CS 473: Fundamental Algorithms, Fall 2011

Homework 1 (due Tuesday, 23:55:00, September 6, 2011)

This homework contains three problems. **Read the instructions for submitting the homework on the course webpage. Read the course policies before starting the homework.**

Collaboration Policy: For this homework, Problems 1–3 can be worked in groups of up to three students.

Each student individually have to also do **quiz 1** online.

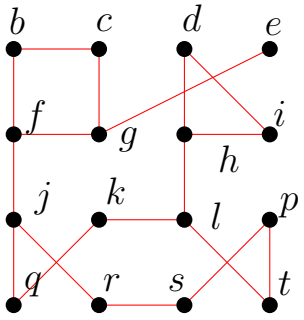
Submission guidelines:

For every group of students, one and only one student have to upload the solutions to the homework to the class moodle page. Together with the solutions to the three questions, the submitting student of the group also have to upload a text file that contains the netid of the submitting students. Specifically, this text file should be named `group.txt`. Each student netid (and nothing else) would be written as is on each line of this text file. As such, an example such file might look like:

```
sariel
jeff7847
mogibogi723
```

Version: **2.11**

1. (45 PTS.) Like a bridge over undirected graph, I will lay me down.

   Given a connected *undirected* graph $G = (V, E)$, an edge $e = (u, v)$ is called a ***bridge***, or a ***cut-edge***, if removing e disconnects the graph into two pieces, one containing u and the other containing v. A vertex u is called a ***separating vertex***, or ***cut-vertex***, if removing u leaves the graph into two or more disconnected pieces; note that u does not count as one of the pieces in this definition. Your goal in this problem is to develop a linear time algorithm to find *all* the bridges of a given graph using **DFS**. Let T be a **DFS** tree of G (note that it is rooted at the first node from which **DFS** is called). For a node v we will use the notation T_v to denote the sub-tree of T hanging at v (T_v includes v).

   (A) (2 PTS.) In the graph shown above, identify all the bridges and cut-vertices (i.e., list all the bridge edges and cut vertices).

   (B) (3 PTS.) Prove that any bridge of G has to be a tree edge in the tree generated by every **DFS**(G). Recall the property of **DFS** in undirected graphs. Why does this show that

the maximum number of bridges is $n-1$? What is a graph that achieves this bound?

(C) (5 PTS.) Suppose $e=(u,v)$ is a tree-edge in **DFS**(G) with $\text{pre}(u) < \text{pre}(v)$ (that means u is the ancestor of v). Prove that e is a bridge if and only if (need to prove both directions) there is no edge from any node in T_v to either u or any of its ancestors.

(D) (5 PTS.) For each node u define:

$$\text{low}(u) = \min \begin{cases} \text{pre}(u) \\ \text{pre}(w) \text{ where } (v,w) \text{ is a back edge for some descendant } v \text{ of } u. \end{cases}$$

Give a linear time algorithm that computes the low value for all nodes by adapting **DFS**(G). Give the altered pseudo-code of **DFS**(G) to do this. There is no need to prove that your code is correct.

(E) (5 PTS.) Give a linear time algorithm that identifies *all* the bridges of G using the low values and the steps above. Specifically, provide pseudo-code for a linear time algorithm to do so. There is no need to prove that your code is correct.

(F) (5 PTS.) Prove that if a graph G contains a bridge, then in any orientation of its edges, the resulting directed graph is not strongly connected.

(G) (20 PTS.) Provide a linear time algorithm that given an undirected graph G, orient its edges such that the resulting directed graph is strongly connected. If no such orientation is possible, the algorithm outputs "not possible:". (Hint: Think how to use (F).)

(It is instructive to run **DFS**(G) on the example graph and compute the pre values and the lowvalues for each node.)

2. (35 PTS.) A party in Vogsphere.

In Vogsphere there are n families. Every family has two children, and you have to organize a party inviting one child from each family (you can not invite both children in a family, because they would start fighting and just ruin the party for everybody). If you fail to invite a child from each family, the skipped family is going to get very upset, and read poetry to you, which is supposedly fatal[1]. For every child, there is a list of its friends (i.e., other children) that must also be invited to the party (otherwise they would not come). Your task is to decide if such a party can be organized.

For social reasons that are not well understood yet, for any two children x and y, if x insist on inviting y, then the sibling of y insist on inviting the sibling of x.

(A) (10 PTS.) Build a graph with $2n$ nodes (one for each child). Put an edge (x,y) if x would come only if you also invite y. Consider the resulting graph G. Show, that if two children of the same family are in the same strong connected component of G, then there is no feasible party.

(B) (15 PTS.) Show (or even better, prove) the converse of (A); that is, show that if no strong connected component in G contains two children from the same family, then there are n invitations that would result in a valid party. (Hint: Consider the graph of connected components and consider its sink. This sink corresponds to a set of children – invite them, do some clean up [i.e., delete children that can no longer be invited], and repeat).

[1] See `http://en.wikipedia.org/wiki/Vogon#Poetry`

(C) (10 PTS.) Provide a linear time algorithm (in n and m [the number of edges in the graph G]) to compute if there is a feasible party, and if so computes the set of invitations to this party.

3. (20 PTS.) Almost strongly connected.

   You are given a DAG G that has the property that one can flip one edge in G and it becomes a strongly connected graph.

   (a) (10 PTS.) Prove that there can not be two sources in the graph G. Similarly, show that there can not be two sinks in G.

   (b) (5 PTS.) Give a concrete description of the edge that must be flipped to get a strongly connected graph.

   (c) (5 PTS.) Give a linear time algorithm (i.e., $O(n + m)$) for computing which edge to flip in G to get the strongly connected graph.