Дискретная математика

Часть 2

Кочетов Юрий Андреевич

http://www.math.nsc.ru/LBRT/k5/dm.html

Лекция 1

Алгоритмы, сортировки, AVL–деревья

Алгоритмы и их сложность

Компьютеры выполняют (пока) лишь корректно поставленные задачи. В частности, они выполняют *алгоритмы* — точно и однозначно понимаемые последовательности команд, при помощи которых решаются определенные вычислительные задачи.

Существуют разные определения понятия алгоритма: рекурсивные функции, машины Тьюринга–Поста, нормальные алгоритмы Маркова и др. Однако множество функций, вычислимых с помощью разных определений алгоритма фактически совпадают.

Пример. Пусть в полном n–вершинном графе каждому ребру сопоставлен вес. Требуется найти каркас (остовное дерево), сумма весов ребер которого минимальна.

Алгоритм: перебрать все каркасы и выбрать среди них каркас минимального веса.

Такой алгоритм требует просмотра n^{n-2} каркасов, что уже при $n \geq 40$ больше 10^{60} вариантов.

Мера качества алгоритма — отрезок времени, требуемый для получения ответа. Будем оценивать этот отрезок в терминах числа арифметических операций (сложения, сравнения и др.)

Для простоты будем считать, что элементарные операции требуют одинаковое время. Будем оценивать $f_A(n)$ — максимальное время работы алгоритма A по всем входам длины не более n.

Нас интересует порядок роста функции $f_A(n)$, а не точные её значения. Будем говорить, что алгоритм *полиномиален*, если для некоторого фиксированного k

$$f_A(n) = O(n^k).$$

Пример. $f_A(n) = O(n^{n-2})$, алгоритм перебора всех каркасов не является полиномиальным.

Как измерять длину входа?

Длиной целого числа m естественно считать $\log m$. Если граф (орграф) G задан матрицей смежности, то длина записи есть $O(n^2)$, где n — число вершин G; если списком смежностей — то $O(m\log n + n)$, где m — число ребер G.

Если дана целочисленная матрица $A = (a_{ij})_{i,j=1}^{n}$, то длина ее записи есть $O(n^2 + \sum_{i,j} \log(a_{ij}+1))$. Но поскольку многие современные ЭВМ отводят разным числам примерно одинаковое место, мы будем считать, что длина записи информации о графе G, заданном списком смежностей, есть $O(m+n)$, где m — число ребер, а n — число вершин графа G.

Быстрая сортировка

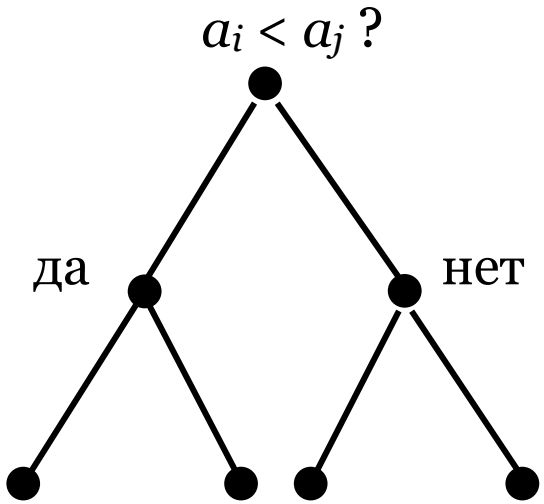
Сортировкой называют упорядочение множества объектов по неубыванию или невозрастанию какого-нибудь параметра.

- Алгоритм пузырька (волновой алгоритм) — $O(n^2)$.
- Алгоритм Фон–Неймана — $O(n \log n)$.
- Пирамидальный алгоритм — $O(n \log n)$.
- QuickSort и др.

Сортировка с помощью сравнений

Лемма 1. Бинарное дерево высоты h содержит не более 2^h листьев.

Дерево решений:



Лемма 2. Высота любого дерева решений, упорядочивающего последовательность из n различных элементов, не менее $\log n!$.

Доказательство. Так как результатом может быть любая из $n!$ перестановок, то в дереве решений должно быть не менее $n!$ листьев. Тогда по лемме 1 высота дерева не меньше $\log n!$. ■

Теорема. В любом алгоритме, упорядочивающем с помощью сравнений, на упорядочивание последовательность из n элементов тратится не менее $cn\log n$ сравнений при некотором $c>0$ и достаточно большом n.

Доказательство. При $n \geq 4$ имеем

$$n! \geq n(n-1)\,(n-2)\ldots(\lceil \tfrac{n}{2} \rceil) \geq (\tfrac{n}{2})^{n/2} \quad ,$$

тогда

$$\log n! \geq (\tfrac{n}{2}) \log (\tfrac{n}{2}) \geq (\tfrac{n}{4}) \log n. \quad \blacksquare$$

Алгоритм Фон–Неймана

На вход подается последовательность чисел $a(1),\ldots, a(n)$. Алгоритм работает $\lceil \log_2 n \rceil$ итераций. Перед началом итерации с номером k ($k = 1, 2, \ldots, \lceil \log_2 n \rceil$) имеется последовательность $a(i\,(1)), \ldots, a(i\,(n))$ тех же чисел, разбитая на группы по 2^{k-1} элементов (последняя группа может быть неполной). Внутри каждой группы элементы упорядочены по неубыванию. Итерация состоит в том, что эти группы разбиваются на пары соседних групп, и элементы упорядочиваются внутри этих новых в два раза больших групп. При этом используется то, что внутри исходных групп элементы уже упорядочены.

$$\left.\begin{matrix} x_1,\ldots,x_m \\ y_1,\ldots,y_m \end{matrix}\right\} \Rightarrow z_1,\ldots,z_{2m}$$

слияние за линейное время

$O(m)$

Упражнение. Показать, что алгоритм Фон–Неймана использует $O(n\lceil \log n \rceil)$ сравнений.

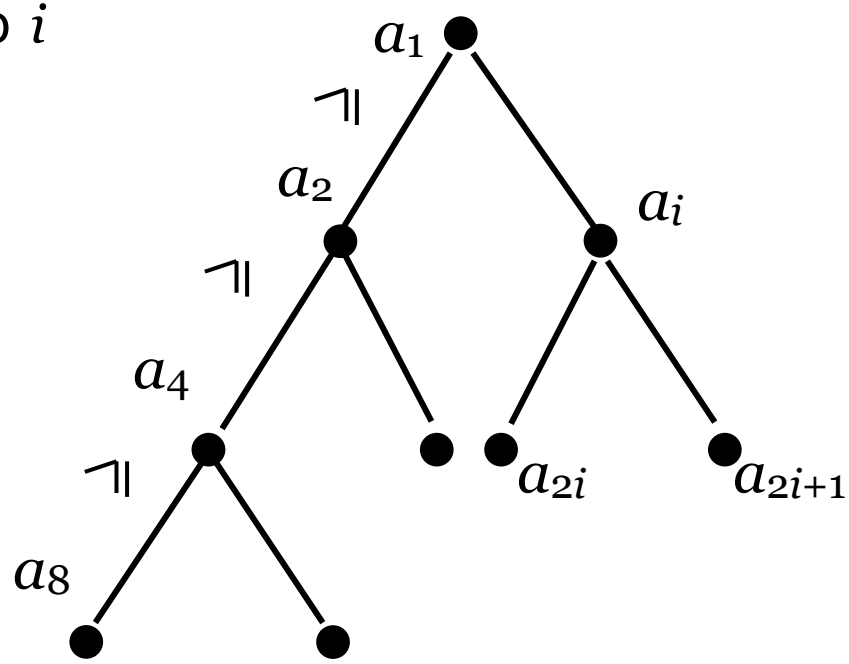
Пирамидальный алгоритм

Два этапа:

1) построение пирамиды: для каждого i

   $a_i \le a_{2i}$ и $a_i \le a_{2i+1}$

2) сортировка массива с помощью пирамиды



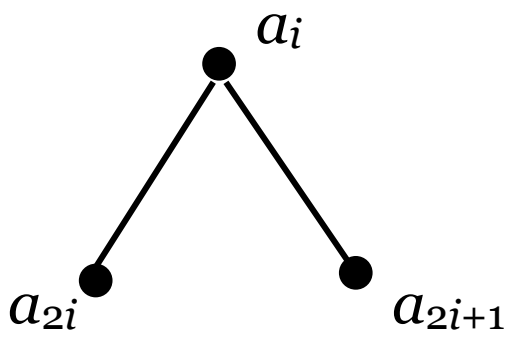
Первый этап

(i, n)–операция состоит в следующем:

Сравниваем a_{2i} и a_{2i+1},

пусть x — меньшее из них.

Если $a_i > x$, то меняем местами a_i и x.



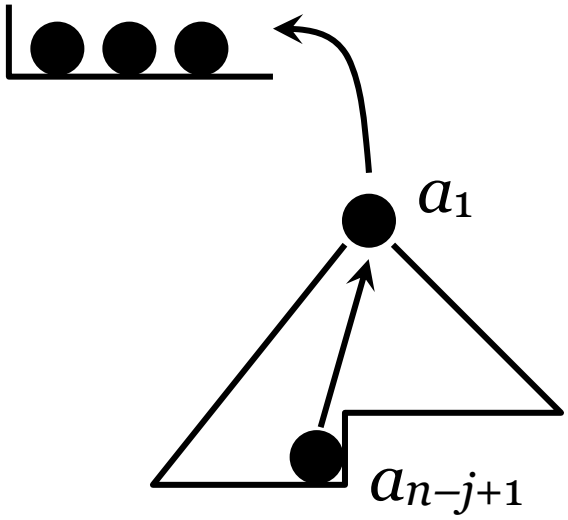
(j, n)–процедура: выполняем (j, n)–операцию; если a_j переместилось вниз, скажем, стало a_{2j+1}, то производим $(2j+1, n)$–операцию и т.д., пока наш элемент a_j не остановится.

Первый этап состоит в последовательном выполнении (j, n)–процедуры для $j = \lceil n/2 \rceil, \lceil n/2 \rceil - 1, \ldots, 1$.

Упражнение. Доказать, что первый этап требует не более $2n$ (i, n)–операций.

Второй этап

На j-й итерации $(j - 1)$ самых малых чисел уже найдены и лежат на «полочке» в нужном порядке. Остальные находятся в пирамиде $(a_i \le \min(a_{2i}, a_{2i+1}))$. Итерация состоит в том, что элемент a_1 из пирамиды кладется на «полку», а на его место ставится элемент a_{n-j+1} из пирамиды и выполняется $(1, n-j)$–процедура.



После выполнения n-й итерации все числа лежат на полке в полном порядке.

Время — $O(n \log n)$.

Замечание. «Полку» можно организовать прямо в пирамиде.

Сбалансированные деревья

Для массива данных требуется

1. Найти элемент
2. Найти k-й по порядку элемент
3. Вставить элемент
4. Удалить элемент

Если упорядочить массив, то 1 и 2 требуют $O(\log n)$ операций,
но 3, 4 — $O(n)$.

Если хранить данные в виде списка, то 3, 4 — $O(\log n)$, 1, 2 — $O(n)$.

Сбалансированные деревья требуют $O(\log n)$ для 1 – 4.

Определение 1. *Высотой* дерева называется максимальная длина пути от корня до листа.

Определение 2. Бинарное дерево называется *сбалансированным* (или *AVL–деревом*), если для любой его вершины высота правого поддерева отличается от высоты левого поддерева не более чем на единицу.

Теорема. Длина ветвей в n-вершинном сбалансированном дереве заключена между $\log_2 n$ и $\frac{3}{2}\log_2 n$.

Доказательство.

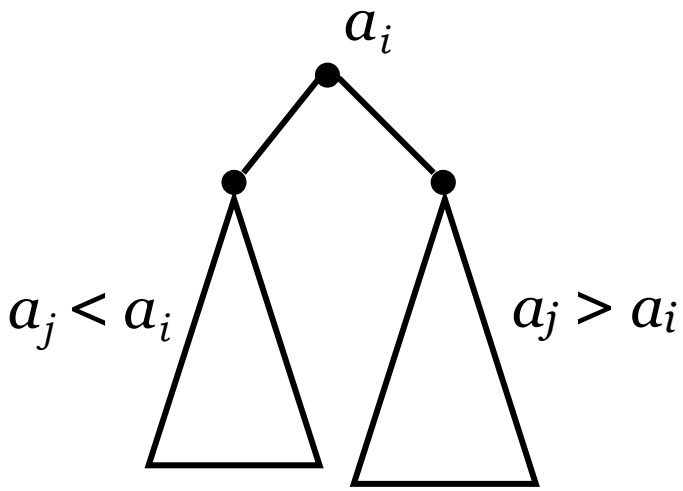
1. Бинарное дерево высоты h не может содержать больше 2^{h+1} вершины, то есть $n \le 2^{h+1}$ или $h+1 \ge \log_2 n$.
2. Наиболее ассиметричное AVL–дерево T_h высоты h имеет наиболее ассиметричное AVL–дерево T_{h-1} высоты $h-1$ в качестве одного из своих поддеревьев и наиболее ассиметричное AVL–дерево T_{h-2} в качестве другого. Обозначим через $n(h)$ число вершин в дереве T_h. Тогда

$$n(h) = n(h-1) + n(h-2) + 1; \quad n(0) = 1,\ n(-1) = 0.$$

Для h=3,4 можно непосредственно проверить, а затем по индукции доказать, что $n(h) > \alpha^{h+1}$, где $\alpha = \frac{1+\sqrt{5}}{2}$.

Следовательно, $n \ge n(h) > \alpha^{h+1}$, откуда $h+1 \le \log n / \log \alpha \approx 1{,}44 \log n$. ■

Пусть в вершинах AVL–дерева расположены элементы массива так, что для любой вершины в левом ее поддереве расположены элементы меньше чем в данной вершине, а в правом поддереве — больше, чем в этой вершине.



Пример. Поиск в AVL–дереве потребует более 25 сравнений, только если дерево состоит из не менее 196 417 вершин.

Случайные деревья

Для сбалансированного дерева длина пути из корня в лист не превышает $1{,}44\log n$.

Для случайного дерева *средняя* длина пути из корня в лист составляет $1{,}39\log n$, но в худшем случае может оказаться равной n.

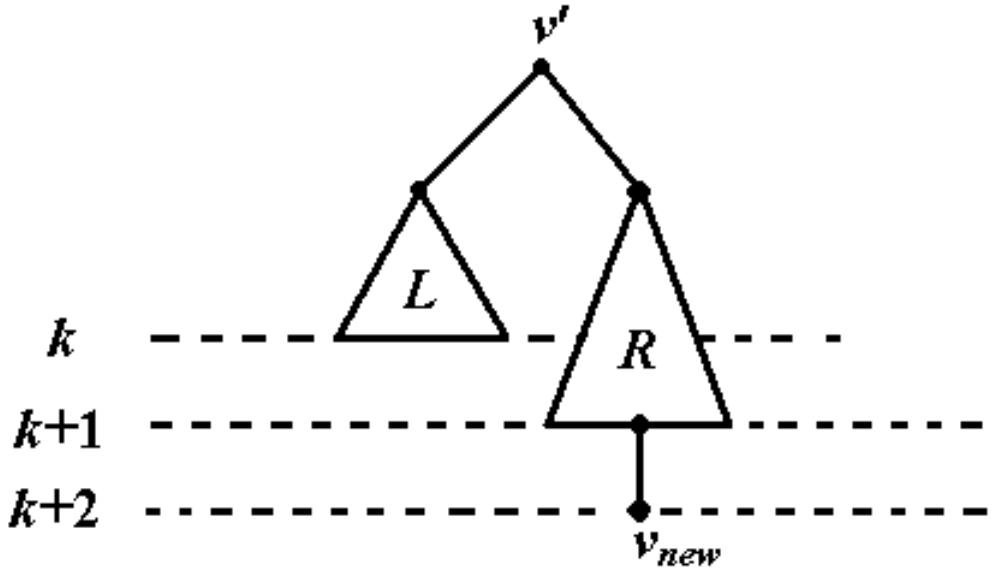
Для сбалансированного дерева *средняя* длина пути составляет $c\log n$, $c = \dfrac{\alpha}{\sqrt{5}\log\alpha} \approx 1{,}04$.

Включение новой вершины в AVL–дерево

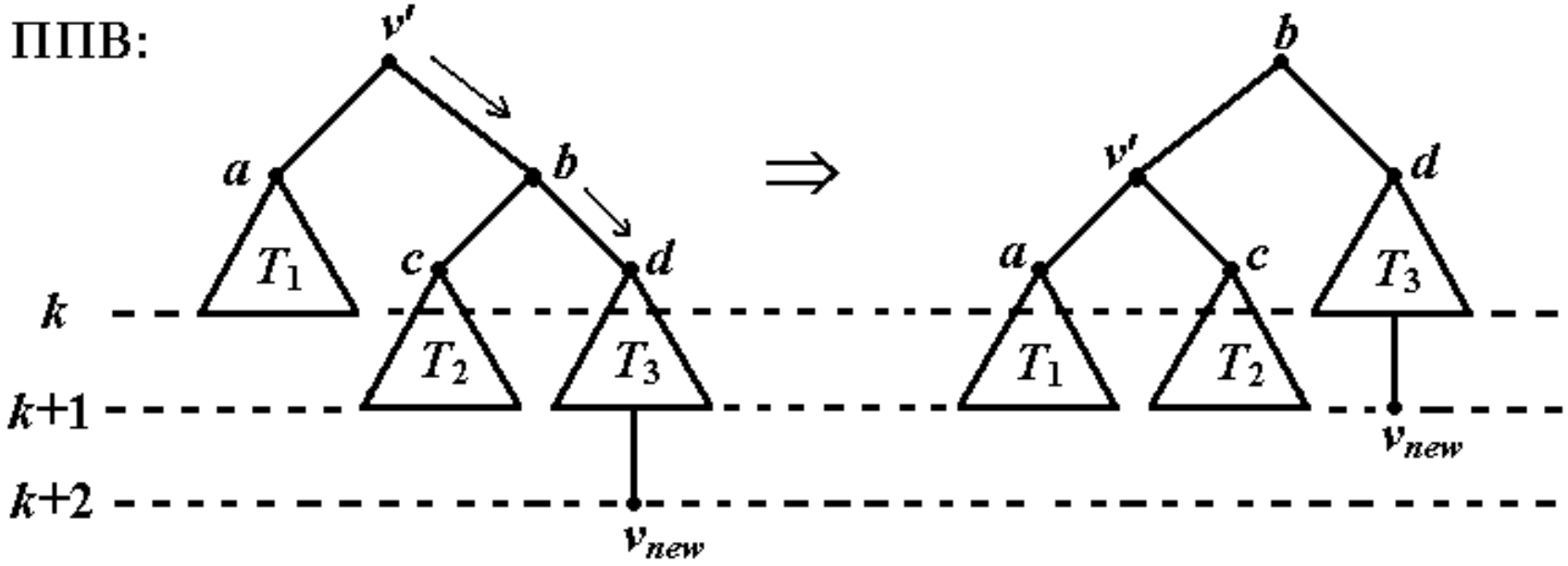
При добавлении новой вершины v_{new} к AVL–дереву мы «скатываем» ее от корня вдоль веток и получаем новый лист (висячую вершину). Дерево остается бинарным, но баланс может нарушиться. Эти нарушения могут возникнуть только у вершин, лежащих на пути от корня к новой вершине.

Будем последовательно подниматься от новой вершины к корню и восстанавливать баланс, если это необходимо.

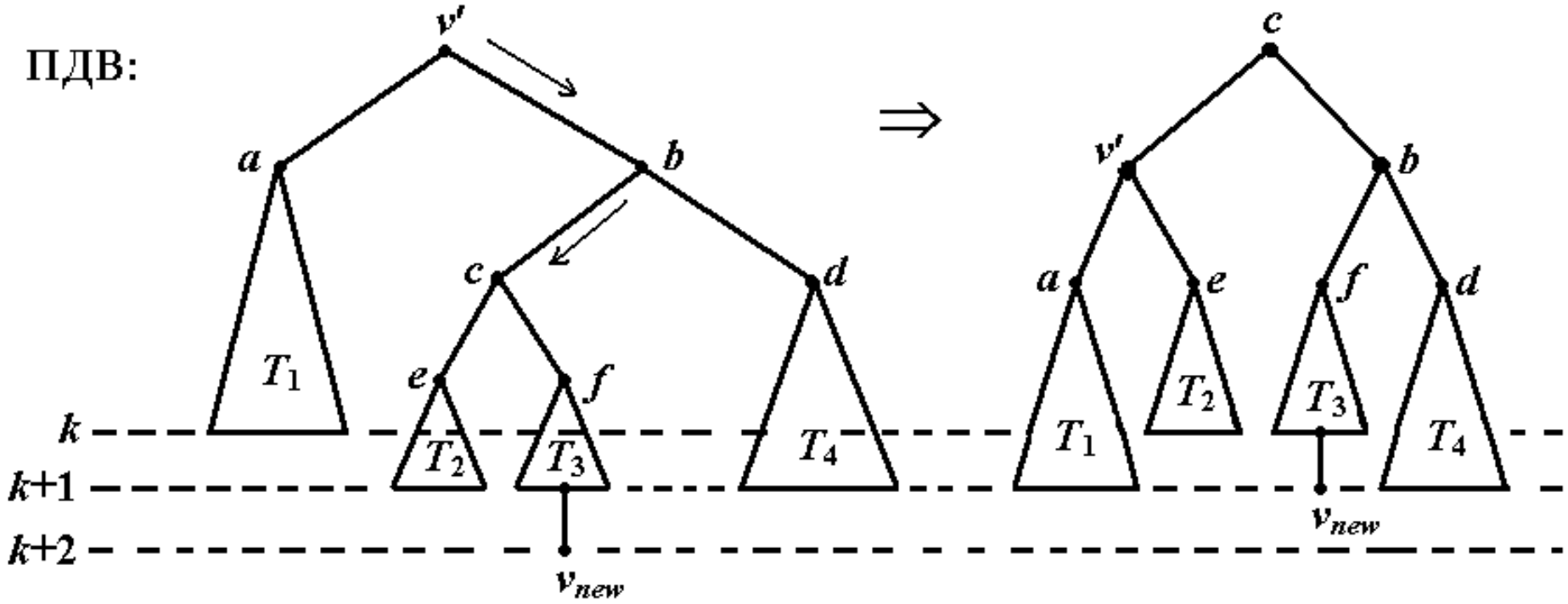
Пусть v' — самая нижняя вершина дисбаланса, то есть наиболее удаленная от корня вершина такая, что ее поддерево с вершиной v_{new} имеет высоту k+2, а другое поддерево — высоту k:



Будем восстанавливать баланс в v' следующим образом. Если первые два шага на (единственном пути) от v' к v_{new} делаются в одном направлении (оба вправо, или оба влево), то применяем правило простого вращения (ППВ).

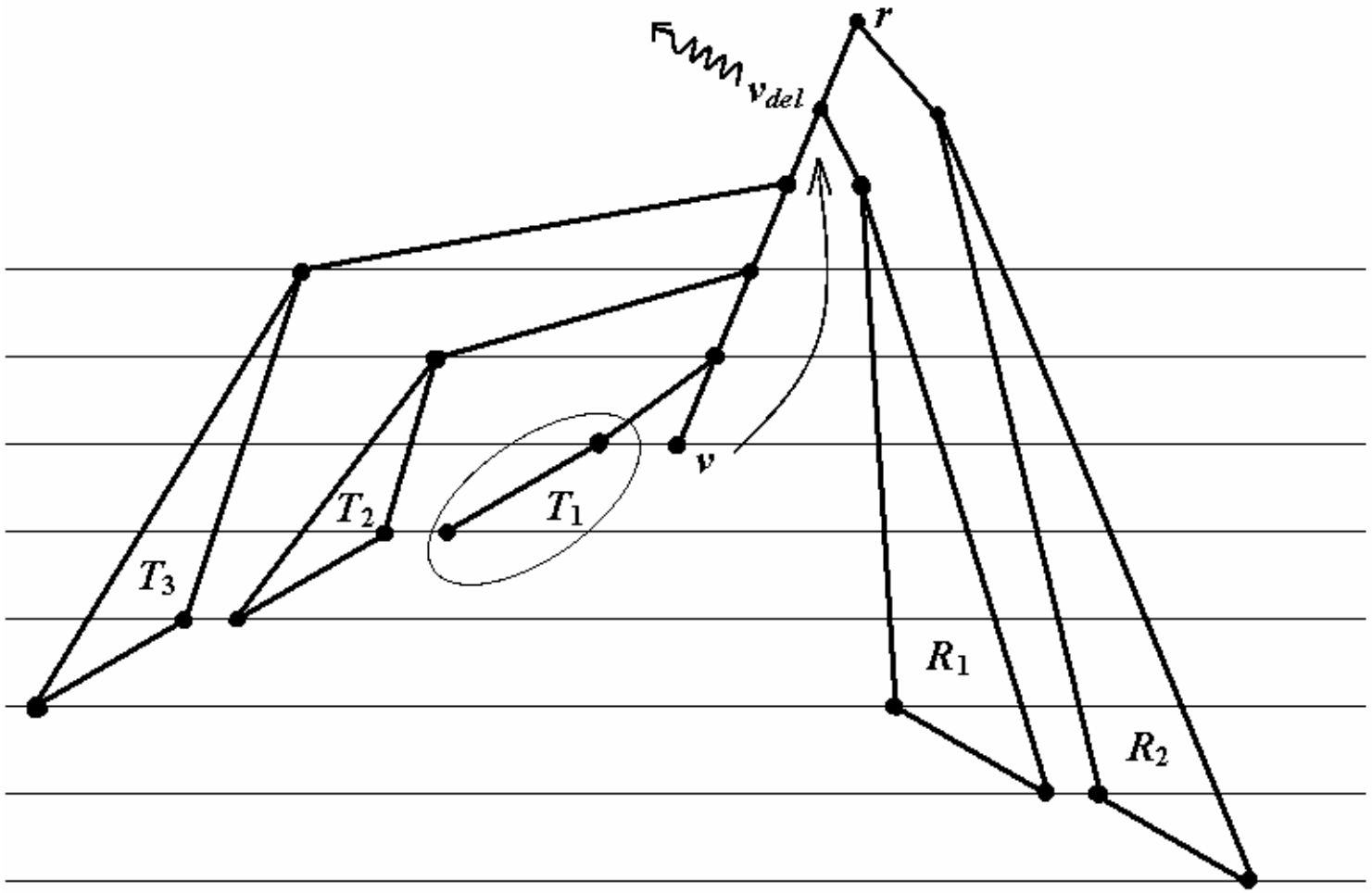


Если первые два шага делаются в разных направлениях, то применяем правило двойного вращения (ПДВ):

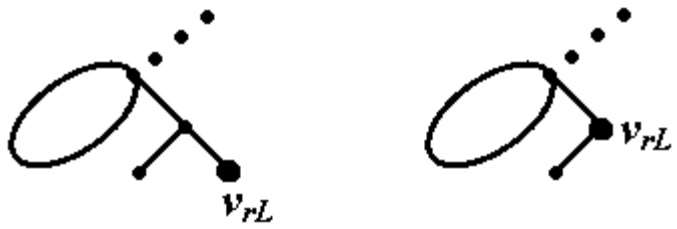


Удаление вершины

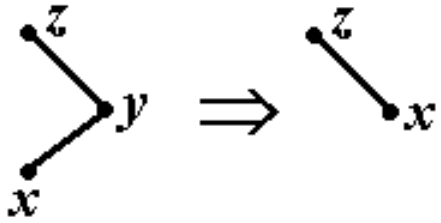
На место удаленной вершины v ставим либо самую правую вершину v_{rL} левого поддерева, либо самую левую вершину v_{Lr} правого поддерева



Нюанс. Каждая из вершин v_{rL} и v_{Lr} может быть либо висячей, либо предвисячей, то есть имеющей в качестве потомков лишь одну вершину (разумеется, висячую):



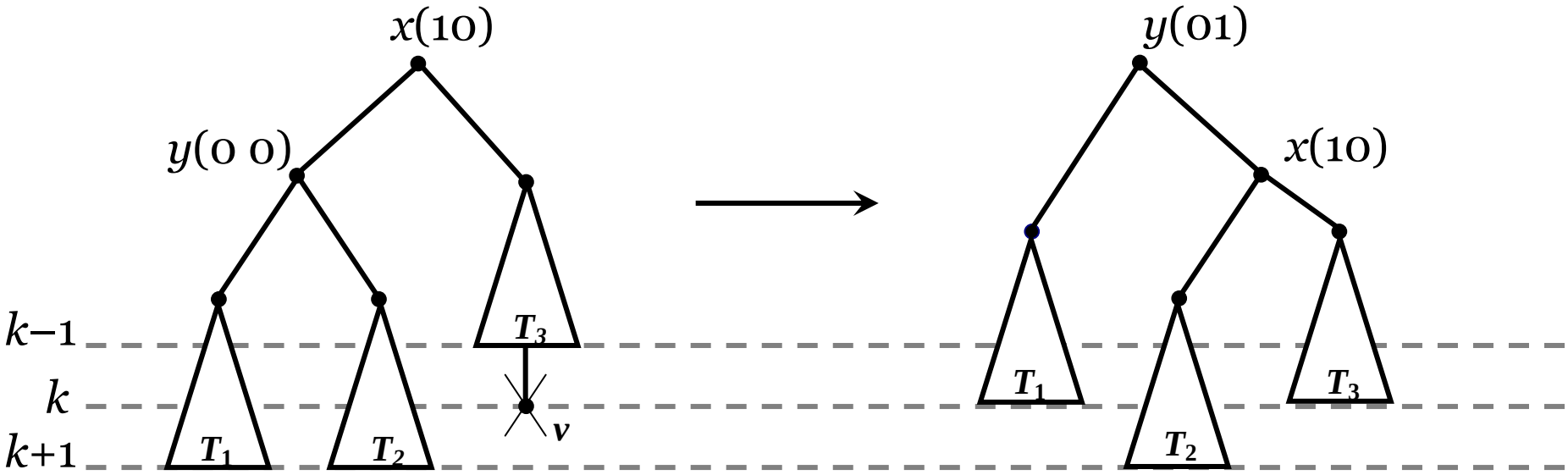
Если на место удаленной вершины встает предвисячая вершина y, то ее (вершины y) потомок x подключается к ее предку z:



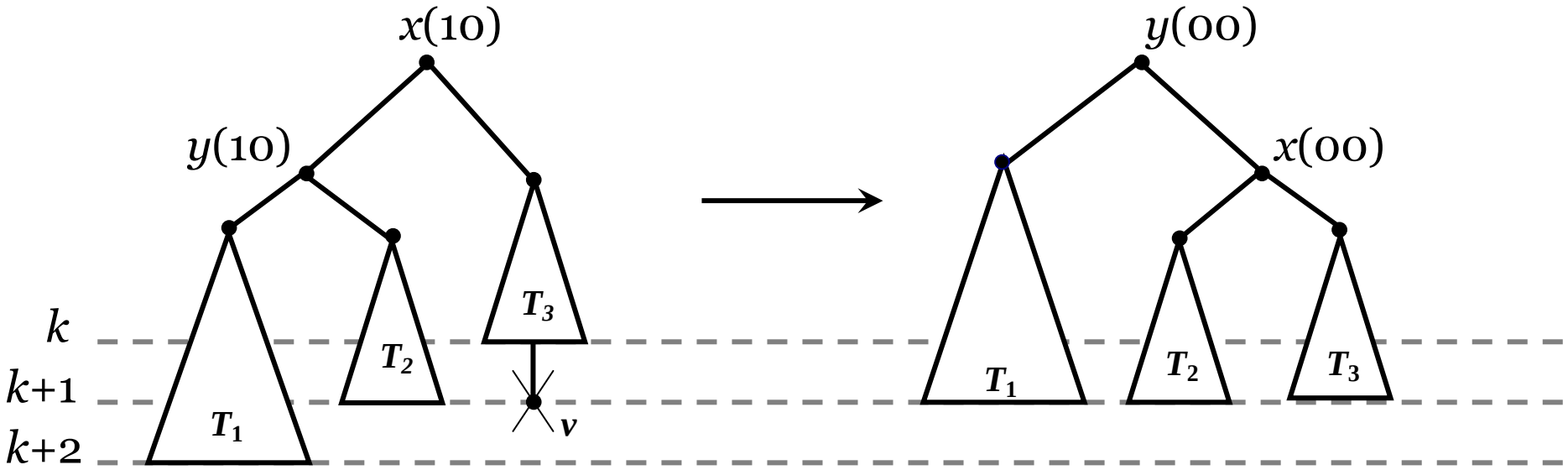
Итак, можно считать, что *всегда удаляем лист*. Последовательно поднимаемся от удаленной вершины v к корню и исправляем структуру дерева, если необходимо.

Пусть к текущей вершине x на пути от v к корню мы пришли справа по короткой ветке. Тогда возможно три случая:

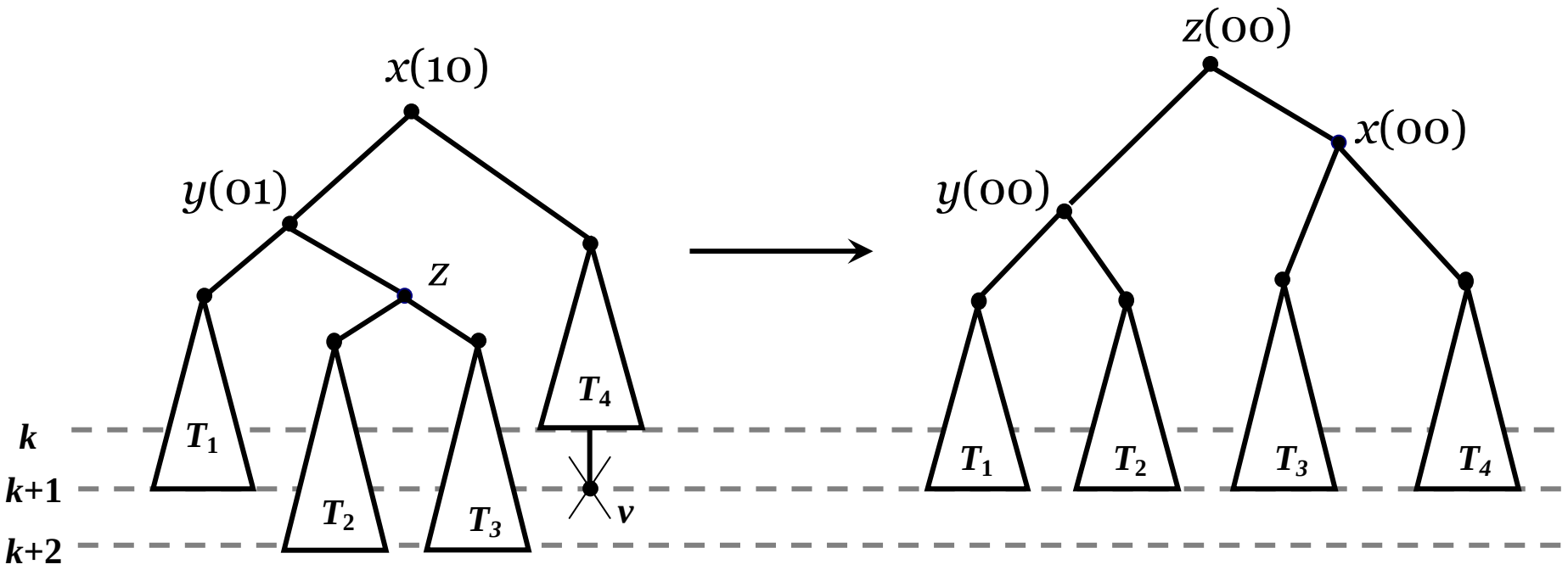
а) В вершине y высоты поддеревьев равны:



б) В вершине y высота левого поддерева больше высоты правого поддерева:



в) В вершине y высота левого поддерева меньше высоты правого поддерева



Отметим, что устранение дисбаланса в одной из вершин, может нарушить баланс в вышестоящих вершинах. На рисунке показан предельный случай этого явления. При начальном дисбалансе лишь в одной вершине (лежащей непосредственно под v) приходится, тем не менее, производить перестройку дерева во всех вершинах на пути к корню.