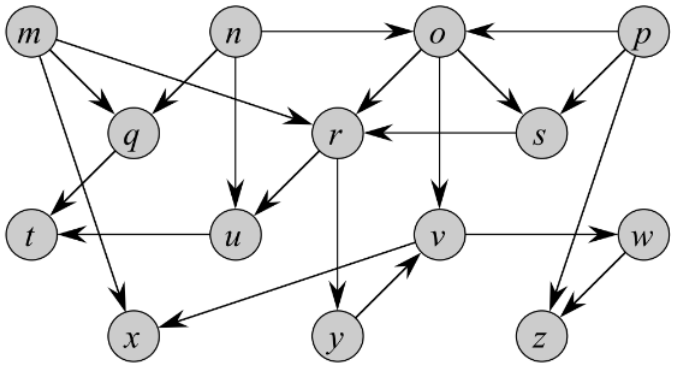
EL9343 Homework 9

Due: Nov. 16th 8:00 a.m.

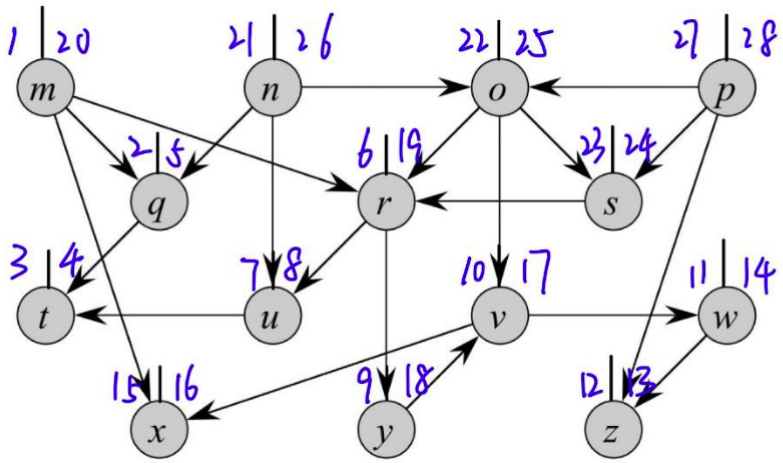
1. Run TOPOLOGICAL-SORT on the graph below. Show the:
   (a) The discovery time and finish time of each node;
   (b) The returned linked-list.

   Assume that the DFS procedure considers (explores) the vertices in alphabetical order, and assume the adjacency list is ordered alphabetically. This assumption also holds for question 2.



Solutions:

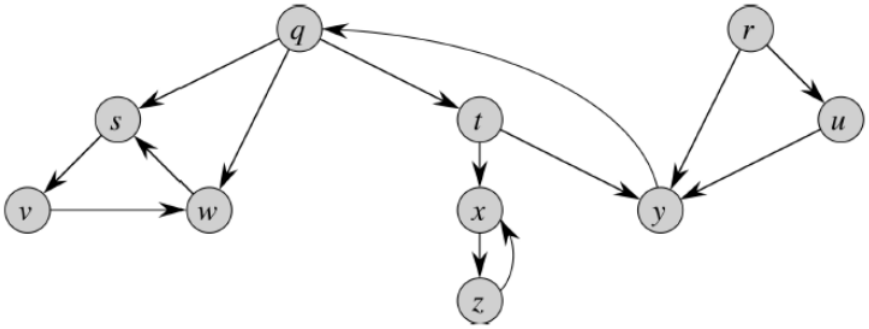
(a) As follows,



(b) The linked list:

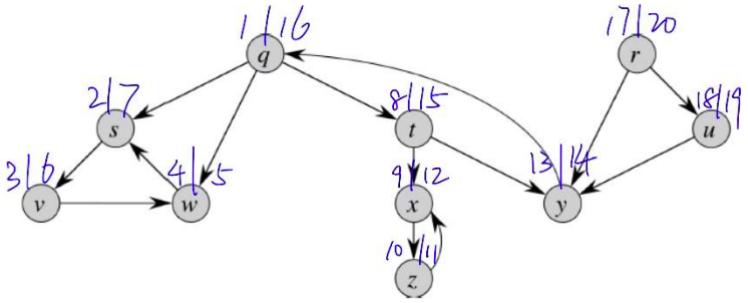
$$p \to n \to o \to s \to m \to r \to y \to v \to x \to w \to z \to u \to q \to t$$

2. Run the procedure STRONGLY-CONNECTED-COMPONENTS on the graph below. Show the:
   (a) The discovery time and finish time for each node after running DFS;
   (b) The DFS forest produced;
   (c) The component DAG.

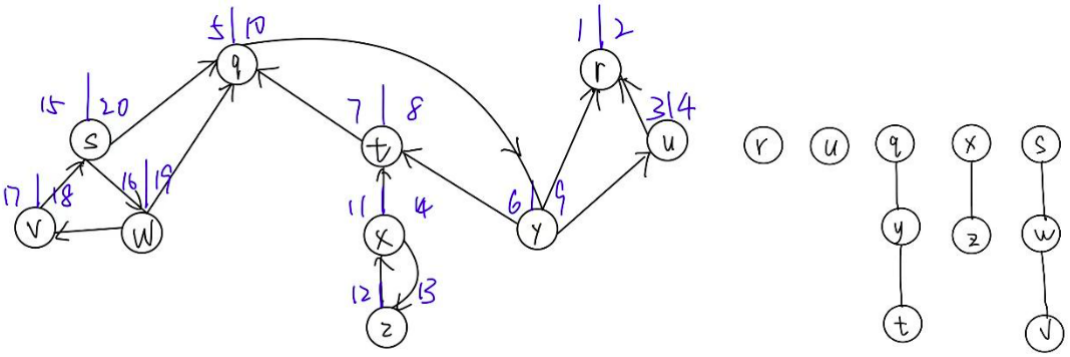


Solutions:

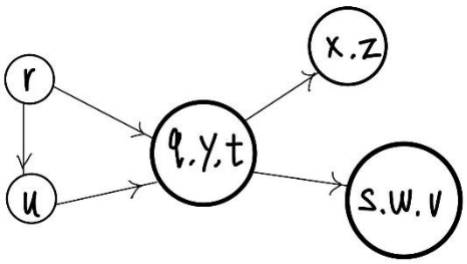
(a) As follows,



(b) 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. As we can see, there are 5 distinct trees, which means the GT has 5 strongly connected components.



(c) SCCs: $\{r\}, \{u\}, \{q, y, t\}, \{x, z\}, \{s, w, v\}$.
Component DAG:



3. (**Sightseeing.**) You are a tourist in a city, and you want to take in as many sights as possible on a single tour. Let's model this as a directed graph $G = (V, E)$, where the nodes in the graph represent the

sights, and the edges represent roads between them. For a path p in the graph, define the *sightseeing value* of p to be the number of *distinct* nodes in p. For a node $v \in V$, define *sightseeing value* of v to be the maximum sightseeing value of any path starting at v. Design an algorithm to find a node of maximum sightseeing value. Your algorithm should run in time $O(|V| + |E|)$. Also, briefly talk about the correctness of your algorithm.
(Note: you don't have to write any codes, just describing your algorithm clearly is enough. A hint is to consider using STRONGLY-CONNECTED-COMPONENTS)
Solutions:

(a) Algorithm:
Run STRONGLY-CONNECTED-COMPONENTS on G to get the component DAG, denoted as $G' = (V', E')$. Time complexity is $O(|V| + |E|)$.
Run TOPOLOGICAL-SORT on G' to get the topological order, as $v_1, v_2, \ldots, v_k \in V', k = |V'|$. Time complexity is $O(|V| + |E|)$.
For $v_i \in V'$, set $c(v_i)$ to be the number of nodes inside this strongly connected component. Time complexity is $O(|V'|) = O(|V|)$.
From $i = k$ to 1, compute $s(v_i)$. If v_i has no out-going edge, $s(v_i) = c(v_i)$; otherwise, $s(v_i) = c(v_i) + \max_{(i,j)\in E'}\{s(v_j)\}$. Time complexity is $O(|E'|) = O(|E|)$, for each edge in E' is checked exactly once.
Pick the $v_i \in V'$ with maximum $s(v_i)$, and choose any v inside this SCC. The sightseeing value of v is $s(v_i)$, which is the maximum sightseeing value. Time complexity is $O(|V|)$.
So the total time complexity is $O(|V| + |E|)$. (Though is not required, time complexity analysis is strongly encouraged.)

(b) Correctness:
When we enter a SCC, we can find a path to visit all the nodes inside this SCC (the path can have repeat edges), and arbitrarily choose one edge to go to the next SCC. Thus, we can see this problem as to start from an SCC and find a path that the sum of the number of nodes in each SCC is maximized.
So in G', we do a TOPOLOGICAL-SORT. Now it is impossible to go from an SCC in the back to an SCC in the front. Then we iterate from the back to the front, to calculate the maximum sightseeing value starting from each SCC, and finally pick the largest one.

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 $A_1A_2 \ldots A_n$. (Dynamic programming)
Solutions:
Define sub-problems as: for $1 \le i \le j \le n$, find the minimum cost to compute the multiplication $A_iA_{i+1} \ldots A_j$. Let the solution to the sub-problem be $S(i, j), i \le j$.
Suppose the "outermost" product is $(A_iA_{i+1} \ldots A_k)(A_{k+1} \ldots A_j)$, then the cost with this "splitting point" k is $S(i, k) + S(k + 1, j) + p_ip_{k+1}p_{j+1}$. To find the optimal cost, we can try out all "splitting point" k and pick the best one. Therefore, we have,

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

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.