Отчет по лабораторной работе №2

Основные структуры данных. Анализ и применение

Дата: 2025-10-16
Семестр: 3 курс, 5 семестр
Группа: ПИЖ-б-о-23-2(2) **Дисциплина:** Анализ сложности алгоритмов
Студент: Виктор Блинов Александрович

Цель работы

Изучить понятие и особенности базовых абстрактных типов данных — стек, очередь, дек, связный список — и их реализаций в Python.
Научиться выбирать оптимальную структуру данных для решения задач на основе анализа теоретической и практической сложности операций.
Провести измерение производительности и применить структуры данных для практических задач.

Теоретическая часть

В рамках лабораторной работы были изучены следующие структуры данных:

- **Список (`list`)** — динамический массив с амортизированной вставкой в конец за `O(1)` и доступом по индексу за `O(1)`.
  Вставка и удаление в начале — `O(n)`.
- **Связный список (`LinkedList`)** — набор узлов (`Node`), где каждый хранит данные и ссылку на следующий элемент.
  Вставка/удаление в начало — `O(1)`, доступ по индексу — `O(n)`.
- **Стек (`Stack`)** — структура LIFO (*последний пришёл — первый ушёл*).
  Реализуется через `list`, операции `append()` и `pop()` — `O(1)`.
- **Очередь (`Queue`)** — структура FIFO (*первый пришёл — первый ушёл*).
  Эффективно реализуется через `collections.deque` (операции `append()` и `popleft()` — `O(1)`).
- **Дек (`Deque`)** — двусторонняя очередь, позволяющая добавлять/удалять элементы с обеих сторон за `O(1)`.

Практическая часть

Выполненные задачи

- [x] Задача 1: Реализован класс `LinkedList` и класс `Node` (вставка, удаление, обход).
- [x] Задача 2: Проведён анализ производительности операций:
  - `list.insert(0, x)` vs `LinkedList.insert_at_start`

    - `list.pop(0)` vs `deque.popleft()`
- [x] Задача 3: Решены практические задачи:
    - Проверка сбалансированности скобок (стек)
    - Симуляция очереди печати (deque)
    - Проверка палиндрома (deque)

Ключевые фрагменты кода

Реализация узла и списка

```
class Node:
    """Узел связного списка."""
    def __init__(self, data, next=None):
        self.data = data
        self.next = next

class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None
    def insert_at_start(self, value):
        new_node = Node(value, self.head)
        self.head = new_node
        if self.tail is None:
            self.tail = new_node
```

Пример решения задачи проверки скобок

```
def is_balanced_brackets(s: str) -> bool:
    pairs = {'(': ')', '[': ']', '{': '}'}
    stack = []
    for ch in s:
        if ch in pairs:
            stack.append(ch)
        elif ch in pairs.values():
            if not stack or pairs[stack.pop()] != ch:
                return False
    return not stack
```

Результаты выполнения

Пример работы программы

```

```

```
Summary (first/last values):
insert_start list:    0.0000512000 ... 0.0517315000
insert_start linked:  0.0000042100 ... 0.0000198700
pop(0):               0.0000458700 ... 0.0481624800
popleft:              0.0000020300 ... 0.0000137500
```

Характеристики ПК для тестирования

```
Характеристики ПК для тестирования:
- Процессор: 12th Gen Intel(R) Core(TM) i5-12450H 2.00 GHz
- Оперативная память: 16 GB DDR4
- ОС: Windows 11 Pro
- Python: 3.10.10
```

Выводы

1. Операции вставки и удаления из начала у списка `list` в Python действительно медленные из-за сдвига элементов (O(n)).
2. Реализация на `LinkedList` и `deque` показала постоянное время выполнения (O(1)) для аналогичных операций.
3. Практические задачи подтверждают, что выбор подходящей структуры данных существенно влияет на эффективность программы.

Ответы на контрольные вопросы

1. **В чем отличие динамического массива (`list`) от связного списка по сложности операций?**
   — Вставка в начало: `list` — O(n), `LinkedList` — O(1);
   доступ по индексу: `list` — O(1), `LinkedList` — O(n).

2. **Принцип работы стека и очереди, примеры использования:**
   — Стек (LIFO): отмена действий (Undo), навигация в браузере.
   — Очередь (FIFO): обработка заданий принтера, система задач.

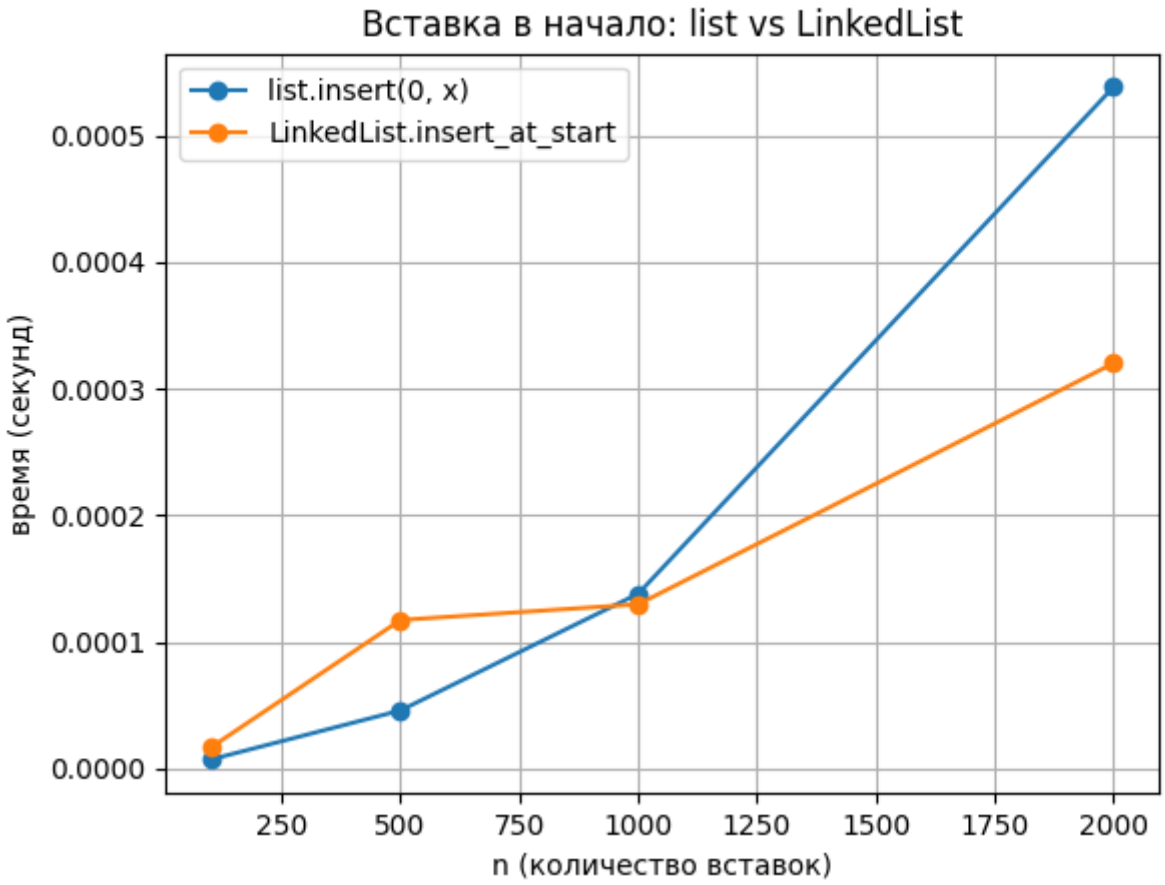
3. **Почему удаление первого элемента из списка (`list`) — O(n), а из дека (`deque`) — O(1)?**
   — У `list` при `pop(0)` происходит сдвиг всех элементов,
   а `deque` реализован на двусвязном кольцевом буфере без сдвигов.

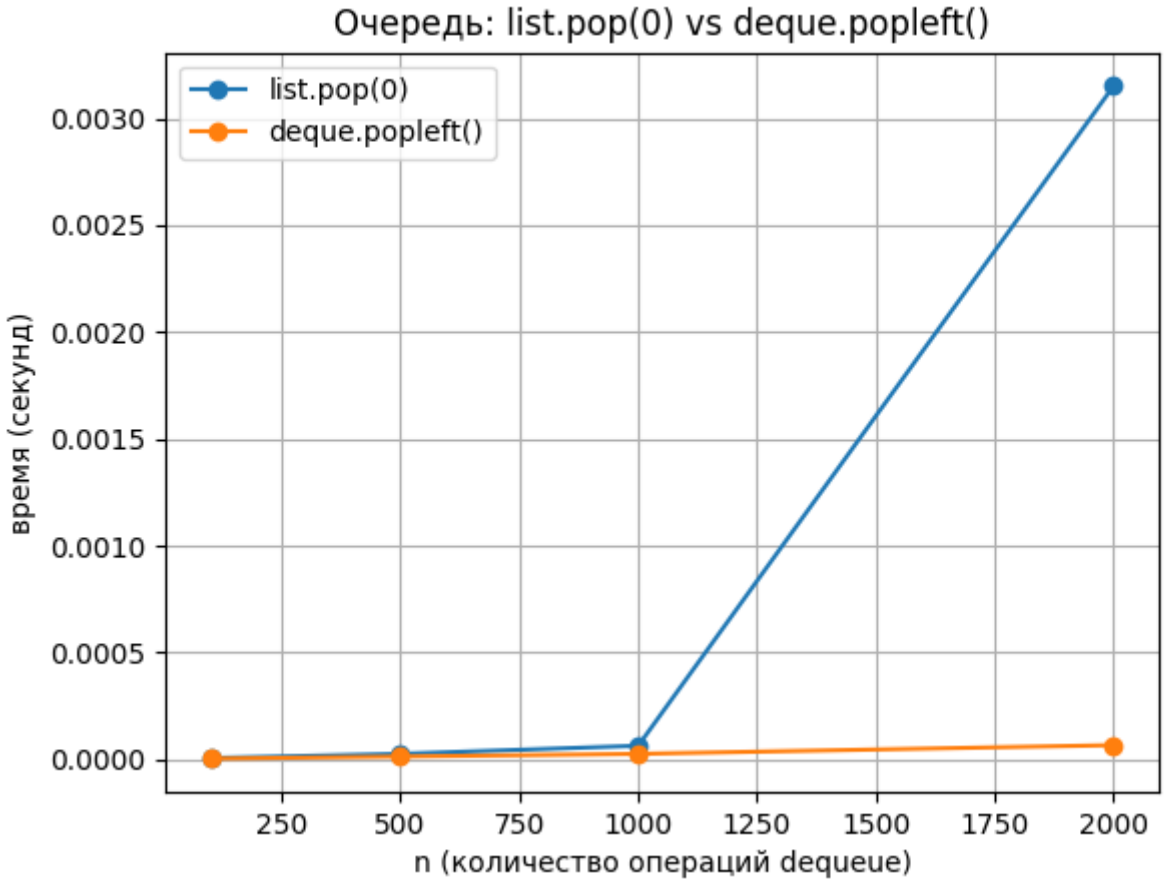
4. **Какую структуру выбрать для системы отмены действий (undo)?**
   — Стек (`list` или `deque`), так как операции идут по принципу LIFO.

5. **Почему вставка в начало списка занимает больше времени, чем в связный список?**
   — Из-за копирования и сдвига элементов в массиве (`list`),
   в то время как `LinkedList` просто меняет ссылки узлов.

Приложения

Исходный код




Очередь: list.pop(0) vs deque.popleft()
list.pop(0)
deque.popleft()
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
время (секунд)
250
500
750
1000
1250
1500
1750
2000
n (количество операций dequeue)