CS 7330

Programming Homework 1 – Linear hashing

Due Date: Nov 4th 11:59pm (with 24 hour late grace period to Nov 6th 11:59pm)

For this program, you are to implement a version of linear hashing.

Linear Hashing Class

You will implement a LinearHashing class, which provide a hash table for integers. The hash function to be used should be the one discussed in class (the rightmost k bits of the number in terms of binary).

Each bucket should start with a single page that store the numbers (the page size – as represented as number of integers. If a page for a bucket is full, extra pages will be fetch and serve as the overflow pages for that bucket.

To make thing simple for this project, you can simulate the pages (and link list) by a vector<int> class in C++ (or a list in python). However, you do have to keep track of how many numbers are there in a bucket, whether a bucket is overflowing, and how many pages are accessed during a search. For instance (e.g. if your pagesize is 3, and you have 5 integers in a current bucket, and you have to access it, then you potentially have to access 2 pages – depending on where the number you want to search is).

The class has the following methods defined:

Constructor

- LinearHashing(int pagesize, int policy, int maxoverflow, float sizelimit)
    - pagesize: the maximum number of integers that can be stored in a bucket
    - policy: denoting when should a split occurs
        - 0 (Default) : A split occurs when an insertion to a bucket and that bucket overflows [or is already overflowing] (as described by the slide)
        - 1: A split occurs only when an insertion occurs and the total number of overflow buckets is at least a certain number (to be specified in another parameter) [not the total number of pages]
        - 2: A split occurs only when an insertion occurs and the total number of integers in the hash table is equal or larger than a certain ratio of the current capacity of the hash table (to be specified by a certain parameter). Notice that the capacity includes the overflow buckets
        - 3: A split occurs when an insertion occurs and the current bucket to be split overflows por is already overflowing].
        - For option 1, 2, 3. It does NOT matter whether the bucket that is being inserted overflows or not.
    - maxoverflow (default 0): the limit of the number of overflow buckets for option 1. For other options, this value is ignored
    - sizelimit (default 1.0) : the capacity of the hash table before splitting occurs for option 2. For other options, this value is ignored.

If there is an invalid parameter (e.g. bucketsize is negative), then the system should print an error message and exit the program.

Notice that for every insertion there can only be one split. Once a split occurred, even if the condition of the split is still valid, it should wait till the next insertion before another split occurs.

Methods

- bool Insert(int x)
  - Insert x into the hash table
  - Return true is a split occurred, false if not
- int Search(int x)
  - Check if x is in the hash table
  - If x is in the hash table, return the number of pages accessed. (notice that where the number is in the bucket can determine how many pages are being accessed)
  - If x is not in the hash table, return the negation of the number of page accessed. If the bucket to be searched is empty, return 0
- void Print(ostream& os)
  - Print the hash table to the output stream. Each bucket (including the overthrow buckets) should be print on a line. The first number should be the bucket number (in binary). Also there should be a " – " symbol between overthrow buckets. You should also print the value of level and pointer (as described in the slides), for ptr, please print the value in binary. A sample output (assume each page store 3 numbers) is as follows:

    00 : 0 12 16 – 48 4
    1 : 1 9 33 – 27 15 13 - 35
    10 : 2 18
    Level: 2
    Ptr: 01

  For Python programmers, you should have two methods: the Print() method print the information to standard input, and PrintFile(fileObj) write the output to the corresponding file handle (created by the open() command).

- int Count()
  - Return the number of numbers of integers in the hash table
  - For example: for the case above, you should return 14
- vector<int> ListBucket(int x)
  - Return the numbers in bucket x (including overflow buckets), the numbers are to be stored in a vector to be returned. The ordering of the numbers does not matter.
  - For example: for the case above, ListBucket(1) should return the vector<int> with content 1, 9 , 33, 27, 15, 13, 35 (in any order)
- LinearHashingStats GetStats()
  - Return the statistical information about the LInearHashing object. (See below)

You can add extra private methods if necessary.

LinearHashingStats class

You should also create a class called LinearHasingStats that store statistics about the linear hashing object. The following information need to be stored:

- Count: The number of integers currently being stored in the hashtable
- Buckets: The number of buckets in the hash table
- Pages: The total number of pagesin the hash table
- OverflowBuckets: The total number of overflow bucekts
- Access: Number of pages (including overflow buckets) accessed since the creation of the hashtable (read and write the same page are considered two accesses).
- AccessInsertOnly: Number of buckets accessed during insertion
- SplitCount: Number of splits that has occurred

There should be a method each to return the value of that is stored: Count(), Buckets(), Pages(), OverflowBuckets(), Access(), AccessInsertOnly(), SplitCount(). The constructor should initialize all values to 0.

The LinearHashing class should have a LinearHashingStats as a member object so that it can capture the statistics.

You are welcomed to make LinearHashing a friend to the LinearHashingStats class – that will make it easier for you to update the values. (Or if you want the LInearHashingStats class variables to be public, that's fine too).

Program files

For students who write their program in C++:

- You should have a header file "lh.h" that contains declarations for the classes
- You should have a file "lh.cpp"/"lh.C" that contains the code to all the methods for the class (THERE SHOULD BE NO main() program inside that file)
- You are welcomed to write your own "lhmain.cpp"/"lhmain.C" file that contains a test program to test your code. I will provide one, and your program needs to be able to compile and run with your code.

For students who write their program in Python:

- You should have a file "lh.py" that contains declarations and implementation for both classes. Once again, no actual code to test the classes should be in that file
- You are welcomed to write your own "lhmain.py" to test the program I will also provide one for you to test.

Notice that for final evaluation I will have a different main program for testing, and not all function will be tested by the program that I provide.

Additional work (extra credit for 5330 students, partly required for 7330 students)

You need to test the performance of your linear hashing algorithm. You should conduct tests to answer the following questions:

- Which of the four policies is the best, in terms of minimum amount of access and maximum space utilization? (Notice that it may turns out different policy is the best under different situation)
- Will the performance change if the integer to be inserted has different distribution (random vs. (close to) uniform vs. skewed?

You should design and conduct experiments to test each of the options. You should write a 2-4 page report describing your testing efforts and also present and interpret your experimental results).

For 5330 students, anything here is extra credit. For 7330 student, you must at least answer the first case for randomly generated integers to be inserted.

What to hand in

You should hand in the 1 (Python) or 2 (C++) files that you use to implement the linear hashing method, together with your report (required for 7330 student, optional for 5330 student) in pdf format. You should zip those files (should be a total of 3-4 files) into a zip file and upload them to Canvas.