CS390 HW6

Pratyanch Jain | Maanas Karwa

March 14, 2025

1 12.15

Q 1.

```
1: function CoinDAG(G(V, E), s, t)
2:     order ← toposort(G(V, E))
3:     maxPath ← 0
4:     for node in order do
5:         maxPath ← max(maxPath, dfs(node))
6:     end for
7: end function
1: function DFS(node)
2:     maxNodePath ← 0
3:     for each neighbour of node do
4:         maxNodePath ← max(maxNodePath, dfs(neighbour))
5:     end for
6:     return maxNodePath + C_node
7: end function
```

Proof of Correctness: We can prove this algorithm computes the s-t walk with maximum sum of C_v where v is a distinct vertex in the walk, using induction.

Since toposort gives us a well-ordering of nodes, no back-edges and cycles are involved. We consider our base case on a sink node. Clearly, since the only nodes on any $s-t$ walk from the sink is the node itself, the return value should be C_{sink}. This holds as our DFS algorithm returns $0 + C_{node}$.

Assuming we are at the *ith* node in our topological ordering. As an inductive step, we assume the maximum reward path starting from any node with a greater topological ordering is already correctly computed. To compute the maximum reward path starting at the current node, the only nodes immediately reachable are its neighbours. Since all its necessarily have a greater topological ordering, their answers have been computed. Taking one step in any direction from our current node would place us on a neighbouring vertex. Since, we already know the answer at the neighbours, taking $C_{node} + max(DFS(neighbour))$ over all neighbours of the current node exhaustively yields the maximum reward path for the current node.

Runtime Analysis + Caching: We can cache the answers to the subproblems

as $dp[node]$ = maximum reward Path starting at node. Since there are n nodes there are n subproblems. Each subproblem takes $O(outdegree(node))$ to compute. Therefore, the overall running time with caching is $O(m+n)$ where there are m edges and n vertices in our DAG.

Q 2.

Since $C_v \geq 0$ for all vertices $v \in V$, we know that taking the coins at any node reachable from s can never decrease the maximum reward along its path.

Claim: Given the maximum (s,t) $walk$, any vertex v present in an SCC containing a node along the maximum $s-t$ $walk$ is also part of the walk.

Proof: Assume there exists a vertex v that is mutually reachable from some node u that is part of the maximum $s-t$ walk, but v is not present in the walk. Since the maximum walk goes through u, we can take a detour to v, collect some coins $C_v > 0$, and reach u again. We can continue the $s-t$ walk as before, but our maximum path now contains C_v as well. Since $C_v > 0$, we have found a greater maximum. Therefore, our path was not maximum. This contradicts our assumption. Therefore, every SCC containing nodes part of the maximum walk is fully visited.

Given this observation, we can collapse each SCC into a node where $C_{node} = \sum_{v \in SCC(node)} C_v$. Collapsing all the SCCs in a graph creates a DAG. We know this fact from multiple problems discussed in class, and homework problems. Since, we have created an algorithm in the previopus question that solves this problem for a DAG, we can just return its output on our condensed graph.

Runtime Analysis: Finding all SCCs in a graph, and condensing it takes $O(m+n)$ time. $CoinDAG$ also takes $O(m+n)$ time. $\therefore$ the overall time complexity is $O(m+n)$.