EL9343 Homework 9

Due: Nov. 17th 11:00 a.m.

1. Assume that the DFS procedure considers (explores) the vertices in alphabetical order, and assume the adjacency list is ordered alphabetically.

   (a) Run TOPOLOGICAL-SORT on Fig.1. Show:

      i. the discovery time and finish time of each node;

      ii. the returned linked-list.

   (b) Run STRONGLY-CONNECTED-COMPONENTS on Fig.2. Show:

      i. The discovery time and finish time for each node after running the first-pass DFS;

      ii. The discovery and finish time for each node in the second-pass DFS on the transposed graph;

      iii. The DFS forest produced by the second-pass, and the component DAG.

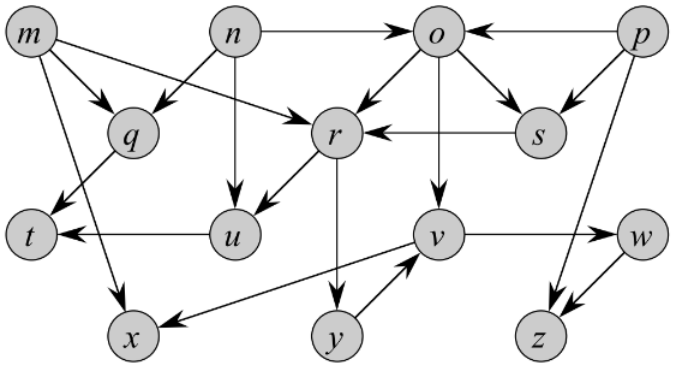


Figure 1: Graph for Q1(a)

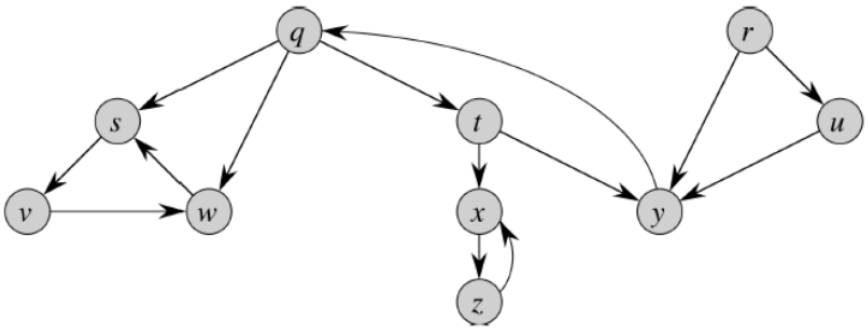
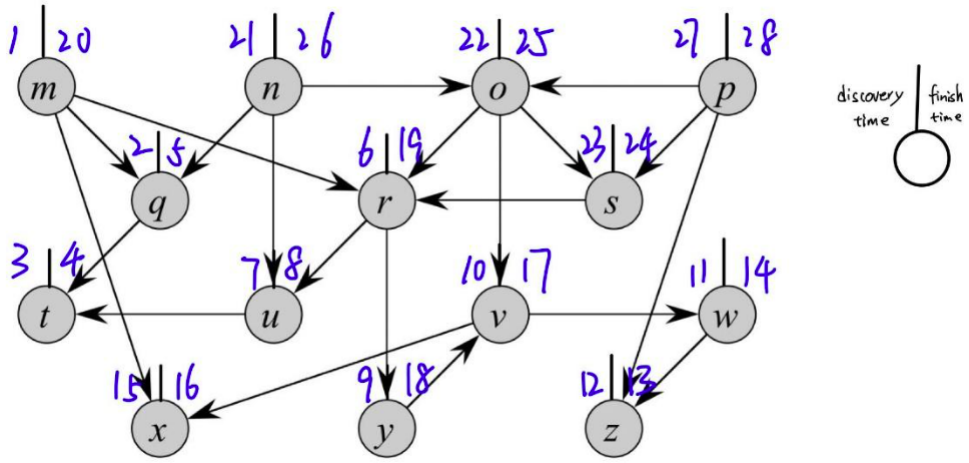


Figure 2: Graph for Q1(b)

Solutions:

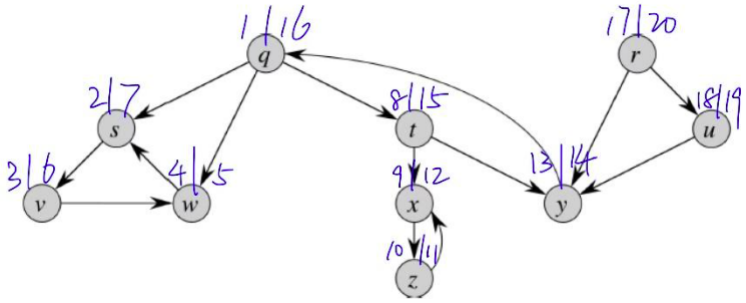
(a) i. As follows,



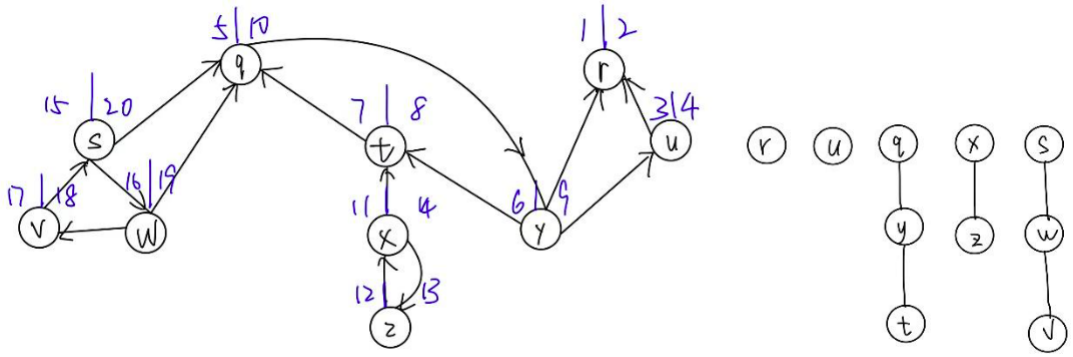
ii. The linked list:

$$p \to n \to o \to s \to m \to r \to y \to v \to x \to w \to z \to u \to q \to t$$

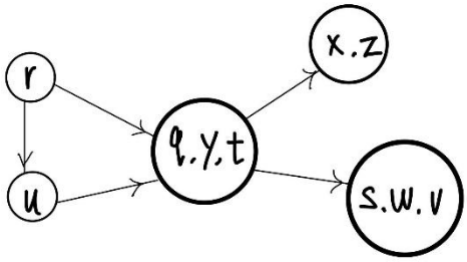
(b) i. As follows,



ii. Run DFS on G^T. Discover time and finish time lie on two sides of each node, the left is the discover time, the right is the finish time.



iii. The DFS forest has 5 distinct trees, as shown on the upper right, which means the GT has 5 strongly connected components. SCCs: $\{r\}, \{u\}, \{q, y, t\}, \{x, z\}, \{s, w, v\}$.
Component DAG:



2. You are given a system of m inequalities involving n variables, $x_1, \ldots, x_n$. Each inequality is of the form

$$x_j \geq x_i + c_{ij}$$

, for some pair of indices $i, j \in \{1, \ldots, n\}$ and some non-negative constant $c_{ij} \geq 0$.

The system is given as a weighted, directed graph G. The graph G has vertices, numbered $1, \ldots, n$, and m edges. Each inequality as above is represented in G as an edge from vertex i to vertex j with weight c_{ij}. Moreover, G is represented in adjacency list format.

(a) First assume that G is acyclic. Your task is to design an algorithm that takes G as input, and computes an assignment to the variables that satisfies the system of inequalities represented by G. More precisely, such a satisfying assignment is a list of non-negative numbers $(a_1, \ldots, a_n)$ that satisfy $a_j \geq a_i + c_{ij}$ for each inequality $x_j \geq x_i + c_{ij}$ in the system of inequalities. Your algorithm should run in time $O(m + n)$.

(b) Now consider the previous problem, but G is a general directed graph (not necessarily acyclic). Design an algorithm that determines if the system of inequalities has a satisfying assignment, and if so, outputs a satisfying assignment. Your algorithm should run in time $O(m + n)$.
Hint: Think carefully about the precise conditions under which the system of inequalities has a satisfying assignment. You may use any standard algorithm as a subroutine. You may also use an algorithm that solves the first part of the problem as a subroutine to solve the second part.

Solutions:

(a) Do a TOPOLOGICAL-SORT on the graph G, and get the rank of each node. Suppose r_i is the rank of the node corresponding to the variable x_i. $r_i = 1$ means x_i is the first one in the returned linked list, while $r_i = n$ means it is the last one. Let $C = \max\{c_{ij}\}$. Assign $x_i = r_i C$ for every i.
For any directed edge (i, j) in the graph, after the topological-sort we know that the ranks of the two nodes satisfy $r_i < r_j$. So we have, $x_j - x_i = (r_j - r_i)C \geq C \geq c_{ij} \implies x_j \geq x_i + c_{ij}$.
The topological-sort takes $O(m+n)$. Get the maximum weight of edges takes $O(m)$. Value assignment takes $O(n)$. So the total running time is $O(m + n)$.

(b) Suppose there is a cycle in G. Since c_{ij} is non-negative, the only situation that could have a satisfying assignment is when all the weights on edges of this cycle is 0, and all the variables of this cycle have the same value. A similar argument could be made on a SCC of G. So we design the algorithm as follows.
Do a STRONGLY-CONNECTED-COMPONENTS on the graph G. Check for each edge, if it is inside one SCC, its weight should be 0. If not 0, terminate for no satisfying assignment. After the checking, each SCC is treated as a node and get the resulting component DAG, G'. Use the algorithm in the previous part to assign the values, and the values of all variables inside a SCC are the same.
If the checking fails, there could not be a satisfying assignment for the reason stated above. If it passes, the algorithm is correct based on the correctness of the previous part.
The running time of STRONGLY-CONNECTED-COMPONENTS is $O(m + n)$. Checking takes $O(m)$. Value assignments take $O(m + n)$. Together is $O(m + n)$.

Note: In part (a), it is also possible to use the values of c_{ij} to write a recursive equation for value assignments as in DP problems. In this case, one may need to watch out for the edge construction part in the component DAG, where the maximum weight between the two SCCs should be picked. Using the maximum weight among all edges, i.e. $C = \max\{c_{ij}\}$, is easier.

3. Your city has n junctions numbered 1 through n. There are m one-way roads between the junctions. As a mayor of the city, you have to ensure the security of all the junctions. To ensure the security, you have to build some police checkpoints. Each checkpoint can only be built at a junction. A checkpoint at junction i can protect junction j if either $i = j$ or the police patrol car can drive from i to j and then drive back to i. Building checkpoints costs some money. As some areas of the city are more expensive than others, building checkpoints at some junctions might cost more money than other junctions. You have to determine the minimum budget B needed to ensure the security of all the junctions.

   Design an algorithm to solve this problem. The input consists of

   (a) a list of costs $c_1, \ldots, c_n$, where c_i is the cost of building a checkpoint at junction i, and
   (b) for every junction i, a list of all pairs (i, j) that represent a one-way road from junction i to junction j.

   The output is (the minimum budget) B.

   You may use any standard algorithms as subroutines. Be sure to state the running time of your algorithm, and to justify your running time bound. Your algorithm should run in time $O(n + m)$.

   Solutions:

   First, construct the directed graph. Each junction is viewed as a node. For each one-way road, add one directed edge between the two nodes. Let G be the resulting graph.

   The idea is, you must pick one and only one junction to build checkpoint for each SCC. If a checkpoint at junction i can protect some other junction j, there should exist paths from i to j and from j to i, which means i and j are inside one SCC. If you pick two junctions inside a SCC, one will be wasted. You should pick the one with minimum cost in each SCC.

   Do a STRONGLY-CONNECTED-COMPONENTS on the graph G. Inside each SCC, pick the junction with minimum cost to build the checkpoint. The costs of all picked junctions add up to B.

   The STRONGLY-CONNECTED-COMPONENTS takes $O(m + n)$. Picking the nodes and adding up takes $O(n)$. Together is $O(m + n)$.

4. (**Matrix Chain Multiplication**) Given two matrices $A : p \times q, B : q \times r$, $C = AB$ is a $p \times r$ matrix, and time to compute C is $p \times q \times r$ (calculating by $C_{ij} = \sum_{k=1}^{q} A_{ik}B_{kj}$).

   Given three matrices $A : p \times q, B : q \times r, C : r \times s$, though $(AB)C = A(BC)$, the time required to compute in the two cases could vary greatly. (E.g., when $p = r = 1, q = s$, one is $2q$ and the other is $2q^2$.)

   Now, given n matrices $A_i, i = \{1, 2, \ldots, n\}$, and A_i has size $p_i \times p_{i+1}$. Find the minimum cost way to compute the multiplication $A_1 A_2 \ldots A_n$. (Dynamic programming)

   Solutions:

   Define sub-problems as: for $1 \leq i \leq j \leq n$, find the minimum cost to compute the multiplication $A_i A_{i+1} \ldots A_j$. Let the solution to the sub-problem be $S(i, j), i \leq j$.

   Suppose the "outermost" product is $(A_i A_{i+1} \ldots A_k)(A_{k+1} \ldots A_j)$, then the cost with this "splitting point" k is $S(i, k) + S(k + 1, j) + p_i p_{k+1} p_{j+1}$. To find the optimal cost, we can try out all "splitting point" k and pick the best one. Therefore, we have,

$$S(i,j) = \begin{cases} 0 & i = j \\ \min_{i \leq k \leq j-1}\{S(i,k) + S(k+1,j) + p_i p_{k+1} p_{j+1}\} & i < j \end{cases}$$

   We can then solve this problem by solving for $S(1, n)$ using either top-down with memoizaiton or bottom-up approach.

   There are in total $O(n^2)$ sub-problem to solve, and solving each sub-problem takes at most $O(n)$ time. The time complexity is $O(n^3)$.

   Note: time complexity is not required in this question, but you can practice on it. There are in total $T(n) = \sum_{i=1}^{n-1} i(n - i)$ choosing "splitting point", and the solution to this summation is $T(n) \approxeq \frac{1}{6}n^3$, which means the complexity $O(n^3)$ is already the best we can have. Try to solve the summation yourself and check with this solution.