CS 161 (Stanford, Winter 2026) Homework 5

Style guide and expectations: We do NOT accept handwritten solutions. Please see the "Homework" part of the "Resources" section on the webpage for guidance on what we look for in homework solutions. We will grade according to these standards. You should cite all sources you used outside of the course material. Please do not distribute this material on any public forum.
Note about tagging your pages on gradescope: Please tag all of your pages to the correct question number on gradescope. We will apply a **5% deduction** to all untagged answers.
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!
Pair submissions: You can submit in pairs for this assignment. If you choose to do this, please submit **one** Gradescope assignment per pair and be sure to tag both partners on your submission. Note that we still encourage exercises to be done solo first.

Exercises. The following questions are exercises. We suggest you do these on your own. As with any homework question, though, you may ask the course staff for help.

1 Exercise: Universality

Trucky the terrapin is organizing a campus pickleball tournament for up to 100 Stanford students! To efficiently track player registrations, Trucky wants to use participants' 8-digit Stanford ID to build a hash table containing 100 buckets.

Trucky still needs to choose a hash family $\mathcal{H} = \{h_m : m \in \{1, \dots, 1000\}\}$, and being a thinking terrapin, wants it to be universal.

Trucky has several ideas, and wants to know if they result in a universal hash family.

[We are expecting: For each candidate formulation, a proof of universality or a counterexample.**]**

1.1 (2 pt.)

Let $h_m(x)$ be the sum of the digits in input x, plus m, truncated to the final 2 digits. For example with $m = 100$, $h_{100}(01234567) = 28$ because $0+1+2+3+4+5+6+7+100 = 128$.

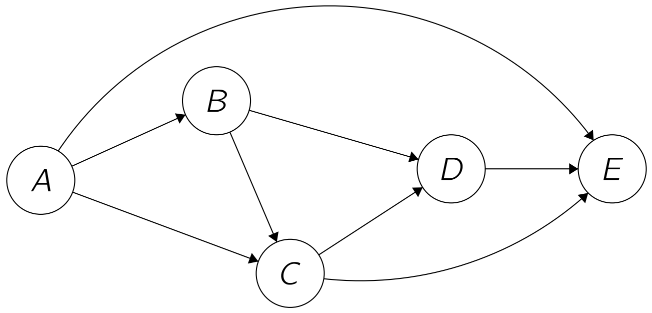
Solution

1.2 (2 pt.)

Let $h_m(x)$ be the final 2 digits of mx. For example, $h_4(01234567)$ = the final 2 digits of $4938268 = 68$.

2 Exercise: BFS and DFS Basics

Consider the following directed acyclic graph (DAG):



2.1 (2 pt.)

Run DFS starting at vertex A, breaking any ties by **reverse** alphabetical order.[1]

(a) What do you get when you order the vertices by **ascending** start time?

(b) What do you get when you order the vertices by **descending** finish time?

[We are expecting: An ordering of vertices. No justification is required.**]**

2.2 (1 pt.)

Run BFS *(not DFS)* starting at vertex D, *treating all edges as undirected*. Break any ties by alphabetical order. What is the order that the nodes are marked by BFS?

[We are expecting: An ordering of vertices. No justification is required.**]**

[1]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. The following questions are problems. You may talk with your fellow CS 161-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, other than your partner (if submitting in pairs).
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

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

3.1 (6 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 $\{0, \dots, T\}$, 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` waddle by.
- `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:

1. Describe how the array B and the function h are initialized.
2. Give pseudocode for `Query`.
3. 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.**]**

3.2 (6 pt.)

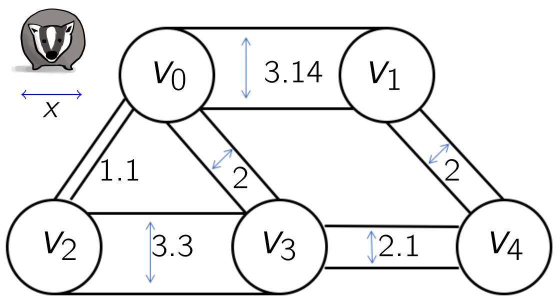
Suppose that you now have k times the space you had in part (a). That is, you can store k arrays $B_1, \ldots, B_k$ and k functions $h_1, \ldots, h_k$ from $\mathcal{H}$. Adapt your data structure from part (a) so that all operations run in time $O(k)$, and the `Query` operation is correct with probability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

[We are expecting: 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.**]**

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 an example below. 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` so that `tunnelWidth(i,j)` returns the width of the tunnel between v_i and v_j if $\{v_i, v_j\}$ is an edge in G. You may assume that the runtime of `tunnelWidth` is $O(1)$. (It is guaranteed 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.

4.1 Is there a path for a given badger? (5 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 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 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 modify an algorithm introduced in class, please, write out the full modified algorithm in pseudocode.**]**

4.2 Find the largest fitting badger (6 pt.)

Design a deterministic algorithm which takes as input G in the format above and integers $s, t \in \{0, \dots, n-1\}$; 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.

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

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.

Hint: Use part (a).

[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.**]**

4.3 Ethics (4 pt.)

Suppose you want to design a network of tunnels that can accommodate the widest of badgers. City planners do something similar: the tunnels are roads, and the badgers represent traffic. Imagine you are tasked with designing a road system that avoids congestion by optimizing for the widest roads possible to accommodate the heaviest amount of traffic the route can get. However, you realized that wider roads actually do not help with traffic congestion (see Building Bigger Roads Actually Makes Traffic Worse). What are some other considerations that are overlooked when we optimize the roads for cars? Here are a few articles for some inspiration:

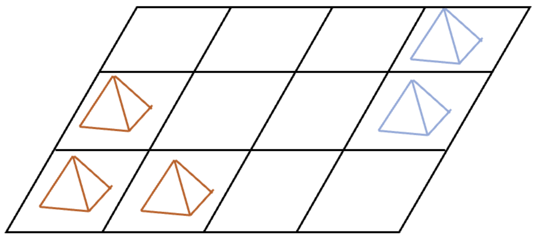
- Speed kills, so why do we keep designing for it?
- Places and non-places
- Life in the Slow Lane
- Widening Highways Doesn't Really Help Traffic

[We are expecting: four to six sentences explaining (1) what other groups of people who share the road we overlook when we only consider car users; (2) what are the consequences they face when traversing a city planned for cars; and (3) what are the consequences that everyone faces when car traffic is encouraged.**]**

5 Summer Camp (6 pt.)

Some Stanford raccoons decided to hold a summer camp event at the Portola Redwoods State Park. The camping ground is divided into an $m \times n$ grid, where each cell can accommodate one standard-sized tent. The Stanford raccoons put up their tents connected on the grid (considering four cardinal directions). The following day, they discover another connected collection of tents put up by raccoons from Berkeley. Two groups of raccoons are willing to make friends with each other and connect their living places into a single collection by putting up some spare tents. Please help design an $\mathcal{O}(mn)$ algorithm that calculates the minimum number of spared tents that the raccoons need to put up.

Example of camping ground layout If run your algorithm on this, the output should be 2 (We need to put up at least 2 tents to connect the two groups of tents).



Input: A nested array A of $m \times n$ where 1 represents a tent and 0 represents an empty camping ground.

Hint: You might want to locate all tents in one connected collection of tent first.

[We are expecting: Pseudocode of the algorithm and a clear English description, **the algorithm should run in $\mathcal{O}(mn)$ time.]**