Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Статья № 1

ЗАО ”Сайенс”

Работу выполнил:
Поздняков А.А.
Группа:
3530904/00104
Руководитель:
Касилов В.А.

Санкт-Петербург
2023

Содержание

1. Введение

Цель работы - разработать многопоточный кэш с методом вытеснения LRU, сравнить собственную реализацию с реализациями в библиотеках с открытым исходным кодом.

Для достижения поставленной цели в работе были поставленны следующие задачи:

1. Изучить подходы к реализации LRU кэша
2. Разработать спецификацию интерфейса разрабатываемого класса
3. Разработать тесты, описывающие поведение разрабатываемого класса
4. Реализовать класс кэша
5. Сравнить собственную реализацию с реализациями в свободно доступных библиотеках

2. Требования

2.1. Спецификация интерфейса

2.1.1. Шаблон класса polylru::LRU< Key, Value >

Данный шаблон класса реализует кэш с методом вытесенения LRU(Least recently used).
`#include <lru.hpp>`

Открытые типы

- typedef std::pair< Key, Value > **value_type**
- typedef Key **key_type**
- typedef Value **mapped_type**

Открытые члены

- **LRU** (size_t capacity)
- Value get (Key key)

  Возвращает значение, установленное для указанного ключа.
- void set (Key key, Value value)

  Устанавлияет в кэше значение с указанным ключом.
- bool contains (Key key) const noexcept

  Проверяет, сохранён ли в кэше элемент с указанным ключом.
- size_t size () const noexcept

  Возвращает текущее количество элементов в кэше.
- size_t capacity () const noexcept

  Возвращает текущую ёмкость кэша.
- void resize (size_t new_cap)

  Изменяет ёмкость кэша.

Подробное описание

template<typename Key, typename Value>
class polylru::LRU< Key, Value >

Данный шаблон класса реализует кэш с методом вытесенения LRU(Least recently used).

Элементы хранятся ассоциативно, в формате ключ-значение, ёмкость кэша задаётся при создании объекта. Размером кэша считается количество элементов, сохранённое в контейнере. Если размер кэша равен его ёмкости, он считается **заполненным**. Возраст элементов определяется количеством обращений к кэшу, в течение которого они не были использованы. Добавление элемента с новым ключом в заполненный кэш влечёт за собой вытесенение из кэша по методу LRU, самый старый элемент удаляется.

Функция resize(size_t) позволяет динамически изменить размер контейнера.

Методы

get() `template<typename Key , typename Value >`
`Value polylru::LRU< Key, Value >::get (`
`Key` *`key`* `)`

Возвращает значение, установленное для указанного ключа.

В случае, если элемент с указанным ключом присутствует в кэше, возвращает значение элемента. Попытка доступа по неизвестному ключу влечёт неопределенное поведение.

resize() `template<typename Key , typename Value >`
`void polylru::LRU< Key, Value >::resize (`
`size_t` *`new_cap`* `)`

Изменяет ёмкость кэша.

Метод уменьшает размер кэша в случае, если переданный размер меньше текущего, при этом происходит удаление самых старых элементов. В случае, если переданный размер больше текущего, кэш увеличивается до указанного размера. Если переданный размер равен текущему, изменений не происходит.

set() `template<typename Key , typename Value >`
`void polylru::LRU< Key, Value >::set (`
`Key` *`key,`*
`Value` *`value`* `)`

Устанавлияет в кэше значение с указанным ключом.

В случае, если элемент с указанным ключом сохранён в кэше, устанавливает новое значение для данного ключа. В случае, если элемент с указанным ключом отсутствует в кэше, добавляет элемент в кэш. Добавление элемента в заполненный кэш приводит к вытеснению самого старого элемента.

size() `template<typename Key , typename Value >`
`size_t polylru::LRU< Key, Value >::size [noexcept]`

Возвращает текущее количество элементов в кэше.

В случае, если размер кэша равен его ёмкости, добавление элемента с новым ключом привёдет к вытеснению самого старого элемента.

Объявления и описания членов класса находятся в файле:

- lib/src/lru/lru.hpp

2.2. Спецификация поведения

- IF the size of cache is 1 AND there is key k1 with value v1 present in cache AND set with k1 v2 is called, THEN v2 overwrites v1
- IF the size of cache is >1 AND there is item k1 v1 in cache AND set with k1 v2 is called, THEN v2 overwrites v1
- IF the value with key k1 was set, THEN get with key k1 returns the samevalue
- IF the constructor is called with positive size, THEN it doesn't throw
- IF the size of cache is 1 AND the value with key k1 is not present incache AND contains is called with key k1, THEN it returns false
- IF x elements stored in cache with capacity y AND resize with z > y is called, THEN all elements are copied to the resized cache AND capacity is changed to z

- IF get with unknown key is called, THEN the behaviour is undefined
- IF x elements stored in cache with capacity y AND resize with z < y is called, THEN y - z oldest elements are eliminated AND z newest elements are copied to the resized cache AND capacity is changed to z
- IF the value with key k1 is not present in cache, THEN contains with key k1 returns false
- IF the value with key k1 is present in cache, THEN contains with key k1 returns true
- IF x elements with different keys were set AND x > capacity AND size is called, THEN it returns capacity
- IF the size of cache is >1 AND there is no free space in cache AND k1 is not present in cache AND set k1 v1 is called, THEN new item displaces the least recently used item
- IF the size of cache is 1 AND there is no more free space in cache AND set is called with new key THEN it displaces least recently used item
- IF x elements with different keys were set AND x <= capacity AND size is called, THEN it returns x
- IF the constructor is called with negative size, THEN it throws std::length_error
- IF the value with key k1 was set, THEN get with key k1 returns the samevalue
- IF the size of cache is 1 AND the value with key k1 was set AND get is called with key k1, THEN it returns the same value
- IF the size of cache is 1 AND the value with key k1 is present in cache AND contains is called with key k1 THEN it returns true
- IF the value with key k1 was set, THEN contains with key k1 returns true
- IF the size of cache is >1 AND there is free space in cache AND set with new key is called, THEN it stores item in cache
- IF the cache was constructed with capacity x AND capacity is called, THEN it returns x
- IF the constructor is called with size 0, THEN it throws std::length_error

3. Список литературы

Rocca M. L. Advanced Algorithms and Data Structures. — 1-е изд. — Manning Publications, 2021. — ISBN 1617295485,9781617295485. — URL: http://gen.lib.rus.ec/book/index.php?md5=47B7D19325B663DC8AE94A1FA17933D8.