Problem 1. A directed graph $G = (V, E)$ is **singly connected** if there is at most one directed path from $u$ to $v$ for all vertices $u, v \in V$. Give an efficient algorithm to determine whether or not a directed graph is singly connected.

Solution. For each vertex $u \in V$, perform a DFS on the given graph $G$. Check if there are any forward edges or cross edges (in the same component)\(^1\) in any one of the searches. If no such edges exist, then the graph is *singly connected*, else not.

Time complexity: $O(|V|(|V| + |E|))$.

The graph is singly connected even with back edges existed. Since back edges implies there is a path $u \leadsto v$ and $v \leadsto u$, which is consistent with the definition of single connectedness.

---

\(^1\)Please refer to the classification of edges in CLRS textbook pp. 546
第二題

Yes, all \( \text{min}(u) \) can be computed in \( O(|V|+|E|) \) time.

**Reason**

**方法一**
1. 先將整張圖的所有 directed edges 全部反向
2. 從 label 爲 1 的那個 node x 開始, 將 x 可以走到的所有 node v, 令 \( \text{min}(v)=L(x) \)
3. 接著從 label 爲 2 的那個 node y 開始, 先檢查自己的 \( \text{min}(y) \) 有沒有被填値, 如果有, 就不做任何事, 如果沒有被填値, 做以下的事情: 對於 y 可以走到的所有 node u, 如果 \( \text{min}(u) \) 還沒被填入値, 就令 \( \text{min}(u)=L(y) \), 如果已經被填入了就不要更改他.
4. 依此類推, 每一個 label 都做相同的事情, 最後可以填完所有的 \( \text{min}(..) \) 值.

(沒有被填入 \( \text{min}(..) \) 值的 node 就代表在原圖中, 他的鄰邊都是 in_degree)

**Correctness:** left to you

**Time:**

- Step1: \( O(|E|) \)
- Step2: \( O(|V|+|E|) \) (use DFS or BFS)

**方法二**

**Dynamic programming:** 要算自己可以走到的最小 label 是多少, 就先去問問所有鄰居可以走到的最小 label, 再跟所有鄰居的 label 取最小值就是答案!
Problem 3. When an adjacency-matrix representation is used, most graph algorithms require time $\Omega(V^2)$, but there are some exceptions. Show that determining whether a directed graph $G$ contains a universal sink—a vertex with in-degree $|V| - 1$ and out-degree 0—can be determined in time $O(V)$, given an adjacency matrix of $G$.

Solution. If vertex $i$ is a universal sink according to the definition, the $i$-th row of the adjacency-matrix will be all “0”, and the $i$-th column will be all “1” except the $a_{ii}$ entry, and clearly there is only one such vertex. We then describe an algorithm to find out if a universal sink really exist.

Starts from $a_{11}$. If current entry $a_{ij} = 0$ then $j = j + 1$ (take one step right); if $a_{ij} = 1$ then $i = i + 1$ (take one step down). In this way, it will stop at an entry $a_{kn}$ of the last row or $a_{mk}$ of the last column ($n = |V|, 1 \leq k \leq |V|$). Check if vertex $k$ satisfies the definition of universal sink, if yes then we found it, if no then there is no universal sink. Since we always make a step right or down, and checking if a vertex is a universal sink can be done in $O(V)$, the total running time is $O(V)$.

If there is no universal sink, this algorithm won’t return any vertex. If there is a universal sink $u$, the path starts from $a_{11}$ will definitely meet $u$-th column or $u$-th row at some entry. Once it’s on track, it can’t get out of the track and will finally stop at the right entry.
Problem 4. Reading 22.3 in textbook. Show that edge \((u, v)\) is

a. a tree edge or forward edge if and only if \(d[u] < d[v] < f[v] < f[u]\).

b. a back edge if and only if \(d[v] < d[u] < f[u] < f[v]\).

c. a cross edge if and only if \(d[v] < f[v] < d[u] < f[u]\).

Solution. First, you have to show the two following lemma:

1. \(u\) is an ancestor of \(v\) \iff \(d[u] < d[v] < f[v] < f[u]\).

2. \(u\) is a descendent of \(v\) \iff \(d[v] < d[u] < f[u] < f[v]\).

Therefore,

a. \((u, v)\) is a tree edge or forward edge
   \iff \(u\) is an ancestor of \(v\) \iff \(d[u] < d[v] < f[v] < f[u]\).

b. \((u, v)\) is a back edge
   \iff \(u\) is a descendent of \(v\) \iff \(d[v] < d[u] < f[u] < f[v]\)

c. \((u, v)\) is a cross edge
   \iff \(v\) has been finished when exploring \((u, v)\) \iff \(d[v] < f[v] < d[u] < f[u]\)