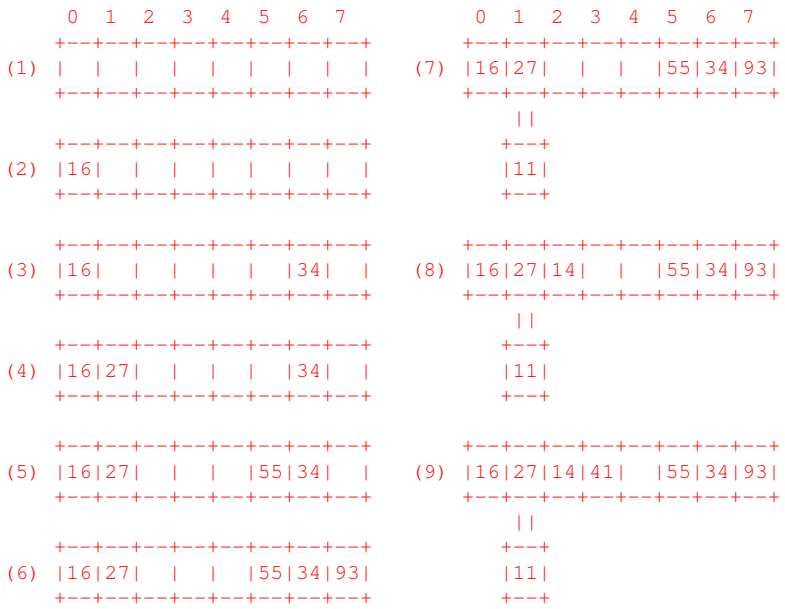
Problem Set 4

All parts are due on October 10, 2017 at 11:59PM. 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. Solutions should be submitted on the course website; there is no coding to submit for this problem set.

Problem 4-1. [30 points] **Hashing Practice**

(a) [10 points] Insert integer keys $\{16, 34, 27, 55, 93, 11, 14, 41\}$ into a hash table with eight empty slots and multiplication hash function $h(k) = (11 * k) \mod 8$, where collisions are resolved by chaining. Draw a picture of the hash table after each insertion.

Solution:



Rubric:

- 10 points for a correct sequence.
- As the algorithm for chain insertion was not specified, 11 and 27 may occur in either order, but should hash to the same location.
- -1 point for each incorrect placement of a key

(b) [10 points] Suppose you want to store n keys into a hash table of size m. Show that if the set of possible keys $[1, N]$ is sufficiently large, so that $(n-1)m < N$, then for any choice of hash function, there will always exist a set of n keys that all hash to the same location.

Solution: The average number of keys that any hash function will hash to any slot is N/m which by defintion is greater than $n-1$. Since we cannot hash a fractional number of keys into a slot, by the pigeonhole principle, some slot must have $\lceil N/m \rceil = n$ keys hashed to it.

Rubric:

- 10 points for a correct proof
- Partial credit may be awarded for a partial proof

(c) [10 points] A *ballot* is a voter's preferred ranking of c electoral candidates, represented as c digits of a base-c number. For example, for an election between 10 candidates, the ballot 5912746083 means this particular voter thinks the eighth candidate is best and the second candidate is worst. The election board wants to construct a hash table of possible ballots to find voters that have the same preference. Show why a division hash function $h(k) = k \mod m$ is a very bad choice of hash function if m is chosen to be $c-1$.

Solution: A ballot is a base-c number, i.e. $\sum_{i=0}^{c-1} \pi(i)c^i$ where $\pi : [0, c-1] \to [0, c-1]$ is a permutation on $[0, c-1]$. $c^i \mod m = c^i \mod (c-1) = (c \mod (c-1))^i = 1^i = 1$. So the hash function hashes every ballot to the same value, namely $\sum_{i=0}^{c-1} \pi(i) \mod (c-1) = \sum_{i=0}^{c-1} i \mod (c-1) = c(c-1)/2 \mod (c-1) = 0$.

Rubric:

- 10 points for a correct argument
- Partial credit may be awarded

Problem 4-2. [40 points] **Linear time**

Show how to solve each problem in either worst case or expected $O(n)$ time.

(a) [10 points] Count the number of repeated numbers in an array of n integers. Example: $\{1, 3, 4, 8, 3, 3, 4\}$ has three repeated numbers.

Solution: For each number, search a hash table for that number. If the search is successful, add one to a count of repeats. If the search is unsuccessful, insert the number into the hash table. This algorithm runs in expected $O(n)$ time because of $O(1)$ expected hash table find and insertion.

Rubric:

- 10 points for a correct algorithm
- Note that counting sort algorithms are not valid here as nothing is known about the distribution of numbers
- Partial credit may be awarded

(b) [10 points] Sort a set of n integers, where each integer x in the set is guaranteed to satisfy $6005n^3 < x \leq 6006n^3$.

Solution: Transform the set of integers by subtracting $6005n^3$ from each, yielding integers between 0 and n^3. Then we can view each as a three digit base-n number, and apply radix sort to sort in worst case $O(n)$ time.

Rubric:

- 10 points for a correct algorithm
- Partial credit may be awarded

(c) [10 points] Given a set of n integers, determine whether any pair of them sum to a given value k. Example: $\{2, 3, 4, 8, 3, 3, 4\}$ contains one pair of numbers that sum to 10, but no pair of numbers that sum to 9.

Solution: Insert each integer into a hash table. Then search in the hash table for the difference between k and each integer. If the difference is found, then you will have found two integers that sum to k. This algorithm runs in expected $O(n)$ time because of $O(1)$ expected hash table find and insertion.

Rubric:

- 10 points for a correct algorithm
- Note that counting sort algorithms are not valid here as nothing is known about the distribution of numbers
- Partial credit may be awarded

(d) [10 points] Let $\mathcal{W}$ be the set of English words. A *substitution cipher* is any bijective mapping $\mathcal{C} : \mathcal{W} \to \mathcal{W}$. To encode a message, simply replace each word w with word $\mathcal{C}(w)$. You and your friend agree on a substitution cipher to encrypt all of your correspondance. Given a long encoded message from your friend (i.e. containing $n = \Omega(|\mathcal{W}|)$ words) decode the message. Keep in mind that $|\mathcal{W}| \ll 26^s$ where s is the length of the longest English word.

Solution:

Use a hash table to store all words in $\mathcal{W}$ along with their ciphered versions, keyed by the ciphered version. Then for each word in the encrypted message, search in the hash table for the word and replace with the associated word. This algorithm runs in expected $O(n)$ time because of $O(1)$ expected hash table find and insertion.

Rubric:

- 10 points for a correct algorithm
- Partial credit may be awarded

Problem 4-3. [30 points] DNA Search

DNA can be represented as a sequence of four letters from the set $\{C, G, T, A\}$. For simplicity, assume the letters are encoded as integers, such that $C = 0, G = 1, T = 2$, and $A = 3$. Certain sequences of letters are known to be undesirable because they result in a genetic disease. For example, the presence of the sequence (C, A, T, T, A, G) might result in an organism that chases kittens.

Dr. Crancis Frick has a theory that it's not just a specific sequence of letters, but rather any permutation of a disease substring could also cause the disease. For example, (G, A, T, A, C, T) might also induce kitten chasing. Given a DNA sequence containing n letters and a potential disease marker containing d letters, help Dr. Frick evaluate if the sequence contains a sub-sequence that is a permutation of the queried disease marker in worst case (or expected) $(d + n)$ time.

(a) [10 points] **Frequencies:** One way to solve this problem is to keep track of how often characters in the alphabet have been seen. Show how to solve Dr. Frick's problem using a frequency table.

Solution: Construct a direct access array with length a, the size of your alphabet, and use it as a scratchpad to store frequencies of characters in a substring. To add a character to the frequency table, simply add one to the slot corresponding to the letter, and to remove a character subtract one. We can calculate the frequency table for a contiguous substring of length d from the one preceeding it in $O(1)$ time. To compare the scratchpad with the frequencies of the disease sequence, we first compute the frequency table of the disease sub-sequence and also the first d-character substring of the n-character string to search, each in $O(d + a)$ time. Then we compare respective frequencies of each letter in $O(a)$ time. We might have to do $O(n)$ of these comparisons, leading to a worst case $O(d + an)$ time algorithm. When a is constant (as in this problem), the runtime is linear, worst case $O(d + n)$ time.

Rubric:

- 10 points for a correct algorithm using a frequency table
- Partial credit may be awarded

(b) [15 points] **Rolling Hash:** If keys are sequences in $[1, \sigma]^r$, i.e. sequences of r characters from an alphabet of size σ, a *rolling hash* function:

$$h_p(k) = \sum_{i=0}^{r-1} k[i] \cdot \sigma^i \mod p$$

satisfies that collisions occur with probability $\Pr_p\{h_p(x) = h_p(y)\} \leq 1/r$ for any $x, y \in [1, \sigma]^r$, for a suitably random choice of a prime p. This is a useful because rolling hashes can be computed from the rolling hashes of similar keys in $O(1)$ time, and two hashes can be compared in $O(1)$ time. Show how to solve Dr. Frick's problem using a rolling hash.

Solution: The idea here is to keep track of both the frequency table, and a rolling hash of the frequency table, for any given substring, and update each of them in constant time per substring. Because a frequency table is invariant over permutations, we will hash the frequency table instead of the string directly. If the alphabet has a DNA letters, the frequency table will have a slots associated with the letters, with each slot holding the number of occurrences (between 0 and d) of the letter associated with the slot. So $k \in [0, d]^a$ represents the frequency table we want to hash with a rolling hash, with $\sigma = d + 1$ and $r = a$. Computing the rolling hash of both the frequency table of the disease sub-sequence and the first d-character sub-sequence of the n-character string to search, can each be computed in $O(d)$ time if we precompute the values of $(d + 1)^i$ modulo p for $i \in [-1, a]$ in $O(a)$ time. However, updating a hash of the frequency table (and the frequency table itself) for each successive d-character sub-sequence can be done in $O(1)$ time using the rolling hash by adding or subtracting an appropriate power of $(d + 1)$ modulo p, which we precomputed. Comparing this hash to the hash of the disease sub-sequence can be done in $O(1)$ time by assumption. Of course if the hashes match, there may have been a collision, and we must spend $O(a)$ time to confirm that the actual frequency tables match. But because collisions occur with frequency at most $1/a$, the expected number of times we have to check false positives is $O(n/a)$, all of which will take $O(n)$ time to check in expectation. Thus the total running time will be expected $O(a + d + n)$. When a is constant (as in this problem), the runtime is linear, expected $O(d + n)$ time.

Rubric:

- 15 points for a correct algorithm using a rolling hash
- Partial credit may be awarded

(c) [5 points] **Compare:** Suppose we wanted to do the same analysis on Martian DNA, containing many more types of letters. If the a letters of the Martian alphabet are encoded by the integers $[1, a]$, how would the running time of the preceding two algorithms change with respect to a?

Solution: When a is not small, the difference between $O(d + an)$ and $O(a + d + n)$ is substantial (quadratic instead of linear), so using a rolling hash is preferable.

Rubric:

- 5 points for a reasonable comparison based on previous parts
- Do not mark off again for incorrect analysis on previous parts