Структуры данных

Данные можно хранить по-разному

- list
- tuple
- dict
- pd.DataFrame

Структура данных

- позволяет хранить связанные по смыслу данные
- позволяет модифицировать эти данные: добавлять, искать, изменять, удалять
- позволяет эффективно решать определённый класс задач

Массив

- Структура данных, которая хранит фиксированное количество последовательно хранящихся значений.
- За счет этого обеспечивается быстрый доступ к элементу по его индексу

Массив

- Структура данных, которая хранит фиксированное количество последовательно хранящихся значений.
- За счет этого обеспечивается быстрый доступ к элементу по его индексу
- Если start – адрес начала массива в памяти, то i-й элемент находится по адресу start + i * sizeof(array[0]), где sizeof(array[0]) – количество памяти, которое занимает один массив.
- Доступ к произвольному элементу всегда за O(1). Массив нельзя увеличить, но можно изменять значения элементов за O(1).

А если мы не знаем заранее, сколько элементов нам понадобится хранить?

А если мы не знаем заранее, сколько элементов нам понадобится хранить?

- Можно расширять массив по ходу добавления данных – динамический массив (вектор)
- При создании нового вектора создадим пустой массив
- При добавлении нового элемента будем переписывать массив, увеличивая его размер на 1

А если мы не знаем заранее, сколько элементов нам понадобится хранить?

- Можно расширять массив по ходу добавления данных – динамический массив (вектор)
- При создании нового вектора создадим пустой массив
- При добавлении нового элемента будем переписывать массив, увеличивая его размер на 1. Получим сложность добавления элемента по времени O(n), где n – текущий размер массива. **Можно ли быстрее?**

А если мы не знаем заранее, сколько элементов нам понадобится хранить?

- Можно расширять массив по ходу добавления данных – динамический массив (вектор)
- При создании нового вектора создадим пустой массив
- При добавлении нового элемента будем переписывать массив, увеличивая его размер **в 2 раза** (или в некоторое другое постоянное число раз). Получаем сложность добавления элемента по времени O(n), если место во «внутреннем» массиве закончилось, и O(1), если оно ещё есть

Какая на самом деле сложность вставки?

- Большинство вставок лёгкие. Трудоемкие вставки происходят все реже с ростом длины массива, но и сложность «трудной» вставки растёт (линейно)

Какая на самом деле сложность вставки?

- Большинство вставок лёгкие. Трудоемкие вставки происходят все реже с ростом длины массива, но и сложность «трудной» вставки растёт (линейно)
- Будем считать среднюю (*амортизационную*) сложность вставки.

Амортизационный анализ вставки в динамический массив

- Пусть в пустой вектор выполнено N вставок и N – степень двойки (для удобства). Трудоемкие вставки происходят $log_2 N$ раз и занимают время, пропорциональное текущему размеру списка, все остальные – лёгкие и занимают константное время. Тогда средняя стоимость одной вставки:

$$\frac{\sum_{i=1}^{log_2 N} O(1)\times 2^i + (N - log_2 N) \times O(1)}{N} =$$

$$= O(1)\times \frac{2^{log_2 N+1} - 2}{N} + O(1) = O(1)\times \frac{2N - 2}{N} = O(1)$$

Итого

- list в Python – это динамический массив
- В Java ArrayList, в C++ std::vector
- Можно реализовать дополнительные операции: pop, shrink_to_fit