

Лекция 7

Декартовы деревья (treaps)

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com

WWW: www.mkurnosov.net

Курс «Структуры и алгоритмы обработки данных»

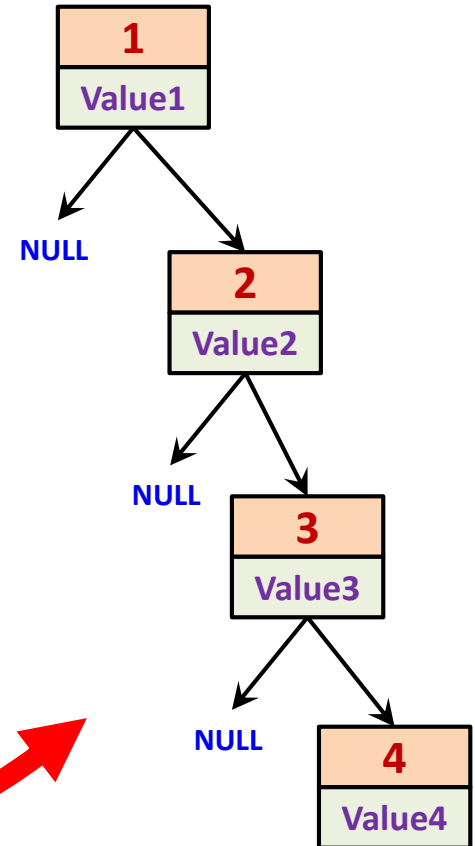
Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)

Осенний семестр, 2015

Двоичные деревья поиска (Binary Search Trees)

1. Операции над BST имеют трудоемкость пропорциональную высоте дерева $O(h)$
2. В среднем случае высота h бинарного дерева поиска $O(\log n)$
3. В худшем случае элементы добавляются по возрастанию (убыванию) ключей – дерево вырождается в список длины n

```
bstree_add(1, value1)  
bstree_add(2, value2)  
bstree_add(3, value3)  
bstree_add(4, value4)
```



В худшем случае бинарное дерево поиска
имеет высоту $O(n)$

Декартово дерево (Treap)

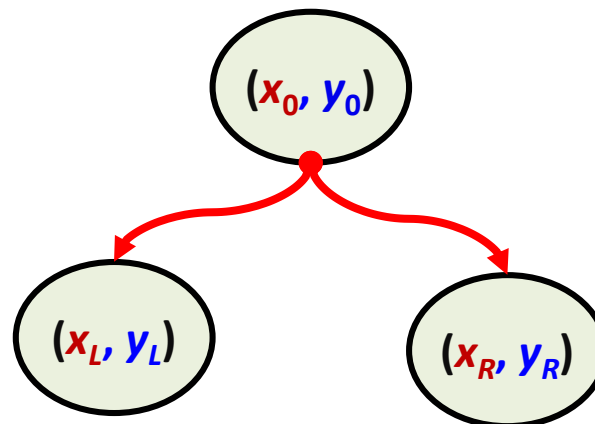
- **Декартово дерево (Treap)** – это структура данных для реализации словаря, объединяющая в себе бинарное дерево поиска и бинарную кучу
- **Происхождение названия:** *treap* = tree + hearp
- Русскоязычный аналог названия:
дуча = дерево + куча, *дерамида* = дерево + пирамида
- **Авторы:** Сидель Раймунд (Seidel Raimund),
Цецилия Арагон (Aragon Cecilia), 1989
- Aragon Cecilia R., Seidel Raimund. **Randomized Search Trees** // Proc. of 30th Symp. Foundations of Computer Science (IEEE Computer Society Press), 1989, pp. 540-545
- Seidel Raimund, Aragon Cecilia R. **Randomized Search Trees** // Algorithmica. – 1996. – 16 (4/5). – pp. 464–497

Декартово дерево (Treap)

- Каждый узел декартова дерева (treap) содержит:
 - ❑ пару (x, y)
 x – ключ бинарного дерева поиска,
 y – приоритет бинарной кучи
 - ❑ указатель *left* на левое поддерево
 - ❑ указатель *right* на правое поддерево
- Предполагается, что все x и y различны

(1) Свойства
дерева поиска:

$$x_L < x_0 < x_R$$

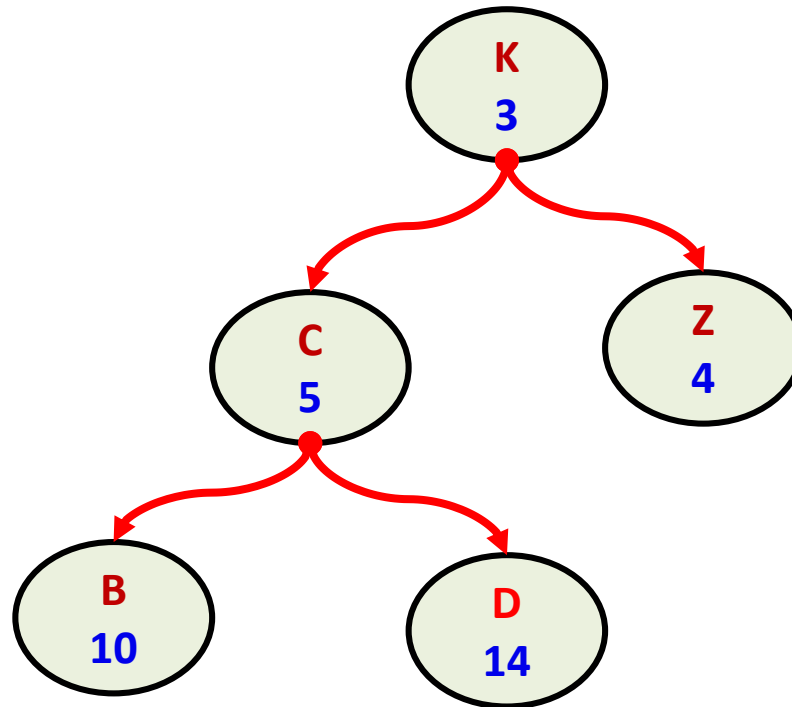


(2) Свойства
бинарной кучи:

$$y_0 < y_L$$

$$y_0 < y_R$$

Пример декартова дерева



Пример декартова дерева:
ключи дерева поиска – буквы алфавита,
приоритет кучи (min-heap) – числа

Операции декартова дерева

Операция	Средний случай (average case)	Худший случай (worst case)
<i>Lookup</i> (x)	$O(\log n)$	Amortized $O(\log n)$
<i>Insert</i> ($x, value$)	$O(\log n)$	Amortized $O(\log n)$
<i>Delete</i> (x)	$O(\log n)$	Amortized $O(\log n)$
<i>Split</i> (x)	$O(\log n)$	$O(n)$
<i>Merge</i> (T_1, T_2)	$O(\log n)$	$O(n)$

- Сложность по памяти $O(n)$

Поиск элемента в декартовом дереве

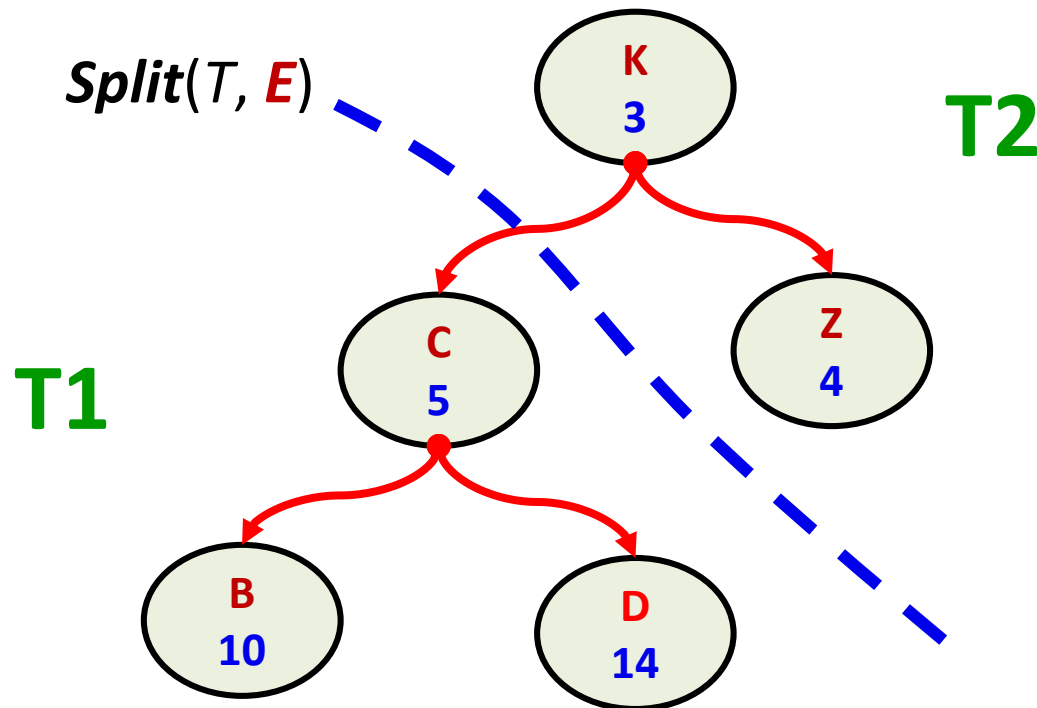
```
function TreapLookup(treap, x)
  while treap != NULL do
    if x = treap.x then
      return treap;
    else if x < treap.x then
      treap = treap.left
    else
      treap = treap.right
    end if
  end while
  return NULL;
end function
```

$$T_{Lookup} = O(h) = O(n)$$

- При поиске элемента приоритет y игнорируется, учитываются только ключи x – как в обычном бинарном дереве поиска
- В худшем случае спуск по дереву осуществляется до листа – выполняется порядка $O(h)$ сравнений

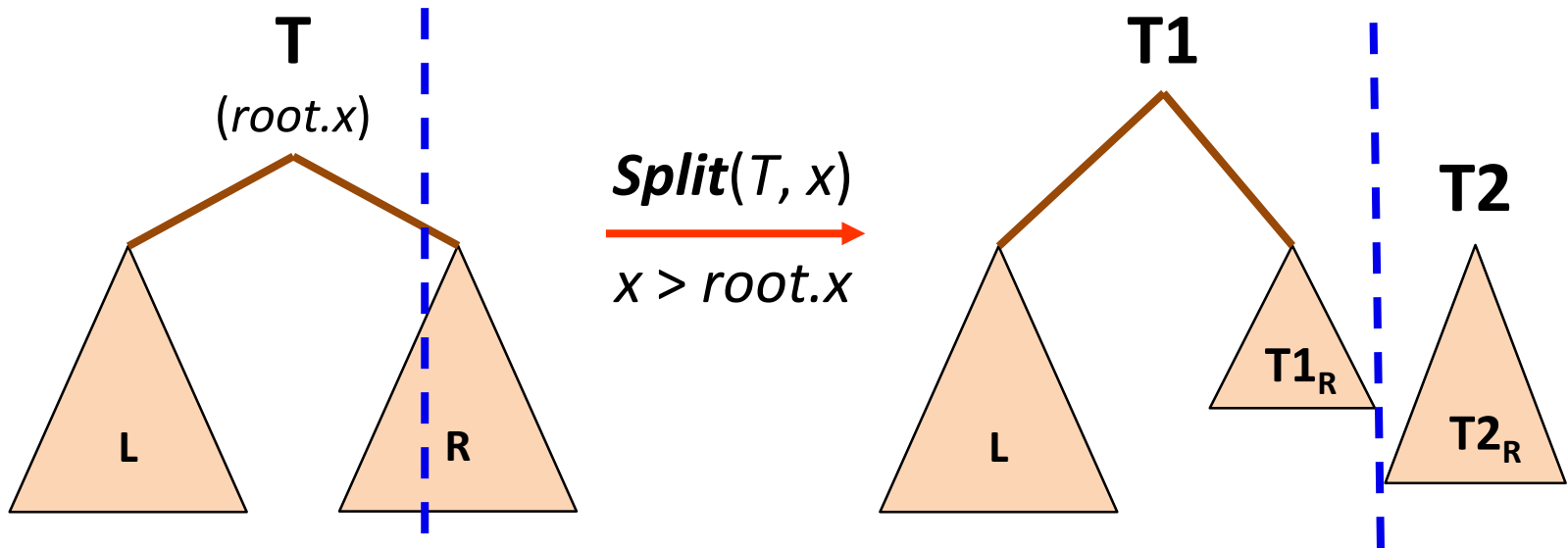
Разбиение декартова дерева (Split)

- Операция ***Split***(T, x) разбивает декартово дерево на два дерева $T1$ и $T2$
- В дереве $T1$ находятся узлы с ключами $node.x \leq x$
- В дереве $T2$ находятся узлы с ключами $node.x > x$



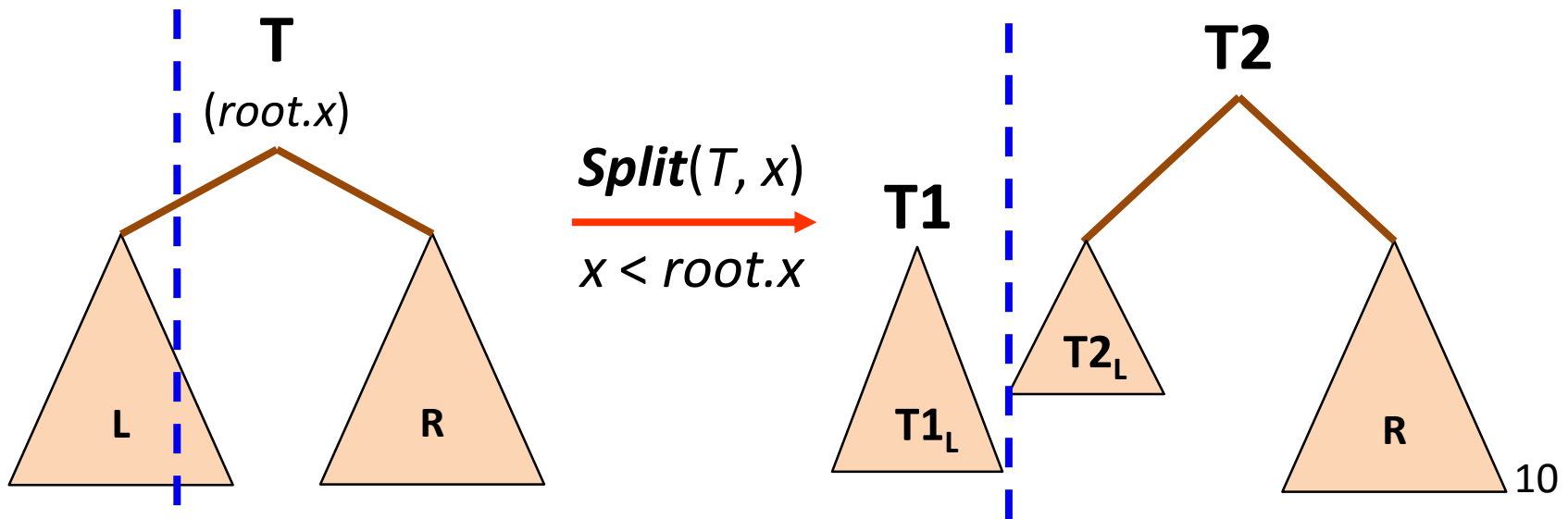
Разбиение декартова дерева (Split)

- **Случай 1:** ключ разбиения $x > root.x$ (ключ в корне)
- Левое поддерево $T1$ совпадает с левым поддеревом корня T
- Для нахождения правого поддерева $T1$ необходимо разбить правое поддерево T по ключу x на $T1_R$ и $T2_R$ и взять $T1_R$
- Дерево $T2$ совпадает с деревом $T2_R$



Разбиение декартова дерева (Split)

- **Случай 2** (симметричный): ключ разбиения $x < \text{root}.x$
- Правое поддерево T_2 совпадает с правым поддеревом корня T
- Для нахождения левого поддерева T_2 необходимо разбить левое поддерево T по ключу x на $T1_L$ и $T2_L$ и взять $T2_L$
- Дерево T_1 совпадает с деревом $T1_L$



Разбиение декартова дерева (Split)

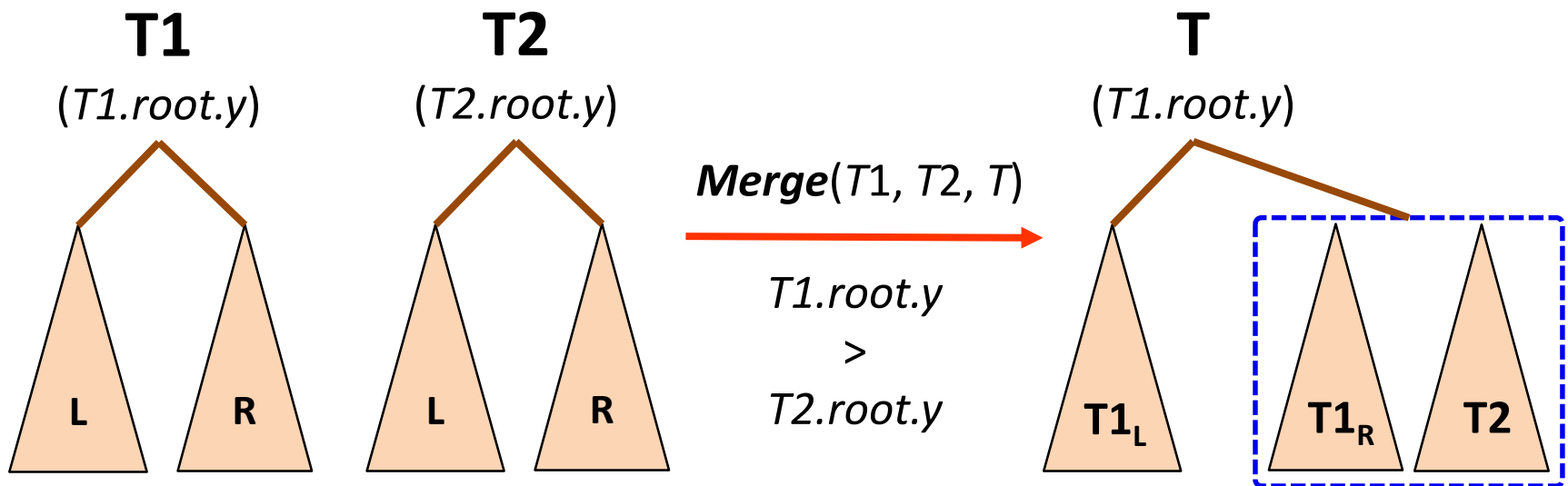
```
function TreapSplit(treap, x, &treap1, &treap2)
    if treap = NULL then
        treap1 = treap2 = NULL
    else if x > treap.x then
        TreapSplit(treap.right, x, treap.right, treap2)
        treap1 = treap
    else
        TreapSplit(treap.left, x, treap1, treap.left)
        treap2 = treap
    end if
end function
```

$$T_{Split} = O(h) = O(n)$$

- В худшем случае требуется спуск по дереву до листа – выполняется порядка $O(h)$ разбиений

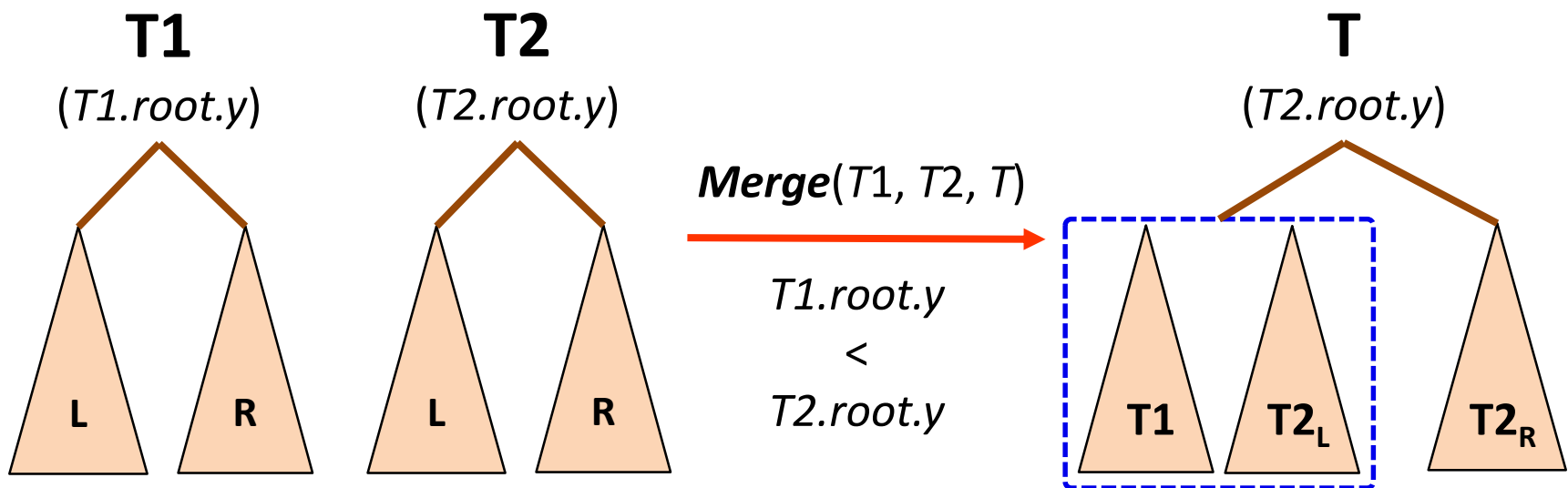
Слияние декартовых деревьев (Merge)

- Операция **Merge**(T_1, T_2) сливает два декартовых дерева T_1 и T_2 в новое дерево T (причем все ключи x дерева T_2 должны быть больше ключей x дерева T_1)
- Случай 1:** Приоритет у корня $T_1 >$ приоритета у корня T_2
- Корень дерева T_1 становится корнем T
- Левое поддерево T_1 становится левым поддеревом T
- Правое поддерево T – это объединение правого поддерева T_1 и дерева T_2



Слияние декартовых деревьев (Merge)

- Операция **Merge**(T_1, T_2) сливает два декартовых дерева T_1 и T_2 в новое дерево T (причем все ключи x дерева T_2 должны быть больше ключей x дерева T_1)
- Случай 2** (симметричный): Приоритет у корня $T_1 <$ приоритета у корня T_2
- Корень дерева T_2 становится корнем T
- Правое поддерево T_2 становится правым поддеревом T
- Левое поддерево T – это объединение T_1 и левого поддерева T_2



Слияние декартовых деревьев (Merge)

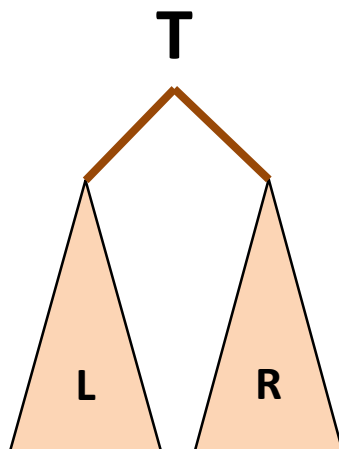
```
function TreapMerge(treap1, treap2, &treap)
  if treap1 = NULL OR treap2 = NULL then
    treap = treap1
    if treap1 = NULL then
      treap = treap2
  else if treap1.y > treap2.y then
    TreapMerge(treap1.right, treap2, treap1.right)
    treap = treap1
  else
    TreapMerge(treap2.left, treap1, treap2.left)
    treap = treap2
  end if
end function
```

$$T_{Merge} = O(h) = O(n)$$

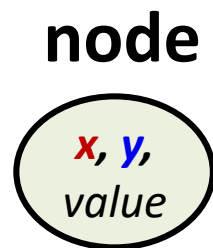
- В худшем случае требуется спуск по дереву до листа – выполняется порядка $O(h)$ разбиений

Добавление элемента в декартово дерево

- Операция ***Insert***($T, x, y, value$) добавляет в декартово дерево узел *node* с ключом x , приоритетом y и значением $value$
- Узел *node* – это декартово дерево из одного элемента
- Можно слить дерево T и дерево *node*, но по требованию операции *Merge*, все ключи дерева *node* должны быть больше ключей T
- В дереве T некоторые ключи могут быть как больше x , так и меньше x



$$\forall x_T \in T: \quad x_T < \text{node}.x$$



Добавление элемента в декартово дерево

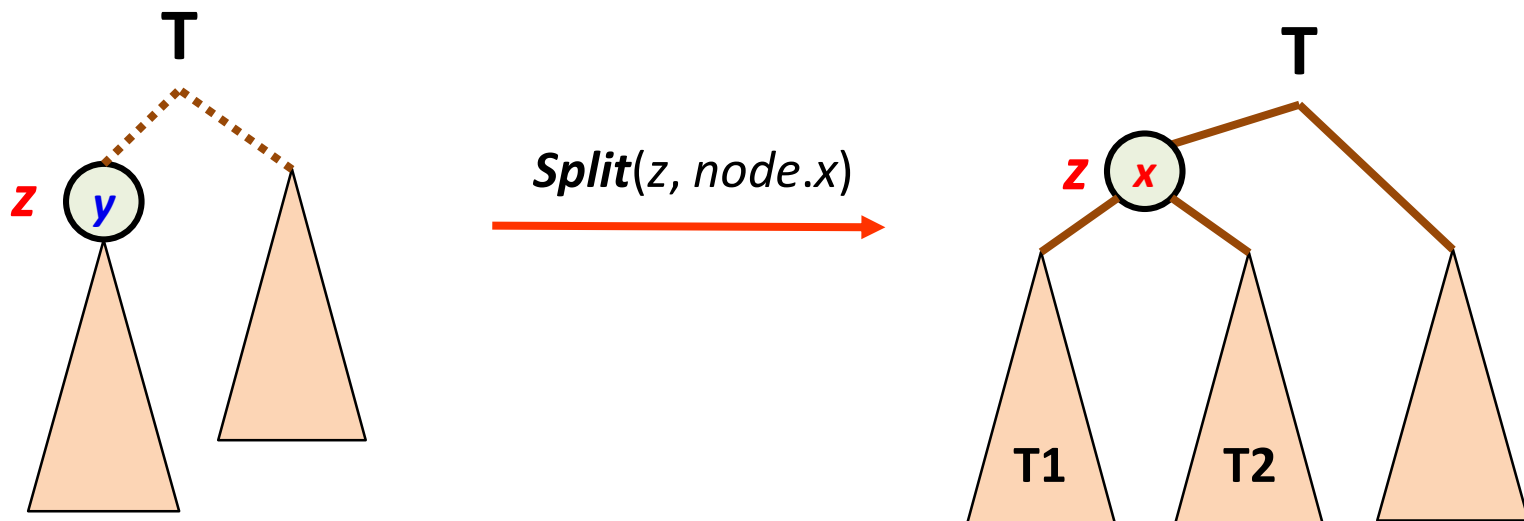
- **Версия 1.** Преобразуем деревья для удовлетворения требований операции *Merge*
- **Разобьём дерево T** по ключу $node.x$ на два поддерева $T1$ и $T2$, таким образом в дереве $T1$ будут узлы с $x \leq node.x$, а в дереве $T2$ с ключами $x > node.x$
- **Сливаем** дерево $T1$ и дерево $node$ в дерево $T1$
- **Сливаем** новое дерево $T1$ и дерево $T2$ в дерево $T1$

```
function TreapInsert(treap, node)
    TreapSplit(treap, node.x, t1, t2)           //  $O(h)$ 
    TreapMerge(t1, node, t1)                   //  $O(h)$ 
    TreapMerge(t1, t2, t1)                     //  $O(h)$ 
    return t1
end function
```

$$T_{Insert} = O(h) = O(n)$$

Добавление элемента в декартово дерево

- **Версия 2 – без использования операции слияния (Merge)**
- Спускаемся по дереву поиска (по ключу $node.x$), останавливаемся на узле z , в котором приоритет $z.y$ меньше приоритета $node.y$
- Разбиваем дерево z по ключу $node.x$: в поддереве $T1$ узлы z с ключами $\leq node.x$, в $T2$ – с ключами $> node.x$
- Делаем $T1$ и $T2$ левым и правым поддеревьями узла $node$
- Дерево $node$ ставим на место узла z



Удаление элемента из декартова дерева

- Версия 1 – с использованием операции **Split**
- Находим в дереве T узел $node$ с ключом x
- Разбиваем дерево T по ключу x на деревья $T1$ и $T2$: $Split(T, x)$
- Отделяем от дерева $T1$ узел с ключом x , для этого разбиваем $T1$ по ключу $x - \epsilon$ на поддеревья $T1$ и $T3$ (содержит ключ x): $Split(T1, x - \epsilon)$
- Сливаем деревья $T1$ и $T2$: $Merge(T1, T2)$

```
function TreapDelete(treap, x)
    node = TreapLookup(treap, x)           //  $O(h)$ 
    TreapSplit(treap, x, t1, t2)           //  $O(h)$ 
    TreapSplit(t1, x - eps, t1, t3)        //  $O(h)$ 
    TreapMerge(t1, t2, t1)                 //  $O(h)$ 
    return t1
end function
```

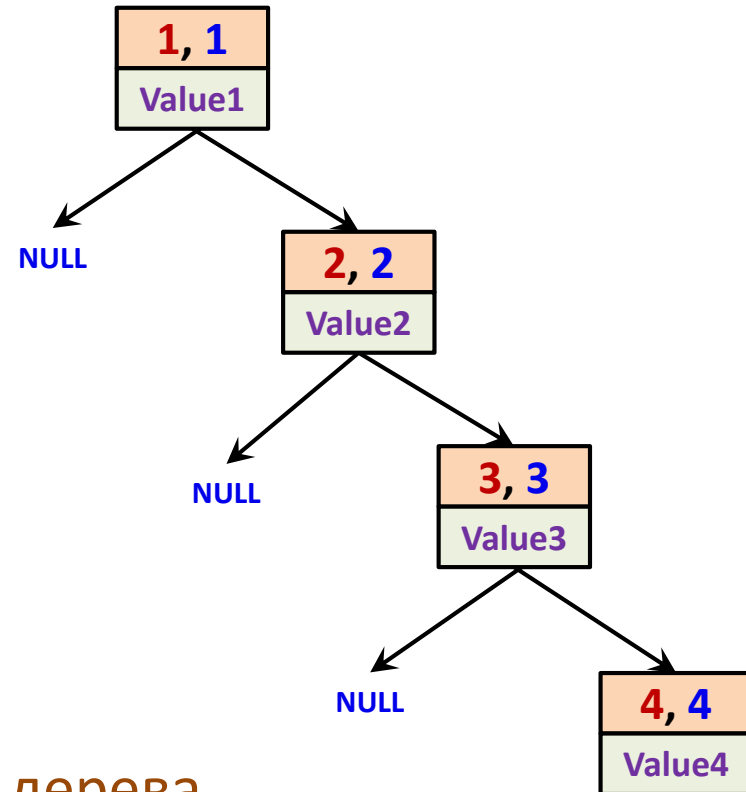
$$T_{Delete} = O(h) = O(n)$$

Удаление элемента из декартова дерева

- Версия 2 – без использования операции Split
- Находим в дереве T узел $node$ с ключом x
- Сливаем левое и правое поддеревья дерева $node$:
 $Merge(node.left, node.right)$
- Результат слияния поддеревьев ставим на место узла $node$

Высота декартова дерева

- В худшем случае высота декартова дерева $O(n)$
- Например, при вставке ключей: $(1, 1), (2, 2), \dots, (n, n)$



- Как избежать такой ситуации?
- Мы можем задавать приоритеты для регулирования высоты дерева

Псевдослучайные приоритеты

Утверждение. В декартовом дереве из n узлов, приоритет у которых является случайной величиной с равномерным распределением, средняя глубина вершины равна $O(\log n)$

Доказательство

- Разобрать самостоятельно:

http://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE



- Таким образом в декартовом дереве с псевдослучайными приоритетами время выполнения операций Split, Merge, Insert, Delete равно $O(\log n)$

Задания

- Разобрать доказательство утверждения о высоте декартова дерева со случайными приоритетами
- Изучить способы построения декартова дерева при условии, что все добавляемые элементы известны заранее (offline):
 $(x_1, y_1), \dots, (x_n, y_n)$
 - ☐ Алгоритм со сложностью $O(n^2)$
 - ☐ Алгоритм с линейной сложностью $O(n)$
- 1. <http://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>
- Прочитать про неявные декартовы деревья
<http://e-maxx.ru/algo/treap>