CS 330, Spring 2025, Homework 2
Due Wednesday, September 17th, 11:59 pm EST, via Gradescope

Homework Guidelines

Collaboration policy If you choose to collaborate on some problems, you are allowed to discuss each problem with at most 3 other students currently enrolled in the class. Before working with others on a problem, you should think about it yourself for at least 45 minutes.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." It is a **violation of this policy to submit a problem solution that you cannot orally explain to an instructor or TA.**

Typesetting Solutions should be typed and submitted as a PDF file on Gradescope. You may use any program you like to type your solutions. LaTeX, or "Latex", is commonly used for technical writing (`overleaf.com` is a free web-based platform for writing in Latex) since it handles math very well. Word, Google Docs, Markdown or other software are also fine.

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise (and concise) description of the algorithm in English and pseudocode (*),
2. a proof of correctness,
3. an analysis of the asymptotic running time of your algorithm.

You may use anything we learned in class without further explanation. This includes using algorithms from class as subroutines, stating facts that we proved in class, e.g., correctness of subroutines, running time of subroutines and use the notation. Your description should be at the level that it is clear to a classmate who is taking the course with you.

You should be as clear and concise as possible in your write-up of solutions.

A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand.

(*) It is fine if the English description concentrates on the high level ideas and doesn't include all the details. But the reader should not have to figure out your solution solely based on the pseudocode. You can also add comments to your pseudocode, in fact that is best practice.

Note: *This homework has one paper-and-pencil problem (the one in this file) and a programming assignment. Please make sure to complete both!*

Problem 1: Square Graph

A large consortium is aggregating transit data from hundreds of agencies (subways, commuter rails, buses, ferries) to build a continent-wide trip planner. To support transfer analysis, they need to construct a "one-transfer map": a graph in which two stations are connected if a passenger can travel between them using at most one transfer (i.e., in one or two rides).

You are given an undirected graph G, represented as an adjacency list (implemented as a nested hash table), where nodes correspond to transit stations and edges correspond to direct connections between two stations. Your task is to design an algorithm that constructs the square G^2 of the graph G: G^2 has the same set of nodes as G. An edge (u, v) appears in G^2 if and only if there exists a path of length 1 or 2 between u and v in G. Paths of length 1 correspond to direct connections. Paths of length 2 correspond to pairs of stations connected through exactly one intermediate station.

Don't forget to include all parts of a formal solutions to an algorithms problem! (See the guidelines on the first page. We won't remind you of this in further assignments, nevertheless it always applies.)

For the runtime analysis give the tightest asymptotic runtime bound you can, expressed as a function of $n = |V|$, $m = |E|$, and d. Hint: You may find the expression $2m = \sum_{u \in V} \deg(u)$ useful where $\deg(u)$ is the degree of node u. We aim for the tightest bound as most often G is sparse, that is, that the maximum node degree is bounded by $d \ll n$. This assumption is realistic, as each station connects to only a few nearby stations out of more than 610,000 transit stops in the U.S. alone.

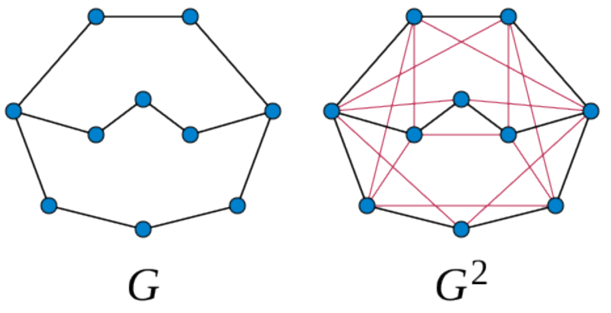

Figure 1: A graph G and its square G^2. The red edges in G^2 represent paths of length 2 in G.

Below is an example on how to represent key graph operations on adjacency lists in pseudocode. Note, that this is not part of the problem solution.

```
Algorithm 1: DummyFunction($a$, $b$)
   /* illustrates basic graph operations, $a$ and $b$ are nodes */
1  $G \leftarrow \{\}$ /* create a new, empty graph in the adjacency list format */
2  $G[a] \leftarrow \{\}$ /* add node $a$ to graph $G$, currently without outgoing edges */
3  $G[b] \leftarrow \{\}$ /* add node $b$ to graph $G$, currently without outgoing edges */
4  $G[a][b] \leftarrow 1$ /* create a directed edge from $a$ to $b$ (the value 1 is an arbitrary
       placeholder; later, it will represent the weight of that edge) */
5  for $u \in G$ do
       /* iterate over all nodes in $G$ */
6      for $v \in G[u]$ do
           /* iterate over all neighbors of node $u$ */
           ⋮
7  return $xx$
```

Teaching Note

This problem is designed to practice working with nested hash tables in pseudocode and to review runtime analysis of nested loops of varying lengths. Since the algorithm is pretty strait forward the "proof" here should merely be a **short** explanation of the main idea.

Solution: Your solution here...

```
Algorithm 2: SquareGraph($G$)
1  $H \leftarrow G$ /* create a new adjacency list equal to $G$ */
2  for $u \in G$ do
       /* iterate over all nodes in $G$ */
3      for $v \in G[u]$ do
           /* iterate over all neighbors of node $u$ */
4          for $w \in G[u]$ do
               /* iterate over all neighbors of node $u$ */
5              if $v \neq w$ then
6                  $H[v][w] \leftarrow 1$ /* if nodes $v$ and $w$ aren't equal, create a directed
                       edge from $v$ to $w$ in the adjacency list $H$ */
7  return $H$
```

The algorithm SquareGraph takes an adjacency list G and returns the square G^2 of the graph G. It does this by iterating through each node, each of that node's neighbors, and then connecting each of those neighbors with each other neighbor.

Proof: The neighbors of each node are all two or fewer edges apart from each other, since they are at least connected through their shared neighbor. By the definition of a square graph, these neighbors should be connected to each other. By iterating through each node and connecting each of that node's neighbors to each other, SquareGraph is able to correctly return the square graph of the given graph.

Lines 4-6 iterates over each neighbor of u, with the runtime being $O(deg(u))$. For each node, the algorithm loops over each pair of neighbors and performs line 6, which has a runtime of $O(1)$. Each iteration of line 4 is performed $deg(u)$ times. Each iteration of line 3 is also performed $deg(u)$ times, resulting in $deg(u)^2$ executions of line 6 for each node. This can also be written as $\sum_{u \in V} \deg(u)^2$. Because $deg(u) \leq d$, this sum can be written as $\sum_{u \in V} d{*}\deg(u)$, or $d{*}\sum_{u \in V} \deg(u)$. We know that $\sum_{u \in V} \deg(u) = 2m$, which gives us $d * 2m$, meaning the final runtime is $O(dm)$.