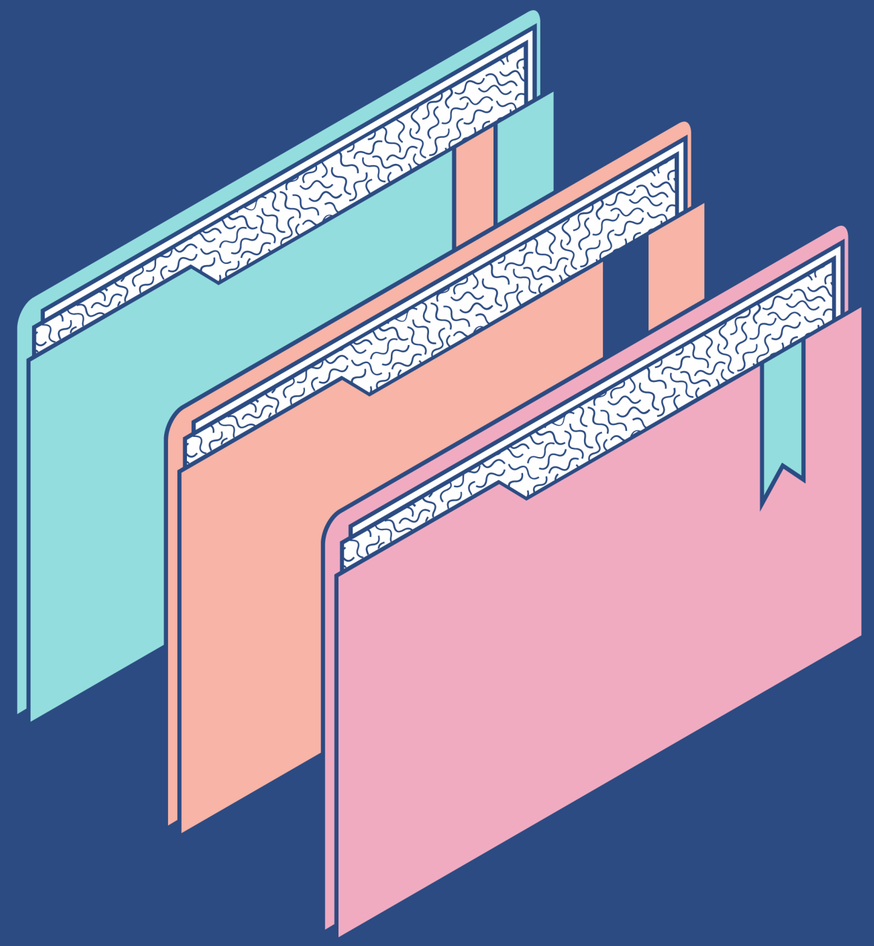
DATA STRUCTURE AND ALGORITHM

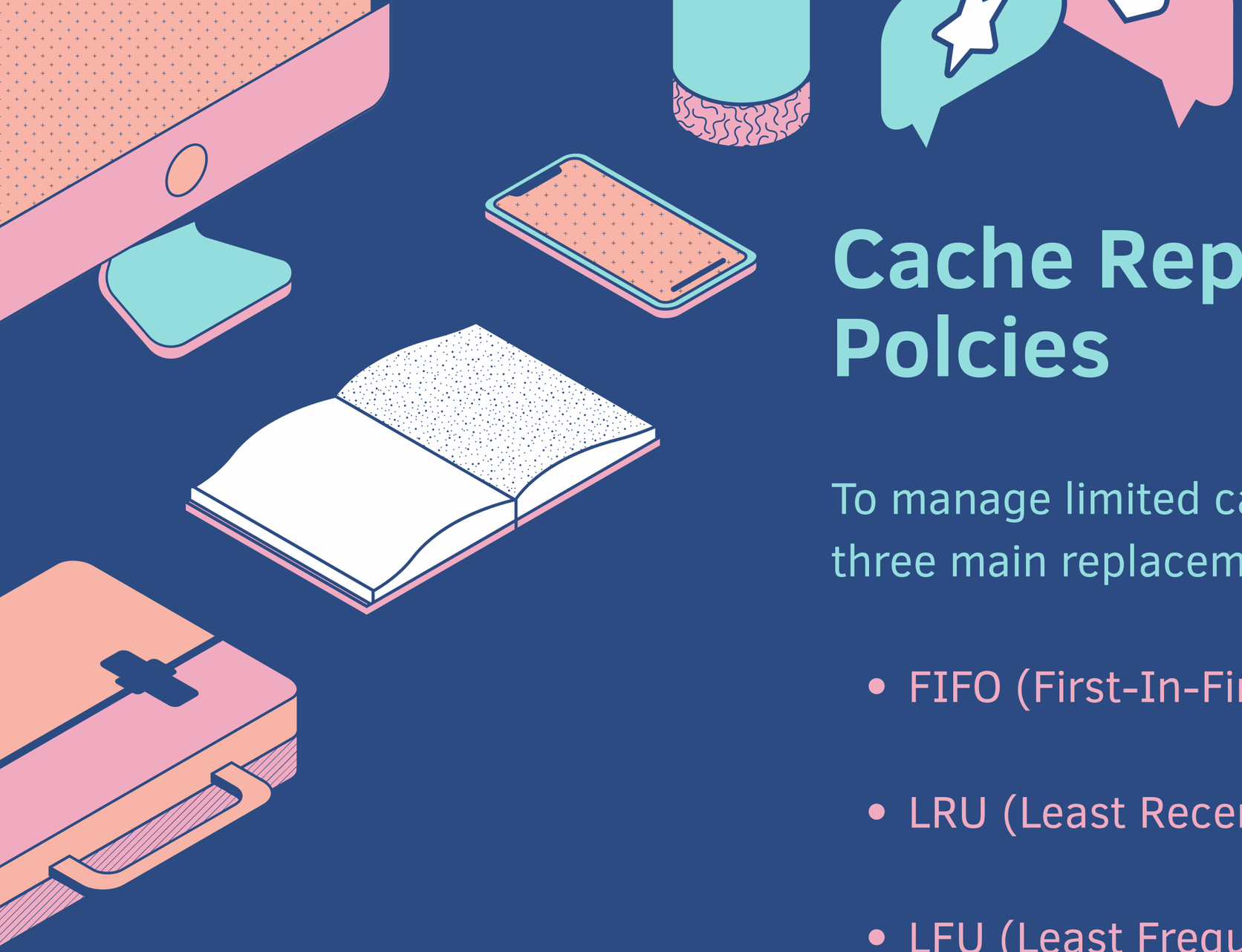
Cache Simulation APP

Simulating FIFO, LRU, and LFU Caching Strategies



What is a Cache?

- WHEN A PROCESSOR EXECUTES A PROGRAM, IT FREQUENTLY ACCESSES DATA FROM MAIN MEMORY.
- HOWEVER, MAIN MEMORY IS SLOW.
- TO SPEED THINGS UP, WE USE A CACHE — A SMALL, FAST MEMORY LOCATED CLOSER TO THE CPU.



Cache Replacement Polcies

To manage limited capacity, the cache follows three main replacement policies:

- FIFO (First-In-First-Out)
- LRU (Least Recently Used)
- LFU (Least Frequently Used)

FIFO – First-In-First-Out

- The oldest inserted item is evicted first.
- Items are removed in the order they were added.

Implementation

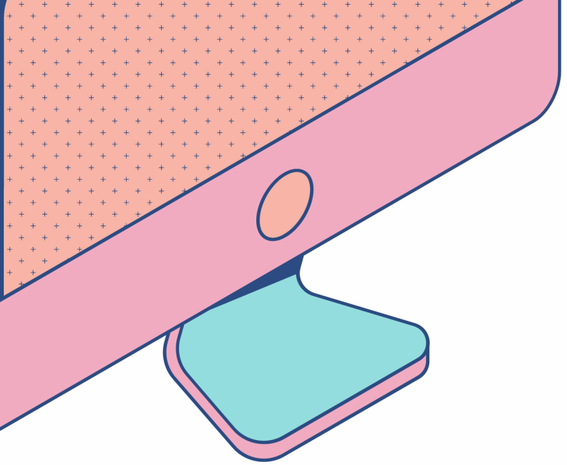
- Implemented using a queue data structure.

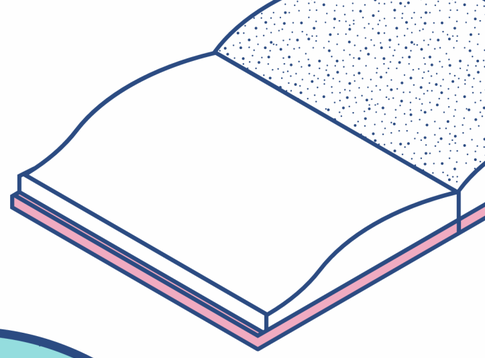
Advantages

- Simple and easy to implement.
- Efficient in scenarios where old data becomes less useful quickly.

Disadvantages

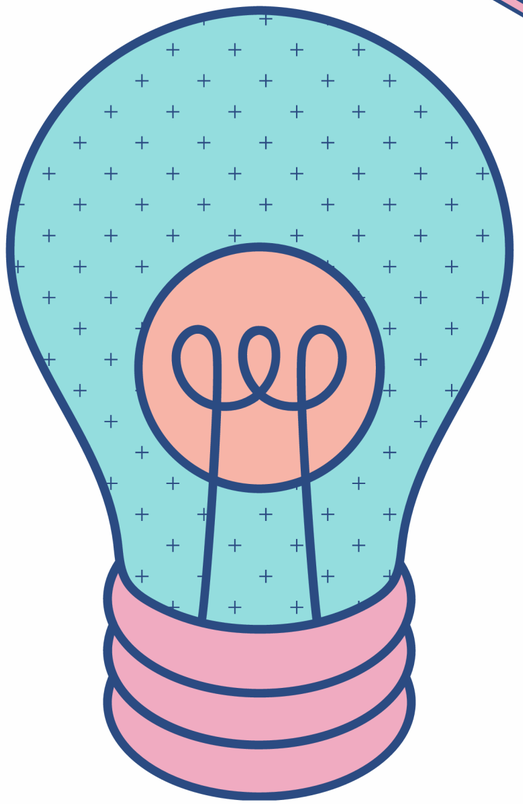
- Doesn’t consider whether data was frequently or recently used.
- May evict useful data prematurely.

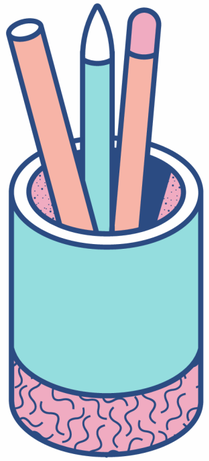


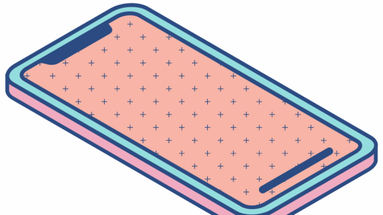


Problem with FIFO

- FIFO removes the oldest item in the cache, based only on arrival order.
- It doesn't consider how often or how recently the data is used.
- This can lead to frequently accessed and relevant data being removed, simply because it arrived earlier.

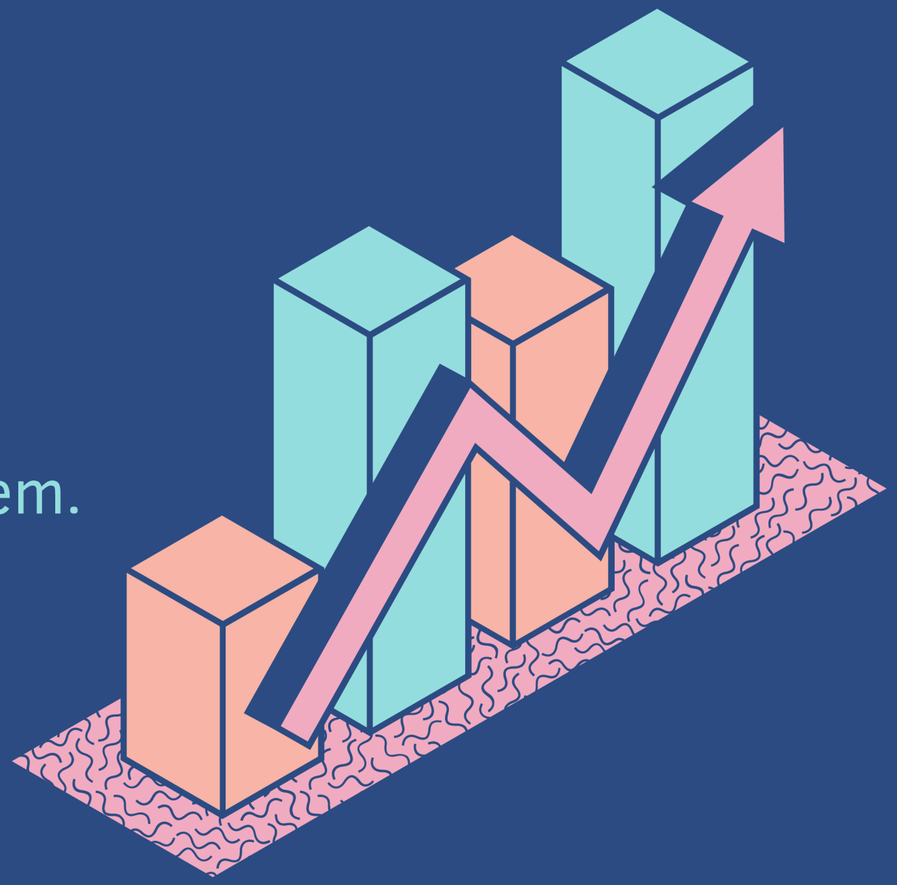






LRU – Least Recently Used

- Removes the least recently accessed item.
- Prioritizes keeping recently used data.



LRU – Data Arrangement

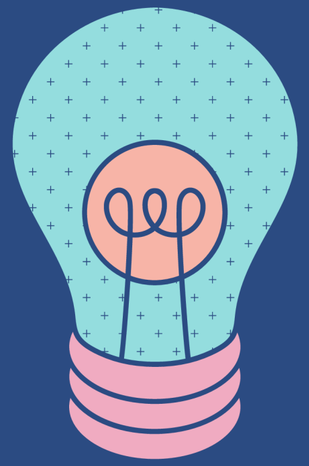
❓ ❓ ❓ ❓ ❓

MOST RECENTLY USED COMES FIRST

!!Every new insertion must obey this law

LEAST RECENTLY USED COMES LAST

NOTE : CACHE IS EMPTY



Inserting data from 001 to 005

LRU – Data Arrangement

001 ? ? ? ?

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

- NOTE : 001 HAS BEEN INSERTED

LRU – Data Arrangement

002 001 ? ? ?

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

- NOTE : 002 HAS BEEN INSERTED. AND IT IS NOT THE MOST RECENTLY USED DATA.

LRU – Data Arrangement

003 002 001 ? ?

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

- NOTE : 003 HAS BEEN INSERTED

LRU – Data Arrangement

004 003 002 001

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

- NOTE : 004 HAS BEEN INSERTED

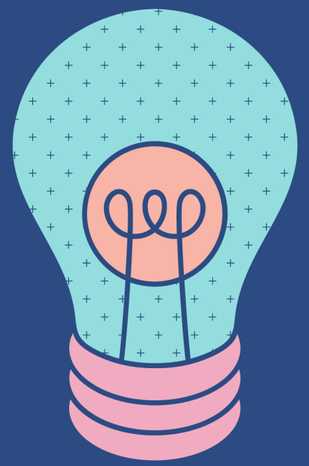
LRU – Data Arrangement

005 004 003 002 001

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

- NOTE : 005 HAS BEEN INSERTED
- CACHE IS FULL



Accessing data in the cache will cause the arrangement to change.

Example: let’s access data 003

LRU – Data Arrangement

005 004 003 002 001

MOST RECENTLY USED COMES FIRST

- NOTE : 003 HAS BEEN ACCESSED

LEAST RECENTLY USED COMES LAST

LRU – Data Arrangement

005 004 ~~003~~ 002 001

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

- NOTE : REMOVE 003 FROM ITS CURRENT LOCATION

LRU – Data Arrangement

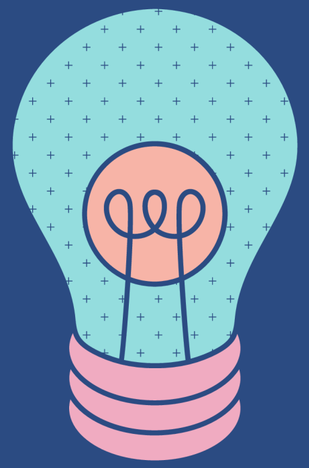
003 005 004 002 001

SHIFTED >>>

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

- NOTE : 003 IS THE CURRENT MOST RECENTLY USED DATA



In order to accommodate for additional data the least recently used data is removed

Example: inserting data 006

LRU – Data Arrangement

003 005 004 002 001

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

LRU – Data Arrangement

003 005 004 002 001

MOST RECENTLY USED COMES FIRST

LEAST RECENTLY USED COMES LAST

LRU – Data Arrangement

006 003 005 004 002

MOST RECENTLY USED COMES FIRST

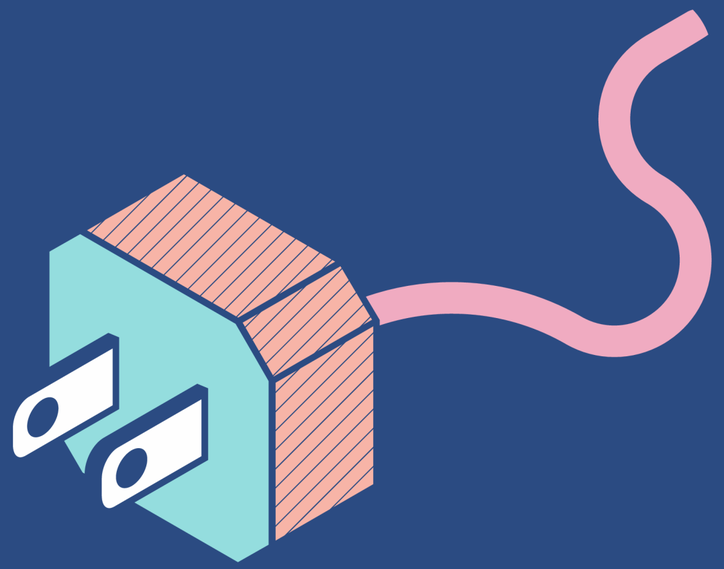
SHIFTED >>>

LEAST RECENTLY USED COMES LAST

- NOTE : NEW DATA HAS BEEN INSERTED

❓ List or Queue

- queue or array cannot efficiently rearrange elements after access.
- Removal or reordering in arrays/queues is O(n) → bad for performance.



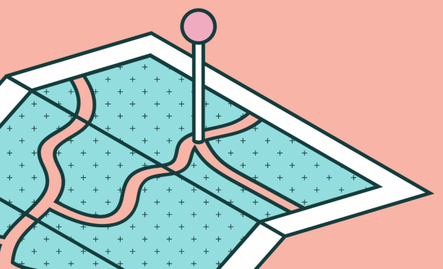
LRU – Solution: Linked List + HashMap

LINKED LIST

- Inbetween removal and addition in O(1)

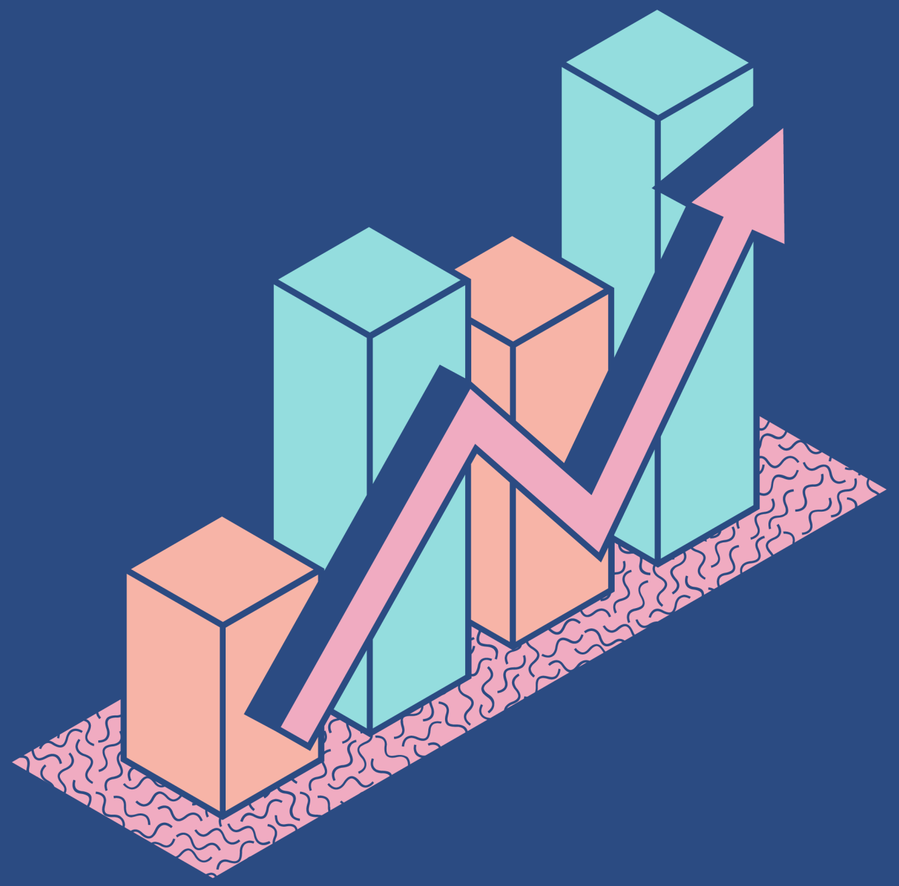
HASHMAP

- Provides O(1) access to nodes in the list.
- Maps keys to their corresponding node in the linked list.



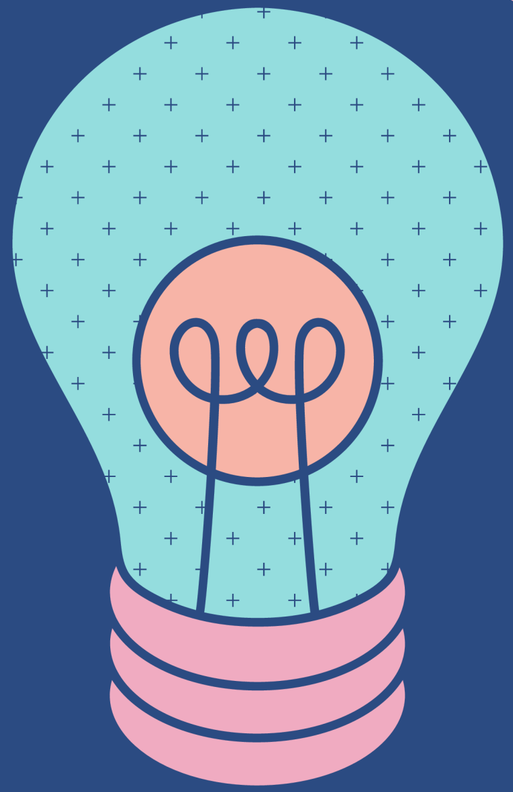
LFU – Least Frequently Used

- Removes the item that is least frequently used.
- If multiple items have the same frequency, evict the least recently used among them.

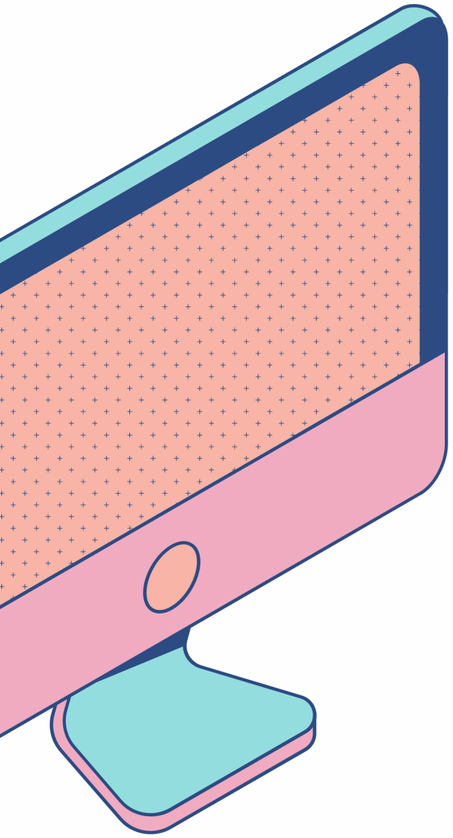


WHY LFU?

SOME DATA MIGHT BE ACCESSED REPEATEDLY BUT NOT RECENTLY. LFU CAPTURES THE "POPULARITY" OF DATA RATHER THAN RECENCY.

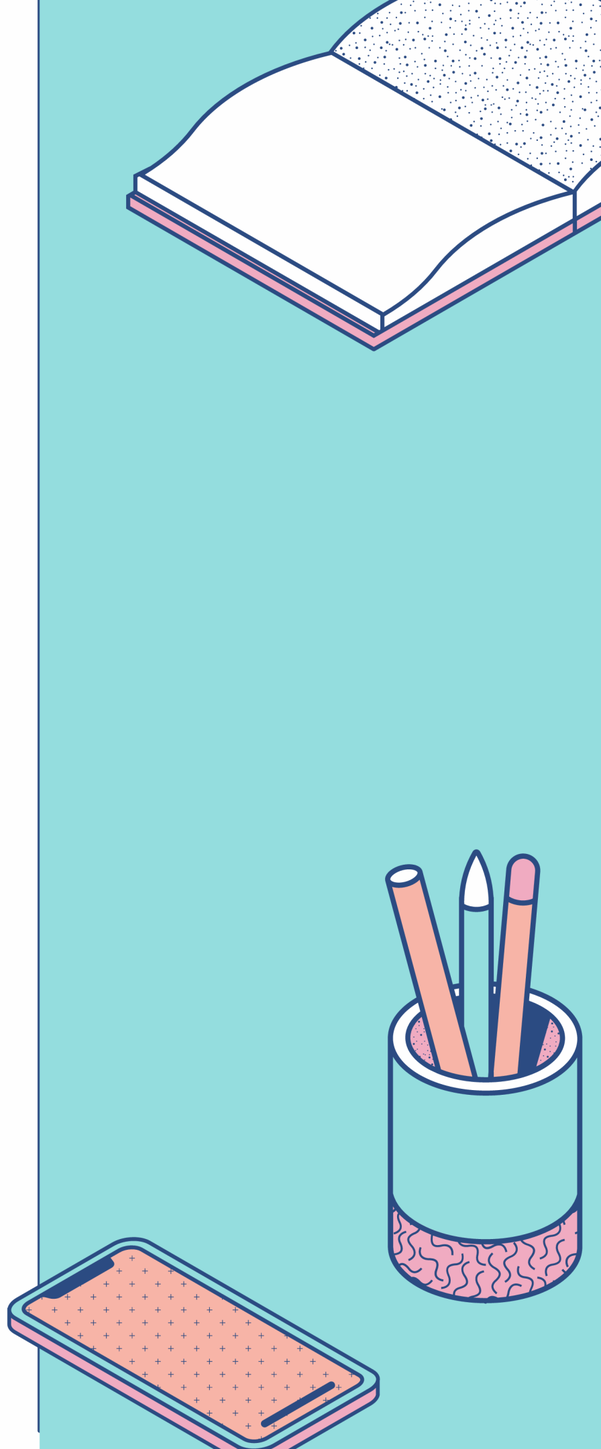


HOW TO TRACK LFU?



MULTIPLE LINKED LIST

- Each linked list represent collection of data accessed the same amount of times.
- Each Elements in the linked list are place in order of access.

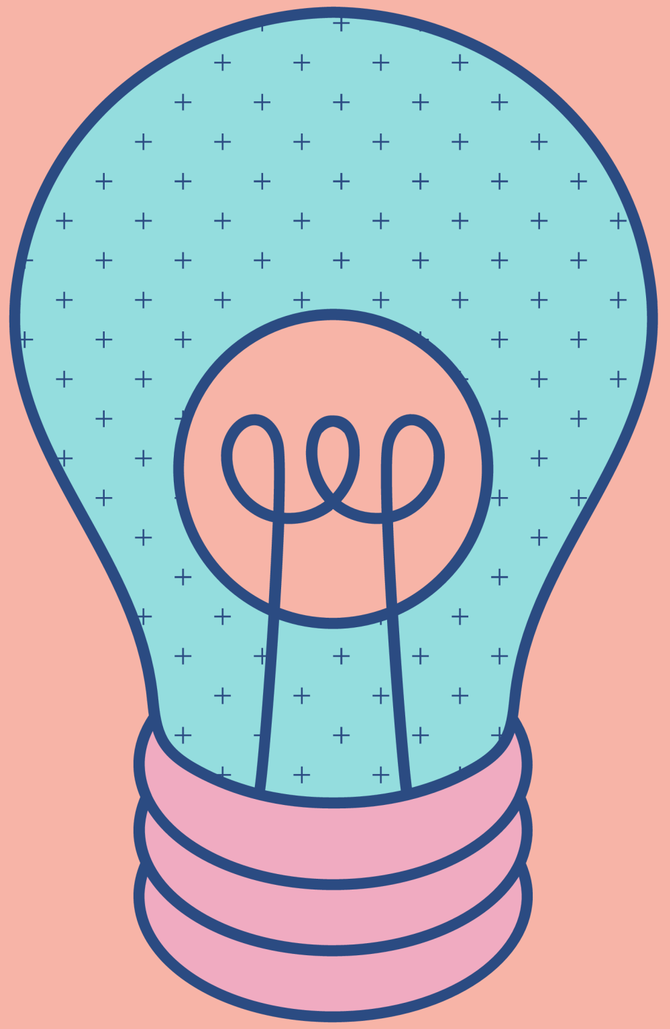


Data Structure

Freq 1: 001 → 002 → 003 ← LFU

Freq 2: 004

- Each frequency count (like Freq 1, Freq 2) maps to a list of keys.
- Inside Freq 1, keys 001, 002, and 003 are ordered by recency.
- LFU eviction would remove the key with the lowest frequency, and if there's a tie (like among keys in Freq 1), it removes the leftmost — which is the least recently used.



LFU – Data Structure Choice

Freq1 : 001 → 002 → 003

Freq2 : 004 → 005

Freq3 : 006

LFU

LFU – Data Structure Choice

Freq1 : 001 → 002 → 003

Freq2 : 004 → 005

Freq3 : 006

LFU

LFU – Data Structure Choice

Freq1 : 001 → 002 → 003

Freq2 : 004 → 005

Freq3 : 006

!!REMOVING HAPPEND IN THE LFU LINKED LIST

LFU – Data Structure Choice

Freq1 : ~~001~~ → 002 → 003

Freq2 : 004 → 005

Freq3 : 006

!!REMOVING HAPPEND IN THE LFU LINKED LIST

LFU – Data Structure Choice

Freq1 : 002 → 003 → 007

<<< SHIFTED

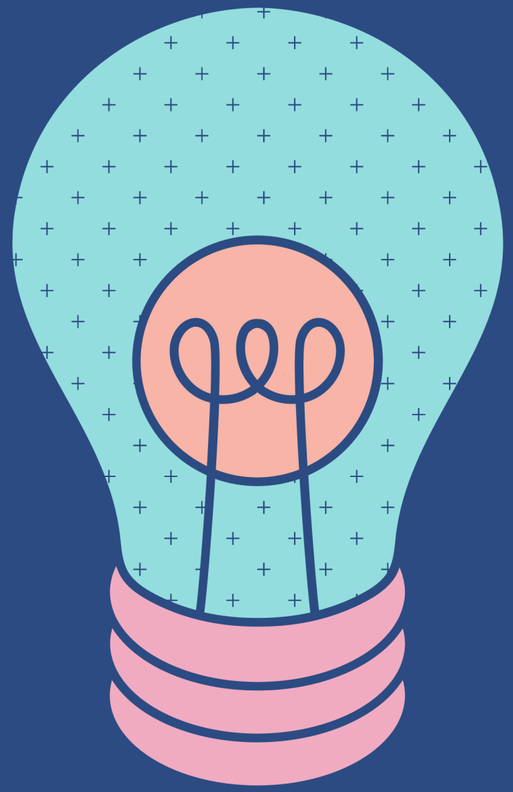
Freq2 : 004 → 005

Freq3 : 006

!!REMOVING HAPPEND IN THE LFU LINKED LIST

Implemention

- EACH NODE WILL HAVE ADDITIONAL ATTRIBUTE CALLED **FREQUENCE**.
- EACH **LINKED LIST** IS MAPPED TO ITS FREQUENCY IN HASHMAP
- WE KEEP TRACK OF THE **MINFREQUENCE**, TO KNOW WHERE TO START REMOVING
- A HASHMAP THAT HOLDS ALL THE **NODE REFERENCE**



Summary of Algorithm

HOW CAN WE PREPARE FOR THE LONG-TERM DIGITAL LEARNING SET-UP?

ALGORITHM	KEY STRUCTURE	EVICTION POLICY	TIME COMPLEXITY	SPACE COMPLEXITY	NOTES
FIFO	queue + Hashmap	Oldest item removed	O(1)	O(capacity)	Simple, order- based
LRU	DLL + HashMap	Least recently used	O(1)	O(capacity)	Access pattern-aware.
LFU	Multi-DLL + HashMap	Least frequently used	O(1)	O(capacity)	Frequency- aware, most complex