

Implement LRU Cache

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](#)).

We strongly recommend that you click here and practice it, before moving on to the solution.

GeeksforGeeks GATE C
S Corner

GeeksforGeeks Practice



**We match you with
great startups.**

Start by taking our online
coding quiz.

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 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 the 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 queue, and add the new node to the front of queue.

Note: Initially no page is in the memory.

C implementation:

```
// A C program to show implementation of LRU cache
#include <stdio.h>
#include <stdlib.h>

// A Queue Node (Queue is implemented using Doubly Linked List)
typedef struct QNode
{
    struct QNode *prev, *next;
```



```

    unsigned pageNumber; // the page number stored in this QNode
} QNode;

// A Queue (A FIFO collection of Queue Nodes)
typedef struct Queue
{
    unsigned count; // Number of filled frames
    unsigned numberOfFrames; // total number of frames
    QNode *front, *rear;
} Queue;

// A hash (Collection of pointers to Queue Nodes)
typedef struct Hash
{
    int capacity; // how many pages can be there
    QNode* *array; // an array of queue nodes
} Hash;

// A utility function to create a new Queue Node. The queue Node
// will store the given 'pageNumber'
QNode* newQNode( unsigned pageNumber )
{
    // Allocate memory and assign 'pageNumber'
    QNode* temp = (QNode *)malloc( sizeof( QNode ) );
    temp->pageNumber = pageNumber;

    // Initialize prev and next as NULL
    temp->prev = temp->next = NULL;

    return temp;
}

```

Videos by GeeksforGeeks

Placement Course

Like us on Facebook

Follow us on Twitter

Recent Comments

Jumpstart **Raspberry Pi** &
Arduino Projects with **Cayenne**

```
// A utility function to create an empty Queue.
// The queue can have at most 'numberOfFrames' nodes
Queue* createQueue( int numberOfFrames )
{
    Queue* queue = (Queue *)malloc( sizeof( Queue ) );

    // The queue is empty
    queue->count = 0;
    queue->front = queue->rear = NULL;

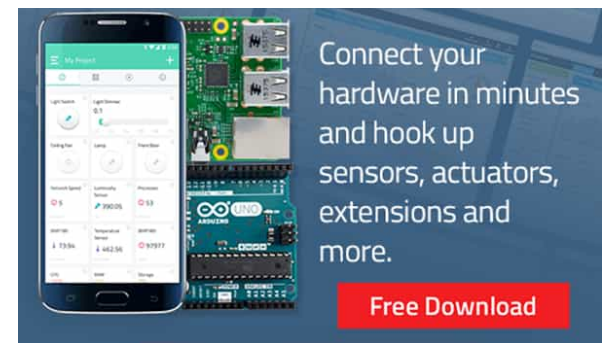
    // Number of frames that can be stored in memory
    queue->numberOfFrames = numberOfFrames;

    return queue;
}

// A utility function to create an empty Hash of given capacity
Hash* createHash( int capacity )
{
    // Allocate memory for hash
    Hash* hash = (Hash *) malloc( sizeof( Hash ) );
    hash->capacity = capacity;

    // Create an array of pointers for referring queue nodes
    hash->array = (QNode **) malloc( hash->capacity * sizeof( QNode* ) );

    // Initialize all hash entries as empty
    int i;
    for( i = 0; i < hash->capacity; ++i )
```



Popular Posts

- [Top 10 Algorithms and Data Structures for Competitive Programming](#)
- [Top 10 algorithms in Interview Questions](#)
- [How to begin with Competitive Programming?](#)
- [Step by Step Guide for Placement Preparation](#)
- [Reflection in Java](#)
- [Memory Layout of C Programs](#)
- [Heavy Light Decomposition](#)
- [Sorted Linked List to Balanced BST](#)
- [Generics in Java](#)
- [Aho-Corasick Algorithm for Pattern Searching](#)
- [Insertion Sort , Binary Search , QuickSort , MergeSort , HeapSort](#)

```

        hash->array[i] = NULL;

    return hash;
}

// A function to check if there is slot available in memory
int AreAllFramesFull( Queue* queue )
{
    return queue->count == queue->numberOfFrames;
}

// A utility function to check if queue is empty
int isEmptyQueue( Queue* queue )
{
    return queue->rear == NULL;
}

// A utility function to delete a frame from queue
void dequeue( Queue* queue )
{
    if( isEmptyQueue( queue ) )
        return;

    // If this is the only node in list, then change front
    if (queue->front == queue->rear)
        queue->front = NULL;

    // Change rear and remove the previous rear
    QNode* temp = queue->rear;
    queue->rear = queue->rear->prev;
}

```

- [Common Interview Puzzles](#)
- [Interview Experiences](#)
- [Advanced Data Structures](#)
- [Design Patterns](#)
- [Dynamic Programming](#)
- [Greedy Algorithms](#)
- [Backtracking](#)
- [Pattern Searching](#)
- [Divide & Conquer](#)
- [Geometric Algorithms](#)
- [Searching](#)
- [Sorting](#)
- [Analysis of Algorithms](#)
- [Mathematical Algorithms](#)
- [Randomized Algorithms](#)
- [Recursion](#)
- [Game Theory](#)

Tags

[Adobe](#)
[Advanced Data Structure](#)
[Amazon](#)
[Arrays](#)
[Articles](#)
[Binary Search Tree](#)
[Bit Magic](#)
[C/C++](#)

```

if (queue->rear)
    queue->rear->next = NULL;

free( temp );

// decrement the number of full frames by 1
queue->count--;
}

// A function to add a page with given 'pageNumber' to both queue
// and hash
void Enqueue( Queue* queue, Hash* hash, unsigned pageNumber )
{
    // If all frames are full, remove the page at the rear
    if ( AreAllFramesFull ( queue ) )
    {
        // remove page from hash
        hash->array[ queue->rear->pageNumber ] = NULL;
        deQueue( queue );
    }

    // Create a new node with given page number,
    // And add the new node to the front of queue
    QNode* temp = newQNode( pageNumber );
    temp->next = queue->front;

    // If queue is empty, change both front and rear pointers
    if ( isEmpty( queue ) )
        queue->rear = queue->front = temp;
    else // Else change the front
    {

```

Puzzles Competitive Programming
 cpp-library Divide and Conquer
Dynamic Programming
 Experienced
 CS-DS-&-Algo GBlog Graph
 Graph Greedy Hash Internship
Interview
Experiences Java-
 Collections Linked Lists
Mathematical Matrix
MCQ Microsoft Misc number-
 digits Oracle Output Pattern Searching
 Project Python Searching
 Sorting Stack STL Strings
 subsequence Technical Scripter
Trees XOR

Subscribe and Never Miss an Article

Email Address

Subscribe

```

        queue->front->prev = temp;
        queue->front = temp;
    }

    // Add page entry to hash also
    hash->array[ pageNumber ] = temp;

    // increment number of full frames
    queue->count++;
}

// This function is called when a page with given 'pageNumber' is
// referenced
// from cache (or memory). There are two cases:
// 1. Frame is not there in memory, we bring it in memory and add
//    to the front
//    of queue
// 2. Frame is there in memory, we move the frame to front of que
ue
void ReferencePage( Queue* queue, Hash* hash, unsigned pageNumber
)
{
    QNode* reqPage = hash->array[ pageNumber ];

    // the page is not in cache, bring it
    if ( reqPage == NULL )
        Enqueue( queue, hash, pageNumber );

    // page is there and not at front, change pointer
    else if (reqPage != queue->front)
    {

```

```

// Unlink requested page from its current location
// in queue.
reqPage->prev->next = reqPage->next;
if (reqPage->next)
    reqPage->next->prev = reqPage->prev;

// If the requested page is rear, then change rear
// as this node will be moved to front
if (reqPage == queue->rear)
{
    queue->rear = reqPage->prev;
    queue->rear->next = NULL;
}

// Put the requested page before current front
reqPage->next = queue->front;
reqPage->prev = NULL;

// Change prev of current front
reqPage->next->prev = reqPage;

// Change front to the requested page
queue->front = reqPage;
}
}

// Driver program to test above functions
int main()
{
    // Let cache can hold 4 pages
    Queue* q = createQueue( 4 );

```



```
// Let 10 different pages can be requested (pages to be
// referenced are numbered from 0 to 9
Hash* hash = createHash( 10 );

// Let us refer pages 1, 2, 3, 1, 4, 5
ReferencePage( q, hash, 1);
ReferencePage( q, hash, 2);
ReferencePage( q, hash, 3);
ReferencePage( q, hash, 1);
ReferencePage( q, hash, 4);
ReferencePage( q, hash, 5);

// Let us print cache frames after the above referenced pages
printf ("%d ", q->front->pageNumber);
printf ("%d ", q->front->next->pageNumber);
printf ("%d ", q->front->next->next->pageNumber);
printf ("%d ", q->front->next->next->next->pageNumber);

return 0;
}
```

Output:

```
5 4 1 3
```

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.

GATE CS Corner Practice

Company Wise Coding

Advanced Data Structure Queue

Advance Data Structures

Advanced Data Structures Queue

Related Posts:

- [Second minimum element using minimum comparisons](#)
- [Count of distinct substrings of a string using Suffix Trie](#)

- Summed Area Table – Submatrix Summation
- Unrolled Linked List | Set 1 (Introduction)
- proto van Emde Boas Trees | Set 1 (Background and Introduction)
- Implement a Phone Directory
- Binary Indexed Tree : Range Update and Range Queries
- Count and Toggle Queries on a Binary Array

[<< Previous Post](#)[Next Post >>](#)

([Login](#) to Rate and Mark)

3.7

Average Difficulty : **3.7/5.0**
Based on **72** vote(s)



Add to TODO List



Mark as DONE

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

Load Comments

@geeksforgeeks, Some rights reserved

[Contact Us!](#)

[About Us!](#)

[Advertise with us!](#)

[Privacy Policy](#)