ECE 606, Fall 2019, Assignment 12

Zhijie Wang, Student ID number: 20856733

zhijie.wang@uwaterloo.ca

December 3, 2019

1. IndSet(G, k)

    $result \leftarrow \mathcal{A}(G, k)$
    if $result =$ "G is not connected" **then**
        $G\_set =$ FindConnectedComponent(G)
        $ind\_set\_size \leftarrow 0$
        for each H in G_set **do**
            $tmp \leftarrow 0$
            for each i from 1 to k **do**
                if $\mathcal{A}(H, i) =$ "yes" **then**
                    $tmp \leftarrow i$
            $ind\_set\_size \leftarrow ind\_set\_size + tmp$
        if $ind\_set\_size \geq k$ **then**
            return true
        else
            return false
    else if $result =$ "yes" **then**
        return true
    else
        return false

    FindConnectedComponent(G)

    $H \leftarrow \{\}$
    for each i from 1 to $|V|$ **do**
        $is\_visited[i] \leftarrow$ false
    for each i from 1 to $|V|$ **do**
        $comp\_vertex = \{\}$
        $comp\_edge = \{\}$
        for each v in $adj[i]$ **do**
            if $is\_visited[v] =$ false **then**
                DFS$(G, is_visited, comp_vertex, comp_edge, v)$
                $comp \leftarrow < comp\_vertex, comp\_edge >$
                $H \cup comp$
    return H

    DFS$(G, is_visited, comp_vertex, comp_edge, v)$

    $is\_visited \leftarrow$ true
    $comp\_vertex \cup v$
    for each p in $adj[v]$ **do**
        if $is\_visited[p] =$ false **then**
            $comp\_edge \cup < v, p >$
            DFS$(G, is_visited, comp_vertex, comp_edge, p)$

    Here is a brief discussion about the algorithm above. IndSet will first check if the graph is connected. If the graph is connected, then output the result of $\mathcal{A}$ directly, else, evoke FindConnectedComponet to find every connected component, then, find the independent set size of each component. If the sum of each independent set size is $\geq$ k, then, output true, else false.

2. (a) For purpose of contradiction, suppose $\forall u, C\backslash\{u\}$ is not a vertex cover of G', then, G' must have some edges, said E' not incident on any vertex of $C\backslash\{u\}$. C is a vertex cover of G, therefore, $\forall e \in E'$ must be incident on u. While one edge should have two end points, therefore, $\forall e \in E'$ should have an end point besides u. Hence, these edges should have a end point with vertex $\in C\backslash\{u\}$, which makes a contradiction.

   (b) If $C = V$, then C must be a vertex cover. If we remove a vertex v and all edges incident on it from G each time and the $C\backslash v$ is still a vertex cover of G, then, after k removal, if $G = \emptyset$, then the k vertexes we have removed can construct a vertex cover. However, this algorithm cannot be implement in deterministic polynomial-time, because for each removal, there might be multiple choices, therefore, it only exists non-deterministic polynomial-time algorithm for this problem.

3. For **NP**-hard part, we reduce it from CLIQUE. Given an instance $< G =< V, E >, k >$, we introduce k new vertexes, which are not incident on any edges. Then we output $< H, k >$, where H is the new graph after introducing these new vertexes. Then we need to prove CLIQUE $\leq_k$ CLIQUEINDSET.

   For the only-if direction, suppose G have a clique set with size of k, then, H must have an independent set with size of k because the k new vertexes we introduced can construct an independent set, therefore, H must have a clique and an independent set with size of k. For the if direction, suppose G don't have a clique with size of k, then, H cannot have a clique and an independent set with size of k because the k new vertexes cannot construct a clique.

   For the **NP** part, given a true instance $< G, k >$, we could adopt as certificate $< a, b >$, where a is a clique of G and b is an independent set of G. Our verification algorithm will check if the size of a and b are both k. The certificate is linear in the size of the instance, and the verification algorithm is at worst polynomial-time.

   Therefore, CLIQUEINDSET is **NP**-complete.

4. a12p4.py