Декартовы деревья

Дискретный анализ 2016/17

1 октября 2016 г.

Литература

- Оригинальная статья: Randomized Search Trees, Siedel, Aragon. `http://vis.lbl.gov/~aragon/pubs/rst96.pdf`
- E-maxx Максима Иванова, статья про декартовы деревья: `http://e-maxx.ru/algo/treap`
- Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.. Алгоритмы: построение и анализ, 2-е издание, М.:Вильямс, 2005, стр. 360-364, задача 13-4, «Дерамиды».

Декартовы деревья

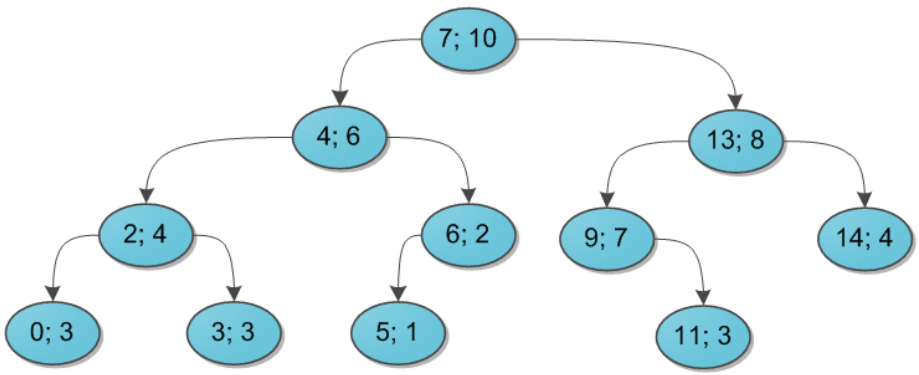
Определение и свойства

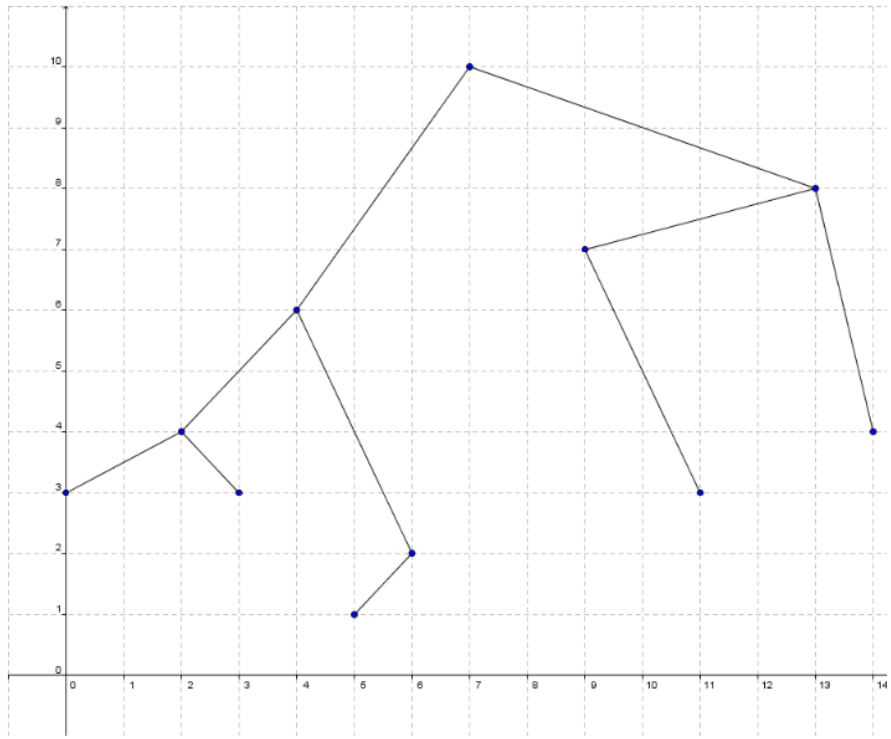
Пусть X — множество пар $< key, priority >$, оба элемента пары взяты из упорядоченных множеств, приоритеты имеют равномерное распределение.
Декартово дерево:

- бинарное дерево над множеством X;
- порядок ключей соответствует порядку в дереве поиска (левый <= корень <= правый);
- приоритеры преставляют собой пирамиду (приоритет родителя больше приоритета потомков).

Для любого множества X, в котором ключи и приоритеты уникальны, декартово дерево существует и единственно.
Другие варианты названия: treap, дерамида, дуча, курево.

Пример



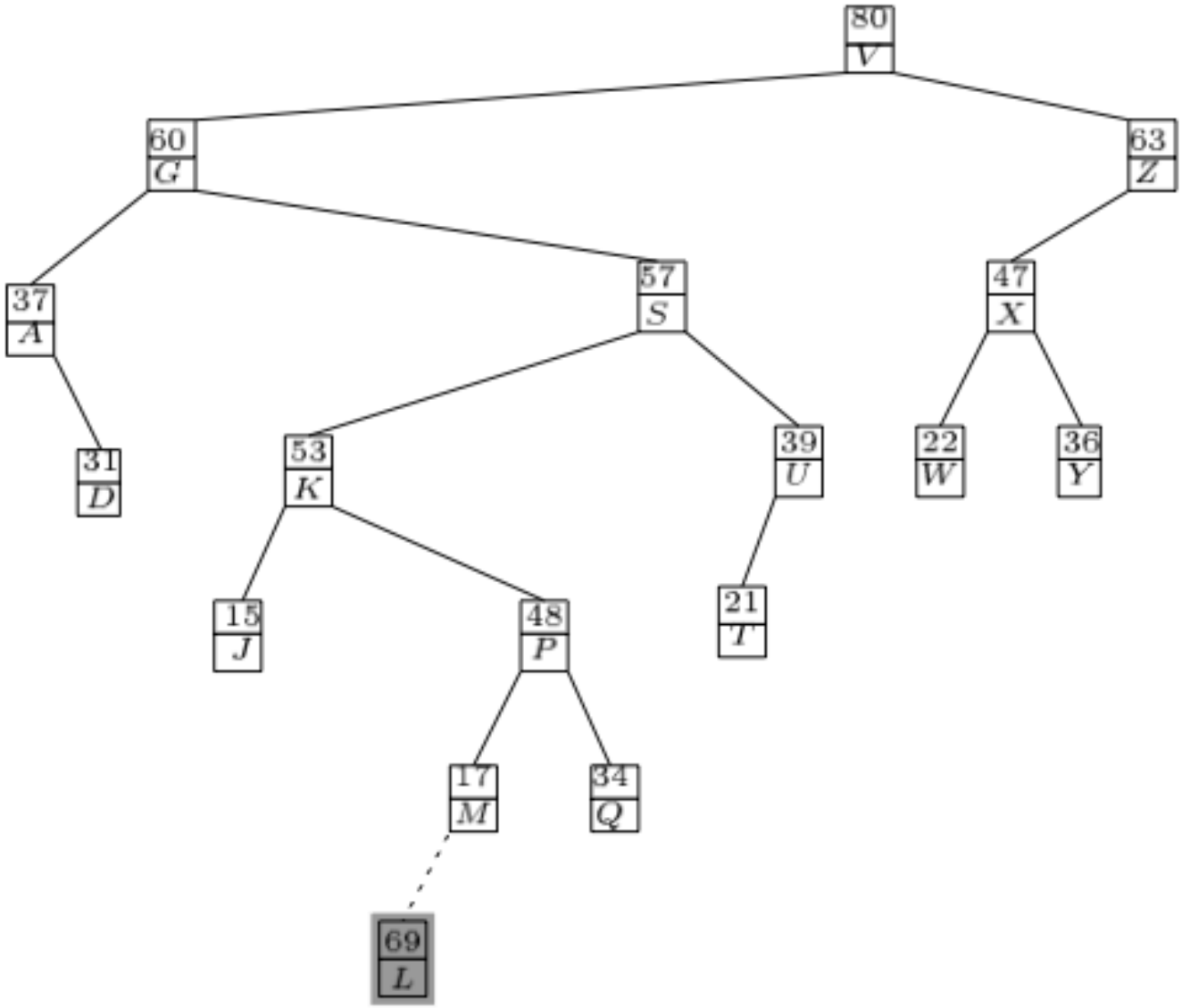


Декартовы деревья

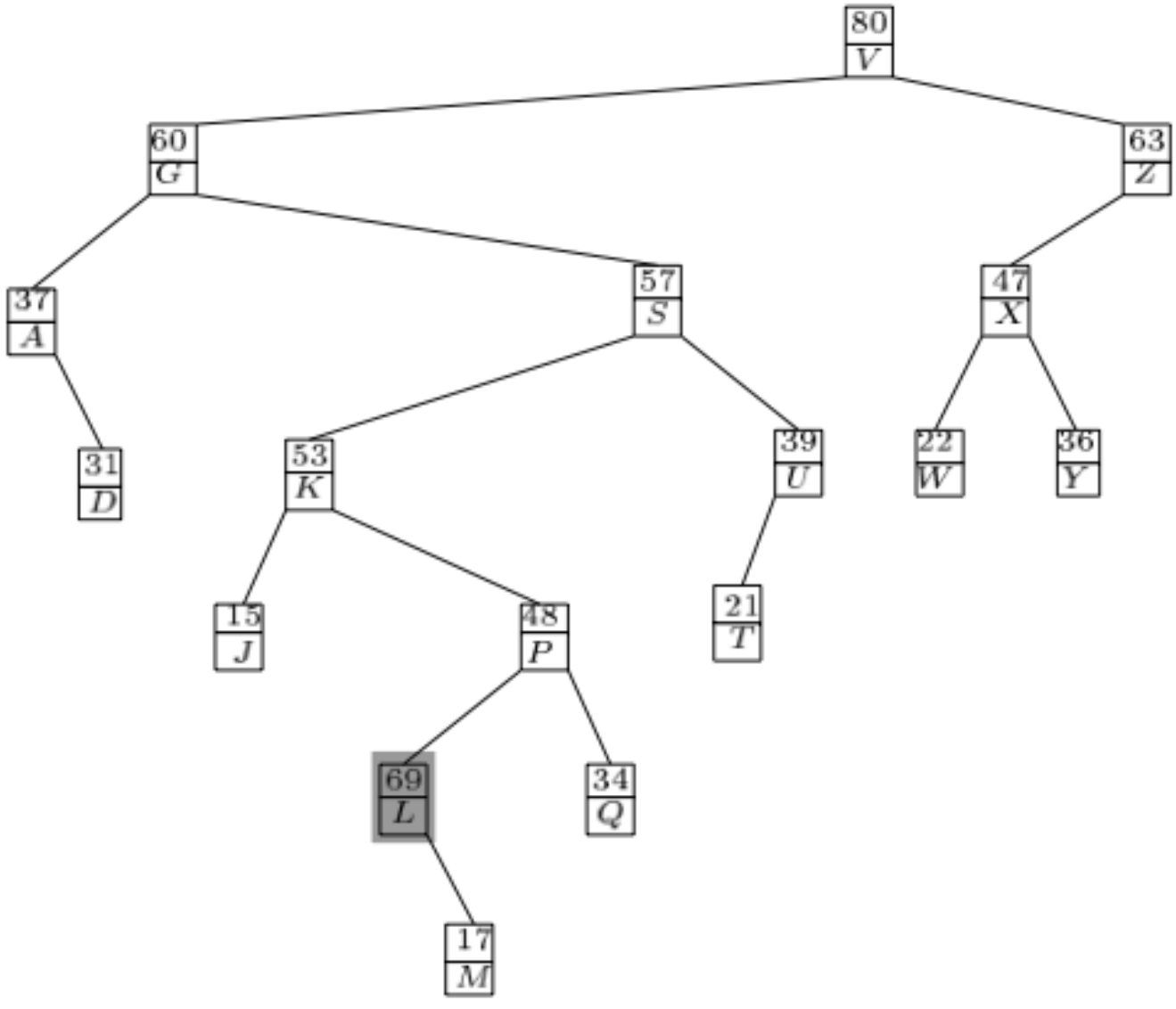
Поиск, вставка и удаление

- Поиск по ключу делается так же, как и в обычном дереве поиска.
- Вставка:
  - вставляем в лист, как в обычном дереве;
  - может нарушиться свойство пирамиды для приоритетов;
  - нужно выполнить некоторое количество поворотов: не нарушают свойства дерева, но могут восстановить пирамиду для приоритетов.
- Удаление:
  - если удаляемый узел — лист, освободить память;
  - иначе — с помощью поворотов сделать узел листом.

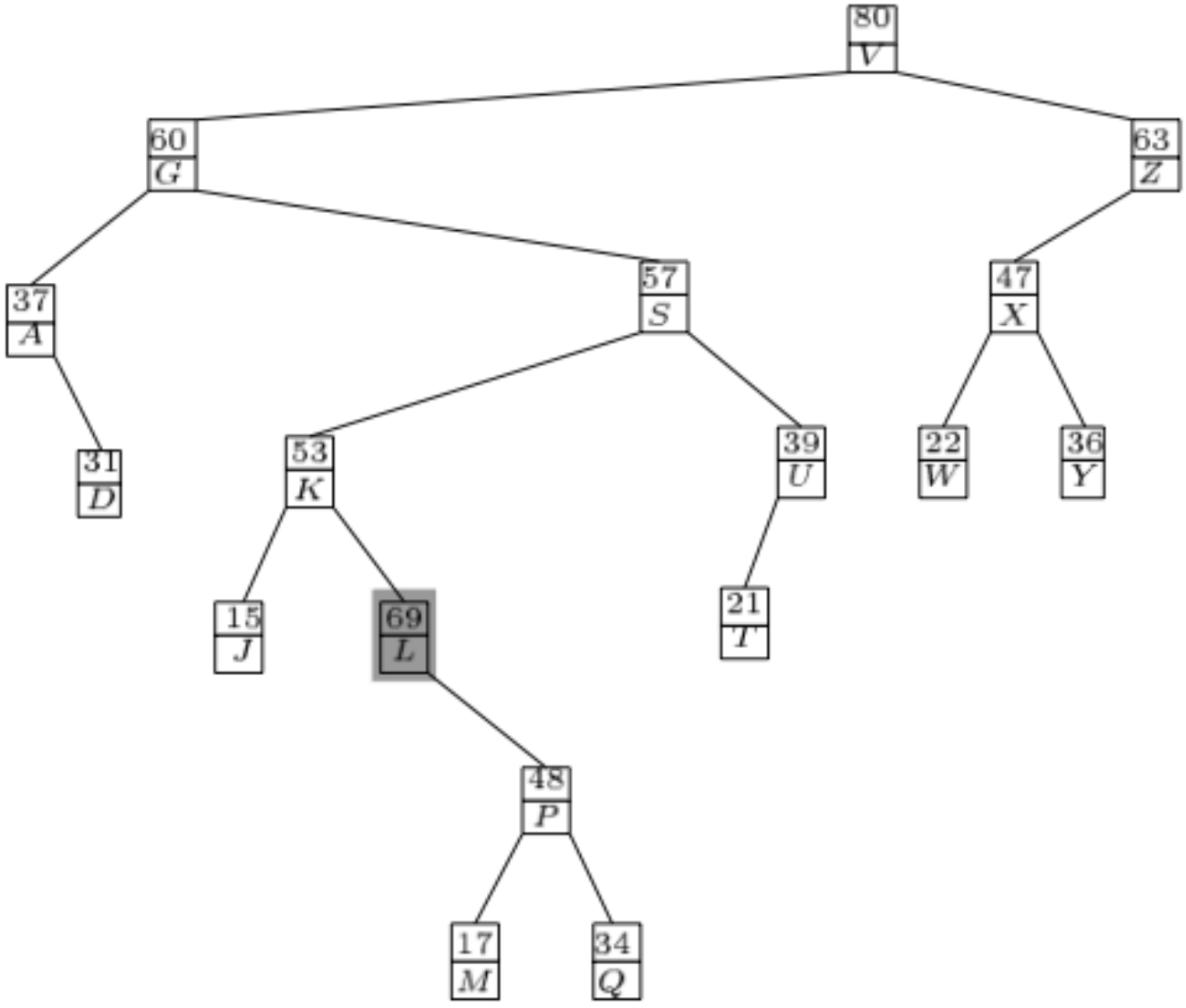
Пример


80
V
60
G
63
Z
37
A
57
S
47
X
31
D
53
K
39
U
22
W
36
Y
15
J
48
P
21
T
17
M
34
Q
69
L

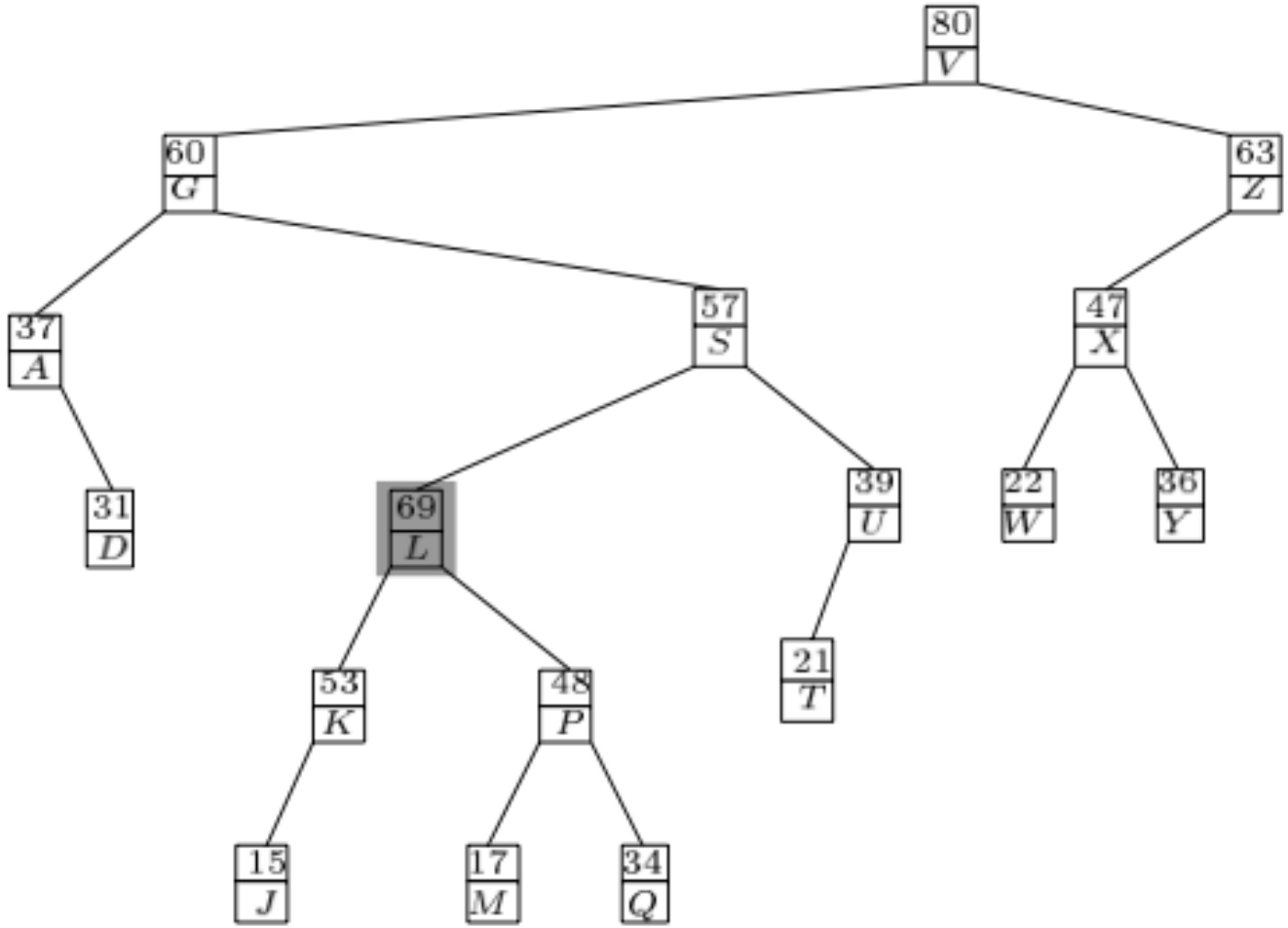
Пример


80
V
60
G
63
Z
37
A
57
S
47
X
31
D
53
K
39
U
22
W
36
Y
15
J
48
P
21
T
69
L
34
Q
17
M

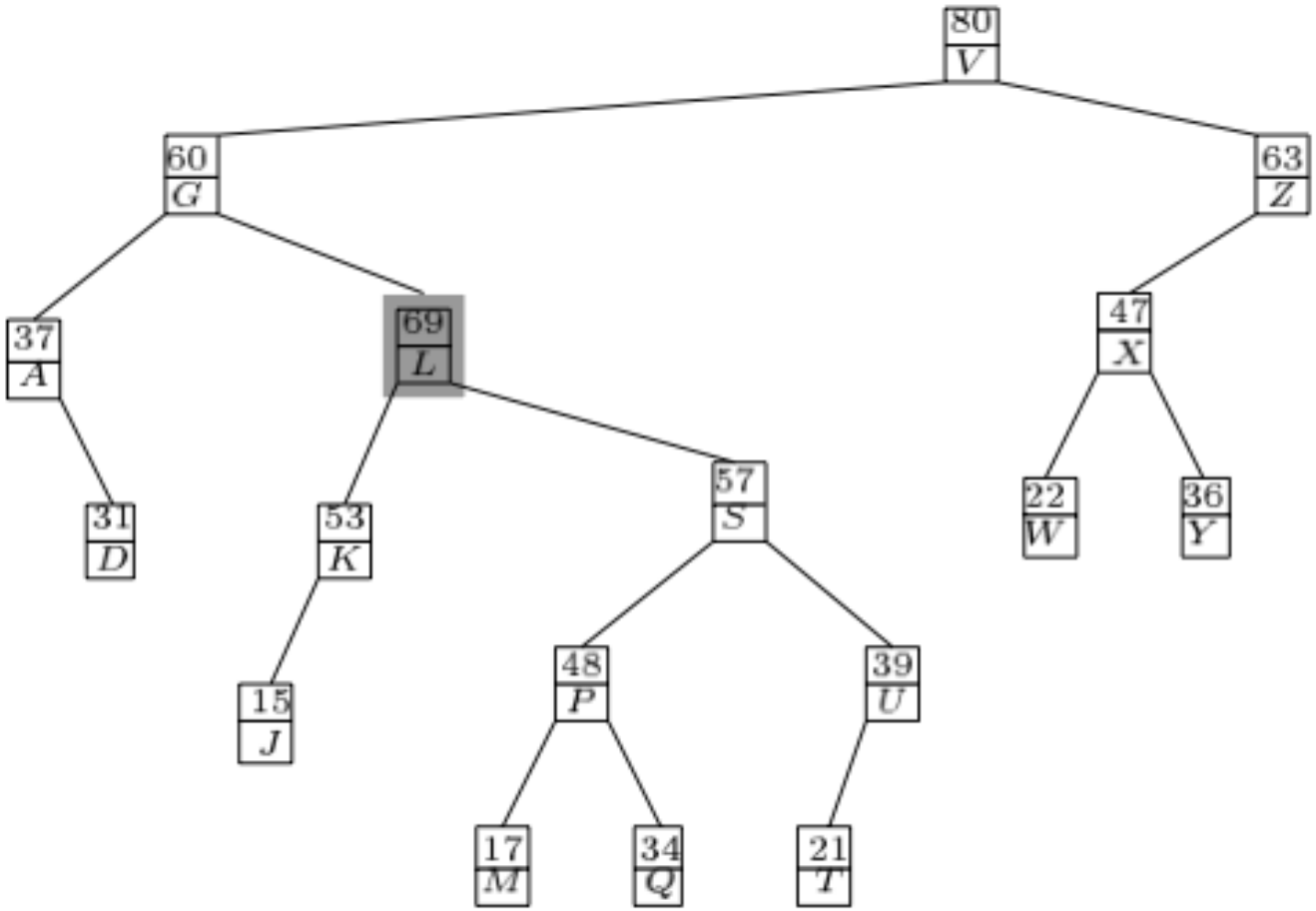
Пример


80
V
60
G
63
Z
37
A
57
S
47
X
31
D
53
K
39
U
22
W
36
Y
15
J
69
L
21
T
48
P
17
M
34
Q

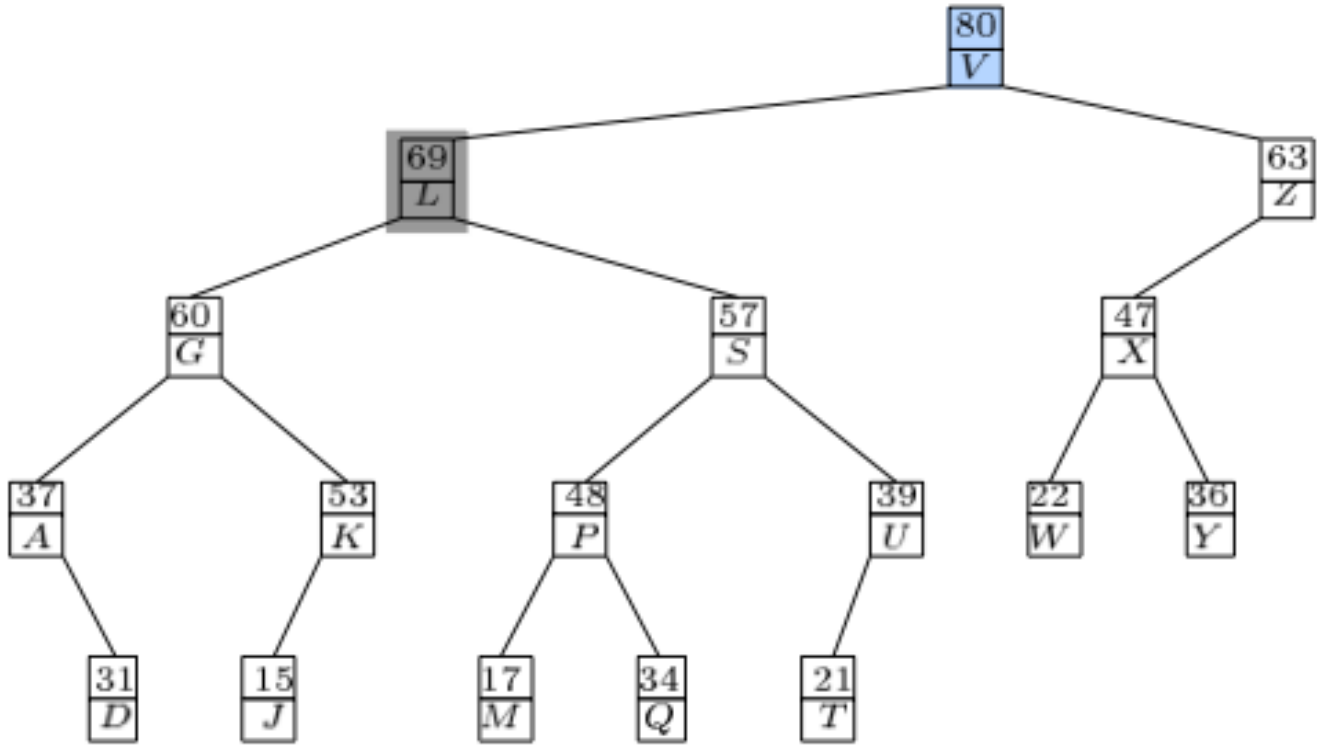
Пример


80
V
60
G
63
Z
37
A
57
S
47
X
31
D
69
L
39
U
22
W
36
Y
53
K
48
P
21
T
15
J
17
M
34
Q

Пример


80
V
60
G
63
Z
37
A
69
L
47
X
31
D
53
K
57
S
22
W
36
Y
15
J
48
P
39
U
17
M
34
Q
21
T

Пример


80
V
69
L
63
Z
60
G
57
S
47
X
37
A
53
K
48
P
39
U
22
W
36
Y
31
D
15
J
17
M
34
Q
21
T

Реализация вставки

$\textsc{Insert}(T, item)$

```
1  x ← INSERT-INTO-TREE(T, item)
2  while p[x] ≠ NULL && priority[p[x]] < priority[x]
3        if x = left[p[x]]
4              RIGHT-ROTATE(p[x])
5        else LEFT-ROTATE(p[x])
```

Реализация удаления

$\textsc{Delete}(T, item)$

1 $x \leftarrow \textsc{Find}(T, item)$
2 **while** $left[x] \neq NULL \parallel right[x] \neq NULL$
3 $\quad y \leftarrow \textsc{Max-Priority}(left[x], right[x])$
4 $\quad$ **if** $y = left[x]$
5 $\quad\quad \textsc{Right-Rotate}(x)$
6 $\quad$ **else** $\textsc{Left-Rotate}(x)$
7 $\textsc{Delete-From-Tree}(T, x)$

Разбиение и слияние

- Если необходимо разбить дерево на два по ключу k (в первом все ключи меньшие k, во втором — большие), можно добавить элемент (k, ∞)
- Если нужно объединить два поддерева, можно сделать их сыновьями корня $(any, -\infty)$ и удалить корень
- Обычно сначала реализуют операция разбиения и слияния, а через них — удаление и вставку

Разбиение

- Разбить дерево T на два по ключу k, чтобы в первом были элементы, меньшие k, а во втором – большие
- Если ключ корня дерева T меньше, чем k, по которому разбиваем, то корень и его левое поддерево – новое дерево T_1 (если больше – аналогично для T_2)
- Рекурсивно разбиваем правое поддерево T по ключу k, получаем T' и T_2
- Дерево с меньшими элементами T' становится правым поддеревом T_1
- T_1 и T_2 — результат разбиения

Реализация разбиения

```
SPLIT(Treap t, key, Treap& t1, Treap& t2)
1  if t = NULL
2        t1 ← t2 ← NULL
3  else if key > key[t]
4        SPLIT(right[t], key, right[t], t2)
5        t1 ← t
6  else
7        SPLIT(left[t], key, t1, left[t])
8        t2 ← t
```

Слияние

- Соединить два декартовых дерева T_1 и T_2 в T, при условии что ключи всех элементов из T_1 меньше ключей всех элементов из T_2
- Корни деревьев T_1 и T_2 имеют приоритеты p_1 и p_2 соответственно
- Если $p_1 > p_2$:
  - корень дерева T_1 – корень их объединения T;
  - левое поддерево остается без изменений;
  - правое поддерево объединяется с T_2, результат становится новым правым поддеревом.
- Иначе — алгоритм выше с инвертированными параметрами

Реализация слияния

```
MERGE(Treap& t, Treap t1, Treap t2)
 1  if t1 = NULL && t2 ≠ NULL
 2        t ← t2
 3  else if t2 = NULL && t1 ≠ NULL
 4              t ← t1
 5  else if priority[t1] > priority[t2]
 6              MERGE(right[t1], right[t1], t2)
 7              t ← t1
 8        else
 9              MERGE(left[t2], t1, left[t2])
10              t ← t2
```

Вставка через разбиение и слияние

- Алгоритм №1:
  - разбиваем дерево по ключу вставляемого элемента k на два новых T_1 и T_2;
  - выполняем слияние T_1 и дерева из одного вставляемого элемента $< k, p >$;
  - выполняем слияние результата с деревом T_2.
- Алгоритм №2:
  - спускаемся по дереву как при поиске, ищем либо $NULL$, либо элемент, приоритет которого меньше, чем у вставляемого элемента;
  - выполняем разбиение найденного поддерева по ключу вставляемого элемента k на два новых T_1 и T_2;
  - $< k, p >$ вставляем как корень вместо найденного дерева, T_1 и T_2 – его левое и правое поддеревья соответственно.

Удаление через разбиение и слияние

- Алгоритм №1:
  - разбиваем дерево по ключу удаляемого элемента k на два новых T_1 и T_2;
  - разбиваем T_1 на T_1' и T_2', чтобы во втором дереве оказались все элементы с ключом k;
  - выполняем слияние T_1' с T_2.
- Алгоритм №2:
  - спускаемся по дереву как при поиске, ищем элемент с нужным ключом;
  - выполняем слияние его левого и правого поддеревьев;
  - вставляем получившееся дерево вместо найденного с корнем k.

Декартовы деревья

Сложность доступа к случайному элементу

Дано декартово дерево, с элементами $x_1 ... x_n$, такими что $x_i.key < x_{i+1}.key$, а все его приоритеты различны

Лемма

x_i является предком x_j тогда и только тогда, когда x_i имеет наибольший приоритет среди всех $x_{i...j}$

Доказательство.

- если приоритет x_i наибольший, он либо предок x_j, либо у них есть общий предок, который лежит в множестве $x_{i+1...j}$, то есть у этого элемента больший приоритет, чем у x_i – противоречие;
- если x_i – предок x_j, то предположим, что есть элемент x_k, $k \in [i+1; j-1]$, приоритет которого больше, тогда либо x_i и x_j лежат по разные стороны от него (противоречие), либо по одну, но тогда нарушается условие возрастания ключей, значит приоритет x_i – наибольший.

Сложность доступа к случайному элементу

- $D(x)$ — глубина узла x, количество узлов от x до корня
- $A_{ij} = I\{x_i \ is \ ancestor \ of \ x_j\}$
- $D(x_l) = \sum_{i=1}^{n} A_{i,l}$
- $E(D(x_l)) = \sum_{i=1}^{n} E(A_{i,l}) = \sum_{i=1}^{n} Pr(x_i \ is \ ancestor \ x_l)$
- Вероятность того, что x_i предок x_j при равномерном распределении приоритеров:

$$E(A_{i,j}) = \frac{1}{|i-j|+1}$$

- Учитывая, что $\sum_{i=1}^{n} \frac{1}{i} \leq \ln(n) + 1$:

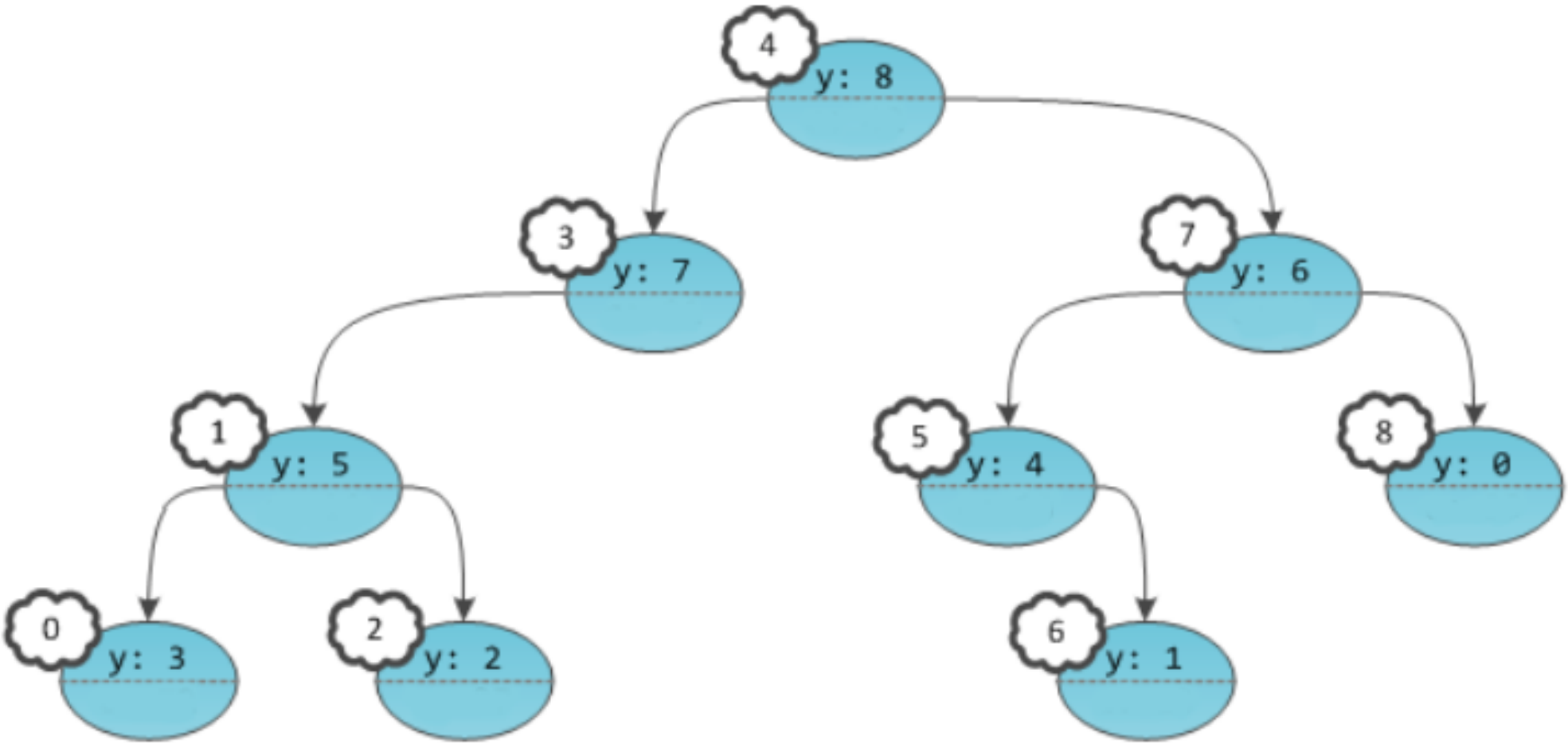
$$E(D(x_l)) = \sum_{i=1}^{n} E(A_{i_l}) \leq \ln(l) + \ln(n-l) + 2$$

- То есть: $E(D(x_l)) = O(\log(n))$

Определение и применение

- Простая модификация обычных декартовых деревьев: неявным ключом будет индекс текущего элемента в массиве, построенном по отсортированным элементам
- Что можно сделать за $O(\log n)$:
  - вставка в массив на любую позицию;
  - удаление произольного элемента;
  - сумма/минимум/максимум/... на произвольном подотрезке.

Пример



Детали

- Если хранить сам порядковый номер, то придется пересчитывать $O(n)$ элементов каждый раз
- Будем хранить количество элементов в поддереве, тогда порядковый номер – количество детей в левом поддереве + количество детей в левых поддеревьях для предков текущего элемента, у которых он в правом поддереве
- Merge не изменяется (не использует информацию о ключах), нужно только обновлять число элементов в поддеревьях
- Split делит не по ключу, а по порядковому номеру (индексу) элемента: если номер корня меньше, чем нужный нам индекс, то вызываем рекурсивно для правого поддерева, уменьшив индекс на порядковый номер корня, иначе – рекурсивный вызов для левого поддерева