CS2040 Tutorial 6

Week 8, starting 3 Oct 2022

Q1 Simulation

In this question we will simulate the operations add(key) and remove(key) on a hash set, denoted by the shorthand `I(k)` and `D(k)` respectively. Note that a **hash map** works on a **<Key, Value> pair**, while in a **hash set**, the **value is the key** itself

The hash table has "table size" of 5, i.e. 5 buckets. The hash function is **h**`(key) = key % 5`
Fill the contents of the hash table after each insert / delete operation:

Use linear probing as the collision resolution technique:

	0	1	2	3	4
I(7)			7		
I(12)			7	12	
I(22)			7	12	22
D(12)			(removed)	12	
I(8)	8				

Use quadratic probing as the collision resolution technique:

	0	1	2	3	4
I(7)					
I(12)					
I(22)					
I(2)					

Use double hashing as the collision resolution technique, $\mathbf{h_2}$`(key) = key % 3`:

	0	1	2	3	4
I(7)					
I(22)					
I(12)					

Use double hashing as the collision resolution technique, $\mathbf{h_2}$`(key) = 7 - (key % 7)`:

	0	1	2	3	4
I(7)					
I(12)					
I(22)					
I(2)					

Q2 Hash Functions

A good hash function is essential for good hash table performance. A good hash function is easy/efficient to compute and attempts to evenly distribute the possible keys. Comment on the flaw (if any) of the following hash functions. Assume the load factor α = number of keys / table size = 0.3 for all the following cases:

(a) The hash table has size 100. The keys are positive even integers. The hash function is

h(*key*) = *key* % 100 not prime, odd slots are wasted

(b) The hash table has size 49. The keys are positive integers. The hash function is

h(*key*) = (*key* * 7) % 49

(c) The hash table has size 100. The keys are non-negative integers in the range of [0, 10000]. The hash function is not even, cluster at the low range

h(*key*) = $\lfloor \sqrt{key} \rfloor$ **% 100**

(d) The hash table has size 1009. The keys are valid email addresses. The hash function is

h(*key*) = (sum of ASCII values of each of the **last 10 characters) % 1009**

See http://www.asciitable.com for ASCII values

(e) The hash table has size 101. The keys are integers in the range of [0, 1000]. The hash function is

h(*key*) = $\lfloor$ ***key* * *random*** $\rfloor$ **% 101**, where **0.0 ≤ *random* ≤ 1.0**

impossible to retrieve/find??

Q3 The Price is Right

A large takeaway food chain has expanded its menu greatly during the pandemic. Their menu contains 4 categories of food: Appetizers, Soups, Mains and Desserts. Each category of food contains **N** items, each having a name and a price in cents. High-end food is sold too, hence the price can be quite large

Given a target amount **k** (in cents), find just one possible selection of an (Appetizer, Soup, Main, Dessert) that costs exactly **k** cents, if exists. What is the time complexity of a brute force algorithm that solves this problem?

Next, design an efficient O($\mathbf{N}^2$) algorithm to solve this problem

Appetizer + Soup -> put in hash table 1
Main + Desser -> put in hash table 2
for i in hashtable1: find pair that gives (k-i) in hash table 2

Question 4 (Online Discussion) – Equal Lists

You are given a **N** x **K** 2D-array of 32-bit integers. Each inner array represents a list of **K** elements. The **N** lists contain distinct sequences. You may perform some pre-processing in O(**NK**) time

After that, you are supposed to run queries. Each query is supposed to determine if a given list of another **K** 32-bit integers is equal to any of the **N** initial lists (the sequence of all **K** elements in both lists are equal). If there is a match, output the index of the match

A query should have a very high probability of running in O(**K**) time, while very rarely running in > O(**K**) time