**CMPS 256** **Problem Set 5**

**Style guide and expectations:** Please see the "Homework" part of the syllabus for guidance on what we are looking for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material.

**What we expect:** Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem!

---

# Exercises

We suggest you do these on your own. As with any homework problem, though, you may ask the course staff for help.

---

1. **[OPTIONAL]** 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. **[OPTIONAL]** 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.

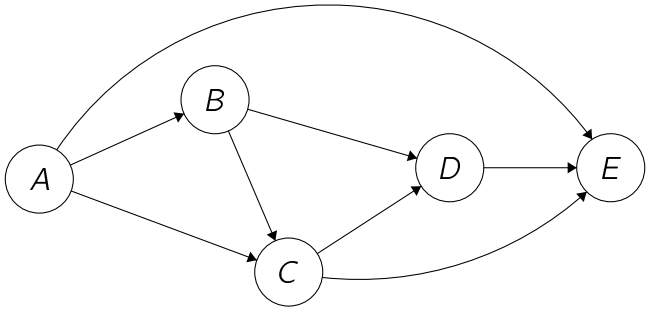
   (a) For $i = 1, 2, 3$, let $h_i(x)$ be the $i$'th least-significant digit of $x$. (For example, $h_3(456) = 4$). Define $\mathcal{H} = \{h_1, h_2, h_3\}$. Is $\mathcal{H}$ a universal hash family?

   (b) 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 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.** Consider the following directed acyclic graph (DAG):




In class, we saw how to use DFS to find a topological ordering of the the vertices; in the graph above, the unique topological ordering is $A, B, C, D, E$. We saw an example where we happened to start DFS from the first vertex in the topological order. In this exercise we'll see what happens when we start at a different vertex. Recall that when you run DFS, if it has reached everything it can but hasn't yet explored the graph, it will start again at an unexplored vertex.

(a) Run DFS starting at vertex $C$, breaking any ties by alphabetical order.[1]

   i. What do you get when you order the vertices by **ascending** start time?

   ii. What do you get when you order the vertices by **descending** finish time?

(b) Run DFS starting at vertex $C$, breaking any ties by **reverse** alphabetical order.[2]

   i. What do you get when you order the vertices by **ascending** start time?

   ii. What do you get when you order the vertices by **descending** finish time?

[**We are expecting:** For all four questions, an ordering of vertices. No justification is required.]

[1] For example, if DFS has a choice between $B$ or $C$, it will always choose $B$. This includes when DFS is starting a new tree in the DFS forest.

[2] For example, when DFS has a choice between $B$ or $C$, it will always choose $C$. This includes when DFS is starting a new tree in the DFS forest.

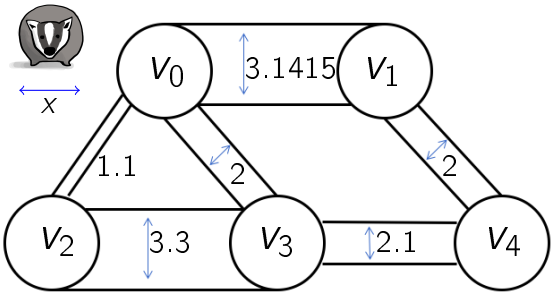
# Problems

- Try the problems on your own *before* asking questions.
- Write up your answers yourself, in your own words. You should never share your typed-up solutions.

---

4. **Badger badger badger.** A family of badgers lives in a network of tunnels; the network is modeled by a connected, undirected 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)` since the graph $G$ is undirected). If $\{v_i, v_j\}$ is not an edge in $G$, then you have no guarantee about what `tunnelWidth(i,j)` returns.

   (a) 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 we have $s = 0$ and $t = 4$. 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) 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 real number $x$ so that there exists a path from $v_s$ to $v_t$ which accomodates a badger of width $x$. 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. **[OPTIONAL] Painted Penguins.** Much to Plucky's delight, 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 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. You'd like to design a randomized data structure to keep track of the penguin colors so that, after all the penguins have gone, you'll be able to answer queries about what colors of penguins appeared in the flock; you'd like your answers to these queries to probably be correct.

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) Suppose that $n = 10m$. Suppose also that you only have space to store:

- An array $B$ of length $n$, consisting of 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`,

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

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) 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.]