Problem 1 (DFS Olympics, 30 pts)

Consider running depth first search on a graph G of n nodes. As in lecture, for each vertex v we will keep track of $\texttt{disc}[v]$ and $\texttt{finish}[v]$, the times when v is first discovered and when v finishes being processed respectively.

For some graph $G = (V, E)$, we can consider $\max_{v\in V} \texttt{disc}[v]$ and $\max_{v\in V} \texttt{finish}[v]$, the largest values of $\texttt{disc}[v]$ and $\texttt{finish}[v]$ in the graph.

(a) Over all graphs $G = (V, E)$ of size $|V| = n$, what is the maximum value of $\max_{v\in V} \texttt{disc}[v]$? What is the minimum value? Give two graphs on 6 vertices satisfying these two values respectively.

(b) Over all graphs $G = (V, E)$ of size $|V| = n$, what is the maximum value of $\max_{v\in V} \texttt{finish}[v]$? What is the minimum value? Give two graphs on 6 vertices satisfying these two values respectively.

(c) Over all graphs $G = (V, E)$ of size $|V| = n$, what is the maximum value of $\max_{v\in V}(\texttt{finish}[v] - \texttt{disc}[v])$? What is the minimum value? Give two graphs on 6 vertices satisfying these two values respectively.

Solution

(a)

The largest possible value of $\max_{v\in V}$ $\texttt{disc}[v]$ is $2n - 1$, and the smallest possible value is n.
For $n = 6$:

- *Maximum:* the empty graph on 6 vertices. DFS discovers each vertex in its own call, so the sixth discovery occurs at time $2 \cdot 6 - 1 = 11$.
- *Minimum:* the directed path $1 \to 2 \to 3 \to 4 \to 5 \to 6$. A single DFS call visits all vertices consecutively at discovery times $1, 2, \ldots, 6$.

(b)

Over all directed graphs $G = (V, E)$ with $|V| = n$, every DFS performs exactly n discoveries and n finishes, so the very last finish happens at time $2n$.
The maximum and minimum values of $\max_{v\in V}$ $\texttt{finish}[v] = 2n$ in every case.
For $n = 6$, in both the empty graph and the 6-vertex path examples above (section (a)), the final finish time is 12.

(c)

The largest possible value of $\max_{v\in V}\big(\texttt{finish}[v] - \texttt{disc}[v]\big)$ is $2n-1$, and the smallest possible value is 1.
For $n = 6$:

- *Maximum:* the directed path $1 \to 2 \to 3 \to 4 \to 5 \to 6$. Vertex 1 is discovered at time 1 and finishes at time 12, so its value of $\max_{v\in V}\big(\texttt{finish}[v] - \texttt{disc}[v]\big)$ is 11.

- *Minimum:* the empty graph. Each vertex is discovered and then immediately finished, so its value of $\max_{v \in V} \big(\texttt{finish}[v] - \texttt{disc}[v]\big)$ is 1.

Problem 2 (Failed Magic, 20 pts)

In Lecture 17, we saw a magic trick that can turn a BFS algorithm into a DFS algorithm by simply replacing the queue with a stack. But I mentioned that this magic trick does not work for all BFS algorithms. In this problem, we will see why.

Below is the BFS algorithm from Lecture 16, but with the queue replaced by a stack attempting to make it into a DFS. Find a counterexmaple, i.e. construct a graph $G = (V, E)$ (can be either directed or undirected), such that for some vertex $s \in V$, running DFS-Visit(G, s) does not correctly give you a DFS tree. The issue is with **the sequence of vertices** that you visit, so you may safely ignore the teal-colored lines for the distances, as they are not used for DFS.

You should specify the graph G (drawing it out is fine) and the vertex s that you run DFS on, along with a quick explanation of what goes wrong with the (incorrect) DFS tree it produces. (*Hint: A simple example only needs 3 vertices.*)

DFS-Visit(G, s)

```
for all $v \in V$ do
    dist($v$) ← $\infty$
    parent($v$) ← $\perp$
    color($v$) ← WHITE
end for
dist($s$) ← 0
color($s$) ← YELLOW
Stack.push($s$)
while Stack is not empty do
    $u$ ← Stack.pop()
    for all $v \in$ Adj[$u$] do
        if color($v$) = WHITE then
            dist($v$) ← dist($u$) + 1
            parent($v$) ← u
            color($v$) ← YELLOW
            Stack.push($v$)
        end if
    end for
    color($u$) ← GREEN
end while
```

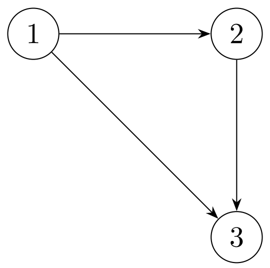
Solution

Consider the directed graph

$$G = \big(\{1, 2, 3\},\ \{(1, 2),\ (2, 3),\ (1, 3)\}\big)$$

and let the start vertex be $s = 1$. Its adjacency lists are

$$\text{Adj}(1) = \{2, 3\}, \quad \text{Adj}(2) = \{3\}, \quad \text{Adj}(3) = \emptyset.$$



A true DFS from 1 (visiting neighbors in numerical order) explores

$$1 \to 2 \to 3,$$

producing parent(2) = 1 and parent(3) = 2.

The DFS-Vist(G, s) does the following:

1. Push 1.
2. Pop 1; scan $\{2, 3\}$: push 2, then push 3.
3. Pop 3 next (LIFO!), set parent(3) = 1.
4. Pop 2 last; when scanning $\{3\}$, vertex 3 is already non-white, so no change.

This (incorrect) "DFS tree" has edges $(1, 2)$ and $(1, 3)$, whereas the correct DFS tree has edges $(1, 2)$ and $(2, 3)$. Hence this 3-vertex graph is a counterexample.

Problem 3 (Cycle Detection, 20 pts)

Explain how to modify an algorithm from class to detect if an undirected graph $G = (V, E)$ has a cycle in time $O(|V|)$ (independent of the number of edges). Justify the correctness and runtime of your proposed algorithm.

(*Hint: Something from the last homework might come handy*)

Solution

We first test the edge count:

- If $|E| \geq |V|$, then any undirected forest on $|V|$ vertices can have at most $|V| - 1$ edges. Hence in $O(1)$ time we conclude "cycle."
- Otherwise $|E| < |V|$, so we run the usual DFS
  - Initially, for every $v \in V$, set $\texttt{color}(v) \leftarrow \texttt{WHITE}$.
  - On first visiting u, set $\texttt{color}(u) \leftarrow \texttt{YELLOW}$.
  - After all neighbors of u are processed, set $\texttt{color}(u) \leftarrow \texttt{GREEN}$.
  - When exploring an edge $\{u, v\}$:
    * If $\texttt{color}(v) = \texttt{WHITE}$, recurse on v.
    * If $\texttt{color}(v) = \texttt{YELLOW}$ and $v \neq \texttt{parent}(u)$, then $\{u, v\}$ is a backward edge in an undirected DFS, so we immediately conclude "cycle."

Correctness. We consider two cases:

Case 1: $|E| \geq |V|$**.** A classical fact about forests is that a connected, acyclic undirected graph on n vertices has exactly $n - 1$ edges, and in general a forest on n vertices has at most $n - 1$ edges. Thus if $|E| \geq |V|$, no matter how the edges are arranged, the graph cannot be acyclic. Concluding "cycle" in this case is therefore always correct.

Case 2: $|E| < |V|$**.** Here we run the usual DFS cycle-detection. In an undirected graph, a cycle exists if and only if DFS ever encounters a "backward edge"—an edge $\{u, v\}$ leading to a vertex v that is currently in the recursion stack (`YELLOW`) but is not the immediate parent of u. No backward-edge test can ever give a false positive (every such edge closes a cycle), and every cycle in the graph must eventually produce at least one backward edge in some DFS tree. Since we perform this test exhaustively over all reachable vertices, we will detect a cycle exactly when one exists.

Runtime.

- The check $|E| \geq |V|$ takes $O(1)$ once $|E|$ is known. (If the edge list is stored explicitly, counting $|E|$ is $O(1)$ if provided, or $O(|E|)$ to scan once; but since we only enter DFS when $|E| < |V|$, even that initial scan is $O(|V|)$.)
- In the case $|E| < |V|$, we run DFS on an adjacency-list representation. DFS takes $O(|V| + |E|)$ time. Since here $|E| < |V|$, we have

$$O(|V| + |E|) \;=\; O(|V| + |V| - 1) \;=\; O(|V|).$$

Combining both cases, the entire procedure runs in $O(|V|)$ time, independent of the original edge count.

Problem 4 (Number of Simple Paths, 30 pts)

Give an algorithm that given a Directed Acyclic Graph (DAG, defined as a directed graph without cycles) and two vertices s, t, returns the *number* of simple paths from s to t. Your algorithm should run in time $O(|V| + |E|)$. Justify the correctness and runtime of your proposed algorithm. (Your algorithm does not need to list the simple paths, only needs to give the count.)

(*Hint: What is the letter after C? What is the letter before Q?*)

Solution

```
COUNTPATHS($G = (V, E)$, $s$, $t$)
1:  Compute a topological ordering of $V$, store it in list $L$
2:  for each $v \in V$ do
3:      $dp[v] \leftarrow 0$
4:  end for
5:  $dp[s] \leftarrow 1$
6:  for each $u$ in $L$ (in topological order) do
7:      for each $v \in \text{Adj}(u)$ do
8:          $dp[v] \leftarrow dp[v] + dp[u]$
9:      end for
10: end for
11: return $dp[t]$
```

Correctness. Since G is a DAG, a topological order L exists. We maintain the invariant that $dp[v]$ equals the number of simple paths from s to v using only vertices earlier than or equal to v in L. Initially only s has $dp[s] = 1$. When we process u, all contributions from paths ending at u have been accumulated, so propagating $dp[u]$ along each outgoing edge correctly adds all new paths to each successor. Because there are no cycles, each simple path from s to t is counted exactly once.

Runtime. Topological sorting runs in $O(|V| + |E|)$. Initializing the dp array takes $O(|V|)$, and the nested loops over each vertex in L and its outgoing edges take $O(|E|)$. Hence the total time is

$$O(|V| + |E|).$$