//C++ implementation of above algorithm

#include<bits/stdc++.h>

using namespace std;

// Function to find page faults using indexes

int pageFaults(int pages[], int n, int capacity)

{

// To represent set of current pages. We use

// an unordered\_set so that we quickly check

// if a page is present in set or not

unordered\_set<int> s;

// To store least recently used indexes

// of pages.

unordered\_map<int, int> indexes;

// Start from initial page

int page\_faults = 0;

for (int i=0; i<n; i++)

{

// Check if the set can hold more pages

if (s.size() < capacity)

{

// Insert it into set if not present

// already which represents page fault

if (s.find(pages[i])==s.end())

{

s.insert(pages[i]);

// increment page fault

page\_faults++;

}

// Store the recently used index of

// each page

indexes[pages[i]] = i;

}

// If the set is full then need to perform lru

// i.e. remove the least recently used page

// and insert the current page

else

{

// Check if current page is not already

// present in the set

if (s.find(pages[i]) == s.end())

{

// Find the least recently used pages

// that is present in the set

int lru = INT\_MAX, val;

for (auto it=s.begin(); it!=s.end(); it++)

{

if (indexes[\*it] < lru)

{

lru = indexes[\*it];

val = \*it;

}

}

// Remove the indexes page

s.erase(val);

// insert the current page

s.insert(pages[i]);

// Increment page faults

page\_faults++;

}

// Update the current page index

indexes[pages[i]] = i;

}

}

return page\_faults;

}

// Driver code

int main()

{

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

int n = sizeof(pages)/sizeof(pages[0]);

int capacity = 4;

cout << pageFaults(pages, n, capacity);

return 0;

}

Output:

6

Another approach: (Without using HashMap)

// C++ program for page replacement algorithms

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

int main()

{

int capacity = 4;

int arr[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

deque<int> q(capacity);

int count=0;

int page\_faults=0;

deque<int>::iterator itr;

q.clear();

for(int i:arr)

{

// Insert it into set if not present

// already which represents page fault

itr = find(q.begin(),q.end(),i);

if(!(itr != q.end()))

{

++page\_faults;

// Check if the set can hold equal pages

if(q.size() == capacity)

{

q.erase(q.begin());

q.push\_back(i);

}

else{

q.push\_back(i);

}

}

else

{

// Remove the indexes page

q.erase(itr);

// insert the current page

q.push\_back(i);

}

}

cout<<page\_faults;

}