Хеш структури C++

Аввакумов володимир

10 листопада 2024 р.

Зміст

1 Вступ

Я обрав свою власну тему, як зрозуміло з назви - це реалізація хеш структур, а саме словаря та сета мовою c++, в подальшому тексті опи-

шу труднощі при створенні, вибрані мною варіанти реалізації та надам інформацію щодо необхідних бібліотек длязбірки проєкту.

2 Коротко про роботу

Документація є в самому коді, ту т я поясню що в мене вийшло. Проєкт містить в собі 4 класи HashSet, LinkedList, HashDict, LinkedList_dict, - сет, зв'язний список як елемент сета, словник, зв'язний список як елемент словника відповідно. Тілом сета є масив, який динамічно збільшується, щойно заповниться більше ніж на 75 відсотків, в комірках масиву знаходяться зв'язні списки. В сет можна додавати елементи, видаляти їх та перевіряти чи є якийсь елемент в сеті, додавання елементу працює наступним чином: ми беремо хеш значення вхідної змінної ділимо його з остачею (беремо остачу від ділення) на розмір масиву та додаємо змінну в мисив да індексом її хешу поділеного на розмір масиву, видалення та перевірка на наявність працюють аналогічно. Щодо словника то він виконує тіж самі функції, але змерігає пари ключ значення, ключ те ж саме що і змінна в сеті, в звязку з цим в свовника є декілька своїх власник методів таких як зміна значення за ключем та діставання знаячення за ключем, це реаізовано перевантаженням методу [ ], під капотом відбувається вже знайома нам логіка з хешуванням ключа та діленням з остачею на розмір.

3 Прийняті рішення для реалізації

Є безліч варіантів для написання хеш структур в основному вони відрізняються методом вирішення колізій, особисто я обрав варіант через зв'язні списки(LinkedList та LinkedList_dict), скажу на перед це дуже впливає на розмір коду та його читабельність, адже методи розділені між двонма класами інших недоліків помічено не було. Далі я вирішив що напишу спочатку сет, а словник від нього успадкую, але, на жаль, написавши його я зрозумів що успадкуватися майже неможливо (адже обидва класи шаблонні, ри чому у першого один шаблонний параметер, а у другого їх 2), тому прийшлось дублювати купу коду.

4 Проблеми що виникли

Найпершою проблемою була наможливість спадкування, але виявилося це добре, я частково переробив функції хешування, щоб вони приймали

розмір масиву як параметр, це виявилось доволі влучним при копіюванні основного буфера в буфер побільше, наступною ж проблемою була не компіляція проєкта, адже лінкер постійно губився, це був жах, я вбив на спроби компіляції біля 4-х годин, перепробував 3 основних варіанти і імперичним методом зрозумів що краще ніж описати та імплементувати функції в хедері для заголовочного класу варіантів немає. остання проблема мене спідкала, коли я написав юніт тести для обох класів і в мене не працювала функція для is_in для класу std::string, виявилось що, по-перше, треба всюди поставити перевірки на невід'ємність хешу, а токож, мабуть айголовніши було те, що c++ доволі калічно переводить клас long long в int. після потрачиних 2-х годин на дебаг я виправив цей недолік.

5 Необхідні бібліотеки для запуску

gtests (googletests)
stl (iostream, stdexcept, type_traits, etc.)
за великим рахуноком сторонні бібліотеки використовуються лише для перевантаження оператора « для std::cout та помилок, ну і для тестів звісно.

6 Тести

6.1 тест для HashSet

6.1.1 конфігураційний файл(CMAKE)

Коротко
Файли: dict/tests/set_test_2.cpp, dict/set/HashSet.h, dict/set/LinkedList.h
Бібліотеки: gtests
CXX STANDETD 20
Детально
cmake_minimum_required(VERSION 3.28)

set(CMAKE_CXX_STANDARD 20)

find_package(GTest REQUIRED)

include_directories($GTEST_INCLUDE_DIRS)

```
add_executable(project_directory
        dict/tests/set_test_2.cpp
        dict/set/HashSet.h
        dict/set/LinkedList.h
)
```

target_link_libraries(project_directory GTest::gtest GTest::gtest_main pthread)

6.1.2 суть тесту

Тест складається з 6-ти підтестів

AddAndIsInTest - Додавання великої кількості цілочислельних значень та перевірка їх наявності

PopMethodLargeData - Додавання великої кількості цілочислельних значень, видалення частини та перевірка на їх відсутність

FloatKeys - Те ж саме що і в AddAndIsInTest але для флоат

StringKeys - Теж саме що і AddAndIsInTest але для std::string

ResizeOnHighOccupancy - тест на те що масив збільшується, він тут зайвий

PopNonExistentElemen - Перевірка на виникнення помилки при спробі видалити неіснуючий елемент

тести перевіряють основний функціонал хеш сету, звісно всі проходяться

6.2 тести для HashDict

6.2.1 конфігураційний файл(CMAKE)

Коротко:
Файли: dict/tests/selected_test.cpp , dict/HashDict.h, dict/LinkedList_dict.h
Бібліотеки: gtests
Варіанти для selected_test: dict_test_1, dict_test_2, dict_test_3
CXX STANDETD 20
Детально:
cmake_minimum_required(VERSION 3.28)

```
set(CMAKE_CXX_STANDARD 20)

find_package(GTest REQUIRED)

include_directories($GTEST_INCLUDE_DIRS)


                    add_executable(project_directory
                    dict/tests/selected_test.cpp
                    dict/HashDict.h
                    dict/LinkedList_dict.h
                    )
target_link_libraries(project_directory GTest::gtest GTest::gtest_main pthread)
```

6.2.2 dict_test_1

Тут доволі багато маленьких підтестів, не бачу сенс їх пояснювати, він потрібен була скоріше для відладки та бета тестування.

6.2.3 dict_test_2

Тест складається з 4-х підтестів

LargeFloatKeys

LargeStringKeys

IntPointerKeys

FloatPointerKeys

Тут всі підтести перевіряють додавання пар ключ-значення чи дітаються значення за ключем за допомогою оператора [ ], відрізняються лише типом ключа, цілочесельний, дійсний, покажчик на цілочисельий, покажчик на дійсний відповідно, це зрзуміло з їхньої назви.

6.2.4 dict_test_3

Тест складається з 3-х підтестів

PopMethodLargeData

BracketOperatorLargeData

MixedTypePopAndAccess

PopMethodLargeData додає велику кількоість пар (int, std::string) видаляє половину, далі перевіряє чи дійсно видалені видалились, а не видалені не видалились
BracketOperatorLargeData теж саме що і в dict_test_2 але також тестується доступ до неіснуючих елементів
MixedTypePopAndAccess схожий на PopMethodLargeData, але видаляє кожен третій елемент.

6.2.5 Загалом про тести

Всі тести, звісно, проходяться ідеально, також зазначу що використовував цей словник в домашній роботі з int ключами та arrey значеннями, проблем не виникло.

7 Самокритика

Опишу основні недоробки.

7.1 Dict

1. При використанні вказівників в якості ключів чи значень, їх треба вручну видаляти, викликає дискомфорт при роботі з класом HashDict.

2. неможливість зразу записати всі елементи, тобто так HashDict d{1:2, 3:4, 5:6}.

3. Недосконаліть хеш функцій. Вони доволі повільні та часто дають повтори.

7.2 Set

Теж саме що і в Dict, але ще є кілька, в зв'язку з тим що я робив його першим і він був такою собі пробою пера

1. Хеш функція повертає число, аке вже потів в потрібній нам функції ділиться на розмір масиву, це збільшує кількість коду.

2. Ітератор в LinkedList доволі зайвий.