Теория алгоритмов.

Что такое алгоритм?

Что такое алгоритм?

- Есть список с числами $[a_1, a_2, \ldots, a_n]$
- Хотим отсортировать его и получить список $[b, b_2, \ldots, b_n]$ такой, что $b_1 \leq b_2 \leq \ \ldots \ \leq b_n$
- Как сортировать?

Алгоритм сортировки, который не нужно использовать в реальной жизни

- Есть список с числами $[a_1, a_2, \ldots, a_n]$
- Шаг 1: случайно перемешиваем значения списка
- Шаг 2: проверяем, что у нас получится сортированный список. Если не получилось, возвращаемся на шаг 1.

Что такое алгоритм?

- **Алгоритм** – точно заданный и понятный вычислителю (компьютеру) порядок действий, направленный на решение какой-либо задачи.

Что такое алгоритм?

- Одни и те же задачи могут решаться по-разному

Что такое алгоритм?

- Одни и те же задачи могут решаться по-разному
- Одно решение может оказаться эффективнее другого

Что такое алгоритм?

- Одни и те же задачи могут решаться по-разному
- Одно решение может оказаться эффективнее другого
- Причём иногда оно эффективно по одному критерию, но проигрывает в другом

Алгоритм сортировки №1 (bogosort)

- Есть список с числами $[a_1, a_2, \ldots, a_n]$
- Шаг 1: случайно перемешиваем значения списка
- Шаг 2: проверяем, что у нас получится сортированный список. Если не получилось, возвращаемся на шаг 1.

В чём недостаток данного алгоритма?

Алгоритм сортировки №1 (bogosort)

- Есть список с числами $[a_1, a_2, \ldots, a_n]$
- Шаг 1: случайно перемешиваем значения списка
- Шаг 2: проверяем, что у нас получится сортированный список. Если не получилось, возвращаемся на шаг 1.

В чём недостаток данного алгоритма?

Подсказка: какова вероятность, что шаг 1 завершится успехом (для простоты предположим, что все значения списка уникальны)?

Алгоритм сортировки №2

- Есть список с числами $[a_1, a_2, \ldots, a_n]$
- Шаг 1: создаём пустой список a_sorted, в котором будет храниться результат
- Шаг 2: находим минимальный элемент в a. Добавляем его в конец a_sorted и удаляем из a.
- Шаг 3: если в a ещё есть хотя бы один элемент, возвращаемся к шагу 1.

Чем этот алгоритм лучше алгоритма №1?

Теория алгоритмов помогает предсказать, какой объём ресурсов будет потреблять алгоритм с ростом объёма входных данных, без необходимости программировать и запускать этот алгоритм.

Теория алгоритмов помогает предсказать, какой объём ресурсов будет потреблять алгоритм с ростом объёма входных данных, без необходимости программировать и запускать этот алгоритм.

Чтобы делать такие предсказания, нужно понимать, как работает наш вычислитель, то есть принять некоторые допущения про то, как устроены хранение и обработка данных в современных компьютерах.

Модель данных и вычислений

- Память (оперативная память) – конечная последовательность битов. Биты поделены на куски – байты. У каждого куска есть числовой адрес.
- Регистры – несколько сотен полей фиксированной длины (32/64 бит) с очень быстрым доступом
- Процессор (CPU, central processing unit) – устройство, изменяющее байты в памяти и проводящее вычисления над данными в регистрах с помощью определённого набора инструкций

Модель данных и вычислений

- Компьютерная программа – набор последовательно исполняемых инструкций процессора (в частности, язык ассемблера – текстовое представление этих инструкций)
- Инструкции записывают или считывают данные из памяти. Считывание или запись данных в любой участок оперативной памяти занимает постоянное время (поэтому память называют RAM)
- У процессора есть некоторый набор базовых инструкций, позволяющий проводить вычисления над целыми и дробными числами.

Представление целых чисел в памяти

Положительное (неотрицательное) число можно представить в двоичной системе следующим образом:

$$n = 2^0 a_0 + 2^1 a_1 + \ldots + 2^{n-1} a_{n-1} = \sum_{i=0}^{n-1} 2^i a_i$$

Тогда $n_{10} = (a_{n-1} a_{n-2} \ldots a_0)_2$

Предположим, у нас есть n бит. Какое максимальное и минимальное неотрицательное число можно записать таким образом?

Представление целых чисел в памяти

Положительное (неотрицательное) число можно представить в двоичной системе следующим образом:

$$n = 2^0 a_0 + 2^1 a_1 + \ldots + 2^{n-1} a_{n-1} = \sum_{i=0}^{n-1} 2^i a_i$$

Тогда $n_{10} = (a_{n-1} a_{n-2} \ldots a_0)_2$

Предположим, у нас есть n бит. Какое максимальное и минимальное неотрицательное число можно записать таким образом? $[0;\ 2^n - 1]$. Это называется беззнаковым (unsigned) представлением числа.

А как быть с отрицательными числами?

- Пусть первый бит числа обозначает знак, а представление отрицательного числа получается следующим образом:

$$-x = not(x) + 1$$

Где not – функция, изменяющая каждый бит числа на противоположный.

Например, для 4-разрядных чисел

$$\text{-1 = not(1) + 1 = not}(0001_2) + 0001_2 = 1110_2 + 0001_2 = 1111_2$$

Это позволяет складывать и вычитать произвольные целые числа таким же образом, как и неотрицательные:

$$\text{-1 + 1 = } 1111_2 + 0001_2 = 0000_2 = 0$$

Какой диапазон чисел получается хранить в таком signed представлении, если на каждое число выделяется n бит?

А как быть с отрицательными числами?

- Пусть первый бит числа обозначает знак, а представление отрицательного числа получается следующим образом:

$$-x = not(x) + 1$$

Где not – функция, изменяющая каждый бит числа на противоположный.

Например, для 4-разрядных чисел

$$\text{-1 = not(1) + 1 = not}(0001_2) + 0001_2 = 1110_2 + 0001_2 = 1111_2$$

Это позволяет складывать и вычитать произвольные целые числа таким же образом, как и неотрицательные:

$$\text{-1 + 1 = } 1111_2 + 0001_2 = 0000_2 = 0$$

Какой диапазон чисел получается хранить в таком signed представлении, если на каждое число выделяется n бит? $[-2^{n-1};\ 2^{n-1}-1]$.

1) Что будет, если результат сложения двух чисел выходит за допустимый диапазон (допустим, процессор не умеет сообщать об ошибках)
2) Как устроен тип данных int в Python3?

А что насчёт дробных чисел?

- Далеко не все дробные числа можно точно представить в памяти компьютера. Почему?

А что насчёт дробных чисел?

- Далеко не все дробные числа можно точно представить в памяти компьютера
- Можно попытаться точно так же перевести число в двоичную систему:

$$n = \sum_{i=0}^{n-1} 2^i \ a_i + \sum_{i=1}^{m} a_{-i} \frac{1}{2^i}$$

Таким образом, фиксированное количество бит выделяется на целую (n) и дробную часть числа (m). Такое представление называется *числом с фиксированной запятой*.

Какими недостатками обладает такое представление?

Числа с плавающей запятой

Представим произвольное действительное число в виде:

$$x = -1^{\text{знак числа}} \times M \times 2^{E}$$

Здесь M – мантисса, дробное число с фиксированной запятой, а E – экспонента (показатель степени) – целое число.

Будем хранить знак отдельным битом, а сразу после него – мантиссу и экспоненту. Таким образом можно представить гораздо больший диапазон чисел.

Языки программирования

- Транслируют команды более высокого уровня в процессорные инструкции низкого уровня.
- Бывают интерпретируемые и компилируемые

Языки программирования

- Транслируют команды более высокого уровня в процессорные инструкции низкого уровня.
- Бывают интерпретируемые (Python, Javascript) и компилируемые (C, C++, Java, Go)
- Python – интерпретируемый. Или нет?

Как работает Python

- Программист пишет код на Python (.py файлы)
- … и пытается запустить его (`python3 your_entry_point.py`)
- Python собирает его в *байт-код* (.pyc файлы) – набор инструкций, понятный виртуальной машине.
- *Виртуальная машина* исполняет байт-код, преобразуя его в низкоуровневые инструкции процессора.

А в чем все-таки разница между языками?

- На любом современном/популярном языке программирования можно вычислить все, что позволяют другие языки
- Скорость разработки важнее скорости работы – можно выбрать более высокоуровневый язык
- Нужно разработать прототип – можно выбрать более простой язык, привлечь менее квалифицированных специалистов и быстрее проверить поставленные гипотезы
- Обработку данных можно делать на Python и это будет быстро – numpy написан на C

А в чем все-таки разница между языками?

- Более низкоуровневые ЯП (C/C++) заставляют вас больше думать про то, как данные хранятся в памяти, высокоуровневые (Python) позволяют фокусироваться на решаемой задаче (но при этом низкоуровневые позволяют эффективнее использовать ресурсы)
- В более высокоуровневых языках обычно больше синтаксического сахара и меньше возможностей выстрелить себе в ногу
- Python позволяет программировать, не думая о том, как работает конкретная платформа

Динамическая типизация

- У каждой переменной в Python есть тип
- Тип переменной может меняться со временем (a = 1; a = “abc”)
- Проверка совместимости типов производится при выполнении программы, в отличие от, скажем, C++
- В Python >= 3.5 есть type hints (модуль typing)

Почитать

- https://rushter.com/blog/python-integer-implementation
- https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%87%D0%B8%D1%81%D0%B5%D0%BB_%D1%81_%D0%BF%D0%BB%D0%B0%D0%B2%D0%B0%D1%8E%D1%89%D0%B5%D0%B9_%D1%82%D0%BE%D1%87%D0%BA%D0%BE%D0%B9 – про float point
- https://habr.com/ru/post/112953/ - еще про float point
- https://opensource.com/article/18/4/introduction-python-bytecode
- https://leanpub.com/insidethepythonvirtualmachine/read - в подробностях про виртуальную машину Python