Алгоритмы и структуры данных

Деревья поиска

CS Center, Новосибирск

Структуры данных map/set

Абстрактная структура данных “ассоциативный массив”

Структуры данных map/set

Абстрактная структура данных “ассоциативный массив”

Методы map:

- insert(KEY, VAL)
- erase(KEY)
- VAL? get(KEY) const

Структуры данных map/set

Абстрактная структура данных “ассоциативный массив”

Методы map:

- insert(KEY, VAL)
- erase(KEY)
- VAL? get(KEY) const

Методы set:

- insert(KEY)
- erase(KEY)
- bool check(KEY) const

Структуры данных map/set

Абстрактная структура данных “ассоциативный массив”

Есть линейный порядок на ключах

BST

BST = binary search tree

BST

BST = binary search tree

Определение:

- Ключ в вершине v = x
- Ключи в левом поддереве v «≤ x»
- Ключи в правом поддереве v «≥ x»

Поиск в дереве

- Начинаем с корня

Поиск в дереве

- Начинаем с корня
- Ключ в текущей вершине «>» => идём в левое поддерево

Поиск в дереве

- Начинаем с корня
- Ключ в текущей вершине «>» => идём в левое поддерево
- Ключ в текущей вершине «<» => идём в правое поддерево

Поиск в дереве

- Начинаем с корня
- Ключ в текущей вершине «>» => идём в левое поддерево
- Ключ в текущей вершине «<» => идём в правое поддерево
- Совпадение => поиск успешно завершен

Поиск в дереве

- Начинаем с корня
- Ключ в текущей вершине «>» => идём в левое поддерево
- Ключ в текущей вершине «<» => идём в правое поддерево
- Совпадение => поиск успешно завершен
- Нет вершины => поиск завершен неуспешно

Поиск в дереве

- Начинаем с корня
- Ключ в текущей вершине «>» => идём в левое поддерево
- Ключ в текущей вершине «<» => идём в правое поддерево
- Совпадение => поиск успешно завершен
- Нет вершины => поиск завершен неуспешно

T = O(h), h - высота

++/-- итератора в дереве

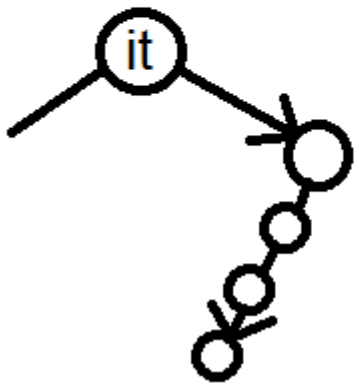
- Как найти succ(it) в BST?

++/-- итератора в дереве

- Как найти succ(it) в BST?
- Разбор случаев

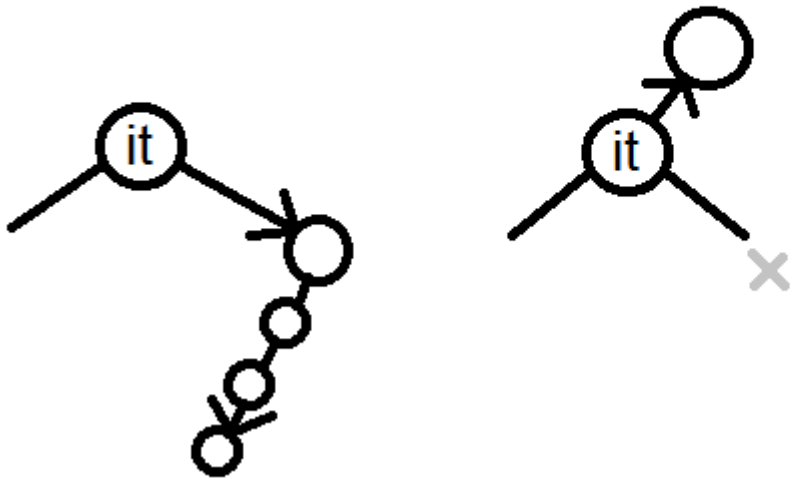
++/-- итератора в дереве

- Как найти succ(it) в BST?
- Разбор случаев:
  - Есть правый сын



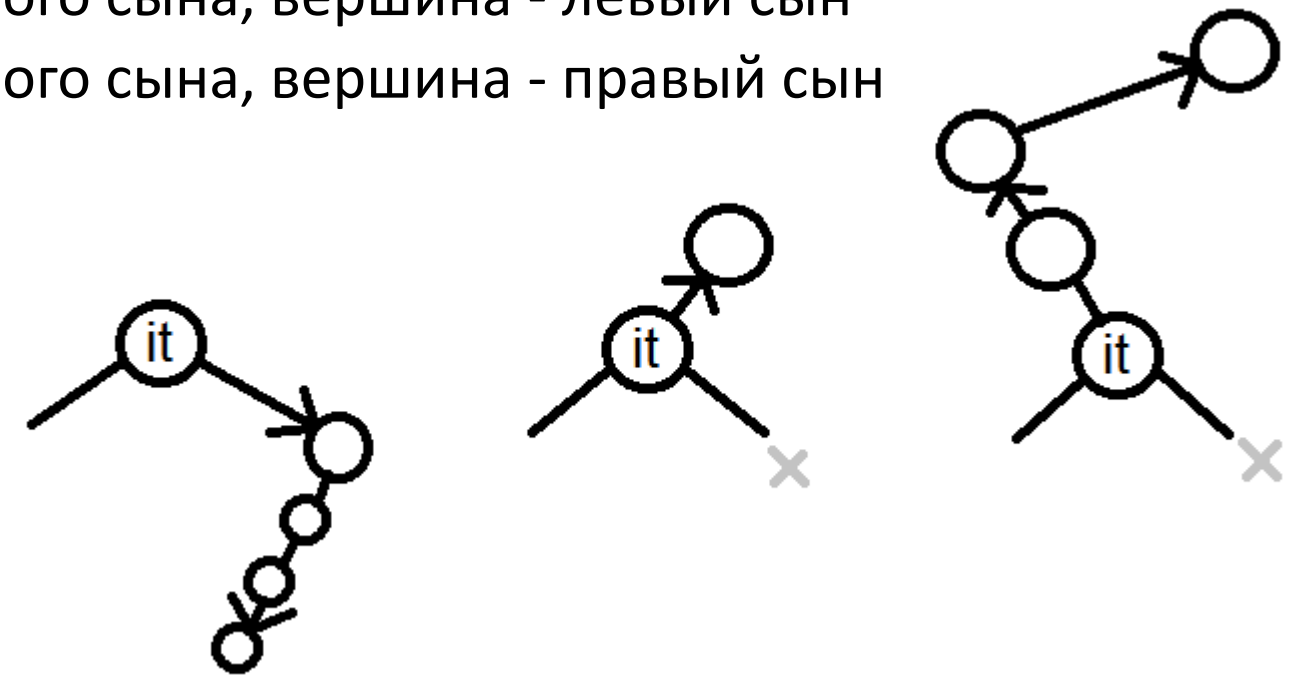
++/-- итератора в дереве

- Как найти succ(it) в BST?
- Разбор случаев:
  - Есть правый сын
  - Нет правого сына, вершина - левый сын



++/-- итератора в дереве

- Как найти succ(it) в BST?
- Разбор случаев:
  - Есть правый сын
  - Нет правого сына, вершина - левый сын
  - Нет правого сына, вершина - правый сын



++/-- итератора в дереве

- T_{succ} - ?

++/-- итератора в дереве

- $T_{succ} = O(h)$

++/-- итератора в дереве

- $T_{succ} = O(h)$
- Суммарное время итерации по всему дереву - ?

++/-- итератора в дереве

- $T_{succ} = O(h)$
- Суммарное время итерации по всему дереву = O(N)
  - По каждому ребру проходим дважды

Обходы дерева

- Pre-order

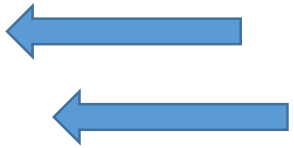
```
go(v) {
    go(v.l)
    go(v.r)
}
```

```
print(v.key)
```

Обходы дерева

- Pre-order
- In-order

```
go(v) {
    go(v.l)
    go(v.r)
}
```

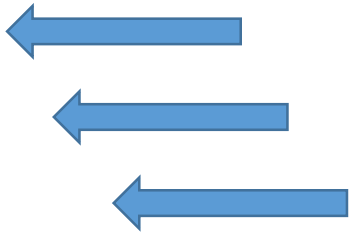


print(v.key)

Обходы дерева

- Pre-order
- In-order
- Post-order

```
go(v) {
    go(v.l)
    go(v.r)
}
```

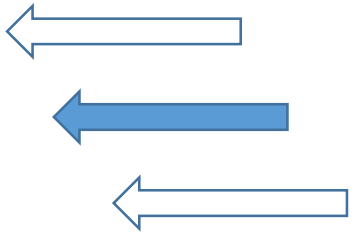


print(v.key)

Обходы дерева

- Pre-order
- In-order – нас интересует
- Post-order

```
go(v) {
    go(v.l)
    go(v.r)
}
```



```
print(v.key)
```

Обходы дерева: используемая память

- Расход памяти - ?

Обходы дерева: используемая память

- Расход памяти = O(h)

Обходы дерева: используемая память

- Расход памяти = O(h)
- Есть указатель на родителя => O(1)

Сбалансированное дерево поиска

- Требование: h = O(logN)

Сбалансированное дерево поиска

- Требование: $h = O(\log N)$
- Достаточное для этого условие:
  - Для любых вершин x, y, $|ch(x)| \leq 1$, $|ch(y)| \leq 1$ верно что $h(x)/h(y) \leq C$

Красно-чёрное дерево | RB-tree

- Сбалансированное BST

Красно-чёрное дерево | RB-tree

- Сопоставим вершинам цвета
  - c: V -> {R, B}

Красно-чёрное дерево | RB-tree

- Сопоставим вершинам цвета
  - c: V -> {R, B}
- Добавим фиктивные листья там, где нет сына

Красно-чёрное дерево | RB-tree

- Сопоставим вершинам цвета
  - c: V -> {R, B}
- Добавим фиктивные листья там, где нет сына
- Свойства:
  1. x – лист => c(x) = B

Красно-чёрное дерево | RB-tree

- Сопоставим вершинам цвета
  - c: V -> {R, B}
- Добавим фиктивные листья там, где нет сына
- Свойства:
  1. x – лист => c(x) = B
  2. y – родитель x, c(x) = R => c(y) = B

Красно-чёрное дерево | RB-tree

- Сопоставим вершинам цвета
  - c: V -> {R, B}
- Добавим фиктивные листья там, где нет сына
- Свойства:
  1. x – лист => c(x) = B
  2. y – родитель x, c(x) = R => c(y) = B
  3. Для всех листьев число чёрных предков одинаково (обозначим bh)

Красно-чёрное дерево | RB-tree

- Сопоставим вершинам цвета
  - c: V -> {R, B}
- Добавим фиктивные листья там, где нет сына
- Свойства:
  1. x – лист => c(x) = B
  2. y – родитель x, c(x) = R => c(y) = B
  3. Для всех листьев число чёрных предков одинаково (обозначим bh)

- Длина любого пути от bh до 2*bh

Красно-чёрное дерево | RB-tree

- Сопоставим вершинам цвета
  - c: V -> {R, B}
- Добавим фиктивные листья там, где нет сына
- Свойства:
  1. x – лист => c(x) = B
  2. y – родитель x, c(x) = R => c(y) = B
  3. Для всех листьев число чёрных предков одинаково (обозначим bh)

- Длина любого пути от bh до 2*bh
  - Следовательно h = O(logN)

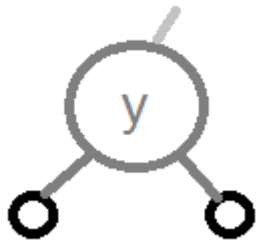
Дерево поиска: вставка и удаление

```
insert(X) {
    pos = search(X)
    pos.addNode(X)
}
```

```
erase(X) {
    pos = search(X)
    if (pos.r) {
        next = succ(pos)
        swap(next, pos)
    }
    pos.deleteNode()
}
```

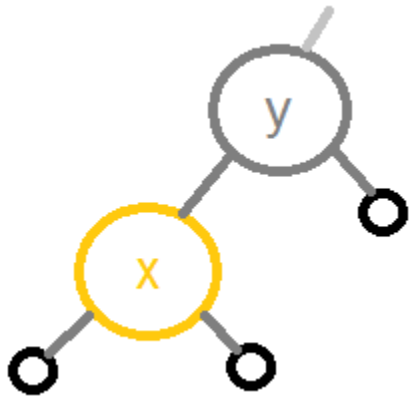
Красно-чёрное дерево: вставка

- Добавляем к у сына x



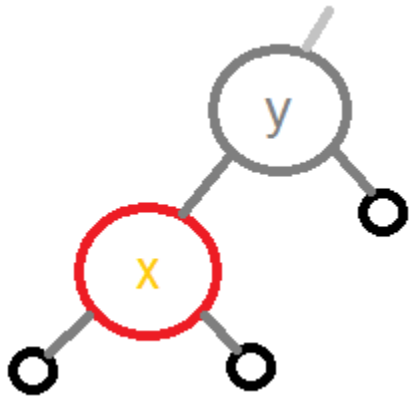
Красно-чёрное дерево: вставка

- Добавляем к у сына x
  - Как покрасить x?



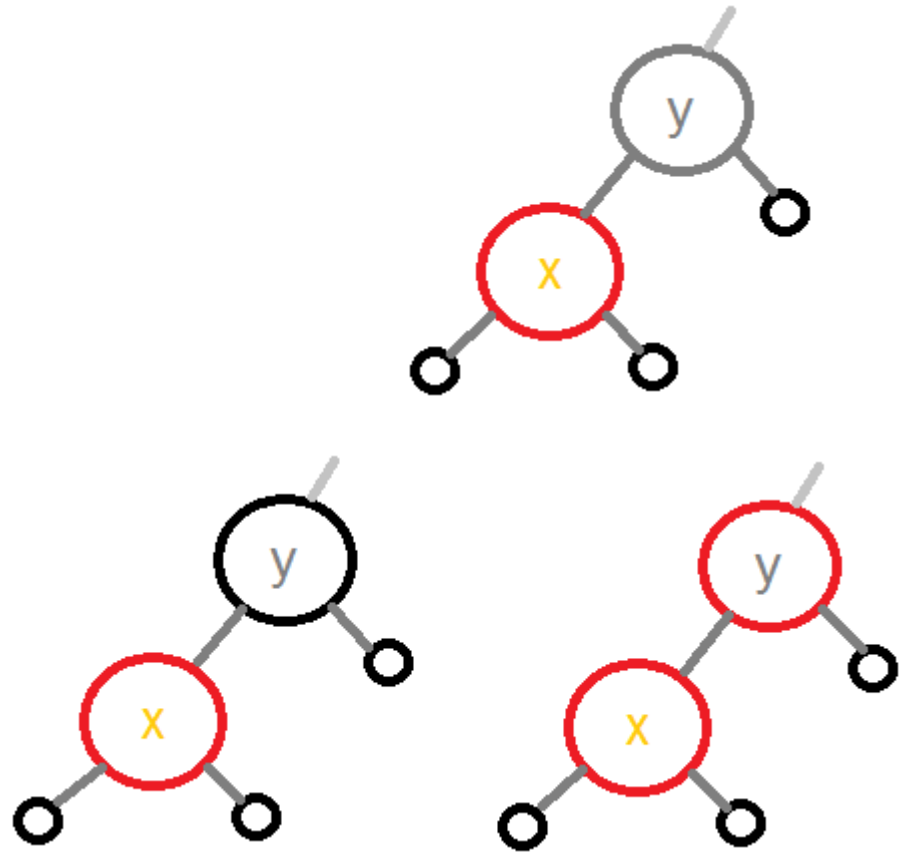
Красно-чёрное дерево: вставка

- Добавляем к у сына x
  - c(x) = R



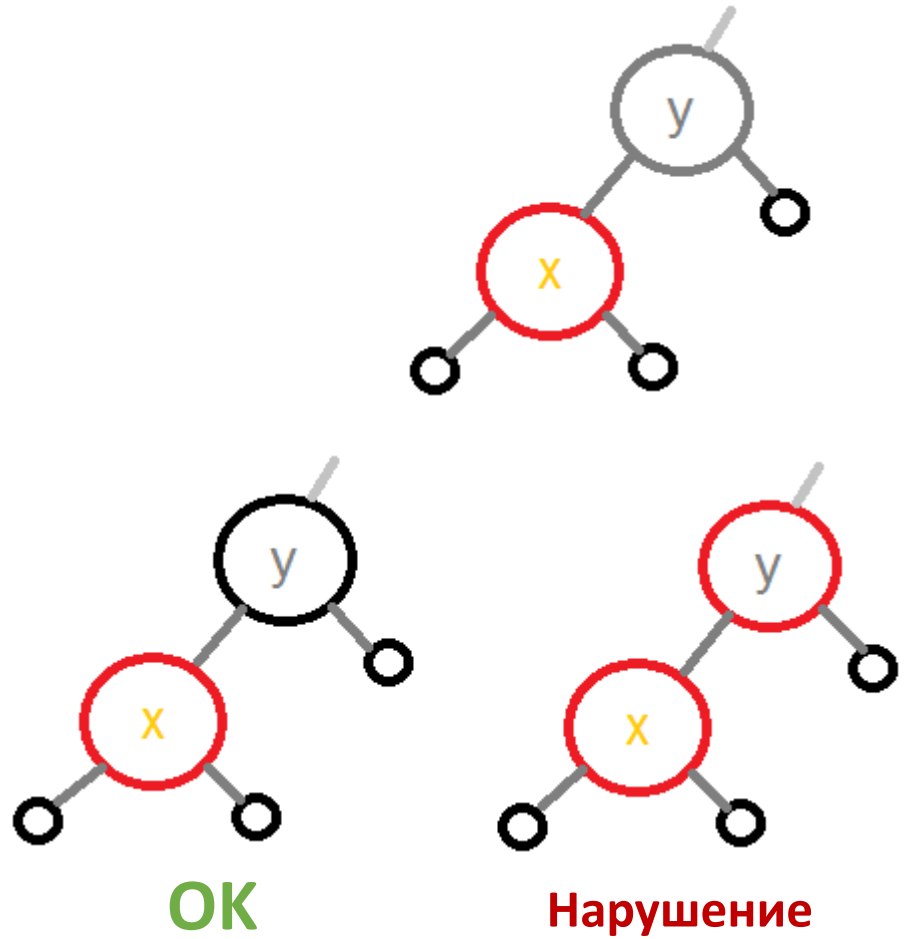
Красно-чёрное дерево: вставка

- Добавляем к у сына x
  - c(x) = R
- Два варианта c(y)



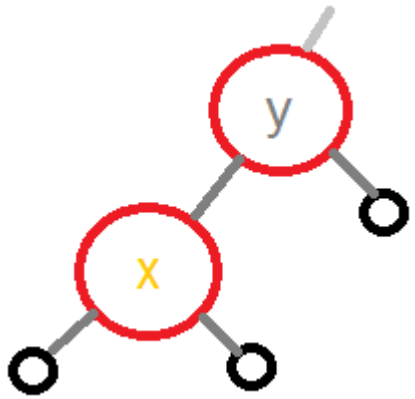
Красно-чёрное дерево: вставка

- Добавляем к у сына x
  - c(x) = R
- Два варианта c(y)
  - c(y) = B – OK
  - c(y) = R – проблема



Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R

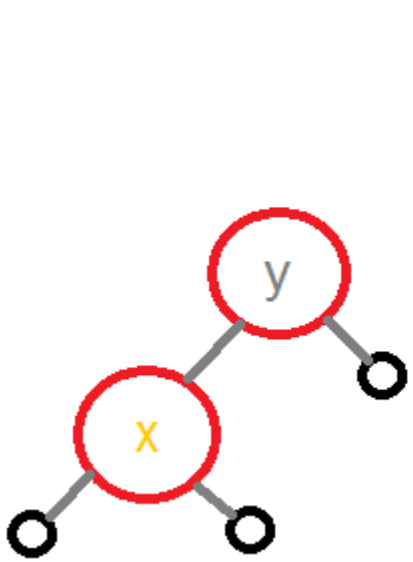


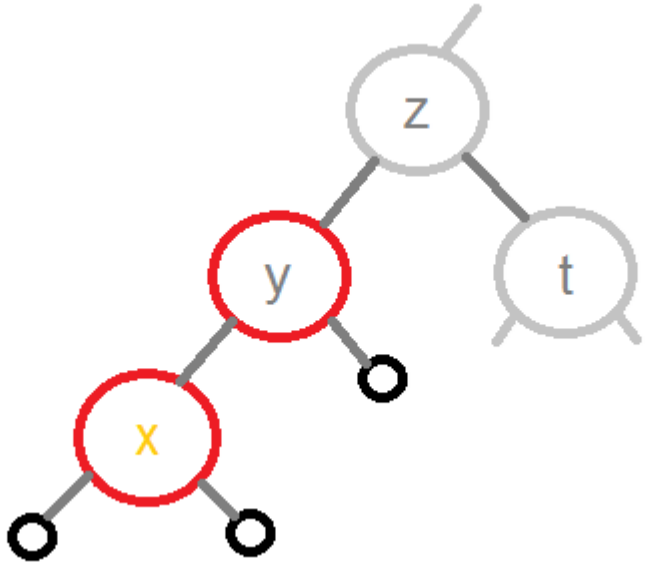
Красно-чёрное дерево: вставка

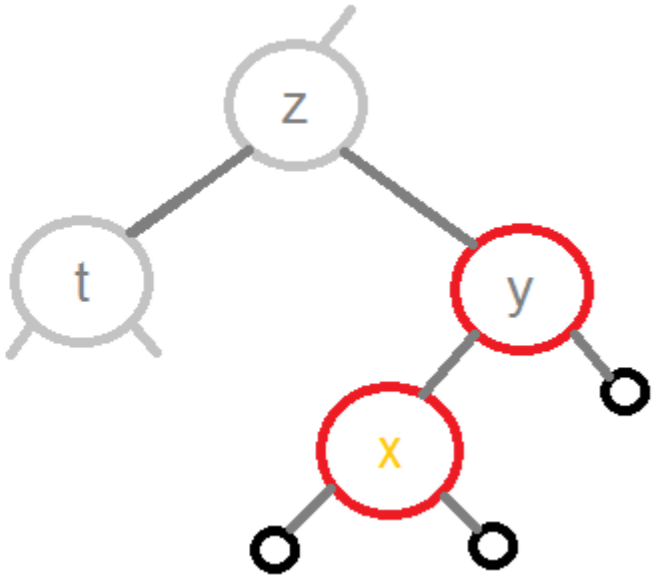
Устраняем проблему c(x) = c(y) = R

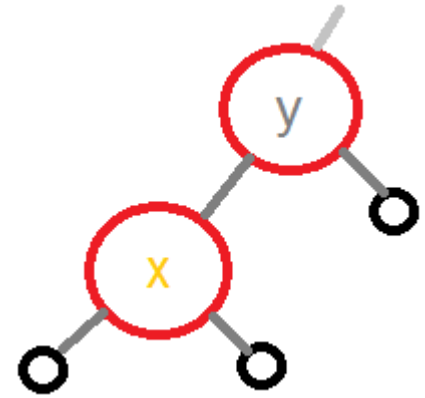
Варианты:

- y – корень
- y – левый или правый сын





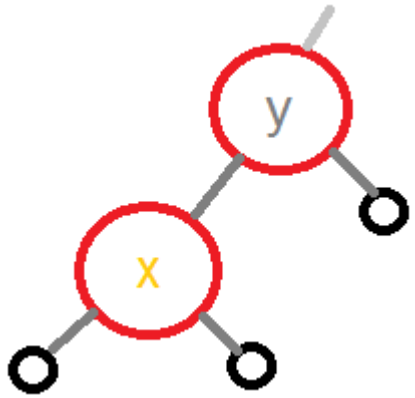


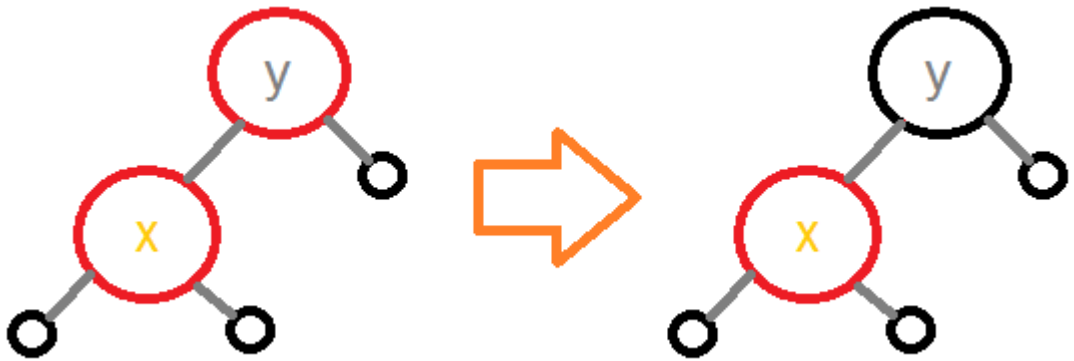


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – корень

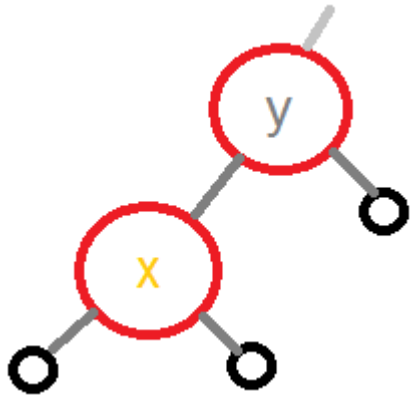
- c(y) := B

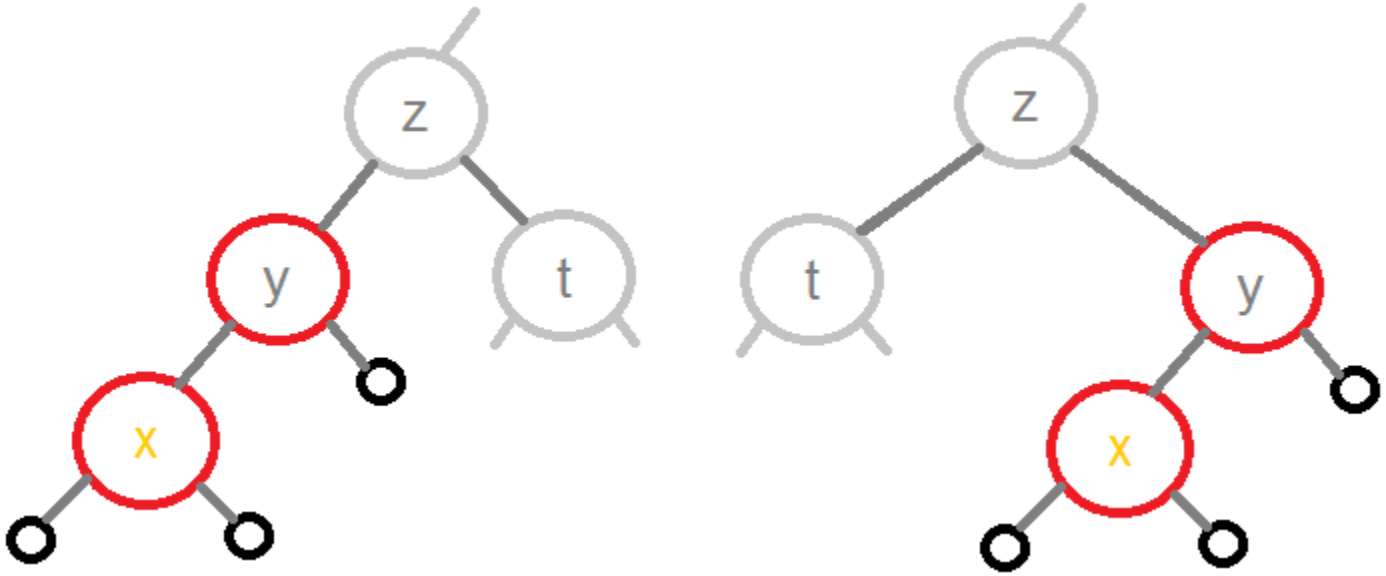




Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

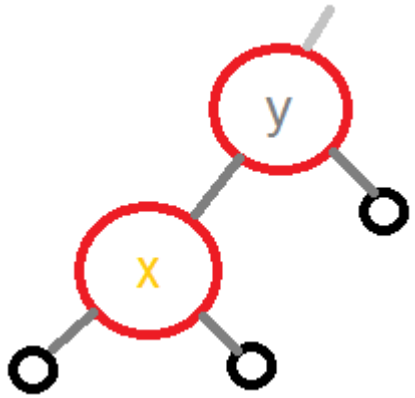


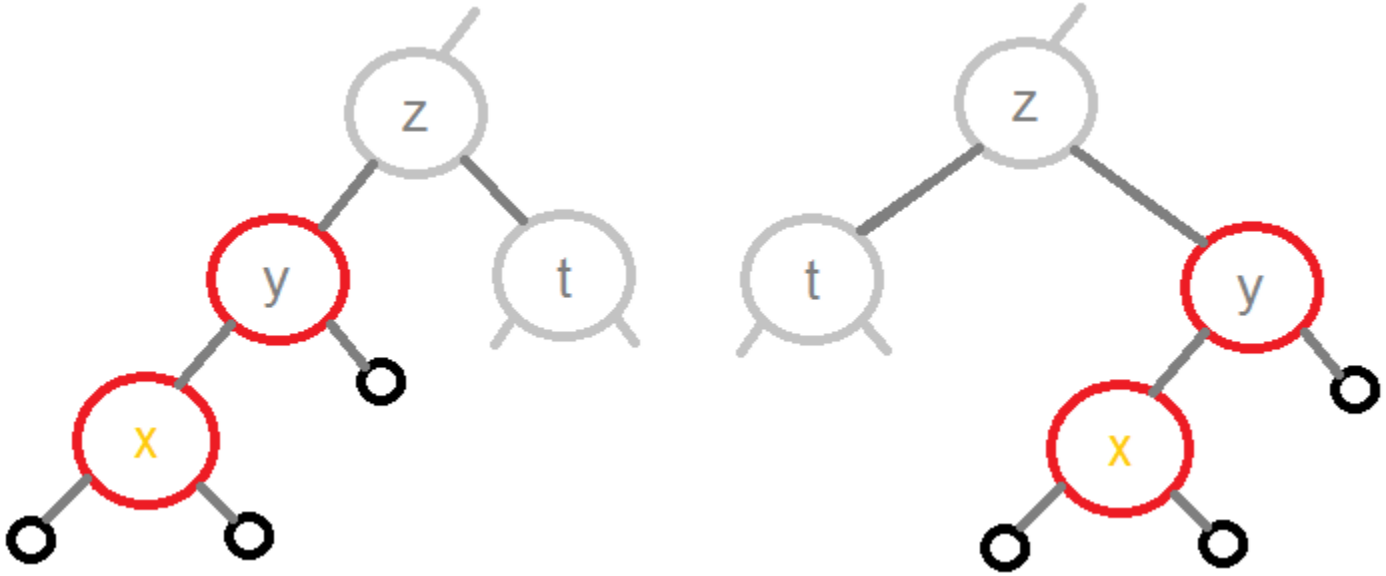


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

z – дед x, c(z) - ?

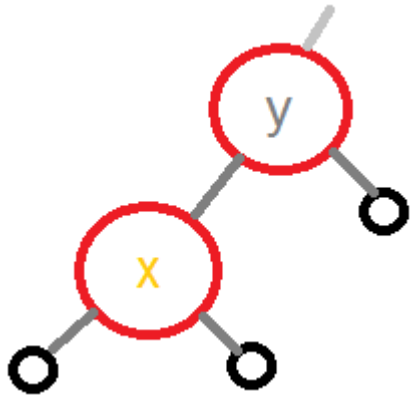


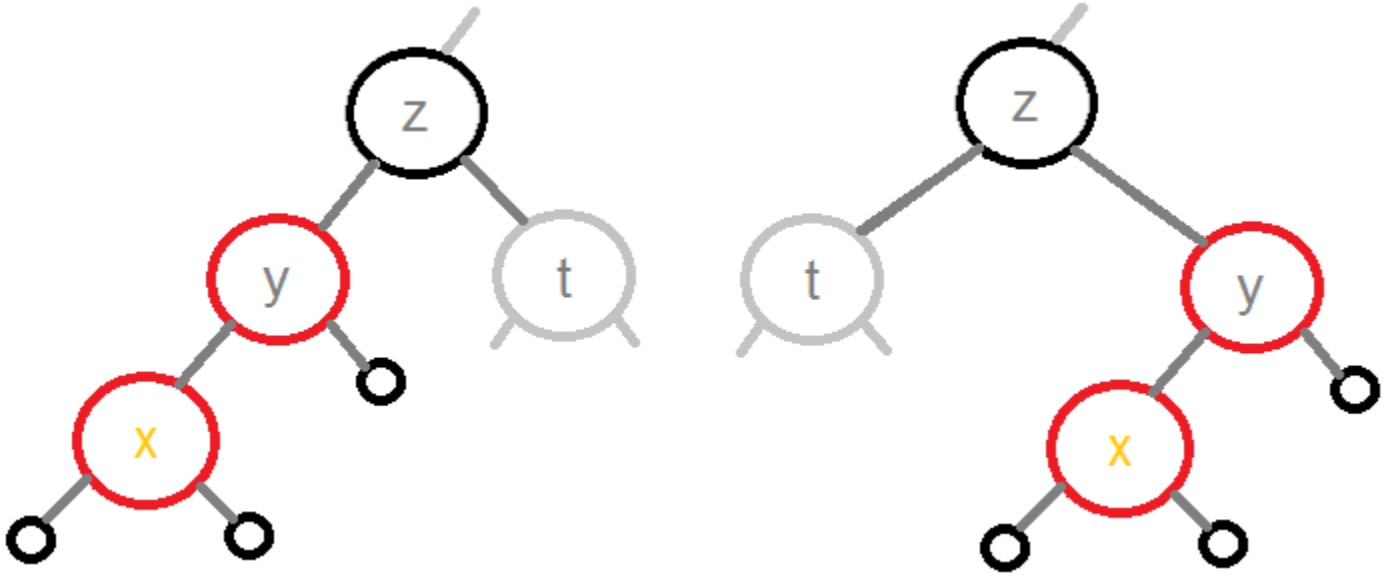


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

z – дед x, c(z) = B



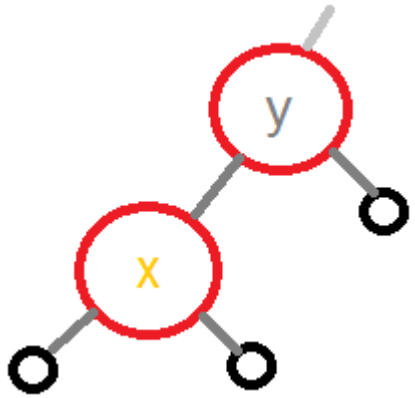


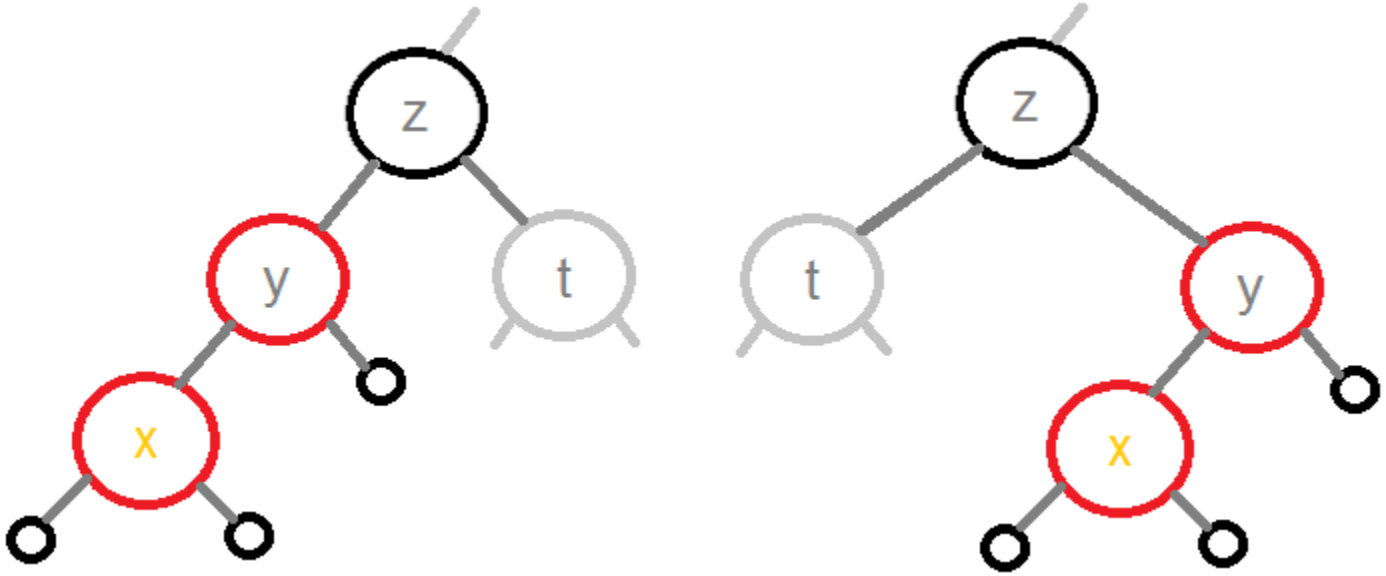
Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

z – дед x, c(z) = B

Всё зависит от дяди t





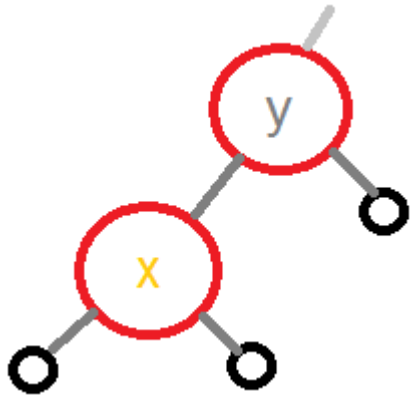
Красно-чёрное дерево: вставка

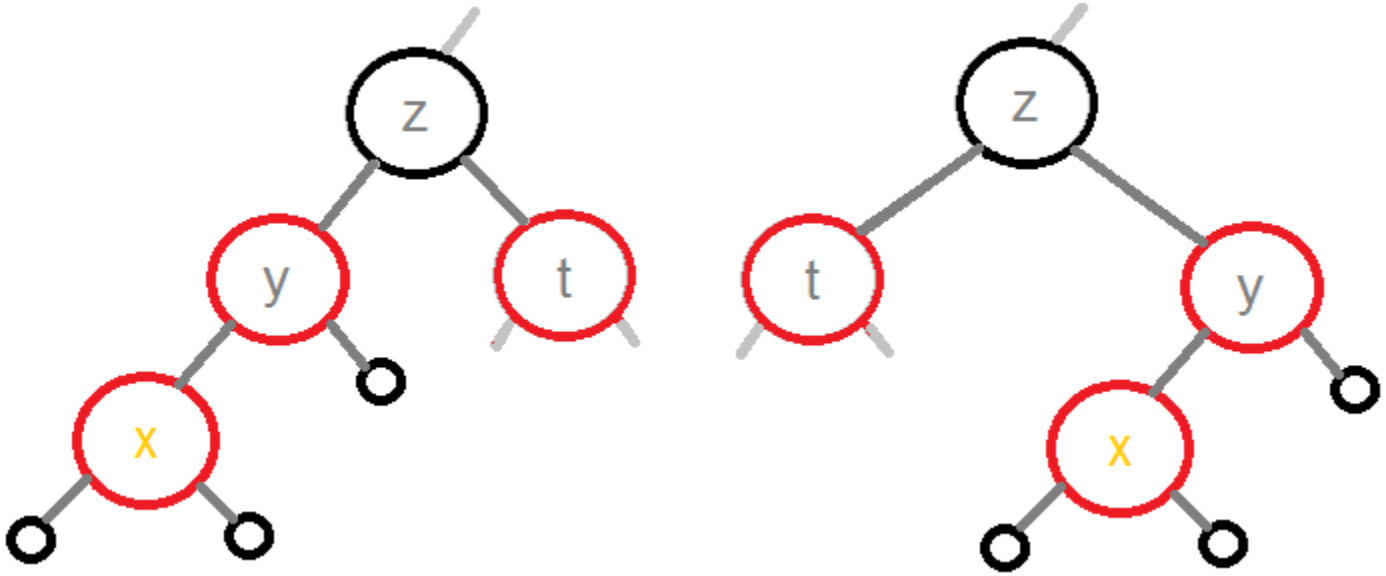
Устраняем проблему c(x) = c(y) = R, y – не корень

z – дед x, c(z) = B

Всё зависит от дяди t

1. c(t) = R

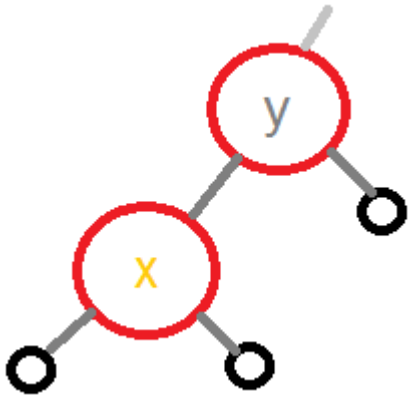


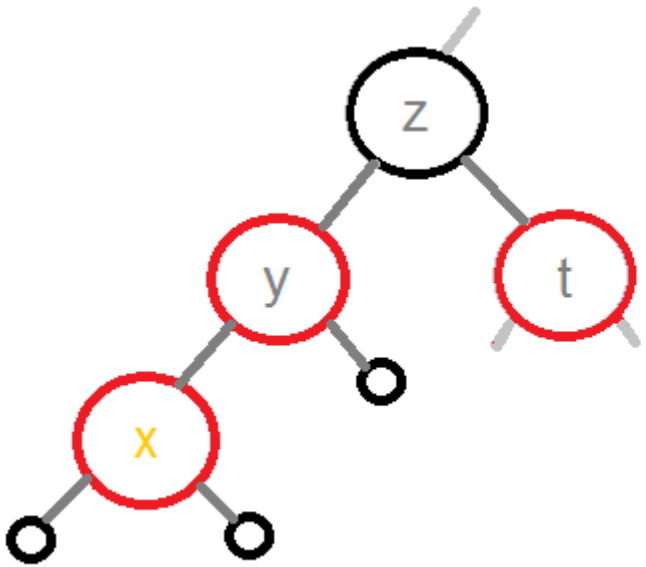


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

1. c(z) = B, c(t) = R

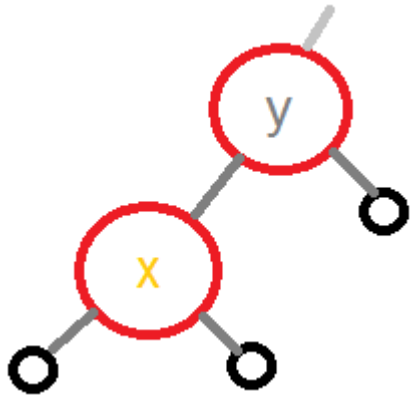


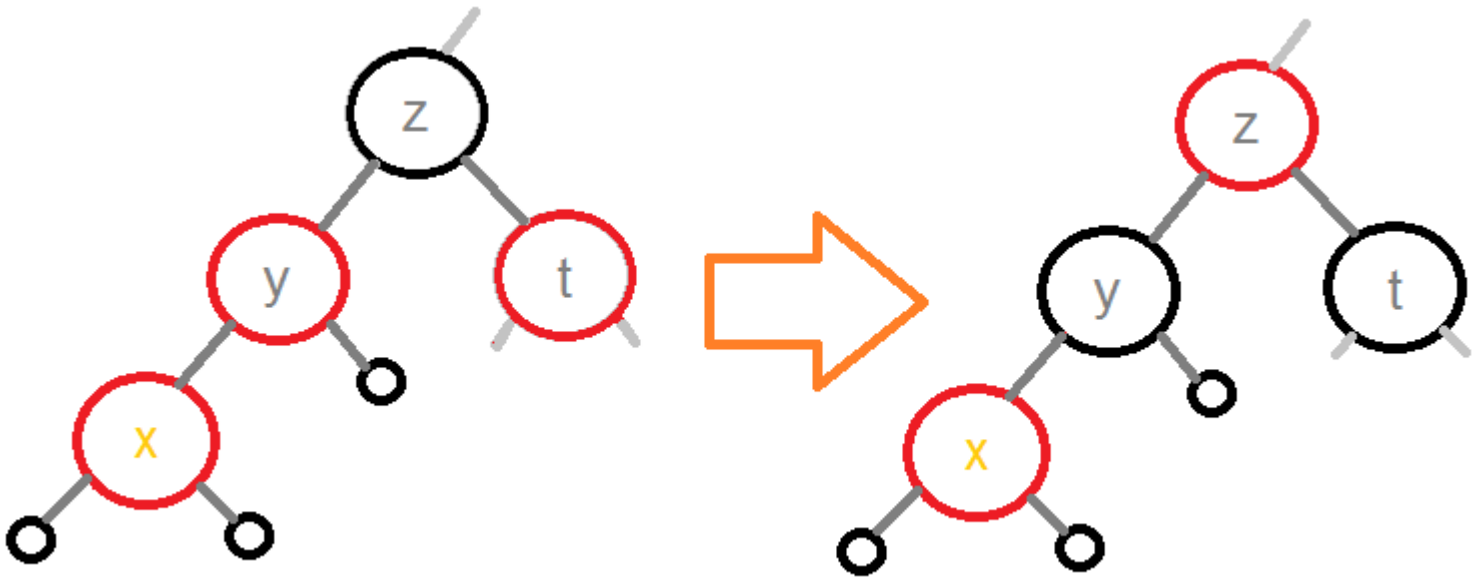


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

1. c(z) = B, c(t) = R
   - Перекрасим c(z) := R, c(y) := c(t) := B

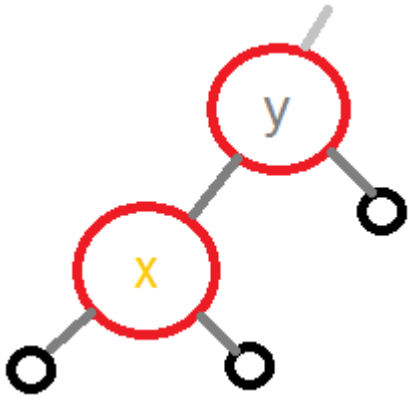


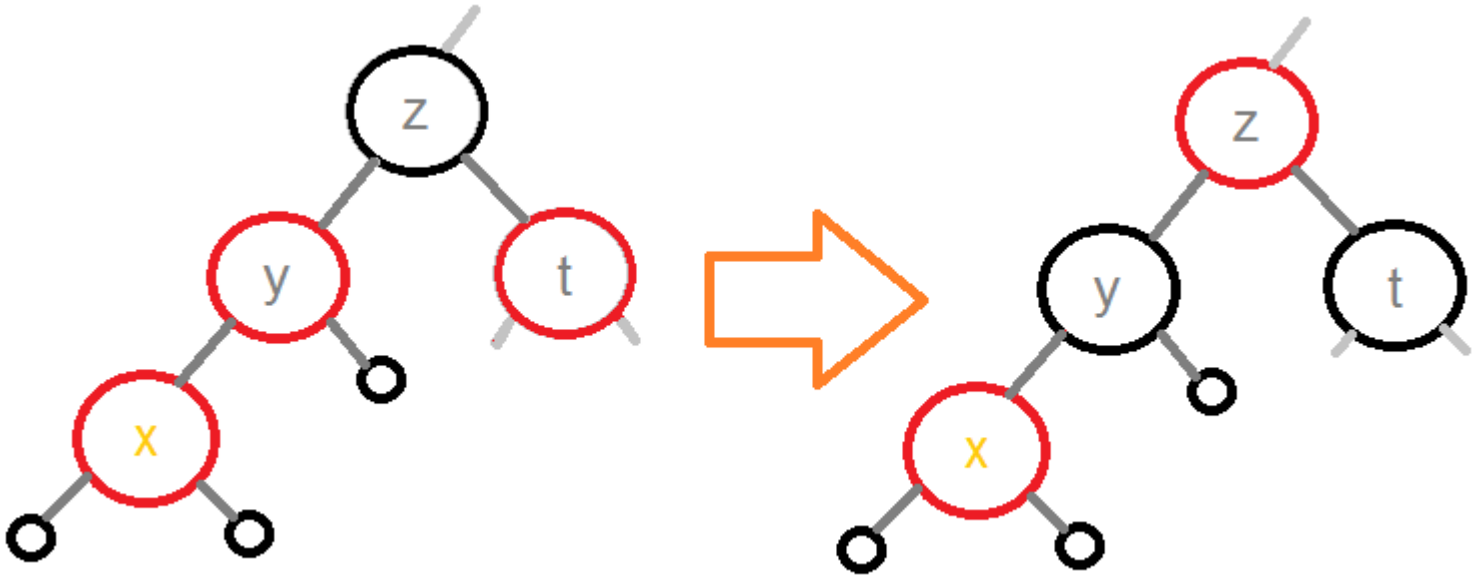


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

1. c(z) = B, c(t) = R
   - Перекрасим c(z) := R, c(y) := c(t) := B
   - Обработаем случай конфликта z с его родителем

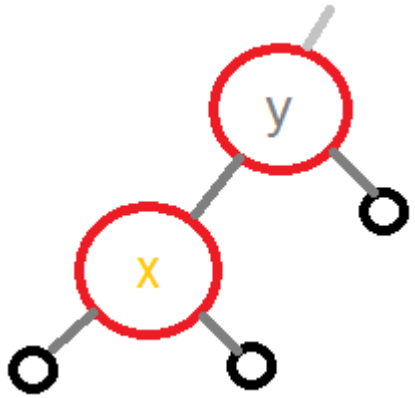


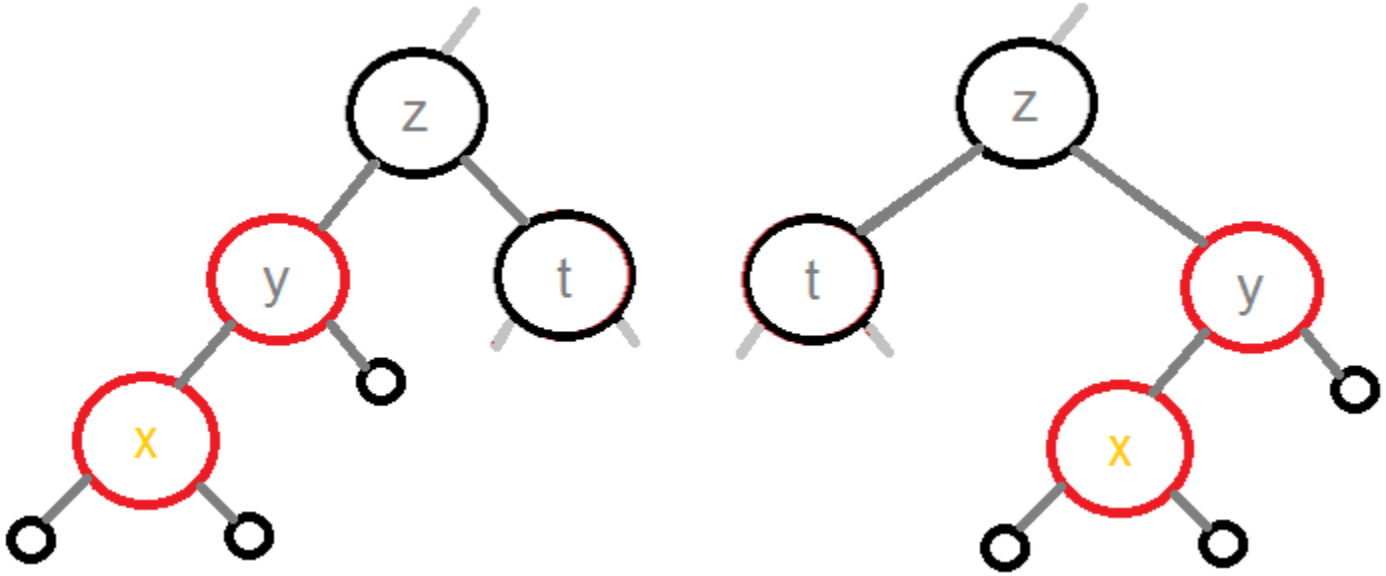


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

2. c(z) = c(t) = B

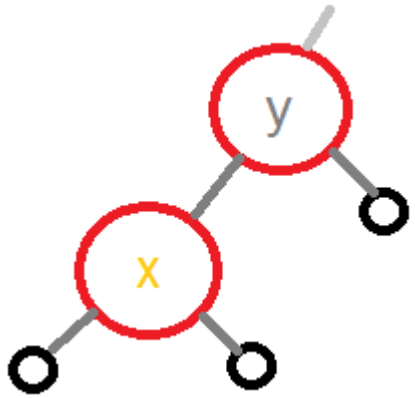


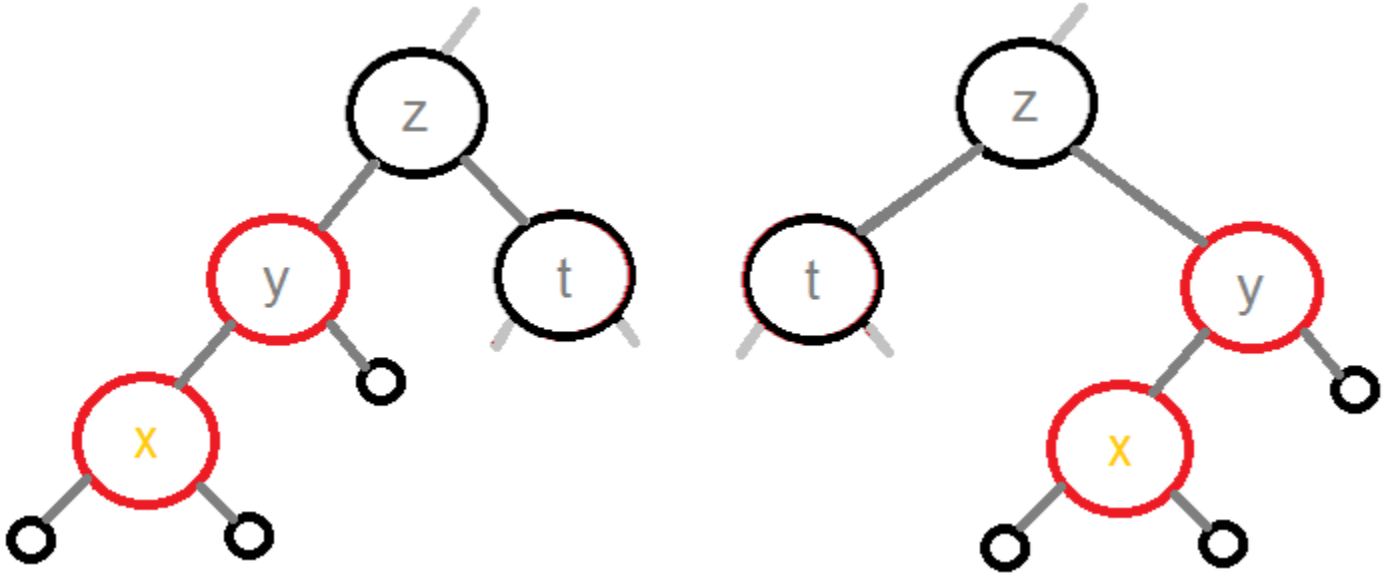


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

2. c(z) = c(t) = B
   - Перекраска не поможет

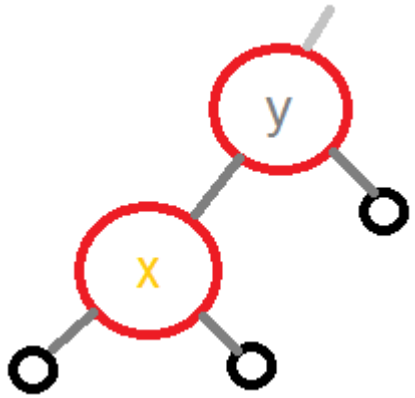


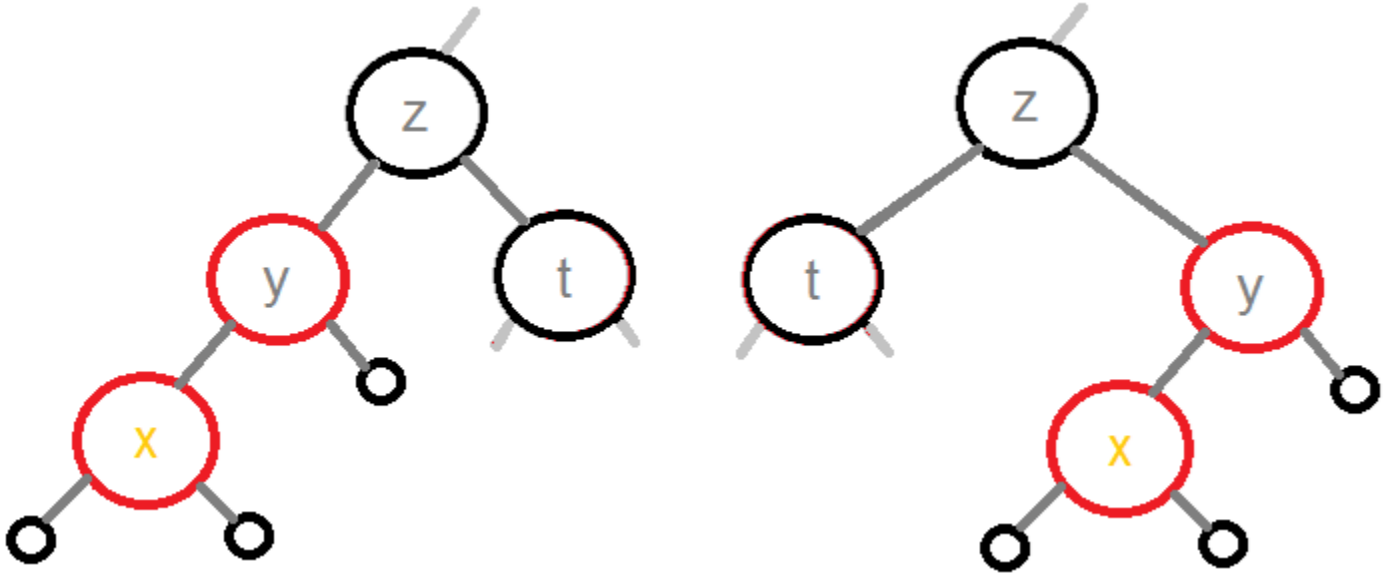


Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

2. c(z) = c(t) = B
   - Перекраска не поможет
   - Нужно поворачивать

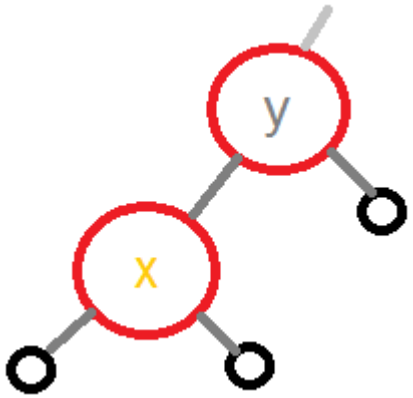


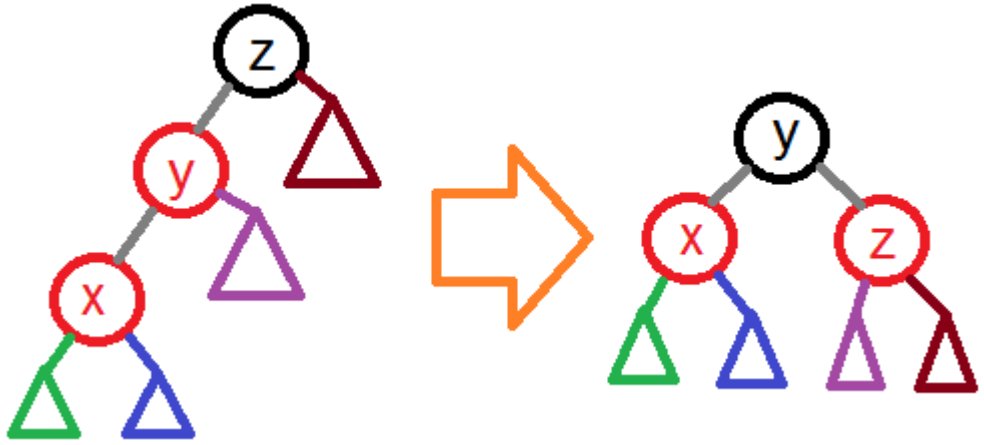


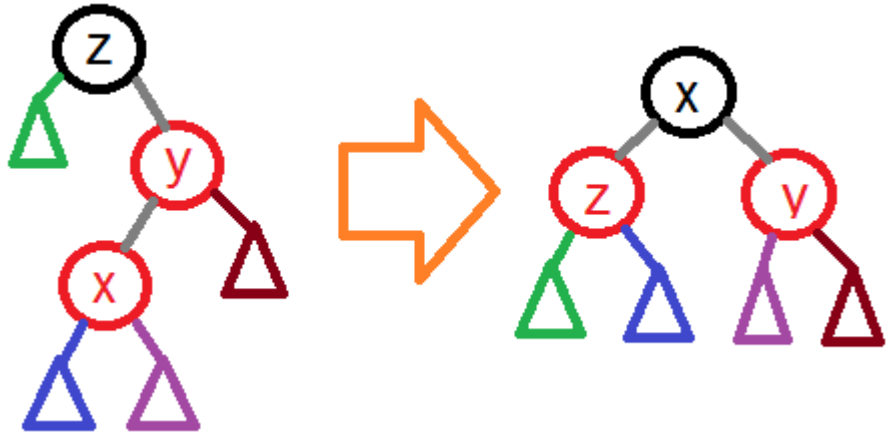
Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

2. c(z) = c(t) = B
   - Перекраска не поможет
   - Нужно поворачивать



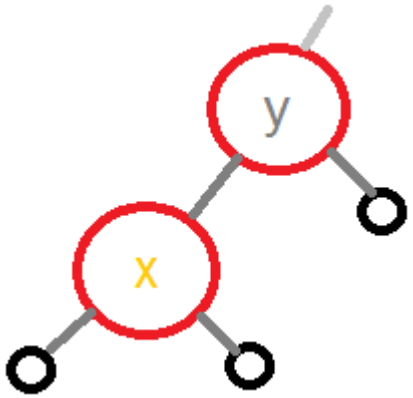


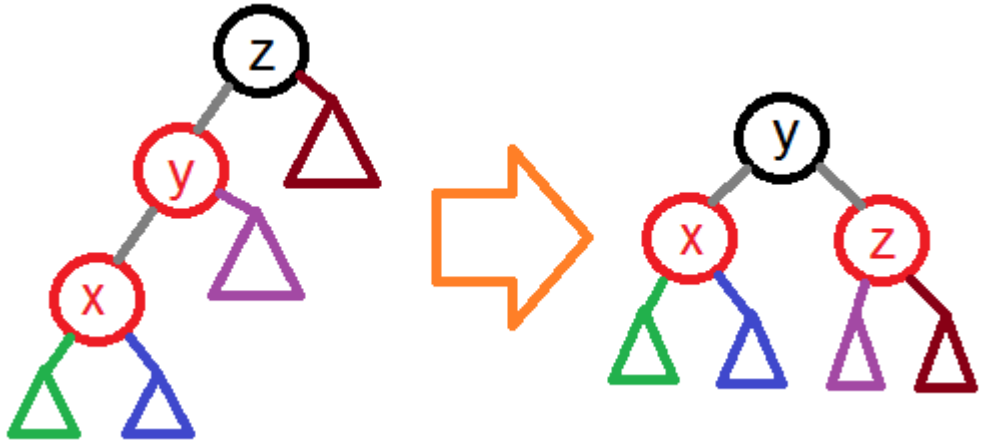


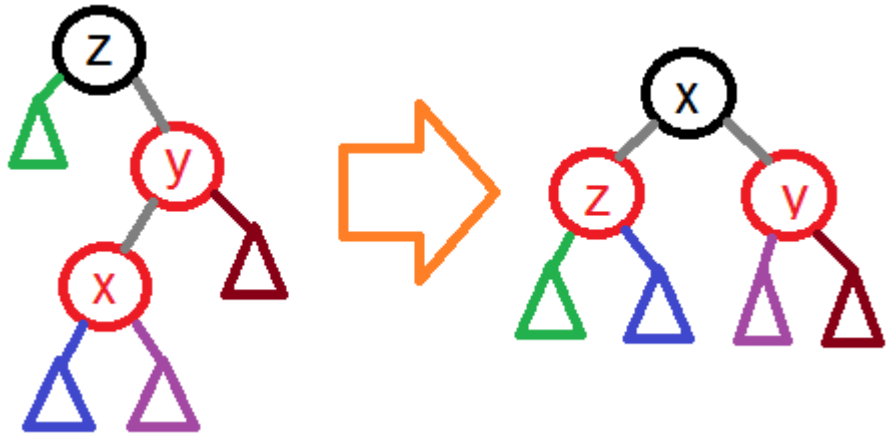
Красно-чёрное дерево: вставка

Устраняем проблему c(x) = c(y) = R, y – не корень

2. c(z) = c(t) = B
   - Перекраска не поможет
   - Нужно поворачивать
   - После поворота новый конфликт не образуется







Красно-чёрное дерево: вставка

Сложность

- $T_{insert} = O(h) = O(logN)$

Красно-чёрное дерево: удаление

- Разбор случаев

Красно-чёрное дерево: удаление

- Разбор случаев
  - достаётся вам в качестве упражнения

Splay-дерево

- BST с амортизированным временем работы

Splay-дерево

- BST с амортизированным временем работы

Определим функцию splay(x)

Splay-дерево

- BST с амортизированным временем работы

Определим функцию splay(x)

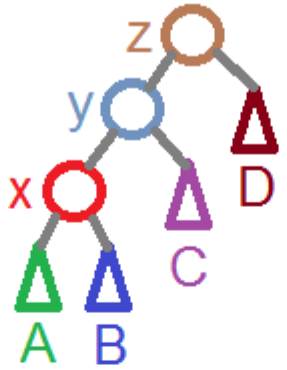
- перемещает x в корень

Splay-дерево

- BST с амортизированным временем работы

Определим функцию splay(x)

- перемещает x в корень
- три случая

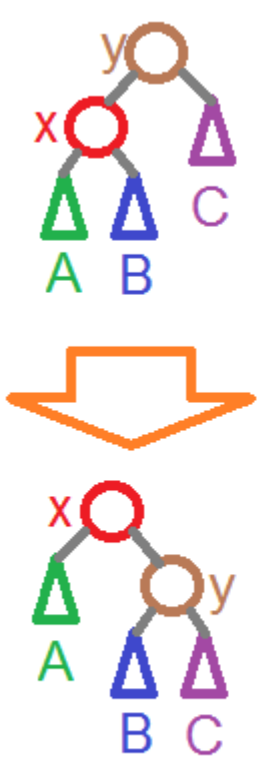


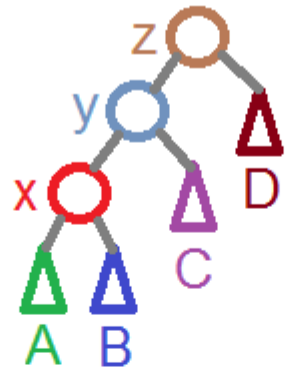
Splay-дерево

- BST с амортизированным временем работы

Определим функцию splay(x)

- перемещает x в корень
- три случая



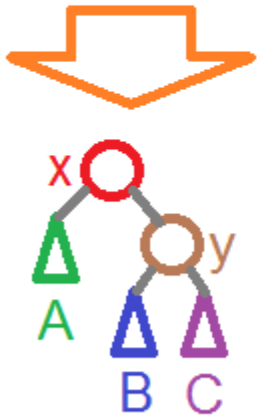


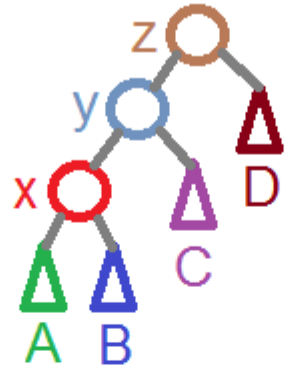
Splay-дерево

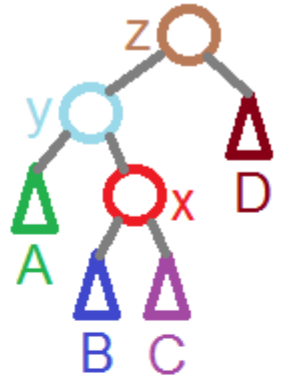
- BST с амортизированным временем работы

Определим функцию splay(x)

- перемещает x в корень
- три случая





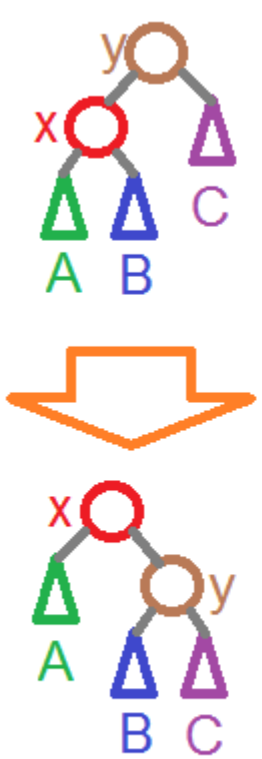


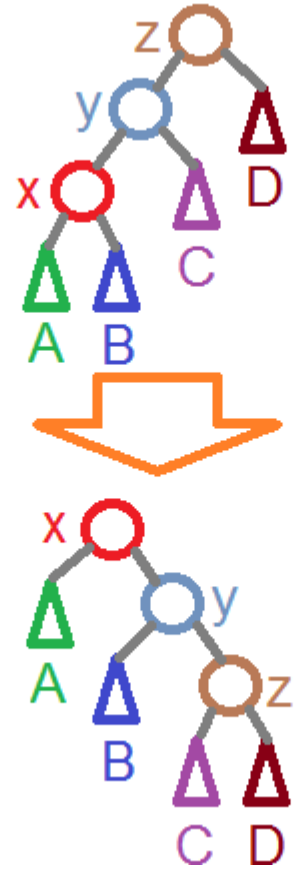
Splay-дерево

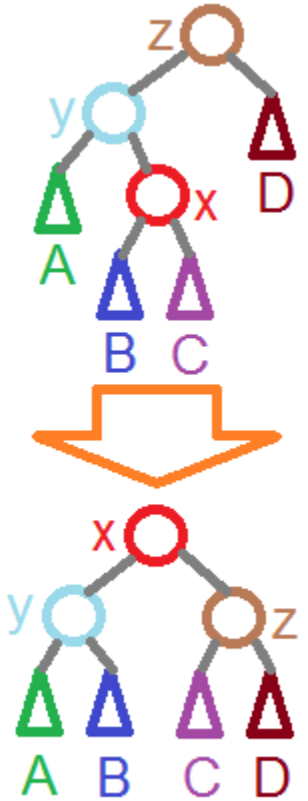
- BST с амортизированным временем работы

Определим функцию splay(x)

- перемещает x в корень
- три случая







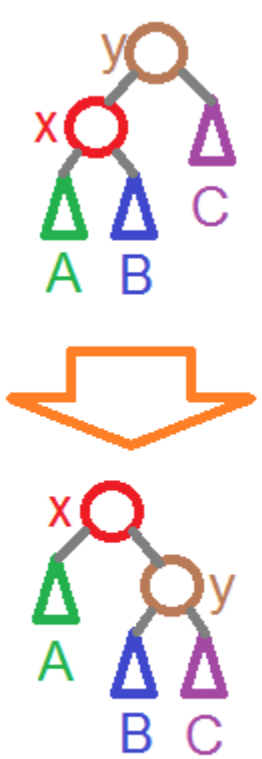
Splay-дерево

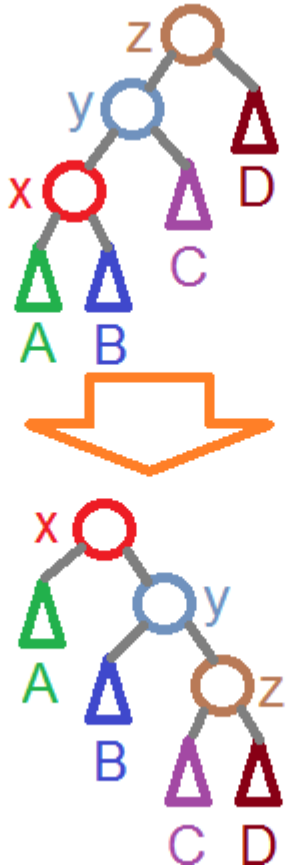
- BST с амортизированным временем работы

Определим функцию splay(x)

- перемещает x в корень
- три случая

- Свойства splay
  1. $T_{splay} = \theta(h(x))$





Splay-дерево

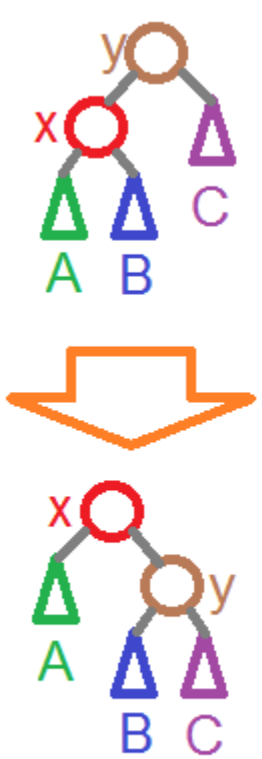
- BST с амортизированным временем работы

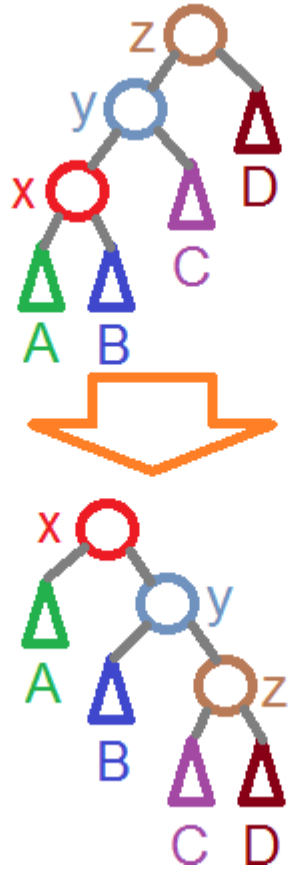
Определим функцию splay(x)

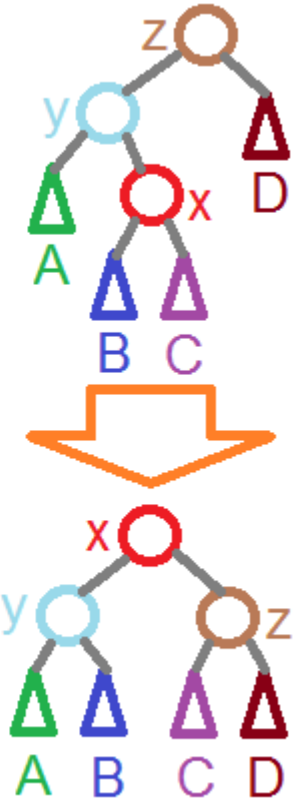
- перемещает x в корень
- три случая

- Свойства splay
  1. $T_{splay} = \theta(h(x))$
  2. Аморт. сложность

     $T''_{splay} = O(\log N)$







Splay: анализ

Определения:

- w(x) = число вершин в поддереве x

Splay: анализ

Определения:

- w(x) = число вершин в поддереве x
- r(x) = floor_log(w(x))

Splay: анализ

Определения:

- w(x) = число вершин в поддереве x
- r(x) = floor_log(w(x))

Определим потенциал $\Phi = \sum_x r(x)$

Splay-дерево: реализация методов

```
splay_find(k) {
    x = find(k)
    ?
}
```

Splay-дерево: реализация методов

```
splay_find(k) {
    x = find(k)
    splay(x)
}
```

Splay-дерево: реализация методов

```
splay_find(k) {
    x = find(k)
    splay(x)
}
```

- $T_{find} = \theta(h)$
- $T_{splay} = \theta(h)$

Splay-дерево: реализация методов

```
splay_find(k) {
    x = find(k)
    splay(x) // оплачивает работу find
}
```

- $T_{find} = \theta(h)$
- $T_{splay} = \theta(h)$

Splay-дерево: реализация методов

```
splay_find(k) {
    x = find(k)
    splay(x) // оплачивает работу find
}
```

- $T_{find} = \theta(h)$
- $T_{splay} = \theta(h)$

Заменим T на T’:

- $T'_{find} = 0$
- $T'_{splay} = T_{find} + T_{splay} = \theta(h)$

Splay-дерево: реализация методов

```
splay_find(k) {
    x = find(k)
    splay(x) // оплачивает работу find
}
```

- $T_{find} = \theta(h)$
- $T_{splay} = \theta(h)$

Заменим T на T’:

- $T'_{find} = 0$
- $T'_{splay} = T_{find} + T_{splay} = \theta(h)$

Аморт. сложность

- $T''_{splay} = O(\log N)$

Splay-дерево: реализация методов

```
splay_insert(k) {
    x = insert(k)
    splay(x) // оплачивает работу insert*
}
```

- $T_{insert} = \theta(h)$
- $T_{splay} = \theta(h)$

Заменим T на T’:

- $T'_{insert} = 0$
- $T'_{splay} = T_{insert} + T_{splay} = \theta(h)$

Аморт. сложность

- $T''_{splay} = O(\log N)$

Splay-дерево: реализация методов

```
splay_insert(k) {
    x = insert(k)
    splay(x) // оплачивает работу insert*
}
```

- $T_{insert} = \theta(h)$
- $T_{splay} = \theta(h)$

Заменим T на T’:

- $T'_{insert} = 0$
- $T'_{splay} = T_{insert} + T_{splay} = \theta(h)$

Аморт. сложность

- $T''_{splay} = O(\log N)$

---

* - могут вырасти ранги

Splay-дерево: реализация методов

```
splay_erase(k) {
    x = find(k)
    erase(x)
    splay(?) – откуда вызывать splay?
}
```

Splay: анализ

Амортизационный анализ

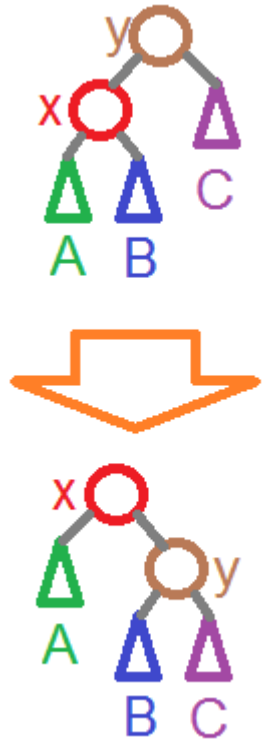
- $T'' = T + \Phi' - \Phi$

Splay: анализ

Амортизационный анализ

- T” = T + Φ’ – Φ

1.

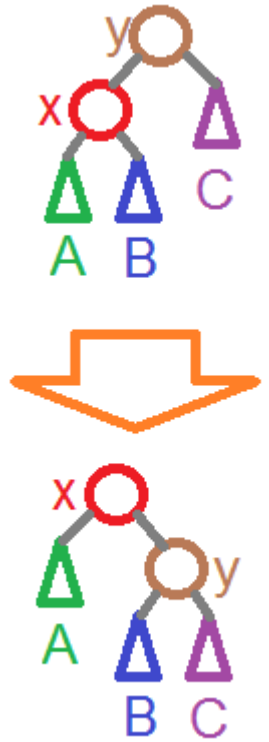


Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

1. $T'' = 1 + 3(r'(x) - r(x))$



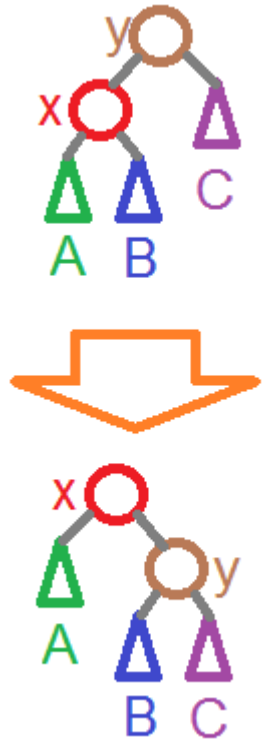
Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

1. $T'' = 1 + 3(r'(x) - r(x))$

- $T = 1$



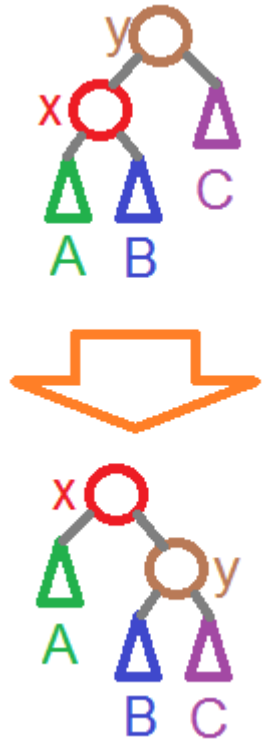
Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

1. $T'' = 1 + 3(r'(x) - r(x))$

- $T = 1$
- $\Phi' - \Phi = r'(x) + r'(y) - r(x) - r(y)$



Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

1. $T'' = 1 + 3(r'(x) - r(x))$

- $T = 1$
- $\Phi' - \Phi = \cancel{r'(x)} + r'(y) - r(x) \cancel{- r(y)}$

Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

1. $T'' = 1 + 3(r'(x) - r(x))$

- $T = 1$
- $\Phi' - \Phi = \cancel{r'(x)} + r'(y) - r(x) \cancel{- r(y)} \le r'(x) - r(x)$

Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

1. $T'' = 1 + 3(r'(x) - r(x))$

- $T = 1$
- $\Phi' - \Phi = \cancel{r'(x)} + r'(y) - r(x) \cancel{- r(y)} \le r'(x) - r(x)$
  - $r'(x) - r(x) \ge 0$

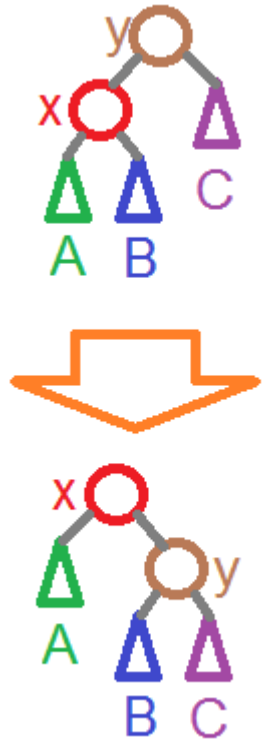
Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

1. $T'' = 1 + 3(r'(x) - r(x))$

- $T = 1$
- $\Phi' - \Phi = \cancel{r'(x)} + r'(y) - r(x) \cancel{- r(y)} \le r'(x) - r(x)$
  - $r'(x) - r(x) \ge 0$
- $\Phi' - \Phi \le 3(r'(x) - r(x))$

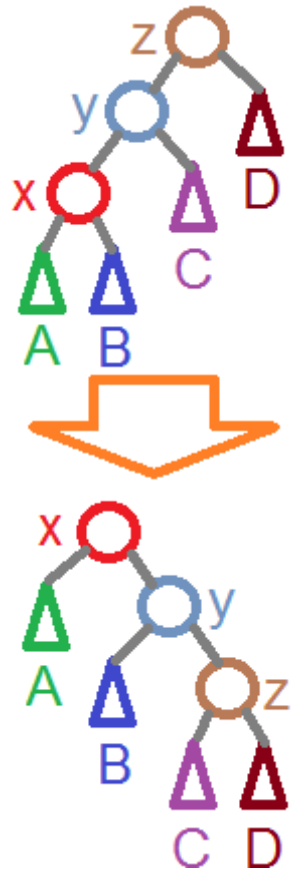


Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

2. $T'' = 3(r'(x) - r(x))$

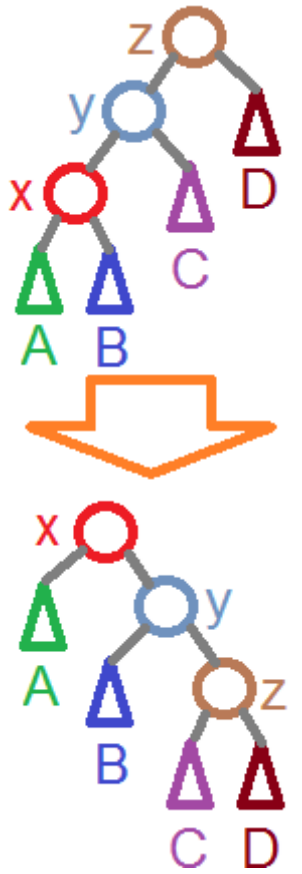


Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

2. $T'' = 3(r'(x) - r(x))$ // без «+1»



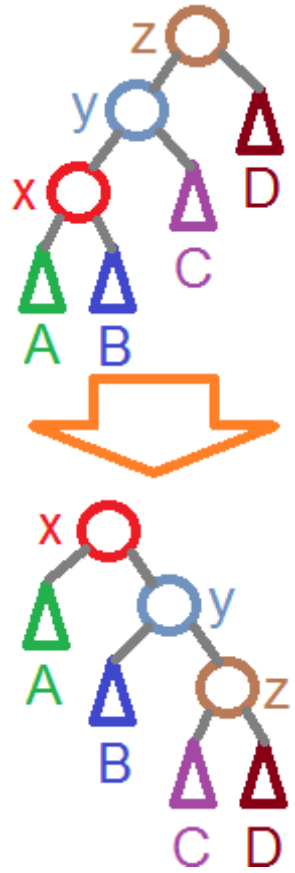
Splay: анализ

Амортизационный анализ

- T” = T + Φ’ – Φ

2. T” = 3(r’(x) – r(x)) // без «+1»

- В итоге для h шагов получим оценку
  - $T''_{splay} = 1 + 3(r'(x) – r(x))$



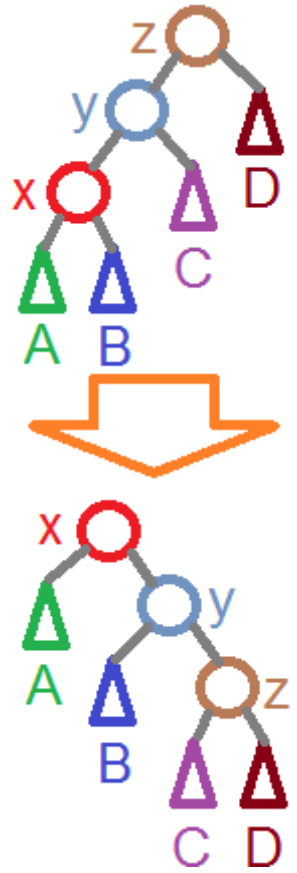
Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

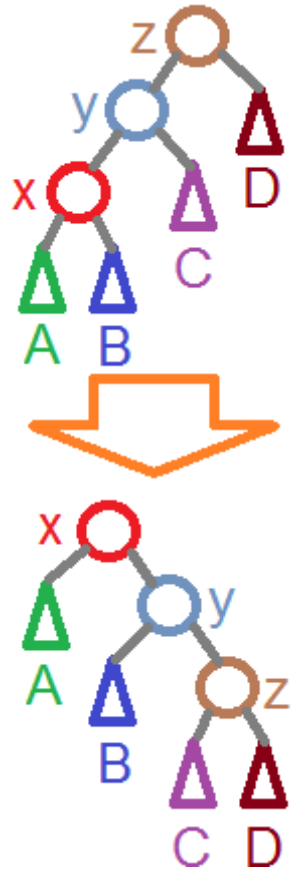
2. $T'' = 3(r'(x) - r(x))$ // без «+1»

- В итоге для h шагов получим оценку
  - $T''_{splay} = 1 + 3(r'(x) - r(x)) = O(\log N)$



Splay: анализ

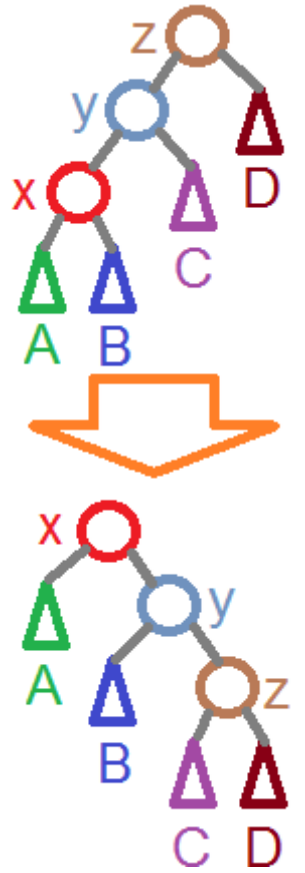
$T'' = 3(r'(x) - r(x))$



Splay: анализ

$T'' = 3(r'(x) - r(x))$

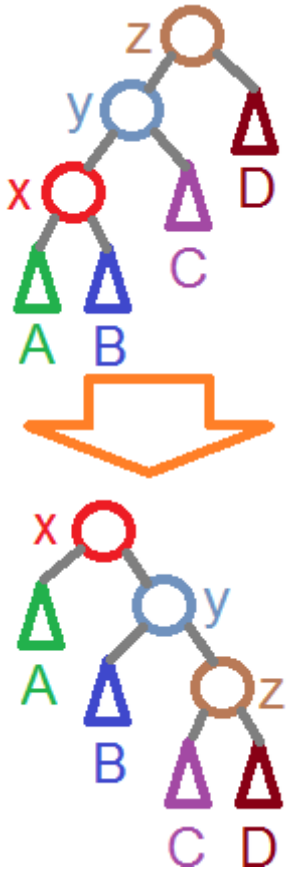
$T'' = T + \Phi' - \Phi$



Splay: анализ

$T'' = 3(r'(x) - r(x))$

$T'' = T + \Phi' - \Phi = 1 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$



Splay: анализ

$T'' = 3(r'(x) - r(x))$

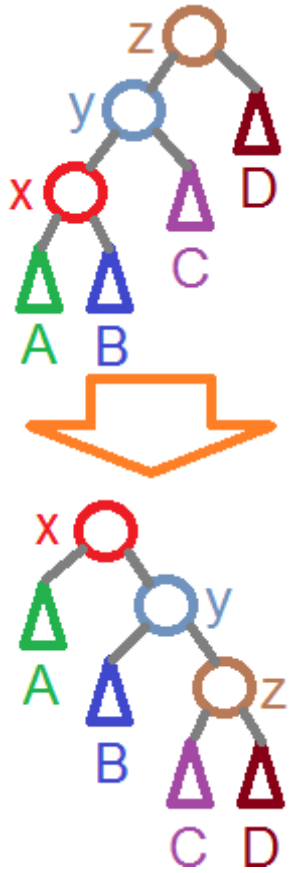
$T'' = T + \Phi' - \Phi = 1 \sout{+ r'(x)} + r'(y) + r'(z) - r(x) - r(y) \sout{- r(z)}$

Splay: анализ

$T'' = 3(r'(x) - r(x))$

$T'' = T + \Phi' - \Phi = 1 \sout{+ r'(x)} + r'(y) + r'(z) - r(x) - r(y) \sout{- r(z)} \leq$

- $r'(y), r'(z) \leq r'(x)$

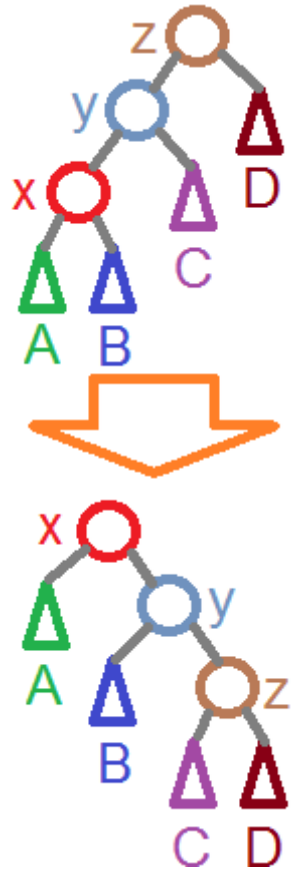


Splay: анализ

$T'' = 3(r'(x) - r(x))$

$T'' = T + \Phi' - \Phi = 1 \sout{+ r'(x)} + r'(y) + r'(z) - r(x) - r(y) \sout{- r(z)} \leq$

- $r'(y), r'(z) \leq r'(x)$
- $r(y), r(z) \geq r(x)$



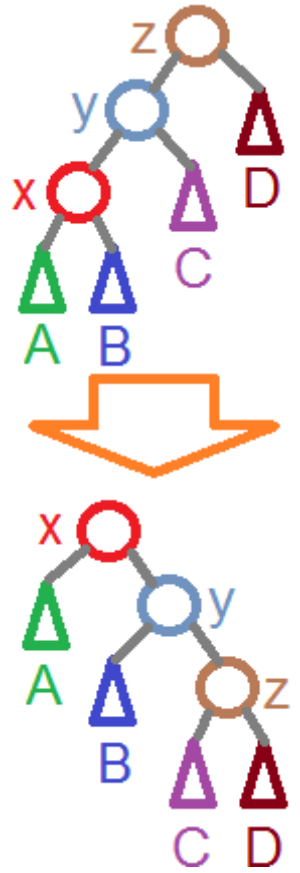
Splay: анализ

$T'' = 3(r'(x) - r(x))$

$T'' = T + \Phi' - \Phi = 1 \sout{+ r'(x)} + r'(y) + r'(z) - r(x) - r(y) \sout{- r(z)} \leq$

- $r'(y), r'(z) \leq r'(x)$
- $r(y), r(z) \geq r(x)$

$\leq 1 + 2(r'(x) - r(x))$



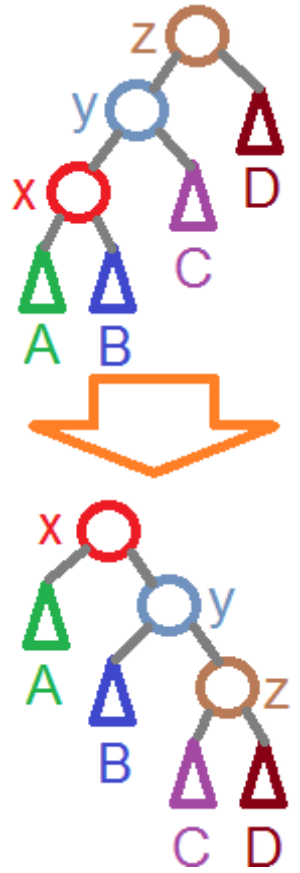
Splay: анализ

$T'' = 3(r'(x) - r(x))$

$T'' = T + \Phi' - \Phi = 1 \sout{+ r'(x)} + r'(y) + r'(z) - r(x) - r(y) \sout{- r(z)} \leq$

- $r'(y), r'(z) \leq r'(x)$
- $r(y), r(z) \geq r(x)$

$\leq 1 + 2(r'(x) - r(x))$



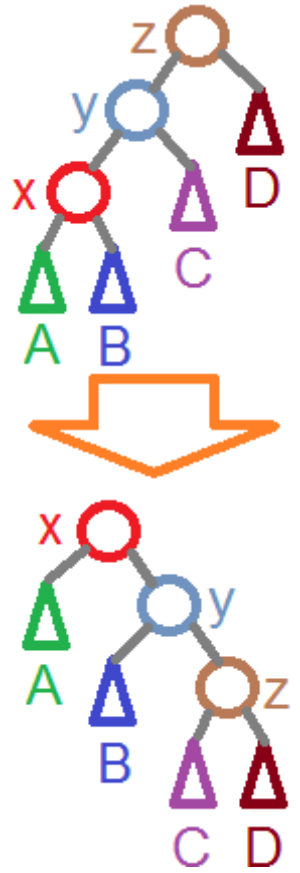
Splay: анализ

$T'' = 3(r'(x) - r(x))$

$T'' = T + \Phi' - \Phi = 1 \sout{+ r'(x)} + r'(y) + r'(z) - r(x) - r(y) \sout{- r(z)} \leq$

- $r'(y), r'(z) \leq r'(x)$
- $r(y), r(z) \geq r(x)$

$\leq 1 + 2(r'(x) - r(x)) \leq 1 + 3(r'(x) - r(x))$



Splay: анализ

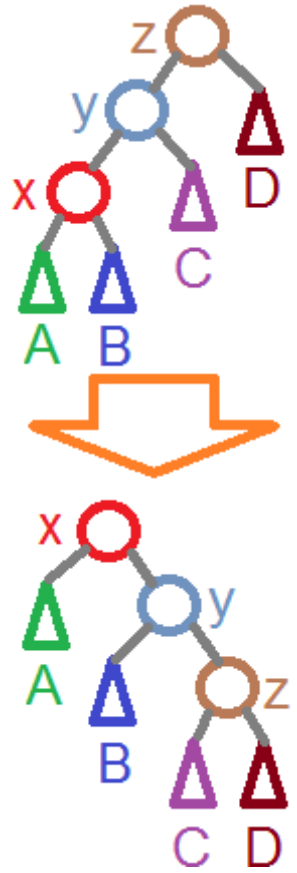
$T'' = 3(r'(x) - r(x))$

$T'' = T + \Phi' - \Phi = 1 \sout{+ r'(x)} + r'(y) + r'(z) - r(x) - r(y) \sout{- r(z)} \leq$

- $r'(y), r'(z) \leq r'(x)$
- $r(y), r(z) \geq r(x)$

$\leq 1 + 2(r'(x) - r(x)) \leq 1 + 3(r'(x) - r(x))$

- Покажем $T'' = 3(r'(x) - r(x))$



Splay: анализ

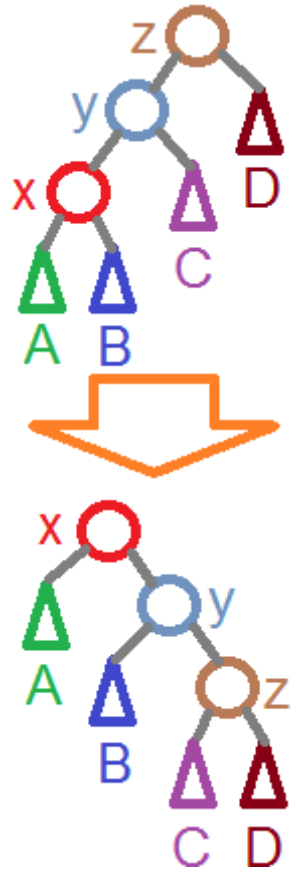
$T'' = 3(r'(x) - r(x))$

$T'' = T + \Phi' - \Phi = 1 \sout{+ r'(x)} + r'(y) + r'(z) - r(x) - r(y) \sout{- r(z)} \leq$

- $r'(y), r'(z) \leq r'(x)$
- $r(y), r(z) \geq r(x)$

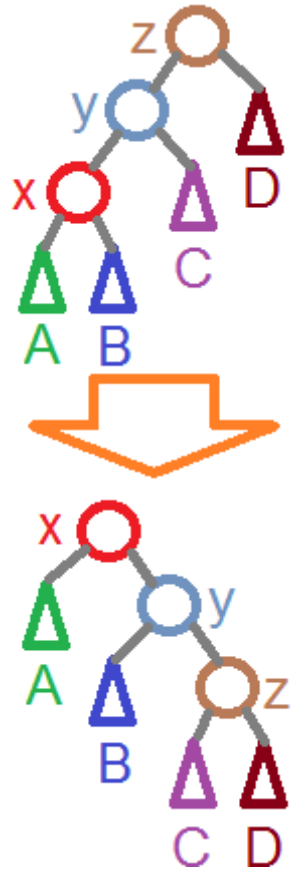
$\leq 1 + 2(r'(x) - r(x)) \leq 1 + 3(r'(x) - r(x))$

- Покажем $T'' = 3(r'(x) - r(x))$
  - Нужно доказать, что $T'' < 1 + 3(r'(x) - r(x))$



Splay: анализ

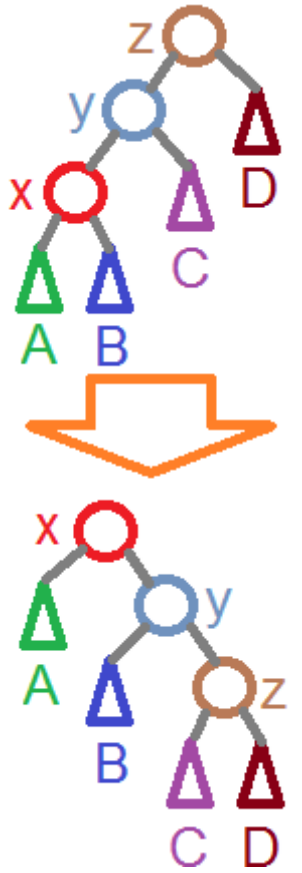
$T'' < 1 + 3(r'(x) - r(x))$?



Splay: анализ

T” < 1 + 3(r’(x) – r(x)) ?

Предположим T” = 1 + 3(r’(x) – r(x))

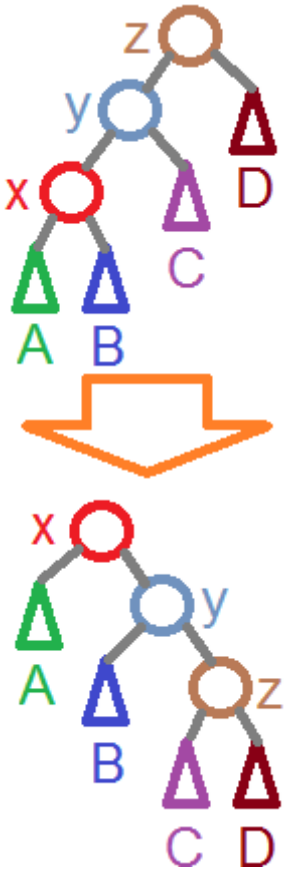


Splay: анализ

$T'' < 1 + 3(r'(x) - r(x))$?

Предположим $T'' = 1 + 3(r'(x) - r(x))$

Тогда $r'(y) + r'(z) - r(x) - r(y) = 3(r'(x) - r(x))$



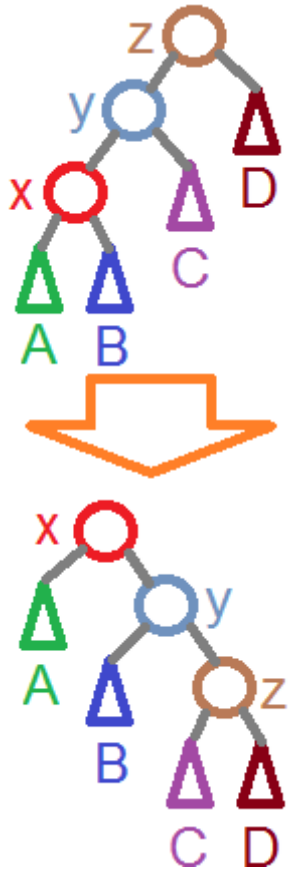
Splay: анализ

$T'' < 1 + 3(r'(x) - r(x))$?

Предположим $T'' = 1 + 3(r'(x) - r(x))$

Тогда $r'(y) + r'(z) - r(x) - r(y) = 3(r'(x) - r(x))$

- Это возможно если
  - $r'(z) = r'(y) = r'(x)$



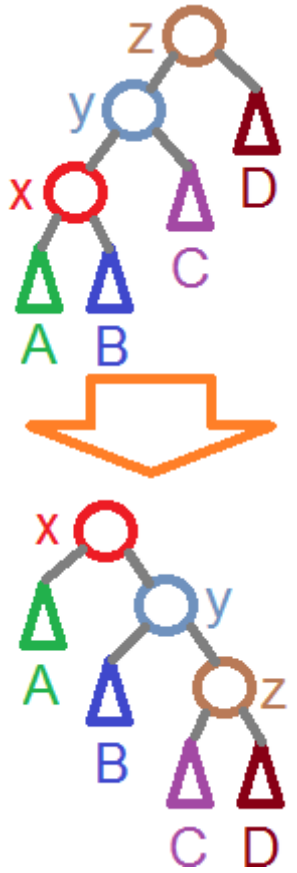
Splay: анализ

$T'' < 1 + 3(r'(x) - r(x))$?

Предположим $T'' = 1 + 3(r'(x) - r(x))$

Тогда $r'(y) + r'(z) - r(x) - r(y) = 3(r'(x) - r(x))$

- Это возможно если
  - $r'(z) = r'(y) = r'(x)$
  - $r(z) = r(y) = r(x)$



Splay: анализ

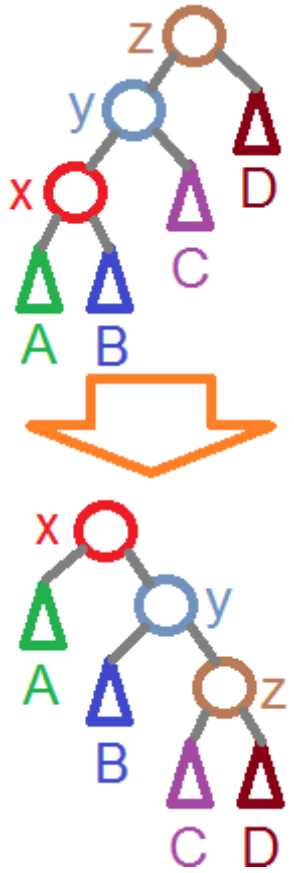
$T'' < 1 + 3(r'(x) - r(x))$?

Предположим $T'' = 1 + 3(r'(x) - r(x))$

Тогда $r'(y) + r'(z) - r(x) - r(y) = 3(r'(x) - r(x))$

- Это возможно если
  - $r'(z) = r'(y) = r'(x)$
  - $r(z) = r(y) = r(x)$

  и
  - $r'(x) = r(x)$



Splay: анализ

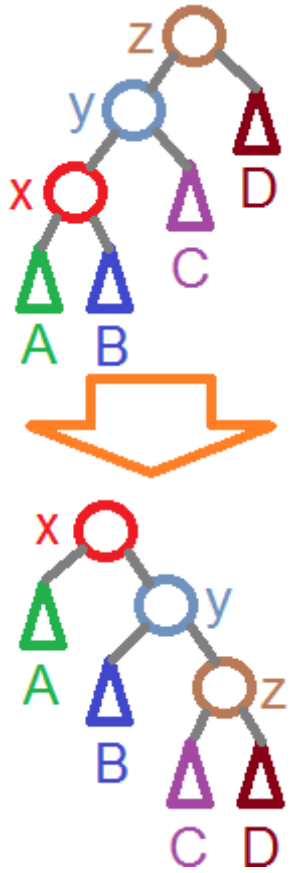
$T'' < 1 + 3(r'(x) - r(x))$?

Предположим $T'' = 1 + 3(r'(x) - r(x))$

Тогда $r'(y) + r'(z) - r(x) - r(y) = 3(r'(x) - r(x))$

- Это возможно если
  - $r'(z) = r'(y) = r'(x)$
  - $r(z) = r(y) = r(x)$

  и
  - $r'(x) = r(x)$
- То есть все эти ранги равны

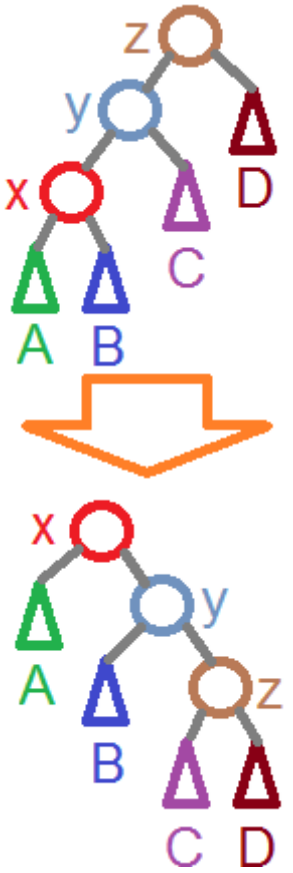


Splay: анализ

$T'' < 1 + 3(r'(x) - r(x))$?

Предположим $T'' = 1 + 3(r'(x) - r(x))$

Тогда все ранги x, y, z равны



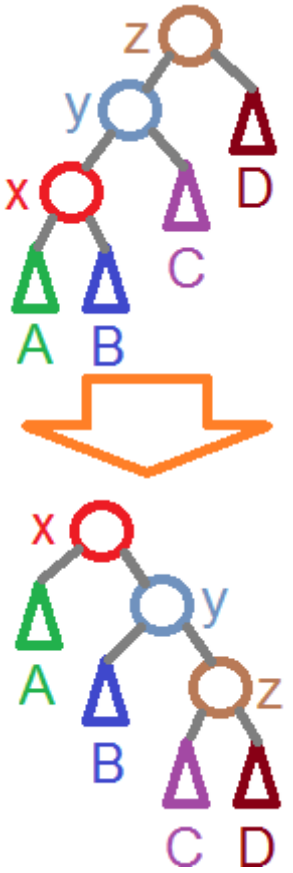
Splay: анализ

T” < 1 + 3(r’(x) – r(x)) ?

Предположим T” = 1 + 3(r’(x) – r(x))

Тогда все ранги x, y, z равны

- w’(x) = 3 + w(A) + w(B) + w(C) + w(D)



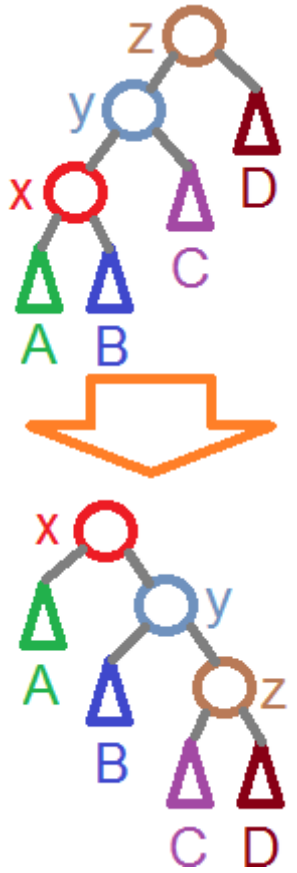
Splay: анализ

T” < 1 + 3(r’(x) – r(x)) ?

Предположим T” = 1 + 3(r’(x) – r(x))

Тогда все ранги x, y, z равны

- w’(x) = 3 + w(A) + w(B) + w(C) + w(D)
- w’(z) = 1 + w(C) + w(D)



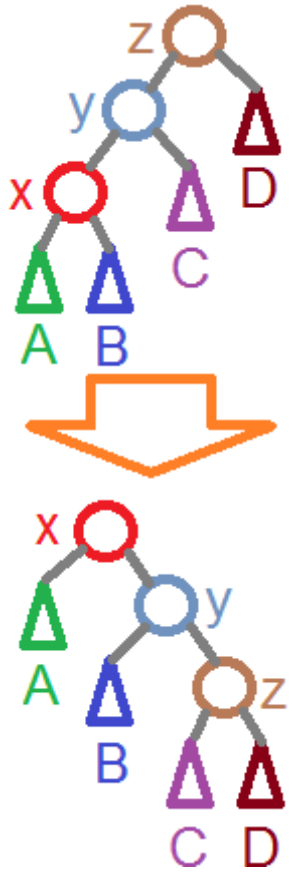
Splay: анализ

T” < 1 + 3(r’(x) – r(x)) ?

Предположим T” = 1 + 3(r’(x) – r(x))

Тогда все ранги x, y, z равны

- w’(x) = 3 + w(A) + w(B) + w(C) + w(D)
- w’(z) = 1 + w(C) + w(D)
- w(x) = 1 + w(A) + w(B)



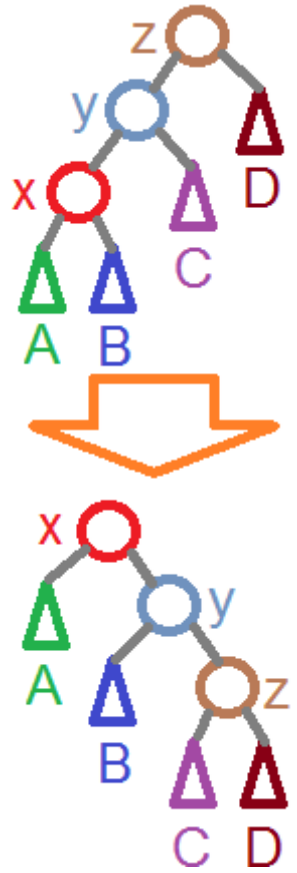
Splay: анализ

$T'' < 1 + 3(r'(x) - r(x))$?

Предположим $T'' = 1 + 3(r'(x) - r(x))$

Тогда все ранги x, y, z равны

- $w'(x) = 3 + w(A) + w(B) + w(C) + w(D)$
- $w'(z) = 1 + w(C) + w(D)$
- $w(x) = 1 + w(A) + w(B)$
- $w'(x) = 1 + w'(z) + w(x)$



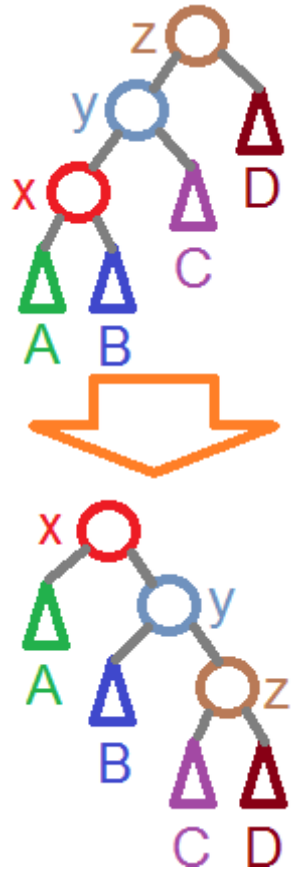
Splay: анализ

T” < 1 + 3(r’(x) – r(x)) ?

Предположим T” = 1 + 3(r’(x) – r(x))

Тогда все ранги x, y, z равны

- w’(x) = 3 + w(A) + w(B) + w(C) + w(D)
- w’(z) = 1 + w(C) + w(D)
- w(x) = 1 + w(A) + w(B)
- w’(x) = 1 + w’(z) + w(x)
  - 2^(r+1) > w’(x), w’(z), w(x) ≥ 2^r



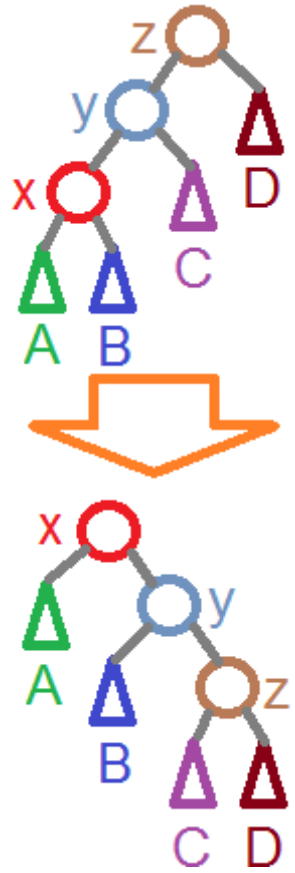
Splay: анализ

T” < 1 + 3(r’(x) – r(x)) ?

Предположим T” = 1 + 3(r’(x) – r(x))

Тогда все ранги x, y, z равны

- w’(x) = 3 + w(A) + w(B) + w(C) + w(D)
- w’(z) = 1 + w(C) + w(D)
- w(x) = 1 + w(A) + w(B)
- w’(x) = 1 + w’(z) + w(x)
  - 2^(r+1) > w’(x), w’(z), w(x) ≥ 2^r
- Противоречие



Splay: анализ

Амортизационный анализ

- $T'' = T + \Phi' - \Phi$

3. $T'' = 3(r'(x) - r(x))$

Splay: анализ

Амортизационный анализ

- T” = T + Φ’ – Φ

3. T” = 3(r’(x) – r(x))

- Остаётся в качестве упражнения

Splay-дерево: сложность методов

$T''_{splay\_find} = O(logN)$

Splay-дерево: сложность методов

$T''_{splay\_find} = O(\log N)$

$T''_{splay\_erase} = O(\log N)$

Splay-дерево: сложность методов

$T''_{splay\_find} = O(\log N)$

$T''_{splay\_erase} = O(\log N)$ – ранг только уменьшится

Splay-дерево: сложность методов

$T''_{splay\_find} = O(logN)$

$T''_{splay\_erase} = O(logN)$ – ранг только уменьшится

$T''_{splay\_insert} = O(logN)$?

Splay-дерево: сложность методов

$T''_{splay\_find} = O(logN)$

$T''_{splay\_erase} = O(logN)$ – ранг только уменьшится

$T''_{splay\_insert} = O(logN)$ – только log(N) рангов вырастут на единицу