Ravi Dayabhai

Introduction to Algorithms: 6.006
Massachusetts Institute of Technology
Instructors: Erik Demaine, Jason Ku, and Justin Solomon

Problem Set 3

Problem Set 3

Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct.

Problem 3-1. [5 points] **Hash Practice** (SEE NEXT PAGE)

(a) [2 points] Insert integer keys `A = [47, 61, 36, 52, 56, 33, 92]` in order into a hash table of size 7 using the hash function $h(k) = (10k + 4) \bmod 7$. Each slot of the hash table stores a linked list of the keys hashing to that slot, with later insertions being appended to the end of the list. Draw a picture of the hash table after all keys have been inserted.

(b) [3 points] Suppose the hash function were instead $h(k) = ((10k + 4) \bmod c) \bmod 7$ for some positive integer c. Find the smallest value of c such that no collisions occur when inserting the keys from `A`.

Problem 3-2. [15 points] **Dorm Hashing**

MIT wants to assign $2n$ new students to n rooms, numbered 0 to $n-1$, in Pseudorandom Hall. Each MIT student will have an ID: a postive integer less than u, with $u \gg 2n$. No two students can have the same ID, but new students are allowed to choose their own IDs after the start of term.

MIT wants to find students quickly given their IDs, so will assign students to rooms by hashing their IDs to a room number. So as not to appear biased, MIT will publish a family $\mathcal{H}$ of hash functions online before the start of term (before new students choose their IDs), and then after students choose IDs, MIT will choose a rooming hash function uniformly at random from $\mathcal{H}$.

New MIT freshmen Rony Stark and Tiri Williams want to be roommates. For each hash family below, show that either:

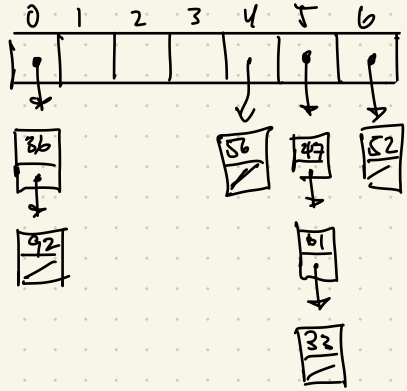
- Rony and Tiri can choose IDs k_1 and k_2 so as to guarantee that they'll be roommates, or
- prove that no such choice is possible and compute the highest probability they could possibly achieve of being roommates.

(a) [5 points] $\mathcal{H} = \{h_{ab}(k) = (ak + b) \bmod n \mid a, b \in \{0, \ldots, n-1\} \text{ and } a \neq 0\}$

(b) [5 points] $\mathcal{H} = \left\{h_a(k) = \left(\left\lfloor \frac{kn}{u} \right\rfloor + a\right) \bmod n \mid a \in \{0, \ldots, u-1\}\right\}$

(c) [5 points] $\mathcal{H} = \{h_{ab}(k) = ((ak + b) \bmod p) \bmod n \mid a, b \in \{0, \ldots, p-1\} \text{ and } a \neq 0\}$ for fixed prime $p > u$ (this is the universal hash family from Lecture 4)

① A



① B 13 ; positions elements are hashed to are

$[6, 3, 0, 4, 5, 2, 1]$

② A Yes, Rory & Tiri can choose k_1, k_2 such that they are guaranteed to be roommates: e.g., choosing $k_1 = c_1 n$, $k_2 = c_2 n$ for positive constants c_1, c_2 keeping $k_1, k_2 \in [0, \dots, u-1]$ guarantees both are put in room $b \bmod n$.

B Yes, again. Since $a \bmod n$ is an arbitrary constant once the hash function is chosen, we just need to select k_1, k_2 such that $\left\lfloor \frac{k_1 n}{u} \right\rfloor = \left\lfloor \frac{k_2 n}{u} \right\rfloor$. This can be done for known n, u by choosing k_1, k_2 such that $\frac{k_1}{u}, \frac{k_2}{u} \in \left[\frac{i}{n}, \frac{i+1}{n}\right)$ where $i \in \{0, \dots, n-1\}$.

C No, the probability of a collision over this universal family of hash functions is $\frac{1}{n}$:

$$\forall k_1 \neq k_2 \in \{0, \dots, u-1\}. \ \underset{h \in H}{P}\left(h(k_1) = h(k_2)\right) = \frac{1}{n}$$

③ ☐ The best we can do in the comparison-based model of computation is $\Theta(n \log n)$. Hence merge sort the n^2 slices.

Problem 3-3. [20 points] **The Cold is Not Bothersome Anyway**

Ice cores are long cylindrical plugs drilled out of deep glaciers, which are accumulations of snow piled on top of each other and compressed into ice. Scientists can divide an ice core into distinct slices, each representing one year of deposits. For each of the following scenerios, describe an **efficient**[1] algorithm to sort n slices collected from multiple ice cores. Justify your answers.

(a) [5 points] Every ice core is given a unique ***core identifier*** for bookkeeping, which is a string of exactly $16\lceil\log_4(\sqrt{n})\rceil$ ASCII characters.[2] Sort the slices by core identifier.

(b) [5 points] The deepest ice cores in the database are up to 800,000 years old. Sort the slices by their ***age***: the integer number of years since the slice was formed.

(c) [5 points] Variation in the amount of snowfall each year will cause a glacier to accumulate at different rates over time. Sort the slices by ***thickness***, a rational number of centimeters of the form m/n^3 between 0 and 4, where m is an integer.

(SEE NEXT PAGE)

(d) [5 points] Elna of Northendelle has discovered that water has ***memory***, but is unable to quantify the memory of a given slice. Luckily, given two slices, she can distinguish which has more memory in $O(1)$ time using her "two-finger algorithm" (touching the slices with her two index fingers). Sort the slices by memory.

Problem 3-4. [20 points] **Pushing Paper**

Farryl Dilbin is a forklift operator at Munder Difflin paper company's central warehouse. She needs to ship exactly r reams of paper to a customer. In the warehouse are n boxes of paper, each one foot in width, lined up side-by-side covering an n-foot wall. Each box contains a known positive integer number of reams, where **no two boxes contain the same number of reams**. Let $B = (b_0, \dots b_{n-1})$ be the number of reams per box, where box i located i feet from the left end of the wall contains b_i reams of paper, where $b_i \neq b_j$ for all $i \neq j$. To minimize her effort, Pharryl wants to know whether there is a ***close*** pair (b_i, b_j) of boxes, meaning that $|i - j| < n/10$, that will ***fulfill*** order r, meaning that $b_i + b_j = r$.

7 pts

(a) [10 points] Given B and r, describe an expected $O(n)$-time algorithm to determine whether B contains a close pair that fulfills order r.

7 pts

(b) [10 points] Now suppose that $r < n^2$. Describe a worst-case $O(n)$-time algorithm to determine whether B contains a close pair that fulfills order r.

(SEE PAGE AFTER NEXT)

[1] By "efficient", we mean that asymptotically faster correct algorithms will receive more points than slower ones.

[2] You may assume a string of k ASCII characters is a pointer to a contiguous sequence of k bytes in memory, where each byte stores an integer from 0 to 127 inclusive representing an ASCII character.
https://en.wikipedia.org/wiki/ASCII

③ $\boxed{A}$ Since the core identifiers are represented by $16 \cdot 8 \lceil \log_4 \sqrt{n} \rceil = 2^7 \lceil \frac{1}{4} \log_2 n \rceil$ bits, we can say that, at most, we can accomodate $2^{2^7 \lceil \frac{1}{4} \log_2 n \rceil}$ unique keys / core identifiers / bit strings (over several consecutive bytes). $2^{2^5 \log_2 n} \leq 2^{2^7 \lceil \frac{1}{4} \log_2 n \rceil} \iff n^{32} \leq 2^{2^7 \lceil \frac{1}{4} \log_2 n \rceil} < n^{33}$.

So we can think of each identifier encoded as a 33-digit base-n integer. Performing radix sort on these n keys is $O(n + 33n) = \Theta(n)$ in all cases.

$\boxed{B}$ Our keys/ages are bounded by a relatively small number: $u \leq 800{,}000$. We could use counting sort wherein the direct access array indices are from $0 - 800{,}000$ and we chain dupe keys; this is $\Theta(u + n + u) = \Theta(2u + n) = \Theta(n)$ where $u = O(n)$.

$\boxed{C}$ Noticing that $m \in \{1, 2, 3, \ldots, 4n^3\}$ we can multiply all thicknesses by n^3 to yield a keyspace $u = 4n^3 < n^4$. So, we can represent our keys as 4-digit base-n integers, perform radix sort, and divide by n^3 to recover the original keys. This takes $\Theta(n + 4n + n) = \Theta(n)$ time.

④ $\boxed{A}$ We can build a hash table H in $O(n)$ time using the number of reams in box i, b_i, as the key and associating the index i with that entry: (b_i, i). Since we are interested in $r - b_i = b_j$ and $|i-j| < \frac{n}{10}$, we can scan the boxes calculating $r-b_i$ to see if object keyed b_j is in the table (each lookup is $O(1)_e$) and if, it is, check if $|i-j| < \frac{n}{10}$ is satisfied. This would take $O(n)_e$ time.

$\boxed{B}$ If we know $r < n^2$, then we can remove all boxes b_i where $b_i \geq n^2$. Then we can perform radix sort on tuples (b_i, i) by representing b_i as a two-digit base-n integer. These two steps each occur in $O(n)$ and $O(n + n\log_2 n^2) = O(n)$ time, resp. With the tuples sorted, we can do the "two sum" pointer algorithm in $O(n)$ time. (The only modification needed is to check whether a found pair satisfying $r = b_i + b_j$ is close (i.e. $|j-i| < \frac{n}{10}$) rather than returning the pair immediately.)

Problem 3-5. [40 points] **Anagram Archaeology**

String A is an ***anagram*** of another string B if A is a permutation of the letters in B; for example, `(indicatory, dictionary)` and `(brush, shrub)` are pairs of words that are anagrams of each other. In this problem, all strings will be ASCII strings containing only the lowercase English letters `a` to `z`.

Given two strings A and B, the ***anagram substring count*** of B in A is the number of contiguous substrings of A that are anagrams of B. For example, if $A =$ `'esleastealaslatet'` and $B =$ `'tesla'`, then, of the 13 contiguous substrings in A of length $|B| = 5$, exactly 3 of them are anagrams of B, namely `('least', 'steal', 'slate')`, so the anagram substring count of B in A is 3.

8 pts

(a) [12 points] Given string A and a positive integer k, describe a data structure that can be built in $O(|A|)$ time, which will then support a single operation: given a different string B with $|B| = k$, return the anagram substring count of B in A in $O(k)$ time.

(SEE NEXT PAGE.)

(b) [3 points] Given string T and an array of n length-k strings $\mathcal{S} = (S_0, \dots, S_{n-1})$ satisfying $0 < k < |T|$, describe an $O(|T| + nk)$-time algorithm to return an array $A = (a_0, \dots, a_{n-1})$ for which a_i is the anagram substring count of S_i in T for all $i \in \{0, \dots, n-1\}$.

20 pts

(c) [25 points] Write a Python function `count_anagram_substrings(T, S)` that implements your algorithm from part (b). Note the built-in Python function `ord(c)` returns the ASCII integer corresponding to ASCII character `c` in $O(1)$ time. You can download a code template containing some test cases from the website.

Had to change how I was building hash table to be $O(|A|)$ from A time

(SEE CODE)

```python
def count_anagram_substrings(T, S):
    '''
    Input:  T | String
            S | Tuple of strings S_i of equal length k < |T|
    Output: A | Tuple of integers a_i:
              | the anagram substring count of S_i in T
    '''
    A = []
    ##################
    # YOUR CODE HERE #
    ##################
    return tuple(A)
```

5 $\boxed{B}$ Basically, perform the algorithm described in $\boxed{A}$ constructing the hash table in $O(|T|)$ on k-length sub-strings in T, then perform the $O(k)$ operation on each of n substrings $s \in S$ in $O(nk)$ time.

5 [A] The general idea is 1+2 build the data structure and 3. describes the requested operation:

1. Encode each substring as a 26-digit base-k integer (or tuple wherein the indices represent frequencies of each character) for the $|A| - k + 1$ substrings in A.
2. Build a hash table using these integers or tuples as keys. These two steps take $O(k|A|)$ time, but k is fixed constant so this is $O(|A|)$ time.

This is sus — $k = O(n)$!

3. Perform encoding described in 1. on string B in $O(k)$ time and lookup in the hash table in $O(1)_e$ time $\Rightarrow$ $O(k)_e$ time.