Advanced Algorithms: Homework 2

Ivani Patel

October 10, 2022

1. In Lesson 3, we talked about the Tarjan algorithm (SCC algorithm). Now, you are required to find an efficient algorithm to solve the following problem. Let G be a directed graph where every node is labeled with a color. Many nodes can share the same color. Let v1,v2,v3 be three distinct nodes of the graph (while the graph may have many other nodes besides the three). I want to know whether the following items are all true: there is a walk A from v1 to v2 and a walk B from v1 to v3 such that

   •α is longer than β;

   •α contains only red nodes (excluding the two end nodes);

   •β contains only green nodes (excluding the two end nodes).

   (a) Given that G is a directed graph where every node is labeled with a color. Also v1,v2,v3 are three distinct nodes of the graph, And there is a walk α from v1 to v2 and a walk β from v1 to v3.

   Algorithm for given conditions:
   Step 1:In graph G beginning from V1 run a depth first search alongside the edges of graph.

   Step 2: As we traverse along the edges, record the finishing time for each node within the node.

   Step 3: Flip the graph so that every one of the edges starting with one node then onto the next are reversed by keeping the finishing time of the node.

   Step 4: Consider 3 values Red, Green, None. If the node is red in colour then then we set Red to the node. If the node is green in colour then then we set Green to the node. If the node is neither red or green in colour then then we set None to the node.

   Step 5: Perform depth first search from V2 to V1 and store one of the values Red, Green, None for each node along the path in an array A1. In A1 don't include V2 and V1.

   Step 6: Perform depth first search from V3 to V1 and store one of the values Red, Green, None for each node along the path in an array A2. In A2 don't include V3 and V1.

   Step 7: If length of A1 > length of A2,
   then length of α > length of β is true

   Step 8: If A1 contains only Red values,
   then α contains only Red nodes is true
   is **Step 9**: If A2 contains only Green values,

   then β contains only Green nodes is true

2. In Lesson 4, we learned network flow. In the problem, capacities on a graph are given constants (which are the algorithm's input, along with the graph itself). Now, suppose that we are interested in two edges e1 and e2 whose capacities c1 and c2 are not given but we only know these two variables are non-negative and satisfying $c1 + c2 < K$. where K is a given positive number (so the K is part of the algorithm's input). Under this setting, can you think of an efficient algorithm to solve network flow problem? This is a difficult problem.

(a) **Method 1**

Step 1: Initialize C1 = 0 and C2 = K - 1*C1 Let G be the network flow

Step 2: Construct a residual graph G_r

Step 3: Send maximum flow f to an augmenting path of G_r, after the flow go through the augmenting path, calculate the current value of C1, C2.

Step 4: Check if C1 + C2 < K: Then update the residual graph and keep repeating till the condition remains true until no augmenting path remains.

Step 5: Check if C1 + C2 >= K: Then increment C1 by 1 and decrement C2 by 1 and repeat Step 3 onwards till the condition becomes true

step 6: Sum all local maximum flow f, we get global maximum flow.

Method 2

Step 1: To start with, consider a flow variable starting with one node then on to the next and afterward consider that there are n nodes on which we want to apply linear programming method.

Step 2: Now organize the edges of graph G as a matrix.

Step 3: Identify the capacity constraint and the observational constraint for the given graph G.

Step 4: To find the flow to be of maximum capacity, consider that the flow that enters thorough sink S comes out of sink as T. Define the conservation constant as:
(x01 = x14+x15), (x03 = x35), (x15+x25+x35 = x57)

Step 5: From the above method identify relationship between the inputs c1, c2 and above values. And store the above values in function.

Step 6: Write a max_flow function to calculate the maximum flow for the given graph G.

3. There are a lot of interesting problems concerning graph traversal — noticing that a program in an abstract form can be understool as a directed graph. Let G be a SCC, where V_0 is a designated initial node. In particular, each node in G is labeled with a color. I have the following property that I would like to know whether the graph satisfies:

   For each inifintely long path α starting from v0, α passes a red node from which, there is an infinitely long path that passes a green node and after this green node, does not pass a yellow node.

   Please design an algorithm to check whether G satisfies the property.

   (a) **Step 1**: Drop all yellow nodes from G by using DFS starting from V_0. And then run SSC on G and we get all looping SCC

   Step 2: Find all green nodes that can reach a looping SCC and name those green nodes as GreenNode. GreenNode can reach a looping SCC after drop all yellow nodes means there was no yellow node after GreenNode in G.

   Step 3: Find all red nodes and name them as RedNode which are able to reach a GreenNode.

   Step 4: check whether G satisfies “there is an infinite walk $\alpha$from V_0, αdoes not pass any RedNode.

   Step 5: Drop all RedNode’s by using DFS starting from V_0. And again run SSC on that.

   Step 6: If there is an αstarting from V_0 can reach a looping SCC, G satisfies the property in step 5, but G does not satisfy the original property.

If there is not an αstarting from V_0 can reach a looping SCC, G does not satisfy the property in step 5, but G satisfies the original property.

4. Path counting forms a class of graph problems. Let G be a DAG where v andv'be two designated nodes. Again, each node is labeled with a color.

   (1). Design an algorithm to obtain the number of paths from v to v' in G.

   (2). A good path is one where the number of green nodes is greater than the number of yellow nodes. Design an algorithm to obtain the number of good paths from v to v' in G.

   (a) Given that G is a directed graph where each node is labeled with a color. Where v and v' are two designated nodes.

      i. An algorithm to obtain the number of paths from v to v' in G.

      Algorithm

      Step 1: Define a variable n = 0 which we use to store the number of paths from v to v'.

      Step 2: Do a topological sort of the DAG graph, find one path form v to v'. If there is no path from v 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'.

      ii. An algorithm to obtain the number of good paths from v to v' in G. A good path is one where the number of green nodes is greater than the number of yellow nodes.

      Algorithm

      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 or yellow, k does 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 does 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 node one.

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 code v, finish backtracking, return n. Then we get the number of good paths from v to v'.