Институт открытых программ дополнительного образования ВШЭ

Python: Инструментальные средства для автоматизации и анализа данных. Часть 2.

Автор курса: Дмитрий Румянцев

Лекция 10

Сложность вычислений и О-нотация

конспект

Москва 2023 г.

Краткий конспект лекции

Под вычислительной сложностью понимают зависимость объема работы, выполняемой алгоритмом, от размера обрабатываемых данных.

Следует делать различие между термином «размер данных» и «объем данных».

Размер – это количество элементов данных. Объем – количество занятых (однобайтовых) ячеек памяти.

Вычислительная (алгоритмическая) сложность есть функция зависимости относительного времени выполнения алгоритма от размера обрабатываемых данных.

Если алгоритм A выполняет обработку данных размером S1 за время T1, а обработку данных такой же структуры, но б***о***льшим размером S2 за время T2, а обработку данных еще большего размера S3 за время T3 и так далее, то в случае если между показателями T1, T2, T3 и так далее может быть найдена какая-то зависимость (линейная, логарифмическая и т.п.), говорят об ***асимптотической сложности*** алгоритма, то есть его зависимости от размера обрабатываемых данных, при стремлении размера входных данных в бесконечность.

Асимптотическая сложность алгоритма описывается т.н. O-нотацией. В O-нотации сложность указывается не в виде временного отрезка, а в количестве условных элементарных операций.

Оценка сложности может быть:

- Наилучшей: минимальная временная оценка;
- Наихудшей: максимальная временная оценка;
- Средней: средняя временная оценка.

Линейная сложность

O(N),

где N – размерность данных.

Линейной сложностью обладают алгоритмы, в которых временн*ы*е затраты на вычисление находятся в прямой линейной зависимости от размера данных.

Сложность O(1):

- присваивание значения переменной
- сложение
- вычитание
- получение элемента последовательности по индексу
- получение длины последовательности и др.

Логарифмическая сложность

O(logN)

Время выполнения алгоритма растет логарифмически с увеличением размера входного массива.

Линейно-логарифмическая сложность

O(NxlogN)

Время выполнения больше чем линейное, но меньше квадратичного.

Например, двоичный поиск N элементов.

Квадратичная сложность: $O(N^2)$

Время выполнения пропорционально квадрату количества элементов в коллекции.

Например, алгоритм сортировки вставками.

Экспоненциальная сложность

$O(2^N)$

Экспоненциальную временную сложность мы можем наблюдать в алгоритмах, в которых количество вычислений удваивается при добавлении каждого нового элемента в набор данных.

Например, вычисление чисел Фибоначчи.

Факториальная временная сложность

O(n!)

Это сложность, при которой количество вычислений в алгоритме прирастает факториально в зависимости от размера набора данных.

Законы сложения и умножения O-нотации для расчетов комбинированной сложности

Закон сложения

Итоговая сложность двух последовательных действий равна сумме их сложностей:

$$O(f(n))+O(g(n))=O(f(n)+g(n))$$

Дополнения к закону сложения

1. Итоговая сложность алгоритма оценивается наихудшим из слагаемых:

$$O(N)+O(\log n)=O(N+\log n)=O(N)$$

2. В итоговой сложности константы отбрасываются

$O(N)+O(N)+O(N)=3\cdot O(N)=O(N)$

3. При ветвлении берется наихудший вариант

```
if condition:
     O(N)
else:
     O(N!)
```

Берется O(n!)

Закон умножения

Итоговая сложность двух вложенных действий равна произведению их сложностей:

$$O(f(n))*O(g(n))=O(f(n)*g(n))$$

Оптимизация

Выявление неэффективных фрагментов кода, т.н. «горячих точек» (*hot spots*) и замена на более эффективный код. Горячие точки – сравнительно небольшие участки кода, на выполнение которых процессор затрачивает наибольшую часть времени.

Сбор статистических данных о времени выполнения фрагментов кода и выявление фрагментов, которые можно оптимизировать, называется ***профилированием***.

Использованные библиотеки и функции

Модуль

`time`

Функция

`time()`

Возвращает число секунд, прошедших с ноля часов, ноля минут и ноля секунд 1 января 1970 года в виде числа с плавающей точкой (т.н. «время Unix»).

`perf_counter()`

Возвращает значение в долях секунды счетчика производительности, т. е. часов с самым высоким доступным разрешением для измерения короткой продолжительности.

Модуль

`random`

Функция

`randint(min,max)`

min – минимально допустимое значение,

max – максимально допустимое значение.

Возвращает случайное целое в заданных пределах.

Модуль

`numpy.random`

Функция

```
randint(low, high=None, size=None, dtype=None)
```

low – минимальное значение случайного числа,

high – максимальное значение случайного числа,

size – размер матрицы,

dtype (необязательный) – желаемый тип результата.

Возвращает матрицу случайных чисел указанного размера.

Модуль

`itertools`

Метод

`permutations(list, range)`

Пример:

`permutations(range(3), 2)`

Возвращает: (0,1), (0,2), (1,0), (1,2), (2,0), (2,1)

Модуль

`requests`

Метод

```
get(url, params=None, **kwargs):
```

url – адрес страницы, которую необходимо загрузить,

params – словарь, список кортежей или байт для отправки в строку запроса,

**kwargs – необязательные аргументы, которые принимает запрос.

Возвращает объект Response, как результат запроса.

Модуль

`cProfile`

Функция

`run(func_call, sort=-1)`

func_call – строка с вызовом функции со всеми необхожимыми параметрами;

sort – сортировка по столбцу таблицы результатов.

Запускает профайлер для указанного вызова функции. Возвращает таблицу таймингов по всем затронутым функциям.

Столбцы таблицы результата работы cProfile.run:

ncalls: количество вызовов данной функции

tottime: общее время, которое работала эта функция без учета времени, потраченного на вызовы внутри тела этой функции.

percall: частное от деления tottime на ncalls

cumtime (cumulative time): совокупное время, потраченное на эту функцию и все внутренние вызовы других функций. Это точное время даже для рекурсивных функций.

percall: частное от деление cumtime на примитивные вызовы.

filename:lineno(function): указывает имя файла, номер строки и название рассматриваемой функции в скобках.