Assignment 2

Kailing Wang 521030910356

October 20.2022

Problem 1. (20 points) Sometimes, we may have more than one shortest path in the graph. We want to determine whether there is more than one different shortest path. Given a graph $G = (V, E)$ with weight $w(e)$, and a start vertex s. Design an efficient algorithm to determine whether there exists more than one shortest path from s to every $u \in V$.

Answer 1.

We usually use Dijkstra algorithm to find a shortest path. For cases we want to find more than one shortest path, we still use Dijkstra.

First, we run Dijkstra algorithm for every $u \in V$, and we get the distance form s to each u, stored in set $d(V)$(we use $w(u, v)$ to represent the edge weight from u to v). Then we reverse the graph and start from a certain u. For all $e'(u, v)$ that start from u, if more than one v_i satisfies $d(u) = d(v_i) + w(u, v_i)$(we assume $w(e) > 0$ and $w(e) = w(e')$), more than one shortest path is already found, for we can reach u on a shortest path through two different vertex.

The pseudo-code is as bellow.

```
Algorithm 1: Multi-Shortest-Path
  Data: $G = (V, E)$ with weight $w(e)$ and start point $s$
  Result: index of vertexs that exists more than one shortest path
1 for each $u \in V$ do
2     run Dijkstra algorithm to calculate $d(s, u)$;
3 Reverse the graph to get $G'(V, E')$ with weight $w(e')$;
4 for each $u \in V$ do
5     find all $e'(u, v)$ that start from $u$;
6     if more than one $v_i$ s.t. $d(u) == d(v_i) + w(u, v_i)$ then
7         Record this $u$ in the result array;
8     else
9         Exists a single such $v_i$, discover $v_i$ on $G'$ similarly;
10        if find a $v$ with multi-shortist-path before reaching $s$ then record $u$;
```

The correctness is obvious: if a vertex is indeed on the shortest path to u, $d(u) = d(v_i) + w(u, v_i)$ is not necessarily true, but v_i that satisfies this condition must be on the shortest path. Before we reach s, all the vertices on existing shortest path will be reached.

The time complexity depends on the data structure. The Dijkstra for all u takes $O(V(E + VlogV))$. The reverse run is recursive, and each u is reached only once. Time complexity is no more than $O(VE)$. The total time complexity is $O(V(E + VlogV))$.

Problem 2. (20 points) Given a directed graph $G = (V, E)$, and a reward r_v for all $v \in V$. The revenue of a vertex u is defined as the maximum reward among all vertices reachable from u (including u itself). Design a linear time algorithm to output every vertex's revenue.

Answer 2.

So defined, a vertex can inherit the revenue from successor. So finding all strongly connect components will be very helpful. But, anyway, here only leaf nodes of a connected component is useful. So my method is: simply find out all the vertices whose out degree is zero. If you start from any vertex and follow the edges through, you will end up in these vertices. Then, simply let the precursor inherit revenue from these vertices.

Here is the pseudo-code.

Algorithm 2: Calculate Revenue

```
Data: Directed graph $G = (V, E)$
Result: A revenue array
1  Initialize set $rev(u) \leftarrow 0, \forall u \in V$;
2  Find SCCs and view then as super vertices;
3  Reverse the graph to get $G'(V', E')$;
4  Do topological sort;
5  for $v_i$ according to the topological order do
6      $rev(v_i) \leftarrow r_{v_i}$;
7      for $v_j$ that there exists $e'(v_i, v_j) \in E'$ do
8          if $r_{v_j} < rev(v_i)$ then
9              $rev(v_j) \leftarrow rev(v_i)$;
10         else
11             $rev(v_j) \leftarrow r_{v_j}$
```

Then let's prove the correctness. Since all the ending vertex are included, all the vertices can be reached in the reversed graph. In other words, in the reversed graph, the direction of an edge means $'$can be reached by$'$, which ensures the correctness. According to the transitivity of the maximum, the algorithm is correct.

As for the time complexity, we've done tarjan and topological sort and visit each vertices. The total time complexity is $O(V + E)$, which is linear.

I spend hours on this problem because I misunderstood the revenue as the sum of r_v that u can reach, which is far more complex if we want linear time complexity.

Problem 3. (25 points) Given a directed graph $G = (V, E)$, a source vertex $s \in V$ and a destination vertex $t \in V$. Design an efficient algorithm to determine whether there is a path from s to t containing every vertex in V ? (Vertices and edges can appear in the path more than once.)

Answer 3.

As I see it, this is also a strongly connected component problem. Here I use Tarjan algorithm. See algorithm 3 for pseudo-code for Tarjan.

Use Tarjan to find all SCCs in the graph. View each SCC as a super node and reconstruct a graph. Then use topological sort using finding the 0-in-degree nodes. s must be the starter because we want to reach any dots from s and t should be in the ending SCC or we can't reach all vertices.

The pseudo-code algorithm4 shows the complete process.

Clearly if so, all the verticies can be reached, and if not, the cases are what I explained before. This shows correctness.

Time complexity bottleneck is the Tarjan, so $O(V + E)$.

Algorithm 3: Tarjan

```
Input: a directed graph $G = (V, E)$ and a starting point s
Result: SCCs
1  Initialize a record $(dfs, low)$ for each vertices;
2  Start DFS from $s$;
3  for each vertex $u$ visited do
4      Let $u$ instack;
5      $dfs(u) \leftarrow$ the visit sequence(or index);
6      Record $u$ as visited;
7      for each $v_i$ that $u$ can reach do
8          if $v_i$ is visited before then
9              $low(u) \leftarrow min(low(u), dfs(v))$;
10         else
11             DFS from $v_i$;
12         if dfs(u)==low(u) then
13             Record a SCC from the stack
```

Algorithm 4: All vertices route

```
Input: a directed graph $G = (V, E)$ and a starting point s
Result: True or False whether there exists an all-vertex-path
1  Apply Tarjan Algorithm to find all SCCs;
2  Treat each SCC as a single vertex and reconstruct the graph;
3  Find a vertex with degree 0 and remove it;
4  if $s$ is not in this SCC then
5      return False
6  repeat
7      find a vertex with degree 0 and remove it
8  until the graph is empty;
9  if $t$ is not in the last SCC then
10     return False
11 return True
```

Problem 4. (35 points) Let $G = (V, E)$ be a directed acyclic graph (DAG). Suppose that G is not strongly connected, and you are allowed to add extra edges into G. What is the minimum number of extra edges to make G strongly connected? In the following questions, we let $H(G)$ and $T(G)$ denote the set of head vertices (no incoming edges) and tail vertices (no outgoing edges). Notice that an isolated vertex is both a head and a tail. Moreover, if all vertices in $T(G)$ can be reached by every vertex in $H(G)$, then we call the graph G is fully reachable.

(a) (5 points) Discuss the minimum number of extra edges we need to make a fully reachable graph strongly connected, and prove your claim.

(b) (10 points) If G is not fully reachable, prove that we can always find an new edge e, such that after adding $e, G' = (V, E \cup \{e\})$ is still a DAG, and we have $|H(G')| = |H(G)| - 1, |T(G')| = |T(G)| - 1$

(c) (10 points) Design an algorithm to find the minimum number of extra edges to make G strongly connected.

(d) (10 points) If the given graph G is not required to be a DAG, design an algorithm to find the minimum number of extra edges to make G strongly connected.

Answer 4.

(a) Conclusion: the minimum number of edges need to be added is $max(|H(G)|, |T(G)|)$.

Proof: First, any vertex in $E \setminus (H(G) \cup T(G))$ can be reached from a certain vertex in $H(G)$. Just go along the edges in the reversed graph and you can always find one. Similarly, they can reach some certain tail vertices. Next, connect $min(|H(G)|, |T(G)|)$ one by one. Last, connect the remaining vertices in $|T(G)|$ to some tail vertices, or some tail vertices to the remaining vertices in $|H(G)|$. Now if you start from any head vertex, you can get to any tail vertices, which can return to all other head vertices, which means you can get to any other vertices.

(b) Obviously the graph is still directed. If we want $|H(G')| = |H(G)| - 1, |T(G')| = |T(G)| - 1$, we are to find an edge to connect a head and a tail. Now we know in graph G, some certain head vertex cannot reach all the tail vertices, and so we can always find a pair of head and tail that is not connected. Connect this tail to the head. Thus we have $|H(G')| = |H(G)| - 1, |T(G')| = |T(G)| - 1$. The head cannot reach the tail so this won't from a cycle.

(c) According to the previous two sub-problems, we can first determine the $H(G)$ and $(T(G))$, and figure out whether the graph is fully reachable. See the pseudo-code in algorithm 5.

The idea is to find out whether each head vertex can reach all tail vertices. If not, simply add one edges from an unreachable tail to this head to decrease $|H(G)|$ and $T(G)$. In the end, the graph should be fully reachable because we ensure the remaining head can reach all tail vertices and the added edges will keep the graph a DAG. Then follow sub-problem (a) to find the answer.

Next we prove the minimum. Strongly connection includes fully reachable, so the construction of fully reachable is a must. During the above construction, each added edge means there is a head cannot reach a tail, and there is no case that a single edge can solve two such pairs. So each added edge necessary. From fully reachable to strongly connection we've proved the minimum.

Algorithm 5: Construct Strongly Connection

```
Input: DAG $G = (V, E)$
Result: The number of edges to be added
1 Initialize num_edge = 0;
2 Traverse all the edges to determine set $T(G)$ and $H(G)$;
3 for each $v_h \in H(G)$ do
4     DFS to find an unreachable tail $v_t$;
5     Add edges $e(v_t, v_h)$ in $G'$;
6     Delete $v_h$ in $H(G)$ and $v_t$ in $T(G)$;
7     num_edge + = 1;
8 num_edge + = $max(|H(G')|, |T(G')|)$
```

The result can be further simplified. In the process, suppose E' edges are added, we have $|H(G)| = |H(G')| + |E'|$ and $|T(G)| = |T(G')| + |E'|$. The total number is still $max(|H(G)|, |T(G)|)$.

Time complexity is $O(V + E)$ for we used DFS.

(d) The process is rather the same. First, we use Tarjan algorithm to find all SCC, and view them as super vertices. Next, apply algorithm5.

Problem 5. How long does it take you to finish the assignment (including thinking and discussing)? Give a score $(1, 2, 3, 4, 5)$ to the difficulty. Do you have any collaborators? Write down their names here.

Answer 5.

I guess 3 hours including writing but not including the time wasted on Problem 2.
Difficulty 3.

Algorithm 6: Construct Strongly Connection

```
Input: $G = (V, E)$
Result: The number of edges to be added
Initialize num_edge = 0;
Initialize a record $(dfs, low)$ for each vertices;
Start DFS from $s$;
for each vertex $u$ visited do
    $dfs(u) \leftarrow$ the visit sequence(or index);
    record $u$ as visited;
    for each $v_i$ that $u$ can reach do
        if $v_i$ is visited before then
            $low(u) \leftarrow min(low(u), dfs(v))$;
        else
            DFS from $v_i$;
        if dfs(u)==low(u) then
            Record a SCC composed of vertices with the same low value;
Simplify each SCC as a single vertex and reconstruct the graph;
Traverse all the edges to determine set $T(G)$ and $H(G)$;
for each $v_h \in H(G)$ do
    DFS to find an unreachable tail $v_t$;
    Add edges $e(v_t, v_h)$;
    Delete $v_h$ in $H(G)$ and $v_t$ in $T(G)$;
    num_edge += 1;
num_edge += $max(|H(G')|, |T(G')|)$
```