CSE 109 – Systems Software

Programming Lab 3

Due: 4/8/2019

Hash Sets

Background

A hash set is a data structure that provides O(1) asymptotic complexity for lookup, insert, and remove operations, in the average case. Depending on the size of the hash set, the choice of the hash function, and how collisions are handled this average case asymptotic performance can be different.

In this lab, you will implement a hash set and investigate its performance characteristics. You will begin by implementing a linked list data structure in C++. The linked list Node will hold an integer, and a pointer to the next element of the linked list:

- int data
- Node* next

The last element in the list will point to NULL. The linked list class will support the following public functions:

- bool insert_tail(int data) – insert a node at the end of the list
- bool insert_head(int data) – insert a node at the start of the list
- bool insert_at_index(int index, int data) – insert a node at the given index
- bool remove_head() – remove the head node, making the second node the head
- bool remove_tail() – remove the last node of the linked list
- bool remove_at_index(int index) – remove the node at the given index
- int find(int data) – returns the index of the node that first contains the given data. Returns -1 if the node is not found in the linked list.
- int length() – returns the number of nodes in the linked list

The functions that return bool will return false if the operation fails (e.g. there isn't enough memory to allocate a new node) and true if the operation succeeds.

The class will also have the following private members:

- int length
- Node* tail

You will also implement appropriate constructors and destructors.

Implementing a Hash Set

You will use your linked list to implement your hash set. The hash set will be an array of linked lists. When you create your hash set, you will specify how many rows will be in the set. Your hash set will have the following public methods:

- bool contains(int key) – returns true if the hash set contains the supplied key, false otherwise
- bool insert(int key) – inserts the given key into the hash set. Returns true on success, false otherwise
- bool remove(int key) – removes the given key from the hash set. Returns true on success, false otherwise

It will also have the following private members:

- LinkedList* table – this is the main data structure of the hash set, an array of linked lists.
- int size – the length of the hash set
- int hash_str(char* string) – takes a character array and returns an integer
- int hash(int prehash) – takes a pre-hashed integer and returns an index in the hash set

You will also implement appropriate constructors and destructors.

Choosing a pre-hash function

You will need to implement an appropriate hash function for your hash table. I suggest trying the following:

```
#define A 54059 /* a prime */
#define B 76963 /* another prime */
#define C 86969 /* yet another prime */
#define FIRSTH 37 /* also prime */
int hash_str(const char* s)
{
  int h = FIRSTH;
  while (*s) {
    h = (h * A) ^ (s[0] * B);
    s++;
  }
  return h; // or return h % C;
}
```

Then you will hash the output of this into an index that fits inside of your hash table.

Evaluation

You will evaluate your hash set using the supplied file of 1000 randomly generated names. You will read this list of names into your program, and store them in a linked list and a hash set. You will do the following three times for

- 10 names
- 100 names
- 1000 names

You will generate some statistics for each data structure. Record how long it takes to add every name to each data structure.

1. Record how long it takes to find each name. What is the average time? (i.e. you'll create the linked list or hash set of 1000 names, and then use the find() or contains() functions to find those names in the data structures. You'll have 1000 times. What is the average time?)
2. Do the above for differently sized hash sets. 10 – 100 – 1000.

In the end you will have 4 sets of data:

1. 10, 100, 1000 Names – Linked List
2. 10, 100, 1000 Names – Hash set size 10
3. 10, 100, 1000 Names – Hash Set size 100
4. 10, 100, 1000 Names – Hash Set size 1000

(you will need to use the time library (http://www.cplusplus.com/reference/ctime/time/)

Deliverables

Your submission will be a zip file containing a directory. The directory will have the following folders:

- lib
    - linked_list.cpp
    - linked_list.h
    - hash_set.cpp
    - hash_set.h
- src
    - main.cpp
- Makefile
- README.md

The Readme will be your project writeup. It will contain:

- Your name
- Lehigh e-mail

- Date
- Section
- Build instructions
- A lab writeup explaining your findings. You will answer the following questions:

1. Which data structure had the best performance in finding?
2. Which data structure had the best performance?
3. What happened to the hash set performance as you increased the number of rows in the hash set? Why did this happen?

Finally, include a graph in your root directory containing all the trial runs. You can save it as a PDF or an image.