CS 573: Algorithms, Fall 2014
Homework 1, due Monday, September 15, 23:59:59, 2014

Version 1.01

Name:	
Net ID:	Alias:

Neatly print your name(s), NetID(s). Staple this sheet to the top of your homework. If you are on campus, submit the homework by submitting it in the homework boxes in the basement of SC.

Note: You will be held accountable for the appropriate responses for answers (e.g. give models, proofs, analysis, etc). For NP-Complete problems you should prove everything rigorously, i.e. for showing that it is in NP, give a description of a certificate and a polynomial time algorithm to verify it, and for showing problems are NP-Hard, you must show that your reduction is polynomial time (by similarly proving something about the algorithm that does the transformation) and proving both directions of the 'if and only if' (a solution of one is a solution of the other) of the many-one reduction.

You are *encouraged* to discuss problems in this homework with people, but should submit your homework in groups of size at most three (this is a hard limit). You must submit a printed copy, and an electronic copy on moodle.

Only of myself I know how to tell,
my world is as narrow as an ant's.
like an ant too my burden I carry,
too great and heavy for my frail shoulder.

My way too - like the ant's to the treetop -
is a way of pain and toil;
a gigantic hand, assured and malicious,
a mocking hand hinders

All my paths are made bleak and tearful
by the constant dread of this giant hand.

Why do you call to me, wondrous shores?
Why do you lie to me, distant lights?
– Only of Myself, Rachel

Required Problems

1. Poly time subroutines can lead to exponential algorithms. (10 pts.)
Show that an algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time, but that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.

2. Beware of algorithms carrying oracles. (70 pts.)
Consider the following optimization problems, and for each one of them do the following:

(I) (2 pts.) State the natural decision problem corresponding to this optimization problem.

(II) (3 PTS.) Either: (A) prove that this decision problem is NP-COMPLETE by showing a reduction from one of the NP-COMPLETE problems seen in class (if you already seen this problem in class state "seen in class" and move on with your life). (B) Alternatively, provide an efficient algorithm to solve this problem.

(III) (5 PTS.) Assume that you are given an algorithm that can solve the decision problem in polynomial time. Show how to solve the original optimization problem using this algorithm in polynomial time.

Prove that the following problems are NP-COMPLETE.

(A) (10 PTS.)

NOT SET COVER

Instance: Collection $\mathcal{C}$ of subsets of a finite set S.
Target: Compute the maximum k, and the sets $S_1, \ldots, S_k$ in $\mathcal{C}$, such that $S \not\subseteq \cup_{i=1}^{k} S_i$.

(B) (10 PTS.)

MAX BIN PACKING

Instance: Finite set U of items, size $s(u) \in \mathbb{Z}^+$ for $u \in U$, an integer bin capacity B.
Target: Compute the maximum k, and a partition of U into disjoint sets $U_1, \ldots, U_k$, such that the sum of the sizes of the items inside each U_i is B or more.

(C) (10 PTS.)

DOUBLE HITTING SET

Instance: A ***ground set*** $U = \{1, \ldots, n\}$, and a set $\mathcal{F} = \{U_1, \ldots, U_m\}$ of subsets of U.
Target: Find the smallest set $S' \subseteq U$, such that S' hits all the sets of $\mathcal{F}$ at least twice. Specifically, $S' \subseteq U$ is a ***double hitting set*** if for all $U_i \in \mathcal{F}$, we have that S' contains at least two element of U_i.

(D) (10 PTS.)

Min Leaf Spanning Tree

Instance: Graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$.
Target: Compute the spanning tree T in G where the number of vertices in T of degree one is minimized.

(E) (10 PTS.)

Cover by paths (edge disjoint).

Instance: Graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$.
Target: Compute the minimum number k of paths $\pi_1, \ldots, \pi_k$ that are edge disjoint, and their union cover all the edges in G.

(F) (10 PTS.)

Cover by paths (vertex disjoint).

Instance: Graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$.
Target: Compute the minimum number k of paths $\pi_1, \ldots, \pi_k$ that are vertex disjoint, and their union cover all the vertices in G.

(G) (10 PTS.)

Partition graph into not so bad, and maybe even good, sets (PGINSBMEGS).

Instance: Graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ and k.
Target: Compute the partition of V into k sets $\mathsf{V}_1, \ldots, \mathsf{V}_k$, such that the number of edges uv of G that have distinct indices i and j, such that $u \in \mathsf{V}_i$, and $v \in \mathsf{V}_j$ is maximized.

3. Independence. (20 PTS.)

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be an undirected graph over n vertices. Assume that you are given a numbering $\pi : V \to \{1, \ldots, n\}$ (i.e., every vertex have a unique number), such that for any edge $ij \in \mathsf{E}$, we have $|\pi(i) - \pi(j)| \leq 20$.
Either prove that it is NP-Hard to find the largest independent set in G, or provide a polynomial time algorithm.