Data Structures

Hashing Homework 1

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Problem #1: Small, upper and digits

- int hash_string(string str, int n)
- Change the function to handle lower letters, upper letters and digits (0-9)

Problem #2: Folding for hashing

- There are several hashing techniques such as Binning and Mid-Square
  - Feel free to google and learn about them
- In the folding technique for a string it goes as following
  - We may follow this idea
  - For every consecutive 4 letters, compute their hash value
  - Sum all blocks and return their final hash
- E.g. for input: aabcdefgAxT334gfg
  - Groups are: aabc, defg, AxT3, 34gf, g
  - Hash each one. Sum all hashes. Return sum within range

Problem 3: Key based on multiple variables

- Assume we have a class that has 3 attributes
- Hash function should return hash value based on the 3 values together, not only one of them
- Utilize the previous 2 functions

```
88 struct SomeObject {
89     const static int INTERNAL_LIMIT = 2147483647;
90     string str1, str2;
91     int number;
92
93     // Convert all 3 elements as a hash value
94     int hash() {...
99 };
```

Problem #4: Rehashing

- In our lecture code:
- Change constructor to:
  - PhoneHashTable(int table_size = 10, double limit_load_factor = 0.75)
  - Which takes a limit for load factor. Whenever we don’t satisfy it ⇒ rehashing
- void rehashing()
  - Create a new table of double size
  - Re-put the elements in the new table
  - Use new data (which has now new hash codes) and remove old ones

Problem #5: Array of linked-list

- In the lecture, we used STL to implement chaining in simple way based on vectors
- In this problem, you will implement a subset of it using array of linked-list
  - Use your own singly-linked list
  - Only implement insert and print_all
    - void put(PhoneEntry phone)
      - Update if exists or add if not
    - void print_all()
  - Don’t implement destructors (for simplicity)

Problem #5: Array of linked-list

- If you don’t know (pointers of pointers, which is simple case here)
- Use vector of linked-list: vector<LinkedHashEntry*> table { };

```
31 class PhoneHashTable {
32
33 private:
34     struct LinkedHashEntry {
35         PhoneEntry item;
36         LinkedHashEntry* next { };
37
38         LinkedHashEntry(PhoneEntry item) : item(item) {
39         }
40     };
41
42     int table_size;
43     LinkedHashEntry **table { };
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."