Exam 1 Review

CSE 101
Winter 2021
Office Hours

**Daniel Kane:** Thursday and Friday 2:30-4:00pm or by appointment
https://ucsd.zoom.us/my/dankane

**TAs:**
Jiabei Han: Monday, Thursday, Friday 4:00-5:00pm pacific over zoom at https://ucsd.zoom.us/j/92571674513.

Vaishakh Ravindrakumar: Monday, Wednesday, Friday 11:00am-12:00pm pacific over zoom at https://ucsd.zoom.us/j/7577412678.

Manish Kumar Singh: Tuesday 4:00-6:00pm and Thursday 5:00-6:00pm pacific over zoom at https://ucsd.zoom.us/j/9029365896.

Chutong Yang: Tuesday 8:00-9:00pm and Thursday 7:00-9:00pm pacific over zoom at https://ucsd.zoom.us/s/5785340529.

**Tutor:**
Harrison Matt Ku: Tuesday, Thursday 1:00-2:30pm pacific over zoom at https://ucsd.zoom.us/my/harrisonku.
Other Review Options

• Old homeworks and exams from problem archive
• Suggested textbook problems from website
Remember

- Read exam instructions carefully
- Complete instructions assignment on gradescope
- Have a plan for how you are going to complete the exam
Graph and Connectivity (Ch 3)

- Graph basics and representation
- Depth First Search
- Connected components
- Pre- and Post- orderings
- DAGs / Topological Sort
- General directed graphs & strongly connected components
Graph Definition

**Definition:** A *graph* $G = (V,E)$ consists of two things:

- A collection $V$ of *vertices*, or objects to be connected.
- A collection $E$ of *edges*, each of which connects a pair of vertices.
Drawing Graphs

- Draw vertices as points
- Draw edges as line segments or curves connecting those points

V = \{A, B, C, D, E\}
E = \{AB, AC, AD, BD, CE, DE\}
explore(v)
    v.visited ← true
    For each edge (v, w)
        If not w.visited
            explore(w)
    w.prev ← v
Theorem: If all vertices start unvisited, 
explore(v) marks as visited exactly the vertices 
reachable from v.
Depth First Search

DepthFirstSearch(G)
   Mark all \( v \in G \) as unvisited
   For \( v \in G \)
      If not \( v.v.\text{visited} \), explore(v)

Runtime \( O(|V|+|E|) \)
Note on Graph Algorithm Runtimes

Graph algorithm runtimes depend on both $|V|$ and $|E|$. (Note $O(|V|+|E|)$ is linear time)
Graph Representations

• Adjacency list: For each vertex store list of neighbors.
  – Needed for DFS to be efficient
  – We will usually assume this representation
Theorem: The vertices of a graph $G$ can be partitioned into connected components so that $v$ is reachable from $w$ if and only if they are in the same connected component.
DFS for computing Connected Components

ConnectedComponents(G)

CCNum ← 0
For v ∈ G
    v.visited ← false
For v ∈ G
    If not v.visited
        CCNum++
        explore(v)

explore(v)

    v.visited ← true
    v.CC ← CCNum
    For each edge (v,w)
        If not w.visited
            explore(w)

Runtime $O(|V|+|E|)$. 
Pre- and Post- Orders

• Keep track of what DFS does & in what order.
• Have a “clock” and note time whenever:
  – Algorithm visits a new vertex for the first time.
  – Algorithm finishes processing a vertex.
• Record values as \( v.pre \) and \( v.post \).
Computing Pre- & Post- Orders

$\text{ConnectedComponents}(G)$

clock $\leftarrow 1$

For $v \in G$

$v$.visited $\leftarrow$ false

For $v \in G$

If not $v$.visited

explore($v$)

v.pre $\leftarrow$ clock

clock++

For each edge $(v, w)$

If not $w$.visited

explore($w$)

v.post $\leftarrow$ clock

clock++

Runtime $O(|V|+|E|)$. 
What do these orders tell us?

**Prop:** For vertices v, w consider intervals $[v.\text{pre}, v.\text{post}]$ and $[w.\text{pre}, w.\text{post}]$. These intervals:

1. Contain each other if v is an ancestor/descendant of w in the DFS tree.
2. Are disjoint if v and w are cousins in the DFS tree.
3. Never interleave $(v.\text{pre} < w.\text{pre} < v.\text{post} < w.\text{post})$
**Directed Graphs**

**Definition:** A directed graph is a graph where each edge has a direction. Goes *from* \( v \) *to* \( w \). Draw edges with arrows to denote direction.
DFS on Directed Graphs

- Same code
- Only follow *directed* edges from $v$ to $w$.
- Runtime still $O(|V| + |E|)$
- $\text{explore}(v)$ discovers all vertices reachable from $v$ following only directed edges.
Topological Orders

**Definition:** A topological ordering of a directed graph is an ordering of the vertices so that for each edge \( (v,w) \), \( v \) comes before \( w \) in the ordering.
Cycles

**Definition:** A cycle in a directed graph is a sequence of vertices $v_1, v_2, v_3, \ldots, v_n$ so that there are edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1)$
Obstacle

**Proposition:** If G is a directed graph with a cycle, then G has no topological ordering.
DAGs

**Definition:** A Directed Acyclic Graph (DAG) is a directed graph which contains no cycles.
Existence of Orderings

**Theorem:** Let $G$ be a (finite) DAG. Then $G$ has a topological ordering.
Algorithm for Topological Sort

TopologicalSort(G)
   Run DFS(G) w/ pre/post numbers
   Return the vertices in reverse postorder

Runtime: $O(|V|+|E|)$. 
Correctness

**Proposition:** If $G$ is a DAG with an edge $v \rightarrow w$ then $w.post < v.post$. 
Strongly Connected Components

**Definition:** In a directed graph $G$, two vertices $v$ and $w$ are in the same Strongly Connected Component (SCC) if $v$ is reachable from $w$ and $w$ is reachable from $v$. 
Metagraph

**Definition:** The metagraph of a directed graph $G$ is a graph whose vertices are the SCCs of $G$, where there is an edge between $C_1$ and $C_2$ if and only if $G$ has an edge between some vertex of $C_1$ and some vertex of $C_2$. 
Result

**Theorem:** The metagraph is any directed graph is a DAG.
**Proposition:** Let $C_1$ and $C_2$ be SCCs of $G$ with an edge from $C_1$ to $C_2$. If we run **DFS** on $G$, the largest postorder number of any vertex in $C_1$ will be larger than the largest postorder number in $C_2$. 
Reverse Graph

**Definition:** Given a directed graph $G$, the reverse graph of $G$ (denoted $G^R$) is obtained by reversing the directions of all of the edges of $G$. 
Algorithm Idea

• Run DFS on $G^R$ computing post-orders.
• Vertex $v$ with largest postorder is in a source SCC of $G^R$, thus a sink SCC of $G$.
• Running $\text{explore}(v)$ finds exactly the SCC of $v$.
• Repeat for largest undiscovered postorder.
Algorithm for computing SCCs

SCCs(G)

Run DFS(\(G^R\)) record postorders
Mark all vertices unvisited
For \(v \in V\) in reverse postorder
    If not \(v \cdot \text{visited}\)
        explore(v) mark component

Just 2 DFSs! Runtime \(O(|V|+|E|)\).
Shortest Paths

**Problem:** Given a graph G with vertices s and t, find the s-t path using the fewest number of edges.
Observation

If there is a length $\leq d$ s-v path, then there is some w adjacent to v with a length $\leq (d-1)$ s-w path.

• For each distance d, compute all vertices at distance d from s.
  – These are the vertices adjacent to vertices at distance d-1.

• Simplify some by using a queue to keep track of unprocessed vertices
BreadthFirstSearch(G, s)

For v ∈ V, dist(v) ← ∞
Initialize Queue Q
Q.enqueue(s)
dist(s) ← 0
While Q not empty
    u ← front(Q)
    For (u,v) ∈ E
        If dist(v) = ∞
            dist(v) ← dist(u)+1
            Q.enqueue(v)
            v.prev ← u

Total runtime: O(|V|+|E|)