Курс “Алгоритмы на python”

Занятие #4
Деревья

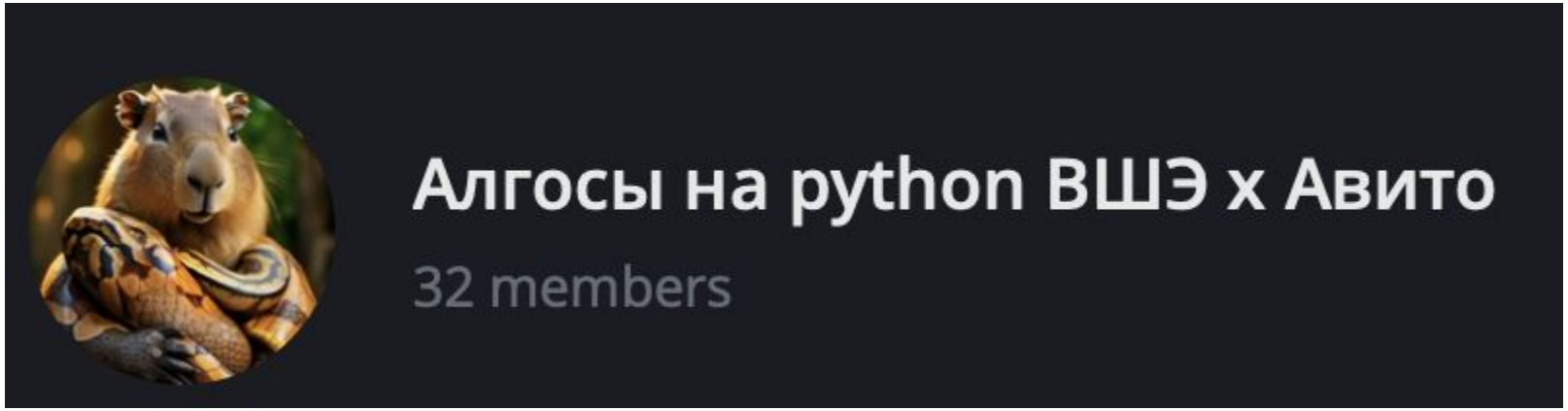
Сентябрь 2025

Avito

Единая точка входа/выхода – степик

- https://stepik.org/course/251189/

Вопросы и обсуждения – чат



Посещаемость

Орг моменты

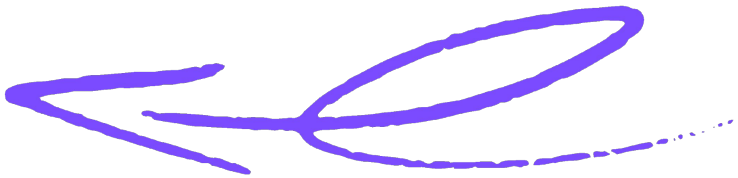
Структура курса «Алгоритмы на питоне»

1 модуль

- Введение в алгоритмы
- Базовые структуры данных
- Хеш-таблицы
- Бинарные деревья поиска
- Кучи
- Сортировки
- Рекурсия

2 модуль

- Графы
- Динамическое программирование
- Алгоритмы в строках
- Алгоритмы в ML и LLM
- Итоговый контест



План занятия

- Часть I. Дерево
- Часть II. Бинарное дерево
- Часть III. Бинарное дерево поиска
- Часть IV. Сбалансированные деревья

4 Деревья

Дерево – ацикличный связный граф

Но пойдем от частного к общему

- **корень**
- **у узла произвольное количество детей**
- **у ребенка ровно один родитель**
- **циклов нет**

Высота дерева h – глубина самого глубокого узла дерева

Деревья. Рисуем на доске

В чем сходство со связным списком?

В детей не попадем без родителя - корня

В чем отличие от связного списка?

В списке – каждый элемент указывает только на одного соседа
В дереве – на несколько

Поиск и обход

Поиск и обход – O(n)

Бинарное дерево

Бинарное дерево

максимум – два потомка – левый, правый

Бинарное дерево. Рисуем на доске.

Поиск и обход – O(n)

Бинарное дерево поиска (BST)

Бинарное дерево поиска (BST). Рисуем на доске

правильные и неправильные деревья

Бинарное дерево поиска (BST). Обходы

Pre-order, NLR [edit]

1. Visit the current node (in the figure: position red).
2. Recursively traverse the current node's left subtree.
3. Recursively traverse the current node's right subtree.

The pre-order traversal is a topologically sorted one, because a parent node is processed before any of its child nodes is done.

Post-order, LRN [edit]

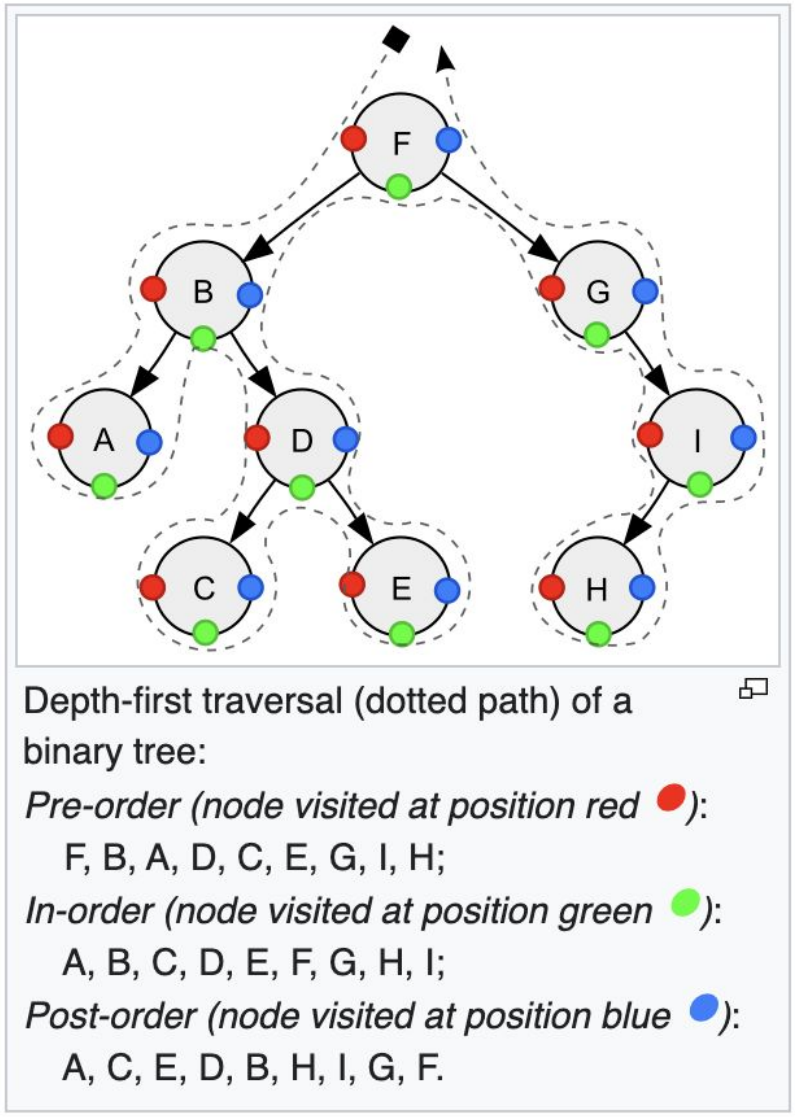
1. Recursively traverse the current node's left subtree.
2. Recursively traverse the current node's right subtree.
3. Visit the current node (in the figure: position blue).

Post-order traversal can be useful to get postfix expression of a binary expression tree.

In-order, LNR [edit]

1. Recursively traverse the current node's left subtree.
2. Visit the current node (in the figure: position green).
3. Recursively traverse the current node's right subtree.

In a binary search tree ordered such that in each node the key is greater than all keys in its left subtree and less than all keys in its right subtree, in-order traversal retrieves the keys in *ascending* sorted order.[7]



Depth-first traversal (dotted path) of a binary tree:

Pre-order (node visited at position red ●):
F, B, A, D, C, E, G, I, H;

In-order (node visited at position green ●):
A, B, C, D, E, F, G, H, I;

Post-order (node visited at position blue ●):
A, C, E, D, B, H, I, G, F.

Бинарное дерево поиска (BST). Обходы. Рисуем на доске. Валидируем BST через inorder

Бинарное дерево поиска (BST). Обходы, продолжение

Reverse pre-order, NRL [edit]

1. Visit the current node.
2. Recursively traverse the current node's right subtree.
3. Recursively traverse the current node's left subtree.

Reverse post-order, RLN [edit]

1. Recursively traverse the current node's right subtree.
2. Recursively traverse the current node's left subtree.
3. Visit the current node.

Reverse in-order, RNL [edit]

1. Recursively traverse the current node's right subtree.
2. Visit the current node.
3. Recursively traverse the current node's left subtree.

Бинарное дерево поиска (BST). Поиск. Рисуем на доске

Бинарное дерево поиска (BST). Вставка. Рисуем на доске

Бинарное дерево поиска (BST). Удаление. Рисуем на доске

Бинарное дерево поиска (BST).

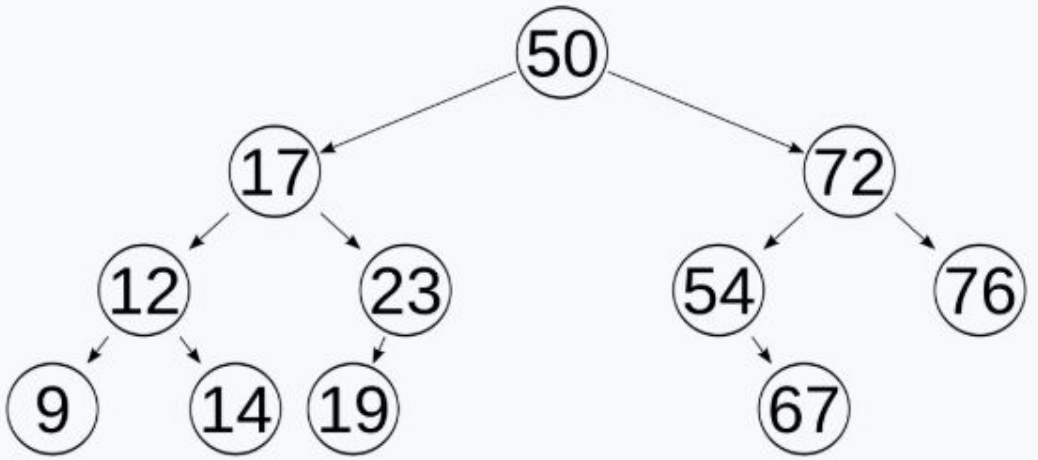
вставка O(h)
удаление O(h)
поиск O(h)
обход O(n)

Способы балансировать дерево

AVL

АВЛ-дерево

англ. *AVL tree*



Тип	дерево поиска
Год изобретения	1962
Автор	Адельсон-Вельский Георгий Максимович и Ландис Евгений Михайлович

AVL. Рисуем малый поворот и большой поворот.

Красно-черное

Red–black tree

Type	Tree
Invented	1978
Invented by	Leonidas J. Guibas and Robert Sedgewick

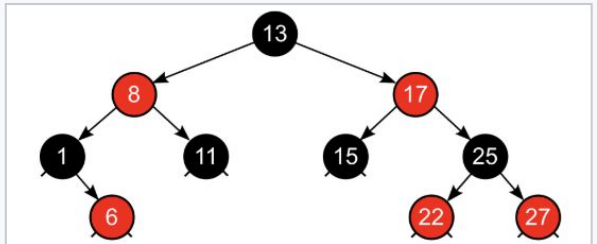
Complexities in big O notation

Space complexity

Space	$O(n)$

Time complexity

Function	**Amortized**	**Worst case**
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$



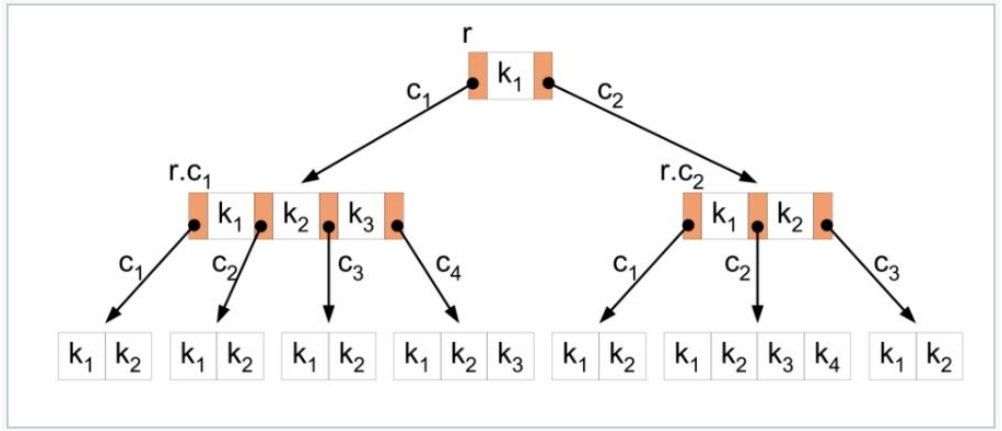
Example of a red-black tree

Properties [edit]

In addition to the requirements imposed on a binary search tree the following must be satisfied by a red–black tree:[17]

1. Every node is either red or black.
2. All null nodes are considered black.
3. A red node does not have a red child.
4. Every path from a given node to any of its leaf nodes goes through the same number of black nodes.
5. (Conclusion) If a node **N** has exactly one child, the child must be red. If the child were black, its leaves would sit at a different black depth than **N**'s null node (which is considered black by rule 2), violating requirement 4.

B-tree



Definition [edit]

According to Knuth's definition, a B-tree of order m is a tree that satisfies the following properties:[9]

1. Every node has at most m children.
2. Every node, except for the root and the leaves, has at least $\lceil m/2 \rceil$ children.
3. The root node has at least two children unless it is a leaf.
4. All leaves appear on the same level.
5. A non-leaf node with k children contains k–1 keys.

Each internal node's keys act as separation values that divide its subtrees. For example, if an internal node has 3 child nodes (or subtrees), then it must have 2 keys: a_1 and a_2. All values in the leftmost subtree will be less than a_1, all values in the middle subtree will be between a_1 and a_2, and all values in the rightmost subtree will be greater than a_2.

Internal nodes

Internal nodes (also known as inner nodes) are all nodes except for leaf nodes and the root node. They are usually represented as an ordered set of elements and child pointers. Every internal node contains a **maximum** of U children and a **minimum** of L children. Thus, the number of elements is always 1 less than the number of child pointers (the number of elements is between L–1 and U–1). U must be either $2L$ or $2L$–1; therefore each internal node is at least half full. The relationship between U and L implies that two half-full nodes can be joined to make a legal node, and one full node can be split into two legal nodes (if there's room to push one element up into the parent). These properties make it possible to delete and insert new values into a B-tree while also adjusting the tree to preserve its properties.

The root node

The root node's number of children has the same upper limit as internal nodes but has no lower limit. For example, when there are fewer than L–1 elements in the entire tree, the root will be the only node in the tree with no children at all.

Leaf nodes

In Knuth's terminology, the "leaf" nodes are the actual data objects/chunks. The internal nodes that are one level above these leaves are what would be called the "leaves" by other authors: these nodes only store keys (at most m-1, and at least m/2-1 if they are not the root) and pointers (one for each key) to nodes carrying the data objects/chunks.

A B-tree of depth n+1 can hold about U times as many items as a B-tree of depth n, but the cost of search, insert, and delete operations grows with the depth of the tree. As with any balanced tree, the cost grows much more slowly than the number of elements.

Some balanced trees store values only at leaf nodes and use different kinds of nodes for leaf nodes and internal nodes. B-trees keep values in every node in the tree except leaf nodes.