Bài tập lớn 2
Hiện thực Cache (phần tiếp theo)

Tháng 8/2022

1 Chuẩn đầu ra

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

- Hiện thực các thao tác với 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 tìm kiếm (BST) để tìm kiếm trên cache và dùng cơ chế LIFO, FIFO 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 thay thế cache khác nhau. Đồng thời, sử dụng các hàm bắm, giải quyết các trường hợp đụng độ trong vấn đề lưu trữ dữ liệu trên bảng băm với không gian hạn chế.

Về cách tìm kiếm, bài tập lớn này vẫn sử dụng sử dụng cây BST (giống như bài tập lớn số 1). Tuy nhiên, các em sẽ phải hiện thực thêm 4 cơ chế thay thế cache như sau:

- MFU (Most Frequently Used): dữ liệu được sử dụng nhiều nhất.
- LFU (Least Frequently Used): dữ liệu được sử dụng ít nhất.
- MFRU (Most Frequently Recently Used): dữ liệu được sử dụng nhiều nhất và gần đây nhất sẽ được chọn.
- LFRU (Least Frequently Recently Used): dữ liệu được sử dụng ít nhất và ít được dùng gần đây nhất 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. Các Interface như read, put, write của cache sẽ không thay đổi. Tuy nhiên, các phương thức khác như: print, preOrder, inOrder được thay thế bằng printRP, printSE và printLP. Ba 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 công cụ tìm kiếm được mô tả trong phần tiếp theo.

- void printLB(): Đầu tiên, 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. Sau đó in chuỗi "Hash table memory/n", và in các giá trị trong bảng bảng băm (có kích thước bằng kích thước bộ nhớ chính) theo thứ tự tăng dần chỉ mục. Biết bảng băm được tạo như sau:

  1. Lần lượt chọn các phần tử trong bộ nhớ chính của cache theo thứ tự tăng dần chỉ mục.
  2. Đối với mỗi phần tử được chọn, vị trí của phần tử trong bảng băm (chỉ mục của bảng băm) sẽ được quyết định thông qua hàm hash (lấy phần dư), cụ thể: **index = key % MAXSIZE**. Với MAXSIZE trong công thức trên là kích thước của bảng băm.
  3. Trong trường hợp đụng độ, cần tìm kiếm vị trí trống kế tiếp trong mảng bằng việc nhìn vào trong ô tiếp theo cho tới khi tìm thấy một ô trống (Linear probing).

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

2.2 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.

Chỉ có một lớp con cụ thể của lớp trừu tượng này đó là: BST. Lớp BST (từ bài tập lớn trước) được sử dụng để tìm kiếm phần tử trên cây BST.

Lớp BST cụ thể phải có một phương thức in để in các giá trị trong bộ nhớ đệm. Cụ thể: in chuỗi "Print BST in inorder:/n", sau đó in mọi phần tử của cây BST

theo trung thứ tự (LNR) rồi in chuỗi "Print BST in preorder:/n" và sau đó là mọi phần tử trong BST theo tiền thứ tự (NLR).

2.3 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:

- **MFU**: sử dụng cơ chế thay thế **Most Frequently Used** để chọn các phần tử được sử dụng nhiều nhất. Một biến đếm được thêm vào mỗi phần tử trong cache để đếm số lần phần tử đó được đọc hoặc ghi. Ngoài ra, max-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 lớn 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 nhỏ 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 lớn nhất. Nếu giá trị biến đếm của hai phần tử con bằng nhau, thì phần tử con lớn nhất được quy ước là phần tử con bên phải. Khi in kết quả, lớp này sẽ in các phần tử trong heap theo **bậc từ lớn đến nhỏ và in từ phải sang trái**.

- **LFU**: sử dụng cơ chế thay thế **Least Frequently Used** để chọn các phần tử được sử dụng ít nhất. Tương tự như MFRU, một biến đếm được thêm vào mỗi phần tử trong cache để đếm số lần phần tử đó được đọc hoặc ghi. Tuy nhiên, min-heap sẽ 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 **bậc từ nhỏ đến lớn và in từ trái sang phải**.

- **MFRU**: sử dụng cơ chế thay thế **Most Frequently Recently Used** để chọn các phần tử được sử dụng nhiều nhất và gần đây nhất trong bộ nhớ cache. Khi in kết quả, lớp này sẽ ưu tiên in các phần tử theo thứ tự phần tử được sử dụng nhiều nhất đến ít nhất. Nếu các phần tử có số lần sử

dụng giống nhau thì tiếp tục xét phần tử nào được sử dụng gần đây nhất thì sẽ được in ra trước.

- **LFRU**: sử dụng cơ chế thay thế **Least Frequently Recently Used** để chọn các phần tử được sử dụng ít nhất và ít được dùng gần đây nhất trong bộ nhớ cache. Khi in kết quả, lớp này sẽ ưu tiên in các phần tử theo thứ tự phần tử được sử dụng ít nhất đến nhiều nhất. Nếu các phần tử có số lần sử dụng giống nhau thì tiếp tục xét phần tử nào ít được sử dụng gần đây nhất thì sẽ được in ra trước.

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

2.4 Instructions

Để hoàn thành bài tập lớn này, sinh viên 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. Sinh viên KHÔNG ĐƯỢC sửa đổi các file main.cpp, main.h
- Sửa đổi các file Cache.h và Cache.cpp để hiện thực cache. Và phải GIỮ NGUYÊN tên các class (ReplacementPolicy, SearchEngine, BST, MFU, LFU, MFRU, LFRU)
- Đảm bảo rằng chỉ có một lệnh **include** trong file Cache.h là **♯include "main.h"** và một **include** trong file Cache.cpp đó là **♯include "Cache.h"**. Ngoài ra, không cho phép có một include nào khác trong các file này

3 Nộp bài

Sinh viên chỉ nộp 2 files: Cache.h và Cache.cpp, trước thời hạn được đưa ra trong site "**Cấu trúc dữ liệu và giải thuật (CO2003)_HK213_ALL**". Sinh viên 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 sinh viên nộp bài cùng một lúc, vì vậy sinh viên nên nộp bài càng sớm càng tốt. Sinh viên sẽ tự chịu rủi ro nếu nộp bài sát hạn chót. Khi quá thời hạn nộp bài, hệ thống sẽ đóng nên sinh viên sẽ không thể nộp nữa. Bài nộp qua email không được chấp nhận.

4 Harmony

Trong đề thi có thể có các câu hỏi liên quan đến nội dung bài tập lớn (phần mô tả câu hỏi sẽ được nêu rõ là dùng để harmony cho bài tập lớn, nếu có) . Điểm

của các câu hỏi harmony sẽ được scale về thang 10 và sẽ được dùng để tính lại điểm của các bài tập lớn. Cụ thể:

- Gọi x là điểm bài tập lớn.
- Gọi y là điểm của các câu hỏi harmony sau khi scale về thang 10.

Điểm cuối cùng của bài tập lớn sẽ được tính theo công thức trung bình điều hòa sau:

$$\textbf{Assignment\_Score} = \textbf{2 * x * y / (x + y)}$$

5 Gian lận

Bài làm của sinh viên sẽ được kiểm tra sự giống nhau thông qua hệ thống phát hiện gian lận MOSS. Tất cả các bài làm có **tỷ lệ giống nhau lớn hơn 30%** đều được xem là gian lận.