TestVagrant

Coding Assignment

OVERVIEW

Create an in-memory store for recently played songs that can accommodate N songs per user, with a fixed initial capacity. This store must have the capability to store a list of song-user pairs, with each song linked to a user. It should also be able to fetch recently played songs based on the user and eliminate the least recently played songs when the store becomes full.

EXAMPLE

Illustration, when 4 different songs were played by a user & Initial capacity is 3:

Let's assume that the user has played 3 songs - S1, S2 and S3.

- The playlist would look like -> S1,S2,S3
- When S4 song is played -> S2,S3,**S4**
- When S2 song is played -> S3,S4,**S2**
- When S1 song is played -> S4,S2,**S1**

EXPECTATIONS

- Adhere to clean coding standards and principles. OOP is recommended.
- Write tests to test the logic

Approach

For solving above question, I have used Least Recently Used (LRU) Cache concept with JAVA as programming language, TestNG for the execution of the Testcase and to build the project I have used Maven project.

LRU cache
Organizes items in order of use, allowing you to quickly identify which item hasn't been used for the longest amount of time.

Cache
an auxiliary memory from which high-speed retrieval is possible.
(for the question assuming cache has a capacity of 3)

-To implement LRU concept I have used two data structures HashMap and Doubly-LinkedList.

1. A HashMap with No of Users as key and the songs played by users of the corresponding node as value.

2. Doubly-linked list- The maximum size of the queue will be equal to the cache size. The most recently songs played is near to the front and the least recently song played will be near to the rear end.

Operation to be performed

Pre-Requisite – I have created a Two pseudo node named as Head & Tail (Doubly-LinkedList) in front and last position.

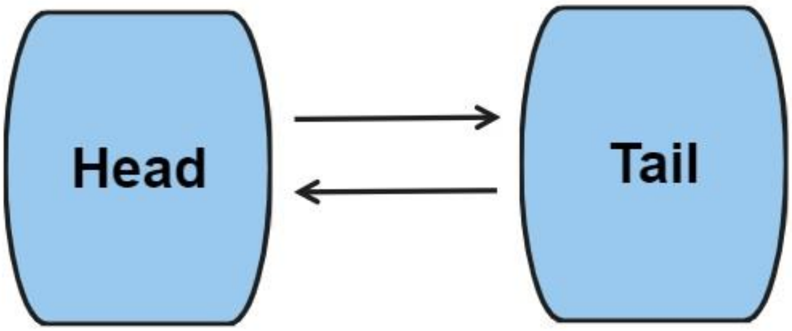


Fig: - 1

1. addNode – In the addNode method I have write the logic how the value is added in the node.

- When first value is added

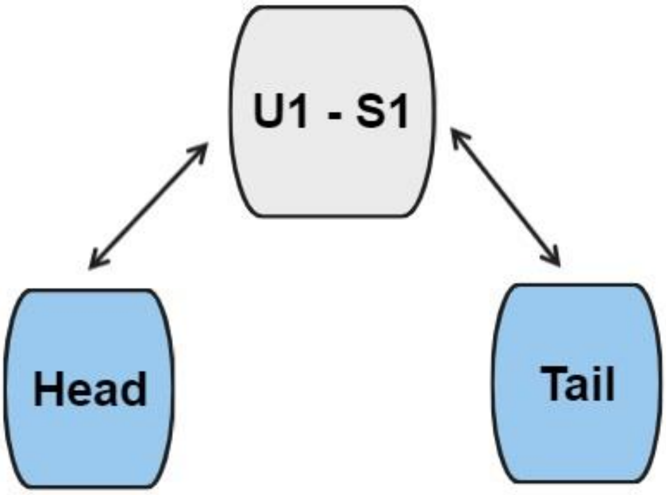


Fig: - 2

- When second value is added – it will store next to the Head

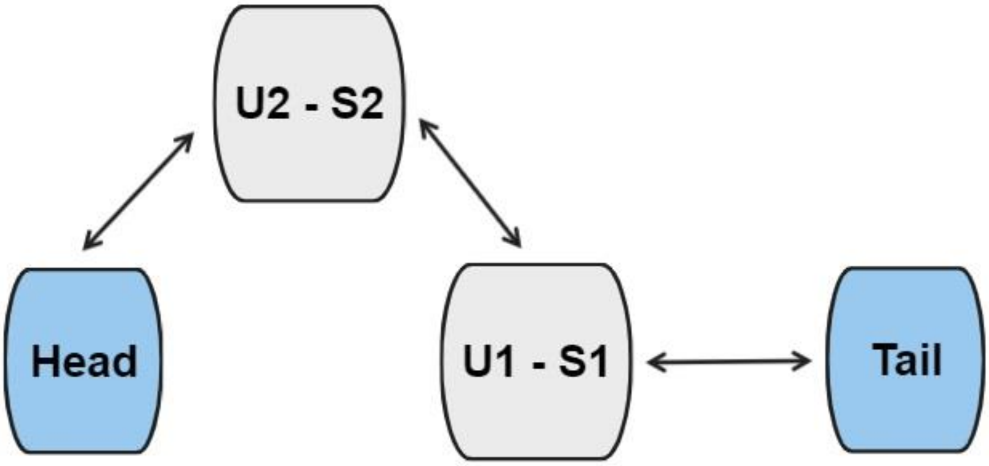


Fig: - 3

- When third value is added – it will store next to the Head

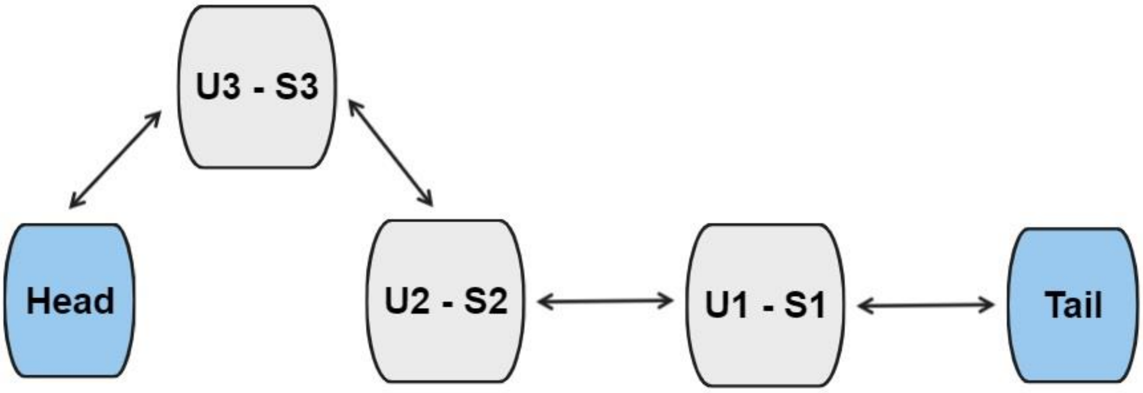


Fig: - 4

- Final structure – and the capacity to store value in cache is full

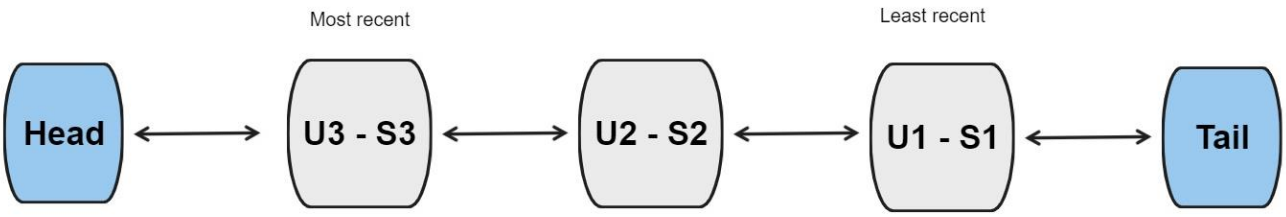


Fig: - 5

2. removeNode- In the removeNode method I have write the logic how the value is removed from the node when cache is full.

Condition – When the new value is come to store in the memory and at the same time it has to maintain the capacity so it will first remove the least recent value (U1-S1) then after that it will store then new value.

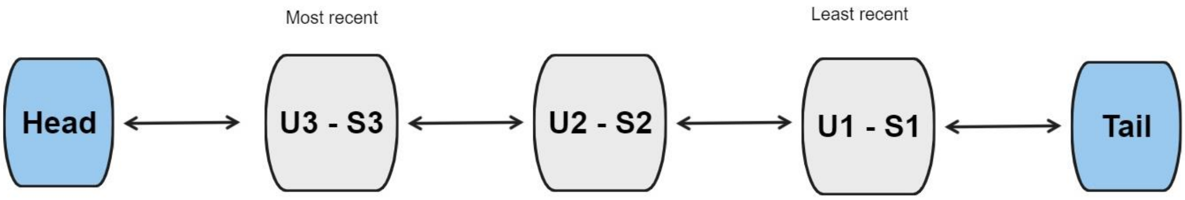


Fig: - 6

- Now after removing it will look like

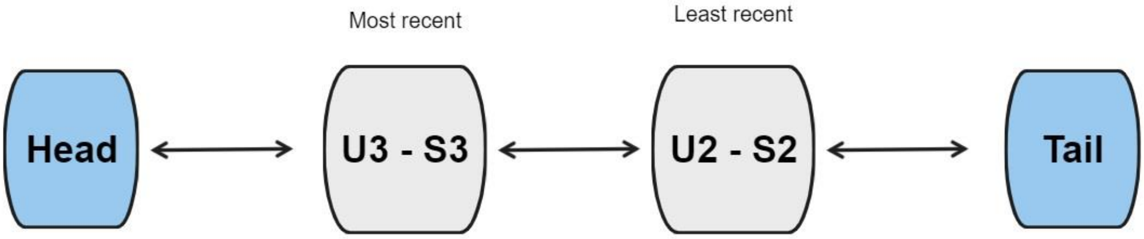


Fig: - 7

3. moveToNode- In this method I have performed above two methods continuously.

condition - Whenever the existing value is called then have to make it as most recent in the play-List by first removing its value then again add to it.

- When cache is full

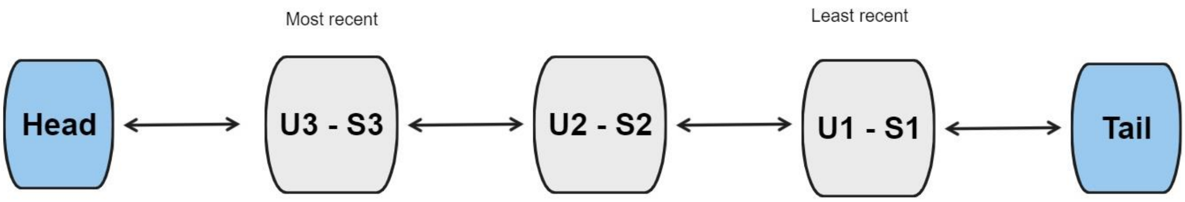


Fig: - 8

- when U1-S1 is called so from the queue it will remove U1-S1

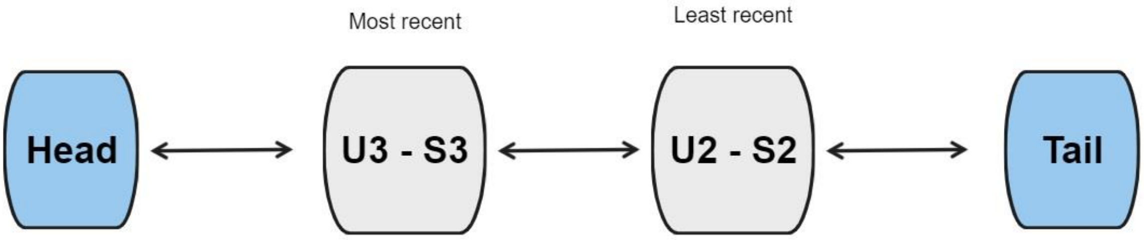


Fig: - 9

- And add it to the front and become most recently

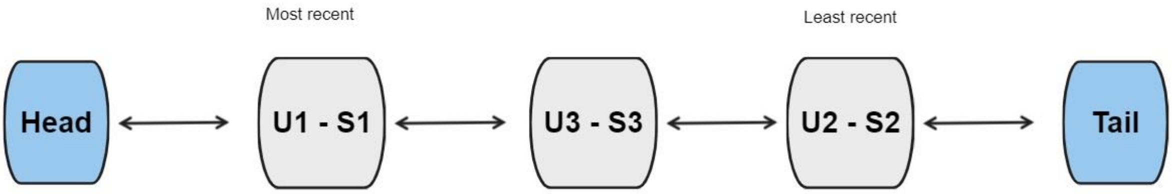
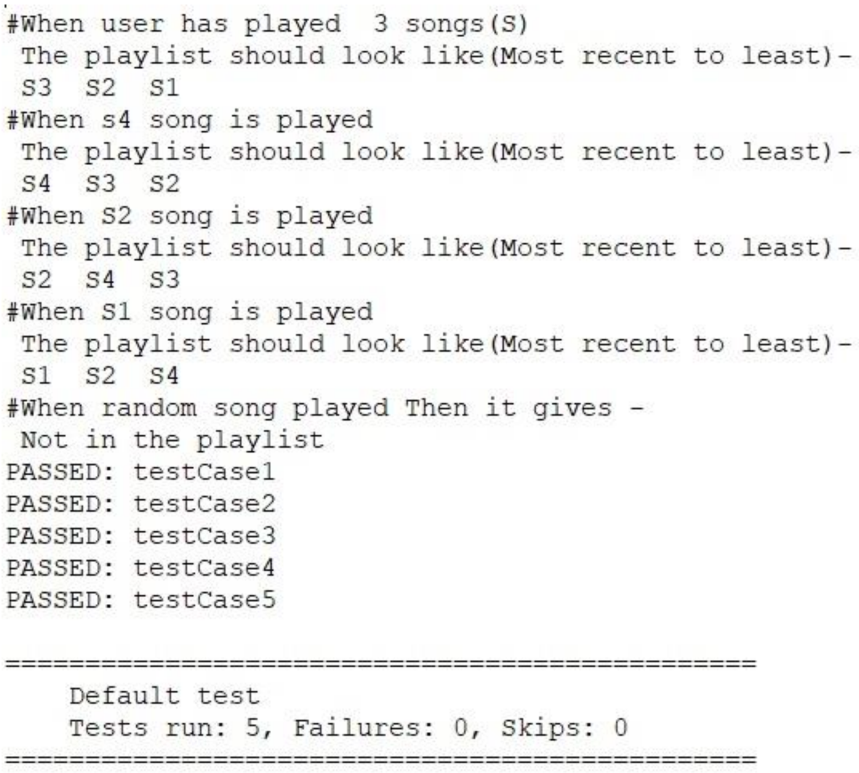


Fig: - 10

<u>Output</u>: -



```
#When user has played  3 songs(S)
 The playlist should look like(Most recent to least)-
 S3  S2  S1
#When s4 song is played
 The playlist should look like(Most recent to least)-
 S4  S3  S2
#When S2 song is played
 The playlist should look like(Most recent to least)-
 S2  S4  S3
#When S1 song is played
 The playlist should look like(Most recent to least)-
 S1  S2  S4
#When random song played Then it gives -
 Not in the playlist
PASSED: testCase1
PASSED: testCase2
PASSED: testCase3
PASSED: testCase4
PASSED: testCase5

===============================================
    Default test
    Tests run: 5, Failures: 0, Skips: 0
===============================================
```

Fig: - 11