GO-13 03: Самописный LRU cache

Описание:

В прошлых заданиях модуля мы рассмотрели способы кеширования данных наших приложений, в этот раз давайте представим, что нас устраивает вариант хранить данные im-memory в сервисе, но нам нужно решить проблемы с размером кеша и ресурсами соответственно. В контексте этой проблемы мы изучим такой алгоритм кеширования, как LRU(Least Recently Used).
Суть алгоритма заключается в том, что задается фиксированный размер кеша (количество элементов в нем - N), и если при добавлении мы превышаем этот предел N, то из кеша вытесняется самый давно используемый элемент, то есть, по сути, мы должны как-то хранить актуальную хронологию запросов к элементам кеша, чтобы всегда иметь возможность вытеснить наименее актуальный.
Этот алгоритм используется, к примеру, в реализации страничного кеша в недрах ядра Linux. А также это дико частый вопрос на собеседованиях, а-ля - напишите свою реализацию))
В этом задании вам нужно будет написать потокобезопасный модуль, который будет реализовывать интерфейс lru кеша.

Полезные ссылки:

- LRU, метод вытеснения из кэша

Задание:

1. Создайте проект с произвольным названием. Данный проект вы можете добавить в свое портфолио, поэтому можете создать проект как на GitHub, так и во внутреннем GitLab.
2. В созданном проекте необходимо написать реализацию интерфейса:

```
type LRUCache interface {
    // Добавляет новое значение с ключом в кеш (с наивысшим приоритетом), возвращает true, если все прошло успешно
    // В случае дублирования ключа вернуть false
    // В случае превышения размера - вытесняется наименее приоритетный элемент
    Add(key, value string) bool

    // Возвращает значение под ключом и флаг его наличия в кеше
    // В случае наличия в кеше элемента повышает его приоритет
```

```
    Get(key string) (value string, ok bool)

    // Удаляет элемент из кеша, в случае успеха возвращает true, в
случае отсутствия элемента - false
    Remove(key string) (ok bool)
```

И конструктор для вашего кеша вида:
```
NewLRUCache(n int) LRUCache
```

Где n - количество элементов.

3. Получение элемента должно происходить за время O(1) (т.е. не зависеть от размера кеша) - соответственно в качестве основного хранилища должна использоваться Hash Map.
4. Для решения вопроса приоритезации смотрите в сторону таких структур данных, как очереди и связные списки.

- Задание со звездочкой: добиться работы со структурой данных, которую вы использовали для приоритезации, также за константное время.

По желанию, можете покрыть тестами)

5. В ответе укажите ссылку на ваш репозиторий.