**CPSC 6109:** [**Advanced**](https://colstate.view.usg.edu/d2l/lp/ouHome/home.d2l?ou=1218642) **Algorithms**

**Spring 2018**

**Assignment #8**

**Student: Lu Lin**

**Due: 11:59 PM Tuesday, April 24**

1. Exercise **22.4-3** on page 614. Give an algorithm that determines whether or not a given undirected graph G = (V, E) contains a cycle. Your algorithm should run in O(V) time, independent of |E|.

Solution:

Back edges are edges (u, v) connecting a vertex u to a vertex v in a depth-first tree. An undirected graph is acyclic (i.e., a forest) iff a DFS yields no back edges.

1. If there’s a back edge, there’s a cycle.
2. If there’s no back edge, then by Theorem 22.10, there are only tree edges, so there is no cycle, the graph is acyclic.

Then we can run DFS as following:

1. if we find a back edge, there exists a cycle.
2. The complexity is O(V) but not O(E+V). Because if there was a back edge, it must be found before seeing |V| distinct edges, because in an acyclic (undirected) forest, |E| ≤ |V| - 1. The number of edges considered is at most the number of vertices considered, thus the total runtime is O(|V|).
3. Exercise **22.5-5** on page 620. Give an O (V+E)-time algorithm to compute the component graph of a directed graph G = (V, E). Make sure that there is at most on edge between two vertices in the component graph your algorithm produces.

Solution:

For given problem, we can compute the set of vertices in each of the strongly connected components (SCC). For each vertex in the component graph, vSCC denotes the SCC that v belongs to. Then for each edge (u, v) in the original graph, we add an edge from uSCC to vSCC if one does not exist yet. The procedure takes a time of O(V+E). From that point, we just spend constant time checking the existence of an edge in the component graph and adding one if needed.

SUEDO CODE:

ConstructComponentGraph(G)

//v denotes vertices

// List is a list in descending order of f (finishing time)

//adj is an Adjacency List

//SCC denotes strongly connected components

//BLACK denotes edge(v, x) is the edge of SCC

DFS(G) // O(V+E)

If v is finish THEN List.add\_front(v)

FOR each v in V // O(V+E)

For each x in v.adj

x.adj.add\_back(v)

DFS(GT) // O(V+E)

FOR v in List

map v to SCC

For each x in v.adj

if x.color = BLACK

if x does not in SCC.adj

SCC.adj.add\_back(x)

Hence the total time complex is O(V+E)