Stephen Woodbury 6/5/17 CMPS 102
I have read and agree to the collaboration policy. Stephen Woodbury.
Collaborators: Hunter Bingam, He was the primary thought lead on this one, I had a more primitive Algorithm that only worked with Graphs where the Vertices could have only up to degree 1. Hunter taught me how to do this, I did not copy anything accept the Proof of correctness for G containing a Cycle Cover, which I now understand.

Assignment 4_2 : Cycle Cover

Algorithm Summary/Synapsis

Set up G'. G' will take G as input, and turn all vertices v into two vertices each, vi and vo. Then an edge will be added from vi and vo. Then a source vertex will be added and will point to all vo's. A sink vertex will be added and will point to all vi's. Set all edge weights to 1. Once this is complete, we now have a bipartite graph modified to run in the Ford Algorithm.

Once we run our ford algorithm, if the flow through the graph is equivalent to the number of vertices in the Graph G (the original Graph G), then we know a cycle cover exists for G represented by the set of edges in G' with f(e) > 0.

Given our flow is equal to |V|, to find our cycle cover, we would iterate through all edges of G' that are not connected to the source or sink that have the form (vi, v'o), ie, it goes from one vertex's out point to another vertex's in point. We would find all edges abiding by these restrictions that also have a flow value of 1 and add them to another graph, G'', which upon termination, will be returned as our subgraph of G containing our cycle covers. If the flow doesn't equal |V|, a message will print saying a cycle cover does not exist.

Proof of Correctness : Termination

Claim: Algorithm terminates after at most [v(f*) <= n] iterations where V(f*) is the value of the max flow and n is the number of nodes in G'.
Pf: Every iteration through the while loop, f increments by one. This is because the while loop only continues to iterate if it has found an augment path, if it has found an augment path, Augment is called, which increases f by 1 before returning. Since we know v(f*) is finite, the algorithm must terminate. v(f*) can't be greater than n intuitively.

Proof of Correctness : Max Flow Found

Claim: Our Algorithm yields the max flow after termination.
Pf: There is a corollary that states that if there is a s-t cut s.t v(f)=Cap(A,B), then f is the max flow of the Graph. If after termination of our algorithm, we took an s-t cut s.t A includes all nodes reachable from s (and s) and B includes all other nodes, the we'd find that our cap(A,B) does indeed equal the value of the flow returned from our algorithm, which makes the flow returned the max flow.

Proof of Correctness : Cycle Cover Found

Claim: Our Algorithm yields a Cycle Cover if one exists
Pf: A cycle cover of a graph is a set of vertex disjoint cycles that cover all vertices. This means that each vertex is part of a cycle exactly once. Thus, a cycle cover of a graph G is a subgraph of G in which each vertex has one incoming edge and one outgoing edge. So, when my algorithm divides

the vertices V into the disjoint sets Vin and Vout and adds the edges (source, Vout) for all v ∈ Vout edges and (Vin, sink) for all v ∈ Vin, each with a capacity of 1, it creates a graph G' which will test whether or not each vertex can have exactly one incoming edge and one outgoing edge.
If there is a perfect bipartite matching in the modified graph G' then there exists a cycle cover of G If there is a perfect bipartite matching in the modified graph, then there will be exactly V flow at the sink. This is because there are exactly V edges in the sets Vin and Vout and each vertex in Vin has an edge connected to the sink with a capacity of 1 and each vertex in vout has an incoming edge from the source with a capacity of 1. Thus there can be at most V flow in graph G' since there are V edges from the source with a capacity of 1. Since all of these edges are incoming edges of the vertices in Vin, this means that each vertex in Vin can contribute one flow to the graph G' . If there is V flow, this means that each vertex in Vout has a flow of value one connecting it to the sink. Since Vin represents the incoming edges of V and Vout represent the outgoing edges of V , if the graph G' has a flow of V , this would mean that there is a subgraph of G in which each vertex has exactly one incoming edge and one outgoing edge. Thus, if there is V flow in G' then there there is a cycle cover of G. 1 Algorithm will find the cycle cover If the flow of G' is less than V, than a cycle cover does not exist. If a cycle cover does exist, then one of the cycle covers will be found. This is because all the edges that originate at a vertex in the set Vout and connect to a vertex in the set Vin are representative of an edge in the original graph G, since that is how the graph G' was created, lets call this set of edges Eorig. These edges all either have a flow of 0 or 1. The set of e ∈ Eorig that have a flow of 1, represent an edge in a subgraph of G that connects the each vertex in such a way that each vertex has exactly one incoming edge and one outgoing edge. Thus, the set of e ∈ Eorig that have a flow of 1 represent a cycle cover of graph G' . Termination Since there is a finite amount of edges and vertices, my algorithm clearly terminates.

<u>Proof of Correctness : Graph Yields a Bipartite Graph</u>

Claim: Our Graph G' is in fact a Bipartite Graph

Pf: It is clear that when dividing the graph into Vin vertices and Vout vertices that the resulting graph G' is bipartite. This is because no two vertices Vin can be connected to each other and no two vertices Vout can be connected to each other. All of the edges are of the type (represented with i and j which can be any two adjacent vertices) (iout, jin). Thus the vertices can be divided into the disjoint sets: in and out.

<u>RunTime Analysis:</u>

Bipartitte Graph Creation : O(m+n), total runtime: ***O(nm)* –** To run Ford Algorithm.

<u>Space Complexity:</u>

O(V'+E'), storing vertices and Edges. V' = Vertices in G', E' = Edges in E'.