

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 use two data structures to implement an LRU Cache.

1. A 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.

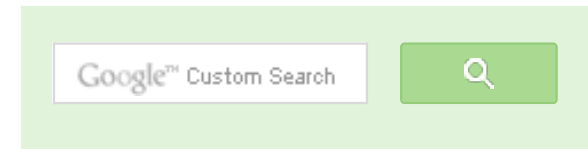
2. A Hash with page number as key and address of the corresponding queue node as value.

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.

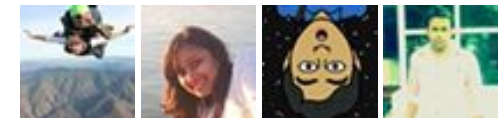
Below is C implementation:



GeeksforGeeks



53,527 people like [GeeksforGeeks](#).



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

[Geometric Algorithms](#)

```
// 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;
}

// 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
```



Popular Posts

All permutations of a given string

Memory Layout of C Programs

Understanding "extern" keyword in C

Median of two sorted arrays

Tree traversal without recursion and without stack!

Structure Member Alignment, Padding and Data Packing

Intersection point of two Linked Lists

Lowest Common Ancestor in a BST.

Check if a binary tree is BST or not

Sorted Linked List to Balanced BST

```

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 )
        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 isEmpty( Queue* queue )
{
    return queue->rear == NULL;
}

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

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

```

MongoDB just got Better.

GO

 **rackspace**
the open cloud company

```
queue->front = NULL;
```

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

```
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
```

```
    {
```

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

```
        queue->front = temp;
```

```
    }
```

```
    // Add page entry to hash also
```

```
    hash->array[ pageNumber ] = temp;
```

```
    // increment number of full frames
```

```
    queue->count++;
```

```
}
```

Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 44 minutes ago

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 1 hour ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 1 hour ago

Sanjay Agarwal bool

tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago


GOPI GOPINATH @admin Highlight this

sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 2 hours ago

AdChoices 

► [Cache Java](#)

► [Cache To](#)


► [Something Cache](#)

AdChoices 

► [Cache Memory](#)

► [Cache Cache](#)

► Implement

AdChoices 

► Array Cache

► Cache String

► Cache Up

```
// This function is called when a page with given 'pageNumber' is refe
// from cache (or memory). There are two cases:
// 1. Frame is not there in memory, we bring it in memory and add to t
//    of queue
// 2. Frame is there in memory, we move the frame to front of queue
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 rquested 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);

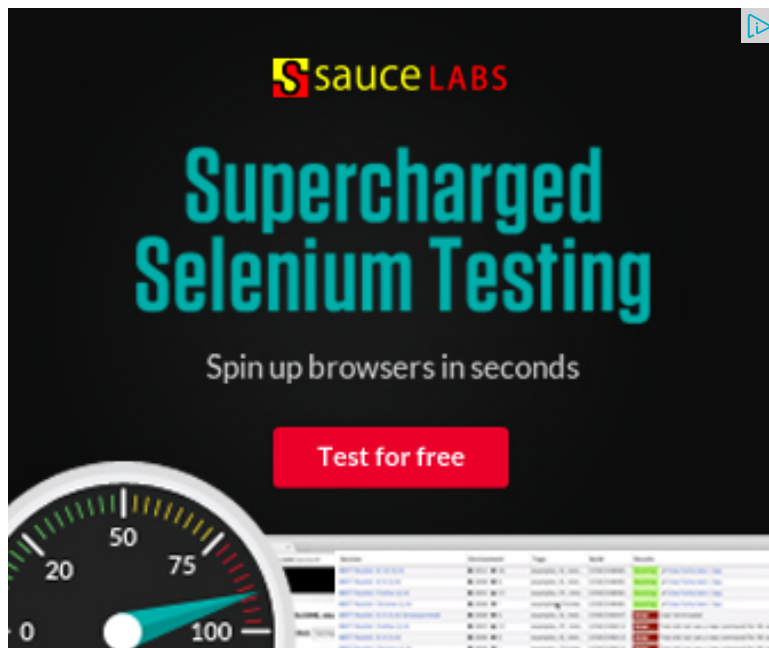
return 0;
}

```

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.



Related Tpoics:

- Given a linked list, reverse alternate nodes and append at the end
- Pairwise swap elements of a given linked list by changing links
- Self Organizing List | Set 1 (Introduction)
- Merge a linked list into another linked list at alternate positions
- QuickSort on Singly Linked List
- Delete N nodes after M nodes of a linked list
- Design a stack with operations on middle element
- Swap Kth node from beginning with Kth node from end in a Linked List



5



Tweet

0



0

Writing code in comment? Please use [ideone.com](https://www.ideone.com) and share the link here.

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team