CptS 223 – Advanced Data Structures in C++

Written Homework Assignment 5: Hashing, Hash Tables, & Intro to Parallel Programming

I. Problem Set:

1. **(50 pts – 15 pts/table)** Starting with an empty hash table with a fixed size of 11, insert the following keys in order into three distinct hash tables (one for each collision mechanism): {12, 9, 3, 0, 42, 98, 70, 1}. You are only required to show the final result of each hash table. In the event that a collision resolution mechanism is unable to successfully resolve, simply record the state of the last successful insert and note that collision resolution failed. For each hash table type, compute the hash as follows:

```
hash(key) = (key * key + 3) % 11
'|' = collision
```

Separate Chaining (buckets)

0	1	2	3	4	5	6	7	8	9	10
	3		0	12 | 98 | 1			9 | 42	70		

To probe on a collision, start at hash(key) and add the current probe(i') offset. If that bucket is full, increment i until you find an empty bucket.

Linear Probing: probe(i') = (i + 1) % TableSize

0	1	2	3	4	5	6	7	8	9	10
	3		0	12	98	1	9	42	70	

Quadratic Probing: probe(i') = (i * i + 5) % TableSize

0	1	2	3	4	5	6	7	8	9	10
	3	98	0	12	42		9	1	70	

(5 pts) For our running hash table, you'll need to decide if you need to rehash. You just inserted a new item into the table, bringing your data count up to 53491 entries. The table's vector is currently sized at 100001 buckets. Calculate the load factor (λ):

N = 53491 elements

M = 100001

N/M = λ

53491/100001 = 0.5349

2. **(20 pts – 5 pts/blank)** What is the Big-O of these actions for a well-designed and properly loaded hash table (load factor is very low) with N elements?

Function	Big-O complexity	
	AVG case	Worst Case
Insert(x)	O(1)	O(n)
Rehash()	O(n)	O(n)
Remove(x)	O(1)	O(n)
Contains(x)	O(1)	O(n)

3. **(10 pts - 5 pts/each)** Enter a reasonable hash function to calculate a hash value for these function prototypes:

```
int hashit( int key, int tablesize )
{

       return key % tablesize;


}



int hashit( std::string key, int tablesize )
{
       char c[];
       c = key.toCharArray();

       int sum = 0;

       for(int i = 0;i < key.length();i++)
       {
            sum += c[i];
            return sum % tablesize;

       }
}
```

4. **(10 pts)** What is *parallel* programming?

Parallel computing is a method for solving large computational tasks. By splitting the tasks into smaller tasks that take place simultaneously, the larger task can be completed faster. CPU cores can process tasks simultaneously as the parallel program assigns individual CPU tasks. Parallel programming is effective when completing large tasks that can utilize a larger computer with more cores. For example, in OpenCV a computer vision library, reading images and processing them in parallel is way faster than processing each one in a serial fashion.

5. **(10 pts)** What are two strategies for *partitioning* in parallel programming?

Different strategies for parallel programming partitioning include task and data parallelism. Task parallelism partitions tasks across the CPU cores while data parallelism partitions the data used across the CPU cores. In task parallelism, large amounts of tasks are split to reduce time and increase efficiency. By utilizing all of the CPU's core count, the program can execute each task in parallel with the other tasks, resulting in a faster computation time. Data parallelism is used to manage a large set of data and spit the data load across different processes. Large stacks of data can be processed quicker in parallel this way.

II. Submitting Written Homework Assignments:

1. On your local file system, create a new directory called HW5. Move your HW5.pdf file into the directory. In your local Git repo, create a new branch called HW5. Add your HW5 directory to the branch, commit, and push to your private GitHub repo created in PA1.
2. Do not push new commits to the branch after you submit your link to Canvas otherwise it might be considered as late submission.
3. Submission: You must submit a URL link of the branch of your private GitHub repository to Canvas.

III. Grading Guidelines:

This assignment is worth 100 points. We will grade according to the following criteria:

- See above problems for individual point totals.