Вопросы из СДО

1. Понятие алгоритма

Алгоритм – это последовательность шагов, предназначенная для решения конкретной задачи или достижения цели.

2. Основные свойства алгоритма

- Дискретность: Алгоритм состоит из отдельных шагов.
- Понятность: Каждый шаг должен быть понятен исполнителю (компьютеру).
- Определённость: Результат каждого шага однозначно определяется.
- Результативность: Алгоритм должен приводить к результату за конечное число шагов.
- Массовость: Алгоритм должен быть применим для решения целого класса задач.

3. Способы описания алгоритма

- Словесный: Обычный текст, как рецепт.
- Графический: Блок-схемы, где каждый блок - это действие.
- Программный: На языке программирования, понятном компьютеру.

4. Линейные алгоритмы

Линейные алгоритмы выполняются последовательно, шаг за шагом, без ветвлений и циклов.

5. Ветвящиеся алгоритмы

Ветвящиеся алгоритмы включают условия, по которым выполнение алгоритма может пойти по разным путям.

6. Циклические алгоритмы

Циклические алгоритмы выполняют определённые действия многократно, пока выполняется заданное условие.

7. Решение уравнения методом деления отрезка пополам

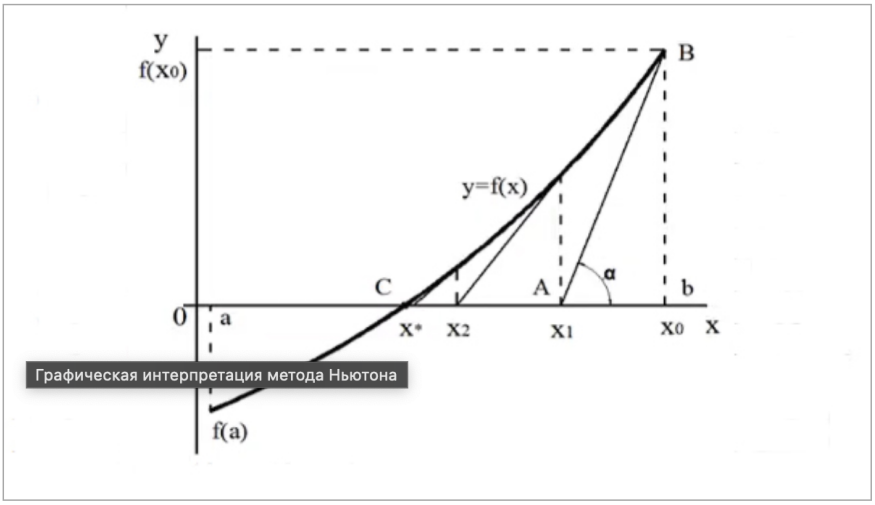
Метод половинного деления относится к серии алгоритмов, построенных по принципу «разделяй и властвуй». Он применяется для поиска корней уравнений. Допустим, есть у нас некоторая функция f(x), известно, что функция монотонна на некотором интервале [L,R]. Монотонность — обязательное требование для использования этого алгоритма, оно означает, что функция либо только возрастает на этом интервале, либо — убывает. В общем, на интервале нет перегибов функции, т.е. точек, в которых производная равна нулю.

Тогда, если на концах интервала функция имеет разные знаки — она обязательно пересекает горизонтальную ось, т.е. имеет корень.

Если $f(L) * f(R) > 0$ — значит функция на этом интервале корня не имеет.

Как же найти где именно находится этот корень? — опять же итеративно. Возьмем точку посередине интервала ($B = L + (R-L)/2$) — по знаку f(B) можно определить где именно находится корень (правее этой точки или левее). Если $f(L)*f(B) < 0$ — то корень находится на интервале [L,B] при этом заменим R на B и повторим процесс. В противном случае корень находится на [B,R]. Вычисления продолжаются до тех пор, пока интервал поиска корня не сузится достаточно сильно, т.е. пока $|R-L| > Eps$ (Eps – заданная погрешность, например 0.001). Схематически метод проиллюстрирован на рисунке ниже

8. Решение уравнения методом касательных



Графическая интерпретация метода Ньютона

От x_0 узнаём значение функции. В этой точке проводим касательную. Касательная пересекает ось X, и мы получаем новую точку x_1. И начинаем всё сначала. Числа x_0, x_1, x_2 и т.д. приближаются к корню уравнения.

Выведем формулу для x_n.

Уравнение касательной:
$y = f(x_0) + f'(x_0)(x-x_0)$

Приравняем к нулю (пересечение с осью X) и выразим x.

$x_1=x_0 - f(x_0) / f'(x_0)$
$x_{n+1}=x_n - f(x_n) / f'(x_n)$

9. Решение уравнения методом хорд

Метод хорд — итерационный численный метод приближённого нахождения корня уравнения.

Половинное деление не учитывает никаких свойств функции F(x), а эта функция может нести в себе очень полезную информацию. Метод хорд предполагает следующее. От точек, ограничивающих кривую (заданные концы отрезка L и R), строится хорда, затем определяется точка её пересечения с осью абсцисс, точка пересечения становится новой границей отрезка, после чего строится новая хорда. Итерационный процесс задается следующей формулой:

$$x_{n+1} = x_n - \frac{(b - x_n) f(x_n)}{f(b) - f(x_n)}$$

---

10. Метод наименьших квадратов. Регрессия

---

11. Алгоритм Евклида для поиска наибольшего общего делителя

Хорошее решение проблемы поиска наибольшего общего делителя двух чисел нашел еще Евклид. В самом простом случае алгоритм Евклида применяется к паре положительных целых чисел и формирует новую пару, которая состоит из меньшего числа и разницы между большим и меньшим числом. Процесс повторяется, пока числа не станут равными. Найденное число и есть наибольший общий делитель исходной пары. Блок-схема алгоритма выглядит следующим образом:

Найти наибольший общий делитель для чисел 128 и 96.

$128 - 96 = 32$

$96 - 32 = 64$

$64 - 32 = 32$

$32 - 32 = 0$

$128 / 96 = 1$ (остаток 32)

$96 / 32 = 3$

Ответ: 32

12. Понятие о факторизации числа

Факторизация числа — это процесс разложения числа на простые множители.

13. Алгоритм Ферма факторизации

Алгоритм Ферма для разделения числа на 2 множителя

Пусть $n = p \cdot q$ – известное целое число, являющееся произведением двух неизвестных простых чисел p и q, которые требуется найти. Большинство современных методов факторизации основано на идее, предложенной еще Пьером Ферма, заключающейся в поиске пар натуральных чисел A и B таких, что выполняется соотношение:

$$n = A^2 - B^2. \tag{2.20}$$

Алгоритм Ферма может быть описан следующим образом:

1. Вычислим целую часть от квадратного корня из n:

$$m = \lceil \sqrt{n} \rceil.$$

2. Для $x = 1, 2, \ldots$ будем вычислять значения

$$q(x) = (m + x)^2 - n, \tag{2.21}$$

до тех пор, пока очередное значение $q(x)$ не окажется равным полному квадрату.

3. Пусть $q(x)$ является полным квадратом, например, числа B: $q(x) = B^2$. Определим $A = m + x$, откуда из равенства $A^2 - n = B^2$ найдем $n = A^2 - B^2 = (A + B) \cdot (A - B)$, и искомые делители p и q вычисляются, как $p = A + B$, $q = A - B$.

14. Алгоритмы поиска простых чисел. Решето Эратосфена

Решето Эратосфена

Алгоритм:

1. Выписать подряд все целые числа от двух до n (2, 3, 4, ..., n).
2. Пусть переменная p=2— первому простому числу.
3. Зачеркнуть в списке числа, кратные p (это будут числа: 2p, 3p, 4p, ...).
4. Найти первое не зачёркнутое число в списке, большее чем p, и присвоить переменной p это значение.
5. Повторять шаги 3 и 4, пока возможно.

Теперь все не зачёркнутые числа в списке — это простые числа от 2 до n.

Пример для n = 24.
a = 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

1 Шаг. Находим все числа, кратные 2 и удаляем их:

a = 2 3 5 7 9 11 13 15 17 19 21 23

2 Шаг. Находим все числа, кратные 3 и удаляем их:

a = 2 3 5 7 11 13 17 19 23

3 Шаг. Находим все числа, кратные 5 и удаляем их:

Таких чисел нет, поэтому на этом шаге алгоритм останавливается.

Теперь a состоит только из простых чисел, которые меньше 24.

15. Числа Мерсенна. Тест Люка-Лемера

Число Мерсе́нна — число вида 2n-1, где n — натуральное число; такие числа примечательны тем, что некоторые из них являются простыми при больших значениях n. Названы в честь французского математика Маре́на Мерсенна, исследовавшего их свойства в XVII веке.

Для проверки числа Мерсенна на простоту существует простой тест Люка-Лемера. Это алгоритм, который предполагает на первом этапе построение последовательности по следующему правилу:

Пусть p — простое нечётное. Число Мерсенна $M_p = 2^p - 1$ простое тогда и только тогда, когда оно делит нацело $(p-1)$-й член последовательности

$$4,\ 14,\ 194,\ 37634,\ \ldots$$ [2],

задаваемой рекуррентно: $S_k = \begin{cases} 4 & k = 1, \\ S_{k-1}^2 - 2 & k > 1. \end{cases}$

16. Псевдослучайные числа

В различных технических и математических приложениях часто необходим ряд случайных чисел. Числа называются случайными в силу того, что в их ряду нет никакой закономерности, зная некоторое количество последовательных чисел ряда, нельзя вычислить следующее.

Алгоритм фон Неймана для генерации псевдослучайных чисел состоит в следующем. Предположим, что некоторое случайное, достаточно большое число уже известно. Определить такое число совершенно не представляет проблему: так как оно первое, то оно может быть просто любым. Пусть, например, нас интересуют 5-значные случайные числа. Возьмем в качестве первого число 14 563. Возведем его в квадрат, получим 212 080 969. Возьмем из середины пять цифр 20 809. Это и будет следующее случайное число. Псевдослучайность получаемой последовательности очевидна. Каждое следующее число совершенно однозначно определяется предыдущим. Но нас это смущать не должно. Если неизвестно, как получены числа, то они выглядят вполне случайно. Но у метода есть более серьезный недостаток. Если последовательность продолжить, то может проявиться так называемый период. Периодом называется строго повторяющаяся последовательность чисел. Впрочем, для алгоритма фон Неймана можно подобрать исходное число, дающее период только после очень большого количества шагов.

17. Метод Карацубы для быстрого умножения двух чисел

Метод Карацубы — это алгоритм для быстрого умножения двух больших чисел, который основывается на принципе декомпозиции чисел на более мелкие части и рекурсивного использования умножения. Этот метод позволяет значительно уменьшить количество элементарных операций по сравнению с классическим методом умножения.

Метод Карацубы

Карацуба высказал достаточно простую идею, позволяющую умножать числа существенно быстрее. Его идея основана на следующем очевидном соотношении:

$$4ab = (a+b)^2 - (a-b)^2, \text{ откуда } ab = \frac{(a+b)^2 - (a-b)^2}{4}$$

Основная идея – разбить исходное число на два меньших, это должно дать экономию. Идея действительно выигрышная. Число X можно разными способами представить в виде суммы двух: $X = X_1 + X_2$. Найдем такое представление, что длина X_1 равна длине X и половина младших разрядов X_1 равна нулю. Тогда длина X_2 равна половине длины X. Пример:

$$4578394\ 9002338 = 45783940000000 + 9002338 = 4578394 * 10^7 + 9002338.$$

При этом если числа являются разной длины или имеют нечетную длину, то целесообразно добавить старший «нулевой разряд». Например, если даны числа 37656 и 6567863, то их следует рассматривать в таком виде:

$00037656 = 0003*10^4 + 7656;$

$06567863 = 0656*10^4 + 7863;$

Будем рассматривать произведение двух чисел в виде (ax+b)(cx+d), где $x=10^k$ (в примере выше k=4). Имеет место следующая цепочка равенств

$$(ax+b)(cx+d) = acx^2 + (ad+bc)x + bd =$$
$$= acx^2 + \left((a+b)(c+d) - ac - bd\right)x + bd\,.$$

Для наших двух чисел имеем:

$(0003*10^4 + 7656)* (0656*10^4 + 7863)=$

$0003*0656*10^8+((0003+7656)*(0656+7863)-0003*0656-7656*7863)*10^4+7656*7863$

В этом выражении произведения 0003*0656 и 7656*7863 повторяются дважды, а всего нужно посчитать 3 произведения четырехразрядных чисел:

1) 0003*0656
2) 7656*7863
3) (0003+7656)*(0656+7863) = 7659*8519

Каждое произведение четырехразрядных чисел можно разбить на произведения двухразрядных, а произведения двухразрядных – на произведения одноразрядных чисел.

18. Алгоритм быстрого возведения в степень

Быстрое возведение в степень

Алгоритм быстрого возведения в степень — алгоритм, предназначенный для возведения числа x в натуральную степень n за меньшее число умножений, чем это требуется в определении.

Пусть $m = (m_k m_{k-1} \dots m_1 m_0)_2$ — двоичное представление степени n. Тогда $n = m_k \cdot 2^k + m_{k-1} \cdot 2^{k-1} + \dots + m_1 \cdot 2 + m_0$, где $m_k = 1, m_i \in \{0, 1\}$ и $x^n = x^{((\dots((m_k \cdot 2 + m_{k-1}) \cdot 2 + m_{k-2}) \cdot 2 + \dots) \cdot 2 + m_1) \cdot 2 + m_0}$.

Функция быстрого возведения в степень

```
function Power(value, pow: int): int
  int result = 1
  while (pow > 0)
    if pow mod 2 == 1
      result *= value
    value *= value
    pow /= 2;
  return result;
```

19. Понятие о рекурсии

Рекурсия — это такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения ее операторов обращается сама к себе. То есть в теле функции она вызывает саму себя.

20. Числа Фибоначчи

Числа Фибоначчи — это последовательность чисел, где каждое следующее число равно сумме двух предыдущих. Обычно последовательность начинается с чисел 0 и 1.

21. Задача о Ханойской башне

Задача о Ханойской башне — это классическая головоломка, которая состоит в перемещении всех дисков с одного стержня на другой, используя третий стержень в качестве промежуточного, с условием, что на более крупный диск нельзя класть меньший.

22. Динамическое программирование

Динамическое программирование (DP) — это метод решения сложных задач путём разбиения их на более простые подзадачи и сохранения результатов этих подзадач для последующего использования.

Основные принципы

1. **Разбиение задачи**: Исходная задача разбивается на несколько подзадач.
2. **Сохранение результатов**: Результаты каждой подзадачи сохраняются для последующего повторного использования.
3. **Комбинирование результатов**: Результаты подзадач комбинируются для получения решения исходной задачи.

23. Задача сортировки массива

Сортировка массива — это процесс упорядочивания элементов массива в определенном порядке, таком как по возрастанию или убыванию.

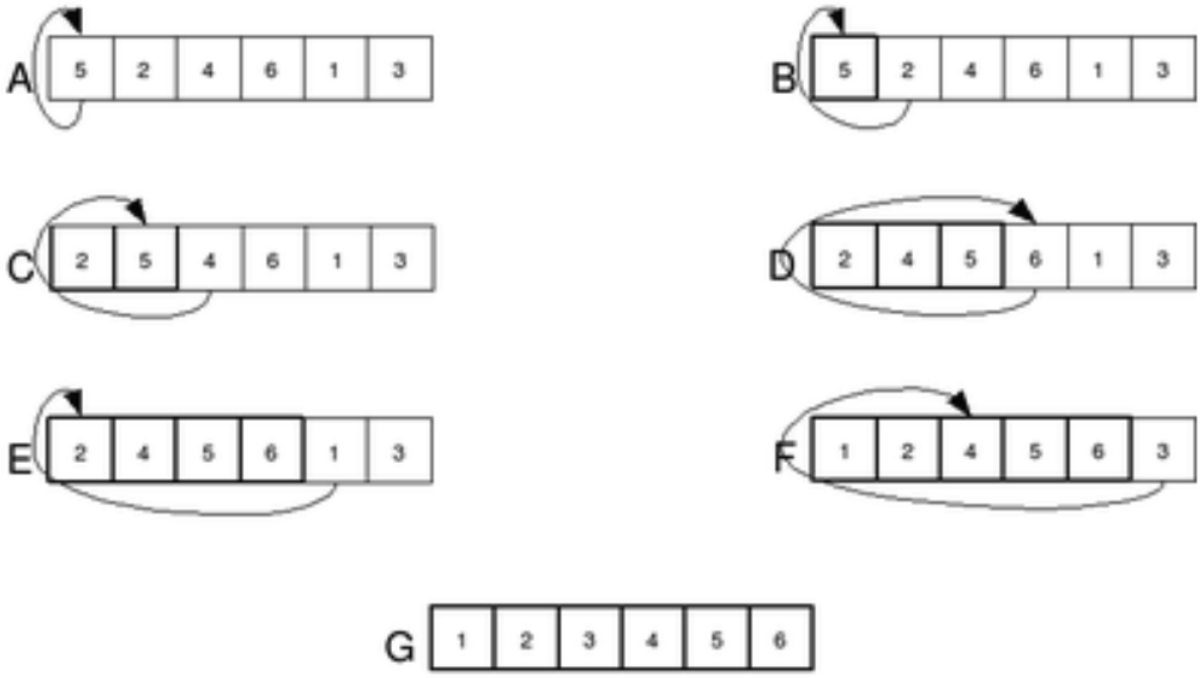
24. Алгоритм пузырьковой сортировки

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется

перестановка элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

25. Алгоритм сортировки вставками

Алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.



26. Алгоритм сортировки выбором

Шаги алгоритма:

1. находим номер минимального значения в текущем списке
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

27. Алгоритм шейкерной сортировки

Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства. Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения. Во-вторых, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный

элемент сдвигается только на одну позицию вправо. Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (то есть части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

28. Алгоритм сортировки Шелла

Алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами. При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при d=1 (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше)

29. Алгоритм быстрой сортировки

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена, известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

Общая идея алгоритма состоит в следующем:

- Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность (см. ниже).
- Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие»[2].
- Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения

30. Алгоритм сортировки

Алгоритм сортировки — это алгоритм для упорядочивания элементов в массиве.

31. Основные структуры данных

Структура данных — это способ организации информации для более эффективного использования. В программировании структурой обычно называют набор данных, связанных определённым образом.

1. Массив (Array)
2. Динамический массив (Dynamic array)
3. Связный список (Linked list)
4. Стек (Stack)
5. Очередь (Queue)

32. Массив и динамический массив

Массив

Одна из самых простых структур данных, которая встречается чаще всего. Именно на массивах основаны многие другие структуры данных: списки, стеки, очереди. Для простоты восприятия можно считать, что массив — это таблица. Каждый его элемент имеет индекс — «адрес», по которому этот элемент можно извлечь. В большинстве языков программирования индексы начинаются с нуля. То есть первый элемент массива имеет индекс не [1], а [0]. Данные в массиве можно просматривать, сортировать и изменять с помощью специальных операций.

Массивы бывают двух видов:

- **Одномерные.** У каждого элемента только один индекс. Можно представить это как строку с данными, где одного номера достаточно, чтобы чётко определить положение каждой переменной.
- **Многомерные.** У каждого элемента два или больше индексов. По сути, это комбинация из нескольких одномерных массивов, то есть вложенная структура.

Как применяют массивы:

- В качестве блоков для более сложных структур данных. Массивы предусмотрены в синтаксисе большинства языков программирования, и на их основе удобно строить другие структуры.
- Для хранения несложных данных небольших объёмов.
- Для сортировки данных.

Динамический массив

В классическом массиве размер задан заранее — мы точно знаем, сколько в нём индексов. А динамический массив — это тот, у которого размер может изменяться. При его создании задаётся максимальная величина и количество заполненных элементов. При добавлении новых элементов они сначала заполняются до максимальной величины, а при превышении сразу создаётся новый массив, с большей максимальной величиной. Элементы в динамический массив можно добавлять без ограничений и куда угодно. Однако, если добавлять их в середину, остальные придётся

сдвигать, что занимает много времени. Поэтому лучше всего динамический массив работает при добавлении элементов в конце.

Как применяют динамические массивы:

- В качестве блоков для структур данных
- Для хранения неопределённого количества элементов

33. Связный список

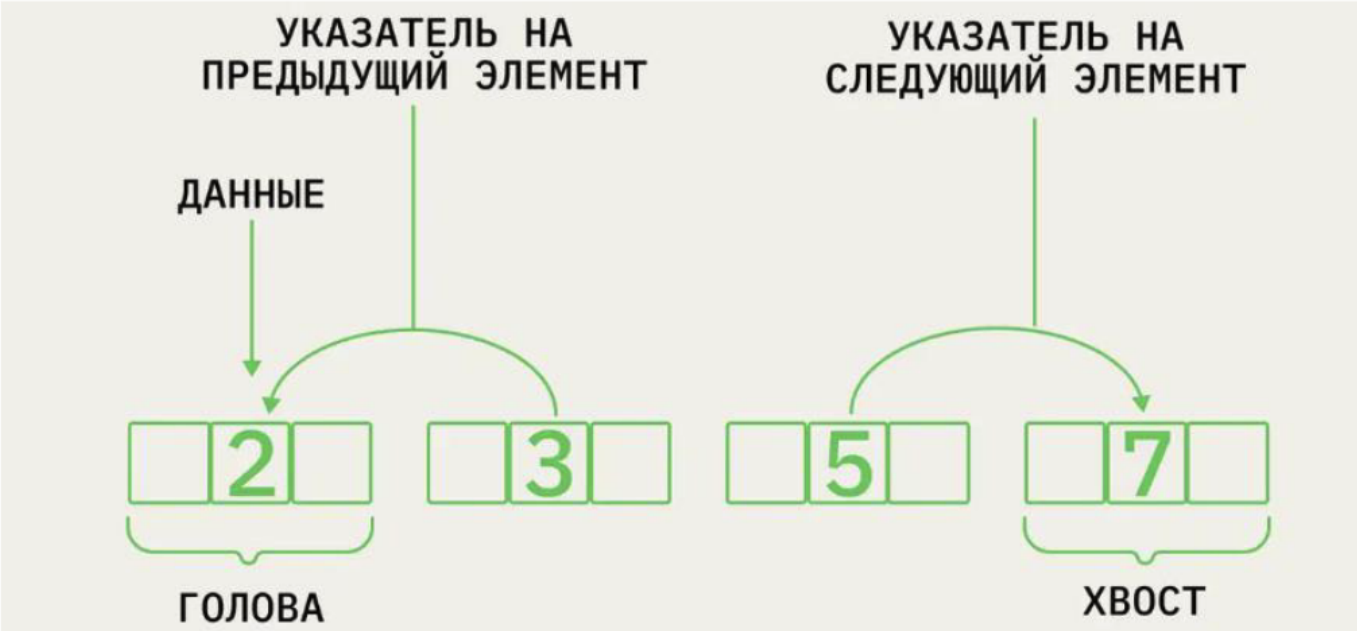
Ещё одна базовая структура данных, которую, как и массивы, используют для реализации других структур. Связный список — это группа из узлов. В каждом узле содержатся:

- Данные
- Указатель или ссылка на следующий узел
- В некоторых списках — ещё и ссылка на предыдущий узел

В итоге получается список, у которого есть чёткая последовательность элементов. При этом сами элементы более разрозненны, чем в массиве, поскольку хранятся отдельно. Быстро перемещаться между элементами списка помогают указатели.

Как применяют связные списки:

- Для построения более сложных структур данных
- Для реализации файловых систем
- Для формирования хэш-таблиц
- Для выделения памяти в динамических структурах данных

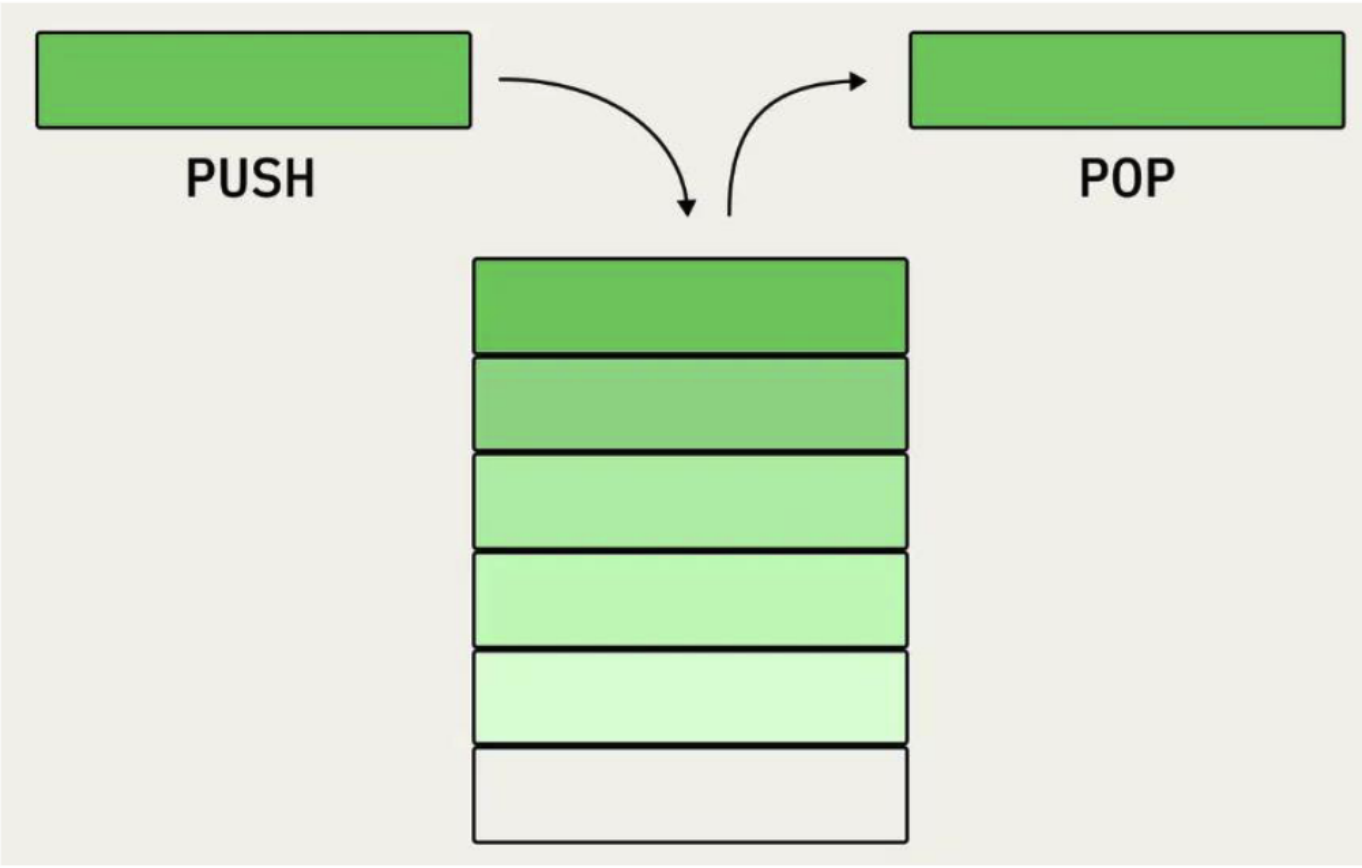


34. Стек

Эта структура данных позволяет добавлять и удалять элементы только из начала. Она работает по принципу LIFO — Last In, First Out (англ. «последним пришёл — первым ушёл»). Последний добавленный в стек элемент должен будет покинуть его раньше остальных.

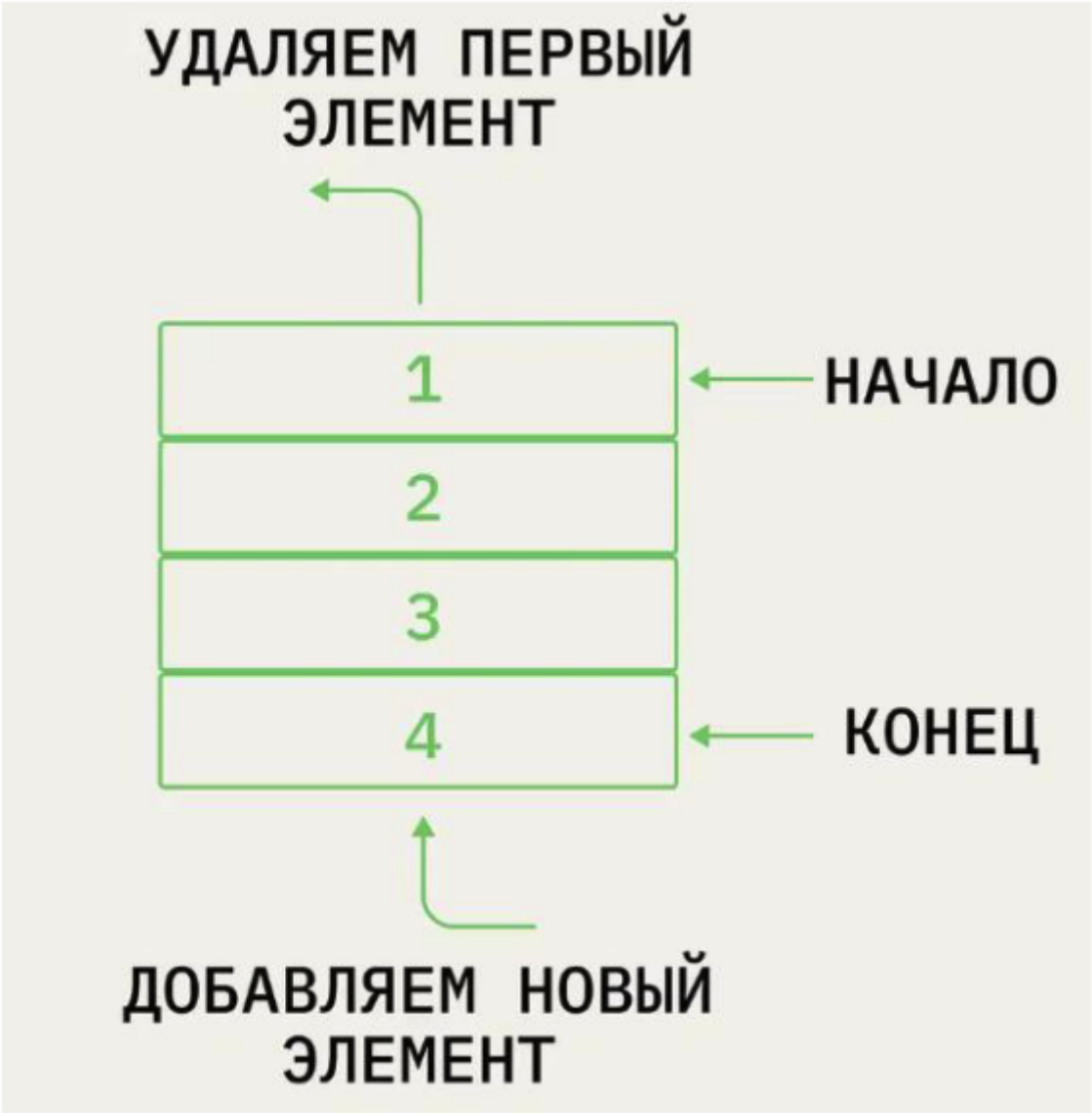
Как применяют стеки:

- Для реализации рекурсии
- Для вычислений постфиксных значений
- Для временного хранения данных, например истории запросов или изменений



35. Очередь

Этот вид структуры представляет собой ряд данных, как и стек. Но в отличие от него она работает по принципу FIFO — First In, First Out (англ. «первым пришёл — первым ушёл»). Данные добавляют в конец, а извлекают из начала.

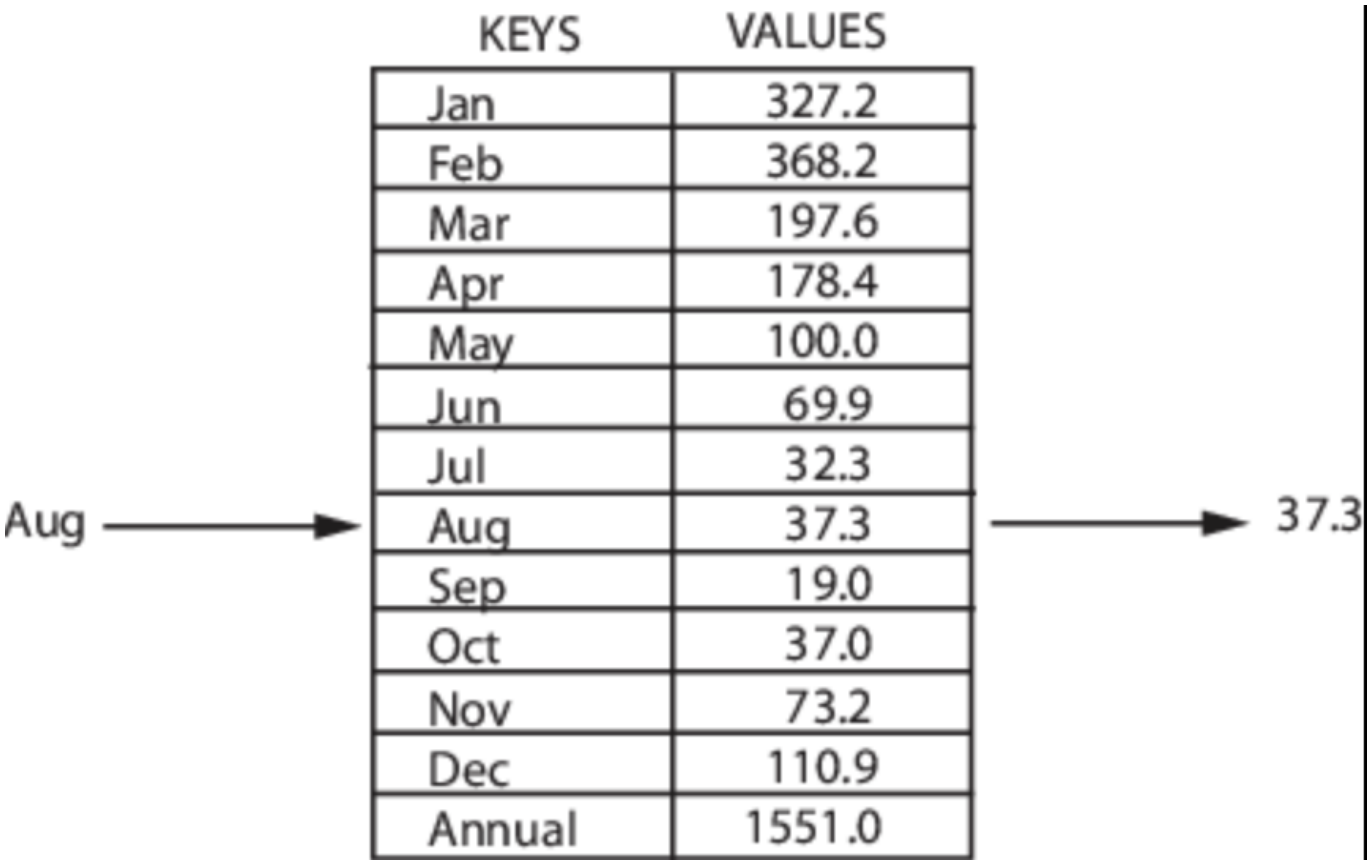


36. Множество

Множество в контексте структур данных представляет собой коллекцию элементов, где каждый элемент уникален (то есть в множестве не может быть повторяющихся элементов).

37. Карта

Карта - это структура данных, содержащая пары ключ-значение.



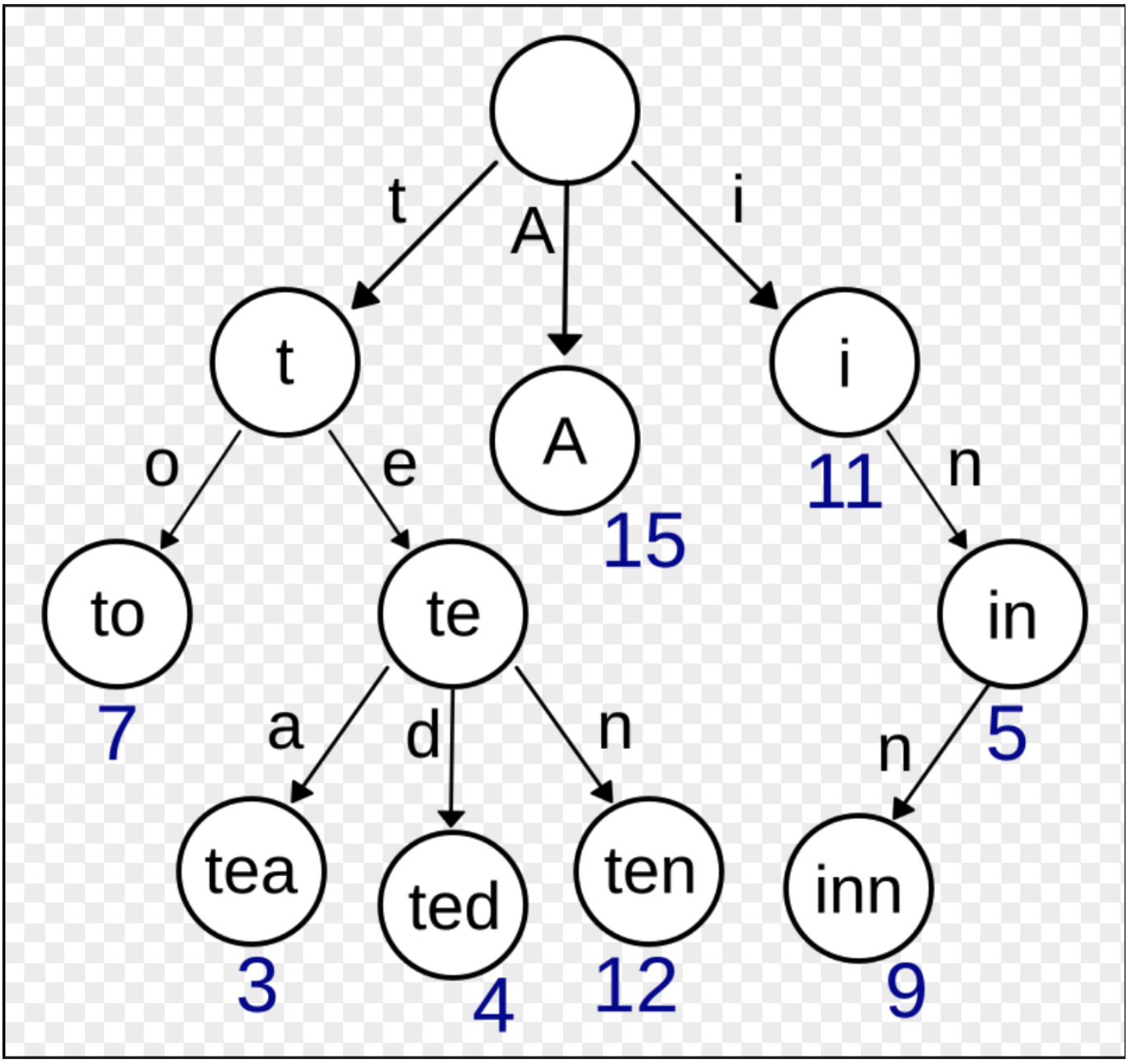
38. Двоичное дерево поиска

Двоичное дерево поиска (англ. binary search tree, BST) — двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

- оба поддерева — левое и правое — являются двоичными деревьями поиска;
- у всех узлов левого поддерева произвольного узла X значения ключей данных меньше либо равны, нежели значение ключа данных самого узла X;
- у всех узлов правого поддерева произвольного узла X значения ключей данных больше, нежели значение ключа данных самого узла X.

39. Префиксное дерево поиска

Trie или префиксное дерево — это особый вид дерева поиска, в котором для ключей узлов обычно используются строки.



40. Сложность алгоритма. Сложность в среднем и худшем случае. O-нотация

Сложность алгоритма — это мера, которая описывает, как быстро возрастает время выполнения алгоритма по мере увеличения размера входных данных. Она может быть выражена с помощью O-нотации, указывающей верхнюю границу времени выполнения алгоритма в зависимости от размера входных данных.

Примеры O-нотации

- **O(1)**: Константная сложность. Время выполнения алгоритма не зависит от размера входных данных. Пример: доступ к элементу в массиве по индексу.
- **O(log n)**: Логарифмическая сложность. Время выполнения алгоритма растёт логарифмически с ростом размера входных данных. Пример: бинарный поиск в отсортированном массиве.
- **O(n)**: Линейная сложность. Время выполнения алгоритма линейно зависит от размера входных данных. Пример: последовательный поиск в неотсортированном массиве.

- **O(n^2)**: Квадратичная сложность. Время выполнения алгоритма пропорционально квадрату размера входных данных. Пример: сортировка пузырьком.
- **O(2^n)**: Экспоненциальная сложность. Время выполнения алгоритма растёт экспоненциально с ростом размера входных данных. Пример: решение задачи коммивояжёра перебором всех возможных путей.

Виды сложности

1. **В среднем случае (Average Case Complexity)**: Это оценка среднего времени выполнения алгоритма для всех возможных входных данных размера *n*. Сложность в среднем случае обычно используется в анализе случайных алгоритмов, где предполагается равновероятное возникновение всех возможных входов.
2. **В худшем случае (Worst Case Complexity)**: Это оценка максимального времени выполнения алгоритма для любого входного набора размера *n*. Сложность в худшем случае является наиболее пессимистичной оценкой времени выполнения, так как она определяет верхнюю границу времени выполнения алгоритма при наихудших условиях входных данных.

42. АВЛ-дерево

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1

Для АВЛ-деревьев сбалансированность определяется разностью высот правого и левого поддеревьев любого узла. Если эта разность по модулю не превышает 1, то дерево считается сбалансированным. Данное условие проверяется после каждого добавления или удаления узла, и определен минимальный набор операций перестройки дерева, который приводит к восстановлению свойства сбалансированности, если оно оказалось нарушено.

	В среднем	В худшем случае
Расход памяти	O(n)	O(n)
Поиск	O(log n)	O(log n)
Вставка	O(log n)	O(log n)
Удаление	O(log n)	O(log n)

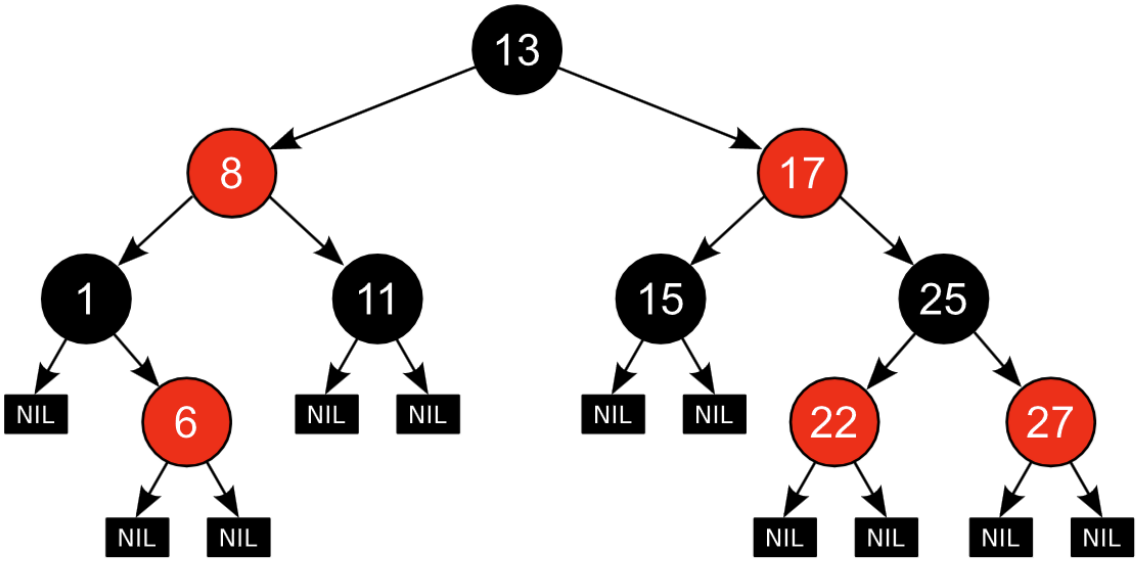
43. Красно-черное дерево

КЧ-деревья – это двоичные деревья поиска, каждый узел которых хранит дополнительное поле color, обозначающее цвет: красный или черный.

Свойства КЧ-деревьев:

1. каждый узел либо красный, либо черный;
2. каждый лист (фиктивный) – черный;
3. если узел красный, то оба его сына – черные;
4. все пути, идущие от корня к любому фиктивному листу, содержат одинаковое количество черных узлов;
5. корень – черный.

Черной высотой узла называется количество черных узлов на пути от этого узла к узлу, у которого оба сына – фиктивные листья. Черная высота дерева – черная высота его корня.



	В среднем	В худшем случае
Расход памяти	O(n)	O(n)
Поиск	O(log n)	O(log n)
Вставка	O(log n)	O(log n)
Удаление	O(log n)	O(log n)

44. Splay-дерево

Самоперестраивающееся дерево – это двоичное дерево поиска, которое, в отличие от предыдущих двух видов деревьев не содержит дополнительных служебных полей в структуре данных (баланс, цвет и т.п.). Оно позволяет находить быстрее те данные, которые использовались недавно. Самоперестраивающееся дерево было придумано Робертом Тарьяном и Даниелем Слейтером в 1983 году.

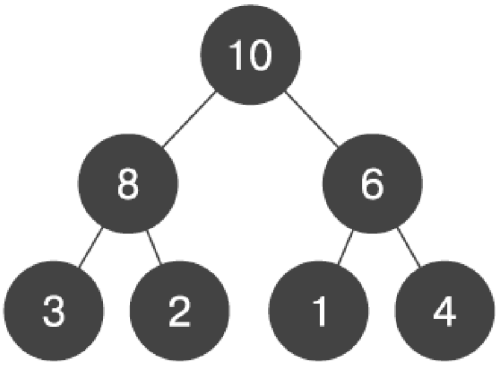
Идея самоперестраивающихся деревьев основана на принципе перемещения найденного узла в корень дерева.

	В среднем	В худшем случае
Расход памяти	O(n)	O(n)
Поиск	O(log n)	O(log n)
Вставка	O(log n)	O(log n)
Удаление	O(log n)	O(log n)

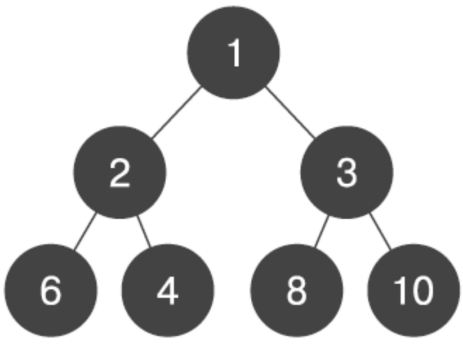
45. Двоичная куча

Куча (англ. heap) — это специализированная структура данных типа дерево, которая удовлетворяет свойству кучи: если B является узлом-потомком узла A, то ключ(A) ≥ ключ(B). Из этого следует, что элемент с наибольшим ключом всегда является корневым узлом кучи, поэтому иногда такие кучи называют max-кучами (в качестве альтернативы, если сравнение перевернуть, то наименьший элемент будет всегда корневым узлом, такие кучи называют min-кучами). Не существует никаких ограничений относительно того, сколько узлов-потомков имеет каждый узел кучи, хотя на практике их число обычно не более двух.

Двоичная куча — та, что создана с помощью двоичного дерева. Иногда ее называют пирамидой.



В максимальной куче узел с наивысшим приоритетом является корнем



В минимальной куче узел с наименьшим приоритетом является корнем

46. Хэш-таблица

Хеш-таблица — линейная структура данных, в которой хранятся пары «ключ — значение» с уникальными ключами.

47. Алгоритм шифрования Цезаря

Шифр Цезаря – это преобразование информации методом замены букв на другие, стоящие от данных через определенное количество символов в алфавите.

48. Блочные алгоритмы шифрования

Блочные алгоритмы шифрования представляют собой методы шифрования, которые оперируют не отдельными символами или битами, а блоками данных фиксированного размера. Эти алгоритмы широко используются для защиты данных в современных криптографических приложениях. Вот несколько основных блочных алгоритмов шифрования: AES, DES, ГОСТ 28147-89

49. Понятие о графе

Граф — это абстрактная математическая структура, которая представляет собой набор вершин (узлов) и рёбер (связей) между этими вершинами.

50. Алгоритм Прима

Алгоритм Прима

Шаг 1 Выберите произвольную вершину и ребро, соединяющее ее с ближайшим (по весу) соседом.

Шаг 2 Найдите не присоединенную (еще) вершину, ближе всего лежащую к одной из присоединенных, и соедините с ней.

Шаг 3 Повторяйте шаг 2 до тех пор пока все вершины не будут присоединены.

51. Число связности

Граф называют *связным*, если любую пару его вершин соединяет какой-нибудь маршрут. Любой общий граф можно разбить на подграфы, каждый из которых окажется связным. Минимальное число таких связных компонент называется *числом связности* графа и обозначается через $c(G)$. Вопросы связности имеют важное значение в приложениях теории графов к компьютерным сетям. Следующий алгоритм применяется для определения числа связности графа.

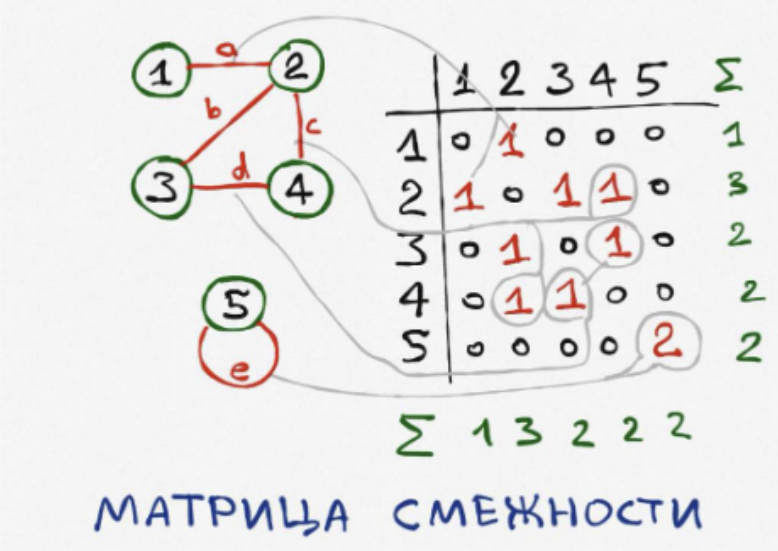
52. Матрицы смежности и инцидентности графа

1. *Матрица смежности*

Это самый популярный и расточительный способ представления графа в памяти. Его уместно использовать, если количество рёбер велико, порядка V2.

Для хранения рёбер используется двумерная матрица размерности [V, V], каждый [a, b] элемент которой равен 1, если вершины a и b являются смежными и 0 в противном случае.

В случае неориентированного графа матрица является симметричной относительно главной диагонали, а сумма каждой строчки и каждого столбца равна степени вершины. В связи с этим, при записи рёбер-петель в матрицу необходимо записывать число 2.



- ✓ Сложность по памяти: O(V2).
- ✓ Сложность перечисления всех рёбер: O(V2).
- ✓ Сложность перечисления всех вершин, смежных с a: O(V).
- ✓ Сложность проверки смежности вершин a и b: O(1).

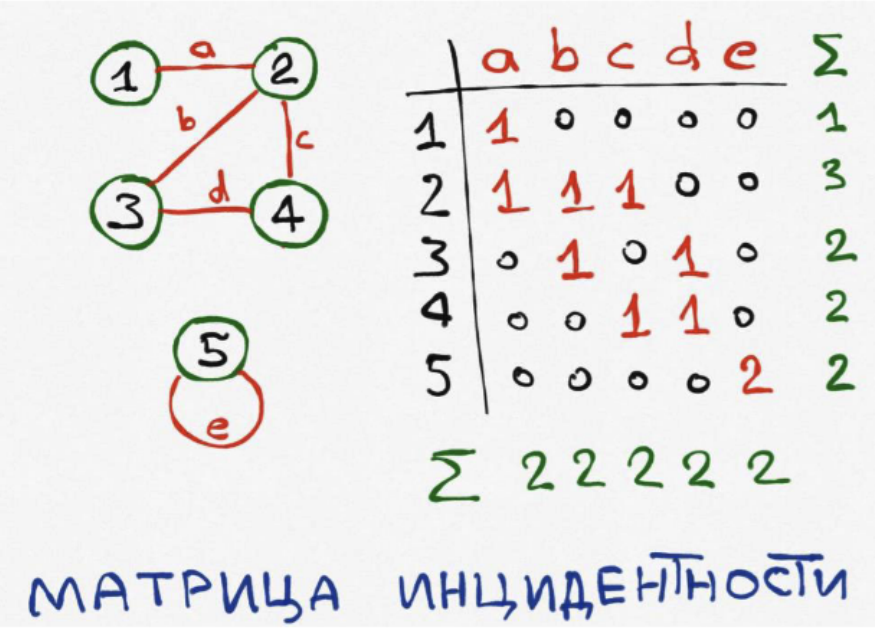
Если граф взвешенный, то элементом матрицы смежности является вес ребра (вместо единицы пишется вес соответствующего ребра)

2. *Матрица инцидентности*

Это самый расточительный способ хранения графа, его уместно использовать, если количество рёбер невелико.

Для хранения используется двумерная матрица размера [V, E], в каждом столбце которой записано одно ребро таким образом: напротив вершин, инцидентных этому ребру, записаны 1, в остальных случаях 0.

Таким образом, сумма чисел в каждом столбце равна 2, а сумма чисел в строчке a равна степени вершины a.



- ✓ Сложность по памяти: O(V x E).
- ✓ Сложность перечисления всех рёбер: O(V x E) - хоть каждое ребро и хранится в отдельном столбце, но для получения информации об инцидентных ему вершинах нужно перебрать все числа в столбце.
- ✓ Сложность перечисления всех вершин, смежных с a: O(V x E).
- ✓ Сложность проверки смежности вершин a и b: O(E) - достаточно пройтись по строчкам a и b в поисках двух единиц.

53. Алгоритм ближайшего соседа

- **Выбор начальной вершины**: Выбирается начальная вершина
- **Поиск ближайшего соседа**: Текущая вершина соединяется с ближайшей ещё не посещённой вершиной. После выбора ближайшей вершины она помечается как посещённая и становится текущей.
- **Повторение**: Шаг 2 повторяются до тех пор, пока все вершины не будут посещены.
- **Возвращение в начальную вершину**: После посещения всех вершин, последняя выбранная вершина соединяется с начальной вершиной, завершая цикл.

54. Расширенный алгоритм Евклида

Расширенный алгоритм Евклида — это расширение алгоритма Евклида, которое вычисляет, кроме наибольшего общего делителя (НОД) целых чисел **a** и **b**, ещё и коэффициенты соотношения Безу, то есть целые **x** и **y**, такие что

$$ax + by = \text{НОД}(a, b).$$

Пусть **d** - наибольший общий делитель чисел **a** и **b**. Тогда выражение **ax+by** всегда кратно **d**. Оказывается, что можно подобрать такие числа **x** и **y**, что **ax+by=d**. Эту задачу решает расширенный алгоритм Евклида. Рассмотрим его рекурсивную реализацию. Пусть функция gcdex получает на вход числа **a** и **b** и возвращает кортеж из трех чисел **d**, **x**, **y**, где **d** - наибольший общий делитель **a** и **b**, а **x** и **y** - такие целые числа, что **ax+by=d**.
Условием окончания рекурсии является **b=0**. В этом случае **d=a, x=1, y=0**. Если же **b≠0**, то вызовем функцию рекурсивно для чисел **b** и **a%b** и получим ответ для исходных чисел.

55. Шифрование с открытым ключом

Создание открытого и секретного ключа

RSA-ключи генерируются следующим образом:

1) выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое);

2) вычисляется их произведение $n = p \cdot q$, которое называется модулем;

3) вычисляется значение функции Эйлера от числа n:

$\varphi(n) = (p-1) \cdot (q-1)$;

4) выбирается целое число

$1 < e < \varphi(n)$, взаимно простое со значением функции $\varphi(n)$;

число e называется открытой экспонентой (англ. public exponent);

обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например,

простые из чисел Ферма: 17, 257 или 65537, так как в этом случае время, необходимое для шифрования с использованием

быстрого возведения в степень, будет меньше;

слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.

5) вычисляется число

d, мультипликативно обратное к числу e по модулю $\varphi(n)$, то есть число, удовлетворяющее сравнению:

$d \cdot e \equiv 1 \pmod{\varphi(n)}$

(d называется секретной экспонентой; обычно оно вычисляется при помощи расширенного алгоритма Евклида);

6) пара (e,n) публикуется в качестве открытого ключа RSA (англ. RSA public key);

7) пара (d,n) играет роль закрытого ключа RSA (англ. RSA private key) и держится в секрете.

5) Шифрование

- Взять *открытый ключ (e, n)*
- Взять *открытый текст* m
- Зашифровать сообщение с использованием открытого ключа
- $c = E(m) = m^e \mod n$

6) Расшифрование

- Взять *шифрованный текст c*
- Взять *закрытый ключ (d, n)*
- Применить закрытый ключ для расшифрования сообщения^

$m = D(c) = c^d \mod n$

With ❤️ by NKTKLN