Assignment: Hash Table

CS3334 Teaching Team

November 9, 2022

1 Overview

You are required to maintain a *hash table* T which supports the following operations:

Insert(x)**:** Insert the key x to T;

Delete(x)**:** Remove the key x from T if exists;

Search(x)**:** Determine if x is contained in T, and if so, return a pointer to x.

It is not allowed to use any C++ STL containers (e.g., std::vector, std::set, std::map).

2 Detailed Requirements

In this project, you are asked to implement and compare different types of hash algorithms to achieve the operations listed above. In particular, it should be organized as follows.

1. Hash function. In this part, you need to try and implement at most 4 different hash functions for numerical data, such as

   - Division method: $h(x) = x \mod M$, where x is the key value, M is the size of the hash table;
   - Mid-square method: $h(x) = x^2 \mod M$;
   - Multiplication method: $h(x) = \lfloor M(xA \mod 1) \rfloor$, where A is a constant real number, mod denotes extraction of the fractional part of xA;
   - Digital folding method: $h(x) = \sum x_i$, where x_i is the i-th digit of x.

   You may also try other hash functions, or propose your own designs. Hash functions for other data types (e.g., strings) are appreciated but not necessary.

2. Collision resolution. In this part, you need to try and implement at most 4 different collision resolutions, such as

   - Separate chaining. In this approach, the collided items are chained together to each entry of the hash table. You may use different data structures to chain those collided items, including
     - linked list;
     - self-balancing binary search tree;
     - another hash table;
   - Open addressing. In this approach, each entry of the hash table stores a key x rather than a pointer to the chained items. Probing is a common approach to achieve this, including
     - linear probing;
     - quadratic probing;
     - double hashing.

In the above probing methods, we assume to adjust the target entry of later arrived items, i.e., first-come entries will not be moved to other entries. However, we may also move those first-come items to other positions and place a newly collided item into the current entry, examples including

- Cuckoo hash;
- Robin Hood hash.

Note that you are **NOT** required to try and implement all examples mentioned above. But you need to explicitly explain what types of hash functions and collision resolutions you choose rather than simply putting the terminology there. It is recommended to discuss the pros and cons, the time complexity for three operations (average time complexity, worst-case time complexity), and other valuable things (e.g., in what kind of scenarios, we should use what kind of methods). Note that *formal* time complexity analysis for collusion resolutions is unnecessary since it requires in-depth mathematical understanding. But some intuitive or informal discussion would be appreciated. The challenging part would be designing some input data that trigger the worst-case running time of the hash algorithms.

3. Experimental Evaluations.

   - In this project, you need to design and generate the test data for your hash algorithm. The generator programs should be written in C++, and the data you generated should meet the constraints in the problem description. A sample generator is available on canvas. In particular, you may generate input data randomly or according to some customized rules.
   - You are required to evaluate and compare the differences in efficiency among different hash algorithms. Figures that intuitively show how the scale of the input data affects the running time, how different hash functions or the function parameters affect the collision rate, and empirical comparison among different methods are appreciated. Since theoretical analysis is too sophisticated for us, experiments would be of great importance.

   The compiling command for compiling your programs should be clearly stated in the report, e.g., `g++ sampleGenerator.cpp -o exec_file`. Some available compiling options are `-std=c++11`, `-lm`, `-O2`.

4. Conclusions.

You may start with further research about the hash algorithms by reading some references.

3 Due Date and Submission

- The project is due at 23:15 on Dec. 4.
- Test your solutions by submitting your codes to the Online Judge system.
- Submit a zip file containing all your materials, i.e., codes, generator programs, and study report. The codes submitted to the OJ should be included as well. For pseudo-randomly generated data, you need to indicate the seed to reproduce your work. For specially constructed data, please include those data files and indicate in your report how to use them.