CSOR W4231.001 – Spring, 2023

Brief solutions to homework 2

1. **Solution to problem 1**

   Run BFS from s. Suppose that t is added to layer $d > n/2$. Then at least one of the layers $1, 2, \ldots, d-1$, say layer i, consists of a single node u *(why?)*. Deletion of u from the graph disconnects s from t: all paths from s to nodes appearing after layer i in the BFS tree must go through layer i, that is, node u, so all such paths are now destroyed. Running time: $O(n+m)$.

2. **Solution to problem 2**

   First suppose that G is strongly connected.

   Algorithm

   Run DFS(G), 2-coloring the vertices in alternating layers.

   If an edge is found joining two nodes in layers of the same color, output `exists odd-length cycle` and terminate.

   Else output `no odd-length cycle`.

   Running time: $O(m+n)$ *why?*

   Correctness of the algorithm follows if we prove the following claim *(why?)*.

   Claim 1. *G has an odd-length cycle if and only if an edge $(u,v) \in E$ joins two nodes in the same layer or two nodes in layers of the same color.*

   Proof. We will prove each direction separately.

   $\Longrightarrow$ *Left as an exercise (straightforward).*

   $\Longleftarrow$ Suppose that an edge (u,v) is found as above. We will exhibit an odd-length cycle in G.

   Case i. (u,v) joins two nodes in a different layer.
   First suppose (u,v) is a back edge. Then the cycle that starts at v, follows tree edge edges to u, and returns to v via the edge (u,v) has odd length since the path from u to v has an even number of edges.
   Next suppose (u,v) is a forward edge. There are two paths from u to v: a path P_1 that follows tree edges and has even length, and the path $P_2 = (u,v)$ of length 1. Since G is strongly connected, there is a path P_{vu} from v back to u of some length x. Then P_{vu} together with one of P_1, P_2 has odd length.

   Case ii. (u,v) joins two nodes in the same layer.
   Let r be the lowest common ancestor of u and v in the DFS tree. Then the tree paths P_{ru} from r to u and P_{rv} from r to v have e_1 edges each. Note that this gives

us two tree paths from r to v, one with even length and one with odd. Since G is strongly connected, there is also a path P_{vr} in G from v back to r with e_2 edges. Then P_{vr} together with one of P_{rv}, $(P_{ru}, (u, v))$ forms an odd-length cycle.

□

If it is unknown whether G is strongly connected or not, first compute all SCCs of G. Then it suffices to run the above test in each SCC (why?).

Running time: $O(n + m)$

3. **Solution to problem 3**

We use pair (a, b) to represent current liters of water in both bottles, respectively. We call it a state.

For next step, we have several possible states:

(i) **FILLUP(1)**: (X, b)

(ii) **FILLUP(2)**: (a, Y)

(iii) **EMPTY(1)**: (0, b)

(iv) **EMPTY(2)**: (a, 0)

(v) **POUR(1, 2)**: (a - min(a, Y-b), b + min(a, Y-b))

(vi) **POUR(2, 1)**: (a + min(b, X-a), b - min(b, X-a))

We make each state (a, b) as a node, and build an edge between (a, b) and all next step states. we do BFS starting from (x, y) and see if we can reach a state $(A, \tilde{y})$ or $(\tilde{x}, A)$. If the answer is yes, the distance from (x,y) is the minimum steps. If the answer is no, it means that there will be no solution.

Proof of Correctness: Since we take all possible operations as edges in our graph, and each operation correspond to an edge with length 1. The fact that BFS can find the shortest path in graph with weights all equals to 1 guarantees the correctness of this problem. If there exists a way to get A liters, it must have a sequence of operations and we can definitely get the optimal operations via BFS in the graph. If there is no solution, our BFS method won't get A liters either.

Time Complexity: $O(n^2 + 6n^2) = O(n^2)$ where n is the maximum value of X and Y. n is 1000 in this problem.

4. **Solution to problem 4**

   (a) Since an x-y path in G means that x implies y, if x and $\neg x$ both appear in the same SCC, then we have that x implies $\neg x$ and $\neg x$ implies x. Then there is no way to assign a truth value to x to satisfy both implication clauses!

   (b) We recursively find sinks in the DAG of SCCs and assign the value 1 to all the *literals* in the sink component. We then remove all the *variables* which have thus been assigned a value from the graph. Time: $O(|V| + |E|)$.

   Note that this process does not create any *unsatisfied* implications: an implication $\ell_1 \Rightarrow \ell_2$ is unsatisfied if $\ell_1 = 1$ and $\ell_2 = 0$.

   - Since we assign the truth value 1 to all literals in the sink SCC, all implications are satisfied there.
   - Now the negations of these literals are set to 0 (by definition). But the negations of all literals in a sink SCC form a source SCC: if $\ell_1 \Rightarrow \ell_2$ appears in G, then $\neg\ell_2 \Rightarrow \neg\ell_1$ appears too; and if an SCC is sink because of an implication $\ell_0 \Rightarrow \ell_1$, then the implication $\neg\ell_1 \Rightarrow \neg\ell_0$ causes the SCC where $\neg\ell_1$ appears to be a source SCC.

     Hence the negations of all the literals in a sink SCC form a source SCC. Since all these negations are set to 0, all the implications involving them are satisfied as well.

   Hence this process constructs a satisfying assignment.

Solution to recommended exercises — DO NOT RETURN

- **Solution to recommended exercise 1**

  1. **Forward direction:** If the root r has only one child, after deleting r, the rest of the graph is connected, so r is not an articulation point. So if the root r is an articulation point then r has at least two children.

     Backward direction: If the root r has at least two children, then there is no edge to connect these sub-trees, i.e. without r, the graph is disconnected. Hence r is an articulation point.

  2. **Forward direction:** If v has a child s such that there is no back edge from s or any descendant of s to a proper ancestor of v, it means that if we remove v, then node s and its descendants cannot reach any other node of the graph. Thus, the graph would be not connected and v is an articulation point.

     Backward direction: If v is a non-root articulation point of G, then after removing v from G, the graph is not connected. Thus, the proper ancestors of v are not connected to some descendants of v. Then there must be a child s such that there is no back edge from s or any descendant of s to a proper ancestor of v.

3. **Algorithm:** We can use the following equation to compute $v.low$ by starting at the leaves of the G_π.

$$v.low = min(v.d, min(y.low), min(w.d))$$

Where y is child of v and (v, w) is a backedge.

Correctness: It's a dynamic programming problem. We can use optimal substructure to prove the correctness.

Running Time: It has the same running time as DFS. Since it is a connected graph, it is $O(E)$.

4. **Algorithm:** We first run DFS and then the algorithm in part c). By part a), we can test if the root is articulation point. By part b), if v has a child s in G_π such that $s.low \leq v.d$, v is not an articulation point. Otherwise, it is an articulation point.

Correctness: If v has a child s in G_π such that $s.low \leq v.d$, then s has a back edge to a proper ancestor of v and thus v cannot be an articulation point.

Running Time: Both DFS and algorithm in part c) run in $O(E)$, therefore the algorithm run in $O(E)$.

5. **Forward direction:** If (u, v) lies on a simple cycle, then there is a cycle $u \to v \to x_1 \to \ldots \to x_n \to u$, such that all of u, v, xi are distinct. Any path that included the edge (u, v) can be modified to include the path $u \to x_n \to \ldots \to x_1 \to v$. Thus, (u, v) is not a bridge. So, if (u, v) is a bridge, then it is not on a simple cycle.

Backward direction: Assume (u, v) is not on a simply cycle and it is not a bridge. If we remove the edge (u, v), there is still a path connecting u and v, $u \to x_n \to \ldots \to x_1 \to v$. Then, the edge (u, v) forms a simple cycle, which contradicts to the assumption. So (u, v) must be a bridge.

6. **Algorithm:** Apply the algorithm in part c). And then iterate all the edges in G_π, if an edge (u, v) satisfies $v.low = v.d$, then the edge (u, v) is a bridge.

Correctness: Any bridge in the graph G must exist in the graph G_π. Otherwise, assume that (u, v) is a bridge and that we explore u first. Since removing (u, v) disconnects G, the only way to explore v is through the edge (u, v). So, we only need to consider the edges in G_π as bridges. If there are no simple cycles in the graph that contain the edge (u, v) and we explore u first, then we know that there are no back edges between v and anything else. Also, we know that anything in the subtree of v can only have back edges to other nodes in the subtree of v. Therefore, we will have $v.low = v.d$ since v is the first node visited in the subtree rooted at v.

Running Time: Computing $v.low$ for all vertex v takes time $O(E)$ as we showed in part c). Looping over all the edges takes time $O(E)$. Thus the total time to compute the bridges in G is $O(E)$.

- **Sketch of solution to recommended exercise 2**

  First toposort the DAG. In the toposorted DAG, let i and j be the labels of s and t respectively. Recall, that all edges in the toposorted DAG go from nodes with smaller indices to nodes with larger indices, so any path from i to j can only use intermediate vertices k with $i < k < j$.

  Let p_k be the number of paths from node i to node k. *Left as an exercise: Give a DP algorithm to compute p_k.* Return p_j.