Bài tập lớn 3
Hiện thực Cache (phần tiếp theo)

TS. Nguyễn Hứa Phùng

Tháng 5/2021

1 Chuẩn đầu ra

Sau khi hoàn thành bài tập lớn này, sinh viên sẽ có khả năng

- Hiện thực thao tác tìm kiếm bằng hash
- Chọn lựa và vận dụng các cấu trúc dữ liệu phù hợp để đạt được các kết quả mong muốn.

2 Giới thiệu

Trong bài tập lớn vừa rồi, các em đã sử dụng cây nhị phân cân bằng (AVL) để tìm kiếm trên cache và dùng cơ chế FIFO bằng hàng đợi để thay thế cache. Tuy nhiên, trong thực tế, có rất nhiều cách tiếp cận khác nhau để hiện thực việc tìm kiếm và thay thế cache. Và dù là cách nào đi nữa thì cũng sẽ có mặt lợi và mặt hại. Trong bài tập lớn này, các em sẽ phải thực hiện nhiều cách tìm kiếm và thay thế cache khác nhau. Bên cạnh đó, các cách tiếp cận này phải được kết hợp trong cùng một hệ thống để cache có thể được sử dụng với nhiều cách khác nhau.

Về cách tìm kiếm, bài tập lớn này yêu cầu hai cách hiện thực, đó là: sử dụng cây AVL và hàm băm. Và 4 cách thực hiện cơ chế thay thế cache như sau:

- LRU (Least Recently Used): dữ liệu được sử dụng trễ nhất sẽ được chọn
- MRU (Most Recently Used): dữ liệu được sử dụng gần đây nhất sẽ được chọn
- LFU (Least Frequently Used): dữ liệu ít được sử dụng nhất sẽ được chọn
- FIFO (First In First Out): dữ liệu cũ nhất trong cache sẽ được chọn

Các cơ chế này phải được hiện thực ở các lớp khác nhau. Khi tạo một cache, nó phải được truyền vào một đối tượng tìm kiếm và một đối tượng khác cho việc thay thế. Chi tiết sẽ được mô tả trong phần tiếp theo.

2.1 Hiện thực Cache

Cache, được triển khai trong lớp Cache, khi nó được khởi tạo sẽ có hai tham số sau: tham số thứ nhất là công cụ tìm kiếm và tham số còn lại là cơ chế thay thế cache. Interface như read, put, write của cache sẽ không thay đổi, trong khi đó, các phương thức khác như: print, preOrder, inOrder được thay thế bằng printRP và printSE. Hai phương thức này được mô tả chi tiết như sau:

- **void printRP()**: in giá trị trong bộ nhớ đệm của cơ chế thay thế . Hàm này phải in ra chuỗi "Print replacement\n" và sau đó gọi phương thức in của cơ chế thay thế tương ứng. Chi tiết cách in cho từng cơ chế thay thế được mô tả trong phần các cơ chế thay thế.
- **void printSE()**: in giá trị trong bộ nhớ đệm của công cụ tìm kiếm. Phương thức này phải in chuỗi "Print search buffer\n" và sau đó gọi phương thức in của công cụ tìm kiếm tương ứng. Chi tiết cách in cho từng công cụ tìm kiếm được mô tả trong phần tiếp theo.

Phần khai báo của lớp Cache nằm trong file main.h.

2.1.1 Công cụ tìm kiếm

Công cụ tìm kiếm được hiện thực dưới dạng một lớp trừu tượng **SearchEngine** được khai báo trong tệp Cache.h. Tuy nhiên các em được phép chỉnh sửa mọi thứ lớp này, ngoại trừ tên của nó.

Có hai lớp con cụ thể của lớp trừu tượng này đó là: AVL và DBHashing. Lớp AVL (từ bài tập lớn trước) được sử dụng để tìm kiếm trên cây AVL trong khi đó lớp DBHashing sử dụng hàm băm kép cho việc tìm kiếm. Hàm tạo của DBHashing phải có 3 tham số: tham số thứ nhất và tham số thứ hai là hai hàm băm (hash1, hash2). Còn tham số thứ ba là kích thước của bảng băm (size). Chúng được sử dụng để xác định vị trí của **key** trong bảng băm bằng hàm:

$$\textbf{hp(key,i) = (hash1(key) + i * hash2(key)) \% size}.$$

Mỗi lớp cụ thể phải có một phương thức in để in các phần tử trong bộ đệm như sau:

- **AVL**: in chuỗi "Print AVL in inorder:\n", sau đó in mọi phần tử của cây AVL theo trung thứ tự (LNR) rồi in chuỗi "In AVL in preorder:\n" và sau đó là mọi phần tử trong AVL theo tiền thứ tự (NLR).
- **DBHashing**: in chuỗi "Prime memory:\n", sau đó in tất cả các phần tử tồn tại trong bộ nhớ chính của cache theo thứ tự tăng dần của chỉ mục

2.2 Các cơ chế thay thế

Cơ chế thay thế được hiện thực bởi một lớp trừu tượng **ReplacementPolicy** được khai báo trong file Cache.h. Các em được phép chỉnh sửa mọi thứ trong lớp này, ngoại trừ tên của nó.

Có 4 lớp con cụ thể để hiện thực các cơ chế thay thế khác nhau như sau:

- **FIFO**: sử dụng cơ chế First-In-First-Out được hiện thực trong bài tập lớn trước. Khi in kết quả, lớp này sẽ in các phần tử trong cache theo thứ tự giảm theo dần thời gian tồn tại của phần tử trong cache (giống như bài tập lớn vừa rồi)
- **MRU**: sử dụng cơ chế Most-Recently-Used để chọn phần tử được sử dụng gần đây nhất bằng cách đọc hoặc ghi để xóa khi cache đầy. Để hiện thực cơ chế này, nên sử dụng danh sách tự quản (self-organizing) kết hợp với phương pháp move-to-front. Khi in kết quả, lớp này sẽ in từ phần tử đầu tiên đến phần tử cuối cùng trong danh sách tự quản.
- **LRU**: sử dụng cơ chế thay thế Least-Recently-Used để chọn phần tử được sử dụng trễ nhất. Cơ chế này cũng sử dụng phương pháp move-to-front nhưng loại bỏ phần tử cuối cùng. Khi in kết quả, lớp này sẽ in giống như lớp MRU.
- **LFU**: sử dụng cơ chế thay thế Least-Frequently-Used để chọn các phần tử ít được sử dụng nhất trong bộ nhớ cache. Một biến đếm được thêm vào mỗi phần tử trong cache để đếm số lần phần tử đó được đọc hoặc ghi. Ngoài ra, min-heap phải được sử dụng để sắp xếp lại các phần tử trong cache dựa trên giá trị của biến đếm. Khi áp dụng re-heap up (di chuyển một phần tử từ node lá lên node gốc), phần tử con sẽ hoán đổi vị trí với phần tử cha của nó khi giá trị biến đếm của phần tử con nhỏ hơn so với phần tử cha của nó. Khi áp dụng re-heap down (di chuyển một phần tử xuống node lá), phần tử cha sẽ hoán đổi với phần tử con của nó khi giá trị biến đếm của phần tử cha lớn hơn hoặc bằng so với phần tử con. Trường hợp phần tử cha có hai con, thì phần tử cha sẽ hoán đổi với phần tử con có giá trị nhỏ nhất. Nếu giá trị biến đếm của hai phần tử con bằng nhau, thì phần tử con nhỏ nhất được quy ước là phần tử con bên trái. Khi in kết quả, lớp này sẽ in các phần tử trong heap theo bật và in từ trái sang phải.

Không có tham số trong việc xây dựng các lớp này. Kích thước của bộ đệm phải là giá trị của biến MAXSIZE được khai báo trong main.h.

2.3 Trình tự thực hiện

Để hoàn thành bài tập lớn này, các em phải:

- Tải xuống tập tin initial.zip và giải nén nó
- Sau khi giải nén sẽ được 4 files: main.cpp, main.h, Cache.cpp và Cache.h. Các em KHÔNG ĐƯỢC sửa đổi các file main.cpp và main.h
- Sửa đổi các file Cache.h và Cache.cpp để hiện thực cache. Và phải giữ nguyên tên các class (ReplacementPolicy, SearchEngine, AVL, DBHashing, FIFO, MRU, LRU, LFU)

- Đảm bảo rằng chỉ có một lệnh **include** trong file Cache.h là **♯include "main.h"** và một lệnh **include** trong file Cache.cpp là **♯include "Cache.h"**. Ngoài ra, không cho phép có một include nào khác trong các file này. Nếu vi phạm yêu cầu này, bài của các em sẽ không được chấm.

3 Nộp bài

Các em chỉ nộp 2 files: Cache.h và Cache.cpp, trước thời hạn được đưa ra trong đường dẫn "Assignment 3 Submission". Có một số testcase đơn giản được sử dụng để kiểm tra bài làm của các em nhằm đảm bảo rằng kết quả của em có thể biên dịch và chạy được. Các em có thể nộp bài bao nhiêu lần tùy ý nhưng chỉ có bài nộp cuối cùng được tính điểm. Vì hệ thống không thể chịu tải khi quá nhiều em nộp bài cùng một lúc, vì vậy các em nên nộp bài càng sớm càng tốt. Các em sẽ phải tự chịu rủi ro nếu nộp bài sát hạn chót. Vi Khi quá thời hạn nộp bài, hệ thống sẽ đóng nên các em sẽ không thể nộp nữa. Bài nộp qua email sẽ không được chấp nhận.

4 Xử lý gian lận

Các em phải tự mình hoàn thành bài tập lớn này này và phải ngăn không cho người khác đánh cắp kết quả của mình. Nếu không, các em sẽ bị xử lý theo quy định của trường vì gian lận.