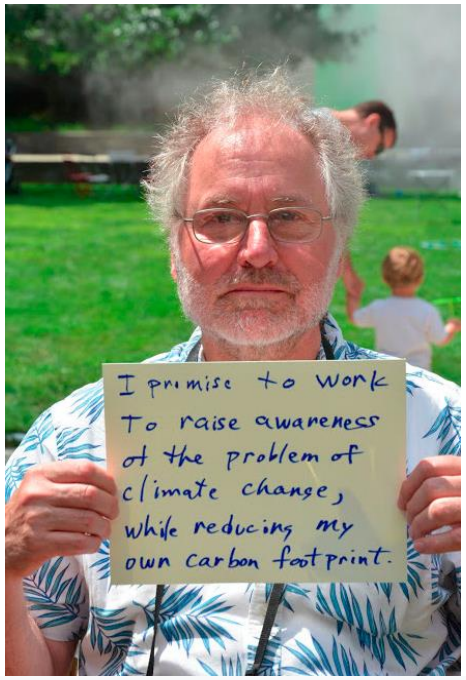
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Splay-дерево

ФПМИ Жидович М.С., 2022 год

Splay-дерево является двоичным деревом поиска. Это дерево принадлежит классу «саморегулирующихся деревьев», которые поддерживают необходимый баланс ветвления дерева, чтобы обеспечить выполнение операций поиска, добавления и удаления за логарифмическое время от числа хранимых элементов. Это реализуется без использования каких-либо дополнительных полей в узлах дерева (как, например, в Красно-чёрных деревьях или АВЛ-деревьях, где в вершинах хранится, соответственно, цвет вершины и глубина поддерева). Вместо этого «расширяющие операции» (splay operation), частью которых являются вращения, выполняются при каждом обращении к дереву.

Born	April 30, 1948 (age 73) Pomona, California
Citizenship	American
Alma mater	California Institute of Technology (BS) Stanford University (MS, PhD)
Known for	Algorithms and data structures
Awards	Paris Kanellakis Award (1999) Turing Award (1986) Nevanlinna Prize (1982)



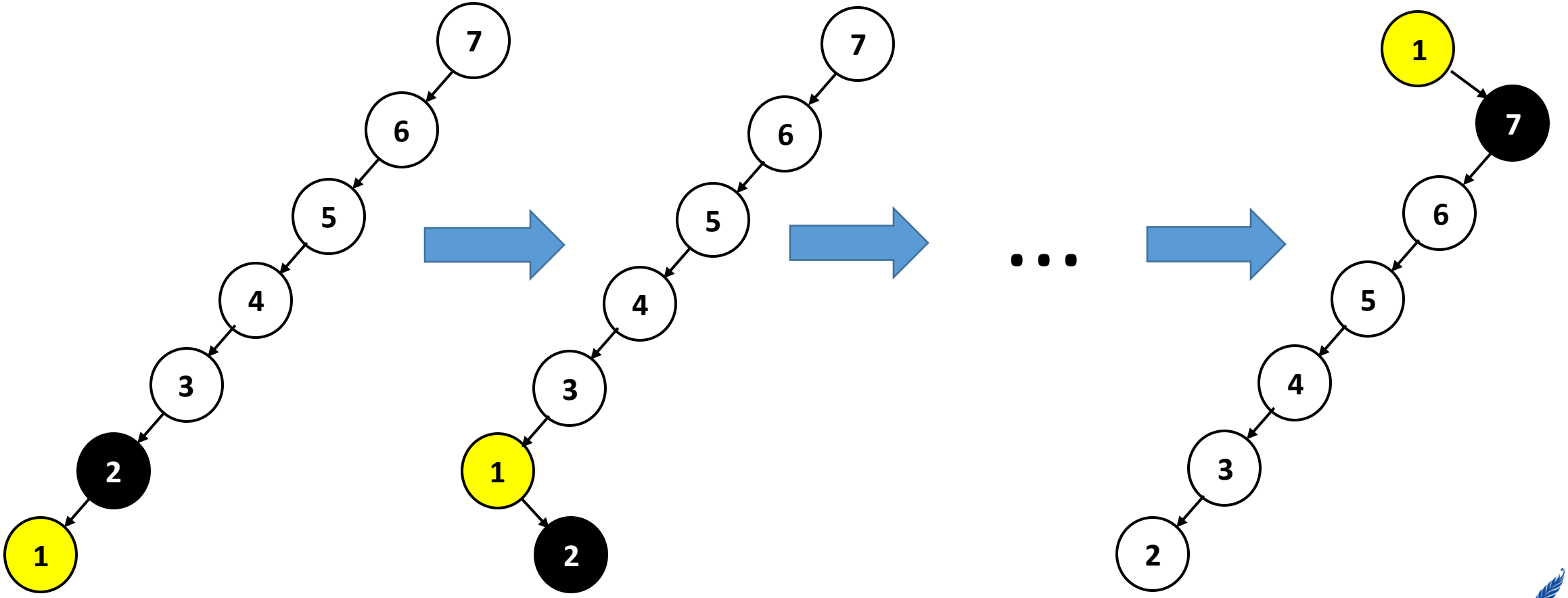
Born	10 December 1953 (age 68) St. Louis,[1] Missouri
Alma mater	University of Illinois at Urbana–Champaign, Stanford University
Awards	Paris Kanellakis Award (1999)
Scientific career	
Fields	Computer science
Institutions	Carnegie Mellon University
Doctoral advisor	Robert Tarjan

Splay-дерево было придумано Робертом Тарьяном и Даниелем Слейтером в 1983 году.

Splay-дерево позволяет находить быстрее те данные, которые использовались недавно. Для того, чтобы доступ к недавно найденным данным был быстрее, надо, чтобы эти данные находились ближе к корню. Этого мы можем добиться, используя различные эвристики:

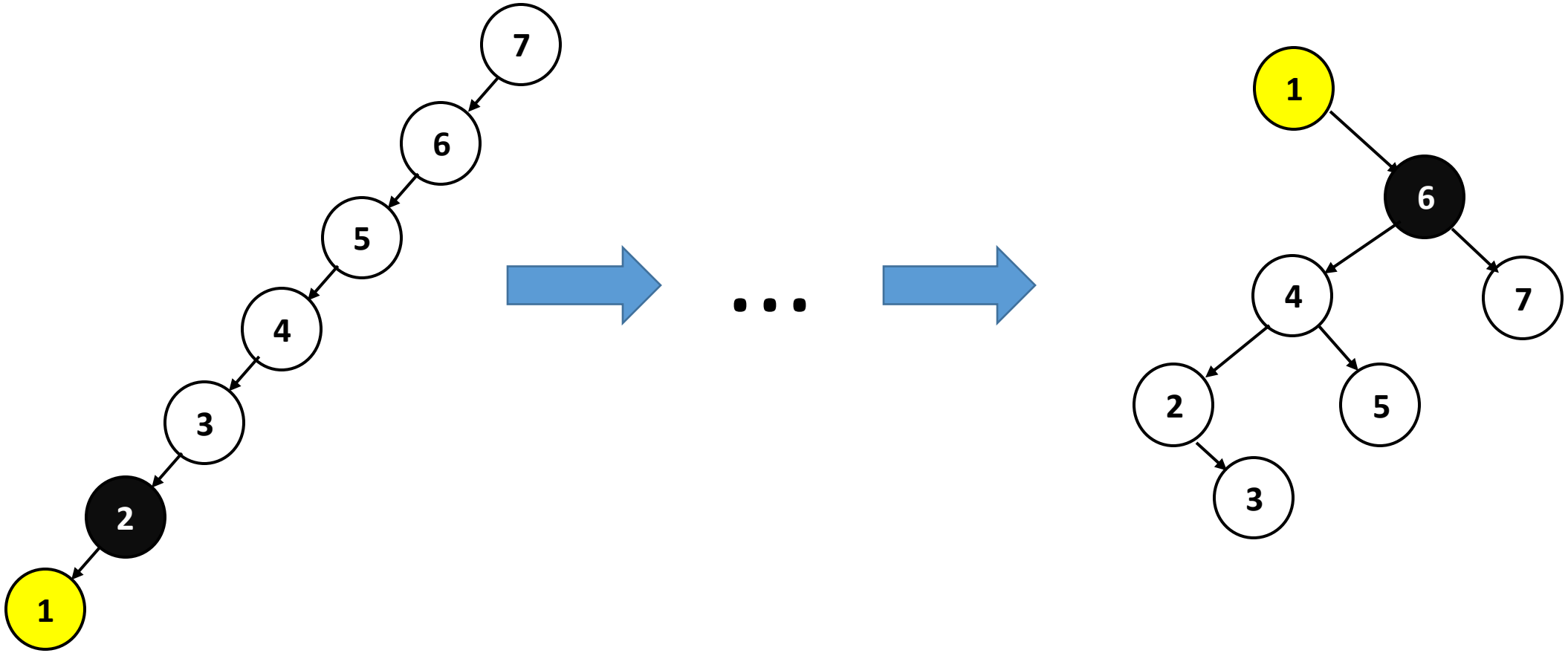
Move to Root

Совершает повороты вокруг ребра (v, parent(v)), пока v не окажется корнем дерева.



Splay

Также совершает повороты, но чередует различные виды поворотов, благодаря чему достигается логарифмическая амортизированная оценка. Будет подробно рассмотрена далее.



При последовательном использовании операций "Move to Root" требуется 6 поворотов, в то время как при использовании операции "Splay" достаточно 3 поворотов.

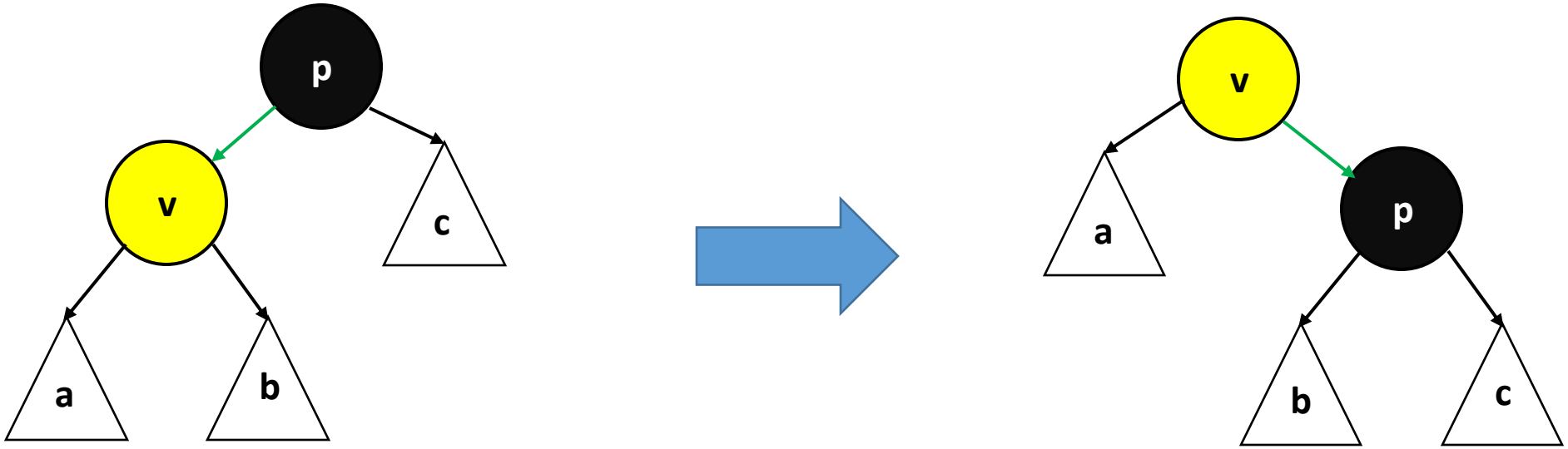
Операции со Splay-деревом

Splay

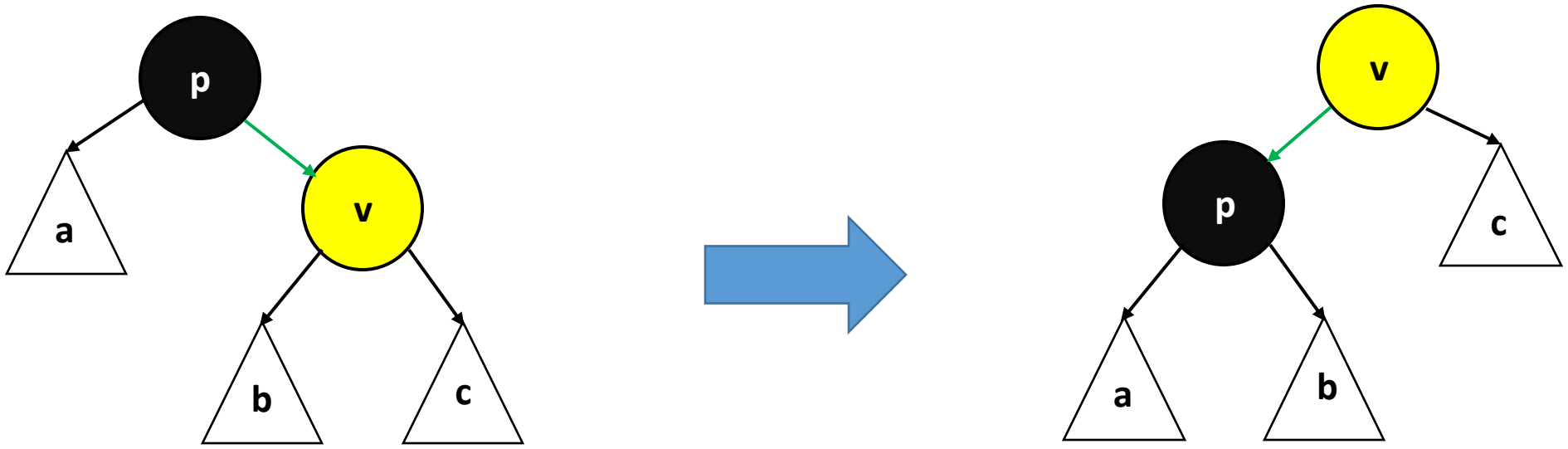
Делится на несколько случаев:

Zig (LL поворот, Правый разворот, Малое правое вращение)
Zag (RR поворот, Левый разворот, Малое левое вращение)
Zig-zig (Левый-левый случай, два разворота вправо)
Zag-zag (Правый-правый случай, два разворота влево)
Zig-zag (LR поворот, Большое правое вращение)
Zag-zig (RL-поворот, Большое левое вращение)

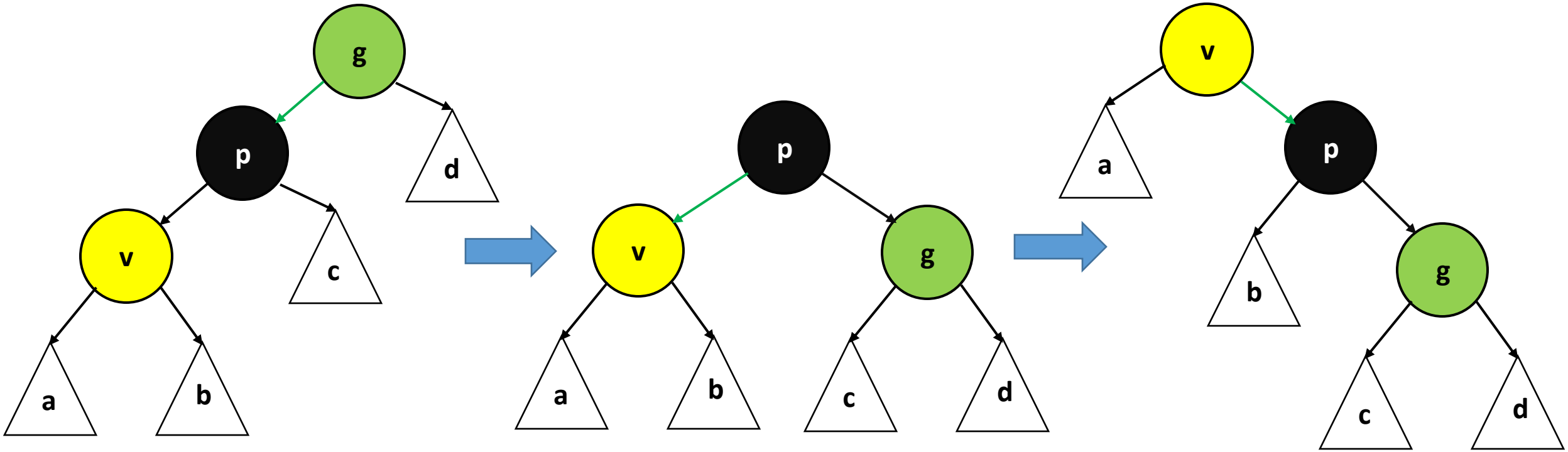
Zig (LL поворот, Правый разворот)



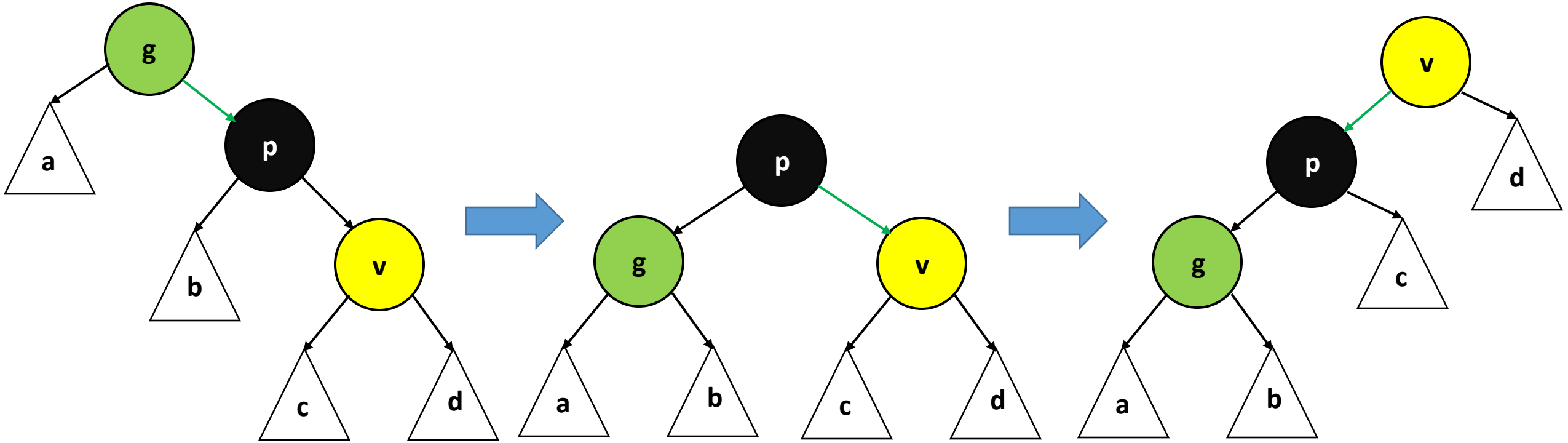
Zag (RR поворот, Левый разворот)



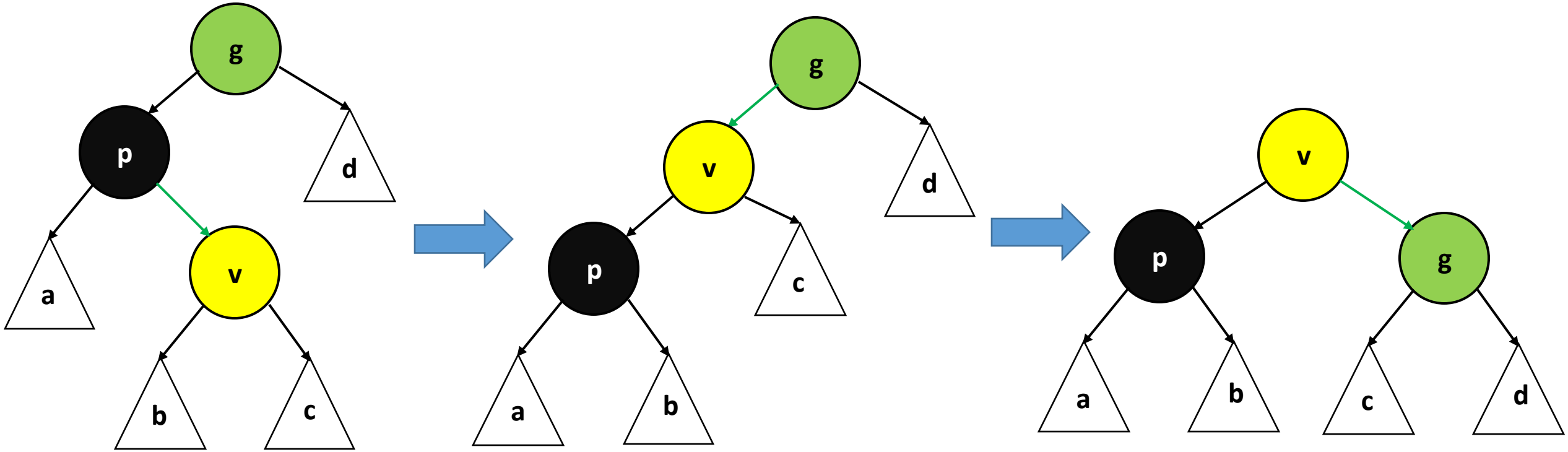
Zig-zig (Левый-левый случай, два разворота вправо)



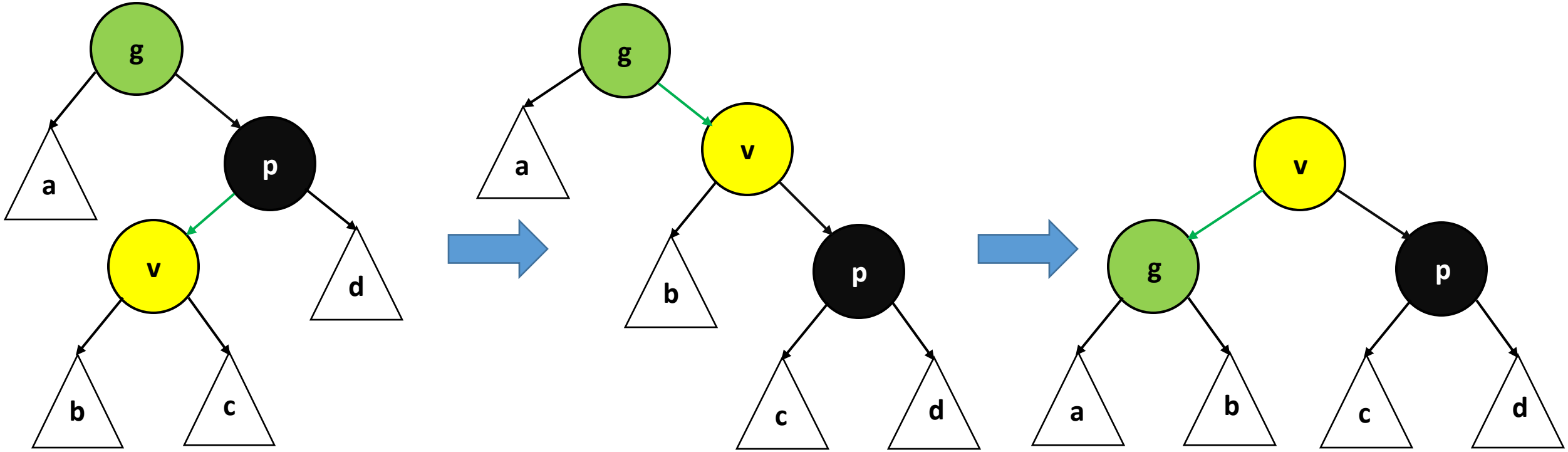
Zag-zag (Правый-правый случай, два разворота влево)



Zig-zag (LR поворот, Большое правое вращение)



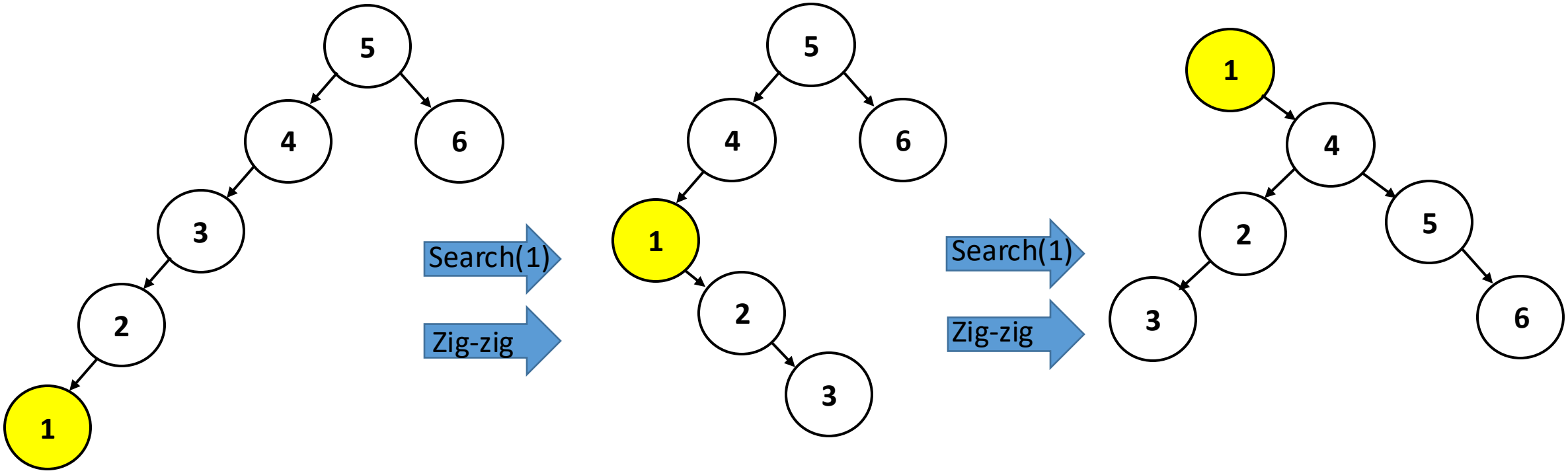
Zag-zig (RL-поворот, Большое левое вращение)



Данная операция занимает $O(h)$ времени, где h — глубина вершины v.

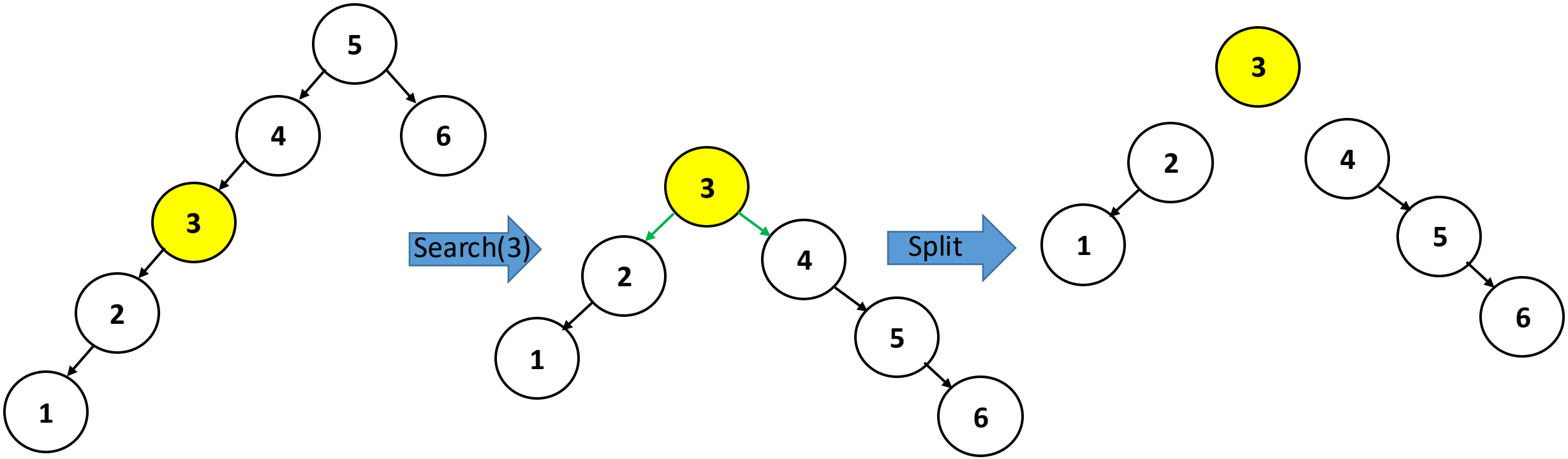
Search

Эта операция выполняется как для обычного бинарного дерева поиска, только после нее запускается операция Splay.



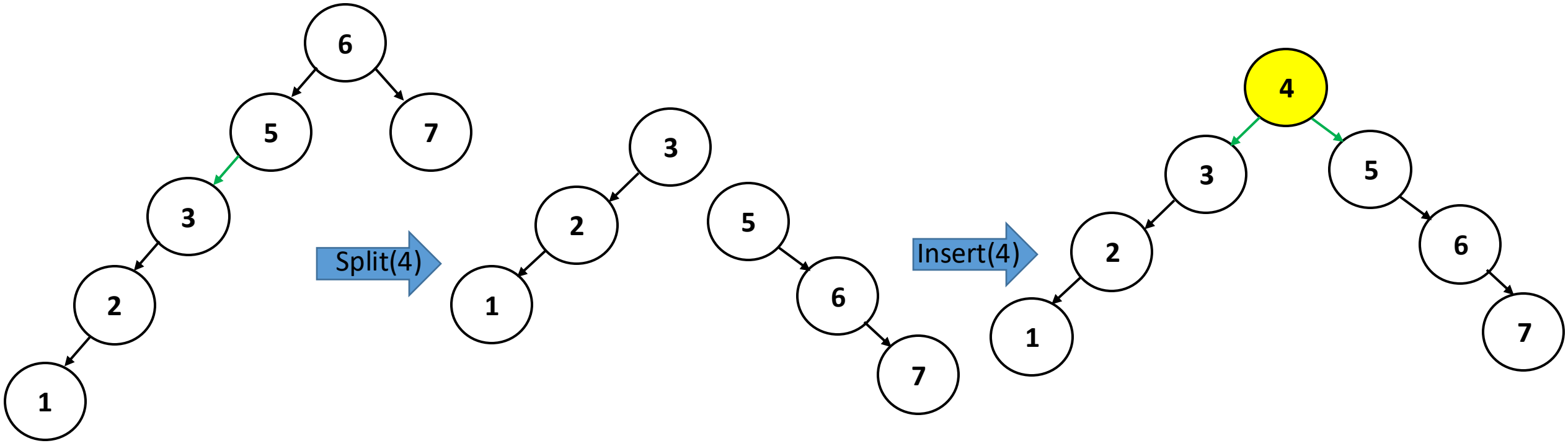
Split

Процедура Split получает на вход ключ и делит дерево на два. В одном дереве все значения меньше ключа, а в другом — больше. Реализуется она просто. Нужно через Search найти ближайшую к ключу вершину, вытянуть ее вверх и потом отрезать либо левое, либо правое поддерево (либо оба).



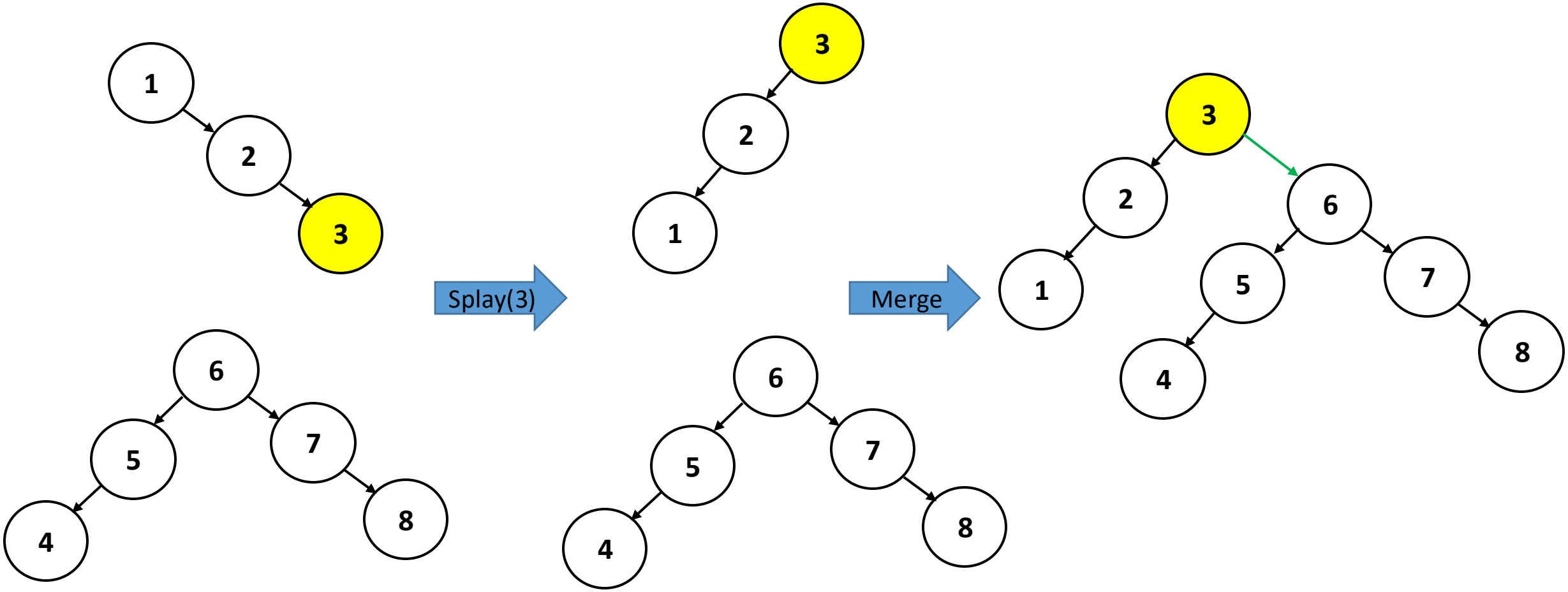
Insert

Чтобы вставить очередной ключ, достаточно вызвать Split по нему, а затем создать новую вершину-корень и полученные деревья подвесить к ней как левое и правое поддеревья соответственно.



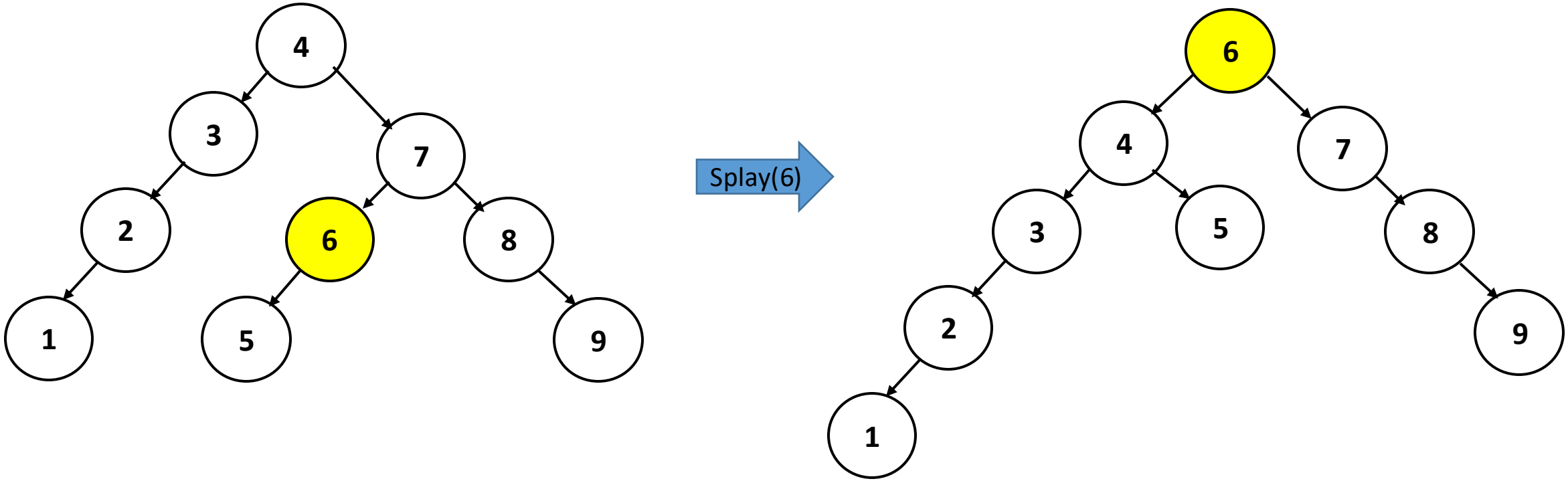
Merge

У нас есть два дерева, причём подразумевается, что все элементы первого дерева меньше элементов второго. Запускаем Splay от самого большого элемента в первом дереве. Элемент становится корнем, при этом у него нет правого ребёнка. Ставим на его место второе дерево и возвращаем полученное дерево.



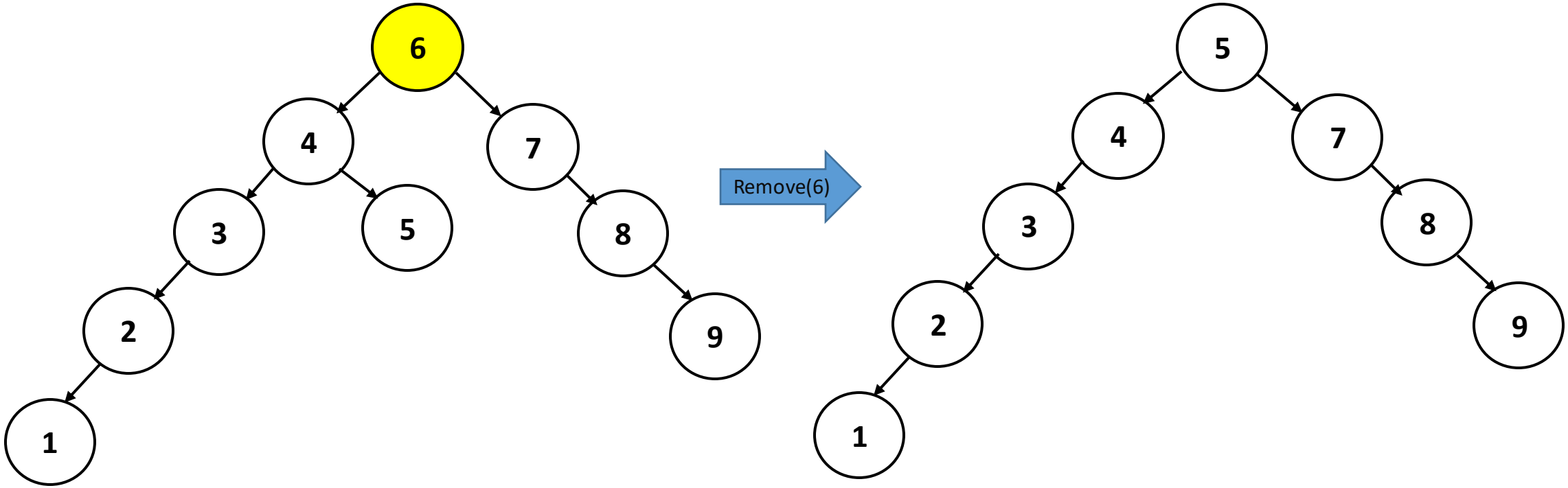
Remove

Для того, чтобы удалить вершину, поднимем ее вверх, а потом выполним операцию Merge для её левого и правого поддеревьев.



Remove(продолжение)

Для того, чтобы удалить вершину, поднимем ее вверх, а потом выполним операцию Merge для её левого и правого поддеревьев.



Чтобы Splay-дерево поддерживало повторяющиеся ключи, можно поступить двумя способами. Нужно либо каждому ключу сопоставить список, хранящий нужную доп. информацию, либо реализовать операцию Search так, чтобы она возвращала первую (в порядке внутреннего обхода) вершину с ключом, большим либо равным заданного.

Заметим, что процедуры удаления, вставки, слияния и разделения деревьев работают за O(1) + время работы операции Search.

Процедура Search работает пропорционально глубине искомой вершины в дереве. По завершении поиска запускается процедура Splay, которая тоже работает пропорционально глубине вершины. Таким образом, достаточно оценить время работы процедуры Splay.

Амортизационный анализ Splay-дерева проводится с помощью метода потенциалов. Потенциалом рассматриваемого дерева назовём сумму рангов его вершин. Ранг вершины v — это величина, обозначаемая *rank*(v) и равная logS(v), где S(v) — количество вершин в поддереве с корнем в v.

Лемма

Амортизационная стоимость операции Splay от вершины v в дереве с корнем r составляет 3(rank(r) - rank(v)) + 1.

Доказательство

Рассмотрим идею доказательства.

Если $v=r$, утверждение очевидно. В противном случае рассмотрим, как меняется ранг. Пусть изначально он равен $rank(v_0)$, а после i-ой итерации - $rank(v_i)$. Для каждого этапа, кроме, быть может, последнего, мы покажем, что амортизационное время на его выполнение можно ограничить сверху величиной $3(rank(v_i)-rank(v_{i-1}))$. Для последнего этапа верхняя оценка составит $3(rank(v_i)-rank(v_{i-1}))+1$. Просуммировав верхние оценки и сократив промежуточные значения рангов мы получим требуемое:

$$1+\sum_{i=1}^{k} 3(rank(v_i)-rank(v_{i-1}))=1+3(rank(v_k)-rank(v_0))=1+3(rank(r)-rank(v))$$

Для этого рассмотрим три вида поворотов и изменение ранга при их использовании: Zig, Zig-zig, Zig-zag(Zag, Zag-zag, Zag-zig аналогично).

Zig(может выполняться только в конце)

Ранги меняются только у вершин v и r

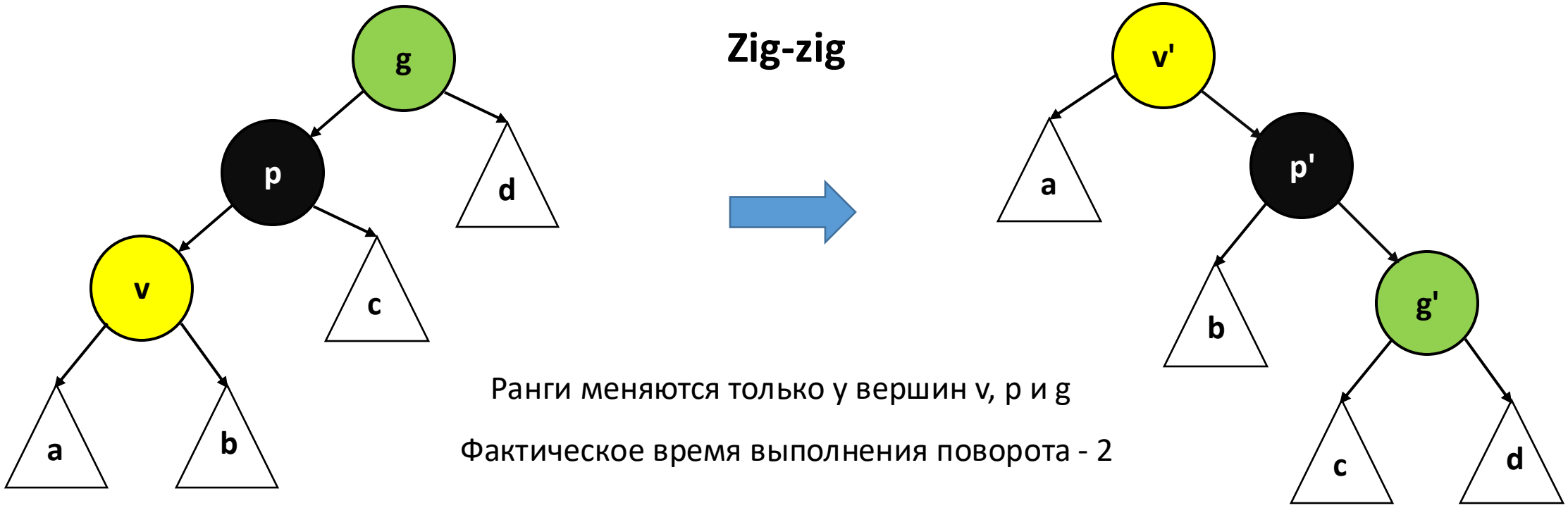
Фактическое время выполнения поворота - 1

$$1+rank(v')+rank(r')-rank(r)-rank(v)=[rank(v')=rank(r)]=1+rank(r')-rank(v)$$

Используем размеры поддеревьев

$$1+rank(r')-rank(v)\le[rank(r')\le rank(v')]\le 1+rank(v')-rank(v)\le 1+3(rank(v')-rank(v))$$

Zig-zig



Ранги меняются только у вершин v, p и g

Фактическое время выполнения поворота - 2

$$2+rank(v')+rank(p')+rank(g')-rank(v)-rank(p)-rank(g)=[rank(v')=rank(g)]=$$

$$=2+rank(p')+rank(g')-rank(v)-rank(p)=[\text{размеры поддеревьев}: rank(p')\le rank(v'), -rank(p)\le -rank(v)]=$$

$$\le 2+rank(v')+rank(g')-2rank(v)$$

Теперь нам осталось показать, что: $-2+2rank(v')-rank(g')-rank(v)\ge 0$ **Тогда получим:**

$$(2+rank(v')+rank(g')-2rank(v))+(-2+2rank(v')-rank(g')-rank(v))=3(rank(v')-rank(v))$$

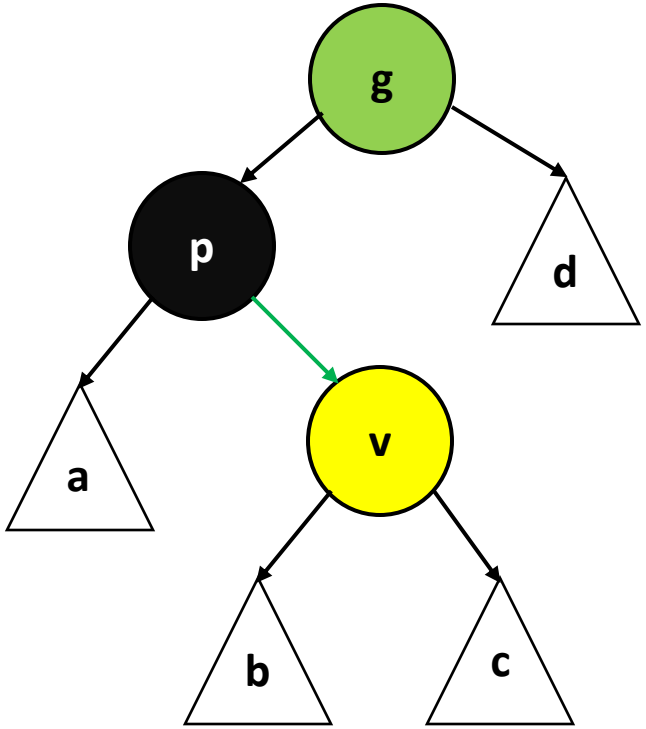
$$\boxed{rank(g')+rank(v)-2\,rank(v')\le -2}$$

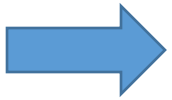
$$rank(g')+rank(v)-2\,rank(v')=\log s(g')+\log s(v)-2\log s(v')=\log\frac{s(g')}{s(v')}+\log\frac{s(v)}{s(v')}$$

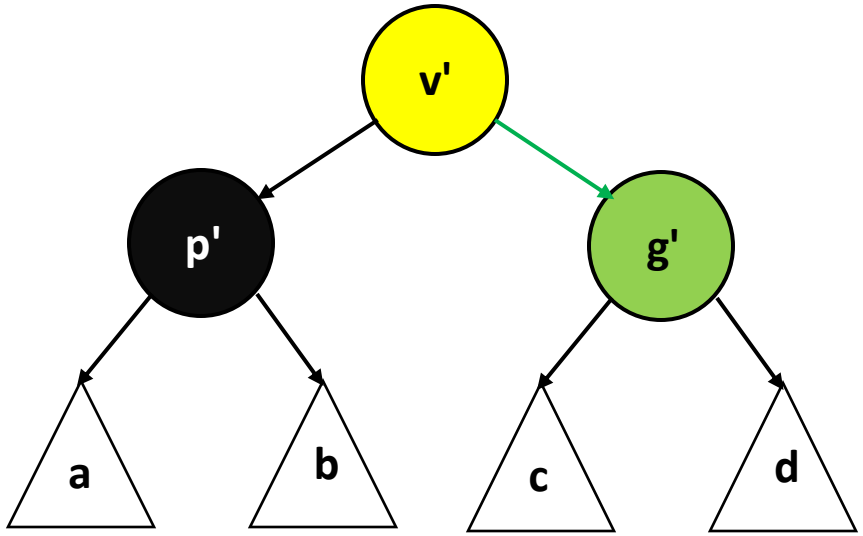
Взглянув на диаграмму, заметим, что $s(v')=s(g')+s(v)+1\Rightarrow\frac{s(g')}{s(v')}+\frac{s(v)}{s(v')}=1-\frac{1}{s(v')}\le 1$

Тогда: $\log\frac{s(g')}{s(v')}+\log\frac{s(v)}{s(v')}\le -2$ Что и требовалось доказать.

Zig-zag







Ранги меняются только у вершин v, p и g

Фактическое время выполнения поворота - 2

$$2+rank(v')+rank(p')+rank(g')-rank(v)-rank(p)-rank(g)=[rank(v')=rank(g)]=$$

$$=2+rank(p')+rank(g')-rank(v)-rank(p)\le[-rank(p)\le-rank(v)]\le2+rank(p')+rank(g')-2\,rank(v)$$

Далее аналогично предыдущему случаю

Таким образом мы разобрали все три случая и получили верхнюю оценку на амортизированное время через ранги.
Заметим, что ранг любой вершины ограничен логарифмом размера дерева. Делаем вывод, что операция Splay амортизационно выполняется за O(logn).

Теорема о статической оптимальности

Пусть — q_i число раз, которое был запрошен элемент . Тогда выполнение запросов поиска на Splay-дереве выполняется за

$$O\left(m+\sum_i q_i \log \frac{m}{q_i}\right)$$

По сути этот факт сообщает следующее. Пусть мы заранее знаем, в каком количестве будут заданы запросы для различных элементов. Мы строим какое-то конкретное бинарное дерево, чтобы отвечать на эти запросы как можно быстрее. Утверждение сообщает, что с точностью до константы Splay-дерево будет амортизационно работать не хуже, чем самое оптимальное фиксированное дерево, которое мы можем придумать.

Теорема о текущем множестве

Пусть — t_i это число запросов, которое мы уже совершили с момента последнего запроса к элементу i ; если еще не обращались, то просто число запросов с самого начала. Тогда время обработки запросов составит

$$O\left(m+n\log n+\sum_i \log\left(t_i+1\right)\right)$$

Этот результат говорит о том, что в среднем недавно запрошенный элемент не уплывает далеко от корня.

Исследование производительности и области применения Splay-деревьев оказалась темой для десятка статей. Часть таких исследований показывает, что в сравнении с другими сбалансированными деревьями они проявляют себя наилучшим образом на практике. Высокая производительность объясняется особенностью реальных данных. Представьте себе ситуацию, когда у нас есть миллионы или даже миллиарды ключей, и лишь к некоторым из них обращаются регулярно, что весьма вероятно для многих типичных приложениях (в среднестатистическом приложении 80% обращений приходятся на 20% элементов).

Splay-деревья стали наиболее широко используемой базовой структурой данных, изобретенной за последние 30 лет, потому что они являются самым быстрым типом сбалансированного дерева поиска для огромного множества приложений.

Splay-деревья используются в Windows NT (в виртуальной памяти, сети и коде файловой системы), компиляторе gcc и библиотеке GNU C++, редакторе строк sed, сетевых маршрутизаторах Fore Systems, наиболее популярной реализации Unix malloc, загружаемых модулях ядра Linux и во многих других программах

Недостатки Splay-дерева

Главный недостаток заключается в том, что высота дерева всё-таки может быть линейной. Например, если выполнить операцию Splay для всех элементов в неубывающем порядке. В таком случае время выполнения операций - O(n).

Однако амортизированная стоимость остаётся логарифмической. Проще говоря, при использовании Splay-дерева мы можем быть уверены, что m операций будут выполнены за время O(mlogn) при достаточно большом m.

Кроме того, изменение структуры дерева во время выполнения поиска усложняет его использование в многопоточной среде(представьте ситуацию, когда несколько потоков выполняют поиск одновременно).

Код реализации Splay-дерева можно найти по ссылке:

https://github.com/magziim/DSA/blob/main/splay_tree.py

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Спасибо за внимание!