HW 1 (due Monday, at noon, February 2, 2015)

OLD CS 473: Fundamental Algorithms, Spring 2015 Version: **1.2**

You also have to do quiz 1 online (on moodle).

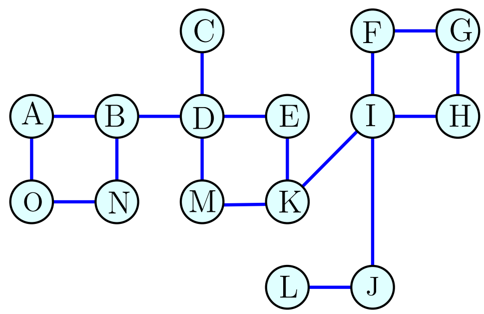
Collaboration Policy: For this homework, Problems 1–3 can be worked in groups of up to three students.

---

1. (40 PTS.) A bridge to nowhere.

Given a connected *undirected* graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$, an edge $e = (u, v)$ is 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 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 and cut-vertices 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 (includes v).

(A) In the graph shown in the figure, identify all the bridges and cut-vertices.



(B) Prove that any bridge of G has to be a tree edge in every **DFS**(G). Prove that the maximum number of bridges in G is $n - 1$, and provide an example realizing this bound.

(C) Suppose $e = (u, v)$ is a tree-edge in **DFS**(G) with $pre(u) < pre(v)$. Prove that e is a bridge if and only if there is no edge from any node in T_v to either u or any of its ancestors.

(D) For each node u define:

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

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.

(E) 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) Prove that the root of the DFS tree is a cut-vertex if and only if it has two or more children.

(G) Prove that a non-root vertex u of the DFS tree T is a cut-vertex if and only if it has a child v such that no node in T_v has a backedge to a *proper* ancestor of u (that is, an ancestor of u which is not u itself).

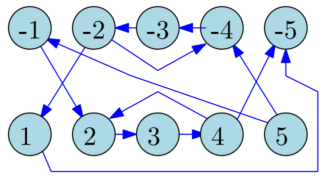
(H) The above two properties can be used to find all the cut-vertices in linear time. Give the pseudo-code for a linear time algorithm to do so. There is no need to prove that your code is correct.

2. (30 PTS.) Partitioning numbers.

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be a **directed graph** with $2n$ vertices: $\mathsf{V} = \{1, \ldots, n, -1, -2, \ldots, -n\}$. This graph has the property that if the edge (u, v) is in the graph, then $(-v, -u)$ is also in the graph. Our purpose is to pick a set X of n vertices in the graph, such that:

(I) There is no directed edge from a vertex of X to a vertex of $\mathsf{V} \setminus X$.
(II) There is no i such that both i and $-i$ are in X.
(III) $|X| = n$.

As an example, consider the following graph:



(A) (5 PTS.) Prove that if i and $-i$ are in the same strong connected component of G, then there is no such partition.
(B) (5 PTS.) Consider a strong connected component $S = \{s_1, \ldots, s_k\}$ of G. Prove that $-S = \{-s_1, \ldots, -s_k\}$ is also a strong connected component of this graph.
(C) (5 PTS.) Prove that if S is a strong connected component of G that is a sink in the meta graph $\mathsf{G}^{\mathrm{SCC}}$, then $-S$ is a source in the meta graph $\mathsf{G}^{\mathrm{SCC}}$.
(D) (5 PTS.) Describe a linear time algorithm that decides if there is a number i such that both i and $-i$ are in the same strong connected component of G.
(E) (10 PTS.) Describe an algorithm that in linear time decides if the desired partition exists, and if it exists it outputs it. Prove the correctness of your algorithm.

3. (40 PTS.) Profitable path.

Consider a DAG G with n vertices and m edges. Each vertex v of G corresponds to a project, with profit p_v (which might be negative, if it is a losing project). A vertex v is ***profitable*** if $p_v > 0$.

(A) (10 PTS.) Show an algorithm that in linear time computes all the vertices that can reach a sink of G via a path that goes through at least one profitable vertex.
(B) (10 PTS.) Show an algorithm that in linear time computes all the vertices that can reach a sink of G via a path that goes through at least β profitable vertices, where β is a prespecified parameter.
(C) (10 PTS.) Show an algorithm, as fast as possible, that computes for all the vertices v in G the most profitable path from v to any sink of G. The ***profit*** of a path is the total sum of the profits of vertices along the path.
(D) (10 PTS.) Using the above, describe how to compute, in linear time, a path that visits all the vertices of G if such a path exists.