SI 335 Spring 2013: Problem Set 4

Your name:

Due: Tuesday, April 30

Instructions: Review the course honor policy: informal *discussions* are permitted, but no *written collaboration* of any kind..

This cover sheet must be the front page of what you hand in. Fill out the left column in the table to the right before you hand it in.

Use separate paper for your solutions and make sure they are submitted in order — please!.

Comments or suggestions about this problem set:

Comments or suggestions about the course so far:

I had discussions with:

Citations (be specific about websites):

Grading rubric:

- **5**: Solution is completely correct, concisely presented, and neatly written.
- **4**: The solution is mostly correct, but one or two minor details were missed, or the presentation could be more concise.
- **3**: The main idea is correct, but there are some significant mistakes. The presentation is somewhat sloppy or confused.
- **2**: A complete effort was made, but the result is mostly incorrect. There may be some basic misunderstandings of the topic or the problem.
- **1**: The beginning of an attempt was made, but the work is clearly incomplete.
- **0**: Not submitted.

Problem	Self-assessment	Final assessment
1		
2		

0.1 Grading

There are two problems in this project. They both have 7 parts (a)-(g), which are completely identical, except that the set-up problems are different. **Each part of each problem is worth a maximum of 10 points**. That, and the fact that **you cannot receive credit for both 1(g) and 2(g)**, means that there are a total of **130 possible points**. The project will be graded on a scale of 100 points = 100 percent. Therefore you do not have to attempt every part, but you may if you want to go for extra credit.

0.2 Known NP-Complete Problems

For your reference, here is a list of the **NP**-Complete Decision Problems that were presented in class. In any of your solutions, you may use the fact that these problems are **NP**-complete.

- `LONGPATH(G,u,v,k)`
  Input: Graph $G = (V, E)$, vertices u and v, integer k
  Output: Does G contain a path from u to v of length at least k?
- `VC(G,k)`
  Input: Graph $G = (V, E)$, integer k.
  Output: Does G have a vertex cover (subset of V) containing at most k vertices?
- `HITSET(L,k)`
  Input: List L of sets $S_1, S_2, \ldots, S_m$, and an integer k
  Output: Is there a "hitting set" H with size at most k such that H contains at least one member of every set S_i?
- `HAMCYCLE(G)`
  Input: Graph $G = (V, E)$
  Output: Des G contain a cycle (path with same starting and ending vertex) that touches every node exactly once?
- `CIRCUIT-SAT(C)`
  Input: Boolean circuit C with m inputs and one output
  Output: Is there a setting of the m inputs to True/False that makes the output stabilize to True?
- `3-SAT(F)`
  Input: Boolean formula F in conjunctive normal form (product of sums) with three literals in every clause
  Output: Does F have a "satisfying assignment" (setting of every variable to True/False so that the entire formula is True)?
- `SPLIT-EVENLY(S,k)`
  Input: Set S of integers
  Output: Can S be partitioned into two subsets A and B such that difference between the sums of the numbers in A and B is at most k? In other words, $A \cup B = S$, $A \cap B = \{\}$, and $|(\sum_{a \in A} a) - (\sum_{b \in B} b)| \leq k$: is this possible?

1 Hungry Hungry Mids

This question is about the following *computational* problem:

King Hall has a bunch of random leftover food items: a single hamburger patty, a bottle of ketchup, a bowl of mashed potatoes, a dill pickle, etc. Each leftover food item has a certain number of calories in it. The question is how many complete meals can be made from these leftover items, with the only restriction being that each "meal" must contain at least a certain number of calories.

Formally, the problem is defined as follows:

> `COMPUTE-MAX-MEALS(L,k)`
> **Input**: List L of integers, and a single integer k. Each integer in L is between 1 and $k-1$.
> **Output**: A partition of L into r subsets $M_1, M_2, \ldots, M_r$ such that the sum of the numbers in each M_i is at least k, and the number of subsets r is as large as possible.

For example, if $L = (5, 3, 3, 8, 6, 10, 11, 5, 7, 4)$ and $k = 20$, then an optimum solution has $r = 3$ and the subsets are $M_1 = (10, 8, 3)$, $M_2 = (11, 5, 5)$, and $M_3 = (7, 4, 6, 3)$.

a) Describe a *greedy* algorithm to solve the `COMPUTE-MAX-MEALS` problem. This should be a fairly simple idea.

b) Give a counterexample to show that your greedy algorithm does not always produce the best possible solution. That is, you give a sample input L, show what your greedy algorithm would produce, and then come up with an even better solution which demonstrates the greedy algorithm is not optimal.

c) Come up with a *decision version* of this problem. Call your decision problem `MEALS`.

   Remember that your decision problem should be roughly the same difficulty as the original problem, up to polynomial-time reductions. In particular, you have to choose your decision problem carefully so that parts (e) and (f) below are at least *possible* to prove (even if you don't prove them yourself).

   You are encouraged to e-mail your idea for this part to your instructor if you're not sure, before proceeding.

d) Show that your `MEALS` problem is in **NP**, using the steps we went over in class for an **NP** proof.

e) Present a polynomial-time reduction from your `MEALS` problem to `COMPUTE-MAX-MEALS`.

f) Show that your `MEALS` problem is **NP**-hard by presenting a reduction from a known **NP**-complete problem to your `MEALS` problem. You may use any of the **NP**-complete problems in the list at the beginning of this assignment for your reduction.

   (Hint: what other decision problem involves the sums of numbers in different subsets?)

g) State what we know about your `MEALS` problem and the `COMPUTE-MAX-MEALS` problem, assuming (d) through (f) have been proven.

2 Party Planner

This question is about the following *computational* problem:

You are planning a party and want to invite a bunch of your friends. Unfortunately, some of your friends and acquaintences don't get along with each other, and bad things will happen if they both show up for the party. So, given the histories of bad blood among your friends, you want to invite the largest group of friends possible to your party, without inviting any two people that don't get along.

Formally, the problem is defined as follows:

> `COMPUTE-MAX-PARTY(F,E)`
> **Input**: A list of friends F, and a list of pairs of enemies E, each pair containing two elements from F.
> **Output**: A subset of P of F, as large as possible, such that no two elements in P are enemies, i.e., for every pair in E, at most one of the pair is in P.

For example, if $F = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 3), (2, 3), (1, 5), (4, 5)\}$, then an optimum solution is $P = \{1, 2, 4\}$.

a) Describe a *greedy* algorithm to solve the `COMPUTE-MAX-PARTY` problem. Your algorithm should be fairly simple.

b) Give a counterexample to show that your greedy algorithm does not always produce the best possible solution.

c) Come up with a *decision version* of this problem. Call your decision problem `PARTY`.

   Remember that your decision problem should be roughly the same difficulty as the original problem, up to polynomial-time reductions. In particular, you have to choose your decision problem carefully so that parts (e) and (f) below are at least *possible* to prove (even if you don't prove them yourself).

   You are encouraged to e-mail your idea for this part to your instructor if you're not sure, before proceeding.

d) Show that your `PARTY` problem is in **NP**, using the steps we went over in class for an **NP** proof.

e) Present a polynomial-time reduction from your `PARTY` problem to `COMPUTE-MAX-PARTY`.

f) Show that your `PARTY` problem is **NP**-hard by presenting a reduction from a known **NP**-complete problem to your `PARTY` problem. You may use any of the **NP**-complete problems in the list at the beginning of this assignment for your reduction.

   (Hint: think about choosing the minimal number of people that *won't* be invited to the party. What decision problem(s) involve finding the smallest possible something?)

g) **NO CREDIT GIVEN IF YOU ALREADY ANSWERED 1(g).**

   State what we know about your `PARTY` problem and the `COMPUTE-MAX-PARTY` problem, assuming (d) through (f) have been proven.