

**Курс «Алгоритмы и алгоритмические языки»
1 семестр 2013/2014**

Лекция 24

Самоперестраивающиеся деревья (*splay trees*)

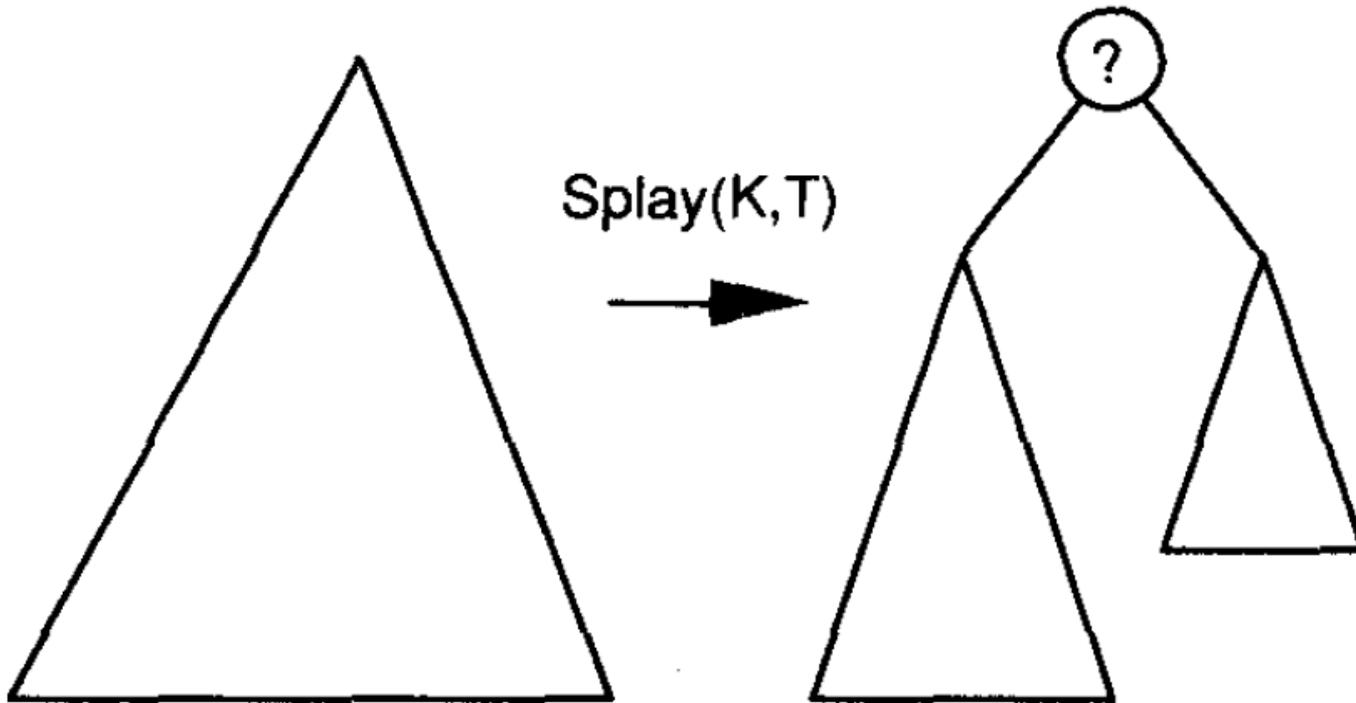
- ◇ Двоичное дерево поиска, не содержащее дополнительных служебных полей в структуре данных (нет баланса, цвета и т.п.)
- ◇ Гарантируется не логарифмическая сложность в худшем случае, а *амортизированная* логарифмическая сложность:
 - ◆ Любая последовательность из m словарных операций (поиска, вставки, удаления) над n элементами, начиная с пустого дерева, имеет сложность $O(m \log n)$
 - ◆ Средняя сложность одной операции $O(\log n)$
 - ◆ Некоторые операции могут иметь сложность $\Theta(n)$
 - ◆ Не делается предположений о распределении вероятностей ключей дерева и словарных операций (т.е. что некоторые операции выполнялись чаще других)
- ◇ Хорошее описание в:
Harry R. Lewis, Larry Denenberg. Data Structures and Their Algorithms. HarperCollins, 1991. Глава 7.3.
<http://www.amazon.com/Structures-Their-Algorithms-Harry-Lewis/dp/067339736X>

Самоперестраивающиеся деревья (*splay trees*)

- ◇ Идея: эвристика Move-to-Front
 - ◆ Список: давайте при поиске элемента в списке перемещать найденный элемент в начало списка
 - ◆ Если он потребуется снова в обозримом будущем, он найдется быстрее
- ◇ Move-to-Front для двоичного дерева поиска: операция $Splay(K, T)$ (подравнивание, перемешивание, расширение)
 - ◆ После выполнения операции $Splay$ дерево T перестраивается (оставаясь деревом поиска) так, что:
 - ◆ Если ключ K есть в дереве, то он становится корнем
 - ◆ Если ключа K нет в дереве, то в корне оказывается его предшественник или последователь в симметричном порядке обхода

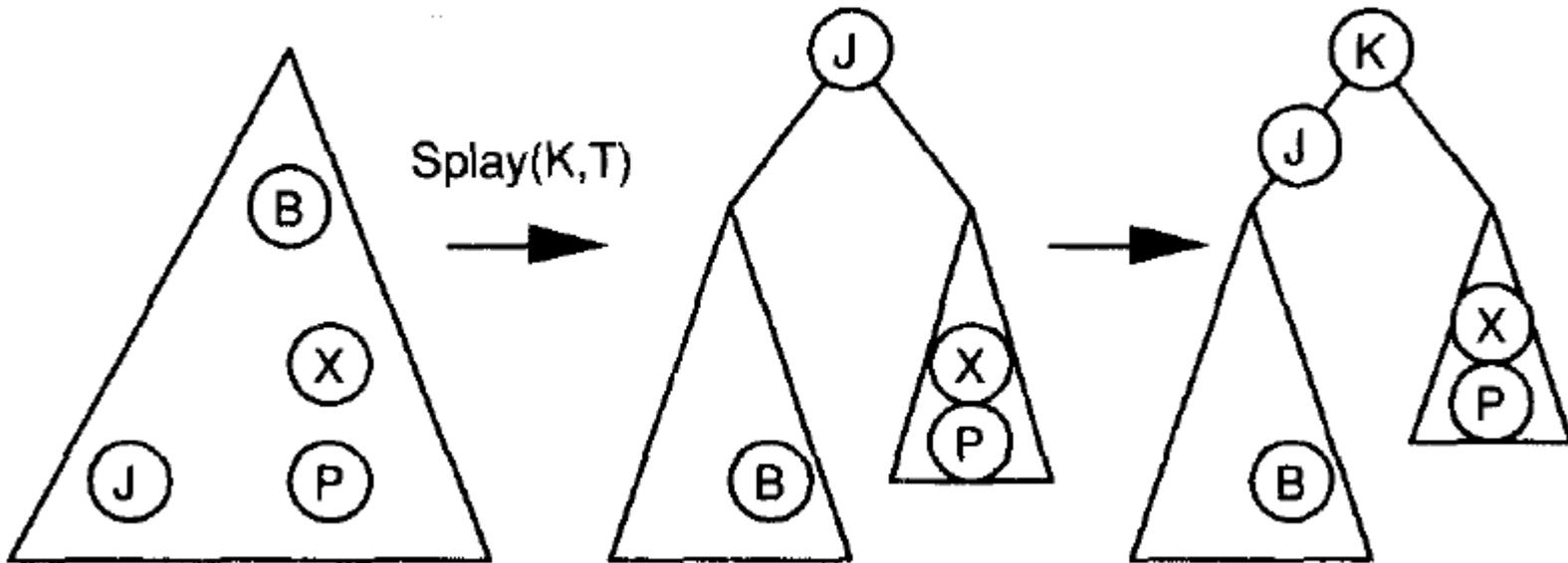
Реализация словарных операций через *splay*

- ◇ Поиск (LookUp): выполним операцию $Splay(K, T)$ и проверим значение ключа в корне:
 - ◆ если значение равно K , то ключ найден



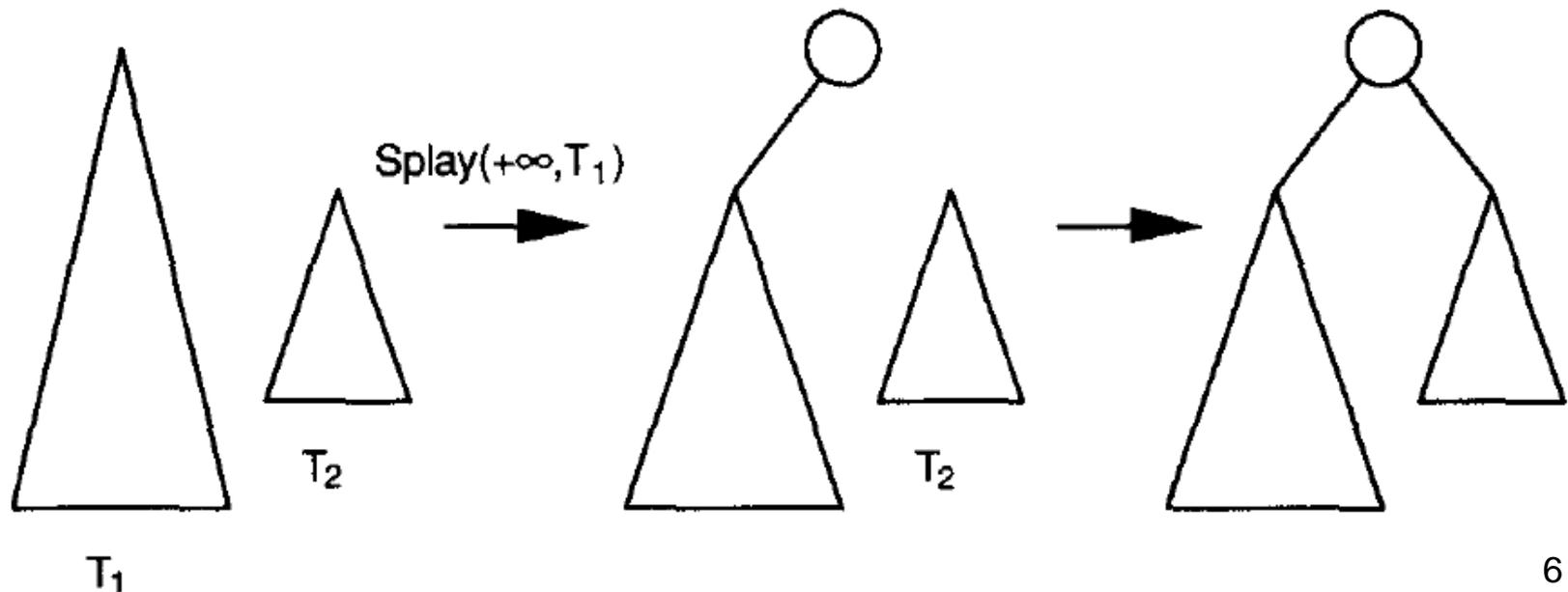
Реализация словарных операций через *splay*

- ◇ Вставка (Insert): выполним операцию $Splay(K, T)$ и проверим значение ключа в корне:
 - ◆ если значение уже равно K , то обновим данные ключа
 - ◆ если значение другое, то вставим новый корень K и поместим старый корень J слева или справа (в зависимости от значения J)



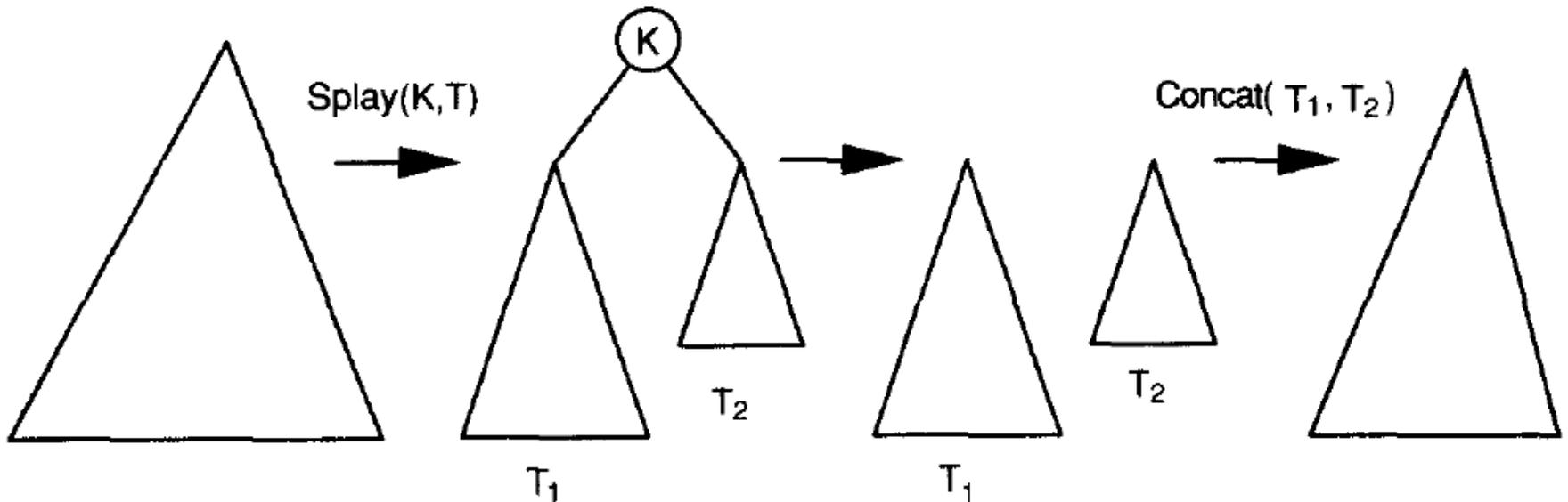
Реализация словарных операций через *splay*

- ❖ Операция *Concat* (T_1, T_2) – слияние деревьев поиска T_1 и T_2 таких, **все** ключи в дереве T_1 **меньше**, чем **все** ключи в дереве T_2 , в одно дерево поиска
- ❖ Слияние (*Concat*): выполним операцию *Splay*($+\infty, T_1$) со значением ключа, заведомо больше любого другого в T_1
 - ◆ После *Splay*($+\infty, T_1$) у корня дерева T_1 нет правого сына
 - ◆ Присоединим дерево T_2 как правый сын корня T_1



Реализация словарных операций через *splay*

- ◇ Удаление (Delete): выполним операцию $Splay(K, T)$ и проверим значение ключа в корне:
 - ◆ если значение **не равно** K , то ключа в дереве нет и удалять нам нечего
 - ◆ иначе (ключ был найден) выполним операцию $Concat$ над левым и правым сыновьями корня, а корень удалим

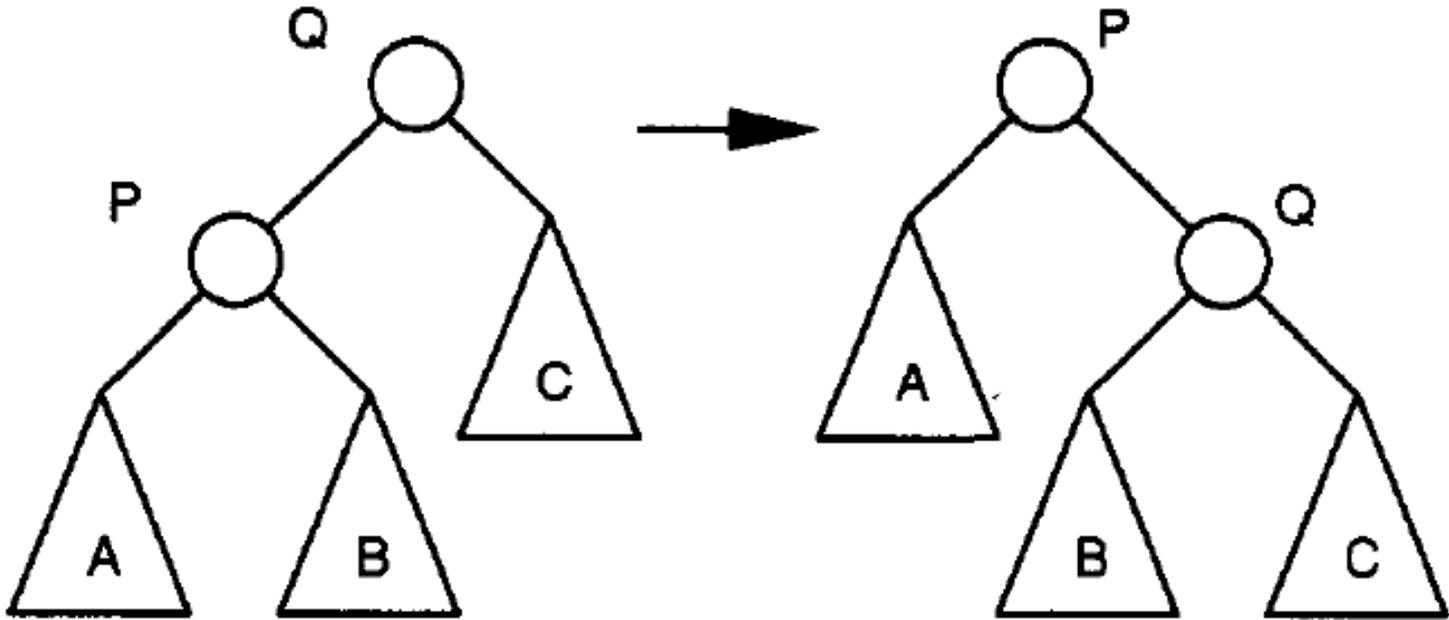


Реализация операции *splay*

- ◇ Шаг 1: ищем ключ K в дереве обычным способом, запоминая пройденный путь по дереву
 - ◆ Может потребоваться память, линейная от количества узлов дерева
 - ◆ Для уменьшения количества памяти можно воспользоваться *инверсией ссылок* (link inversion)
 - перенаправление указателей на сына назад на родителя вдоль пути по дереву плюс 1 бит на обозначение направления
- ◇ Шаг 2: получаем указатель P на узел дерева либо с ключом K , либо с его соседом в симметричном порядке обхода, на котором закончился поиск (сосед имеет единственного сына)
- ◇ Шаг 3: возвращаемся назад вдоль запомненного пути, перемещая узел P к корню (узел P будет новым корнем)

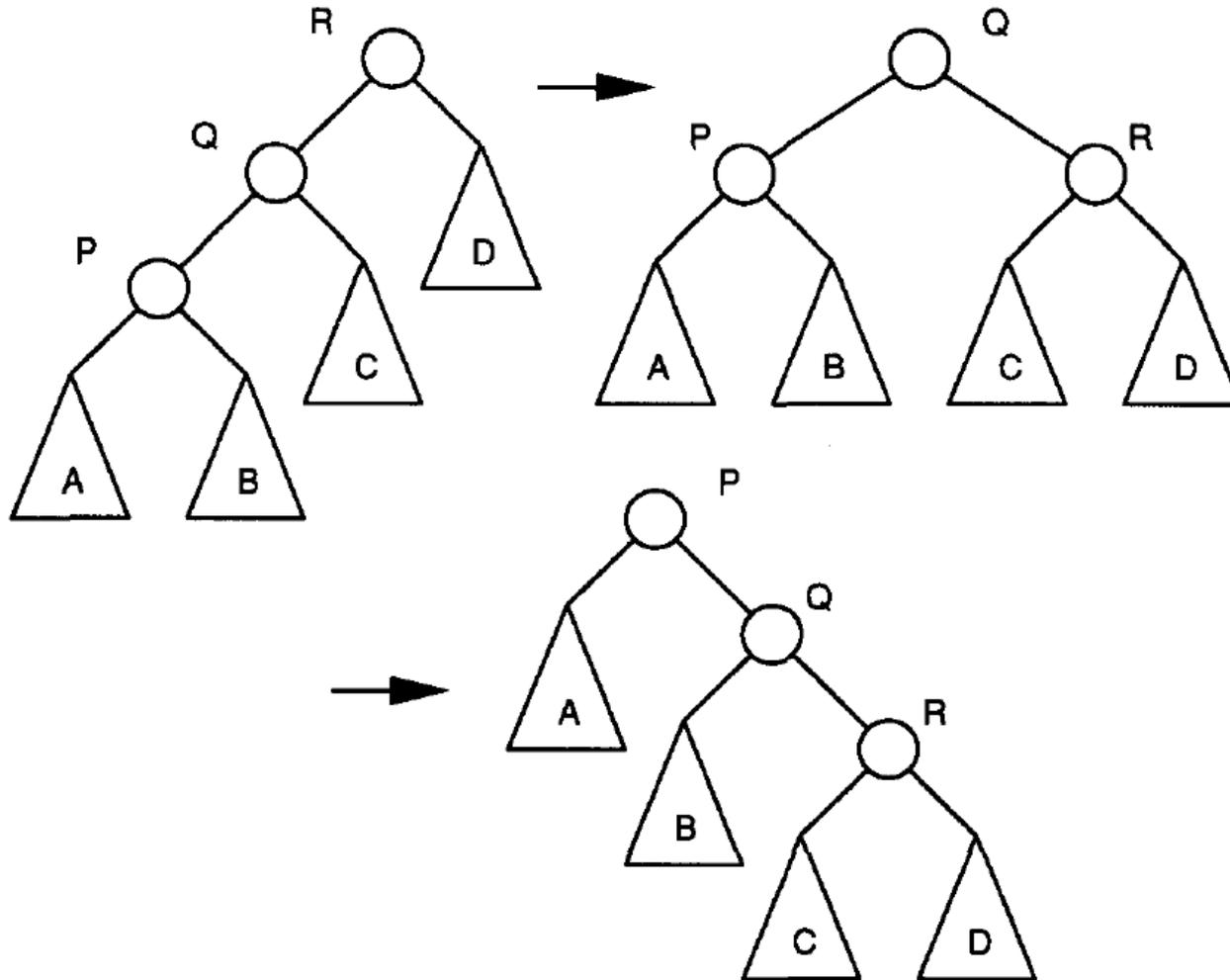
Реализация операции *splay*

- Шаг 3а): отец узла P – корень дерева (или у P нет деда)
 - выполняем однократный поворот налево или направо



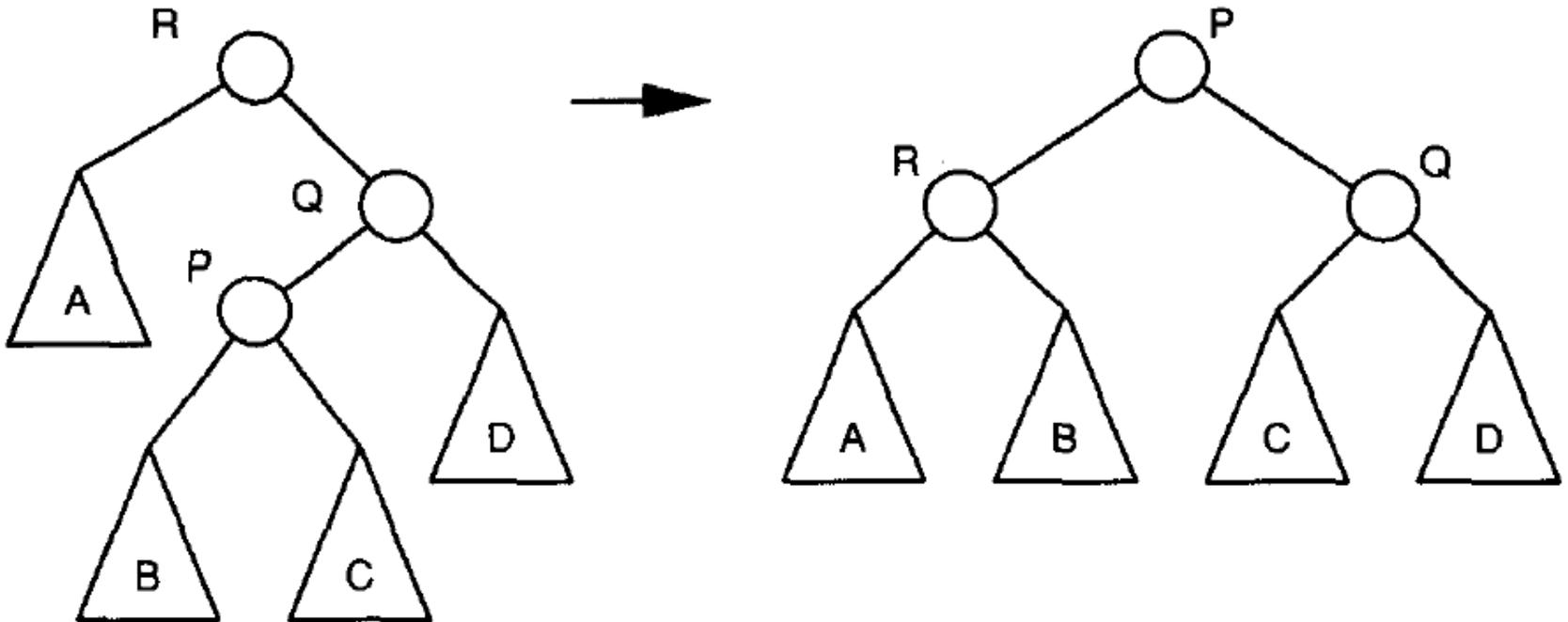
Реализация операции *splay*

- Шаг 3б): узел P и отец узла P – оба левые или правые дети
 - выполняем два однократных поворота направо (налево), сначала вокруг деда P , потом вокруг отца P



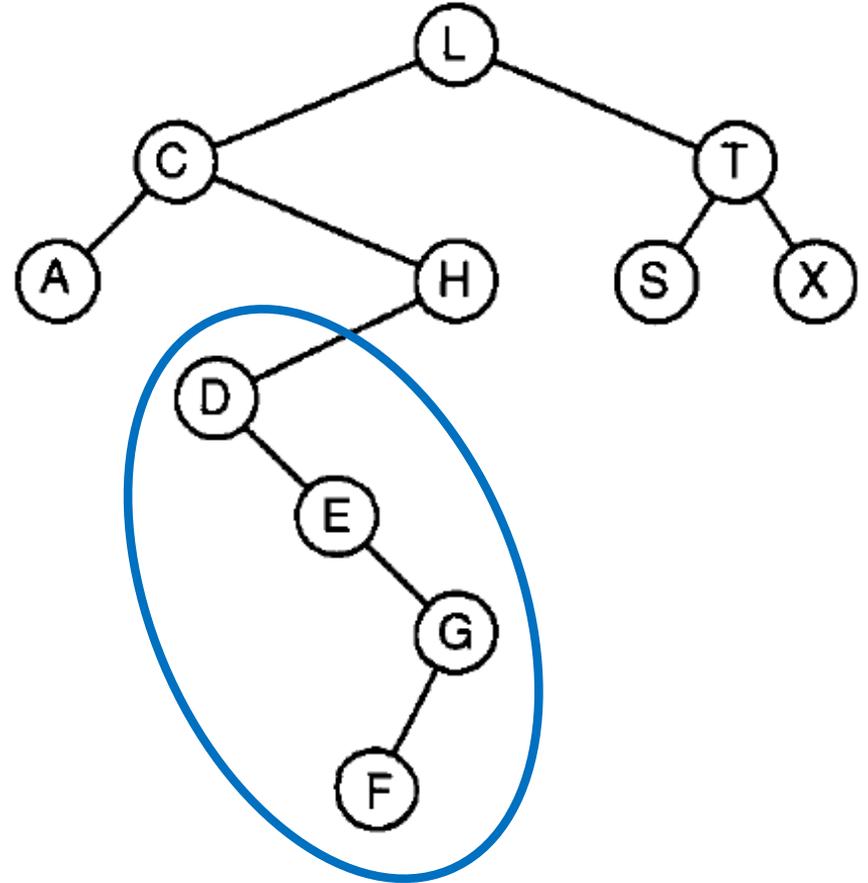
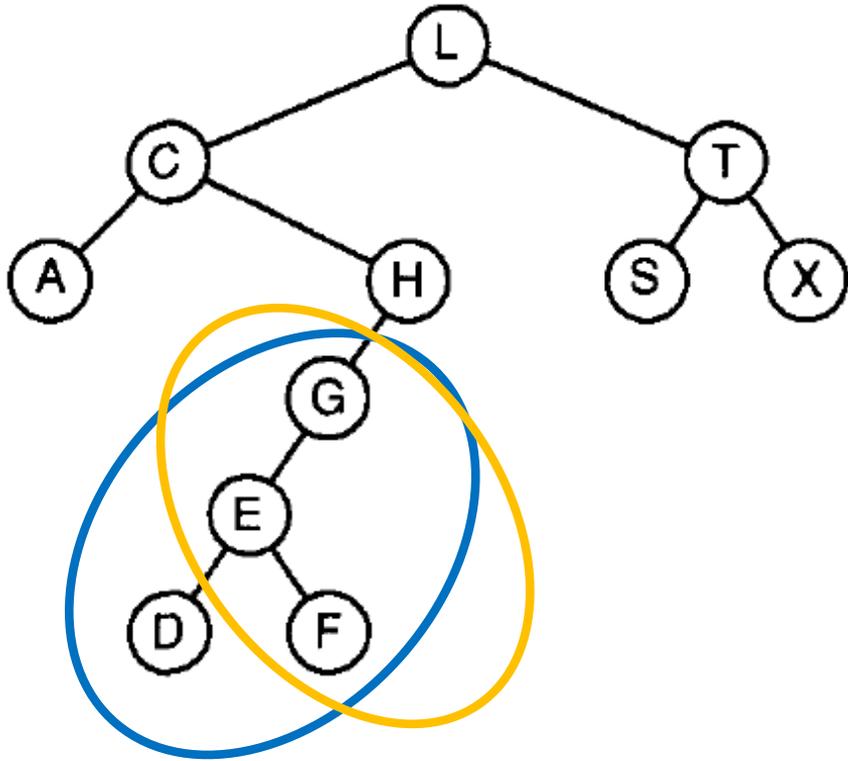
Реализация операции *splay*

- ♦ Шаг 3в): отец узла P – правый сын, а P – левый сын (или наоборот)
 - ♦ выполняем два однократных поворота в противоположных направлениях (сначала вокруг отца P направо, потом вокруг деда P налево)



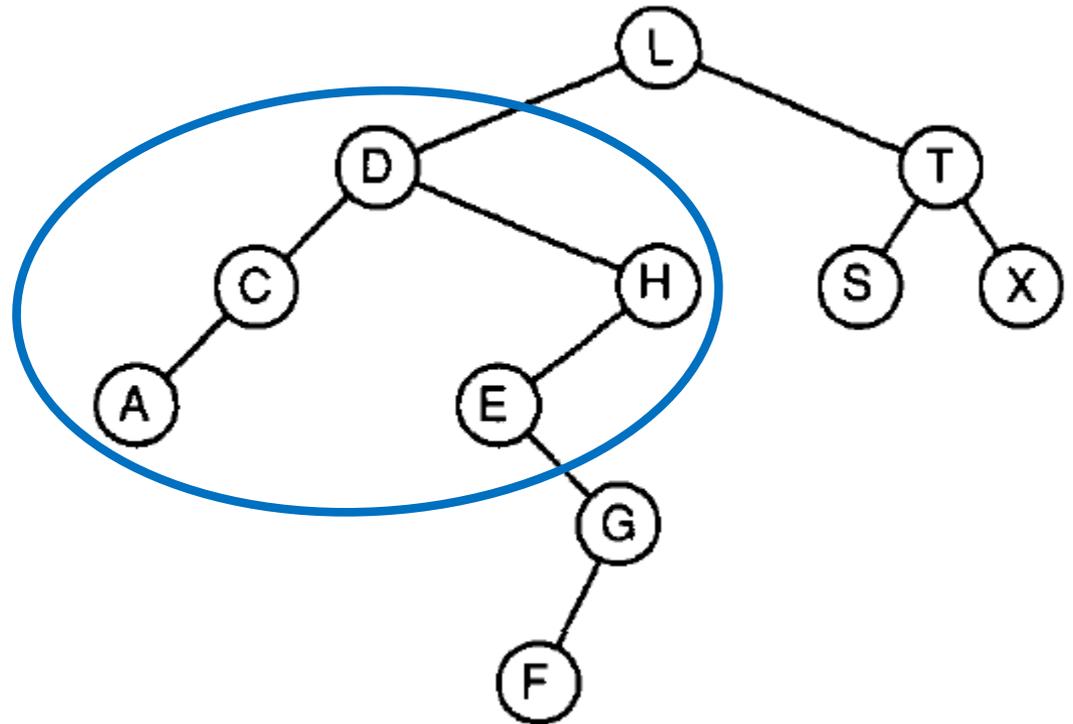
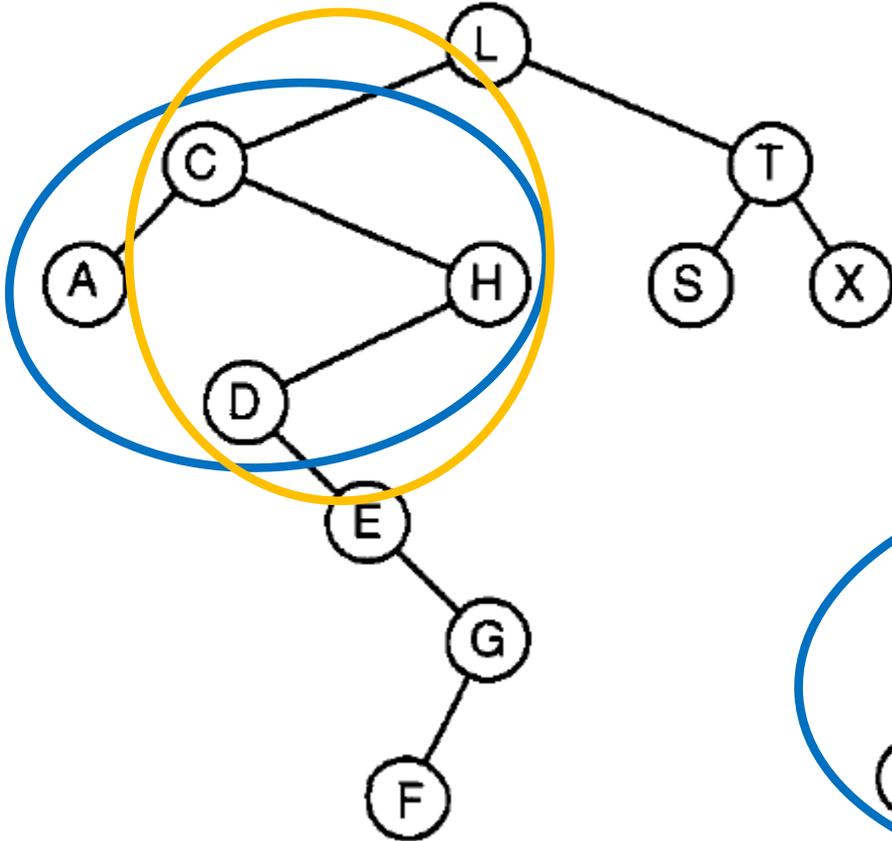
Пример операции *splay* над узлом *D*

◇ Случай б): отец узла *D* (*E*) и сам узел *D* – оба левые сыновья



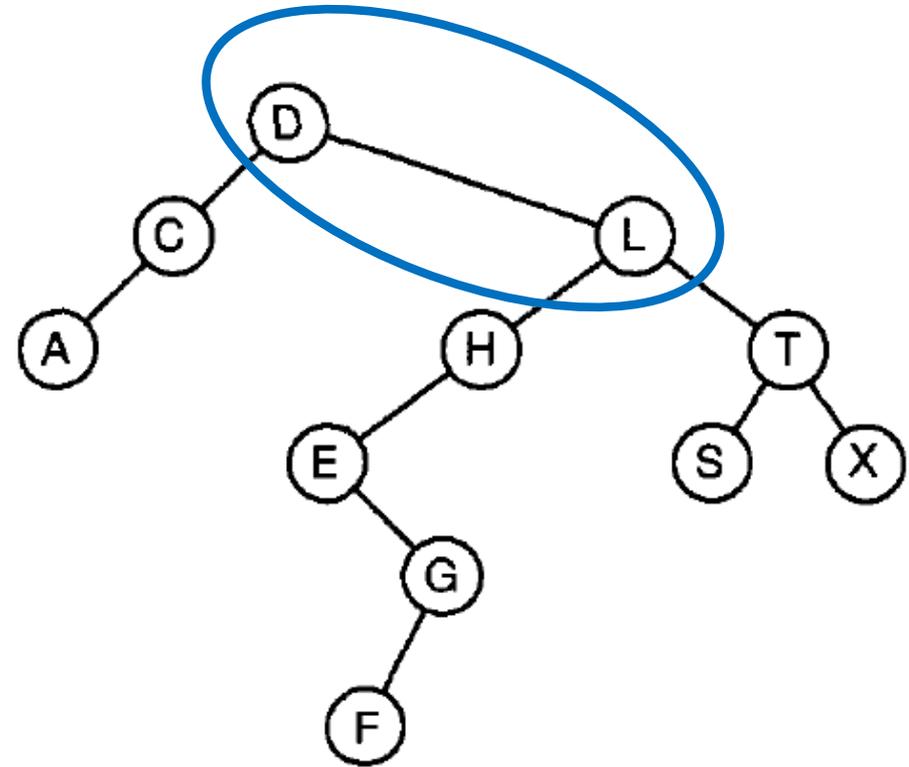
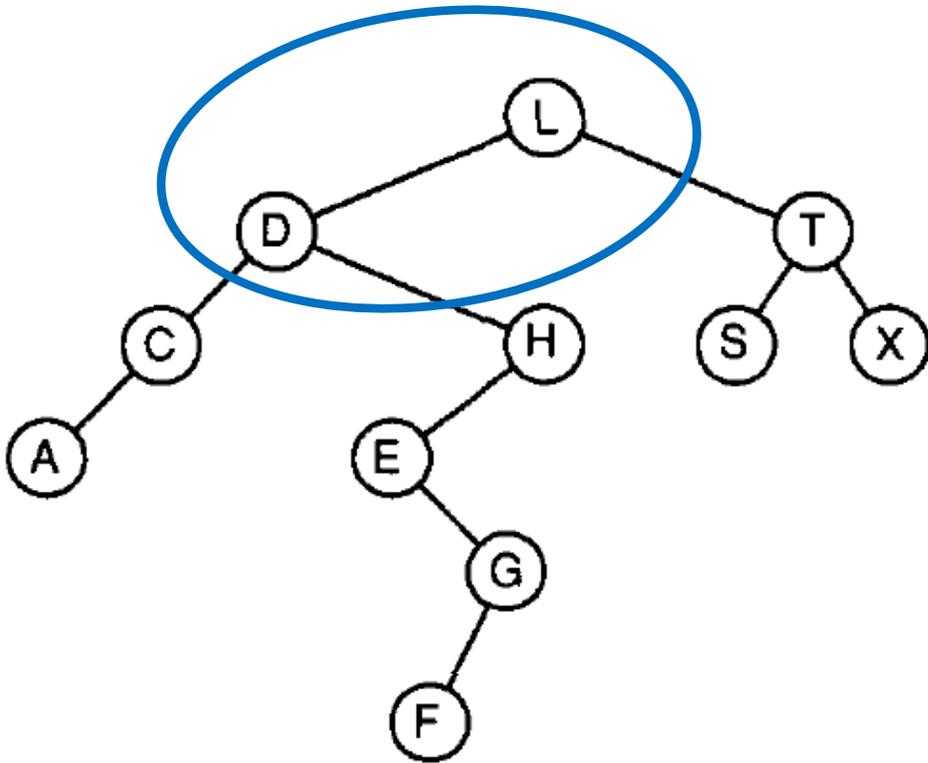
Пример операции *splay* над узлом *D*

◇ Случай в): отец узла *D* (*H*) – правый сын, а сам узел *D* – левый сын



Пример операции *splay* над узлом *D*

◇ Случай а): отец узла *D* (*L*) – корень дерева



Сложность операции *splay*

- ◇ Пусть каждый узел дерева содержит некоторую сумму денег.
 - ◆ Весом узла является количество ее потомков, включая сам узел
 - ◆ Рангом узла $r(N)$ называется логарифм ее веса
 - ◆ Денежный инвариант: во время всех операций с деревом каждый узел содержит $r(N)$ рублей
 - ◆ Каждая операция с деревом стоит фиксированную сумму за единицу времени
- ◇ Лемма. Операция *splay* требует *инвестирования* не более чем в $3\lfloor \lg n \rfloor + 1$ рублей с **сохранением** денежного инварианта.
- ◇ Теорема. Любая последовательность из m словарных операций на самоперестраивающемся дереве, которое было изначально пусто и на каждом шаге содержало не более n узлов, занимает не более $O(m \log n)$ времени.
 - ◆ Каждая операция требует не более $O(\log n)$ инвестиций, при этом может использовать деньги узла
 - ◆ По лемме инвестируется всего не более $m(3\lfloor \lg n \rfloor + 1)$ рублей, сначала дерево содержит 0 рублей, в конце содержит ≥ 0 рублей – $O(m \log n)$ хватает на все операции.