Lab07-Graph Exploration

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2018.

∗ If there is any problem, please contact TA Xinyu Wu.
∗ Name:Juncheng Wan Student ID:516021910620 Email: 578177149@qq.com

1. **BFS Tree:** Similar to DFS, BFS yields a tree, (also possibly forest, but **just consider a tree** in this question) and we can define **tree, forward, back, cross** edges for BFS. Denote $Dist(u)$ as the distance between node u and the source node in the BFS tree. Please prove:

   (a) For both undirected and directed graphs, no forward edges exist in the graph.

   Proof. For undirected graph, suppose that there is a forward edge $e = (u, v)$, where v is the descendant of u. Therefore, u is the ancestor of v and when BFS visits u, v has been visited. Because u is the ancestor of v, $Dist(v) > Dist(u)$. According to the algorithm of **BFS**, because u and v are connected, **BFS** will visit u after visiting v and v's brothers and set $Dist(u) = Dist(v) + 1$, which contradicts that $Dist(v) > Dist(u)$.

   For directed graph, suppose that there is a forward edge $e = u- > v$. Therefore, u is the ancestor of v and when BFS visits u, v has been visited. If v is exactly the child of u, consider three conditions:

   i. There is only one edge $e = u- > v$. According to BFS, v should be visited after u, which contradicts that when BFS visits u, v has been visited.
   ii. There is another edge $e' = u'- > v$ and u' is visited after u. According to BFS, v is visited after u. This contradicts that when BFS visits u, v has been visited.
   iii. There is another edge $e' = u'- > v$ and u' is visited before u. According to BFS, v should be the descendant of u' and thus can not be the descendant of u. This contradicts that v is the descendant of u.

   If v is the nonchild descendant of u, there should be a path from u to v and this path is in the BFS tree. According to what analyzed above, the child of u is visited after u, the child of the child of u is visited after u, $\cdots$. Equally, the father of v is visited before v and after u, which contradicts that when BFS visits u, v has been visited. □

   (b) There are no back edges in undirected graph, while in directed graph each back edge (u, v) yields $0 \le Dist(v) \le Dist(u)$.

   Proof. For undirected graph, suppose that there is a back edge $e = (u, v)$, where u is the descendant of v. Therefore, v is the ancestor of u and when BFS visits u, v has been visited. Because v is the ancestor of u, BFS will visit u by e after visiting v and v's brothers. Therefore, e can't be a back edge.

   For directed graph, suppose that there is a back edge $e = u- > v$, where u is the descendant of v. Therefore, $0 \le Dist(v) \le Dist(u)$. □

   (c) For undirected graph, each cross edge (u, v) yields $Dist(v) = Dist(u)$ or $|Dist(v) - Dist(u)| = 1$, while for directed graph, each cross edge (u, v) yields $Dist(v) \le Dist(u)+1$.

   Proof. For undirected graph, if $e = (u, v)$ is a cross edge. Without losing generality, set $Dist(v) - Dist(u) \ge 2$ and u is the brother of the grandparent (or older) of v. According to BFS, u is visited before v's parent as u is the brother of v's parent's parent (or older). As u and v is connected, BFS will visit v after visiting u and u's brothers. Therefore, $Dist(v) = Dist(u) + 1$, which contradicts that $Dist(v) - Dist(u) \ge 2$.

For directed graph, if $e = u- > v$ is a cross edge. Set $Dist(v) - Dist(u) \geq 2$ and u is the brother of the grandparent (or older) of v. According to BFS, u is visited before v's parent as u is the brother of v's parent's parent (or older). As u and v is connected, BFS will visit v after visiting u and u's brothers. Therefore, $Dist(v) = Dist(u) + 1$, which contradicts that $Dist(v) - Dist(u) \geq 2$. $\square$

2. **Second-best Minimum Spanning Tree:** Let $G = (V, E)$ be an undirected, connected graph whose weight function is $w : E \to \mathbb{R}^+$. Assume that $|V| \leq |E|$ and the weight of each edge is distinct. Let $\mathcal{T}$ be the set of all spanning trees of G, and let T' be the minimum spanning tree of G. The **second-best minimum spanning tree** is a spanning tree T'' such that $w(T'') = \min_{T \in \mathcal{T}/\{T'\}} w(T)$, where $w(T) = \sum_{e \in T} w(e)$.

   (a) Prove that G contains $u, v, x, y \in V$, $(u, v) \in T'$ and $(x, y) \notin T'$, such that $(T'/(u, v)) \cup (x, y)$ is a second-best minimum spanning tree.

   Proof. Assume that there is a spanning tree T_1 and T_1 has an edge e_1 which is not in T. We abandon e_1, and T_1 becomes two connected components. Because T is a spannin tree, there must be an edge e_1' in T that can connect these two components. Besides, e_1' is the minimal edge that connects the two components. Otherwise, it will contradict with the Prime Algorithm. After this change, T_1 becomes T_2.

   Therefore, each time we can find an edge in T that replaces an edge in T_K and transform T_k to T_{k+1} satisfying $w(T_k) \geq w(T_{k+1})$. Assume that $T_n = T$. In fact, any spanning tree can transform to T, and before they become T there must be a state that they are different from T only by an edge (T_{n-1}). Because there are limited spanning trees in a finite graph, we can choose the minimal $w(T_{n-1})$, and this T_{n-1} is what we find. $\square$

   (b) Let T be a spanning tree of G, for any two vertices $u, v \in V$, let $\max[u, v]$ denote the edge with maximum weight on the simple path between u and v in T. Please design an algorithm with time complexity $O(|V|^2)$ which computes $\max[u, v]$ for all $u, v \in V$ and $u \neq v$, given T. Briefly explain your idea and write your algorithm in *pseudo code.*

   Solution. I use Dynamic Programming to compute $\max[u, v]$. Set the vertices in a path in T with index $v_1, v_2, \cdots, v_n$. I have:

$$max[v_i, v_j] = max\{[v_j, v_{j-1}], max[v_i, v_{j-1}]]\}$$

   For all $i, j = 1, 2, \cdots, |V|$, I have a two dimension array to compute $max[v_i, v_j]$. According to Dynamic Programming, the time complexity is $O(|V|^2)$. Following is the pseudo code:

Algorithm 1: DP-MaxEdge

Input: T: a spanning tree of G; the weight of each edge in G.

Output: max$[u, v]$: the edge with maximum weight on the simple path between u and v in T.

```
1  $MAX[|V|][|V|] \leftarrow \Phi$ ;
2  M-Compute-Max($MAX[v_i][v_j]$) {
3      if $i = j$ then
4          return 0 ;
5      if $i < j$ then
6          $MAX[v_i][v_j] = max\{weight[v_j, v_{j-1}], M - Compute - Max[v_i, v_{j-1}]\}$ ;
7      return $MAX[v_i][v_j]$ ;
8  }
9  return M-Compute-Max($MAX[v_1][v_{|V|}]$) ;
```

$\square$

(c) Explain your idea to compute the second-best minimum spanning tree of G based on your analysis of Problem 2b, and analyze the time complexity of your idea.

Solution. First, compute the max$[u, v]$ for all $u, v \in V$ for the Minimum Spanning Tree T. The time complexity of this process is $O(|V|^2)$.

Second, compute the differences of all vertice pairs in T, $diff(u, v) = weight(u, v) - max[u, v]$, where $weight(u, v)$ is the weight of an edge not in T. If there is not an edge in G and not in T that connects u and v, set $diff(u, v) = \infty$. The time complexity of this process $O(|V|^2)$, assuming that we set the time complexity of elementary operation of comparing and adding is the same.

Third, find the minimal $diff(u, v)$ and the edge between u, v (this edge is not in T). We get the second-best minimum spanning tree of G. The time complexity of this process is $O(log|V|)$ if we use Binary Search.

The time complexity of this algorithm is $O(|V|^2 + |V|^2 + log|V|) = O(|V|^2)$. $\square$

3. **Counting Connected Components:** The file *Data-CCC.txt* represents a graph $G = (V, E)$. Inside, the first row shows $|V|$ (left) and $|E|$ (right). There are $|E|$ rows after the first row. Each row represents an edge, where the left element is the ID of starting vertex while the right is that of ending vertex. The ID of vertices is from 0 to $|V| - 1$. Please solve problems below.

   (a) Suppose G is an undirected graph, Please write code by C/C++ which outputs *the number of connected components (CC)* and *the largest CC*. Source code should be named as *Code-CC.cpp*, and you need to write a file named *Output-CC-Your ID.txt* (Exp. Output-CC-5140219173.txt), in which the first row is the number of CC, and the second row is the ID of vertices in the largest CC sorted in **increasing** order.

   (b) Suppose G is a directed graph, Please write code by C/C++ which outputs *the number of strongly connected components (SCC)* and *the largest SCC*. Source code should be named as *Code-SCC.cpp*, and you need to write a file named *Output-SCC-Your ID.txt* (Exp. Output-SCC-5140219173.txt), in which the first row is the number of SCC, and the second row is the ID of vertices in the largest SCC sorted in **increasing** order.

   (c) (Optional Subquestion with Bonus) Visualize the above two graphs. Please use a *metanode* to denote a CC/SCC, and the *radius* of a metanode should be positively proportional to the size of the corresponding CC/SCC.

Remark: Please include .pdf, .tex, .cpp and .txt files in your .rar files with standard names.
Please do **NOT** include *Data-CCC.txt* in your uploaded file.