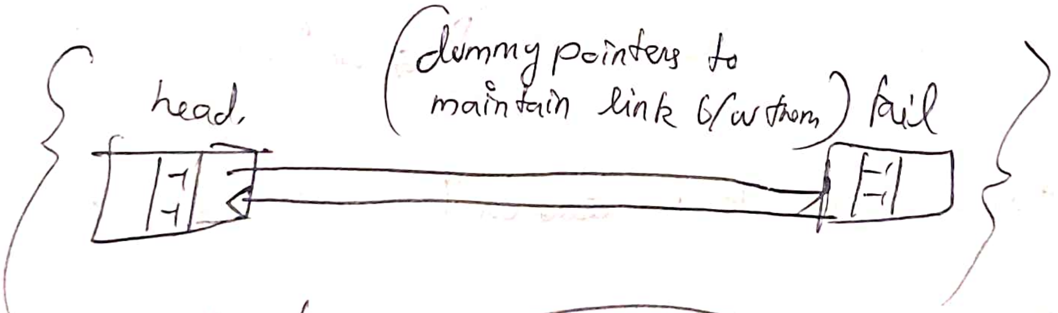
(Amazing Learning about LinkedHashSet and LinkedHashMap)

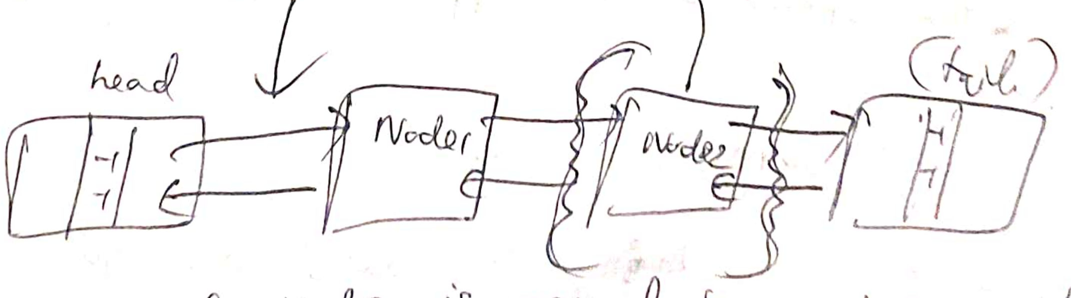
6th April 2025, today while trying to solve the problem, LFU cache, I discovered how flexible and how amazing the collection Framework is of Java.

I came across the LFU cache, problem and to solve LFU you need to have solved LRU cache. Because in LFU cache, when there are two elements with the least frequency. Then removal happens in LRU fashion in that cache

In LRU cache we maintain.



then insertion happens



if Node2 is accessed, then, it is moved to the front in the DLL well now, in the LFU cache, I was planning to maintain

this map<Integer, LRUCache>

→ I was implementing the LRU cache myself, and writing all the put, get and evict() operations myself

And the code kept on getting messy

So, then I saw the solution.

In the solution, I saw. instead of maintaining,

map<Integer, LRUCache> we are maintaing.

map<Integer, LinkedHashSet<Integer>>

→ or Node maybe.

→ well this was new to me as I don't often use Linked hashset,

But the operations it supports is amazing.

O(1) → insert, remove, contains,

while maintainging the order of insertion.

So I got intriguied and below is my research and introduction to LinkedHashSet and LinkedHashMap.

Code Snippet.

```
LinkedHashSet<Node> minFreqNodes = freqNodes.get(minFreq);
   Node evictedNode = minFreqNodes.iterator().next();
```

First node inserted,

```
     minFreqNodes.remove(Node);
```

So LRU caches purpose is served, ~~the just~~

for a get → we get it and remove from the set and reinsert it at the end.

put → insert it at the end normally

I asked chatgpt, how does LinkedHashSet provide O(1) insertion, contains, removal.

LinkedHashSet is backed by HashSet,

which calls the Constructor of LinkedHashMap<>();

(P.T.O)

```
public class LinkedHashSet<E> extends HashSet<E>{

    public LinkedHashSet(int initialCapacity, float loadFactor){
        super(initialCapacity, loadFactor, true);
    }
    :
    ;
}
```

LinkedHashSet and HashSet are in the same package, So it can access it.

```
HashSet<E> extends AbstractSet<E>...{
    :
    : many Constructors!!
    ;
    notice no access type,

    HashSet(int initialCapacity, float loadFactor, boolean dummy){
        map = new LinkedHashMap<>(initialCapacity, loadFactor);
    }
```

ignored.

No access type means its package-private, I did not know that, package-private mean classes in the same package can access this constructor.

Constructor of LinkedHashMap<K,V> extends Hashmap<K,V>

```
public LinkedHashMap(int initialCap, float loadfactor){
        super(initialCap, loadFactor);
        accessOrder = false;
}
```

Now what does this mean?

This means, that order of insertion is preserved and don't order keys by their access order.

***

If accessOrder=true, then LinkedHashMap basically becomes a LRU cache, whatever key you get() or put() is taken to the end of the map, marking it as the most Recently Used.

***

Well, well, Janab. LinkedHashMap provides you this capability as well, to make it behave as a LRU cache.

```
public LinkedHashMap(int initCap, float loadFactor, boolean accessOrder);
```

→ became an LRU cache.

```
LRU = new LinkedHashmap<>(16, 0.75f, true);
```

Difference ~~fro~~ in LinkedHashMap from a normal HashMap, is that it maintains a doubly linked list and has (Key, Node) → map maintain in Hash map.

The entry subclass of Map lookes something like.

```
public LinkedHashMap<K,V> extends Hashmap <K,V> ~ {

  static class Entry <K,V> extends HashMap.Node<K,V>{
       Entry <K,V> before, After;  → DLL

       Entry(int hash, K key, V value,   ) { . ~ }

  };

}.
```