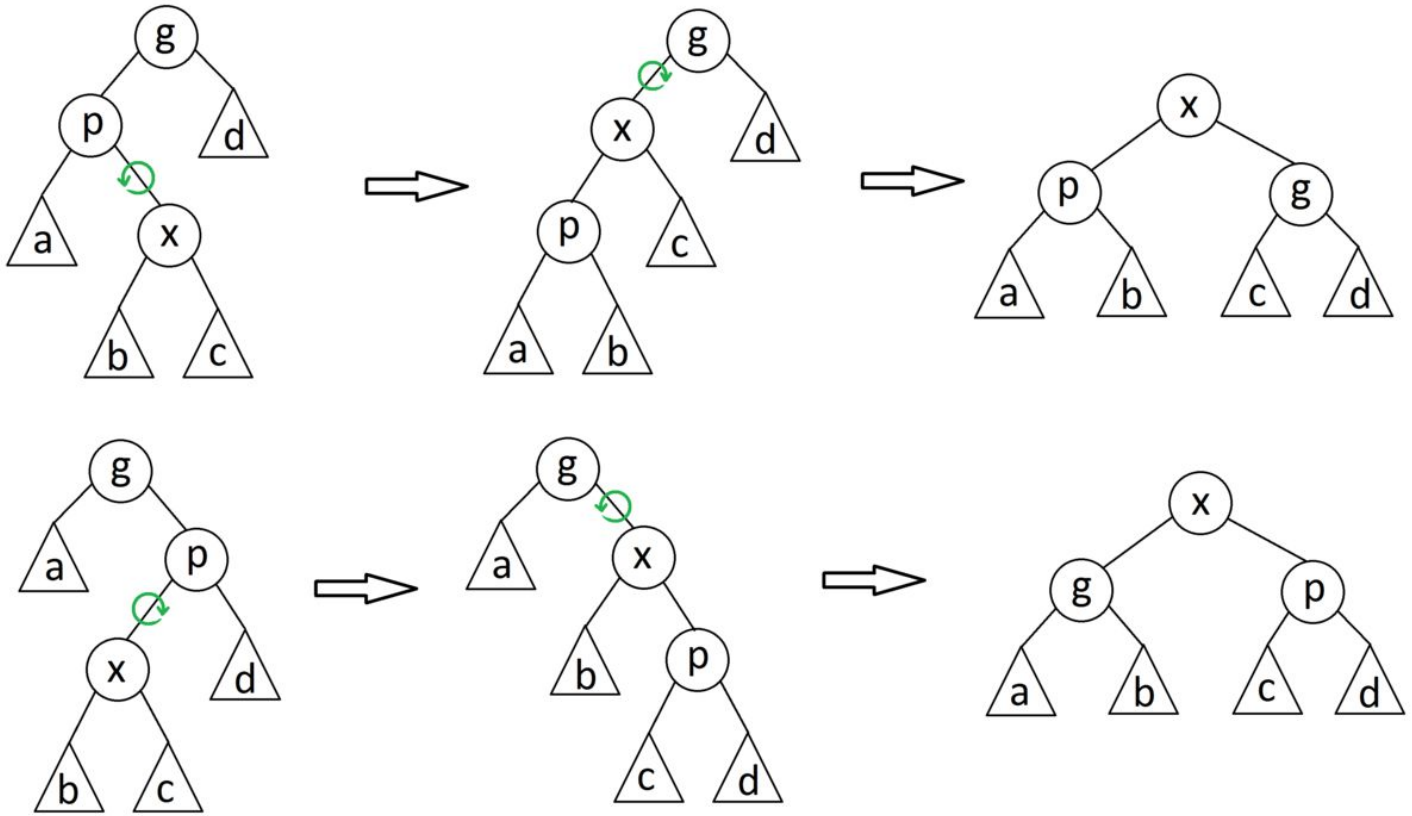
Zig-поворот



Zig-Zig поворот



Zig-Zag поворот



Splay

```
Перекосить():
  если родитель пусто то
    выход
  дед = родитель.родитель
  если дед пусто то
    Zig()
    выход
  иначе
    если (дед.левый==родитель) == (родитель.левый==текущий)  и
         (дед.правый==родитель) == (родитель.правый==текущий)  то
      ZigZig()
    иначе
      ZigZag()

    Перекосить()
```

Поиск

- Как обычном дереве поиска
- В конце вызывается функция Splay

Удаление

- Сначала вызывается функция Splay
- Удаляем корень
- Слияние поддеревьев

Тест производительности

test set	representation	BST	AVL	RB	splay
normal	plain	5.22	4.62	4.49	4.03
	parents	4.97	4.47	4.33	4.00
	threads	5.00	4.63	4.51	4.03
	right threads	5.12	4.66	4.57	4.06
	linked list	5.05	4.79	4.59	4.03
sorted	plain	*	4.21	4.90	2.91
	parents	*	4.04	4.70	2.90
	threads	*	4.25	5.02	2.89
	right threads	*	4.24	5.02	2.87
	linked list	*	4.31	4.98	2.83
shuffled	plain	5.98	5.80	5.69	6.54
	parents	5.80	5.68	5.51	6.61
	threads	5.80	5.77	5.65	6.56
	right threads	5.89	5.80	5.71	6.63
	linked list	5.88	6.06	5.84	6.64

Table 6: Times, in seconds, for 5 runs of the unsorted and sorted versions of the cross-reference collator for each kind of tree. *Pathological case not measured.

test set	representation	BST	AVL	RB	splay
normal	plain	3.24	3.09	3.01	2.80
	parents	3.10	2.94	2.86	2.81
	threads	3.12	3.04	2.98	2.82
	right threads	3.21	3.06	3.02	2.85
	linked list	3.19	3.21	3.08	2.87
shuffled	plain	3.23	3.28	3.24	3.51
	parents	3.13	3.12	3.05	3.55
	threads	3.15	3.20	3.15	3.55
	right threads	3.22	3.22	3.19	3.60
	linked list	3.24	3.43	3.32	3.64

Table 7: Times, in seconds, for 50 runs of a reduced version of the cross-reference collator for each kind of tree designed to fit within the processor cache.

P6 performance counters [24] directly confirms that reducing the test set size reduces additional cache misses in the shuffled case from 248% to 33%.

Рандомизированные деревья

- Это вид двоичного дерева поиска
- Хранит уровень, как АВЛ

Insert

```
Вставить(ключ)
  если random() % (узел.высота+1) == 0 то
       ВставитьВКорень(узел);
  если текущий.ключ > ключ то
       если левый пусто то
            текущий.левый = новый Лист(ключ);
       иначе
            текущий.левый = левый.Вставка(ключ);
  иначе
       если правый пусто то
            текущий.правый = новый Лист(ключ);
       иначе
            текущий.правый = правый.Вставка(ключ);

  ПересчитатьВысоту();
```

Insert

```
Вставить(ключ)
  если random() % (узел.высота+1) == 0 то
       ВставитьВКорень(узел);
  если текущий.ключ > ключ то
       если левый пусто то
            текущий.левый = новый Лист(ключ);
       иначе
            текущий.левый = левый.Вставка(ключ);
  иначе
       если правый пусто то
            текущий.правый = новый Лист(ключ);
       иначе
            текущий.правый = правый.Вставка(ключ);

  ПересчитатьВысоту();
```

Вставить в корень - а-ля splay

PROGWARDS
Академия компьютерного мастерства

```
ВставитьВКорень(ключ):
 узел = обычнаяВставка(ключ)
  нашРодитель = родитель
 пока узел.родитель != нашРодитель то
    если узел.родитель.левый == узел то
        ВращатьНаправо()
    иначе
        ВращатьНалево()
```

Рандомизированные деревья

- Вставка в рандомизированное дерево эквивалентно вставке в BST рандомизированных ключей
- Необходимость хранить уровнь
- Затраты времени на ГПСЧ на каждом шаге поиска

Заключение

- Красно-черные дерево - приблизительно сбалансированное дерево
- TreeMap в Java - это красно-черное дерево
- Splay-дерево - вершина всплывает в корень при каждой операции
- По тестам, RBT выигрывает у AVL немного, но по всем тестам, и у Splay только в одном тесте
- Рандомизированное дерево эквивалентно обычному BST со случайной последовательностью ключей