CS 161 **Problem Set 4**

Spring 2023 **Due: Friday, May 12** at 1pm on Gradescope.

NOTE THE 2-DAY EXTENSION to give you time to recover from the exam :)

(However, HW5 will be back on our regular schedule, so manage your time accordingly!)

Style guide and expectations: Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we are looking for in homework solutions. We will grade according to these standards.

Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

Collaboration policy: You may do the HW in groups of size up to three. Please submit one HW for your whole group on Gradescope. (Note that there is an option to submit as a group). See the "Policies" section of the course website for more on the collaboration policy.

Please list all members of your homework group and all other students in the class you have collaborated with on this homework, in accordance with the Collaboration Policy, at the beginning of each homework submission.

Exercises

We recommend you do the exercises on your own before collaborating with your group.

---

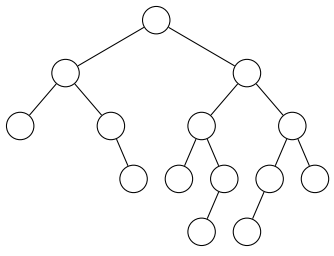
1. **(4 pt.)** [**Coloring RB Trees.**]

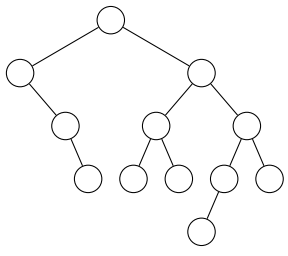
   Can you color in the nodes of the trees below to make legitimate red-black trees? For each tree, either color the nodes to make a valid red-black tree, or say that no such coloring exists.

   Note: the LaTeX code to make these trees is included in the template. If you'd like to use this code, you can color a node by editing it like:

   `\node[draw,circle,fill=red]` or `\node[draw,circle,fill=black]`.

   Or you can just color the nodes in your favorite drawing program and include the image with `\includegraphics{my_image.png}`.





   [**We are expecting:** *For each tree, either an image of a colored-in red-black tree or a statement "No such red-black tree." No justification is required.*]

2. **(2 pt.)** [**Filling in a gap from Lecture 8**] Let h be a uniformly random hash function that maps values u in a set U to the output set $\{1, ..., n\}$. Prove that if $u_i \neq u_j$, that the probability $\Pr[h(u_i) = h(u_j)] = \frac{1}{n}$.

   [**We are expecting:** *A short formal proof. It doesn't need to be long, but make sure that you explicitly use the fact that $h(u_i)$ and $h(u_j)$ are independent for $i \neq j$.*]

3. **(6 pt.)** Let $\mathcal{H}$ be a family of hash functions that map $\mathcal{U} = \{0,1\}^{100}$ down to 10 buckets.

   (a) **(2 pt.)** Suppose that there are two distinct strings, u_1 and u_2, in $\mathcal{U}$, so that for $|\mathcal{H}|/2$ of the functions $h \in \mathcal{H}$, $h(u_1) = h(u_2)$. Can $\mathcal{H}$ be a universal hash family? Must $\mathcal{H}$ be a universal hash family?[1]Why or why not?

   [**We are expecting:** *A yes/no answer to each question, and a short explanation. If you say $\mathcal{H}$ could be a universal hash family but doesn't need to be, give two examples of hash families that fit the description, one which is universal and one which is not.*]

   (b) **(2 pt.)** Suppose that for *every* pair of (distinct) strings, u_1 and u_2 in $\mathcal{U}$, there are at most $|\mathcal{H}|/10$ functions $h \in \mathcal{H}$ so that $h(u_1) = h(u_2)$. Can $\mathcal{H}$ be a universal hash family? Must $\mathcal{H}$ be a universal hash family? Why or why not?

   [**We are expecting:** *Same as previous part.*]

   (c) **(2 pt.)** Suppose that for any sequence of 10 (distinct) strings, $u_1, \ldots, u_{10}$ in $\mathcal{U}$ and any $a = 1, \ldots, 10$, there are at most $|\mathcal{H}|/10$ functions $h \in \mathcal{H}$ such that $h(u_1) = h(u_2) = \cdots = h(u_{10}) = a$. Can $\mathcal{H}$ be a universal hash family? Must $\mathcal{H}$ be a universal hash family? Why or why not?

   [**We are expecting:** *Same as previous part.*]

[1]Here, by "Can $\mathcal{H}$ be a universal hash family," we mean "is there a universal hash family $\mathcal{H}$ that meets this condition?" and by "Must $\mathcal{H}$ be a universal hash family," we mean "Is every family $\mathcal{H}$ of functions that meet this condition also a universal hash family?"

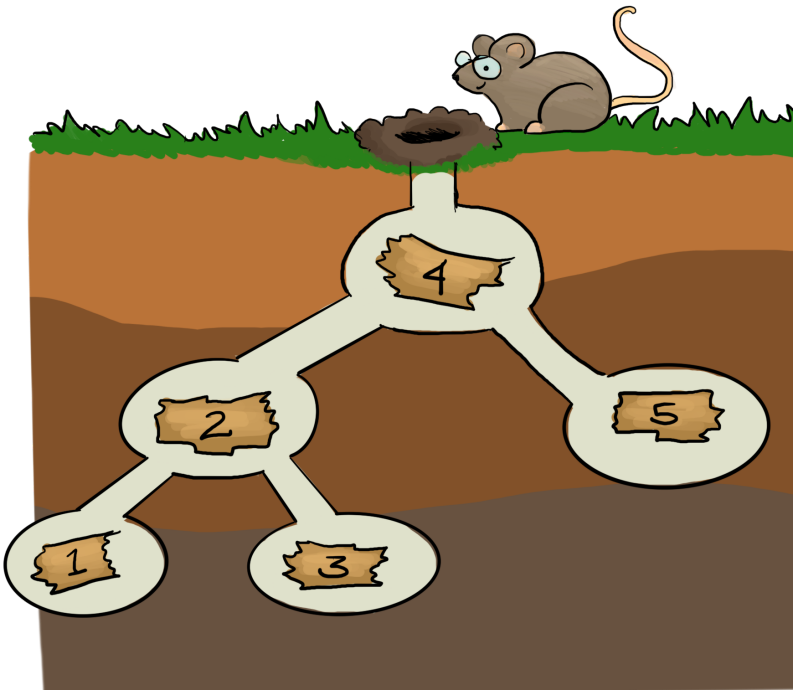
Problems

4. **(8 pt.) [Random Tunnels]**

Every spring, Vonix the Vole secures n pieces of tree bark for safekeeping (and eating). Each piece of tree bark is labeled with a number $1, \ldots, n$, in the order that it was collected. So the first piece of bark Vonix finds will be labeled "1," the second will be labeled "2," and so on.

Vonix wants to store the pieces of bark in an organized manner. They have been following the CS161 lectures, and come up with the idea of creating a balanced binary search tree! With Vonix's system, each piece of bark will have its own chamber in a series of connected underground tunnels. Vonix can then search for a specified piece of bark by traveling down the tunnel system, and going right or left at each step of the search depending on the bark's labeled value. Each piece of bark will be placed into the underground network one at a time, and Vonix will dig to create any new necessary underground chambers and paths.

However, there's one problem: Vonix can't modify the parts of the tunnel system that have already been dug, and doesn't want to move the bark pieces once they are in place. But Vonix also doesn't want the depth of the tunnel system to be too large, as then it will take too long to travel down to a specified piece of bark. The depth of the tunnel system is defined as $1 + h$ (since the root node is 1 unit deep), where h is the height of the corresponding tree. For example, the depth of the tunnel system pictured below would be 3.



(a) **(1 pt.)** Recall that Vonix's n pieces of bark are labeled in the order that they are collected, which is also the order that they are placed into the tunnel system. Also recall that Vonix doesn't move the bark pieces once they are in place. Explain why the depth of the tunnel system must then be $\Omega(n)$.

[**We are expecting:** *One or two sentences of explanation.*]

(b) **(3 pt.)** To get around the issue above, Vonix decides to give each element a *random* key. More precisely, Vonix chooses one of the $n!$ permutations $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ uniformly at random. Then Vonix inserts bark piece 1 with the key $\pi(1)$, bark piece 2 with the key $\pi(2)$, and so on.

With this system, what is the probability that Vonix's tunnel system has depth exactly n?

[**Hint:** *There are a few ways to do this problem.*

Hint for one approach: As each piece of bark is inserted, what's the probability that you are on track for a tunnel system of depth exactly n?

Hint for another approach: What does a tunnel system of depth exactly n looks like, and how many of them are there? How many total orderings are there?]

[**We are expecting:** *An exact probability, with a short explanation.*]

(c) **(3 pt.)** Vonix, who as we mentioned above is caught up on their CS161 lectures, sees a parallel between their randomized system and QuickSort (with a random pivot). What is the relationship between the tunnel system that Vonix dug in part (b) and QuickSort? Be as precise as you can.

[**We are expecting:** *A relationship like "the tunnel system that Vonix digs corresponds to [some object having to do with QuickSort]," and explain why.*]

[**Hint:** *Try picking a few permutations and build Vonix's tree using them. In particular, once you pick $\pi(1)$, what does that imply about where the pieces of tree bark 2 through n end up, relative to piece 1?*]

(d) **(1 pt.)** Based on your answer to the above, what do you think is the *expected* depth of the tunnel system that Vonix digs in part (b)?

[**We are expecting:** *An answer like "The expected depth of Vonix's tunnel system is $O(\_\_\_\_)$ and an informal explanation.*]

(e) **(0 pt.) Food for thought! Not required or even worth bonus points.** How would you prove formally that your answer from the previous part is correct?

[**We are expecting:** *Nothing, this part is not required.*]

5. **(5 pt.) [Vonix's ideal data structure?]** Vonix is pretty happy with the data structure you helped them create in Problem 4. However, their friend Mango the Magpie wants to one-up them, and proposes a new data structure (called a `mangoTree`), which is a special type of self-balancing binary search tree. Mango is intending to use the `mangoTree` to help them store shiny objects that they've found on the ground. (Each of these objects has a key, which is a measure of its shininess. The keys aren't nice things like small integers, but they are pairwise comparable: that is, Mango can tell you which of two objects is shinier.) According to Mango, a `mangoTree` has the following properties:

   - It is a binary search tree, that stores objects based on their "shininess" key.
   - It supports operations INSERT, DELETE, and SEARCH; and the BST property is maintained whenever we INSERT or DELETE something, just like a Red-Black tree.
   - It takes $O(\log n)$ time to do DELETE and SEARCH, just like a Red-Black tree; but they've improved the (worst-case) time to do INSERT to be $O(1)$.

   Mango the Magpie is excited to INSERT all of their shiny objects into their `mangoTree` very quickly. But Vonix is a bit skeptical—Mango has always been a little flighty.

   Prove that Mango the Magpie must have made a mistake, and that it is *impossible* to design a data structure with these three guarantees.

   [**Hint:** *You don't actually need to use the guarantees about* SEARCH *or* DELETE *to show that this is impossible...*]

   [**We are expecting:** *A proof. (It doesn't have to be long!)*]

6. **(4 pt.) [A Fair Exchange]**

After Mango shows Vonix their shiny object collection, Vonix would like to exchange one of their pieces of bark for one of Mango's shiny objects. Because Mango and Vonix are close friends, they would like the exchange to be fair. However, they have a difficult time agreeing on which shiny object and which piece of bark to exchange. Recall that Mango's objects are sorted by their shininess, while Vonix ranks bark in the order that they were collected.

Two things are *incommensurable* when we lack a common measure of value. Incommensurability makes it difficult to establish ranking relationships, such as "more than" or "less than", "better than" or "worse than." Using the concept of incommensurability, explain why Vonix and Mango have a difficult time agreeing on what is a fair trade.

Can you think of a real-life example where we might model things as commensurable (that is, comparable; for example, if we are going to run a sorting algorithm on them, or maximize the total value of a subset of items), but where they are actually incommensurable?

[**We are expecting:** *2-4 sentences explaining (1) why the values of Mango and Vonix's objects are incommensurable, and (2) a real-life example of something that is incommensurable that we might model as commensurable in order to solve the problem algorithmically.*]

7. **[Please Go Fill Out This Group-Work Survey] (1 pt.)**

   In collaboration with Professor Shima Salehi of the Stanford Ed School and her team, we have created a short google form on your experience with group-work in this class, which is linked below. SUNet IDs will be collected, but we will only use hashes of IDs for the study, so the responses will be effectively anonymous. The form is very short, and will take ≈ 1 minute to fill out: `https://forms.gle/iRwHGZt9mhMAEPBV6`.

   We really appreciate all your feedback and help with making CS161 the best possible learning experience it can be! If you would like to give more general feedback on the class as a whole, please check out the many possible options highlighted in the linked Ed post[2].

   Have you submitted your answers to the google form above?

   [**We are expecting:** *Yes*]

[2] `https://edstem.org/us/courses/38246/discussion/2970522`

8. **(2 BONUS pt.) [Hash tables all the way down]** *Note: this is long and we wanted to give you a break after the midterm, so it's optional! But you might want to try it to get more practice with hash tables.*

[**We are expecting:** *Nothing! This problem is optional. But we'll give one bonus point for a solid attempt at at least four out of eight of the parts, and two bonus points for a solid attempt at all of the parts.*]

Let $\mathcal{H}$ be a universal hash family that maps a universe $\mathcal{U}$ into n buckets, and suppose that for each $h \in \mathcal{H}$, $h(x)$ can be evaluated in time $O(1)$. We saw in class how to implement a data structure, `hashTable`, that stores n items from a universe $\mathcal{U}$, and supports INSERT, SEARCH and DELETE, each in *expected* time $O(1)$. Some of these had a worse *worst-case* time, but the data structure was *dynamic*, meaning that we could keep INSERTing and DELETE-ing elements as long as we wanted to.

In this exercise, we'll design a data structure, `staticHashTable`, so that:

- `staticHashTable` supports the operations BUILD and FIND.
- BUILD$(u_1, \ldots, u_n)$ takes n objects in $\mathcal{U}$, and builds a `staticHashTable` data structure containing those elements.
- FIND(u_i) returns u_i if it is in the data structure; and returns "Nope!" if it's not.

Our eventual goal will be:

- BUILD runs in *expected* time $O(n)$.
- FIND runs in *worst-case* time $O(1)$.
- `staticHashTable` is a bucket-based data structure with $O(n)$ buckets, just like a hash table.

(a) One way to design `staticHashTable` is just to use the regular `hashTable` that we saw in class. We'll BUILD the `staticHashTable` by `Insert`ing all of the items into the `hashTable`; and then we'll use `hashTable`'s SEARCH method to implement FIND. For this baseline:

i. What is the expected running time of BUILD? What is the worst-case running time?

ii. What is the expected running time of FIND? What is the worst-case running time?

iii. How much space does the data structure use?

(b) In the previous part, you should have gotten that the worst-case running time of FIND was way worse than the expected running time. (If you didn't get that, go back and try again!) We'll next try to improve on the worst-case running time of FIND by increasing the number of buckets.

Suppose that we now have a *different* hash family $\mathcal{H}'$, so that for all $h \in \mathcal{H}'$, $h : \mathcal{U} \to \{1, 2, \ldots, n^2\}$ (that is, each hash function h maps the universe into n^2 buckets instead of n). Suppose that $\mathcal{H}'$ is also a universal hash family, meaning that

$$\mathbb{P}[h(u_i) = h(u_j)] \leq n^{-2}$$

for all $u_i \neq u_j$ in the universe $\mathcal{U}$.

Fix n distinct items $u_1, \ldots, u_n \in \mathcal{U}$. For $h \in \mathcal{H}'$, let k be the number of collisions among *all* of these items. That is, k is the number of (unordered) pairs $\{i, j\}$, where $i, j = 1, \ldots, n$, so that $i \neq j$, but $h(u_i) = h(u_j)$. Show that the expected value of k is at most $1/2$, where the expectation is over a random choice of h in $\mathcal{H}'$.

[**Hint:** *Remember that the number of unordered pairs* $\{i, j\}$ *so that* $i \neq j$ *is* $\binom{n}{2} = n(n-1)/2 \leq n^2/2$.]

(c) Using the previous part, show that the probability that k is greater than 0 is at most $1/2$.

[**Hint:** *Recall that, for a non-negative random variable* X*, Markov's inequality says that* $\mathbb{P}[X \geq t] \leq \frac{\mathbb{E}[X]}{t}$.]

(d) **[Fast Find but too many buckets!]** Come up with a way to implement `staticHashTable` that uses n^2 buckets, so that BUILD takes expected time $O(n^2)$; and FIND takes *worst-case* time $O(1)$. (Assume also that all $h \in \mathcal{H}'$ can be applied to an element of $\mathcal{U}$ in time $O(1)$.)

[**Hint:** *Take inspiration from the previous part. What's likely to happen if you hash into* n^2 *buckets instead of* n*? And how can you make sure that will definitely happen?*]

(e) **[Not-so-fast Find, but $O(n)$ buckets and not too many collisions!]** Come up with a way to implement `staticHashTable` that uses n buckets, so that BUILD takes expected time $O(n)$; *and* so that k (the number of total collisions, defined above) is (in the worst-case) at most n.

[**Hint:** *Consider doing the same sort of calculation that you did in parts (b)-(d), and use the same idea to come up with your data structure.*]

(f) **[Interlude...]** Suppose you are putting n items $u_1, \ldots u_n$ into m buckets. Let k be the total number of collisions, as above. Let s_i be the number of items in bucket i. Show that (no matter what the bucketing is) $\sum_{i=1}^{m} s_i^2 = 2k + n$.

[**Hint:** *Notice that* $s_i = \sum_{j=1}^{n} \mathbf{1}[u_j$ *is in bucket* $i]$*, where the* $\mathbf{1}$ *is an indicator random variable.*]

(g) **[Finally!]** Explain how to implement `staticHashTable` that uses $O(n)$ buckets, so that BUILD takes expected time $O(n)$, and FIND takes *worst-case* time $O(1)$.

[**Hint:** *Combine (d) and (e) to get the best of both worlds! And use (f) for the analysis...*]

(h) **[Reflecting]** Does your answer from part (g) [or the ideas that went into it] also work as an implementation of `hashTable`? (That is, if we wanted to be able to INSERT and DELETE things? Why or why not?