Фундаментальные структуры данных, которые вам следует знать для практического программирования

или к чему быть готовым на собеседовании

Источник перевода

Никлаус Вирт, швейцарский ученый-информатик, в 1976 году написал книгу под названием «*Алгоритмы + Структуры данных = Программы*».

Через 40 с лишним лет это тождество остается в силе. Вот почему соискатели, желающие стать программистами, должны продемонстрировать, что знают структуры данных и умеют их применять.

Практически во всех задачах от кандидата требуется глубокое понимание структур данных. При этом не столь важно, выпускник ли вы (закончили университет или курсы программирования), либо у вас за плечами десятки лет опыта.

Иногда в вопросах на интервью прямо упоминается та или иная структура данных, например, «дано двоичное дерево». В других случаях задача формулируется более завуалированно, например, «нужно отследить, сколько у нас книг от каждого автора».

Изучение структур данных — незаменимое дело, даже если вы просто стараетесь профессионально совершенствоваться на нынешней работе. Начнем с основ.

Что такое структура данных?

Если коротко, структура данных — это контейнер, информация в котором скомпонована характерным образом. Благодаря такой «компоновке», структура данных будет эффективна в одних операциях и неэффективна — в других. Наша цель — разобраться в структурах данных таким образом, чтобы вы могли выбрать из них наиболее подходящую для решения конкретной стоящей перед вами задачи.

Зачем нужны структуры данных?

Поскольку структуры данных используются для хранения информации в упорядоченном виде, а данные — самый важный феномен в информатике, истинная ценность структур данных очевидна.

Не важно, какую именно задачу вы решаете, так или иначе вам придется иметь дело с данными, будь то зарплата сотрудника, биржевые котировки, список продуктов для похода в магазин или обычный телефонный справочник.

В зависимости от конкретного сценария, данные нужно хранить в подходящем формате. У нас в распоряжении — ряд структур данных, обеспечивающих нас такими различными форматами.

Наиболее распространенные структуры данных

Сначала давайте перечислим наиболее распространенные структуры данных, а затем разберем каждую по очереди:

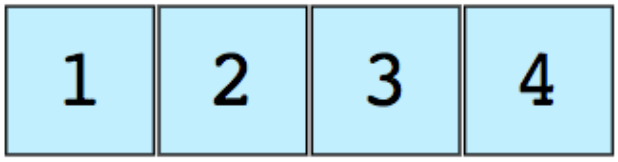
1. Массивы
2. Стеки
3. Очереди
4. Связные списки
5. Деревья

6. Графы
7. Боры (в сущности, это тоже деревья, но их целесообразно рассмотреть отдельно).
8. Хеш-таблицы

Массивы

Массив — это простейшая и наиболее распространенная структура данных. Другие структуры данных, например, стеки и очереди, производны от массивов.

Здесь показан простой массив размером 4, содержащий элементы (1, 2, 3 и 4).



Массив

Каждому элементу данных присваивается положительное числовое значение, именуемое индексом и соответствующее положению этого элемента в массиве. В большинстве языков программирования элементы в массиве нумеруются с 0.

Существуют массивы двух типов:

- Одномерные (такие, как показанный выше)
- Многомерные (массивы, в которые вложены другие массивы)

Простейшие операции с массивами

- Insert — вставляем элемент на позицию с заданным индексом
- Get — возвращаем элемент, занимающий позицию с заданным индексом

- Delete – удаляем элемент с заданным индексом
- Size – Получаем общее количество элементов в массиве

Вопросы по массивам, часто задаваемые на собеседованиях

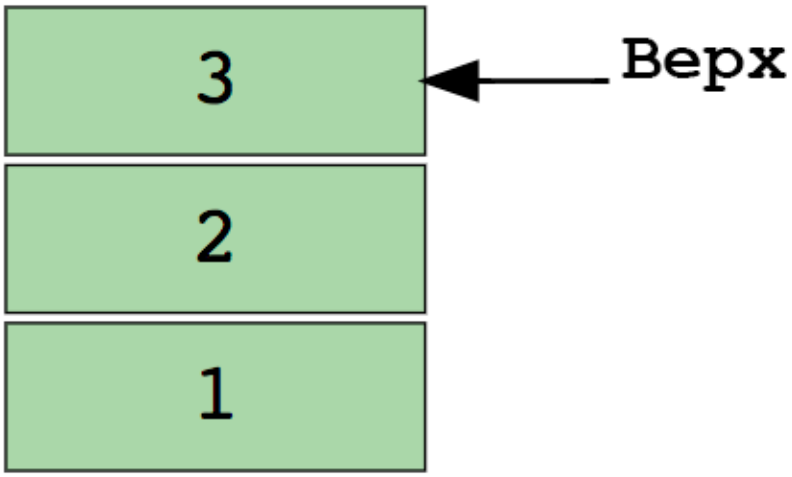
- Найти второй минимальный элемент массива
- Найти неповторяющиеся целые числа в массиве
- Объединить два отсортированных массива
- Переупорядочить положительные и отрицательные значения в массиве

Стеки

Всем известна знаменитая опция «Отмена», предусмотренная почти во всех приложениях. Задумывались когда-нибудь, как она работает? Смысл такой: в программе сохраняются предшествующие состояния вашей работы (количество сохраняемых состояний ограничено), причем, они располагаются в памяти в таком порядке: последний сохраненный элемент идет первым. Одними массивами такую задачу не решить. Именно здесь нам пригодится стек.

Стек можно сравнить с высокой стопкой книг. Если вам нужна какая-то книга, лежащая около центра стопки, вам сначала придется снять все книги, лежащие выше. Именно так работает принцип [[LIFO]] (Последним пришел – первым вышел).

Так выглядит стек, содержащий три элемента данных (1, 2 и 3), где 3 находится сверху – поэтому будет убран первым:



Стэк

Простейшие операции со стеком:

- Push – Вставляет элемент в стек сверху
- Pop – Возвращает верхний элемент после того, как удалит его из стека
- isEmpty – Возвращает true, если стек пуст
- Top – Возвращает верхний элемент, не удаляя его из стека

Вопросы о стеке, часто задаваемые на собеседованиях

- Вычислить постфиксное выражение при помощи стека
- Отсортировать значения в стеке
- Проверить сбалансированные скобки в выражении

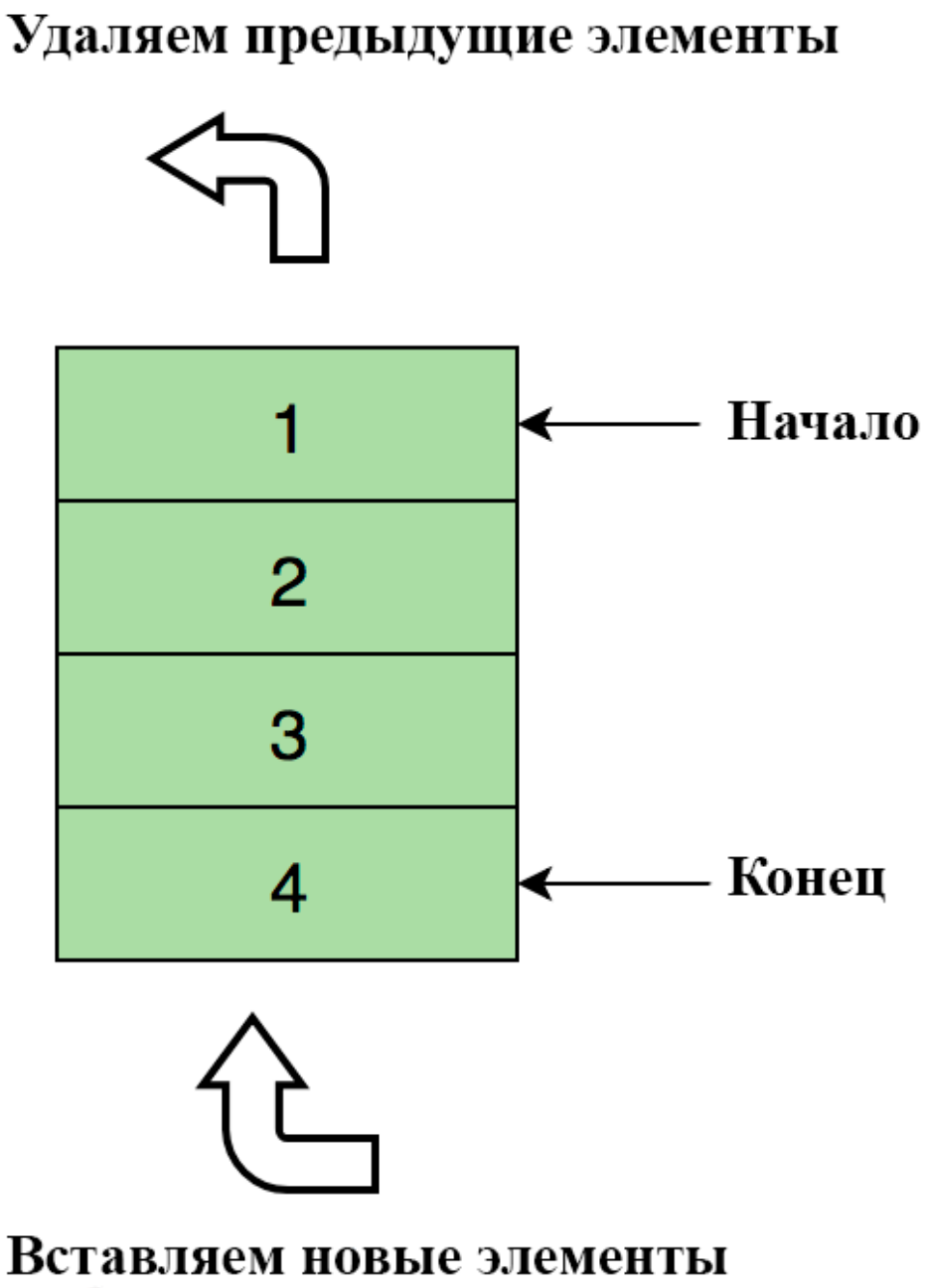
Очереди

Очередь, как и стек – это линейная структура данных, элементы в которой хранятся в последовательном порядке. Единственное существенное отличие между стеком и очередью заключается в том, что в очереди вместо [[LIFO]] действует принцип [[FIFO]] (Первым пришел – первым вышел).

Идеальный реалистичный пример очереди – это и есть очередь

покупателей в билетную кассу. Новый покупатель становится в самый хвост очереди, а не в начало. Тот же, кто стоит в очереди первым, первым приобретет билет и первым ее покинет.

Вот изображение очереди с четырьмя элементами данных (1, 2, 3 и 4), где 1 идет первым и первым же покинет очередь:



Очереди

Простейшие операции с очередью

- Enqueue() — Добавляет элемент в конец очереди
- Dequeue() — Удаляет элемент из начала очереди
- isEmpty() — Возвращает true, если очередь пуста
- Top() — Возвращает первый элемент в очереди

Вопросы об очередях, часто задаваемые на собеседованиях

- Реализуйте стек при помощи очереди
- Обратите первые k элементов в очереди
- Сгенерируйте двоичные числа от 1 до n при помощи очереди

Связный список

Связный список — еще одна важная линейная структура данных, на первый взгляд напоминающая массив. Однако, связный список отличается от массива по выделению памяти, внутренней структуре и по тому, как в нем выполняются базовые операции вставки и удаления.

Связный список напоминает цепочку узлов, в каждом из которых содержится информация: например, данные и указатель на следующий узел в цепочке. Есть головной указатель, соответствующий первому элементу в связном списке, и, если список пуст, то он направлен просто на null (ничто).

При помощи связных списков реализуются файловые системы, хеш-таблицы и списки смежности.

Вот так можно наглядно изобразить внутреннюю структуру связного списка:

Связный список

Существуют такие типы связных списков:

- Односвязный список (однонаправленный)
- Двусвязный список (двунаправленный)

Простейшие операции со связными списками:

- *InsertAtEnd* – Вставляет заданный элемент в конце связного списка
- *InsertAtHead* – Вставляет заданный элемент в начале (с головы) связного списка
- *Delete* – Удаляет заданный элемент из связного списка
- *DeleteAtHead* – Удаляет первый элемент в связном списке
- *Search* – Возвращает заданный элемент из связного списка
- *isEmpty* – Возвращает true, если связный список пуст

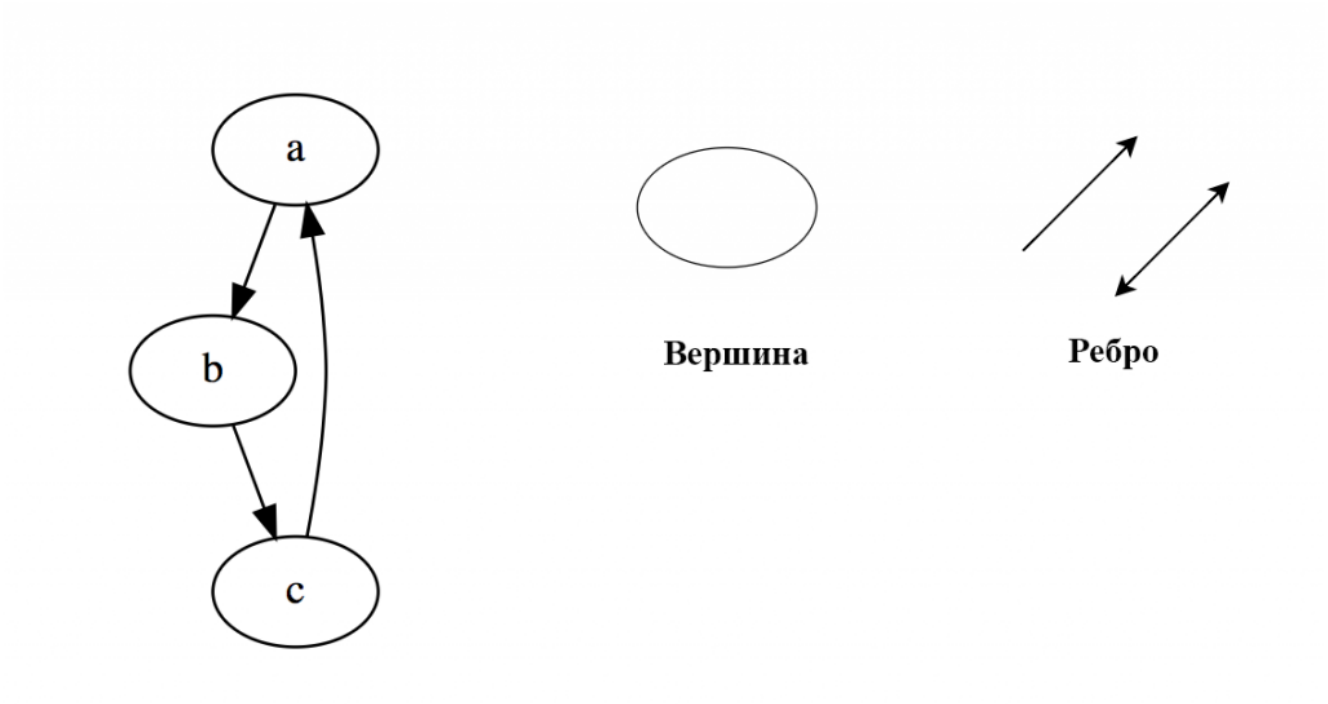
Вопросы о связных списках, часто задаваемые на собеседованиях:

- Обратите связный список
- Найдите петлю в связном списке
- Возвратите N-ный узел с начала связного списка
- Удалите из связного списка дублирующиеся значения

Графы

Граф – это множество узлов, соединенных друг с другом в виде сети. Узлы также называются вершинами. **Пара (x,y)** называется **ребром,** это означает, что вершина **x** соединена с вершиной **y**. Ребро может иметь вес/стоимость – показатель, характеризующий,

насколько затратен переход от вершины x к вершине y.



Графы

Типы графов:

- Неориентированный граф
- Ориентированный граф

В языке программирования графы могут быть двух видов:

- Матрица смежности
- Список смежности

Распространенные алгоритмы обхода графа:

- Поиск в ширину
- Поиск в глубину

Вопросы о графах, часто задаваемые на собеседованиях:

- Реализуйте поиск в ширину и поиск в глубину
- Проверьте, является граф деревом или нет
- Подсчитайте количество ребер в графе

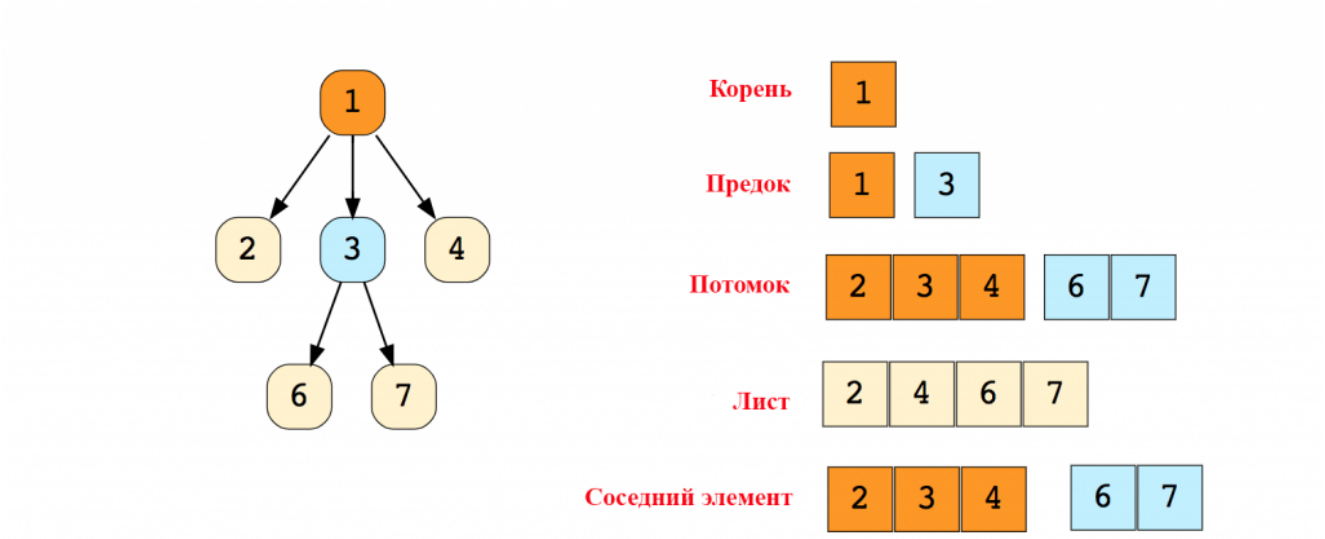
- Найдите кратчайший путь между двумя вершинами

Деревья

Дерево — это иерархическая структура данных, состоящая из вершин (узлов) и ребер, которые их соединяют. Деревья подобны графам, однако, ключевое отличие дерева от графа таково: в дереве не бывает циклов.

Деревья широко используются в области искусственного интеллекта и в сложных алгоритмах, выступая в качестве эффективного хранилища информации при решении задач.

Вот схема простого дерева и базовая терминология, связанная с этой структурой данных:



Деревья

Существуют деревья следующих типов:

- [[N-арное дерево]];
- [[Сбалансированное дерево]];
- [[Двоичное дерево]];
- [[Двоичное дерево поиска]];
- [[АВЛ-дерево]];
- [[Красно-черное дерево]];
- [[2–3 дерево]].

Из вышеперечисленных деревьев чаще всего используются двоичное дерево и двоичное дерево поиска.

Вопросы о деревьях, часто задаваемые на собеседованиях:

Найдите высоту двоичного дерева

Найдите k-ное максимальное значение в двоичном дереве поиска

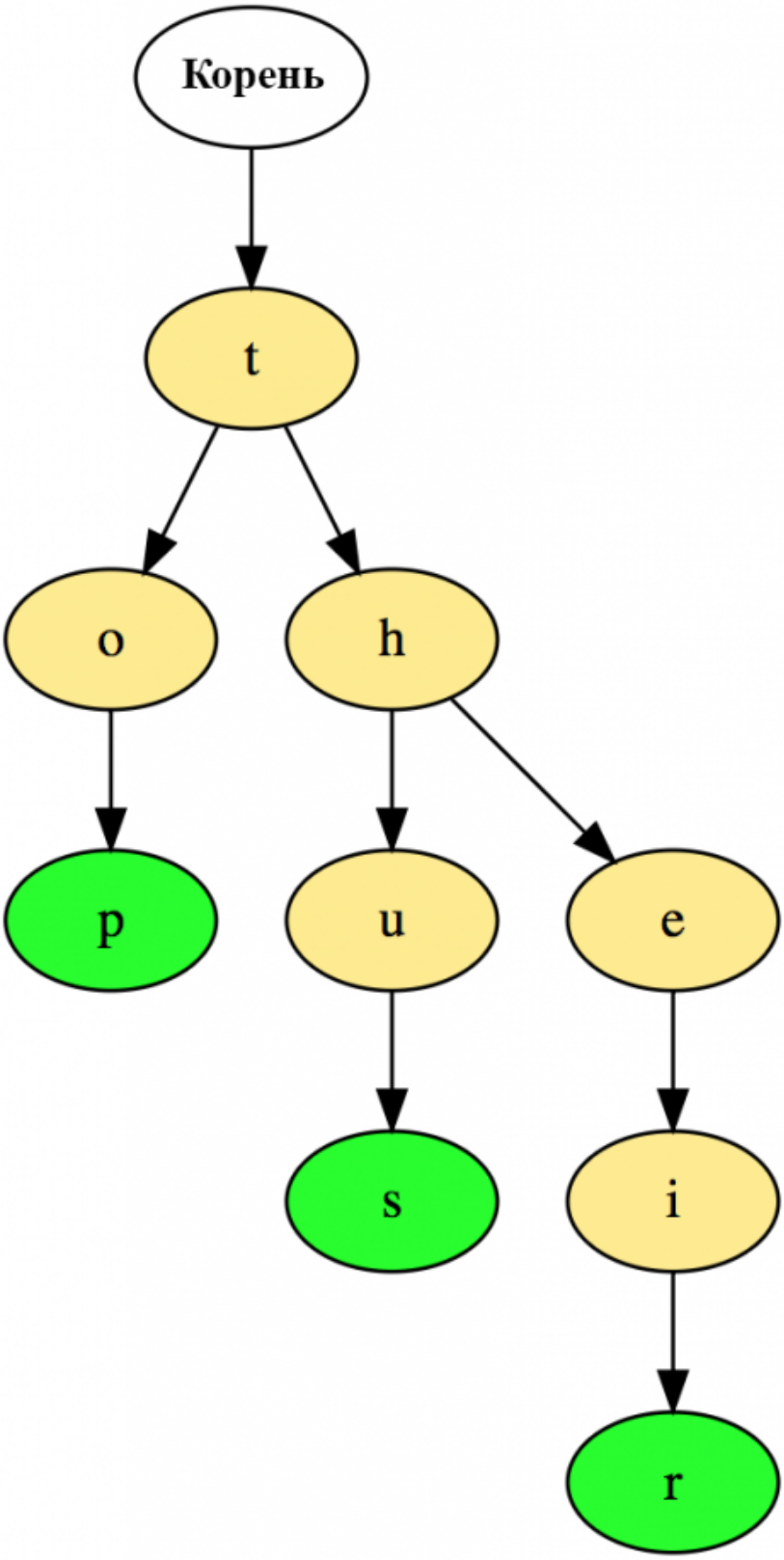
Найдите узлы, расположенные на расстоянии “k” от корня

Найдите предков заданного узла в двоичном дереве

Бор

Бор, также именуемый «префиксное дерево» — это древовидная структура данных, которая особенно эффективна при решении задач на строки. Она обеспечивает быстрое извлечение данных и чаще всего применяется для поиска слов в словаре, автозавершений в поисковике и даже для IP-маршрутизации.

Вот как три слова top (верх), thus (следовательно), and their (их) хранятся в бору:



Бор

Слова располагаются в направлении сверху вниз, и зеленые узлы

«p», «s» и «r» завершают, соответственно, слова «top», «thus» и «their».

Вопросы о борах, часто задаваемые на собеседованиях:

- Подсчитайте общее количество слов, сохраненных в бору
- Выведите на экран все слова, сохраненные в бору
- Отсортируйте элементы массива при помощи бора
- Постройте слова из словаря, воспользовавшись бором
- Создайте словарь T9

Хеш-таблица

Хеширование – это процесс, применяемый для уникальной идентификации объектов и сохранения каждого объекта по заранее вычисленному индексу, именуемому его «ключом». Таким образом, объект хранится в виде «ключ-значение», а коллекция таких объектов называется «словарь». Каждый объект можно искать по его ключу. Существуют разные структуры данных, построенные по принципу хеширования, но чаще всего из таких структур применяется **хеш-таблица**.

Как правило, хеш-таблицы реализуются при помощи массивов.

Производительность хеширующей структуры данных зависит от следующих трех факторов:

- Хеш-функция
- Размер хеш-таблицы
- Метод обработки коллизий

Ниже показано, как хеш отображается на массив. Индекс этого массива вычисляется при помощи хеш-функции.

Хеш-таблица

Вопросы о хешировании, часто задаваемые на собеседованиях:

- Найдите симметричные пары в массиве
- Отследите полную траекторию пути
- Найдите, является ли массив подмножеством другого массива
- Проверьте, являются ли массивы непересекающимися

Выше описаны восемь важнейших структур данных, которые определенно нужно знать, прежде чем идти на собеседование по программированию.

Удачи и интересного обучения!

[Источник перевода]