CS2040S: Data Structures and Algorithms

Discussion Group Problems for Week 8

For: March 10–March 14

Problem 1. Hashing Basics

Problem 1.a. Try hashing items $[42, 24, 18, 36, 52, 0, 47, 45, 60, 27, 32, 7]$ with the following hash function $h(x) = x\ mod\ 7$ into a table of 7 buckets. Assume that we are using chaining to handle hash collisions.

What does each each bucket look like?

Problem 1.b. We typically use linked lists to store the items in the bucket. But what if we used a balanced binary search tree (like an AVL tree) instead?

Does anything slow down? Does anything speed up? In which cases is this favourable?

Problem 1.c. The goal of hash tables are to store $(key, value)$ pairs. Here's a question, at each bucket, is storing just the $(value)$ sufficient? Or do we need to store the entire $(key, value)$ pair? Why do you think so? For example, for a $(key, value)$ pair of $(17, 200)$. At the bucket $h(17)$, is storing (200) sufficient, or do you need to store $(17, 200)$?

Problem 2. Coupon Chaos!

Mr. Nodle has some coupons that he wishes to spend at his favourite cafe on campus, but there are different types of coupons. In particular, there are t distinct coupon types, and he can have any number of each type (including 0). He has n coupons in total.

He wishes to use one coupon a day, starting from day 1. He wishes to use his coupons in ascending order and will use up all his coupons that are of a lower type first before moving on to the next type. Nodle wishes to build a calendar that will state which coupon he will be using.

- The list of coupons will be given in an array. An example of a possible input is: $[5, 20, 5, 20, 3, 20, 3, 20]$. Here, $t = 3$, and $n = 8$. The output here would be $[3, 3, 5, 5, 20, 20, 20, 20]$.
- Since the menu at the cafe that he frequents is not very diverse, there aren't many different types of coupons. So we'll say that t is much smaller than n.

Give as efficient an algorithm as you can, to build his calendar for him.

Problem 3. Data Structure 2.0

Implement a data structure RandomizedSet with the following operations:

1. RandomizedSet() which initializes the data structure.
2. Insert(val) which inserts an item *val* into the set if not present.
3. Remove(val) which removes the item *val* from the set if present.
4. GetRandom() which returns a random element from the current set of elements. Every element must have an **equal probability** of being returned.

Insert, Remove and GetRandom must work in expected $O(1)$ time! Hint: a Hash Table might come in handy!

Assume that the maximum number of elements present in the RandomizedSet will never exceed a reasonable number n.

Problem 4. Data Structure 3.0

Let's try to improve upon the kind of data structures we've been using so far a little. Implement a data structure with the following operations:

1. Insert in $O(\log n)$ time
2. Delete in $O(\log n)$ time
3. Lookup in $O(1)$ time
4. Find successor and predecessor in $O(1)$ time

Problem 5. The Missing Element (Optional)

Let's revisit the same old problem that we've discussed at the beginning of the semester, finding missing items in the array. Given n items in no particular order, but this time possibly with duplicates, find the first missing number (if we were to start counting from 1), or output "all present" if all values 1 to n were present in the input.

For example, given $[8, 5, 3, 3, 2, 1, 5, 4, 2, 3, 3, 2, 1, 9]$, the first missing number here is 6.

Bonus: (no need for hash functions): Can we do the same thing using $O(1)$ space? i.e. in-place.