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
```

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

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

(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):

Λ = N/M = (53491)/(100001) = 0.53490

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
Insert(x)	Avg: O(1)
Rehash()	Avg: O(N)
Remove(x)	Avg: O(1)
Contains(x)	Avg: O(1)

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 )
{
   int hashValue = 0;
   hashValue = key%tablesize;

   if(hashValue<0)
        hashValue = hashValue + tablesize;

   return hashValue;

}
```

```
int hashit( std::string key, int tablesize )
{
   int hashValue = 0;

   for(int i = 0; i<key.length(); i++){
        hashValue = (29*hashValue)+key[i];
   }

   hashValue = hashValue%tablesize;

   if(hashValue<0)
        hashValue = hashValue + tablesize;

   return hashValue;

}
```

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

Parallel programming is where multiple CPU cores work on some problem in parallel, using partitioning to solve a task more efficiently or handle more data in less time.

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

1. Task Parallelism – This is when each CPU core is given a different set of tasks to perform.
2. Data Parallelism – This is when each CPU core handles a different set of data, but performs the same tasks on that data.

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.