ChenruiXu 515 Homework 2

chenruixu

September 2021

1 Question 1

Step 1: Start from node v1, traverse the whole graph.

Step 2: Record the corresponding final time stamps.

Step 3: Reverse the direction of every edge, and mark it as G_1 with DFN.

Step 4: According to the corresponding time records, search begin the node v2 with the destination is v1.

Step 5: Create a bool variable called color1 and initial value of it is True. By moving forward from one node to another, check the color of the node, if color is red, keep the variable to be True and if color is not red, change True to False then. Name after the walk to be α (with no node v1 and v2).

Step 6: According to the corresponding time records, search begin the node v3 with the destination is v1.

Step 7: Create a bool variable called color2 and initial value of it is True. By moving forward from one node to another, check the color of the node, if color is blue, keep the variable to be True and if color is not blue, change True to False then. Name after the walk to be β (with no node v1 and v3).

Step 8: Compare the length of *alpha* and *beta* for first question

Check color1 for the second question

Check color2 for the third question

2 Question 2

We can use linear programming method to solve this problem.

So for corresponding constrains of x1 and x2 (we name after them in one and two so that we can easily pick them, other edges we will name from x3 with corresponding constrains from c3) are c1 and c2. So if we want max $c_1 + c_2$, we turn it into LP problem.

Set the x1 is the edge from a to b and x2 is the edge from c to d. As all flow into node a equals flow out a. So $x1 = Aflowin - Aflowout(excludex1)$ and $x2 = Cflowin - Cflowout(excludex2)$

So we can rewrite the formula $c1 + c2 < k$ into $x1 + x2 < k$ and it can be rewrite into

$$max(Aflowin - Aflowout(excludex1) + Cflowin - Cflowout(excludex2))$$

with given k and all flow edges have constrains $x_i <= c_i$ for all int $i >= 3$. Since k and all other edges constrains are given as inputs, the rest of the algorithm is calculating the LP.

3 Question 3

Step 1: Traverse the whole SCC graph and detect yellow nodes.

Step 2: Delete all yellow nodes and name after the new graph G'.

Step 3: Traverse the graph G' again by using deep first search and record time stamps.

Step 4: Reverse edges and based on the time to generate new SCCs.

Step 5: For those SCCs which contain one green node and other nodes (include other green nodes, and also, other nodes are necessary, which means the number of other nodes is larger than zero), mark them as we will use them later and there is no yellow nodes for sure.

Step 6: Since the oringinal Graph is SCC, we only need to find red nodes with its degree is larger than zero (Otherwise the node is useless in terms of this question).

So, it reach the requires that from v_0 passes red nodes and then green nodes with no yellow followed (as those SCCs marked in the Step 5)

$v0 - XXXXXX - red - XXXXX - SCC(withgreen, noyellow)$

4 Question 4

4.1 Question4.1

Step 1: Initial one path stack and ram stack,$count = 0$.

Step 2: Push v into path stack and corresponding neighbour nodes into ram stack as one element (the element itself is a list).

Step 3: Pop the first element of the list of the top of ram stack and push it into path stack if the list is not a null list (at the same time delete the corresponding element in the list). Push the corresponding neighbour nodes as a list into the ram stack. If the element of the list is the element of the path stack, get rid of it.

Step 4: Repeat Step 3 until the top of ram stack is null list (if the top is not v'). Pop both stacks' top at the same time.

Step 5: Repeat Step 3 and Step 4 until we meet v' as top of the path stack. $Count+ = 1$

Step 6: Repeat Step 3,4,5 until the path stack is empty. Return count.

4.2 Question4.2

Step 1: Initial one path stack and ram stack,$count = 0$.

Step 2: Push v into path stack and corresponding neighbour nodes into ram stack as one element (the element itself is a list).

Step 3: Pop the first element of the list of the top of ram stack and push it into path stack if the list is not a null list (at the same time delete the corresponding element in the list). Push the corresponding neighbour nodes as a list into the ram stack. If the element of the list is the element of the path stack, get rid of it.

Step 4: Repeat Step 3 until the top of ram stack is null list (if the top is not v'). Pop both stacks' top at the same time.

Step 5: Repeat Step 3 and Step 4 until we meet v' as top of the path stack. As elements in the path stack have color attribute. When v' is the top of the path stack, mark green as one mark yellow as negative one. Sum all numbers together, if positive $count+ = 1$, else no action.

Step 6: Repeat Step 3,4,5 until the path stack is empty. Return count.