# Intro to Algorithms HW 5

Greg Stewart

March 9, 2018

## Q1 Design a linear time algorithm to check if a graph is bipartite or not.

One way to analyze graphs is to color them. Neighboring vertices may not have the same color. In the case of a bipartite graph, only two colors are needed, as the vertices are essentially split into two groups. So the algorithm is as follows:

Store the graph as an adjacency list to ensure linear time. Starting from a vertex $v$, label the vertex as a color, say red, then visit each of its neighbors and label each vertex in $N(v)$ as a different color, say green. Proceed from each of the neighbors to their neighbors, labeling these ones red again. Do this over the graph , from neighbor to neighbor, and when you land on a vertex that has already been colored, compare the color with that of the vertex you just came from—if they are the same, the graph is not bipartite. If they are different for all cases where this happens, the graph is bipartite. To account for all possibilites, after doing this for a connected component, if there are unchecked vertices, move to those and do the described algorithm on them until all vertices have been visited.

## Q2 Answer the following:

(a) *Prove that a non-empty DAG must have at least one source.*

Recall the a DAG has no cycles by definition, and a source has no incoming edges. Let's assume that there is a DAG $G$ where every vertex in the graph has an incoming edge. Then $G^R$ is a DAG where every vertex has an outgoing edge, which means if one follows the edges from some vertex in the graph, the path would make a cycle in at most $|V|$ steps. But this is a contradiction—there should not be a cycle in an acyclic graph. Therefore $G$ must have a vertex with all outgoing edges—$G$ must have a source.

(b) *What is the time complexity of finding a source in a directed graph or to determine such a source does not exist, if using adjacency matrix? Describe the algorithm.*

In an adjacency matrix, a source vertex's column is full of 0's, since there are no incoming edges. So to find a source or determine whether one does not exist, each column of the adjacency matrix must be checked in its entirety for nonzero terms. Looking through each column in the matrix is $O(|V|^2)$ since there are $|V|$ vertices and each column is size $|V|$.

(c) *Same questions as (b), but using adjacency list.*

Assume an adjacency list where each vertex has a list of outgoing edges. Initialize a boolean array of size $|V|$ to all True, so each element of the array represents a vertex in the graph. Go over the list of outgoing edges of the first node—each vertex to which the first vertex has an outgoing edge cannot be a source, since it has incoming edges, so set the respective values in the array to False. Now move through each vertex in the graph and do the same thing for every one. At the end, only the vertices corresponding to elements in the array that remained True are sources in the graph. Each vertex is visited and each edge checked, so this algorithm is $O(|V| + |E|)$.

## Q3 Describe a linear time algorithm to compute the neighbor degree for each vertex in an undirected graph.

This algorithm uses an adjacency list representation of a graph: an array of vertex labels which points to a list of its neighbors. First, scan the graph once to store the degree of each vertex. Create an array of size $|V|$

to store the degrees of each corresponding vertex, then starting at the first vertex in the adjacency list, store the size of the list of neighbors in the array of degrees, for each vertex in the graph. Then, create another array of size $|V|$ to store the neighbor degrees with all values initialized to 0, and for each vertex $v$ in the graph, loop through its list of neighbors $N(v)$ and look up each of the neighbor vertex's degree, adding this value to the value stored at the location in the neighbor degree array corresponding to $v$.

**Q4 Consider a directed graph that has a weight $w(v)$ on each vertex $v$. Define reachability weight:**

$$r(v) = \max\{w(u) \mid u \text{ is reachable from } v\}$$

(a) *Assume the graph is a DAG. Describe a linear time algorithm to compute the reachability weight for all vertices.*

If the graph is stored as an adjacency list, then create the array $r$ of size $|V|$, and find the sources of the graph as previously described, and start at one. Visit each of the nodes it has an outgoing edge to, and keep track of the maximum weight of each, storing that value as $r(v)$. Then, recursively do this through each node that the source directs to, until you're unable to find a new node. Do this from each source in the graph to ensure all nodes are visited.

(b) *Assume the graph is a general directed graph (possible cycles). Describe a linear time algorithm to find reachability weight for all vertices.*

Again use an adjacency list. Start from any vertex $v$ and loop through its outgoing edges to find the maximum weight of its outgoing neighbors, and store this in $r(v)$. Recurse over each of the outgoing neighbors, and their neighbors, etc. To account for possible cycles, keep a bool array of visited nodes, and once visited set the corresponding element in the array to True. When only previously visited nodes are available, find unvisited nodes from the bool array and begin again there.