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 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.]
4. (4 pt.) Consider the following directed acyclic graph (DAG):

In class, we saw how to use DFS to find a topological ordering of 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 $B$ and $C$, 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 $B$ and $C$, it will always choose $C$. This includes when DFS is starting a new tree in the DFS forest.
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.

5. (8 pt.) [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 $v_0 \rightarrow v_1 \rightarrow v_4$ or by $v_0 \rightarrow v_3 \rightarrow v_4$). 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 $\text{tunnelWidth}$ which runs in time $O(1)$ so that if $\{v_i, v_j\}$ is an edge in $G$, then $\text{tunnelWidth}(i, j)$ returns the width of the tunnel between $v_i$ and $v_j$. (Notice that $\text{tunnelWidth}(i, j) = \text{tunnelWidth}(j, i)$ since the graph is $G$ undirected). If $\{v_i, v_j\}$ is not an edge in $G$, then you have no guarantee about what $\text{tunnelWidth}(i, j)$ returns.

[Actual questions on next page.]
(a) (4 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 \textbf{True} if there is a path from \( v_s \) to \( v_t \) that a badger of width \( x \) could fit through, or \textbf{False} if no such path exists.

(For example, in the example above we have \( s = 0 \) and \( t = 4 \). Your algorithm should return \textbf{True} if \( 0 < x \leq 2 \) and \textbf{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.

\textbf{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).

\textbf{We are expecting:} Pseudocode \textbf{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 format.

(b) (4 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 real number \( x \) so that there exists a path from \( v_s \) to \( v_t \) which accommodates 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.

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

\textbf{Note:} The same note about pseudocode holds as in part (a).

\textbf{HINT: Use part (a).}

\textbf{We are expecting:} Pseudocode \textbf{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.
6. (8 pt.) *(Wake up, Sheeple!)* You arrive on an island with $n$ sheep. The sheep have developed a pretty sophisticated society, and have a social media platform called Baaaahtter (it’s like Twitter but for sheep). Some sheep follow other sheep on this platform. Being sheep, they believe and repeat anything that they hear. That is, they will re-post anything that any sheep they are following said. We can represent this by a graph, where $(a) \rightarrow (b)$ means that $(b)$ will re-post anything that $(a)$ posted. For example, if the social dynamics on the island were:

```
  Shifra the sheep
  ↓
Shakira the sheep
  ↓
Sherman the sheep
  ↓
Sugar the sheep
```

then Sherman the Sheep follows Sugar the Sheep, and Sugar follows both Shakira and Shifra, and so on. This means that Sherman will re-post anything that Sugar posts, Sugar will re-post anything by Shifra and Shakira, and so on. (If there is a cycle then each sheep will only re-post a post once).

For the parts below, let $G$ denote this directed, unweighted graph on the $n$ sheep. Let $m$ denote the number of edges in $G$.

(a) (2 pt.) Call a sheep an **influencer** if anything that they post eventually gets re-posted by every other sheep on the island. In the example above, both Shifra and Shakira are influencers. Prove that all influencers are in the same strongly connected component of $G$, and every sheep in that component is an influencer.

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

(b) (4 pt.) Suppose that there is at least one influencer. Give an algorithm that runs in time $O(n+m)$ and finds an influencer. You may use any algorithm we have seen in class as a subroutine.

**[We are expecting: The following things:**

- Pseudocode *OR* a very clear English description of your algorithm
- an informal justification that your algorithm is correct
- an informal justification that the running time is $O(n + m)$

You may use any statement we have proved in class without re-proving it.]

(c) (2 pt.) Suppose that you don’t know whether or not there is an influencer. Give an algorithm that runs in time $O(n+m)$ and either returns an influencer or returns *no influencer*. You may use any algorithm we have seen from class as a subroutine, and you may also use your algorithm from part (b) as a subroutine.

**[We are expecting: The following things:**

- Pseudocode *OR* a very clear English description of your algorithm
- an informal justification that your algorithm is correct
- an informal justification that the running time is $O(n + m)$

You may use any statement we have proved in class without re-proving it.]

---

3 Also my new start-up idea
Feedback

There’s no “correct” answer here, and it is completely anonymous.

7. (1 pt.) Please fill out the following poll, which is a mid-quarter feedback survey. It is a little bit longer, but we really do read every comment so please keep submitting your thorough feedback!

   https://forms.gle/8ew1QhVroGJ619Dc8

Did you fill out the poll?

[We are expecting: The answer “yes.”]