Лекция 5.
Декартовы деревья

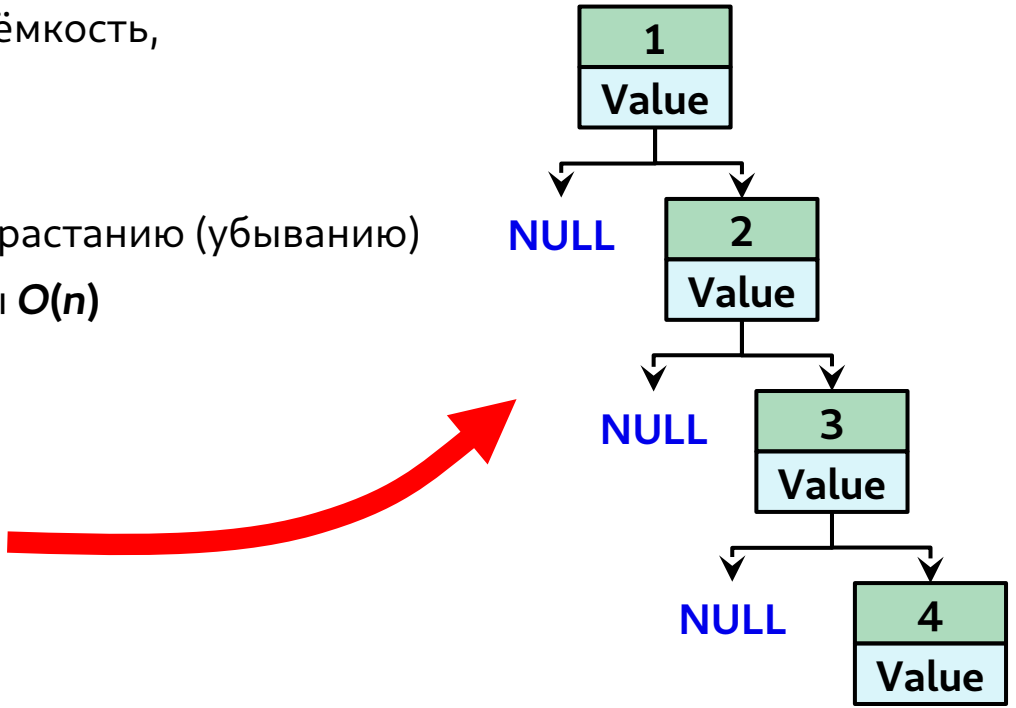
Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Осенний семестр, 2021 г.

Двоичные деревья поиска

- Операции над двоичным деревом имеют трудоёмкость, пропорциональную высоте ***h*** дерева
- В среднем случае высота дерева ***O*(log*n*)**
- В худшем случае элементы добавляются по возрастанию (убыванию) ключей — дерево вырождается в список длины ***O*(*n*)**

bstree_add(1, value)
bstree_add(2, value)
bstree_add(3, value)
bstree_add(4, value)



Декартово дерево (*treap*)

- **Декартово дерево**[1,2] (*treap*) — это структура данных для реализации словаря, объединяющая в себе бинарное дерево поиска и бинарную кучу
- **Происхождение названия: *treap*** = tree + heap
- Русскоязычный аналог названия: ***дуча*** = дерево + куча, ***дерамида*** = дерево + пирамида

- **Авторы:** Raimund G. Seidel, Cecilia R. Aragon, 1989
  Aragon C. R., Seidel R. G. **Randomized search trees** // Foundations of Computer Science, 1989., 30th Annual Symposium on. — IEEE, 1989. — С. 540-545.
  Seidel R., Aragon C. R. **Randomized search trees** // Algorithmica. — 1996. — Т. 16. — №. 4-5. — С. 464-497.

Декартово дерево (*treap*)

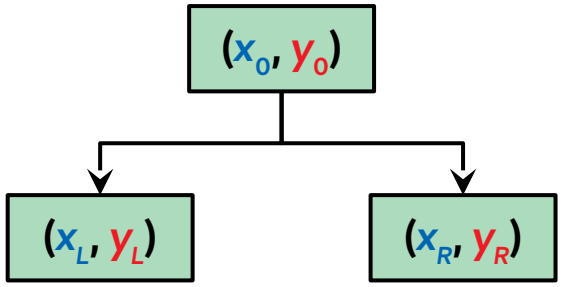
Каждый узел декартова дерева (*treap*) содержит:

- **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 бинарного дерева:

$$x_L < x_0 < x_R$$

2. Свойства бинарной кучи (*max-heap*):

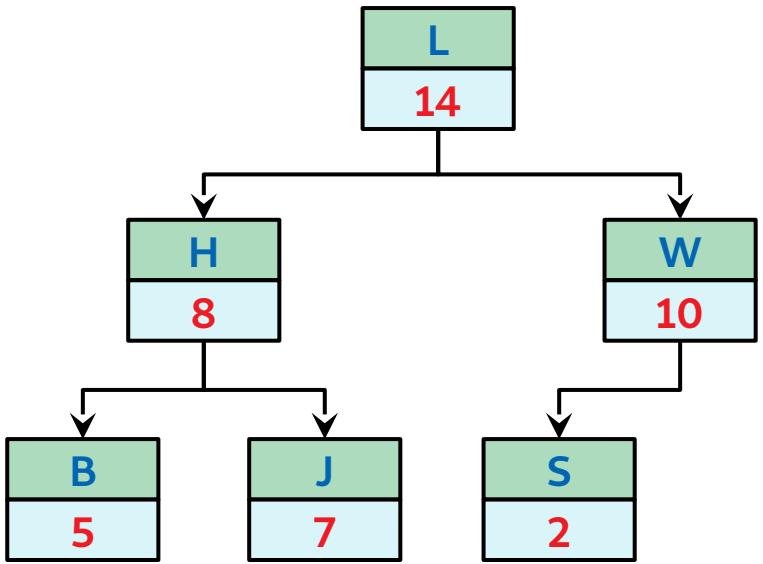
$$y_0 > y_L$$

$$y_0 > y_R$$

Декартово дерево (*treap*)

Пример декартова дерева:

ключи дерева поиска — буквы алфавита, **приоритет** кучи (*max-heap*) — числа



Декартово дерево (*treap*)

Операция	Средний случай (average case)	Худший случай (worst case)
Add(x, y, *value*)	$O(\log n)$	Amortized $O(\log n)$
Lookup(x)	$O(\log n)$	Amortized $O(\log n)$
Remove(x)	$O(\log n)$	Amortized $O(\log n)$
Split(x)	$O(\log n)$	$O(n)$
Merge(T_1, T_2)	$O(\log n)$	$O(n)$

Сложность по памяти (space complexity): $O(n)$

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
```

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

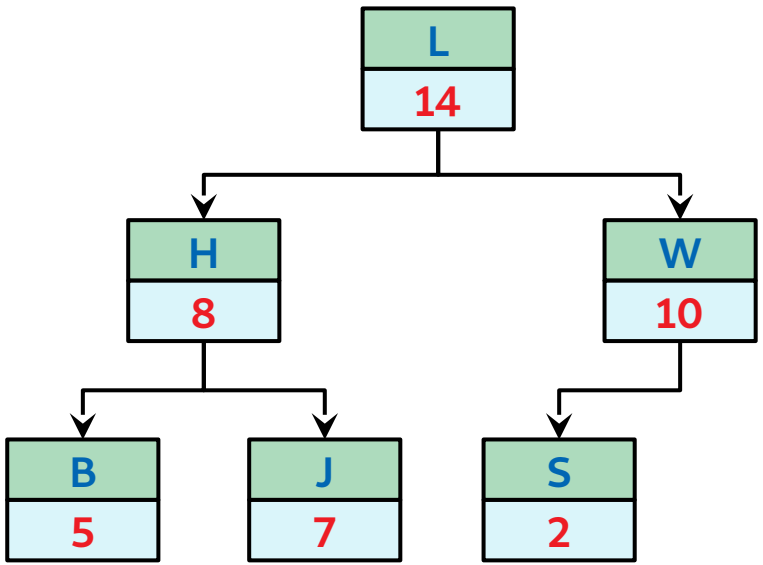
- При поиске элемента приоритет *y* игнорируется, учитываются только ключи *x* — как в обычном бинарном дереве поиска
- В худшем случае спуск по дереву осуществляется до листа — выполняется порядка $O(h)$ сравнений

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* ≤ *x*
- В дереве T2 находятся узлы с ключами *node.x* > *x*

***Split*(*T*, K)**

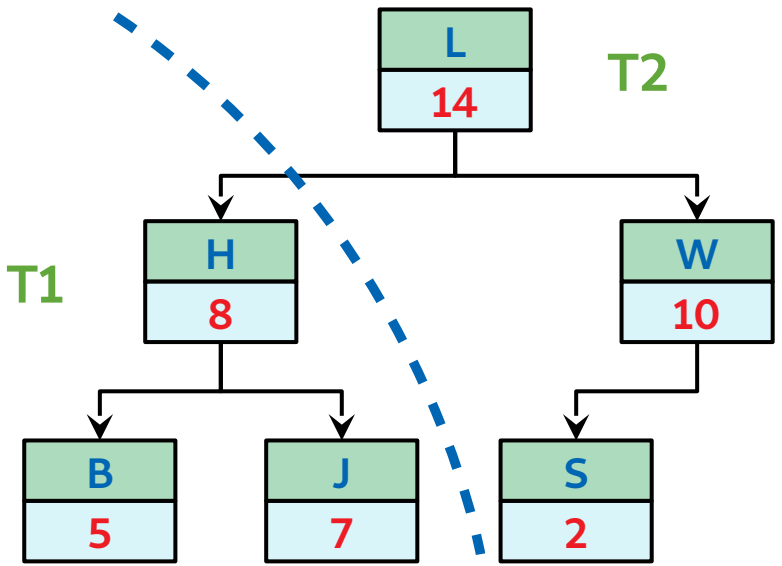


Разбиение декартова дерева (*Split*)

Операция ***Split*(*T*, *x*)** разбивает декартово дерево на два дерева T1 и T2

- В дереве T1 находятся узлы с ключами *node.x* ≤ *x*
- В дереве T2 находятся узлы с ключами *node.x* > *x*

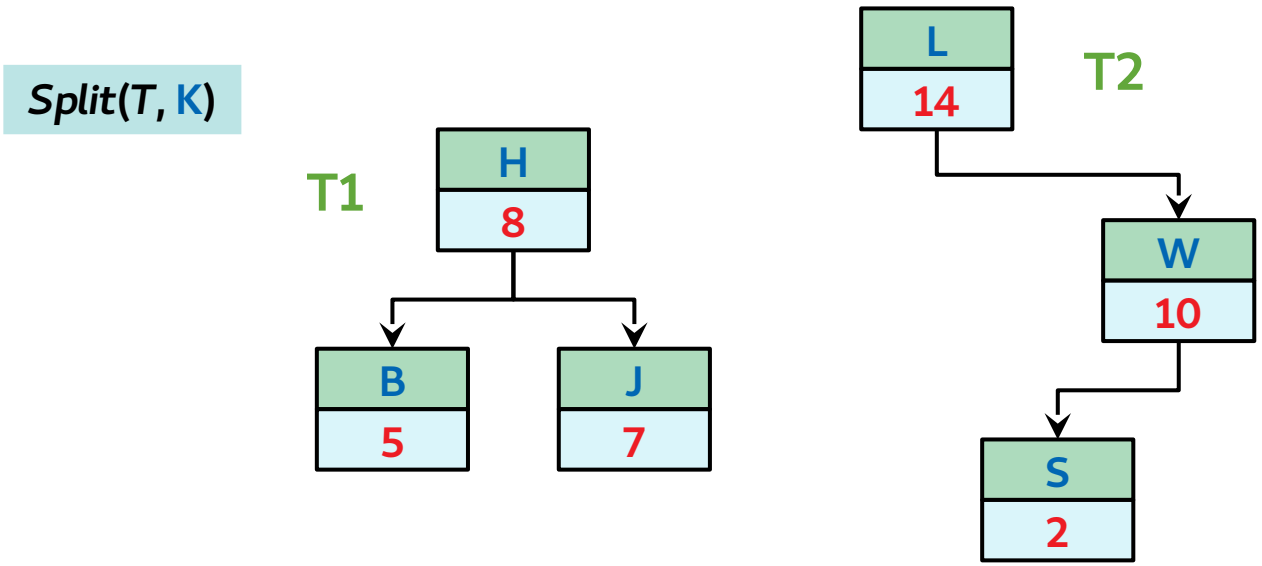
***Split*(*T*, K)**



Разбиение декартова дерева (*Split*)

Операция ***Split*(*T*, *x*)** разбивает декартово дерево на два дерева T1 и T2

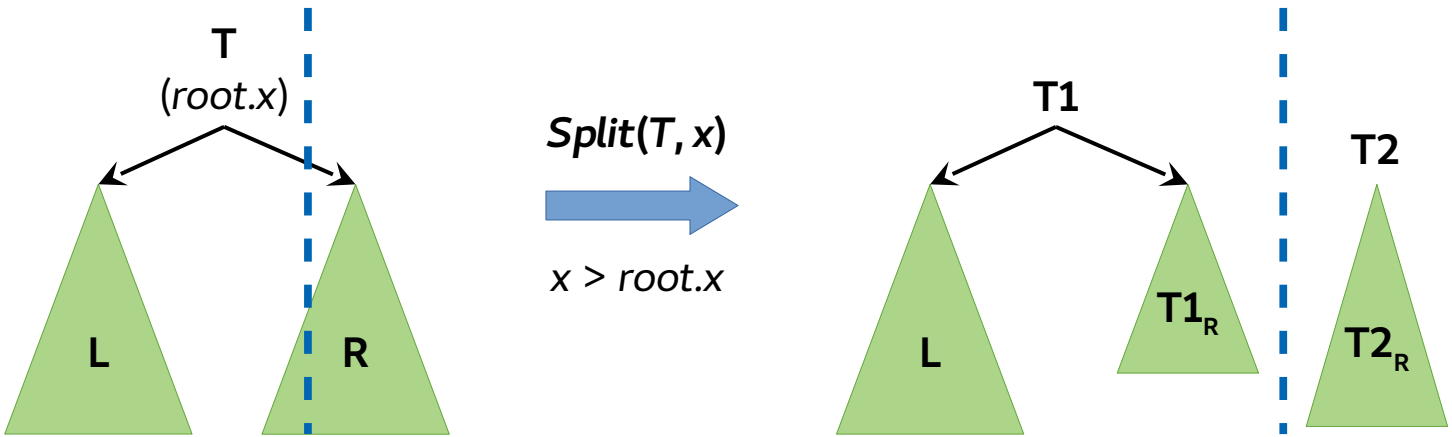
- В дереве T1 находятся узлы с ключами *node.x* ≤ *x*
- В дереве T2 находятся узлы с ключами *node.x* > *x*



Разбиение декартова дерева (*Split*)

Случай 1: ключ разбиения больше ключа в корне, $x > root.x$

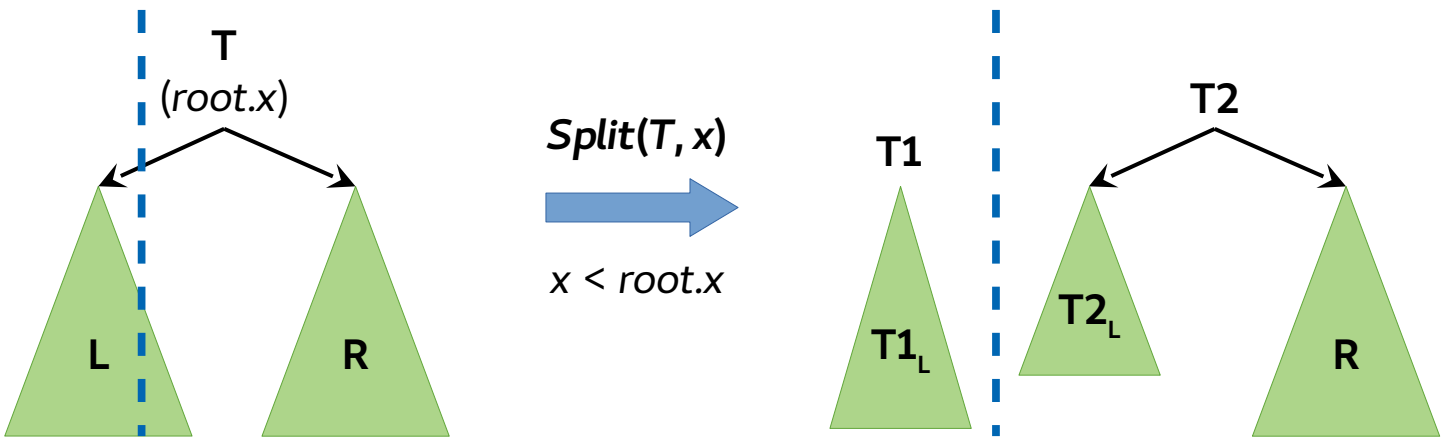
- Левое поддерево T1 совпадает с левым поддеревом корня T
- Для нахождения правого поддерева T1 необходимо разбить правое поддерево T по ключу x на $T1_R$ и $T2_R$ и взять $T1_R$
- Дерево T2 совпадает с деревом $T2_R$



Разбиение декартова дерева (*Split*)

Случай 2 (симметричный): ключ разбиения меньше ключа в корне, $x < root.x$

- Правое поддерево T2 совпадает с правым поддеревом корня T
- Для нахождения левого поддерева T2 необходимо разбить левое поддерево T по ключу x на $T1_L$ и $T2_L$ и взять $T2_L$
- Дерево T1 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
```

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

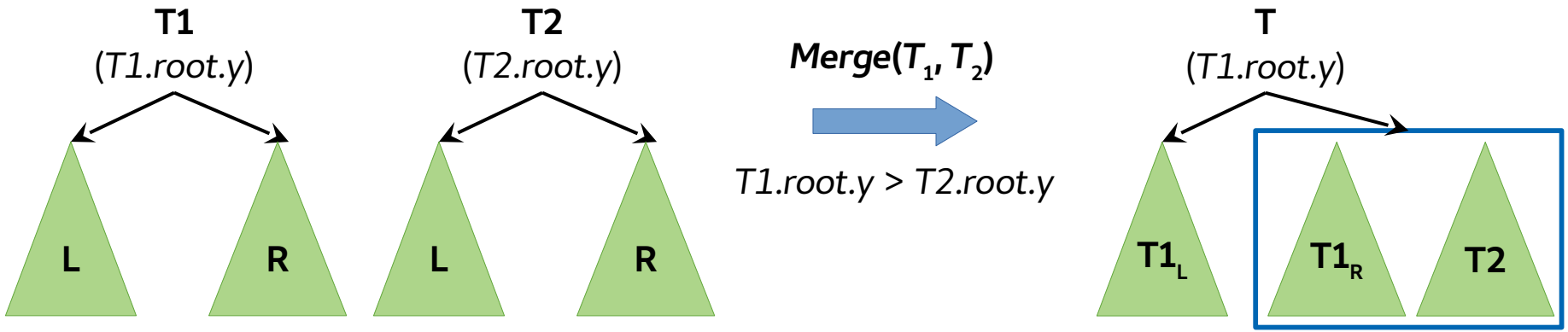
- В худшем случае требуется спуск по дереву до листа — выполняется порядка $O(h)$ разбиений

Слияние декартовых деревьев (*Merge*)

Операция ***Merge***(T_1, T_2) сливает два декартовых дерева T1 и T2 в новое дерево T (причём, все ключи *x* дерева T2 должны быть больше ключей *x* дерева T1)

Случай 1: Приоритет корня T1 больше приоритета корня T2, *T1.root.y* > *T2.root.y*

- Корень дерева T1 становится корнем T
- Левое поддерево T1 становится левым поддеревом T
- Правое поддерево T — это объединение правого поддерева T1 и дерева T2

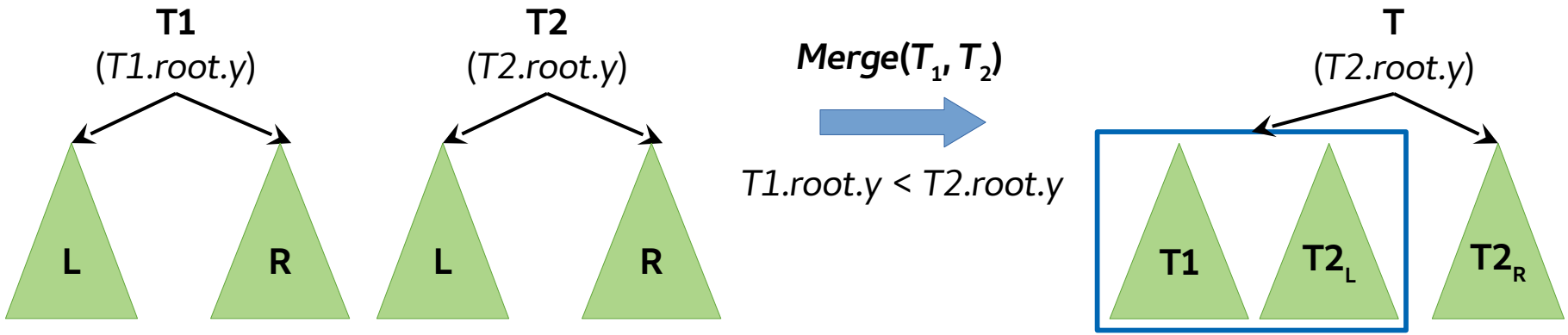


Слияние декартовых деревьев (*Merge*)

Операция ***Merge*(T_1, T_2)** сливает два декартовых дерева T1 и T2 в новое дерево T (причём, все ключи *x* дерева T2 должны быть больше ключей *x* дерева T1)

Случай 2 (симметричный): Приоритет корня T1 меньше приоритета корня T2, *T1.root.y* < *T2.root.y*

- Корень дерева T2 становится корнем T
- Правое поддерево T2 становится правым поддеревом T
- Левое поддерево T — это объединение левого поддерева T2 и дерева T1



Слияние декартовых деревьев (*Merge*)

```
function TreapMerge(treap1, treap2, &treap)
    if treap1 = NULL OR treap2 = NULL then
        treap = treap1
        if treap1 = NULL then
            treap = treap2
        end if
    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
```

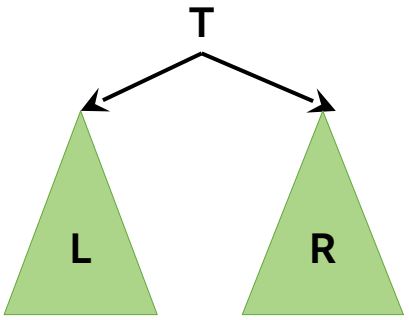
$T_{Merge} = O(h) = O(\log n)$

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

Добавление элемента в декартово дерево

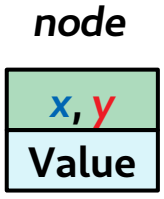
Операция ***Insert*(*T*, *x*, *y*, *value*)** добавляет в декартово дерево узел *node* с ключом *x*, приоритетом *y* и значением *value*

- 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*



?

$$\forall\, x_T \in T : x_T < node.x$$



Добавление элемента в декартово дерево

Вариант 1 — преобразование деревьев для удовлетворения требований операции *Merge*:

- Разобьём дерево T по ключу *node.x* на два поддерева T_1 и T_2, таким образом в дереве T_1 будут узлы с $x \le$ *node.x*, а в дереве T_2 с ключами $x >$ *node.x*
- Сливаем дерево T_1 и дерево *node* в дерево T_1
- Сливаем новое дерево T_1 и дерево T_2 в дерево T_1

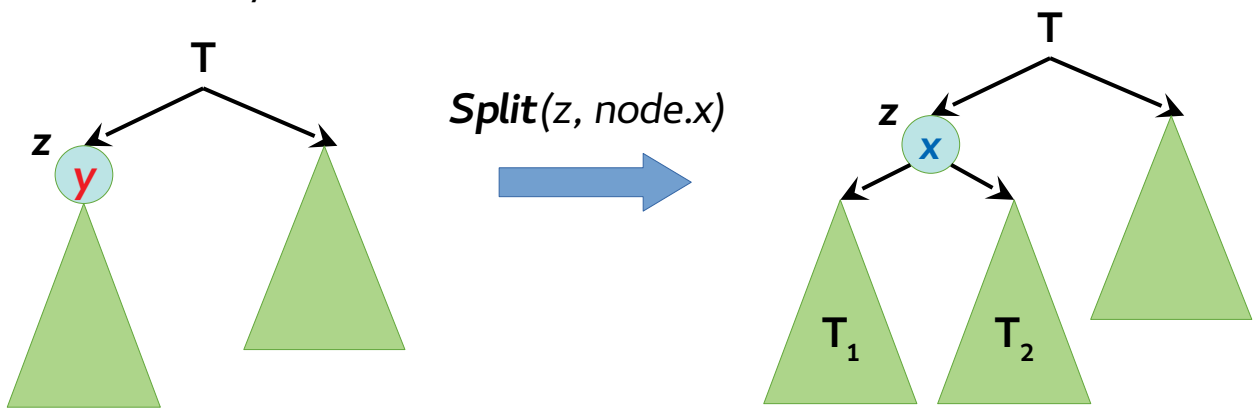
```
function TreapInsert(treap, node)
    TreapSplit(treap, node.x, t1, t2)     // O(h)
    TreapMerge(t1, node, t1)              // O(h)
    TreapMerge(t1, t2, t1)                // O(h)
    return t1
end function
```

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

Добавление элемента в декартово дерево

Вариант 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 T_1 узлы *z* с ключами ≤ *node.x*, в T_2 — с ключами > *node.x*
- Делаем T_1 и T_2 левым и правым поддеревьями узла *node*
- Дерево *node* ставим на место узла *z*



Удаление элемента из декартова дерева

Вариант 1 — с использованием операции разбиения (*Split*)

- Находим в дереве T узел *node* с ключом x
- Разбиваем дерево T по ключу x на деревья T_1 и T_2: ***Split***(T, x)
- Отделяем от дерева T_1 узел с ключом x, для этого разбиваем T_1 по ключу $x - \varepsilon$ на поддеревья T_1 и T_3 (содержит ключ x): ***Split***$(T_1, x - \varepsilon)$
- Сливаем деревья T_1 и T_2: ***Merge***(T_1, T_2)

```
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
```

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

Удаление элемента из декартова дерева

Вариант 2 — без использования операции разбиения

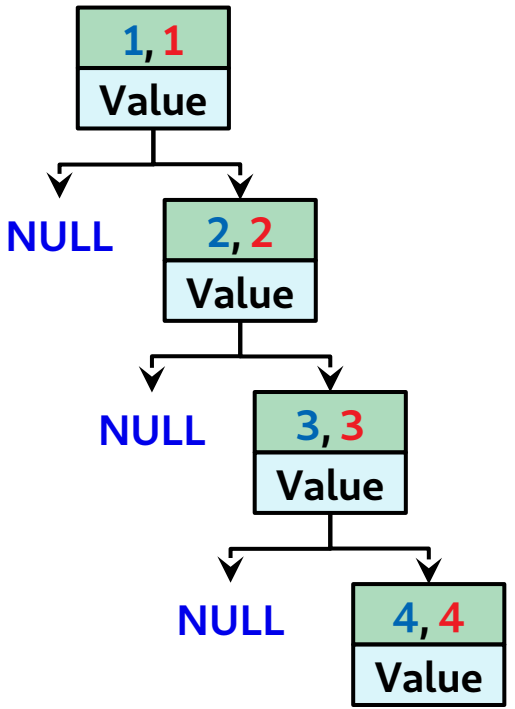
- Находим в дереве *T* узел *node* с ключом *x*
- Сливаем левое и правое поддеревья дерева node: ***Merge***(*node.left*, *node.right*)
- Результат слияния поддеревьев ставим на место узла *node*

Высота декартова дерева

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

Как избежать такой ситуации?

Мы можем задавать приоритеты для регулирования высоты дерева



Псевдослучайные приоритеты

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

Доказательство:

- Пусть x_k — вершина с k-ым по величине ключом
- Введём индикаторную величину $A_{i,j}$:

$$A_{i,j} = \begin{cases} 1, & x_i \text{ является предком } x_j \\ 0, & \text{иначе} \end{cases}$$

- $d(v)$ — глубина вершины v, может быть выражена через число её предков:

$$d(x_k) = \sum_{i=1}^{n} A_{i,k}$$

Псевдослучайные приоритеты

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

Доказательство (продолжение):

- Для индикаторной величины $A_{i,j}$ справедливо следующее:

$$M[A_{i,j}]=P[A_{i,j}=1]$$

- Следовательно, математическое ожидание глубины отдельно взятой вершины:

$$M[d(x_k)]=\sum_{i=1}^{n} P[A_{i,k}=1]$$

(по свойству линейности математического ожидания)

Псевдослучайные приоритеты

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

Доказательство (продолжение):

- Для подсчёта средней глубины вершин в дереве определим вероятность того, что вершина x_i является предком вершины x_k: $P[A_{i,k} = 1]$.
- Введём обозначение: $X_{i,k}$ — множество ключей $\{x_i, \dots, x_k\}$, если $i < k$, или $\{x_k, \dots, x_i\}$, если $i > k$.

$$X_{i,k} = X_{k,i};$$

$$|X| = |k - i| + 1.$$

Лемма. Для любых $i \neq k$, x_i является предком x_k тогда и только тогда, когда x_i имеет наибольший приоритет среди $X_{i,k}$.

Псевдослучайные приоритеты

Лемма. Для любых $i \neq k$, x_i является предком x_k тогда и только тогда, когда x_i имеет наибольший приоритет среди $X_{i,k}$.

Доказательство:

- Если вершина x_i — корень, то она является предком x_k и имеет максимальный приоритет среди всех вершин, следовательно, и среди $X_{i,k}$
- Если корень — x_k, то x_i не является предком x_k, и x_k имеет максимальный приоритет в декартовом дереве; следовательно, x_i не имеет наибольший приоритет среди $X_{i,k}$
- Если корнем является какая-либо другая вершина x_m, тогда, если x_i и x_k лежат в разных поддеревьях, то $i < m < k$ или $i > m > k$; следовательно, x_m содержится в $X_{i,k}$. В таком случае x_i не является предком x_k, и наибольший приоритет среди $X_{i,k}$ имеет вершина x_m.
- Если x_i и x_k лежат в одном поддереве, доказательство применяется по индукции. ■

Псевдослучайные приоритеты

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

Доказательство (продолжение):

- Так как распределение приоритетов равномерное, каждая вершина среди $X_{i,k}$ может иметь максимальный приоритет:

$$P[A_{i,k}=1]=\begin{cases}\dfrac{1}{k-i+1}, & k>i\\ 0, & k=i\\ \dfrac{1}{i-k+1}, & k<i\end{cases}$$

Псевдослучайные приоритеты

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

Доказательство (продолжение):

- Подставляем последнее в формулу с математическим ожиданием глубины вершины

$$M[d(x_k)] = \sum_{i=1}^{n} P[A_{i,k} = 1] = \sum_{i=1}^{k-1} \frac{1}{k-i+1} + \sum_{i=k+1}^{n} \frac{1}{i-k+1}$$

- Используя неравенство

$$\sum_{i=1}^{n} \frac{1}{i} < 1 + \ln n$$

, получаем

$$M[d(x_k)] < \ln(k) + \ln(n-k+1) = O(\ln n)$$

Псевдослучайные приоритеты

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

Доказательство (окончание):

- $\log n$ отличается от $\ln n$ в константу раз, поэтому $\log n = O(\ln n)$; следовательно, $\boldsymbol{M[d(x_k)] = O(\log n)}$. ■

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

Дальнейшее чтение

- Разобрать доказательство утверждения о высоте декартова дерева со случайными приоритетами

- Изучить способы построения декартова дерева при условии, что все добавляемые элементы известны заранее (*offline*): $(x_1, y_1), \ldots, (x_n, y_n)$

  1. Алгоритм с квадратичной сложностью $O(n^2)$
  2. Алгоритм с линейной сложностью $O(n)$

- Прочитать про *неявные декартовы деревья*

ご清聴ありがとうございました!

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Осенний семестр, 2021 г.

Гармонические числа

- Сумму обратных величин первых n последовательных чисел натурального ряда называют n-м **гармоническим числом**:

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

- Число H_n — это площадь под ступенчатой функцией, которую можно определить, вычисляя значения функции $1/x$ для целых чисел от 1 до n
- Гармонические числа являются дискретизированной версией функции *натурального логарифма*

$$\ln n = \int_1^n \frac{1}{x} dx; \quad H_n = \sum_{i=1}^{n} \frac{1}{i}$$

- Границы n-го гармонического числа определяются* как

$$\ln n < H_n < 1 + \ln n, \quad \text{при } n > 1$$

* Грэхем Р. Л., Кнут Д. Э., Паташник О. **Конкретная математика. Математические основы информатики.** — 3-е изд. — М.: Мир, 2009. — 703 с. [с. 303-309]