ECE 406, Fall 2020, Assignment 2

1. Consider the following set of functions $\mathscr{F}$. Given a finite universe of keys U, and a number of buckets, $m \in \mathbb{N}$, let $\mathscr{F}$ be the set of all functions with domain U and codomain $\{0, 1, \ldots, m-1\}$. E.g., if $U = \{a, b\}, m = 3$, then $\mathscr{F} = \{f_1, \ldots, f_9\}$, where:

$$\begin{array}{lll} f_1(a) = 0, f_1(b) = 0 & f_2(a) = 0, f_2(b) = 1 & f_3(a) = 0, f_3(b) = 2 \\ f_4(a) = 1, f_4(b) = 0 & f_5(a) = 1, f_5(b) = 1 & f_6(a) = 1, f_6(b) = 2 \\ f_7(a) = 2, f_7(b) = 0 & f_8(a) = 2, f_8(b) = 1 & f_9(a) = 2, f_9(b) = 2 \end{array}$$

   (a) Prove that such a set $\mathscr{F}$ is universal. That is, for any two distinct keys $k, l \in U$, the number of functions in $\mathscr{F}$ for which k and l collide is $\leq |\mathscr{F}|/m$.

   In the example above, f_1, f_5, f_9, are the functions for which the two keys in U collide. There are $3 \leq 9/3$ of them only.

   (b) Would you say that this is a practical way of generating a universal set from which you can then pick a hash function uniformly at random? Why or why not?

   Hint: what are some good things from the standpoint of practicality about the approach in your textbook and Lecture 3(a)? Could the approach here also have those good things?

2. Consider the problem, given inputs (i) an array $A[1, \ldots, n]$ of distinct integers, and (ii) $r \in \{1, \ldots, n\}$, identify the item in $A[1, \ldots, n]$ of rank r. Assume also we use the divide-n-conquer strategy from Lecture 3(c) — we choose a pivot v, we split (or partition) around v, check whether v is our solution, and if not, recurse on the right or left piece as appropriate.

   Suppose you have access to an oracle, call it GetPivot, that, if invoked with such an array $A[1, \ldots, n]$, returns a pivot v as follows.

   - If $n < 5$, the behaviour of GetPivot is undefined.
   - If $n \geq 5$, GetPivot does the following:
     - It perceives $A[1, \ldots, n]$ as comprised of sequences of 5 items each: $A[1, \ldots, 5]$, $A[6, \ldots, 10]$, …, $A[n-4, \ldots, n]$. (For simplicity, assume n is divisible by 5.)
     - It computes the median of each $A[i, \ldots, i+4]$. Call these medians $m_1, m_2, \ldots, m_{n/5}$.
     - It then computes the median, m, of those medians. That is, m is the median of $\left[m_1, m_2, \ldots, m_{n/5}\right]$.
     - It returns m as the pivot v.

   Show that if $n \geq 5$, we use this oracle GetPivot to identify the pivot around which to split, and the pivot it returns is not our answer and we recurse, then whatever piece we recurse on has size at most $\lceil 7/10 \cdot n \rceil$. If it makes things easier for you, you're allowed to assume that n is a multiple of 10, and therefore of 5.

   (Not related to what you need to do, but a consequence of realizing such a subroutine prudently is that the recurrence for the worst-case time-efficiency is $T(9n/10) + \Theta(n)$, which has solution $T(n) = \Theta(n)$. That is, we would have a linear-time deterministic algorithm for the

selection problem from Lecture 3(c). And yes, this is a real algorithm which someone devised. It is simple and ingenious.)

Hint: we recurse either with all the items that are $< m$, or $> m$ — ignore the one piece whose median is m itself. Group the $n/5$ 5-item pieces into two groups: G_{lo} comprises ones whose median of the 5 items is $< m$, and G_{hi} comprises ones whose median is $> m$. We have $\approx 1/2 \times n/5$ pieces in each of G_{lo} and G_{hi}. Now ask: given $A[i, \ldots, i+4] \in G_{lo}$, how many of $A[i], \ldots, A[i+4]$ can possibly be $< m$? Similarly, given $A[j, \ldots, j+4] \in G_{hi}$, how many of $A[j], \ldots, A[j+4]$ can possibly be $< m$? This should lead you to an upper-bound, as a fraction of n, on the maximum number of items in $A[1, \ldots, n]$ that can be $< m$. Similarly, work out an upper-bound, as a fraction of n, on the maximum number of items in A that can be $> m$.

3. You plan to drive from city A to city B and want to get to your destination with as few stops as possible under the following two constraints:

   (a) You will drive by day only, and rest by night.

   (b) There is a sequence of potential stops along the way each of which is acceptable to you, and every stop you make to rest at night has to be at one of those stops.

   Prove that this problem possesses the following greedy choice property:

   > As you arrive at each stop, ask whether you can make it to the next stop before nightfall. If yes, continue driving today. If not, this is your stop for the night.

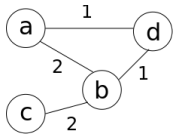
   Hint: suppose $O = \langle o_1, \ldots, o_k \rangle$ is an optimum, i.e., of minimum size, sequence of stops. Suppose $G = \langle g_1, \ldots, g_l \rangle$ is the sequence of stops made by making the above greedy choice. We know $l \geq k$. We seek to prove: $l = k$. Prove first, by induction on k, that any stop g_i must be at least as close to the destination B (or equivalently, at least as far from the source A) as the corresponding o_i. Once you prove that, prove that g_{k+1} cannot exist.

4. Given as inputs (i) a non-empty weighted undirected graph $G = \langle V, E, l \rangle$ where $l : E \to \mathbb{Z}^+$, i.e., edge-weights are all positive integers, and, (ii) two distinct vertices $a, b \in V$, we want to find the number of distinct shortest paths $a \rightsquigarrow b$.

   (a) Devise a polynomial-time algorithm for the problem and describe your algorithm briefly and precisely in prose in a manner that a TA can understand how your algorithm works. Also give a brief and crisp argument for correctness, and a brief and crisp analysis of the worst-case running-time as a function of the size of the input, n in $O(\cdot)$ notation. (So no need for comprehensive proofs.)

   (b) **[python3]** Code your algorithm in python3. See Learn for a skeleton `a2p4b.py` file, and a tester file. We encode G as a list of adjacency lists of pairs. Each pair is a $\langle$neighbour, edge-weight$\rangle$. Following is an example graph, and its encoding. We have

mapped: $a \mapsto 0, b \mapsto 1, c \mapsto 2, d \mapsto 3$.



`[[[1, 2],[3, 1]], [[0, 2],[3,1],[2,2]], [[1,2]], [[1, 1],[0, 1]]]`

If we are given the above graph and the vertices a, b as input, i.e., `[0,1]` in python3, then the correct answer is 2 because there are two distinct shortest paths $a \rightsquigarrow b$: $\langle a, b\rangle$ and $\langle a, d, b\rangle$. If, instead, our input vertices are a, d, the correct answer is 1; the only shortest path is $\langle a, d\rangle$.

5. We have a building of n floors, and a set of k identical eggs. We want to identify the highest floor from which if we drop an egg, the egg does not break. (Presumably, if the egg breaks when dropped from a certain floor, then it breaks from all floors above that floor as well. Similarly, if the egg does not break when dropped from a certain floor, then it does not break when dropped from any lower floor either.)

   We are forced to do some kind of trial-n-error. For example, we could start at the lowest floor and drop an egg. If it does not break, the next higher floor, and so on. This would result, in the worst-case, in n drops of an egg, but we need one egg only. Or we could do binary search, i.e., drop an egg from the middle floor, and then search lower or higher. We converge in $\log n$ drops, but it would cost us $\log n$ eggs in the worst-case.

   We want to engage in as few egg-drops as possible in the worst-case, but we are constrained to k eggs only. Devise a divide-n-conquer algorithm that asymptotically converges faster the more eggs we have. That is, suppose the worst-case number of drops of an egg you need for a building of n floors given k eggs is $d(k, n)$. Your algorithm should have the property: for all $k \in \mathbb{N}$, $\lim_{n\to\infty} d(k+1, n)/d(k, n) = 0$.

   Hint: suppose the building has 81 *floors and we want to compare the situation that we have* 3 *eggs, vs. the situation that we have* 4 *eggs. When we have* 4 *eggs, with your first egg, narrow down to a sequence of* 27 *floors by perceiving the building as comprised of* 3 *sequences of* 27 *floors each. Then, with your second egg, narrow down to a sequence of* 9 *floors. Then to a sequence of* 3 *floors with your third egg, and then linear-search on those* 3 *floors with your last egg. For the case that you have* 3 *eggs only, first perceive the building as comprised of* $(81)^{1/3}$ *sequences, each of* $(81)^{2/3}$ *floors. Then* $(81)^{1/3}$ *sequences each of* $(81)^{1/3}$ *floors, and then finally linear-search on* $(81)^{1/3}$ *floors.*