<https://github.com/striver79/SDESheet/blob/main/LFUCacheCppLeetcode>

|  |
| --- |
| struct Node { |
|  | int key, value, cnt; |
|  | Node \*next; |
|  | Node \*prev; |
|  | Node(int \_key, int \_value) { |
|  | key = \_key; |
|  | value = \_value; |
|  | cnt = 1; |
|  | } |
|  | }; |
|  | struct List { |
|  | int size; |
|  | Node \*head; |
|  | Node \*tail; |
|  | List() { |
|  | head = new Node(0, 0); |
|  | tail = new Node(0,0); |
|  | head->next = tail; |
|  | tail->prev = head; |
|  | size = 0; |
|  | } |
|  |  |
|  | void addFront(Node \*node) { |
|  | Node\* temp = head->next; |
|  | node->next = temp; |
|  | node->prev = head; |
|  | head->next = node; |
|  | temp->prev = node; |
|  | size++; |
|  | } |
|  |  |
|  | void removeNode(Node\* delnode) { |
|  | Node\* delprev = delnode->prev; |
|  | Node\* delnext = delnode->next; |
|  | delprev->next = delnext; |
|  | delnext->prev = delprev; |
|  | size--; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | }; |
|  | class LFUCache { |
|  | map<int, Node\*> keyNode; |
|  | map<int, List\*> freqListMap; |
|  | int maxSizeCache; |
|  | int minFreq; |
|  | int curSize; |
|  | public: |
|  | LFUCache(int capacity) { |
|  | maxSizeCache = capacity; |
|  | minFreq = 0; |
|  | curSize = 0; |
|  | } |
|  | void updateFreqListMap(Node \*node) { |
|  | keyNode.erase(node->key); |
|  | freqListMap[node->cnt]->removeNode(node); |
|  | if(node->cnt == minFreq && freqListMap[node->cnt]->size == 0) { |
|  | minFreq++; |
|  | } |
|  |  |
|  | List\* nextHigherFreqList = new List(); |
|  | if(freqListMap.find(node->cnt + 1) != freqListMap.end()) { |
|  | nextHigherFreqList = freqListMap[node->cnt + 1]; |
|  | } |
|  | node->cnt += 1; |
|  | nextHigherFreqList->addFront(node); |
|  | freqListMap[node->cnt] = nextHigherFreqList; |
|  | keyNode[node->key] = node; |
|  | } |
|  |  |
|  | int get(int key) { |
|  | if(keyNode.find(key) != keyNode.end()) { |
|  | Node\* node = keyNode[key]; |
|  | int val = node->value; |
|  | updateFreqListMap(node); |
|  | return val; |
|  | } |
|  | return -1; |
|  | } |
|  |  |
|  | void put(int key, int value) { |
|  | if (maxSizeCache == 0) { |
|  | return; |
|  | } |
|  | if(keyNode.find(key) != keyNode.end()) { |
|  | Node\* node = keyNode[key]; |
|  | node->value = value; |
|  | updateFreqListMap(node); |
|  | } |
|  | else { |
|  | if(curSize == maxSizeCache) { |
|  | List\* list = freqListMap[minFreq]; |
|  | keyNode.erase(list->tail->prev->key); |
|  | freqListMap[minFreq]->removeNode(list->tail->prev); |
|  | curSize--; |
|  | } |
|  | curSize++; |
|  | // new value has to be added who is not there previously |
|  | minFreq = 1; |
|  | List\* listFreq = new List(); |
|  | if(freqListMap.find(minFreq) != freqListMap.end()) { |
|  | listFreq = freqListMap[minFreq]; |
|  | } |
|  | Node\* node = new Node(key, value); |
|  | listFreq->addFront(node); |
|  | keyNode[key] = node; |
|  | freqListMap[minFreq] = listFreq; |
|  | } |
|  | } |
|  | }; |
|  |  |
|  | /\*\* |
|  | \* Your LFUCache object will be instantiated and called as such: |
|  | \* LFUCache\* obj = new LFUCache(capacity); |
|  | \* int param\_1 = obj->get(key); |
|  | \* obj->put(key,value); |
|  | \*/ |