CS2040S: Data Structures and Algorithms

Recitation 6

Goals:

- Understand the merits of hash functions on their own
- Appreciate hashing as a means to fingerprint and summarize data
- Illustrate how hashing can minimize communication complexity
- Explore problem-specific collision resolution strategies

Note: In this recitation, you may assume that hashing takes $O(1)$.

Problem 1. Drug Discovery

In the bid to find a potential cure for the COVID-19, research labs around the world are racing to study the effects different drugs have on the coronavirus. Viruses respond to drugs via *mutations*. A mutation occurs when there are changes (often small subsequences) along the original genome sequence. Suppose you now work for an international bioinformatics organization which maintains a huge open-access and canonical DNA databank (e.g. GenBank). Your organization is supporting the drug discovery effort by not only releasing the canonical sequence of the virus to labs all over the world, but also delivering it in a special format that would efficiently facilitate the comparisons between different mutations. In this way, all a research lab needs to do is to first download a local copy of the canonical unmutated sequence (in the special format), update it with the mutations from their experimental results, then quickly compare it with the mutated sequence from another lab which ran a different drug experiment. The mutational differences and similarities of the virus in response to different drugs reveal important drug properties to scientists, which in turn help them formulate more effective drugs.

In its raw format (i.e. before special formatting), a virus genome sequence is presented as a list of records where each record contains a subsequence of 60 characters with each character representing a nitrogenous base (e.g. Adenine, Guanine, Cytosine, Thymine). This is illustrated in Table 1 below.

As the chief of data in the organization, you are tasked to design the special format which would make comparisons between 2 genome sequences efficient. Having been a former student of CS2040S, you know that the "special format" necessarily entails a clever data-structure (DS) that is well-suited for the comparison operation.

Problem 1.a. Design a tree-based DS that can capture a list of n records (i.e a genome sequence) such that when given the tree for another list (i.e a mutated sequence), you can *efficiently* (read: better than $O(n)$ time) determine which are their records that differ from one another.

Record#	Subsequence
1	AAAGGTTTAT ACCTTCCCAG GTAACAAACC AACCAACTTT CGATCTCTTG TAGATCTGTT
2	CTCTAAACGA ACTTTAAAAT CTGTGTGGCT GTCACTCGGC TGCATGCTTA GTGCACTCAC
3	GCAGTATAAT TAATAACTAA TTACTGTCGT TGACAGGACA CGAGTAACTC GTCTATCTTC
4	TGCAGGCTGC TTACGGTTTC GTCCGTGTTG CAGCCGATCA TCAGCACATC TAGGTTTCGT
5	CCGGGTGTGA CCGAAAGGTA AGATGGAGAG CCTTGTCCCT GGTTTCAACG AGAAAACACA
...	... ...

Table 1: Source: Severe acute respiratory syndrome coronavirus 2 2019-nCoV/Japan/KY/V-029/2020 RNA, complete genome from GenBank

Problem 2. To Download or Not to Download

Alice has a very large collection of digital photos, consisting of several hundred gigabytes of data. In order to ensure that her photos are stored safely, she prudently maintains backup copies of all her photos on *Mazonia Storage*—a cloud-based server storage service.

Recently, some of Alice's local photos are lost because her hard drive got infected by *Claudius*, a computer virus. Worse, Claudius corrupted all of the filenames, replacing them with text from Shakespeare's *Hamlet*.

Alice's goal now is to recover her missing photos. It may be helpful to note that since Alice is meticulous in organizing and curating her collection, there exists no duplicated photos within.

Given that Alice has so many photos, it would take a very long time to download all of them from Mazonia Storage's server. The practical approach is therefore to download only the missing photos, i.e., those that were deleted by the virus.

For this problem, we shall denote the following:

- Let n be the original number of photos Alice have *remotely* (i.e. the original amount)
- Let $m < n$ be the remaining number of photos Alice have *locally*
- Let $r_1, r_2, \ldots, r_n$ be the photos on Alice's *remote* cloud server
- Let $\ell_1, \ell_2, \ldots, \ell_m$ be the photos on Alice's *local* computer

Suppose for now, the virus had only erased a single consecutive block of photos. Given that Alice's photos were stored on her local hard drive in sequence, the virus simply deleted a contiguous subsequence of photos of length $\delta = n - m$. This is illustrated as follows.

$\ell_1, \ell_2, \ldots, \ell_{j-1}$	$\ell_j, \ldots, \ell_{j+\delta-1}$	$\ell_{j+\delta}, \ldots, \ell_{n-1}, \ell_n$
	Deleted photos	

Unfortunately, Alice has no idea which photos were deleted. Moreover, Alice's internet connection is very slow, so much so that even transmitting a file containing n numbers will take too long! To

make matters worst, Mazonia charges its clients for every bit transmitted to and fro the server. Due to these reasons, Alice is forced to limit the amount of communication with the remote server as much as possible

Further suppose for now that Alice cleverly devised a *perfect hash function* h which will not produce any collisions on her original set of n photos and that this function is available on both the server and her local computer.

For the next three problems parts, your task is to help Alice design algorithms to identify the deleted photos. In addition, your solution must satisfy the following communication and space constraints:

Constraint 1: Transmit at most $O(\log n)$ hash values and a constant number of additional integer values to and fro the Mazonia server

Constraint 2: Incur only an additional $O(1)$ space

Problem 2.a. Come up with a solution in which hash values are computed over *contiguous subsequences* of photos. Explain how and why it works and provide its running time.

Problem 2.b. Come up with a solution in which hash values are computed on *individual* photos only. Explain how and why it works and provide its running time.

Problem 2.c. What if the deletions done by the virus occurred randomly throughout the photo sequence (i.e. no longer contiguous deletions). How might you modify your earlier solutions to solve for this?

In reality, perfect hash functions are rare. Suppose now that h is *not guaranteed* to be a perfect hash function. Suppose further that the virus deleted photos randomly in the photo sequence instead of in a contiguous block.

Alice proposes the following algorithm:

1. Pick *any* hash function h that maps a photograph to an integer in the range $[1, n]$
2. For each photo $\ell_i : i \in [1, m]$ on Alice's *local* computer,
   2.1. Compute its hash value $h(\ell_i)$
   2.2. Save $h(\ell_i)$ to a local file H_ℓ
3. For each photo r_i on the *remote* server:
   3.1. Compute its hash value $h(r_i)$
   3.2. Download $h(r_i)$ to Alice's local computer
   - If $h(r_i)$ is not found in H_ℓ, download photo r_i
   - Else, continue the loop

Problem 2.d. What are Alice's objectives of using a hash function in this scheme? What is the key to success in achieving those objectives?

Problem 2.e. Is H_ℓ a hash table?

Problem 2.f. Alice claims that this scheme will efficiently restore *all* the missing photos to her computer. Is she right? Explain why or why not.

Problem 2.g. What if Alice modified her solution by adding separate chaining to H_ℓ? Would this serve as an effective solution?

Alice also proposed a second scheme in which she randomly picks a hash function until she finds one that fits her criteria:

1. Let k be some integer (which may depend on n and m)
2. Repeat:
   - 2.1. *Randomly* pick a hash function h that maps a photograph to an integer in the range $[1, k]$
   - 2.2. For each photo $\ell_i : i \in [1, m]$ on Alice's *local* computer
     - 2.2.1. Compute its hash value $h(\ell_i)$
     - 2.2.2. Save $h(\ell_i)$ to a local file H_ℓ
   - 2.3. For each photo $r_i : i \in [1, n]$ on the *remote* server
     - 2.3.1. Compute its hash value $h(r_i)$
     - 2.3.2. Save $h(r_i)$ to remote file H_r
   - 2.4. Download H_r to Alice's local computer
   - 2.5. If $(|H_r| - |H_\ell|) = (n - m)$,
     - 2.5.1. Download the photos r_i whose hash value $h(r_i)$ is in H_r but not in H_ℓ
     - 2.5.2. Terminate the repeat loop
   - 2.6. Else, continue the loop to look for a better hash function

Problem 2.h. An interesting criteria for picking the random hash function is stated in Step 2.5.. Why didn't Alice simply chose the condition to be $|H_r| = n$ `&&` $|H_\ell| = m$?

Problem 2.i. If we think about H_r and H_ℓ respectively as the set of hash values *before* and *after* a set of deletion operations, what is the "invariance" in the desired hash function here?

Problem 2.j. In the second scheme, what is Alice's objective of using a hash function? (*Hint:* look at the "invariance".) Why does the criteria $(|H_r| - |H_\ell|) = (n - m)$ satisfy this objective? Show that when the loop terminates, it means Alice has correctly downloaded all the missing photos.