Assignment 4

CMPT307
Summer 2020
Assignment 4
Due Wed Aug 5 at 23:59
4 problems, 40 points.

1. Let $G = (V, E)$ be a directed graph with weighted edges, edge weights can be positive, negative, or zero. Suppose vertices of G are partitioned into k disjoint subsets $V_1, V_2, \ldots, V_k$; that is, every vertex of G belongs to exactly one subset V_i. For each i and j, let $\delta(i, j)$ denote the minimum shortest-path distance between vertices in V_i and vertices in V_j, that is

$$\delta(i, j) = \min\{dist(u, v) \mid u \in V_i \text{ and } v \in V_j\}$$

Describe an algorithm to compute $\delta(i, j)$ for all i and j. For full credit, your algorithm should run in $O(VE + kV \log V)$ time. (10 points)

Solution: To solve this problem, there are two key issues: first is how to represent and efficiently measure shortest path between subsets rather than vertices. Second is how to deal with the negative weighted path.

For the first issue, we add $2k$ vertices $\{s_1, s_2, \ldots, s_k\} \cup \{t_1, t_2, \ldots, t_k\}$, and add weight-0 edges to connect s_i to every $v \in V_i$ and every $v \in V_i$ to $t_i, i = 1, 2, \ldots, k$. In this case, to find the minimum shortest-path distance between vertices in V_i and vertices in V_j is equivalent to finding shortest-path distance from s_i and t_j. We denote $G' = \{V', E'\}$ to represent the new graph and have

$$V' = V \cup \{s_1, s_2, \ldots, s_k\} \cup \{t_1, t_2, \ldots, t_k\} \quad (1)$$

$$E' = E \cup \{(s_i, v), (v, t_i) | v \in V_i, i = 1, 2, \ldots, k\} \quad (2)$$

.

For the second issue, we use re-weighting similar to the Johnson's Algorithm. Below is the algorithm, which is a modified Johnson's Algorithm.

Time complexity analysis: step-1 takes $O(V + E)$ time cost, step-2 takes $O((V + k)(cE) + V + (V + E)) = O(VE)$ time cost, step-3 takes $O(k * (E + V \log V + k)) = O(kE + kV \log V)$ time cost. So the time complexity of our algorithm is $O(VE + kV \log V)$.

Algorithm 1: Shortest_Path($G, \mathcal{V}$)

```
// G is the given directed graph;
// $w(u,v)$ denotes the weight of edge $(u,v)$;
// $\mathcal{V} = \{V_1, V_2, \dots V_k\}$;
$dist \leftarrow$ a $k \times k$ matrix;
// Step-1: build $G'$ from G;
$G' \leftarrow \{V', E'\}$ build using Eq.(1),(2) on $G$;
// Step-2: reweight $G'$;
$G'' \leftarrow \{V'', E''\}$ where $V'' = V' \cup \{o\}$ and $E'' = \{E' \cup \{o, v\} : v \in V'\}$;
if Bellman-Ford($G''$, $o$) is False then
    return NIL;
end
foreach $v \in V''$ do
    $h(v) = \delta(o, v)$(from Bellman-Ford);
end
foreach edge $(u,v) \in E'$ do
    $\hat{w}(u,v) = w(u,v) + h(u) - h(v)$;
end
// Step-3: repeatly run Dijkstra to find shortest path from $s_i$ to $t_j$
 foreach $s_i \in \{s_1, s_2, \dots, s_k\}$ do
    run Dijkstra($G', \hat{w}, s_i$) to compute $\delta(s_i, v)$ for all $v \in V'$;
    for each $V_j$ do
        $dist[i,j] = \delta(s_i, t_j) - h(s_i) + h(t_j)$;
    end
end
```

2. Let $G = \{V, E\}$ be a flow network in which every edge has capacity 1 and the shortest-path distance from s to t is at least d. (10 points)

   (a) Prove that the value of the maximum (s,t)-flows is at most E/d.

   Solution: Suppose that the value of max flow is larger thant E/d, written as $|f|_m > E/d$. Since each edge is with capacity 1, according to the definition of flow, there exist $|f|_m$ paths from s to t such that each two paths are disjoint (no overlapping edges). Each path contribute value 1 to the flow. Since shortest path distance from s to t is $\geq d$, each path contains $\geq d$ edges. So the total number of edges within the max flow $\geq |f|_m * d > E/d * d = E$. This contradicts with the definition of E which is the number of edges within the graph.

   (b) Now suppose that G is simple, meaning that for all vertices u and v, there is at most one edge from u to v. Prove that the value of the maximum (s,t)-flow is at most $O(V^2/d^2)$.

   Solution: Let's divide vertices V into subsets $\{V_1, V_2, \ldots\}$, V_i contains vertices whose shortest path distance to s is i. There are more than d subsets and t will not belong to the first $d-1$ subsets.

   Let's consider V_i and V_{i+1}. There does not exists edges from previous subsets of V_i to subsequent subsets of V_{i+1}. That is, edges satisfying below equation doesn't exist:

$$\{(u,v) | u \in V_a, v \in V_b, a < i, b \geq i+1 \text{ } or \text{ } a \leq i, b > i+1\}$$

   This is easy to prove. If such (u,v) exists, the vertex v would not be allocate to V_b since there is a path from s to v across u with distance $a+1$. This contradicts with how we build V_i.

   In this case, define a cut (S,T) of G such that $S = \{s\} \cup V_1 \cup \ldots \cup V_i$ and $T = V_{i+1} \cup V_{i+2} \cup \ldots$. Edges from S to T is exactly the edges from V_i to V_{i+1}. So the capacity of the cut is the total number of edges from V_i to V_{i+1}, written as $\sum_{u \in V_i} \sum_{v \in V_{i+1}} 1$.

   Consider the average situation for big O. First, pair the V_i as V_1 with V_2, V_3 with V_4, etc. There are at least $(d-1)/2$ such pairs. Let V_i and V_{i+1} be the pair with the least number of vertices overall. By pigeonhole principle, this number is at most $n/((d-1)/2) = 2n/(d-1)$. If x is the number of vertices in V_i, then $(2n/(d-1) - x)$ is the number of vertices in V_{i+1}. The maximum number of edges from V_i to V_{i+1} is $x * (2n/(d-1) - x)$, which attains a maximum of $n^2/(d-1)^2$ when $x = n/(d-1)$. So the V_i to V_{i+1} cut has at most $O(n^2/d^2)$ edges. This is a bound on the max flow, since the max flow is no more than the capacity of of any cut. So the value of max flow is at most $O(V^2/d^2)$.

3. A cycle cover of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover every vertex in G. Describe and analyze an

efficient algorithm to find a cycle cover for a given graph, or correctly report that no cycle cover exists. (10 points)

Hint: use bipartite matching. But G is not bipartite, so you'll have to use a graph derived from G.

Solution: To solve this problem, we build a new bipartite graph G' from G. Below is the the brief description, the time complexity of this algorithm is $O(VE)$.

(a) build a bipartite graph G' from G. The left vertex set of G' is $L = V$, meanwhile the right set is $R = V$ (R is a copy of V). For each edge $(u, v) \in E$, add an edge (u_L, v_R) to G' where u_L represents the vertex u in set L and v_R the vertex v in set R.

(b) find the maximum bipartite matching of G' using Ford-Fulkerson algorithm. If the maximum bipartite matching is a perfect matching (all vertices are matched), this matching corresponds to a cycle cover of G. If the maximum matching is not a perfect matching, then there is no cycle cover exists.

Algorithm 2: Cycle_Cover($G(V, E)$)

```
// Step-1: build $G'(V \cup V, E')$, a bipartite graph generated from G;
build $G'$ with $L = V$ and $R = V$;
for each edge $(u, v) \in E$ do
    add edge $(u_L, v_R)$ to $E'$;
end
// Step-2: find maximum bipartite matching of $G'$;
$G''(V'', E'') \leftarrow$ a flow network built from $G'$ with source $s$ and sink $t$;
$f \leftarrow$ Ford-Fulkerson($G'', s, t$);
$M \leftarrow$ set of edges $(u_L, v_R)$ been used in $f$, $u_L \in L, v_R \in R$;
if $|M| == |V|$ then
    // cycle cover exists;
    $cover = \{\}$;
    for $(u_L, v_R) \in M$ do
        add $(u, v)$ to $cover$;
    end
    return $cover$;
else
    // no cycle cover;
    return NIL;
end
```

4. Solve the equation by using an LUP decomposition. (For full credit, show

your detail steps.) (10 points)

$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ -4 & 5 & 9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 8 \\ -9 \end{pmatrix}$$

Solution: We use

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

after LUP decomposition, we have

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -1.5 & 1 \end{pmatrix}, U = \begin{pmatrix} 1 & -2 & 1 \\ 0 & 2 & -8 \\ 0 & 0 & 1 \end{pmatrix}$$

Use the decomposition, we have

$$LUx = Pb$$

We define $y = Ux$ then compute $Ly = Pb$ and have

$$y = \begin{pmatrix} 0 & 8 & 3 \end{pmatrix}^T$$

Then have

$$x = \begin{pmatrix} 29 & 16 & 3 \end{pmatrix}^T$$