Декартовы деревья

Дискретный анализ 2012/13

Андрей Калинин, Татьяна Романова

1 октября 2012 г.

Литература

- Оригинальная статья: Randomized Search Trees, Siedel, Aragon. http://people.ischool.berkeley.edu/ aragon/pubs/rst96.pdf
- E-maxx Максима Иванова, статья про декартовы деревья: http://e-maxx.ru/algo/treap
- Посты на habrahabr'e: http://habrahabr.ru/post/101818/ и т. п.

Раздел

Проблемы обычных деревьев

- Бинарное дерево поиска при добавлении упорядоченных данных вырождается в линейный список.
- Способы балансировки (avl, красно-черные, 2-3-деревья) сложно реализовать.
- Нет быстрых операций слияния двух деревьев или разделения по ключу.

Рандомизированные деревья

- Разные подходы:
  - Случайное поведение алгоритма: иногда вставляем в лист, иногда в середину дерева (http://habrahabr.ru/post/145388/)
  - Добавление случайного значения к ключу (декартовы деревья)
- Оцениваем время работы в среднем: например, делаем большое количество операций поиска и считаем, сколько всего узлов посетили и делим на количество операций.
- Реализация проще, пригождаются в некоторых других задачах, не гарантируют точное время доступа к элементу.

Раздел

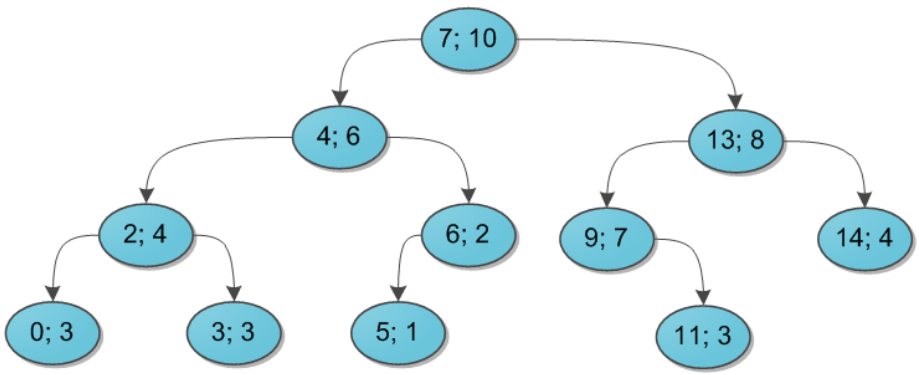
Определение и свойства

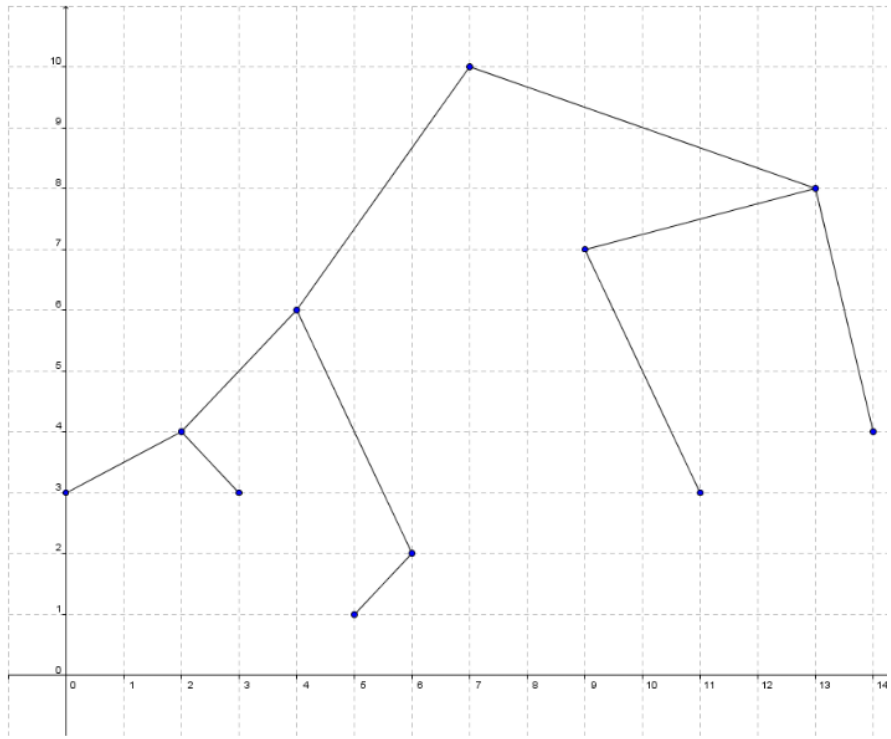
Пусть X — множество пар $< key, priority >$, оба элемента пары взяты из упорядоченных множеств, приоритеты имеют равномерное распределение.
Декартово дерево:

- 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: дерамида, дуча, treap.

Пример



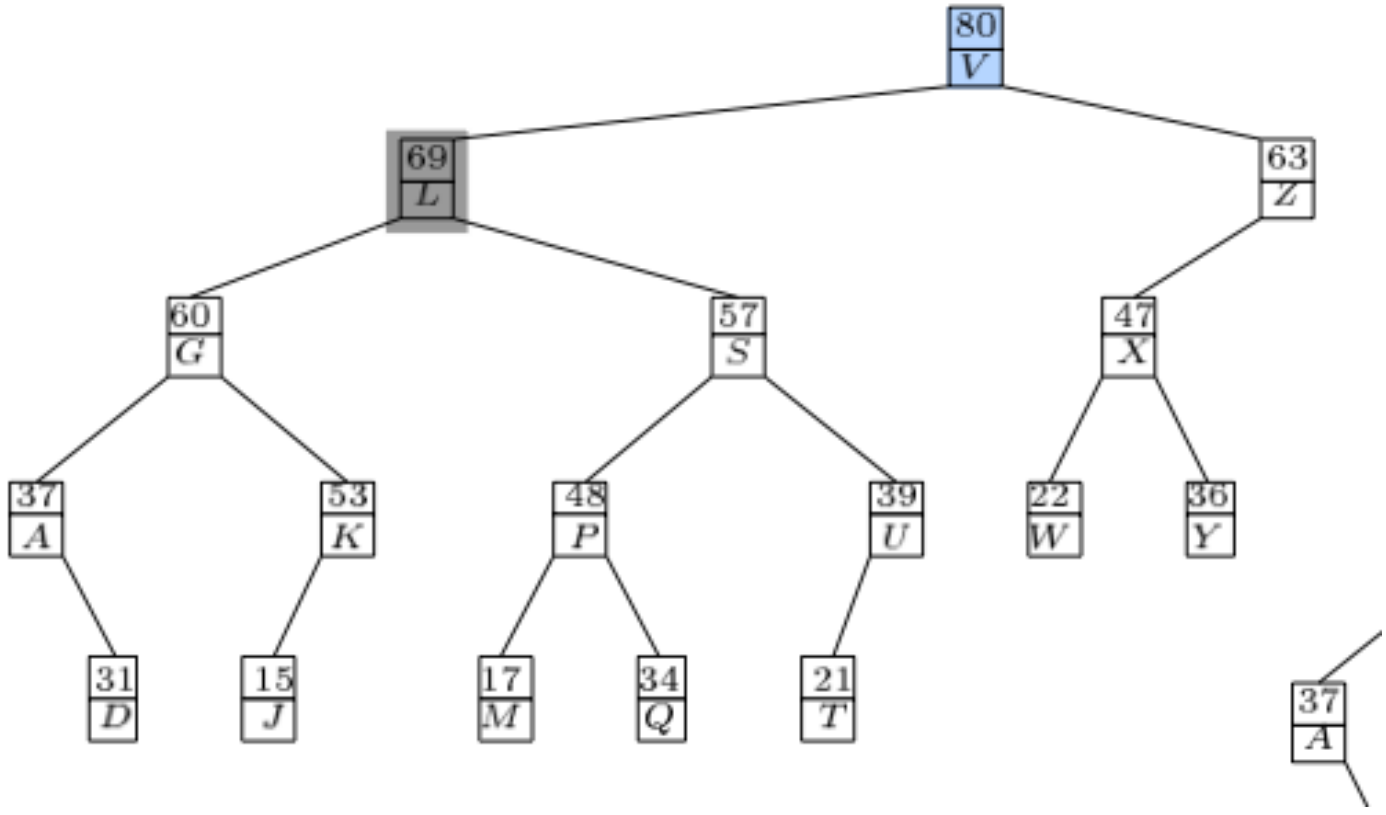


Раздел

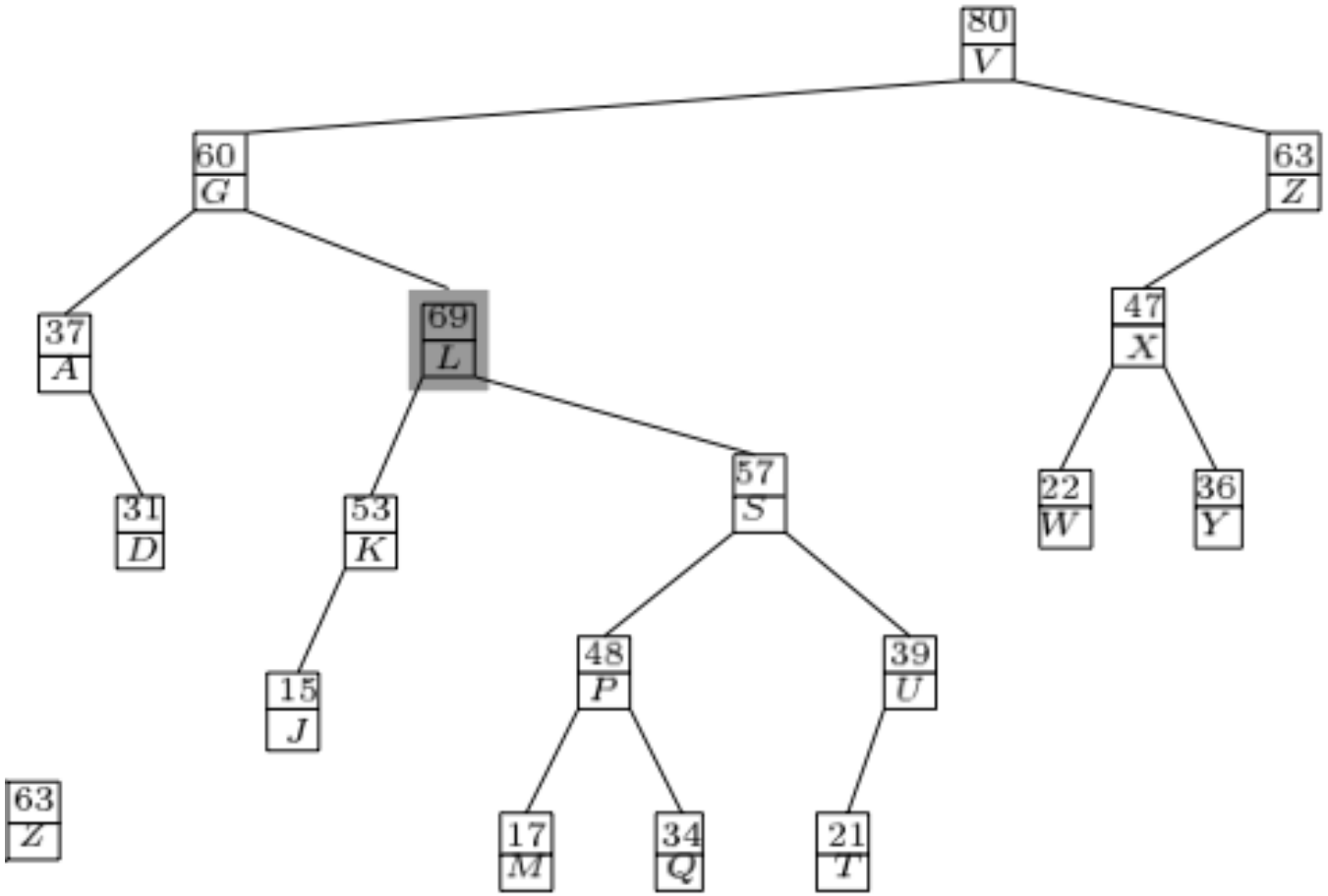
Поиск, вставка и удаление

- 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,
  - в противном случае, с помощью поворотов сделать узел листом.

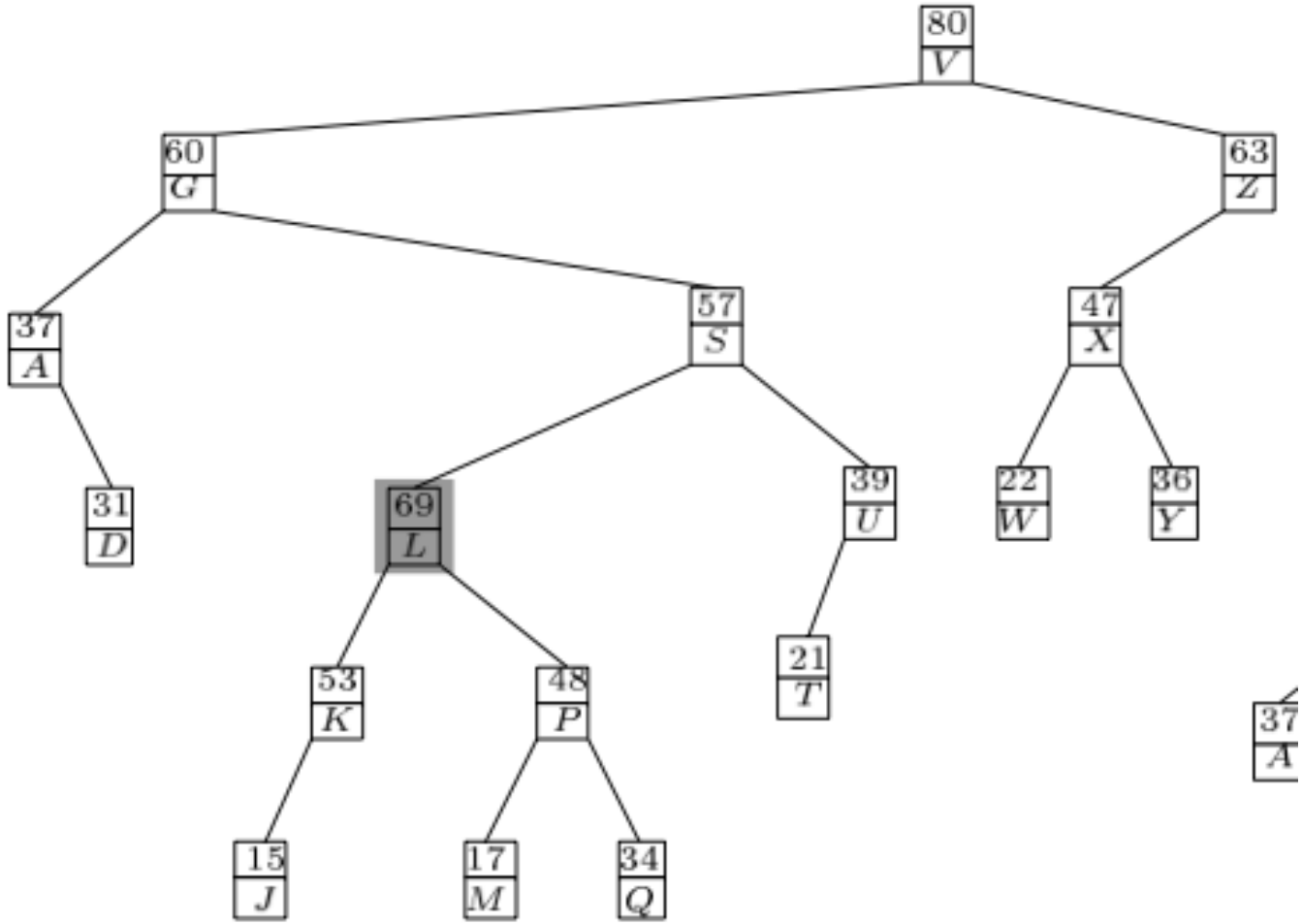
Пример


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
37
A

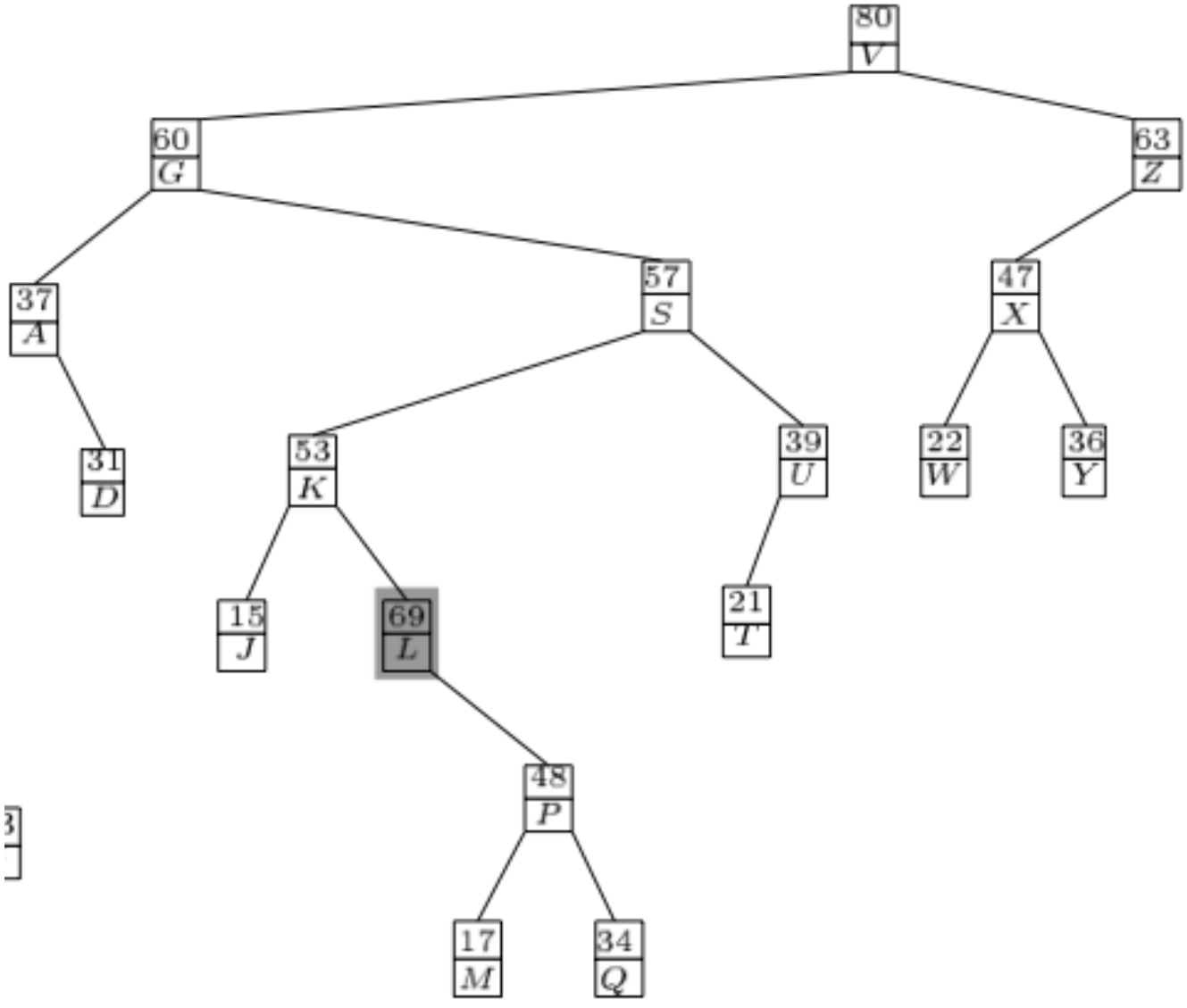
Пример


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
63
Z
17
M
34
Q
21
T

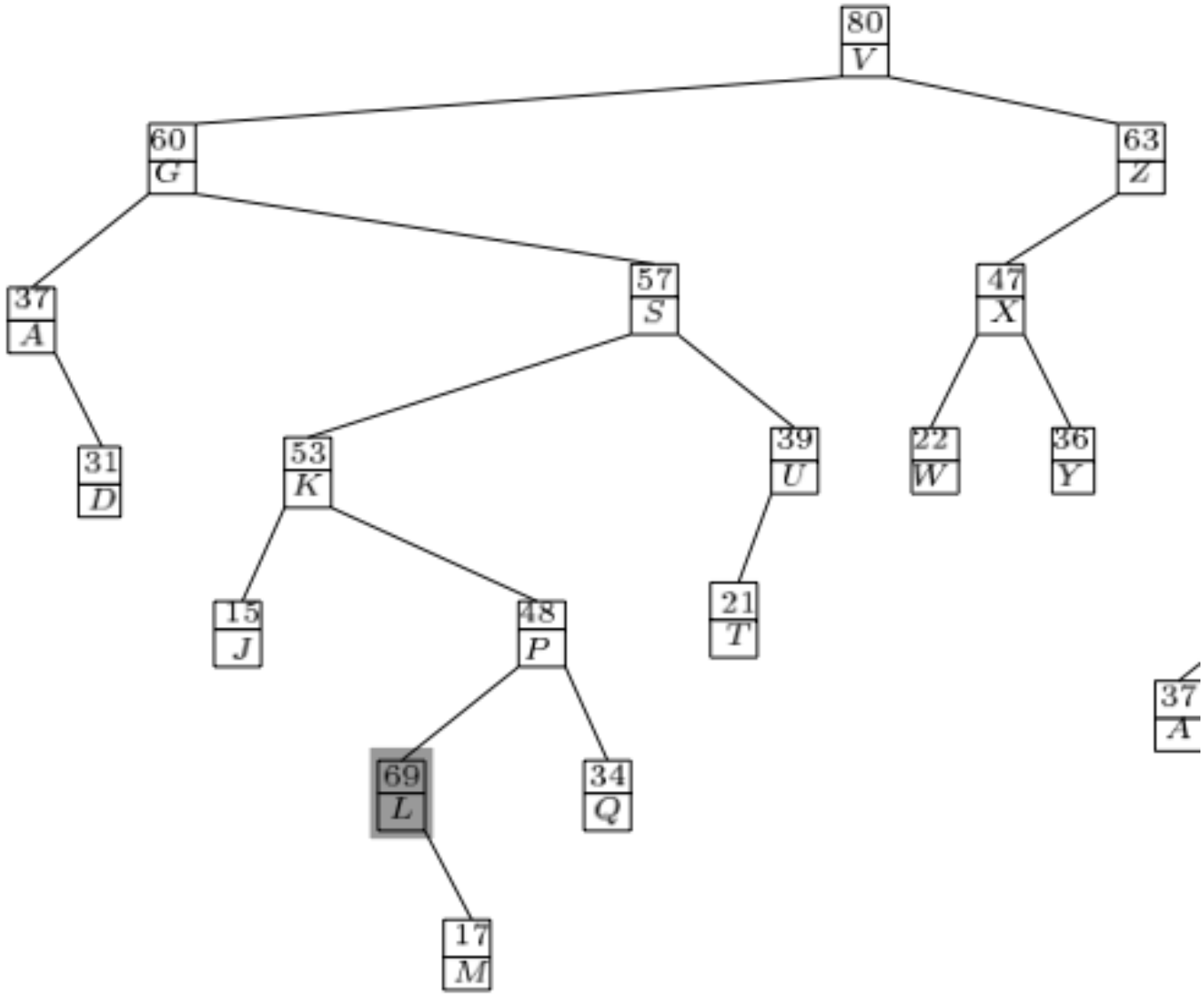
Пример


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
37
A
15
J
17
M
34
Q

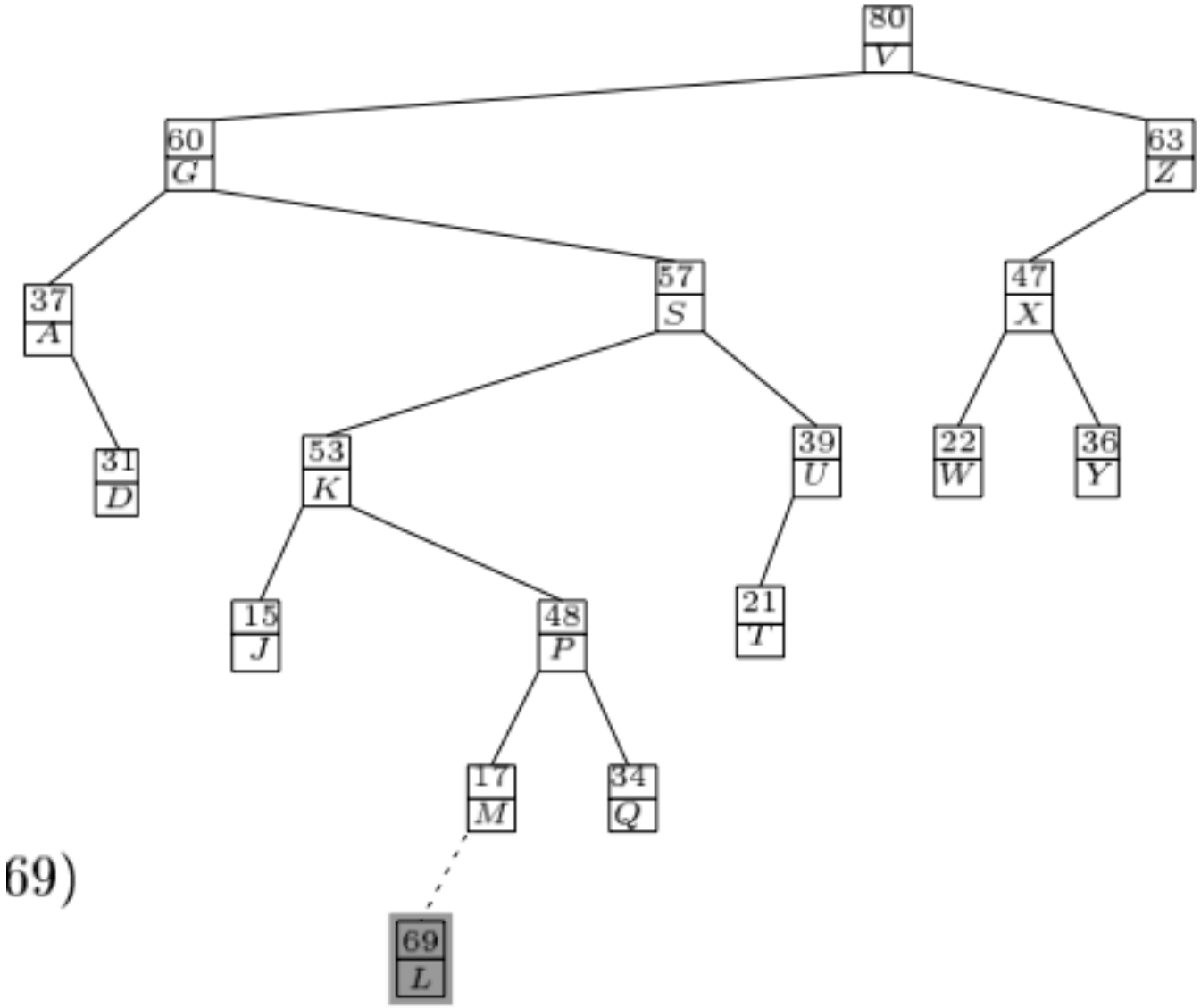
Пример


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
53
K
39
U
22
W
36
Y
15
J
48
P
21
T
37
A
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
48
P
21
T
17
M
34
Q
69)
69
L

Реализация вставки

```
function EMPTY-TREAP() : treap
    tnull→[priority,lchild,rchild] ← [−∞,tnull,tnull]
    return( tnull )

procedure TREAP-INSERT( (k,p) : item, T : treap )
    if T = tnull then T ← NEWNODE()
                       T→[key,priority,lchild,rchild] ← [k,p,tnull,tnull]
    else if k < T→key then TREAP-INSERT( (k,p) ,T→lchild )
                           if T→lchild →priority > T→priority then ROTATE-RIGHT( T )
    else if k > T→key then TREAP-INSERT( (k,p) ,T→rchild )
                           if T→rchild →priority > T→priority then ROTATE-LEFT( T )
    else (* key k already in treap T *)
```

Реализация удаления

```
procedure TREAP-DELETE ( k : key, T : treap )
    tnull→key ← k
    REC-TREAP-DELETE( k, T )
procedure REC-TREAP-DELETE ( k : key, T : treap )
    if k < T→key then REC-TREAP-DELETE( k, T→lchild )
    else if k > T→key then REC-TREAP-DELETE( k, T→rchild )
    else ROOT-DELETE( T )
procedure ROOT-DELETE( T : treap )
    if IS-LEAF-OR-NULL( T ) then T ← tnull
    else if T→lchild→priority > T→rchild→priority then ROTATE-RIGHT( T )
                                                       ROOT-DELETE( T→rchild )
                                                  else ROTATE-LEFT( T )
                                                       ROOT-DELETE( T→lchild )
```

Разбиение и слияние

- Если необходимо разбирть 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.

Раздел

Сложность доступа к случайному элементу

- Дано декартово дерево, с элементами $x_1 ... x_n$, такими что $x_i.key < x_{i+1}.key$.
- $D(x)$ — глубина узла x, количество узлов от x до корня.
- $A_{i,j}$ — характеристическая функция, равна 1, если x_i предок x_j, иначе 0.
- $D(x_l) = \sum_{1 \le i \le n} A_{i,l}$.
- $E(A_{i,j}) = a_{i,j}$. Из линейности мат. ожидания $E(D(x_l)) = \sum_{1 \le i \le n} a_{i,l}$.
- $a_{i,j} = E(A_{i,j}) = Pr(A_{i,j} = 1) = Pr(ancestor(x_i, x_j))$

Сложность доступа к случайному элементу

- x_i будет предком x_j, тогда и только тогда, когда среди всех элементом от i до j x_i имеет наибольший приоритет.
- Вероятность того, что x_i предок x_j при равномерном распределении приоритеров:

$$a_{i,j} = \frac{1}{|i-j|+1}$$

- Тогда: $E(D(x_l)) = H_l + H_{n+1-l} - 1 < 1 + 2 \cdot \ln(n)$

Раздел

Определение и применение

- Простая модификация обычных декартовых деревьев: неявным ключем будет индекс декущего элемента в массиве, построенном по отсотрированным элементам
- Что можно будет сделать за O(log n):
  - Вставка в массив на любую позицию
  - Удаление произольного элемента
  - Сумма, min, max на произвольном подотрезке.

Детали

- Хранить сам порядковый номер нерационально, придется пересчитывать O(n) элементов каждый раз.
- Будем хранить количество элементов в поддереве, тогда порядковый номер можно найти, спускаясь по дереву от корня.
- Теперь можно находить произвольные подотрезки и функции на них (поддерживая дополнительное значение в элементах).