Алгоритми пошуку на основі дерев

Мета лекції

- Розглянути визначення та види дерев пошуку (двійкові упорядковані, випадкові, збалансовані у висоту АВЛ дерева, В дерева, червоно-чорні дерева).

- Основні прийоми зниження трудомісткості пошуку в деревовидних структурах.

Основні поняття

Пошук - процес знаходження конкретної інформації в раніше створеній множині даних.

Ключ пошуку - це поле, за значенням якого відбувається пошук.

Фактори вибору алгоритму пошуку

- Спосіб представлення даних
- Впорядкованість даних
- Обсяг даних
- Розташування у зовнішній або внутрішній пам’яті

Чому саме “дерева”?

Можливо зменшити число порівнянь ключів з еталоном, якщо виконати організацію дерева особливим чином, тобто розташувати його елементи за певними правилами. При цьому в процесі пошуку буде переглянуто не все дерево, а окреме піддерево.

Такий підхід дозволяє класифікувати дерева в залежності від правил побудови.

Двійкові (бінарні) дерева

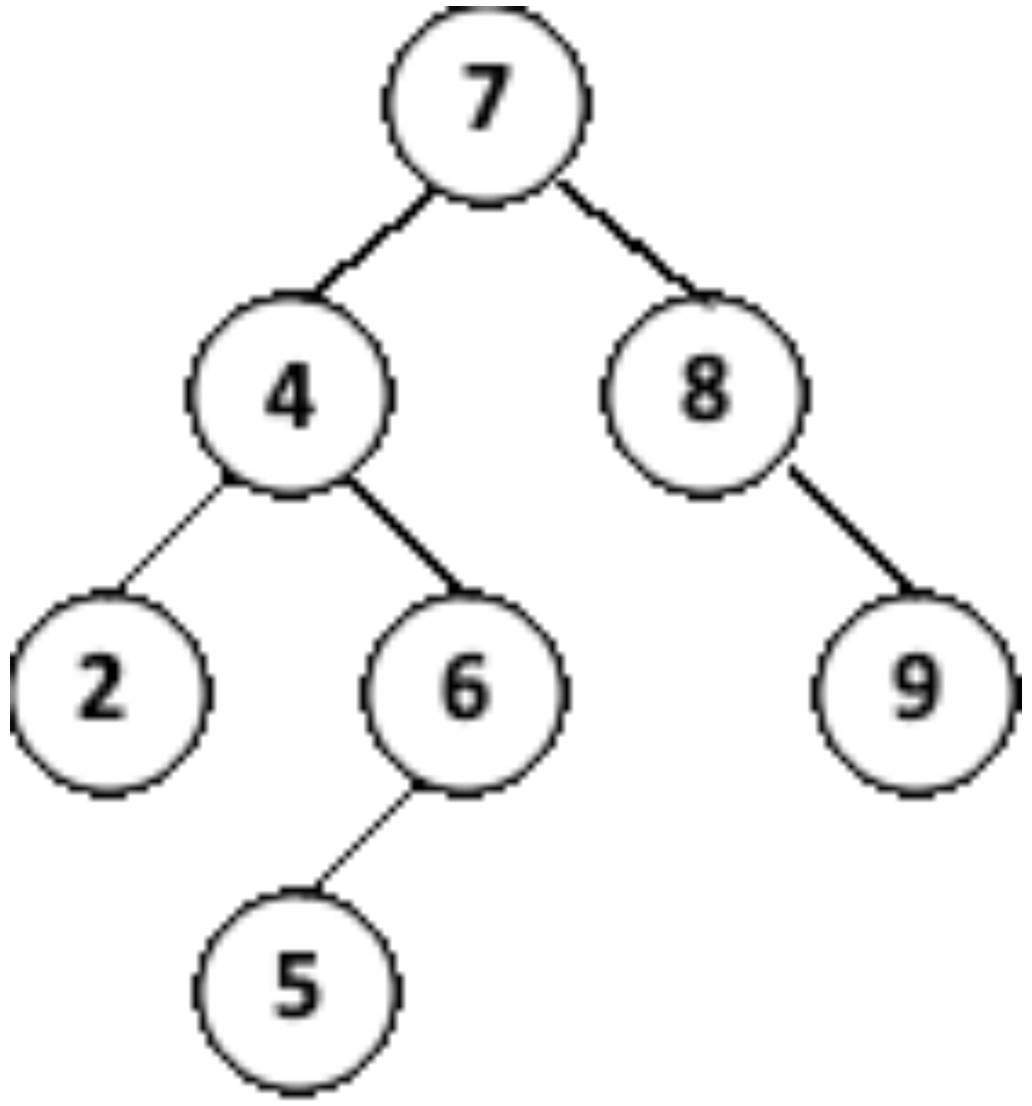
Двійкові дерева являють собою ієрархічну структуру, в якій кожен вузол має не більше двох нащадків. Кожне піддерево також в свою чергу є деревом (кожне з яких може бути порожнім).

Пошук на таких структурах **не дає виграшу** по виконанню в порівнянні з лінійними структурами того ж розміру, так як необхідно в гіршому випадку виконати обхід всього дерева. Тому інтерес представляють двійкові впорядковані дерева.

Двійкові впорядковані дерева пошуку (BST)

Двійкове дерево впорядковано, якщо для будь-якої його вершини x справедливі такі властивості:

- Всі елементи в лівому піддереві менше елемента, що зберігається в x
- Всі елементи в правому піддереві більше елемента, що зберігається в x
- Всі елементи дерева різні



Ефективність бінарних дерев пошуку

Бінарні дерева пошуку набагато ефективніші в операціях пошуку, аніж лінійні структури, в яких витрати часу на пошук пропорційні $O(n)$, де n - розмір масиву даних.

Тоді як в повному бінарному дереві цей час пропорційний в середньому $O(log_2 n)$ або $O(h)$, де h - висота дерева (хоча гарантувати що h не перевищує $log_2 n$ можна лише для збалансованих дерев, які є більш ефективними).

Найгірший випадок - виродження дерева у лінійний список, складність пошуку - $O(n)$.

Основні поняття

Повне (закінчене) двійкове дерево — таке двійкове дерево, в якому кожна вершина має нуль або двоє дітей.

Збалансоване дерево — різновид бінарного дерева, в якому кількість рівнів вершин під коренем є мінімальною.

Ідеально збалансоване дерево - називається дерево, у якого для кожної вершини виконується вимога: число вершин в лівому і правому піддереві розрізняється не більше ніж на 1.

АВЛ збалансованість — є менш строгою умовою збалансованості, бінарне дерево є AVL збалансованим, якщо для кожної вершини виконується вимога: висота лівого і правого піддерев розрізняються не більше, ніж на 1.

Частково впорядкованим є таке двійкове дерево, в якому виконуються перші дві властивості, але зустрічаються однакові елементи. Надалі йтиметься тільки про двійкові впорядковані дерева.

Основними операціями, що здійснюються з впорядкованими деревами, є:

- пошук вершини
- додавання вершини
- видалення вершини
- друк дерева
- очищення дерева

Випадкові дерева
(Randomized BST)

Випадкові дерева пошуку являють собою впорядковані бінарні дерева пошуку, при створенні яких елементи (їх ключі) вставляються в випадковому порядку. При створенні таких дерев використовується той же алгоритм, що і при додаванні вершини в бінарне дерево пошуку.

При надходженні елементів у випадковому порядку отримуємо дерево з мінімальною висотою $h \sim log_2 n$ (величина не є гарантованою, а май ймовірністний характер).

При надходженні елементів в упорядкованому порядку відбувається побудова вироджених дерев пошуку (вони перетворюються в лінійний список).

Оптимальні дерева (Optimal BST)

У двійковому дереві пошук одних елементів може відбуватися частіше, ніж інших, тобто існують ймовірності пошуку k-го елемента і для різних елементів ці ймовірності неоднакові. Можна припустити, що пошук в дереві в середньому буде швидшим, якщо ті елементи, які шукають частіше, будуть перебувати ближче до кореня дерева.

Нехай відомі ймовірності p_i звернень до i-и вузлів з ключами k_i . Задача організувати бінарне дерево пошуку таким чином щоб мінімізувати математичне очікування числа порівнянь при пошуку. Для цього припишемо кожному вузлу у визначенні довжини шляху вагу, рівну ймовірності звернення до цього вузла. Тоді зважена довжина шляху дерева - це сума всіх шляхів від кореня до кожного вузла, помножені на ймовірність звернення до даного вузла:

$$P_T = \sum_{t=1}^{n} p_i * h_i$$

Задача мінімізувати дану довжину шляху. Очевидно, що вузли з більшими ймовірностями звернення повинні знаходитись ближче до кореня дерева.

Побудова оптимальних дерев пошуку реалізується за допомогою динамічного програмування.

АВЛ дерева (AVL Trees)

АВЛ дерево — збалансоване по висоті двійкове дерево пошуку, для кожної вершини якого виконується вимога: висота лівого і правого піддерев розрізняються не більше, ніж на 1.

У 1962 році два радянських математика: Г.М. Адельсон-Бєльський і Е.М. Ландіс - ввели менш суворе визначення збалансованості і довели, що при такому визначенні можна написати програми додавання і / або видалення, які мають логарифмічну складність і зберігають дерево збалансованим.

Не всяке збалансоване по АВЛ дерево ідеально збалансовано, але всяке ідеально збалансоване дерево збалансовано по АВЛ.

Доведено, що висота h АВЛ дерева з n ключами лежить в діапазоні від $log_2(n+1)$ до $1.44log_2(n+2) - 0.328$.

Складність пошуку гарантована (в будь-якому випадку) - $O(log_2 n)$

АВЛ дерева

В процесі додавання та видалення вузлів в АВЛ дереві можливе виникнення ситуації **розбалансування** піддерев**.**

Кожний вузол крім самого значення зберігає ще одне значення - фактор балансування.

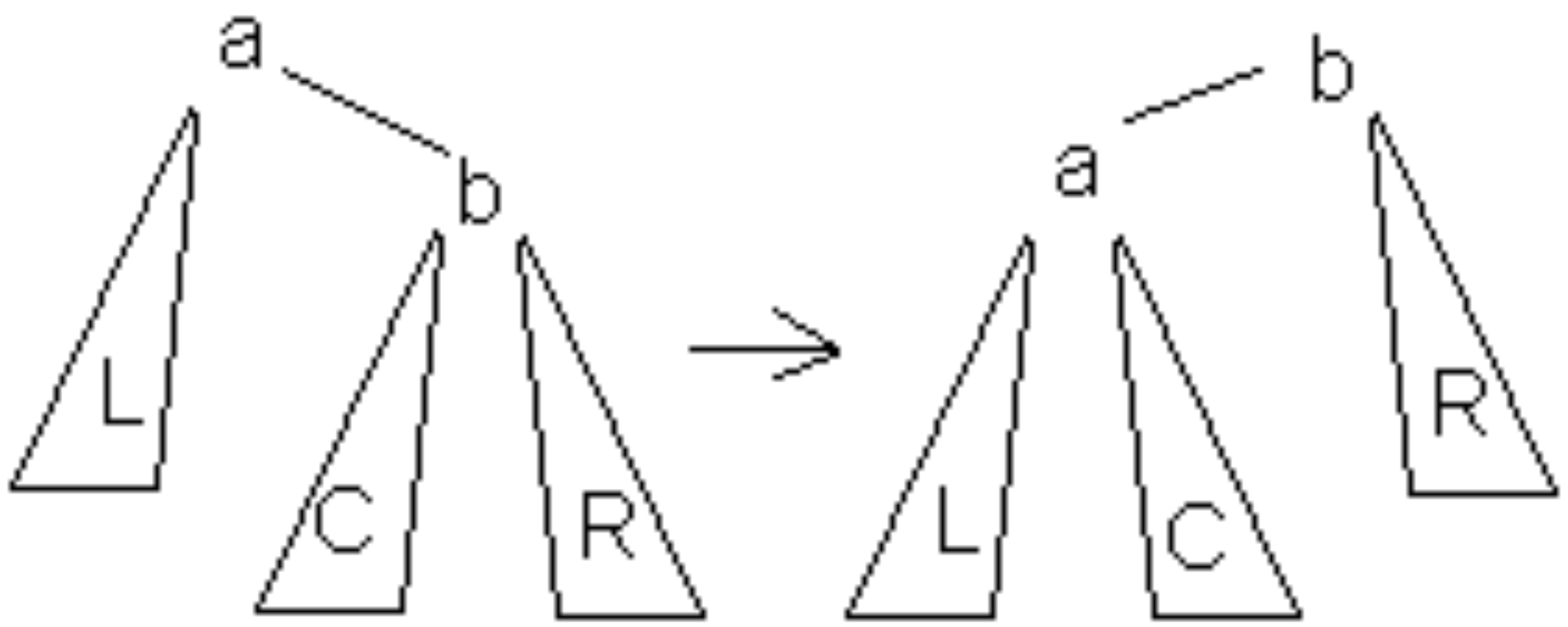
$$BalanceFactor(node) = Height(RightSubtree(node)) - Height(LeftSubtree(node))$$

В ситуації коли фактор балансування для певного вузла більший за 1 - необхідно привести дерево до збалансованого вигляду.

АВЛ дерева

Використовують 4 типи обертань для балансування АВЛ дерев.

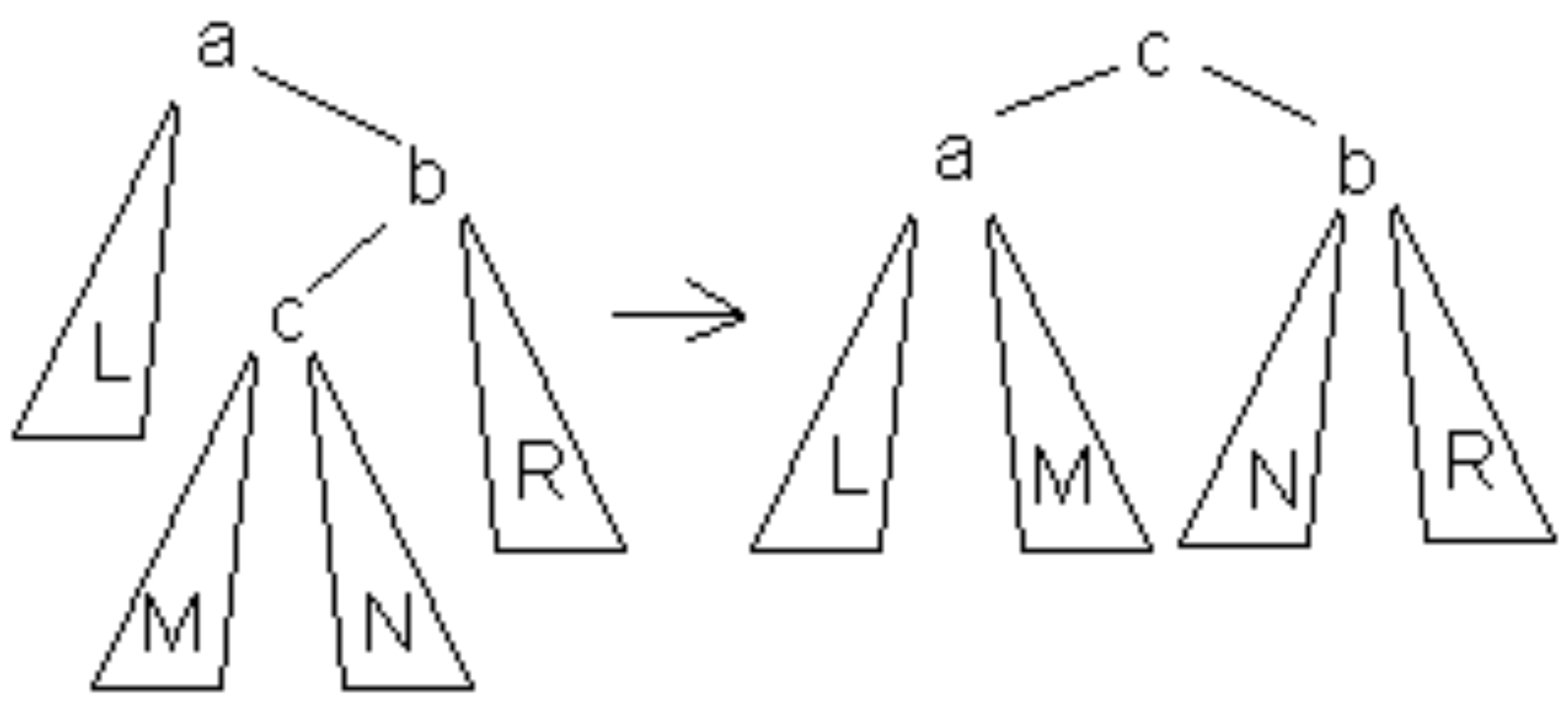
Мале ліве обертання використовується тоді, коли (висота b-піддерева - висота L) = 2 і висота C <= висота R.



АВЛ дерева

Використовують 4 типи обертань для балансування АВЛ дерев.

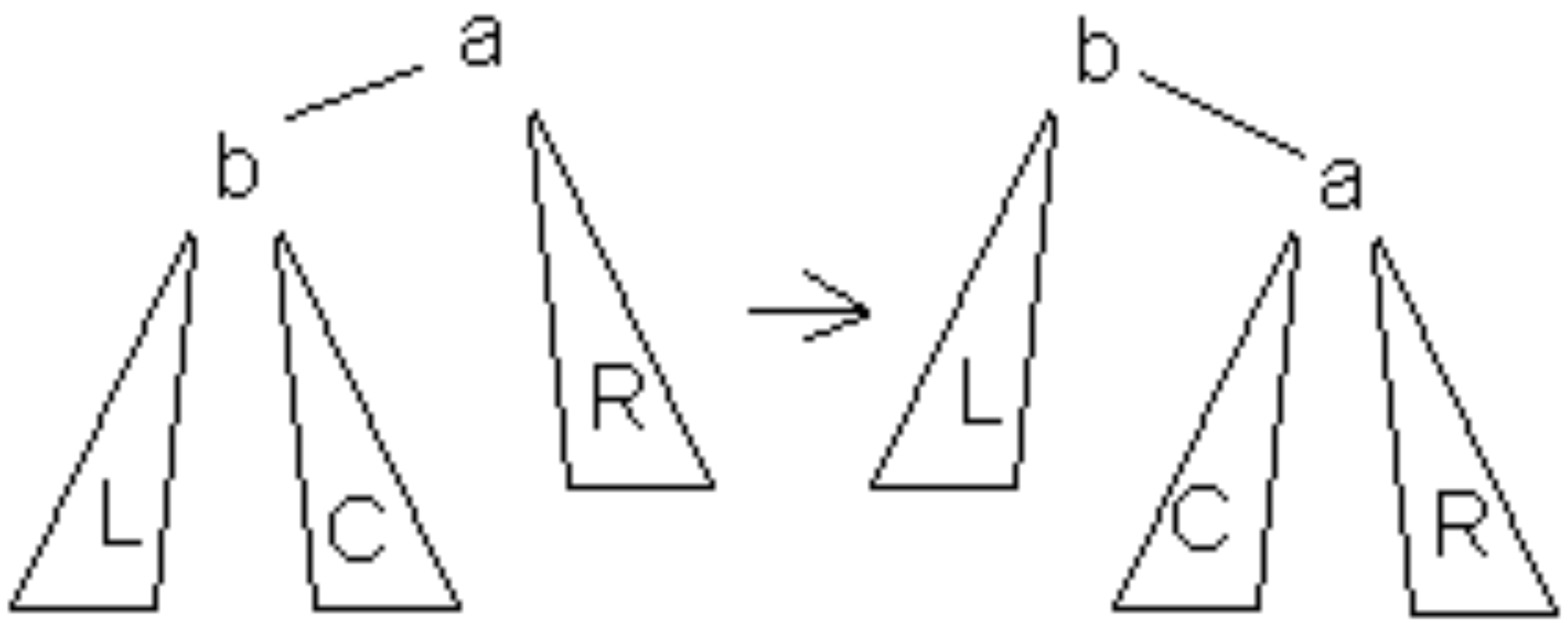
Велике ліве обертання використовується тоді, коли (висота b-піддерева - висота L) = 2 і висота C-піддерева > висота R.



АВЛ дерева

Використовують 4 типи обертань для балансування АВЛ дерев.

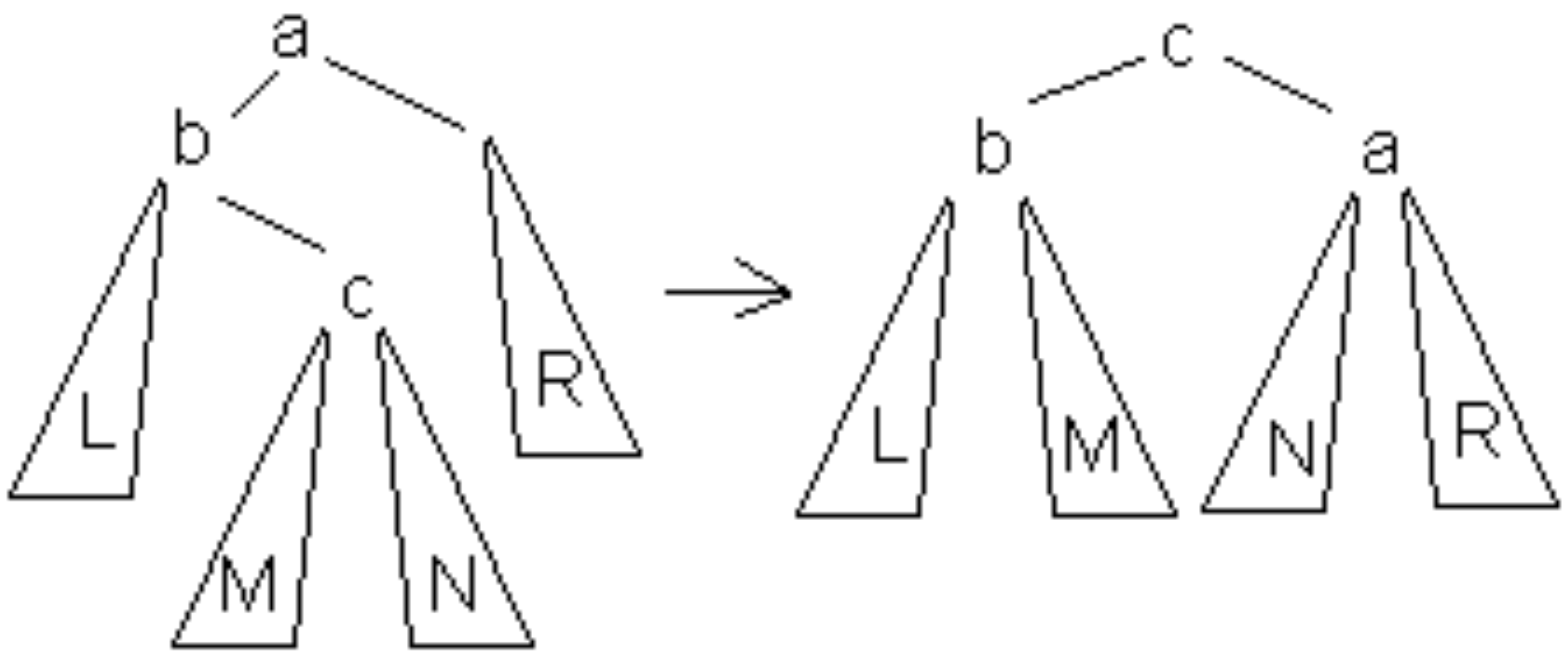
Мале праве обертання використовується тоді, коли (висота b-піддерева — висота R) = 2 і висота C <= висота L.



АВЛ дерева

Використовують 4 типи обертань для балансування АВЛ дерев.

Велике праве обертання використовується тоді, коли (висота b-піддерева — висота R) = 2 і висота C -піддерева> висота L.



АВЛ дерева

Алгоритм додавання нової вершини в збалансоване АВЛ дерево складатиметься з наступних трьох основних кроків:

Крок 1. Пошук по дереву.

Крок 2. Вставка елемента в місце, де закінчився пошук, якщо елемент

відсутній.

Крок 3. Відновлення збалансованості (перевіряємо для кожної вершини від місця додавання до кореня).

Перший крок необхідний для того, щоб переконатися у відсутності елементу в дереві, а також знайти таке місце вставки, щоб після вставки дерево залишилося впорядкованим. Третій крок являє собою зворотний прохід по шляху пошуку: від місця додавання до кореня дерева. У міру просування цим шляхом коригуються показники збалансованості прохідних вершин, і проводиться балансування там, де це необхідно. Додавання елемента в дерево **ніколи не вимагає більше одного повороту**.

АВЛ дерева

Алгоритм видалення вершини в збалансованому АВЛ дереві складатиметься з наступних трьох основних кроків:

Крок 1. Пошук по дереву.

Крок 2. Видалення елемента з дерева.

Крок 3. Відновлення збалансованості дерева (зворотний прохід).

Перший крок необхідний, щоб знайти в дереві вершину, яка повинна бути видалена. Третій крок являє собою зворотний прохід від місця з якого вилучено елемент. Операція видалення може зажадати перебалансування всіх вершин уздовж дороги назад до кореня дерева, тобто порядку $log_2 n$ вершин.

Дерева цифрового (порозрядного) пошуку (DST)

Деякі методи пошуку не порівнюють на кожному кроці повні значення ключів пошуку, а продивляються ключі лише невеликими фрагментами.

Принциповою перевагою методів порозрядного пошуку є принятна складність у найгіршому випадку при відсутності складності балансування дерев.

Алгоритми операцій пошуку та вставки аналогічні тим, що реалізовані в бінарному дереві пошуку, з однією різницею. Проходження по гілками дерева виконується не в результаті порівняння повних ключів, а використовуються тільки обрані розряди ключа. На першому рівні використовується старший розряд, на другому рівні - наступний за ним і т.д. поки не зустрінеться зовнішній вузол.

Дерева цифрового (порозрядного) пошуку (DST)

Дерева цифрового (порозрядного) пошуку (DST)

Якщо дані випадково розподілені, тоді в середньому випадку складність пошуку $O(logh)$, де h - висота дерева.

В найгіршому випадку $O(b)$, де b — кількість розрядів в ключі для пошуку.

Дерева цифрового (порозрядного) пошуку (DST)

Основні переваги дерева цифрового порозрядного пошуку:

- Вставка, пошук та видалення простіші ніж в BST та AVL деревах.
- Дерево не потребує додаткової інформації для підтримки збалансованості тому що <u>глибина дерева обмежена довжиною ключа пошуку</u>.
- Потребує менше пам’яті ніж BST and AVL дерева.

Проблеми стандартних дерев пошуку

Розглянемо об'ємну базу даних, представлену у вигляді одного з вищезгаданих дерев. Очевидно, що ми не можемо зберігати все дерево в оперативній пам'яті, в ній зберігаємо лише частину інформації, а все інше зберігається на сторонньому носії (наприклад HDD, швидкість доступу до якого набагато менша). Збалансовані дерева будуть вимагати від нас $log(n)$ звернень до стороннього носія. При великих n - це неприйнятно.

Дану проблему вирішують B дерева. Час виконання операцій в них також пропорційний висоті дерева, але вона мінімізована спеціально для ефективної роботи з дисковою пам'яттю (мінімізує кількість звернень вводу-виводу).

$h_{min} = \lceil log_m(n+1) \rceil - 1$, де m - максимальна кількість дітей що може мати вузол

$$h_{max} = \lfloor log_d \frac{n+1}{2} \rfloor, d = \lceil \frac{m}{2} \rceil$$

B-дерева

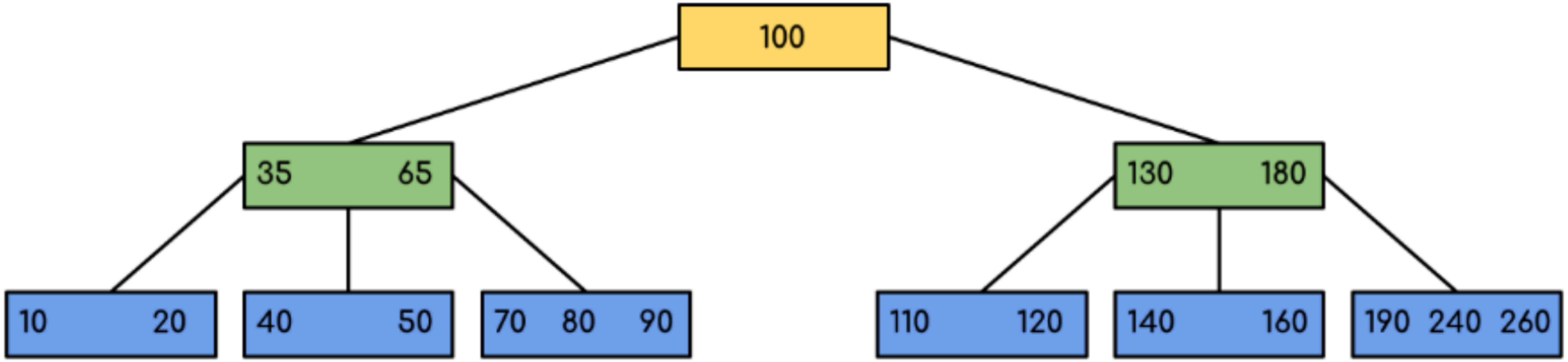
B-дерево являється ідеально збалансованим, тобто глибина всіх його листів однакова. B-дерево має наступні властивості (t - параметр дерева, називається мінімальною степінню B-дерева, не менший від 2):

- Кожен вузол, крім кореневого, містить не менше $t - 1$ ключів і кожен внутрішній вузол має по меншій мірі t дочірніх вузлів. Якщо дерево не є пустим - корінь повинен мати хоча б один ключ.
- Кожен вузол, крім кореневого, має не більше $2t - 1$ ключів і не більше чим $2t$ дочірніх внутрішніх вузлів.
- Корінь має від 1 до $2t - 1$ ключів, якщо дерево не пусте і від 2 до $2t$ дітей при висоті більшій від 0.
- Кожен вузол дерева, крім листових, що має ключі $k_1, \ldots, k_n$ має $n + 1$ дочірніх елементів. i - ий дочірній елемент має ключі з відрізку $[k_{i-1}; k_i]$. $k_0 = -\infty$, $k_{n+1} = \infty$
- Ключі у кожному вузлі впорядковані в неспадаючому порядку.
- Всі листові вузли знаходяться на одному рівні.

B-дерева

Мінімізація звернень до стороннього носія відбувається завдяки мінімізації висоти дерева. Висота дерева в свою чергу мінімізується завдяки розміщенню максимальної кількості ключів в один вузол. Розмір вузла в B-деревах зазвичай рівний розміру блоку даних на сторонньому носії.

B-дерева



Червоно-чорні дерева

Різновид збалансованого бінарного дерева пошуку, вершини якого мають додаткові властивості, зокрема колір - червоний або чорний. Дані біти кольору використовуються для того, щоб зберігати збалансованість дерева.

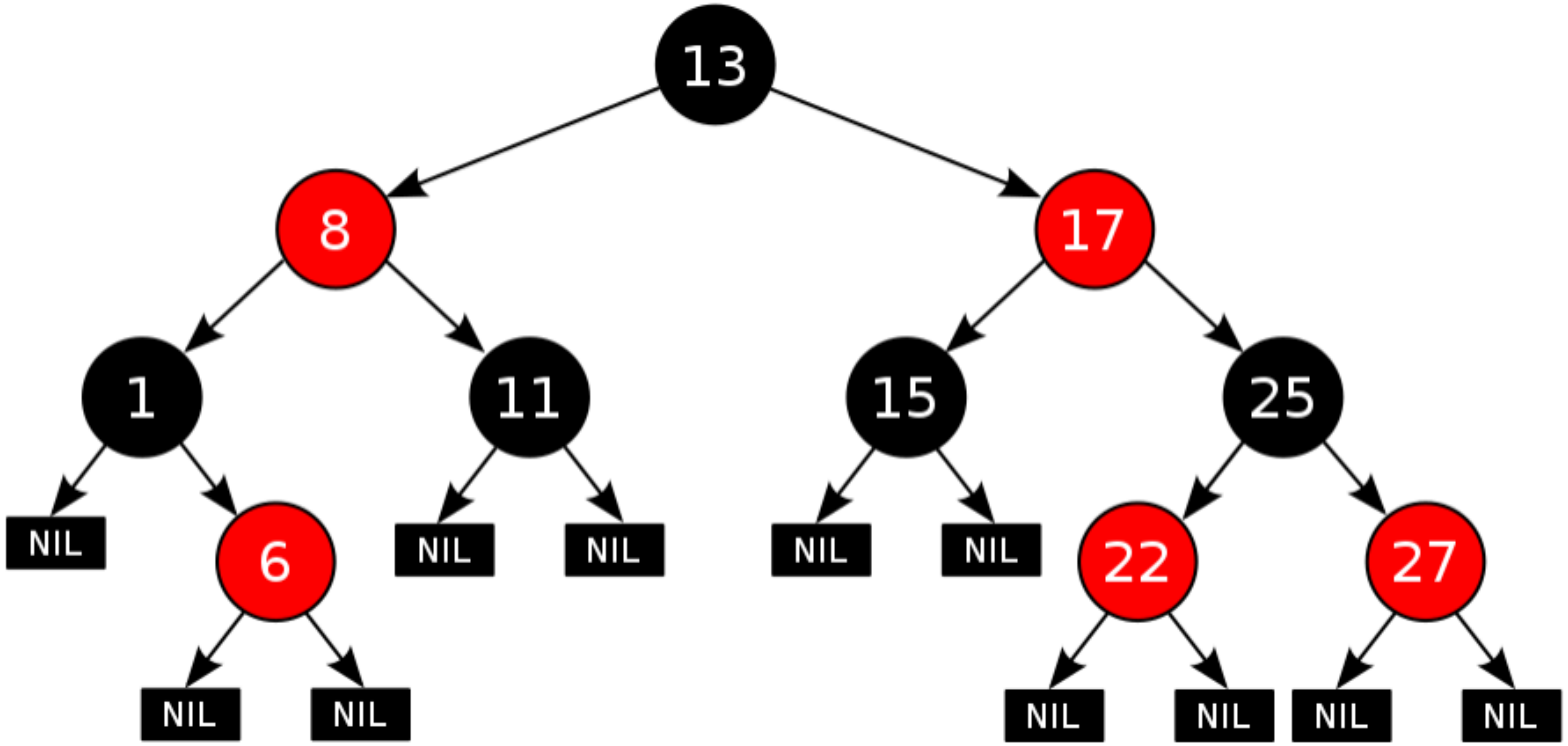
За допомогою спеціальних трансформацій гарантується що висота дерева h не перевищуватиме $O(logn)$.

Червоно-чорні дерева

Властивості:

- Кожна вершина або червона, або чорна
- Корінь дерева - чорний
- Кожен лист - чорний
- Якщо вершина червона, обидві її дочірні вершини - чорні (інакше, батько червоної вершини — чорний).
- Усі прості шляхи від будь-якої вершини до листів мають однакову кількість чорних вершин.

Червоно-чорні дерева



Червоно-чорні дерева

Такі властивості надають червоно-чорному дереву додаткового обмеження: <u>найдовший шлях з кореня до будь-якого листа перевищує найкоротший шлях не більше ніж вдвічі</u>. В цьому сенсі таке дерево можна назвати збалансованим. Зважаючи на те, що час виконання основних операцій з бінарними деревами пошуку залежить від висоти, таке обмеження гарантує їхню ефективність в найгіршому випадку, чого звичайні бінарні дерева гарантувати не можуть.

Дякую за увагу!