Challenge #2

Problem:
Imagine your company has created a service which can translate small snippets of text (say, up to 140 characters long) from one language to another (e.g., from English to Danish). The translation is computationally expensive, and your backend systems are having difficulty keeping up with the growing popularity of the service.

To reduce load on the backend systems, you are tasked with designing a cache to store commonly-translated snippets. Assume it is not an option to use existing caching systems.

Describe how your caching system would handle the following operations (for example, what algorithms/data structures would you use, and why):

- a) Locate the translation of a given snippet, if it is present in the cache
- b) Determine which snippet should be removed from the cache to make room for new translations. This should be the snippet which has been in the cache for the longest time without having had a 'hit'

Solution:
In the case described above I would use LRU (least recently used) design, which evict least recently used entry. The way it would work is as follows:

- If a snippet exists in the cache, we return it quickly.
- If it doesn't exist and the cache has extra storage slots, we fetch the snippet from the database and return it to the client. In addition we insert the recently fetched snippet in the cache.
- If the cache is full, we delete the snippet that hasn’t been ‘hit’ for the longest time and insert the recently fetched one.

The required operations would be: fast lookup, insert and delete. To achieve fast lookup we need to use Hashtable or HashMap. For insert and delete in O(1) time we would use something like queue, stack or sorted array. In our case we can use queue implemented by a Doubly Linked List.
In building an LRU cache these two data structures actually work coherently to achieve the design. HashMap holds the keys and values as reference of the nodes of Doubly Linked List. Doubly Linked List is used to store list of nodes (snippets) with most recently used node at the head of the list. When more nodes are added to the list, least recently used ones are moved to the end of the list with node at tail being

the least recently used. This way it’s easy to determine which node in the cache was hit last, it’s easy to remove and insert new ones and are easy to lookup.
This company have a clear problem and needs a clear solution. In other cases I would suggest W-TinyLFU eviction policy. Which removes the problem with nodes used frequently in the past and still saved in the cache. W-TinyLFU calculates frequency within a time window. However it may not be fitting for a translation service like this one.