EL-GY 9343 Homework 9

Yihao Wang
yw7486@nyu.edu

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;

The discover time and the finish time are denoted with green numbers and red numbers respectively in Fig.1.

ii. the returned linked-list.

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

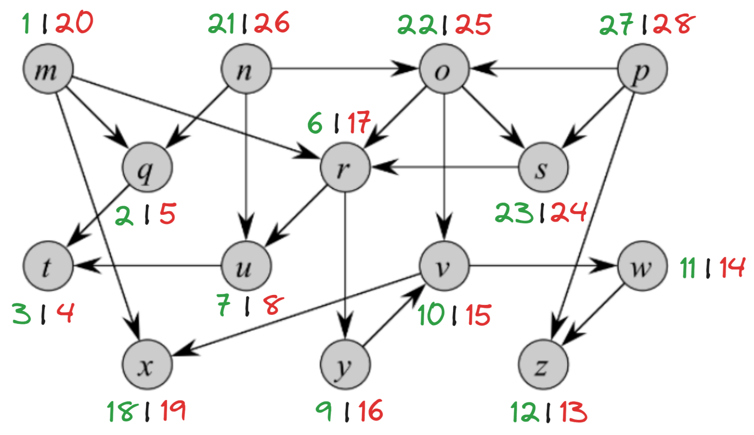


Figure 1: Graph for Q1(a)

(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;

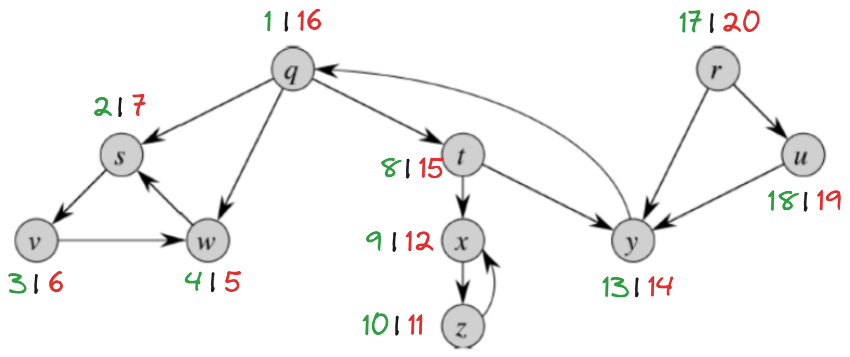
The discover time and the finish time are denoted with green numbers and red numbers respectively in Fig.2a.

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

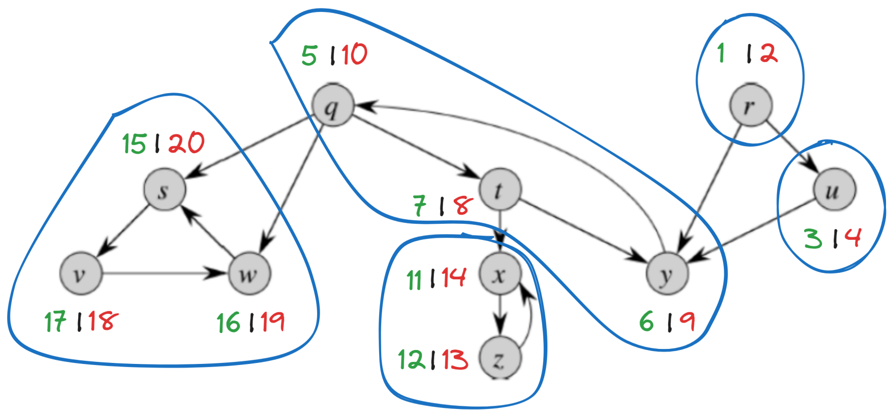
The discover time and the finish time are denoted with green numbers and red numbers respectively in Fig.2b.

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

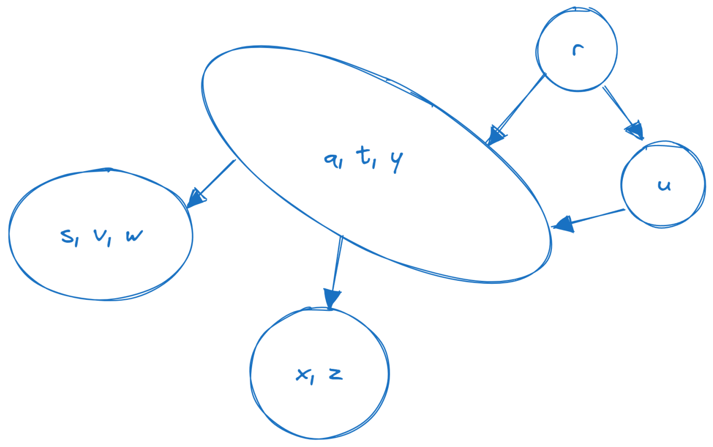
The DFS forest has been denoted with blue circles in Fig.2b, and the component DAG is shown in Fig.2c.



(a) first-pass on G



(b) second-pass on G^T



(c) component DAG

Figure 2: Graph for Q1(b)

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 $O(m + n)$.
The algorithm is shown in Algo.1, which runs in time complexity of $O(m + n)$.

Algorithm 1 AssignAcyclic

Input: Graph G, G's vertices $\{v_1, \ldots, v_n\}$ and G's edge weights $\{c_{ij}\}$,
Output: An assignment $\{a_1, \ldots, a_n\}$ that satisfies inequalities

```
1: Run TOPOLOGICAL-SORT on $G$, which yields a sorted version of vertices
   $\{v_{k_1}, v_{k_2}, \ldots, v_{k_n}\}$, such that $a_{k_p} \leq a_{k_{p+1}}$ for any $1 \leq p \leq n-1$ after assignment
2: for $i$ in $1, \ldots, n$ do                                  ▷ $O(n)$
3:     $a(v_i) \leftarrow 0$                    ▷ $a(v_i)$ is an alias of $a_i$ for clarity
4: end for
5: for $i$ in $1, \ldots, n$ do                                  ▷ $O(n)$
6:     for $u_j$ in Adj($v_{k_i}$) do                            ▷ $O(m)$
7:         $a(u_j) \leftarrow \max(a(u_j), a(v_{k_i}) + c_{k_i,j})$   ▷ ensure that $a_j \geq a_{k_i} + c_{k_i,j}$
8:     end for
9: end for
```

(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.
The algorithm is shown in Algo.2, which runs in time complexity of $O(m + n)$.

Algorithm 2 AssignGeneral

```
Input: Graph $G$, $G$'s vertices $\{v_1, \dots, v_n\}$ and $G$'s edge weights $\{c_{ij}\}$,
Output: An valid assignment $\{a_1, \dots, a_n\}$, or False if inequalities can not be satisfied
 1: Run STRONGLY-CONNECTED-COMPONENTS on $G$, which yields a component DAG
    ▷ $O(m + n)$
 2: for $C$ in component DAG do                                   ▷ $O(n)$
 3:     if some edge weights within $C$ is not 0 then             ▷ $O(m)$
 4:         return False
 5:     end if
 6: end for
 7: Run AssignAcyclic on the component DAG                        ▷ $O(m + n)$
 8: for $C$ in component DAG do                                   ▷ $O(n)$
 9:     Assign same values for all vertices within $C$
10: end for
```

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 $c_1, \dots, c_n$, 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 $O(n + m)$.

The algorithm is shown in Algo.3, which runs in time complexity of $O(m + n)$.

Algorithm 3 MinBudget

```
Input: Adjacency list $(i, j)$, costs $\{c_i\}$
Output: Minimum budget $B$
 1: Run STRONGLY-CONNECTED-COMPONENTS using adjacency list         ▷ $O(m + n)$
 2: $B \leftarrow 0$
 3: for $C$ in component DAG do                                   ▷ $O(n)$
 4:     $B \leftarrow B + \min(\{c_i\} \in C)$
 5: end for
 6: return B
```

4. 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 $C_{ij} = \sum_{k=1}^{p} A_{ik}B_{kj}$).
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 $A_i, i = \{1, 2, ..., n\}$, and A_i has size $p_i \times p_{i+1}$. Find the minimum cost way to compute the multiplication $A_1A_2 \dots A_n$. (Dynamic programming)
The algorithm is shown in Algo.4, which runs in time complexity of $O(n^3)$.

Algorithm 4 MatMul

```
Input: Matrix shapes $\{p_i \times p_{i+1}\}$ for $i = \{1, 2, ..., n\}$
Output: Minimum cost to compute the multiplication $A_1A_2 \dots A_n$
 1: let $c[1 \dots n]$ and $s[1 \dots n]$ be new arrays
 2: $c[1] \leftarrow 0$, $s[1] \leftarrow 1$
 3: for $j = 2 \dots n$ do
 4:     $q = +\infty$
 5:     for $i = 1 \dots j - 1$ do                                          ▷ $O(j^2)$
 6:         $c_1 \leftarrow p_{i+1} \sum_{k=i+2}^{j} (p_k p_{k+1})$   ▷ $c_1$ is the cost to compute $A_{i+1} \dots A_j$   ▷ $O(j - i)$
 7:         $c_2 \leftarrow p_1 p_{i+1} p_{j+1}$          ▷ $c_2$ is the cost to compute $(A_1 \dots A_i)(A_{i+1} \dots A_j)$
 8:         if $q > c[i] + c_1 + c_2$ then
 9:             $q \leftarrow c[i] + c_1 + c_2$
10:             $s[j] \leftarrow i$
11:         end if
12:     end for
13:     $c[j] \leftarrow q$                            ▷ $c[j]$ is the minimum cost to compute $A_1 \dots A_j$
14: end for
15: return $c[n]$
```