Section 04: Heaps and Hashing

Section Problems

1. Big-$\mathcal{O}$ Red-Black Trees and BSTs

Write down a tight big-$\mathcal{O}$ for each of the following. Unless otherwise noted, give a bound in the worst case.

(a) Insert and find in a BST.

(b) Insert and find in a Red-Black tree.

(c) Finding the minimum value in a Red-Black tree containing n elements.

(d) Finding the k-th largest item in a Red-Black tree containing n elements.

(e) Listing elements of a Red-Black tree in sorted order

2. Hashing

(a) Suppose we have a hash table that uses separate chaining and has an internal capacity of 12. Assume that each bucket is a linked list where new elements are added to the front of the list.

Insert the following elements in the EXACT order given using the hash function $h(x) = 4x$ (don't worry about resizing the internal array):

0, 4, 7, 1, 2, 3, 6, 11, 16

(b) Suppose we have a hash table with an initial capacity of 12. We resize the hash table by doubling the capacity. Suppose we insert integer keys into this table using the hash function $h(x) = 4x$.

Why is this choice of hash function and initial capacity suboptimal? How can we fix it?

3. Heaps

(a) Insert the following sequence of numbers into a *max heap*:

[10, 7, 15, 17, 12, 20, 6, 32]

(b) Now, insert the same values into a *min heap*.

4. Analyzing dictionaries

(a) What are the constraints on the data types you can store in a Red-Black tree? When is a Red-Black tree preferred over another dictionary implementation, such as a HashMap?

(b) When is using a BST preferred over a Red-Black tree?

5. Design

Imagine a database containing information about all trains leaving the Washington Union station on Monday. Each train is assigned a departure time, a destination, and a unique 8-digit train ID number.

What data structures you would use to solve each of the following scenarios. Depending on scenario, you may need to either (a) use multiple data structures or (b) modify the implementation of some data structure.

Justify your choice.

(a) Suppose the schedule contains 200 trains with 52 destinations. You want to easily list out the trains by destination.

(b) In the question above, trains were listed by destination. Now, trains with the same destination should further be sorted by departure time.

(c) A train station wants to create a digital kiosk. The kiosk should be able to efficiently and frequently complete look-ups by train ID number so visitors can purchase tickets or track the location of a train. The kiosk should also be able to list out all the train IDs in ascending order, for visitors who do not know their train ID.

Note that the database of trains is not updated often, so the removal and additions of new trains happen infrequently (aside from when first populating your chosen DS with trains).

Food For Thought

6. Heaps: Sorting and Reversing

(a) Suppose you have an array representation of a heap. Must the array be sorted?

(b) Suppose you have a sorted array (in increasing order). Must it be the array representation of a valid min-heap?

(c) You have an array representation of a min-heap. If you reverse the array, does it become an array representation of a max-heap?

7. Algorithm design

When writing mathematical expression, we typically write expressions in *infix* notation: in the form `NUM OPERATOR NUM`. An example of an expression written in infix notation is $4 + 6 * 5$. This expression evaluates to 34.

An alternative way we can write this expression is using *post-fix* notation: in the form `NUM NUM OPERATOR`. For example, consider the following expression written in post-fix notation:

$$4, 6, 5, *, +$$

This expression is interpreted in the following way:

- Read and store 4
- Read and store 6
- Read and store 5
- Multiply the last two stored values (and remove them from storage), then store the result
- Add the last two stored values (and remove them from storage), then store the result

The last result stored is the final "output". In this case, the expression above also evaluates to 34.

(a) Explain how you might apply or adapt the ADTs and data structures you've studied so far to evaluate an expression written in post-fix notation. Assume you accept the expression you need to evaluate as a string.

(b) Give pseudocode for this algorithm.

Challenge Problems

8. Random Hash Functions

In class we talked about various strategies to minimize collisions. In this question we discuss how to use randomness to "spread out" collisions from a small set of very bad inputs into a larger set of almost-always-fine inputs. The last two parts of this problem are beyond the scope of this course, but are interesting nonetheless.

For simplicity, assume our keyspace (the set of possible keys) is the set $\{0, 1, 2, \ldots, 2^{30} - 1\}$. Suppose we have a hashtable of size 2^{10}. Let a be an odd integer less than 2^{30}.

Consider the hash function

$$h_a(x) = \left\lfloor \frac{(ax) \mod 2^{30}}{2^{20}} \right\rfloor$$

Notice that the function changes depending on the value of a we choose, so this is really a set of possible functions.

(a) Show that for any a, h_a outputs an integer between 0 and $2^{10} - 1$ (i.e. we can use this as a hash function for our table size)

(b) Choose $a = 1$, i.e. the hash function simplifies to

$$h_1(x) = \left\lfloor \frac{x \mod 2^{30}}{2^{20}} \right\rfloor$$

For this function, find a large set of elements that all hash to 0.

(c) Let x, y be any of the two elements you found in the last part. Choose a few thousand values of a, and check whether $h_a(x) = h_a(y)$ for each of them (write code for this part). For what fraction of these hash functions do x, y collide? If the values of the hash function were totally random, how often would you expect collisions?

(d) The following statement is true (explaining why is beyond the scope of the course): For any x, y if you choose a at random, the probability that $h_a(x) = h_a(y)$ is at most $2/2^{10}$.

Use this fact, or your observations in the last part, to explain why you might decide to choose a random a instead of just choosing $a = 1$ (hint: imagine you know someone is using the hash function with $a = 1$, how can you use the first part to slow their code down? Can you do the same for a random a?)