$$\begin{tikzcd} a \arrow[r] & b \arrow[r] & c \arrow[r] & d \end{tikzcd}$$

SLL

$$\begin{tikzcd} a \arrow[r] & b \arrow[r] & c \arrow[r] & d \arrow[lll, bend left] \end{tikzcd}$$

CLL

$$\begin{tikzcd} {} & a \arrow[l] \arrow[r, shift left] & b \arrow[l, shift left] \arrow[r, shift left] & c \arrow[l, shift left] \arrow[r, shift left] & d \arrow[l, shift left] \arrow[r] & {} \end{tikzcd}$$

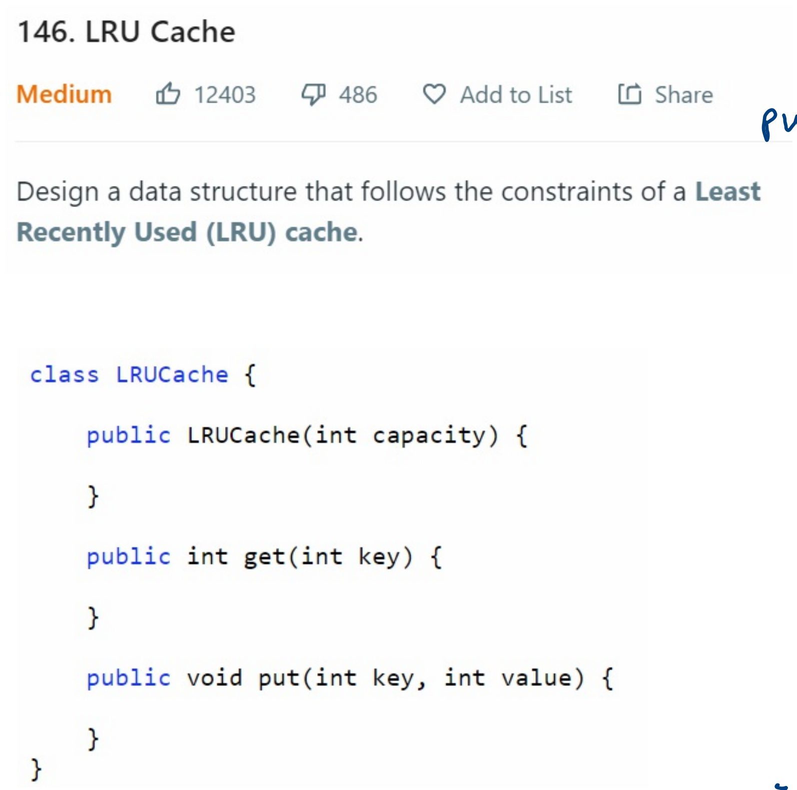
$$\begin{tikzcd} {} \arrow[r, shift left, "\text{next}"] & {} \arrow[l, shift left, "\text{prev}"] \end{tikzcd}$$

DLL

node : value,
next,
prev

$$\begin{tikzcd} a \arrow[r, shift left] \arrow[rrr, bend right=60, "\text{a's prev}"'] & b \arrow[l, shift left] \arrow[r, shift left] & c \arrow[l, shift left] \arrow[r, shift left] & d \arrow[l, shift left] \arrow[lll, bend left=80, "\text{d's next}"] \end{tikzcd}$$

CDLL



146. LRU Cache

Medium 12403 486 Add to List Share

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

```
class LRUCache {

    public LRUCache(int capacity) {

    }

    public int get(int key) {

    }

    public void put(int key, int value) {

    }
}
```

```
put (int key, int value) {

    if (map.ck(key) == true) {
        -> change value
        -> make it most recent

    }
    else {

        addlast();
        if (dru.size() > cap) {
            removeFirst();
        }
    }
}
```

```
get (int key) {
    if( map. ck (key) = = present) {
        Node n = map.get(key);
        removeNode(n);  ]
        addlast(n);     ] -> make 'n' most recent
        return n.value;
    }
    else {
        return -1;
    }
}
```