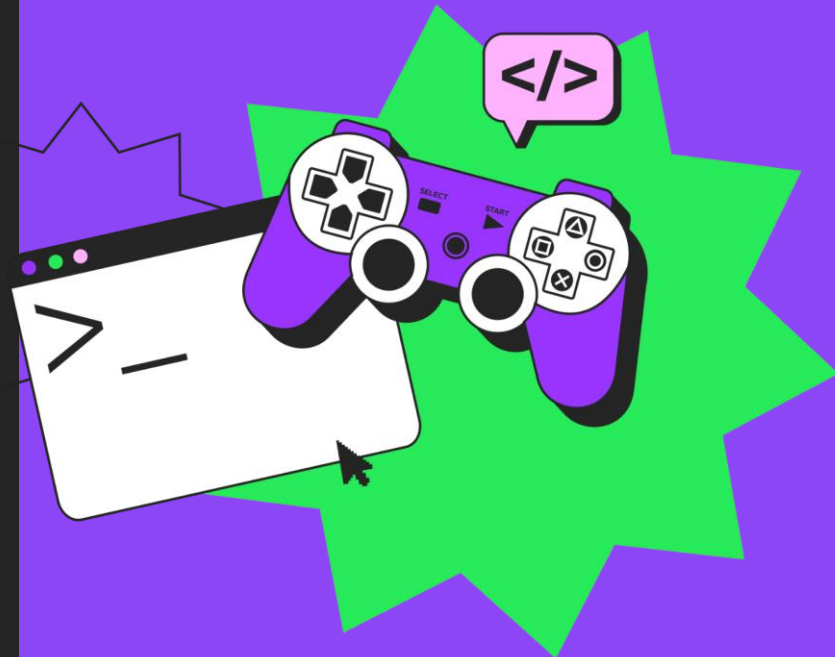


Структуры данных. Хеш-таблица. Дерево

Урок 4 Алгоритмы и структуры данных





План курса



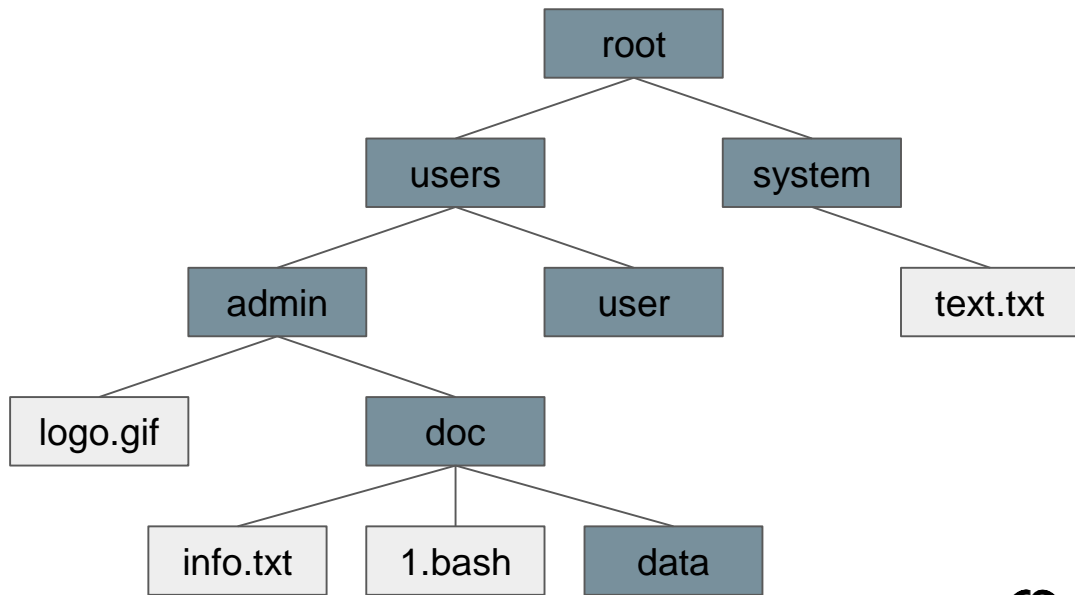
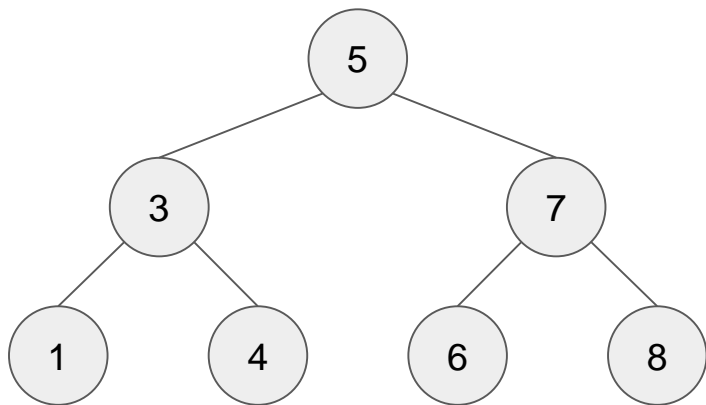


Что будет на уроке сегодня

- ✚ Что такое “дерево”
- ✚ Алгоритмы поиска элементов в дереве
- ✚ Особенности структуры хеш-таблицы
- ✚ Особенности структуры дерева
- ✚ Что такое “хеш-таблица”

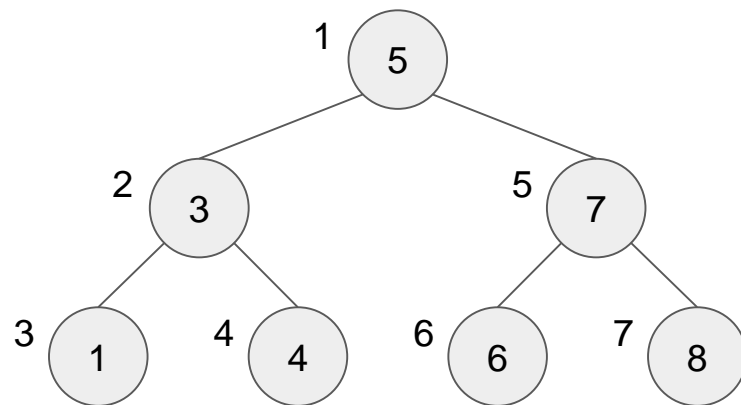
Дерево

Это структура данных, представленная в виде набора связанных узлов.



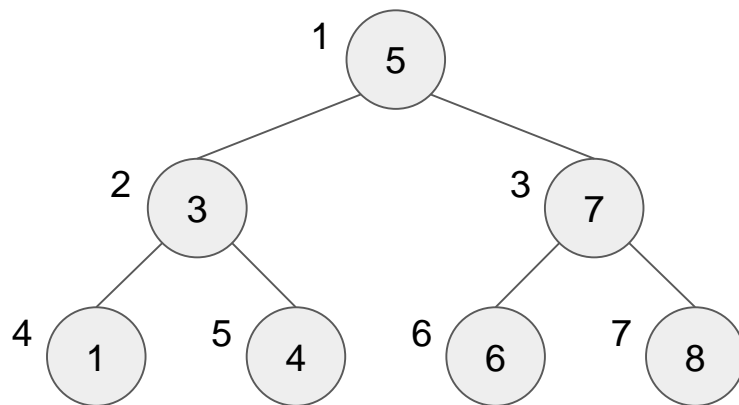
Обход в глубину

Рекурсивный обход узлов дерева



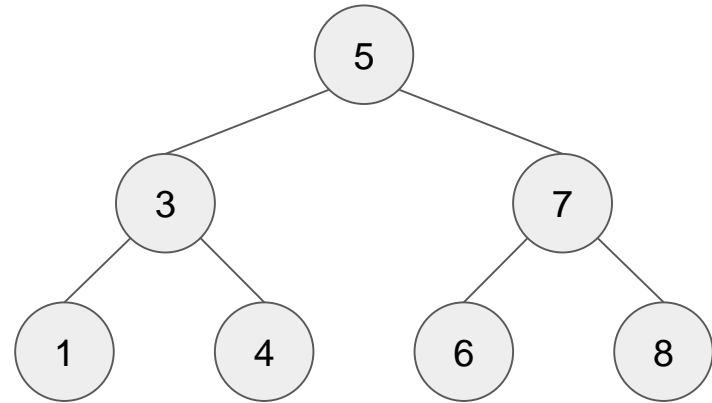
Обход в ширину

Циклический обход узлов дерева



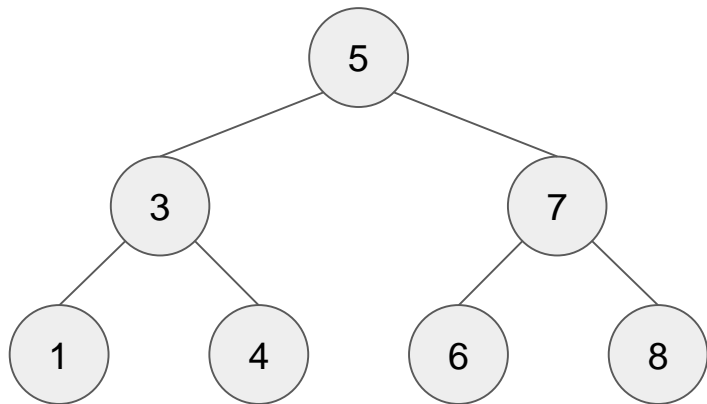
Бинарное дерево

Бинарным деревом называют частный случай дерева, где все элементы обязательно строго уникальны, каждый родитель имеет не более 2 детей, при этом левый ребенок всегда меньше родителя, а правый – больше.

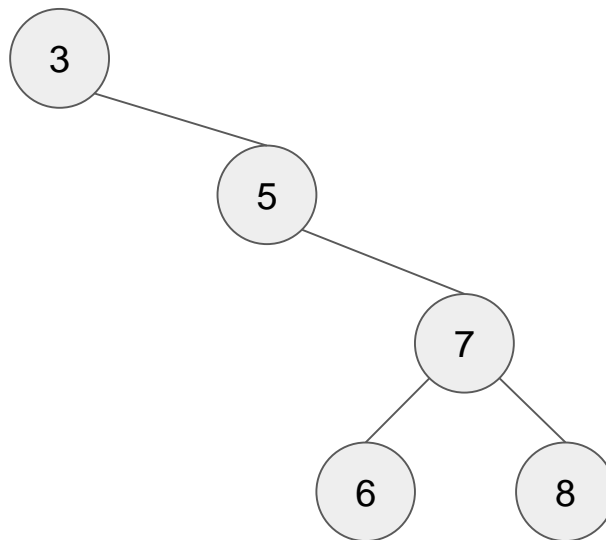


Сбалансированное дерево

Сбалансированным деревом называют частный случай бинарного дерева, у которого выполняется следующее требование: *для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу.*

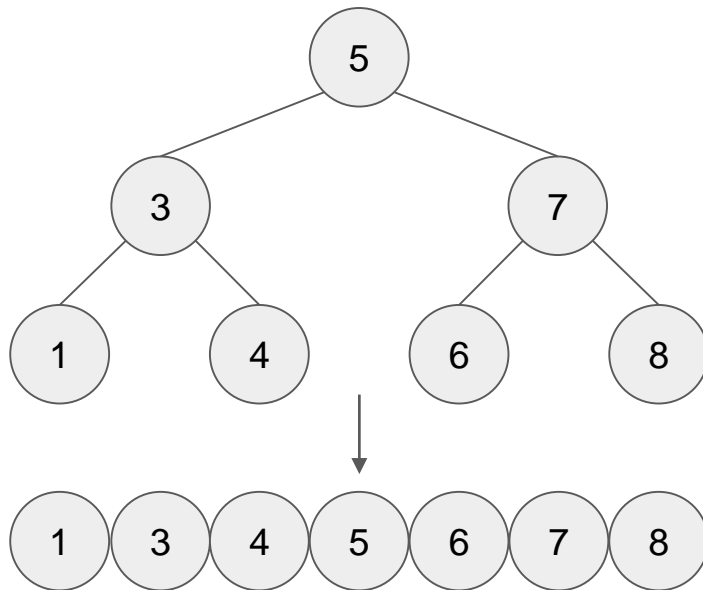


А такое дерево не считается сбалансированным



Поиск по сбалансированному дереву

Сбалансированное дерево дает нам идеальную структуру для бинарного поиска – корень такого дерева — это его центральный элемент – количество элементов справа и слева от него различается не более чем на единицу, что характерно для выбора стартовой позиции в бинарном поиске. Таким образом, сложность поиска по сбалансированному дереву составляет **$O(\log n)$** , что дает очень высокую производительность.



Хеш-таблица

Структура данных, представляющая собой ассоциативный массив использующий хеш-функцию для выполнения операций добавления, удаления и поиска элементов.

В свою очередь **ассоциативным массивом** называют структуру данных, которая хранит пары ключ – значение, где ключ каждой пары является уникальным в пределах всего массива данных.

Важной особенностью хеш-таблиц является, при некотором разумном допущении, получить сложность каждой из перечисленных операций равной **$O(1)$**



Этапы проектирования

Массив объектов “ключ + значение”

a: test	b: test	c: data	d: info
---------	---------	---------	---------

Минусы подхода:

- Сложность $O(n)$ для поиска элемента по ключу



Хеш-функция

Хеш-функцией называется специальный алгоритм, позволяющий преобразовать входные данные произвольного размера и состава в битовую строку фиксированной длины.

Популярные хеш-алгоритмы:

- MD5
- SHA-1
- SHA-256
- SHA-384
- SHA-512



Этапы проектирования

Использование хэш-функции для ключа как вычисление индекса элемента - $f(key) = i$

1	2	3	4
a: test	b: test	c: data	d: info

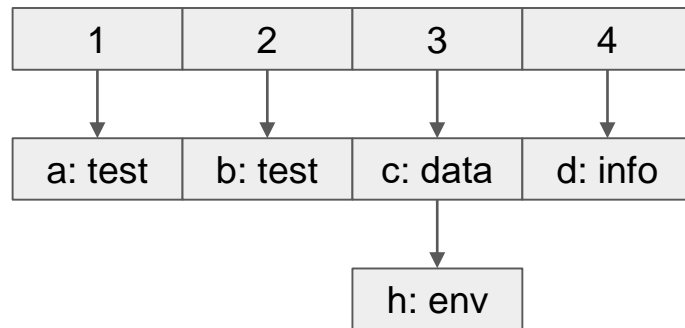
Минусы подхода:

- Адресное пространство в 8 байт — это не только положительные значения – Integer включает в себя значения от $-2\ 147\ 483\ 648$ до $2\ 147\ 483\ 647$, а обратиться к индексу с отрицательным номером невозможно.
- Каждый раз выделять в адресном пространстве место под массив размером $2\ 147\ 483\ 647$ очень расточительно с точки зрения эксплуатации. Скорее всего у вас не будет элементов хотя бы на одну сотую часть этой размерности, а значит адреса в памяти будут заняты впустую
- Наличие коллизий не позволит однозначно занять 1 ячейку массива строго одним элементом. На один и тот же индекс может претендовать несколько элементов, чей хеш-код даст одно и тоже значение



Этапы проектирования

Учитываем наличие коллизий. Храним в ячейке не один объект, а список объектов



Минусы подхода:

- Перебор списка имеет сложность $O(n)$

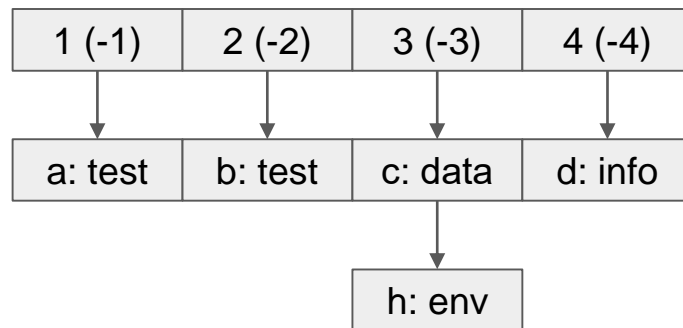
Плюсы подхода:

- При корректной хеш-функции шанс коллизий минимален, а значит и размер списка будет не более нескольких значений и сложностью перебора можно пренебречь



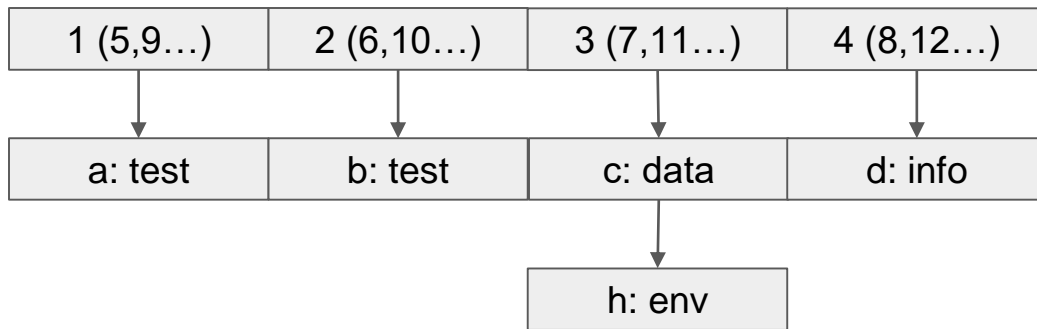
Этапы проектирования

Используем результат хеш-функции по модулю, чтобы избежать отрицательных индексов



Этапы проектирования

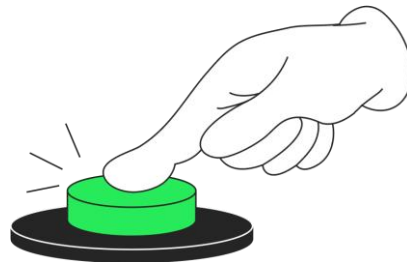
Уменьшаем количество бакетов и используем остаток от деления на их количество для определения индекса. Так при 4 бакетах и результате хеш-функции 9, объект будет добавлен в ячейку с индексом $(9\%4=1)$ 1



Этапы проектирования





Чтобы сохранять сложность поиска $O(1)$, с ростом объема данных в хеш-таблице необходимо корректировать количество бакетов и проводить перераспределение данных между ними.

Благодаря этому, каждый бакет будет содержать не более нескольких значений, что позволит пренебречь $O(n)$ сложностью поиска по списку и сохранить $O(1)$ сложность поиска по индексу массива для данных любого объема






Итоги урока

-  Познакомились со структурой данных “дерево”
-  Узнали о бинарных и сбалансированных деревьях
-  Научились обходить узлы дерева горизонтально и вертикально
-  Узнали внутреннюю структуру хеш-таблицы

Итоги курса



Спасибо 
за внимание

