"CS 374" Fall 2015 — Homework 7

Due Tuesday, October 27, 2015 at 10am

••• Some important course policies •••

- **You may work in groups of up to three people.** However, each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the names and NetIDs of each person contributing.
- **You may use any source at your disposal**—paper, electronic, or human—but you ***must*** cite ***every*** source that you use. See the academic integrity policies on the course web site for more details.
- **Submit your pdf solutions in Moodle.** See instructions on the course website and submit a separate pdf for each problem. Ideally, your solutions should be typeset in LaTex. If you hand write your homework make sure that the pdf scan is easy to read. Illegible scans will receive no points.
- **Avoid the Three Deadly Sins!** There are a few dangerous writing (and thinking) habits that will trigger an automatic zero on any homework or exam problem. Yes, we are completely serious.
  - Give complete solutions, not just examples.
  - Declare all your variables.
  - Never use weak induction.
- Unlike previous editions of this and other theory courses we are not using the "I don't know" policy.

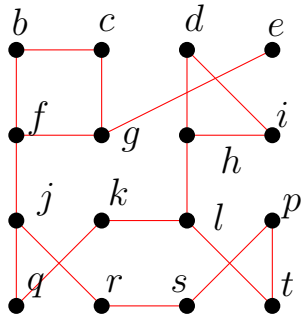
See the course web site for more information.

If you have any questions about these policies,
please don't hesitate to ask in class, in office hours, or on Piazza.

---

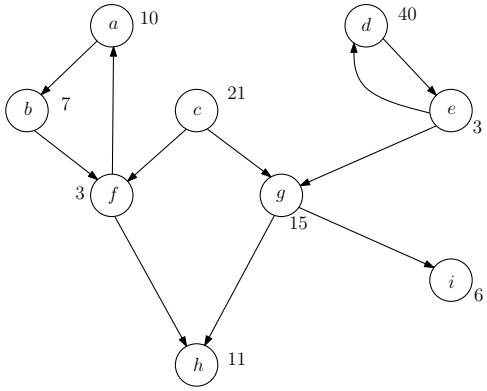
1. Given an undirected connected graph $G = (V, E)$ an edge (u, v) is called a cut edge or a bridge if removing it from G results in two connected components (which means that u is in one component and v in the other). The goal in this problem is to design an efficient algorithm to find *all* the cut-edges of a graph.

   - What are the cut-edges in the graph shown in the figure?



   - Given G and edge $e = (u, v)$ describe a linear-time algorithm that checks whether e is a cut-edge or not. What is the running time to find all cut-edges by trying your algorithm for each edge? No proofs necessary for this part.
   - Consider *any* spanning tree T for G. Prove that every cut-edge must belong to T. Conclude that there can be at most $(n - 1)$ cut-edges in a given graph. How does this information improve the algorithm to find all cut-edges from the one in the previous step?
   - Suppose T is a spanning tree of G rooted at r. Prove that an edge (u, v) in T where u is the parent of v is a cut-edge iff there is no edge in G with one end point in T_v (sub-tree of T rooted at v) and one end point outside T_v.
   - Use the property in the preceding part to design a linear-time algorithm that outputs all the cut-edges of G. You don't have to prove the correctness of the algorithm but you should point out how your algorithm ensures the desired property.

2. Let $G = (V, E)$ be a directed graph. Each vertex $v \in V$ has a weight $w(v)$ associated with it. Given a vertex $s \in V$ let $\alpha(s) = \max\{w(v) \mid v \text{ can reach } s \text{ in } G\}$ be the maximum weight among the weights of all nodes that can reach s in G. In the figure below $\alpha(b) = 21$ and $\alpha(i) = 40$.



Describe an efficient algorithm that given a graph G and the weights $w(v), v \in V$, computes $\alpha(s)$ for every $s \in V$. To this end first consider the case when G is strongly connected and when G is a DAG. Full credit for a *linear-time* algorithm.

A formal proof is not needed but you need to give an explanation of your algorithm and why it achieves the desired running time.

3. Let $G = (V, E)$ be a directed graph with non-negative edge lengths $\ell(e), e \in E$. You are also given a subset of the nodes $X \subset V$ that are called *dangerous*. Given two nodes s, t describe an efficient algorithm to find a shortest path from s to t among all paths that contain at most one dangerous node. You can assume that s and t are not dangerous. Ideally the algorithm should run in time proportional to one single-source shortest path computation.