CS 170 HW 11

Due **2020-11-16, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer “Yes”, “Yes but anonymously”, or “No”

2 Path TSP and Cycle TSP

This is a solo question.

In the Traveling Salesman Problem (TSP), we are given an undirected graph G with non-negative weights and asked to find a minimum weight cycle that visits each vertex exactly once.

In the s-t Traveling Salesman Problem (s-t TSP), we are given an undirected graph G with non-negative weights, two vertices s and t, and are asked to find a minimum weight path that starts at s, visits all other vertices exactly once, and ends at t.

(a) Consider the following reduction from s-t TSP to TSP: Take G and add a weight 0 edge between s and t to get a new graph G'. Show that if a s-t TSP solution of weight w exists in G, then a TSP solution of weight w exists in G'.

(b) Despite what you proved in part (a), why is the reduction in part (a) not valid?

(c) Give a valid reduction from s-t TSP to TSP. Prove correctness for your reduction. No runtime analysis needed.

3 SAT and Integer Programming

Consider the 3SAT problem, where the input is a set of clauses and each one is a OR of 3 literals. For example, $(x_1 \vee \overline{x}_4 \vee \overline{x_7})$ is a clause which evaluated to true iff one of the literals is true. We say that the input is satifiable if there is an assignment to the variables such that all clauses evaluate to true. We want to decide whether the input is satifiable.

On the other hand, consider the integer linear programming feasibility problem: We are given a set of variables and constraints in terms of these variables (we are not given an objective). The constraints are either linear inequalities, or the 0-1 constraints $x_i \in \{0, 1\}$. We want to decide if it possible to assign the variables values that satisfy all the constraints.

Give a reduction from 3SAT to integer linear programming feasibility, and briefly justify its correctness. No runtime analysis needed.

4 Convex Hull

Given n points in the plane such that no three points are collinear, the *convex hull* is the list of points, in counter-clockwise order, that describe the convex polygon that contains all the other points. Imagine a rubber band is stretched around all of the points: the set of points it touches is the convex hull. You can also play around with defining your own set of points and seeing what the polygon should look like at `http://cs.yazd.ac.ir/cgalg/AlgsVis/ConvexHull.html`

In this problem we'll show that the convex hull problem and sorting reduce to each other in linear time.

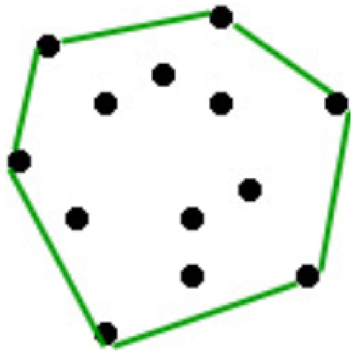

Figure 1: An instance of convex hull: the convex hull is the six points connected by green lines.

(a) Fill in the following algorithm for convex hull; you do not need to prove it correct. What is its runtime?

```
procedure ConvexHull(list of points $P[1..n]$)
    Set $low$ := the point with the minimum $y$-coordinate, breaking ties by minimum $x$-coordinate.
    Create a list $S[1..n-1]$ of the remaining points $p \in P$ sorted increasingly by the angle between the vector $p - low$ and the vector $(1, 0)$ (i.e the x-axis) .
    Initialize $Hull$ := [$low$, $S[1]$]
    for $p \in S[2..n-1]$ do
        <fill in the body of the loop>
    Return $Hull$
```

(b) Now, find a linear time reduction from sorting to convex hull. In other words, given a list of real numbers to sort, describe an algorithm that transforms the list of numbers into a list of points, feeds them into convex hull, and interprets the output to return the sorted list. Then, prove that your reduction is correct.

(Note that your reduction should not create three points that are collinear, per the definition of the convex hull problem. Hint: For each number a in the list, create a point $(a, f(a))$, where $f(a)$ is some simple function of a. Your choice of f should ensure that every point is in the convex hull.)

5 Hitting Set

In the Hitting Set Problem, we are given a family of finite integer sets $\{S_1, S_2, \ldots, S_n\}$ and a budget b, and we wish to find an integer set H of size $\leq b$ which intersects every S_i, if such an H exists. In other words, we want $H \cap S_i \neq \emptyset$ for all i.

Show that the Hitting Set Problem is NP-complete. (Hint: Hitting Set generalizes one of the problems covered in Chapter 8 of the textbook).

6 NP Basics

This is a solo question.

Assume A reduces to B in polynomial time. In each part you will be given a fact about one of the problems. What information can you derive of the other problem given each fact?

1. A is in **P**.
2. B is in **P**.
3. A is **NP**-hard.
4. B is **NP**-hard.

7 Upper Bounds on Algorithms for NP Problems

Parts a and c of this problem are solo questions. You may collaborate on part b.

(a) Recall the 3-SAT problem: we have n variables x_i and m clauses, where each clause is the OR of at most three literals (a literal is a variable or its negation). Our goal is to find an assignment of variables that satisfies all the clauses, or report that none exists.

Give a $O(2^n m)$-time algorithm for 3-SAT. Just the algorithm description is needed.

(b) Using part (a) and the fact that 3-SAT is NP-hard, give a $O(2^{n^c})$-time algorithm for every problem in NP, where c is a constant (that can depend on the problem). Just the algorithm description and runtime analysis is needed. An informal algorithm description is fine.

(This result is known as $NP \subseteq EXP$.)

(c) Recall the halting problem from CS70: Given a program (as e.g. a .py file), determine if the program runs forever or eventually halts. Also recall that there is no finite-time algorithm for the halting problem. Let us define the input size for the halting problem to be the number of characters used to write the program.

Given an instance of 3-SAT with n variables and m clauses, we can write a size $O(n+m)$ program that halts if the instance is satisfiable and runs forever otherwise. So there is a polynomial-time reduction from 3-SAT to the halting problem.

Based on this reduction and part (b): Is the halting problem NP-hard? Is it NP-complete? Justify your answer.

8 (Extra Credit) Orthogonal Vectors

In the 3-SAT problem, we have n variables and m clauses, where each clause is the OR of (at most) three of these variables or their negations. The goal of the problem is to find an assignment of variables that satisfies all the clauses, or correctly declare that none exists.

In the orthogonal vectors problem, we have two sets of vectors A, B. All vectors are in $\{0,1\}^m$, and $|A| = |B| = n$. The goal of the problem is to find two vectors $a \in A, b \in B$ whose dot product is 0, or correctly declare that none exists. The brute-force solution to this problem takes $O(n^2 m)$ time: We compute all $|A||B| = n^2$ dot products between two vectors in A, B, and each dot product takes $O(m)$ time.

Show that if there is a $O(n^c m)$-time algorithm for the orthogonal vectors problem for some $c \in [1, 2)$, then there is a $O(2^{cn/2} m)$-time algorithm for the 3-SAT problem. For simplicity, you may assume in 3-SAT that the number of variables must be even.