SI 335 Spring 2015: Problem Set 4

Due: Wednesday, April 29

Your scheduled presentation time:

Group member:

Group member:

Group member:

Group member:

Instructions: Review the course honor policy: you may not use any human sources outside your group, and must document anything you used that's not on the course webpage.

This cover sheet must be the front page of what you hand in. Use separate paper for the your written solutions outline and make sure they are neatly done and in order. Staple the entire packet together.

Comments or suggestions about this problem set:

Comments or suggestions about the course so far:

Citations (be specific about websites):

Grading rubric:

A: Solution meets the stated requirements and is completely correct. Presentation is clear, confident, and concise.
B: The main idea of the solution is correct, and the presentation was fairly clear. There may be a few small mistakes in the solution, or some faltering or missteps in the explanation.
C: The solution is acceptable, but there are significant flaws or differences from the stated requirements. Group members have difficulty explaining or analyzing their proposed solution.
D: Group members fail to present a solution that correctly solves the problem. However, there is clear evidence of significant work and progess towards a solution.
F: Little to no evidence of progress towards understanding the problem or producing a correct solution.

Problem	Final assessment
1	
2	
3	
4	

0.1 Special rules for this problem set

1. You need to present and turn in your own work, alone, but you can have informal discusion with classmates. You must clearly document which problems you discussed, and with whom! And remember, anything written down in your discussion has to be thrown away.
2. Since there are four problems, and you are in a group of 1, you may skip one of the four problems.
3. You need to turn in a complete, written solution to all three problems you do. We may not have time to get to all of them during the presentation time.

0.2 Problems 1-3

The first three problems on this set all ask you to do the same thing, but for different computational problems. For each computational problem, you need to do the following:

- Define a decision problem version of the computational problem given.
- Show that your decision problem is **NP**-complete. (This means first proving it's in **NP**, then proving it's **NP**-hard.)
- Show that the original computational problem is also **NP**-hard.
- Come up with a strategy to solve the original computational problem, to get an optimum solution. Of course your algorithm will not be polynomial-time! But for full credit, your algorithm should be faster than brute-force. You might try and come up with a good lower bound to use with a branch-and-bound approach (like we saw for TSP), or use a dynamic programming approach where the table size depends on some parameter that might be more than polynomial-size in the input (like we saw for the change-making problem).
  Or do something else entirely - something you read about or come up with on your own. In any case, you need to convince me that your algorithm will *always* return the optimum solution, and it should be as efficient as possible.

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: Does 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?

1 Hungry Hungry Mids

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 $M_3 = (7,4,6,3)$.

2 Party Planner version 1

(This is the same problem from #3 on PS3!)

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 $P = \{1,2,4\}$.

3 Workout circuit

You are trying to improve your PRT score by working out for n minutes every day. You have a list of m possible exercises to do. Each exercise takes a certain number of minutes, and each will improve your PRT score by a certain (integer) amount. Oh, and you can only do each exercise *at most twice*; otherwise you'll get too tired. Your task is to figure out which exercises you should do within n minutes' time in order to increase your PRT score the most.

Formal problem definition:

`WORKOUT(n,E)`
Input: An integer number of minutes n, and a list of m pairs of integers

$$E = (t_1, b_1), (t_2, b_2), \ldots, (t_m, b_m),$$

where each t_i is how many minutes that exercise takes and b_i is the boost to your PRT score that exercise will give you.
Output: For each i from 1 to m, an integer c_i indicating how many times that exercise should be performed. Each c_i must be 0, 1, or 2. The sum of all the times must be less than n, and the sum total of PRT boost should be maximum.

For example, if $n = 10$ and there are $m = 4$ exercises: pushups $(2, 10)$, burpees $(3, 5)$, side plank $(5, 12)$, and running $(7, 21)$, then the optimum solution would be to do 2 sets of pushups and 1 set of side plank, for a total time of 9 minutes and a total PRT gain of 32 points.

(Note that you don't have to take up the entire n minutes, you just need to maximize your PRT score without going over n minutes and without doing any exercise more than twice.)

4 Shoe Matching

a.k.a. the multiple Cinderellas problem.

NOTE: This is not like the first 3 problems! It's not **NP**-hard! You should actually be able to come up with a fast algorithm here.

Scenario:

You and a group of friends all decide to jump in the lake for a swim. Naturally, you all leave your shoes up on the beach. And for some reason, you all have exactly the same looking shoes.

While you are in the water, a practical joker removes all the labels from the shoes and mixes them all around. So when you and your friends get out of the water, you are met with a big pile of shoes and you have no idea whose is whose. All you can do is start trying them on until everyone finds their own pair.

Knowing that you are an algorithms expert, your friends turn to you for the quickest way to sort all this out.

Specifics:

You *must* solve the problem under the following assumptions. No "side-channel" solutions like saying that you all find the practical joker, take his money, and buy yourselves new sneakers.

1) There are n friends and $2n$ shoes, consisting of n left shoes and n right shoes.
2) You can immediately tell the difference between a left shoe and a right shoe. Otherwise, there is no way to tell the difference between any shoes besides trying them on.
3) You cannot compare shoes to shoes directly. You cannot compare feet to feet directly.
4) An individual can try on one shoe at a time.
5) After trying on a shoe, the individual knows whether it was too small, too big, or just right.
6) For each of the n friends, there is exactly one shoe in the pile (no more, no less) that fits each of their feet perfectly.

Tasks

a) Devise an algorithm to match everyone with their shoes. You should be able to describe your algorithm in words or using pseudocode, or both.

b) Analyze the running time of your algorithm, in terms of the number of shoes that must be tried on, in total, by everyone. Try to come up with a big-Theta bound.

Hint: Perhaps you would like to `partition` somehow.