LRU Cache Implementation

Difficulty Level : Hard

How to implement LRU caching scheme? What data structures should be used?
We are given total possible page numbers that can be referred. We are also given cache (or memory) size (Number of page frames that cache can hold at a time). The LRU caching scheme is to remove the least recently used frame when the cache is full and a new page is referenced which is not there in cache. Please see the Galvin book for more details (see the LRU page replacement slide here).

Recommended: Please solve it on "***PRACTICE***" first, before

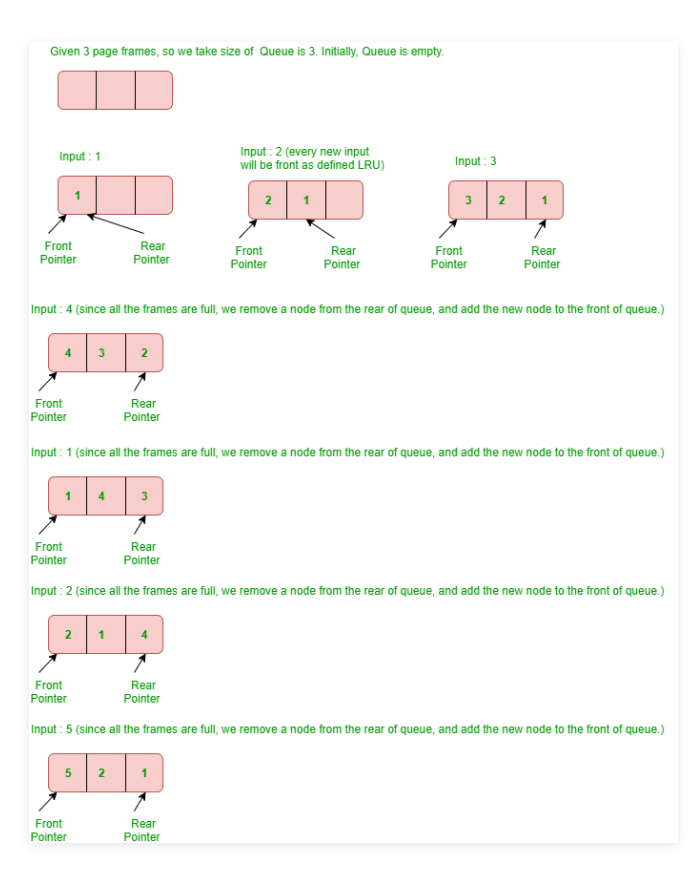
We use two data structures to implement an LRU Cache.

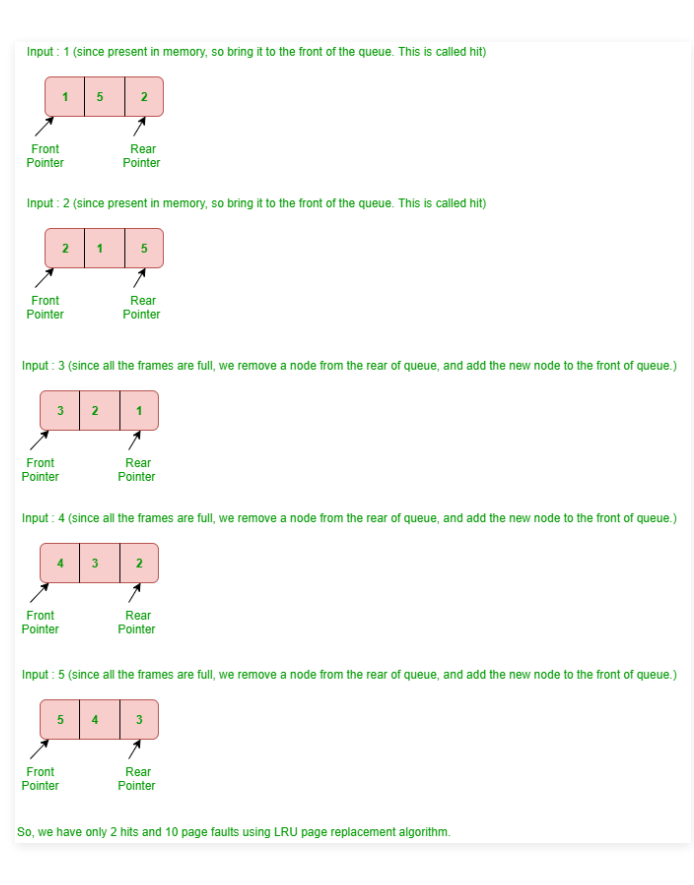
1. **Queue** which is implemented using a doubly linked list. The maximum size of the queue will be equal to the total number of frames available (cache size). The most recently used pages will be near front end and least recently pages will be near the rear end.
2. **A Hash** with page number as key and address of the corresponding queue node as value.

When a page is referenced, the required page may be in the memory. If it is in the memory, we need to detach the node of the list and bring it to the front of the queue.
If the required page is not in memory, we bring that in memory. In simple words, we add a new node to the front of the queue and update the corresponding node address in the hash. If the queue is full, i.e. all the frames are full, we remove a node from the rear of the queue, and add the new node to the front of the queue.
Example – Consider the following reference string :

```
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
```

Find the number of page faults using least recently used (LRU) page replacement algorithm with 3 page frames.
Explanation –





Note: Initially no page is in the memory.

C++ using STL | C | Java

```cpp
// We can use stl container list as a double
// ended queue to store the cache keys, with
// the descending time of reference from front
// to back and a set container to check presence
// of a key. But to fetch the address of the key
// in the list using find(), it takes O(N) time.
// This can be optimized by storing a reference
//     (iterator) to each key in a hash map.
#include <bits/stdc++.h>
using namespace std;

class LRUCache {
    // store keys of cache
    list<int> dq;

    // store references of key in cache
    unordered_map<int, list<int>::iterator> ma;
    int csize; // maximum capacity of cache

public:
    LRUCache(int);
    void refer(int);
    void display();
};

// Declare the size
LRUCache::LRUCache(int n)
{
    csize = n;
}

// Refers key x with in the LRU cache
void LRUCache::refer(int x)
{
    // not present in cache
    if (ma.find(x) == ma.end()) {
        // cache is full
        if (dq.size() == csize) {
            // delete least recently used element
            int last = dq.back();

            // Pops the last elmeent
            dq.pop_back();

            // Erase the last
            ma.erase(last);
        }
    }

    // present in cache
    else
        dq.erase(ma[x]);

    // update reference
    dq.push_front(x);
    ma[x] = dq.begin();
}

// Function to display contents of cache
void LRUCache::display()
{

    // Iterate in the deque and print
    // all the elements in it
    for (auto it = dq.begin(); it != dq.end();
         it++)
        cout << (*it) << " ";

    cout << endl;
}

// Driver Code
int main()
{
    LRUCache ca(4);

    ca.refer(1);
    ca.refer(2);
    ca.refer(3);
    ca.refer(1);
    ca.refer(4);
    ca.refer(5);
    ca.display();

    return 0;
}
// This code is contributed by Satish Srinivas
```

Output

```
5 4 1 3
```

Java Implementation using LinkedHashMap.
The idea is to use a LinkedHashSet that maintains insertion order of elements. This way implementation becomes short and easy.

Java

```java
// Java program to implement LRU cache
// using LinkedHashSet
import java.util.*;

class LRUCache {

    Set<Integer> cache;
    int capacity;

    public LRUCache(int capacity)
    {
        this.cache = new LinkedHashSet<Integer>(capacity
        this.capacity = capacity;
    }

    // This function returns false if key is not
    // present in cache. Else it moves the key to
    // front by first removing it and then adding
    // it, and returns true.
    public boolean get(int key)
    {
        if (!cache.contains(key))
            return false;
        cache.remove(key);
        cache.add(key);
        return true;
    }

    /* Refers key x with in the LRU cache */
    public void refer(int key)
    {
        if (get(key) == false)
           put(key);
    }

    // displays contents of cache in Reverse Order
    public void display()
    {
      LinkedList<Integer> list = new LinkedList<>(cache)

      // The descendingIterator() method of java.util.Li
      // class is used to return an iterator over the el
      // in this LinkedList in reverse sequential order
      Iterator<Integer> itr = list.descendingIterator();

      while (itr.hasNext())
            System.out.print(itr.next() + " ");
    }

    public void put(int key)
    {

      if (cache.size() == capacity) {
            int firstKey = cache.iterator().next();
            cache.remove(firstKey);
        }

        cache.add(key);
    }

    public static void main(String[] args)
    {
        LRUCache ca = new LRUCache(4);
        ca.refer(1);
        ca.refer(2);
        ca.refer(3);
        ca.refer(1);
        ca.refer(4);
        ca.refer(5);
        ca.display();
    }
}
```

Output

```
5 4 1 3
```

Python implementation using OrderedDict
This article is compiled by Aashish Barnwal and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Attention reader! Don't stop learning now. Get hold of all the important DSA concepts with the **DSA Self Paced Course** at a student-friendly price and become industry ready. To complete your preparation from learning a language to DS Algo and many more, please refer **Complete Interview Preparation Course.**

In case you wish to attend live classes with industry experts, please refer **Geeks Classes Live**

Like 0

Previous
Least Frequently Used (LFU) Cache Implementation

Next
Implement Stack using Queues

RECOMMENDED ARTICLES

Page : 1 2 3

01 Least Frequently Used (LFU) Cache Implementation
12, Jul 18

02 Locality of Reference and Cache Operation in Cache Memory
12, Nov 18

03 Difference between RAM and Cache
11, Aug 20

04 DNS Spoofing or DNS Cache poisoning
12, Jan 18

05 Cache Memory Design
25, Jun 19

06 How to Implement Reverse DNS Look Up Cache?
10, Dec 14

07 How to Implement Forward DNS Look Up Cache?
17, Dec 14

08 Multilevel Cache Organisation
19, Dec 18

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : _Gaurav_Tiwari, spattk, MadhaviVaddepalli, Majorssn, nirajtechi, sudhanshublaze

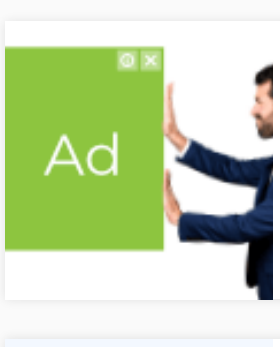
Article Tags : Amazon, cpp-unordered_map, MakeMyTrip, Morgan Stanley, Snapdeal, STL, Advanced Data Structure, GATE CS, Operating Systems, Queue

Practice Tags : Morgan Stanley, Amazon, Snapdeal, MakeMyTrip, Operating Systems, Queue, STL

Improve Article | Report Issue

WHAT'S NEW


DSA Self Paced Course
Learn DSA with recorded video lectures and aim top product-based companies
View Details


Ad free experience with GeeksforGeeks Premium
View Details


Advanced DSA Live Classes for Interview Preparation
Prepare advanced DSA to crack interviews at top product-based companies with these Live Online Classes
View Details

MOST POPULAR IN ADVANCED DATA STRUCTURE

- Disjoint Set Data Structures
- Insert Operation in B-Tree
- Design a Chess Game
- Red-Black Tree | Set 2 (Insert)
- XOR Linked List – A Memory Efficient Doubly Linked List | Set 1

MOST VISITED IN GATE CS

- ACID Properties in DBMS
- Layers of OSI Model
- Types of Operating Systems
- Normal Forms in DBMS
- Functions of Operating System

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments