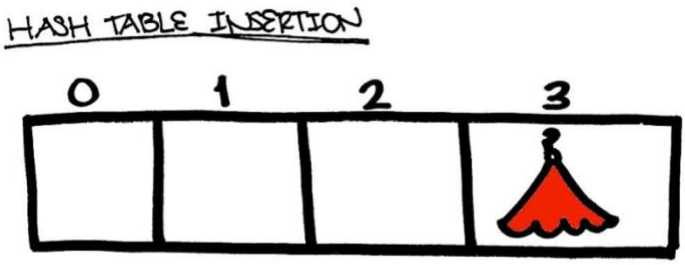
CS 202- Data Structures

Assignment-3

Hash Tables and Sorting Algorithms

Due Date: ***May 11th 2020, 11:55 pm***

This assignment can be run on any platform but please share with us the exact platform details over which the assignment was carried out and run so that the course staff can grade accordingly.



The course policy about plagiarism is as follows:

1. Students must not share actual program code with other students.
2. Students must be prepared to explain any program code they submit.
3. Students must indicate with their submission any assistance received.
4. All submissions are subject to plagiarism detection.
5. Students cannot copy code from the Internet.

Students are strongly advised that any act of plagiarism will be reported to the Disciplinary Committee.

Section 1: Hash Tables

Task 1: Hash Functions

In this task of this assignment, you are expected to implement (a) the polynomial hash code from the textbook (Section 9.2.3) and (b) bitwise
hash code function as shown below:

Let str be a string of size *n*
str = s0s1s2…sn-1
e.g., (in case of str = "Hello", 'H' is s0, 'e' is s1 … 'o' is s4)

a) **Polynomial Hash:**
Polynomial hashes can greatly minimize collisions while hashing strings. They work by taking a non-zero constant a where $a \neq 1$, and calculating:

$$(s_0a_0+s_1a_1+\ldots+s_{n-1}a_{n-1})$$

Note that by incorporating positions of characters into the calculation, permutation of the same string will result in a different hash. E.g. "stop", "tops" and "pots" will all have distinct hashes than if we were to simply accumulate corresponding Ascii/Unicode values for the characters.

Initialize polynomial_hash = 0
For every si in str
polynomial_hash += s_i * a_i

b) **Bitwise Hash:**
Initialize bitwise_hash = 0
For every si in str
bitwise_hash ^= (bitwise_hash << 5) + (bitwise_hash >> 2) + si

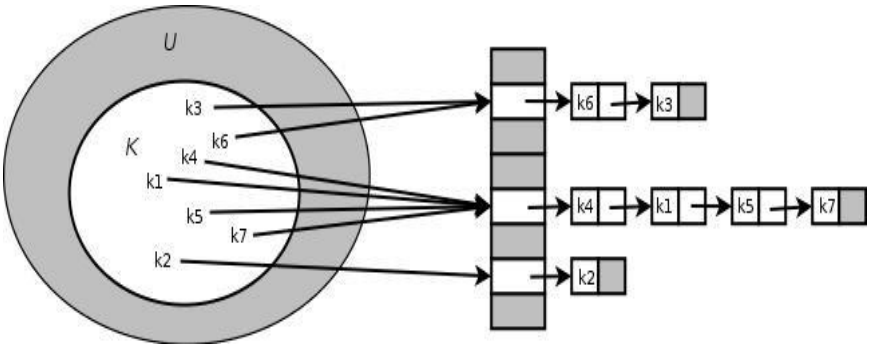
You will also need to implement the division method compression function for both (a) and (b).

The value of the parameter ***a*** in the polynomial hash code should be configurable.

Task 2(a):

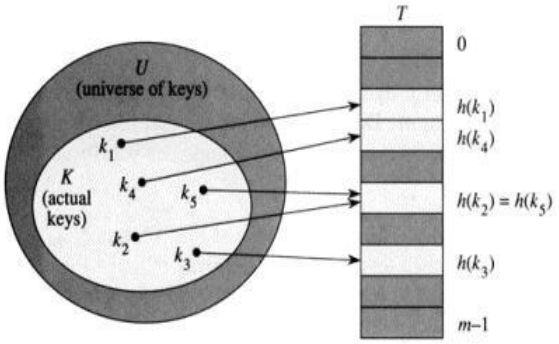
The specifics of the first hash table are as follows:

The first hash table will use chaining, where you will be required to use the LinkedList from previous assignments. This HashTable will be created with a fixed size. It should support the insert, deletion and lookup commands. The constructor should take the size of the table as a parameter. Use hash functions implemented in the Task 1.



Task 2(b):

Now you will try out the same hash function with a different hash table, which should use open addressing with linear probing. This Hash Table will initially be created with a small size; it must support resizing along with insert, deletion and lookup. Use hash functions implemented in the Task 1.



Note: In order to compile the test cases, you will be required to give the following flags:
-pthread –std=c++11
As shown in the following example:
g++ test_chaining.cpp -pthread –std=c++11

See the end of the document for more details on using pthread.

Section 2: Sorting Algorithms

This part of the assignment consists of two compulsory tasks, and one optional part which you should try and attempt. Know that there are NO bonus marks for attempting the optional part. They will just help boost your knowledge.
Tasks 3 and 4 involve the implementation of different sorting algorithms. Task 5 involve the heap data structure and heap sort algorithm. For tasks 3 and 4, we have provided some code in generator.cpp, which generates a sequence of numbers between 1 and n. Based on user input, the sequence will be random, sorted, reverse sorted or almost sorted. For each task, you have to sort this input. You will be required to sort these

numbers in arrays, as well as lists for some of the algorithms. Make sure that your algorithm works for negative values as well.

The header file sorts.h has declarations for all the sort functions. You must implement all of these functions in sorts.cpp. You may also define additional functions if it aids you in any way. But you must implement the functions already declared in sorts.h. The generator.cpp file contains functions that generate random input cases so that you can observe your implementation. You are also given a standard implementation of the Linked List data structure in the list.h and list.cpp files. You must write all your code in sorts.cpp. No other file should be altered. After writing your code in sorts.cpp, just compile generator.cpp and run. For final testing you are provided individual test cases for each sorting algorithm.

Every function that you must implement should accept an integer vector as input. The implementations must internally duplicate the vector into an array or linked list (as mandated by each task). Once the sorting is complete, the numbers must be put back into a vector in sorted order and that vector should be returned e.g., for Task 2, in InsertionSort(), you should take all elements present in the input vector, put them into an array and then apply the insertion sort algorithm. After sorting, put all the elements from the array into a vector (you can overwrite into the old vector or store in a new vector) and then return that.

Task 3: Insertion Sort (array based)

For this task, implement the in-place Insertion Sort algorithm using an array.

Task 4: Mergesort (using linked lists):

In this task, you need to implement the Merge sort algorithm using a linked list.

Optional: Task 5

Task 5a: Heap Data Structure

In this task, you are expected to implement the heap data structure. You are provided the heap.h, heap.cpp and heap_test.cpp files. You are only required to make changes to the "heap.cpp" file and you are expected to implement all the methods in that file. You can test your implementation using the heap_test.cpp file. You will be using this implementation of heap data structure in task 5b.

Task 5b: Heapsort (using heaps)

For this task, implement the array based Heapsort algorithm using the heap implementation in task 5a.

IMPORTANT NOTE:

If you are working on Windows, pthread is not supported natively. It is **highly** recommended that you use a linux based VM. However, if that is not an option you can download an opensource pthread implementation for windows here. A detailed step by step process for Visual studio (Visual C++) users is available here.

However, incase you feel you are unable to run the test cases on Windows, you are welcome to email your TAs with your file and ask them to run the test cases for you and inform you of your results. However make sure to do so well before the deadline as it may take some time for TAs to get back to you.

The following test cases require flags:

```
g++ test_insertion_sort.cpp -pthread –std=c++11
g++ test_merge_sort.cpp -pthread –std=c++11
g++ test_heap_sort.cpp -pthread –std=c++11
```

BEST OF LUCK