**LFU Cache**

#include <bits/stdc++.h>

class LFUCache {

public:

struct CacheNode {

int k, v;

CacheNode \*prev, \*next;

CacheNode(int key, int val) {

k = key;

v = val;

}

};

CacheNode \*head = new CacheNode(-1, -1);

CacheNode \*tail = new CacheNode(-1, -1);

int capacity;

unordered\_map<int, CacheNode\*> cache;

LFUCache(int capacity) {

this->capacity = capacity;

head->next = tail;

tail->prev = head;

}

void addNode(CacheNode \*newNode) {

CacheNode \*temp = tail->prev;

newNode->prev = temp;

newNode->next = tail;

temp->next = newNode;

tail->prev = newNode;

}

void deleteNode(CacheNode \*delNode) {

head->next = delNode->next;

delNode->next->prev = head;

}

int get(int key) {

if (cache.find(key) != cache.end()) {

CacheNode \*res = cache[key];

int x = res->v;

cache.erase(key);

deleteNode(res);

addNode(res);

cache[key] = tail->prev;

return x;

}

return -1;

}

void put(int key, int value) {

if (cache.find(key) != cache.end()) {

CacheNode \*existingNode = cache[key];

cache.erase(key);

deleteNode(existingNode);

}

if (cache.size() == capacity) {

cache.erase(head->next->k);

deleteNode(head->next);

}

addNode(new CacheNode(key, value));

cache[key] = tail->prev;

}

};