Problems 1 and 2 require you to have read the lecture notes or watched the lecture, but should be doable. Problem 3 can be attempted before the lecture, but it's alright to skip it.

Problem 4 requires a couple non-trivial ideas, but is interesting both conceptually and algorithmically. Problems 5 and 6 are quite important: make sure you attempt both (especially Problem 6), and read the solutions.

Problem 7 is worth doing if you have time, but not crucial.

# Warm-up

**Problem 1.** Check your understanding: summarise the key differences between a hash table and a Bloom filter, in terms of time and space complexity and guarantees provided.

**Problem 2.** Prove the claim made in class: the expected time complexities of INSERT, LOOKUP, and REMOVE with separate chaining are all $O(1 + \alpha)$, where $\alpha = n/m'$ is the load of the hash table. What is their *worst-case* time complexity?

# Problem solving

**Problem 3.** Give an example of a universal hash family $\mathcal{H}$ from a universe $\mathcal{X}$ to a set $\mathcal{Y}$ for which the inequality is not always an equality:

$$\Pr_{h \sim \mathcal{H}} \left[ h(x) = h(x') \right] \leq \frac{1}{|\mathcal{Y}|} \qquad \text{for all distinct } x, x' \in \mathcal{X}$$

**Problem 4.** Given three arrays $A$, $B$, and $C$ each containing $n$ positive integers, the task is to decide if there exist $1 \leq i, j, k \leq n$ such that $A[i] + B[j] = C[k]$. We aim for an algorithm running in (expected) time $O(n^2)$. *(We assume that, given a suitable hash function, we can evaluate it on any given input in constant time.)*

a) As a warm-up, describe an $O(n^3)$-time deterministic algorithm.

b) Describe an efficient $O(n^2)$ (expected) time algorithm.

c) Prove its correctness, and expected time complexity.

d) Analyze its worst-case time complexity. Can you get $O(n^2)$ here as well?

**Problem 5.** (Perfect Hashing) Consider the following *two-level hashing* strategy: as in separate chaining, we will use a hash table $A$ of size $m' = O(n)$ to contain our $n$ items, and deal with collisions by having each of the $m'$ buckets handle its hashed elements on its own. But instead of having a linked list for each bucket, we will instead use a secondary *hash table* for each bucket. Here we focus on the case where all $n$ elements are inserted at once at the beginning, and we want to focus on the lookups.

  a) Suppose that bucket $k$ has $n_k$ of the $n$ elements hashed to it. What should be the size of the hash table $A_k$ (the hash table in in bucket $k$) to guarantee it only has a collision with probability $1/2$?

  b) Briefly describe how to do the batch insertion of all $n$ elements (initialisation of the data structure).

  c) Analyse the expected time complexity of a lookup to your hash table.

  d) Analyse the expected space complexity of the overall data structure, and show it is $O(n)$.

**Problem 6.** We will analyse the error probability of the Bloom filter seen in class. We will focus on the error rate, that is, how frequently we would expect LOOKUP to make a mistake, "on average." In what follows, assume we inserted a dataset $S$ of $n$ elements into the Bloom filter. We will make the following (false, but convenient) assumption that we have truly random hash functions: the $(h_i(x))_{i,x}$ are fully independent across elements $x \in \mathcal{X}$ and hash functions $1 \le i \le k$, and $h_i(x)$ is uniformly distributed in $\{1, 2, \ldots, m'\}$ for every $i$ and every $x$:

$$\forall i, x, y, \quad \Pr[h_i(x) = y] = \frac{1}{m'}$$

  a) Fix any $1 \le i \le m'$. After inserting $n$ elements into our Bloom filter, what is the probability $p_i$ that the $i$-th bit of our array $A$ is set to 1?
  Let $B := \frac{m'}{n}$ be the average number of extra bits used per element. Using the approximation $1 + x \approx e^x$ (very accurate for small $x$), show that $p_i \approx 1 - e^{-k/B}$.

  b) *Error rate:* What is the probability that, when calling LOOKUP($x$) on a key which was *not* inserted (not part of the $n$ keys from $S$), the value returned is yes?

  c) Say you have a target per-element storage value $B$ in mind: $B = 8$ bits. What is the number of hash functions $k$ you should use to minimise the probability of error?

  d) For the setting $B = 8$, and the choice of $k$ above, what is the error rate you should expect?

  e) Let's use $k = 6$ hash functions and explore the trade-off between space (parameter $B$) and error rate – we could decide to use more space than 8 bits per element. What is the expected error rate if you increase $B$ to 12 bits? 16? 32?

| **Advanced** |
| --- |

**Problem 7.** Augment the Bloom filter data structure seen in class to add a REMOVE operation. Analyse the resulting guarantees (performance, error probability, space and time complexities).