Jinyang Ruan

011696096

CPT_S 515 Advanced Algorithms HW#2

1) Step 1: Run DFS along the edges of graph *G* starting from v_1, and the ending nodes are v_2 and v_3.

   Step 2: Record all the finishing time on all nodes between two ending nodes.

   Step 3: Reverse every edge in *G*, and we get *G'* by keeping the finish time of each node.

   Step 4: Set three kinds of color value to every node,

   If the node is red in color, we set Red to that node,

   If the node is green in color, we set $Green$ to that node,

   If the node is neither red nor green in color, we set *N* to that node.

   Step 5: Run DFS from v_2 to v_1 in *G'*, store all color values for each node along the walk in an array A_1, excluding the value of v_1 and v_2.

   Step 6: Run DFS from v_3 to v_1 in *G'*, store all color values for each node along the walk in an array A_2, excluding the value of v_1 and v_3.

   Step 7: Compare the length of two arrays,

   If the length of A_1 is longer than A_2, then,

   "α is longer than β" is true

   Step 8: Check if A_1 contains only the value Red, if so, then

   "α contains only red nodes" is true

   Step 9: Check if A_2 contains only the value $Green$, if so, then

   "β contains only green nodes " is true

2) Step 1: Since $c_1 + c_2 < K$, we have $c_1 + c_2 + \varepsilon = K, \varepsilon\ is\ a\ nonnegative\ constant.$

   Step 2: Initialize $c_1 = 0$, $c_2 = K - \varepsilon$, $f(e) = 0, e \in \boldsymbol{E}$. Then, we have a network flow G' with K.

   Step 3: Construct a residual graph G_r.

   Step 4: Send maximum flow f to an augmenting path of G_r, after the flow go through the augmenting path, calculate the current value of c_1 and c_2.

   Step 5: If current value $c_1 + c_2 \geq K$, then increase c_1 starting value by 1, decrease c_2 starting value by 1, then repeat doing step 4 and step 5 until satisfying the condition $c_1 + c_2 < K$.

   Step 6: If current value $c_1 + c_2 < K$, then update residual graph G_r, keep doing step 4 until there is no augmenting path remains. Sum all local maximum flow f, we get global maximum flow.

3) Step 1: Drop all yellow nodes from G by using DFS starting from v_0, then we obtain G'.

   Step 2: Run SCC on G', then we get all looping SCC's.

   Step 3: Find all green nodes that can reach a looping SCC in G', define those green nodes as NodeA. NodeA can reach a looping SCC after drop all yellow nodes means there was no yellow node after NodeA in G.

   Step 4: Find all NodeB's which are red and could be able to reach a NodeA.

   Step 5: Translate original question to check whether G satisfies "there is an infinite walk α from v_0, α does not pass any NodeB".

   Step 6: Drop all NodeB's by using DFS starting from v_0, then we obtain G''.

   Step 7: Run SCC on G''.

Step 8: If there is an α starting from v_0 can reach a looping SCC in G'', G satisfies the property in step 5, but G does not satisfy the original property.

If there is not an α starting from v_0 can reach a looping SCC in G'', G does not satisfy the property in step 5, but G satisfies the original property.

4) Q1. Step 1: Define a variable $n = 0$ which is used to store the number of paths from v to v'.

Step 2: Do a topological sort of the DAG graph, find one path from v to v'. If there is not path from v leads to v', return $n = 0$, else $n = n + 1$.

Step 3: Based on the path we have found in step 2, backtrack one node to take another path until there is no other child of this node that have not been taken, then keep backtracking one node. If the path does not lead to v', discard this path. If the path leads to v', save this path as original path, $n = n + 1$, recursively doing step 3.

Step 4: After backtracking to source node v, finish backtracking, return n. Then we get the number of paths from v to v'.

Q2. Step 1: Define a variable $k = 0$ which is used to count difference between green and yellow nodes, and a variable $n = 0$ which is used to store the number of good paths from v to v'.

Step 2: Do a topological sort of the graph, for each step, before deleting the node and all edges start from that node, determine the color of the node.

If node is green in color, $k = k + 1$;

If node is yellow in color, $k = k - 1$.

If the node is neither green nor yellow, k dose not change.

Step 3: Find one path from v to v', determine if $k > 0$;

If $k > 0$, then $n = n + 1$,

If $k \leq 0$, n does not change.

Step 4: Keep the value of k. If the node before the position you are going to backtrack is green, $k = k - 1$, . If the node before the position you are going to backtrack is yellow, $k = k + 1$. If the node before the position you are going to backtrack is neither green nor yellow, k dose not change.

Step 5: Backtrack one node to take another path until there is no other child of this node that have not been taken, then keep backtracking one node.

If the path does not lead to v', discard this path. If the path leads to v', determine the value of k. If $k > 0$, then $n = n + 1$.

Step 6: After backtracking to source node v, finish backtracking, return n. Then we get the number of good paths from v to v'.