Exercises

Please do the exercises on your own.

1. **(2 pt.)** Suppose that $h : \mathcal{U} \to \{0, \ldots, n-1\}$ is a uniformly random function. That is, for each $x \in \mathcal{U}$, $h(x)$ is distributed uniformly at random in the set $\{0, \ldots, n-1\}$, and the values $\{h(x) : x \in \mathcal{U}\}$ are independent. Prove that for any $x \neq y \in \mathcal{U}$,

$$\mathbb{P}_h\{h(x) = h(y)\} = \frac{1}{n}.$$

   Above, notice that x and y are fixed and the probability is over the choice of h.

   [**We are expecting:** *A short but rigorous proof.*]

2. **(4 pt.)** Let $\mathcal{U} = \{000, 001, 002, \ldots, 999\}$ (aka, all of the numbers between 0 and 999, padded so that they are three digits long) and let $n = 10$. For each of the following hash families $\mathcal{H}$ consisting of functions $h : \mathcal{U} \to \{0, \ldots, n-1\}$, decide whether $\mathcal{H}$ is universal or not, and justify your result with a formal proof.

   (a) **(2 pt.)** For $i = 1, 2, 3$, let $h_i(x)$ be the i'th least-significant digit of x. (For example, $h_2(456) = 5$). Define $\mathcal{H} = \{h_1, h_2, h_3\}$. Is $\mathcal{H}$ a universal hash family?

   (b) **(2 pt.)** For $a \in \{1, \ldots, 9\}$, let $h_a(x)$ be the least-significant digit of ax. (For example, $h_2(123)$ is the least-significant digit of $2 \times 123 = 246$, which is 6). Define $\mathcal{H} = \{h_i : i = 1, \ldots, 9\}$. Is $\mathcal{H}$ a universal hash family?

   [**HINT:** *To show that something is* not *a universal hash family, you could find two distinct elements $x, y \in \mathcal{U}$ so that when $h \in \mathcal{H}$ is chosen randomly, the probability that $h(x) = h(y)$ is larger than it's supposed to be.*]

   [**We are expecting:** *For each part, a yes/no answer and a rigorous proof using the definition of a universal hash family.*]

3. **(4 pt.)** Give one example of a *connected* undirected graph on four vertices, A, B, C, and D, so that both depth-first search and breadth-first search discover the vertices in the same order when started at A. Give one example of a *connected* undirected graph where BFS and DFS discover the vertices in a different order when started at A.

   Above, *discover* means the time that the algorithm first reaches the vertex. Assume that both DFS and BFS iterate over neighbors in alphabetical order.

   Note on drawing graphs: You might try `http://madebyevan.com/fsm/` which allows you to draw graphs with your mouse and convert it into LaTeX code. (By default it makes directed graphs; you can add an arrow in both directions to get something that approximates an undirected graph).

   [**We are expecting:** *A drawing of your two graphs and an ordered list of vertices discovered by BFS and DFS for each of them.*]

Problems

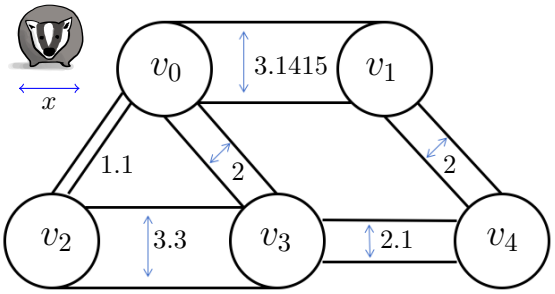
You may talk with your fellow CS161-ers about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

---

4. **(6 pt.) [Badger badger badger.]** A family of badgers lives in a network of tunnels; the network is modeled by an undirected connected graph G with n vertices and m edges (see below). Each of the tunnels have different widths, and a badger of width x can only pass through tunnels of width $\geq x$.

   For example, in the graph below, a badger with width $x = 2$ could get from v_0 to v_4 (either by $v_0 \to v_1 \to v_4$ or by $v_0 \to v_3 \to v_4$). However, a badger of width 3 could not get from v_0 to v_4.



   The graph is stored in the adjacency-list format we discussed in class. More precisely, G has vertices $v_0, \ldots, v_{n-1}$ and is stored as an array V of length n, so that $V[i]$ is a pointer to the head of a linked list N_i which stores integers. An integer $j \in \{0, \ldots, n-1\}$ is in N_i if and only if there is an edge between the vertices v_i and v_j in G.

   You have access to a function `tunnelWidth` which runs in time $O(1)$ so that if $\{v_i, v_j\}$ is an edge in G, then `tunnelWidth(i,j)` returns the width of the tunnel between v_i and v_j. (Notice that `tunnelWidth(i,j)`=`tunnelWidth(j,i)`). If $\{v_i, v_j\}$ is not an edge in G, then you have no guarantee about what `tunnelWidth(i,j)` returns.

[Actual questions on next page.]

(a) **(3 pt.)** Design a deterministic algorithm which takes as input G in the format above, integers $s, t \in \{0, \ldots, n-1\}$, and a desired badger width $x > 0$; the algorithm should return **True** if there is a path from v_s to v_t that a badger of width x could fit through, or **False** if no such path exists.

(For example, in the example above, if $s = 0$ and $t = 4$ then your algorithm should return **True** if $0 < x \leq 2$ and **False** if $x > 2$).

Your algorithm should run in time $O(n + m)$. You may use any algorithm we have seen in class as a subroutine.

Note: In your pseudocode, make sure you use the adjacency-list format for G described above. For example, your pseudocode should *not* say something like "iterate over all edges in the graph." Instead it should more explicitly show how to do that with the format described. (We will not be so pedantic about this in the future, but one point of this problem is to make sure you understand how the adjacency-list format works).

[**We are expecting:** *Pseudocode* ***AND*** *an English description of your algorithm, and a short justification of the running time. You should make sure to use the adjacency-list representation of G described above in your pseudocode. You can use any algorithms we have seen from class as a subroutine, but if you significantly modify them make sure to be precise about how this interacts with the adjacency-list representation.*]

(b) **(3 pt.)** Design a deterministic algorithm which takes as input G in the format above and integers $s, t \in \{0, \ldots, n-1\}$; the algorithm should return the largest value x so that there exists a path from v_s to v_t which accomodates a badger of width x. Notice that since G is connected, there is a path from v_s to v_t in G. Your algorithm should run in time $O((n + m)\log(m))$. You may use any algorithm we have seen in class as a subroutine.

[**HINT:** *Use part (a).*]

Note: Don't assume that you know anything about the tunnel widths ahead of time. (e.g., they are not necessarily bounded integers).

Note: The same note about pseudocode holds as in part (a).

[**We are expecting:** *Pseudocode* ***AND*** *and English description of your algorithm, and a short justification of the running time. You should make sure to use the adjacency-list representation of G described above in your pseudocode. You can use any algorithms we have seen from class as a subroutine, but if you significantly modify them make sure to be precise about how this interacts with the adjacency-list representation.*]

5. **(8 pt.)** **[Painted Penguins.]** A large flock of T painted penguins will be waddling past the Stanford campus next week as part of their annual migration from Monterey Bay Aquarium to the Sausalito Cetacean Institute. Painted Penguins (not to be confused with pedantic penguins) are an interesting species. They can come in a huge number of colors—say, M colors—but each flock of T penguins only has m colors represented, where $m < T$. The penguins will waddle by one at a time, and after they have waddled by they won't come back again.

   For example, if $T = 7$, $M = 100000$ and $m = 3$, then a flock of T painted penguins might look like:

   `seabreeze, seabreeze, indigo, ultraviolet, indigo, ultraviolet, seabreeze`

   You'll see this sequence in order, and only once. After the penguins have gone, you'll be asked questions like "How many `indigo` penguins were there?" (Answer: 2), or "How many `neon orange` penguins were there?" (Answer: 0).

   You know m, M and T in advance, and you have access to a universal hash family $\mathcal{H}$, so that each function $h \in \mathcal{H}$ maps the set of M colors into the set $\{0, \ldots, n-1\}$, for some integer n. For example, one function $h \in \mathcal{H}$ might have $h(\texttt{seabreeze}) = 5$.

   (a) **(5 pt.)** Suppose that $n = 10m$. Suppose also that you only have space to store:
   - An array B of length n, which stores numbers in the set $\{0, \ldots, T\}$, and
   - one function h from $\mathcal{H}$.

   Use the universal hash family $\mathcal{H}$ to create a randomized data structure that fits in this space and that supports the following operations in time $O(1)$ in the worst case (assuming that you can evaluate $h \in \mathcal{H}$ in time $O(1)$):
   - `Update(color)`: Update the data structure when you see a penguin with color `color`.
   - `Query(color)`: Return the number of penguins of color `color` that you have seen so far. For each query, your query should be correct with probability at least 9/10. That is, for all colors `color`,

$$\mathbb{P}\{\texttt{Query}(\texttt{color}) = \text{ the true number of penguins with color } \texttt{color}\ \} \geq \frac{9}{10}.$$

   To describe your data structure:
   i. Describe how the array B and the function h are initialized.
   ii. Give pseudocode for `Query`.
   iii. Give pseudocode for `Update`.

   [**We are expecting:** *A description following the outline above (including pseudocode), and a short but rigorous proof that your data structure meets the requirements. Make sure you clearly indicate where you are using the property of universal hash families.*]

   (b) **(3 pt.)** Suppose that you now have k times the space you had in part (a). That is, you can store k arrays $B_1, \ldots, B_k$ and k functions $h_1, \ldots, h_k$ from $\mathcal{H}$. Adapt your data structure from part (a) so that all operations run in time $O(k)$, and the `Query` operation is correct with probability at least $1 - \frac{1}{10^k}$.

   [**We are expecting:** *As in part (a), a description following the outline above (except say how all arrays B_i and functions h_i are initialized), and a short but rigorous proof that your data structure meets the requirements. Make sure you clearly indicate where you are using the property of universal hash families.*]