Amazon以前只要处理200种商品，然后现在要处理200million个。

大家都用一个getItemDetail(id)的function去得到商品信息，但是这个API非常慢。

现在我有一个list<string>,你有什么方法可以speed up吗？

我就说那就用hashtable吧，每次call一下就存进去,在table里了，就不call了。 他说那如果我call这个函数很多次，那怎么办比如第一次 input[1,2] 第二次 input[1,3]

我这时才反应过来要用cache,此时已经过去25分钟了。

adam很开心，说exactlly, that's what I am looking for. Excellent.

然后adam说，这个limit很贵啊，你不能无限存啊，没钱啊我们。

那就用LRU吧，他说对对对，你设计一个？

然后就开始设计LRU, 用double-linked list ＋ hashmap

因为API太慢，所以每当你call一次api的时候，你就把它存到cache里，当下次调用的时候先检查cache再决定用不用api 这个方法应该是很基础的，只是换了一个背景，拖了好久才反应过来。

class LRU Cache as global variable;

class LRUCache{

public:

struct DListNode{

int key;

int val;

DListNode\* prev;

DListNode\* next;

DListNode(int x, int y): key(x), val(y), prev(NULL), next(NULL){}

};

LRUCache(int capacity) {

size = capacity;

head = NULL;

tail = NULL;

}

int get(int key) {

if(table.count(key) == 0)

return -1;

int value = table[key]->val;

update(key);

return value;

}

void set(int key, int value) {

if(table.count(key)){

update(key);

table[key]->val = value;

return;

}

DListNode\* node = new DListNode(key, value);

table[key] = node;

if(!head){

head = node;

tail = node;

}

else{

node->next = head;

head->prev = node;

head = node;

}

if(size > 0) size--;

else{

table.erase(tail->key);

tail = tail->prev;

tail->next = NULL;

}

}

// update to head

void update(int key){

DListNode\* curr = table[key];

if(curr == head) return;

if(curr == tail){

tail = tail->prev;

tail->next = NULL;

}

else{

curr->prev->next = curr->next;

curr->next->prev = curr->prev;

}

curr->prev = NULL;

curr->next = head;

head->prev = curr;

head = curr;

}

DListNode\* head;

DListNode\* tail;

int size;

unordered\_map<int, DListNode\*> table;

};