Introduction to Algorithms	**Problem Set 11**
CS 4820, Spring 2018	**Due 11:59pm Tuesday, May 8**

Hand in your solution electronically using CMS. Collaboration is encouraged while solving the problems, but:

1. **list the names of those with whom you collaborated;**
2. **you must write up the solutions in your own words.**

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time.

(1) *(10 points)*
For each of the following optimization problems, present an integer program whose optimum value matches the optimum value of the given problem. The combined number of variables and constraints in your integer program should be polynomial in the size of the given instance of the optimization problem.

(i). Set Cover. Given a universal set $\mathcal{U}$ and a collection of subsets $S_1, S_2, \ldots, S_m \subseteq \mathcal{U}$, what is the minimum size of a subcollection $\{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ whose union is $\mathcal{U}$?

(ii). Independent Set. Given a graph $G = (V, E)$, find an independent set of maximum cardinality.

(iii). Max-3SAT. Given a set of Boolean variables $x_1, x_2 \ldots, x_n$ and a set of clauses $C_1, C_2, \ldots, C_m$ each consisting of a disjunction of 3 literals from the set $\{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$, what is the maximum number of clauses that can be satisfied by a truth assignment?

(iv). Max-Cut. Given an undirected graph $G = (V, E)$, what is the maximum size of a cut? (A cut (A, B) is any partition of the vertex set V into two nonempty subsets. The size of a cut is equal to the number of edges with one endpoint on each side of the partition.)

It is not necessary to prove that your answer is valid. However, you should explain the interpretation of your notation well enough that we completely understand the structure of your integer program.

Example: In the weighted vertex cover problem, one is given a graph $G = (V, E)$ and a non-negative weight w_v for every vertex $v \in V$. One is asked to find the minimum total weight of a vertex cover.

Answer: The equivalent integer program is:

$$\begin{aligned} \min \quad & \sum_{v \in V} w_v x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1 \ \text{ for all edges } e = (u, v) \\ & x_v \in \{0, 1\} \ \text{ for all vertices } v \end{aligned}$$

Interpretation: Decision variable x_u equals 1 if vertex u is included in the vertex cover, 0 otherwise.

(2) *(15 points)*
Recall that in the Knapsack Problem, one is given a set of items numbered $1, 2, \ldots, n$, such that the i^{th} item has value $v_i \geq 0$ and size $s_i \geq 0$. Given a total size constraint B, the problem is to choose a subset $S \subseteq \{1, 2, \ldots, n\}$ so as to maximize the combined value, $\sum_{i \in S} v_i$, subject to the size constraint $\sum_{i \in S} s_i \leq B$. The input is assumed to satisfy $s_i \leq B$ for all $i \in \{1, \ldots, n\}$.

(a) Consider the following *greedy algorithm*, GA.

(i). For each i, compute the *value density* $\rho_i = v_i/s_i$.

(ii). Sort the remaining items in order of decreasing ρ_i.

(iii). Choose the longest initial segment of this sorted list that does not violate the size constraint.

Also consider the following *even more greedy algorithm*, EMGA.

(i). Sort the items in order of decreasing v_i.

(ii). Choose the longest initial segment of this sorted list that does not violate the size constraint.

For each of these two algorithms, give a counterexample to demonstrate that its approximation ratio is not bounded above by any constant C. (Use different counterexamples for the two algorithms.)

(b) Now consider the following algorithm: run GA and EMGA, look at the two solutions they produce, and pick the one with higher total value. Prove that this is a 2-approximation algorithm for the Knapsack Problem, i.e. it selects a set whose value is at least half of the value of the optimal set.

(c) By combining part (b) with the dynamic programming algorithm for Knapsack presented in class, show that for every $\delta > 0$, there is a Knapsack algorithm with running time $O(n^2/\delta)$ whose approximation ratio is at most $1 + \delta$. [Recall that the algorithm presented in class had running time $O(n^3/\delta)$.] In your solution, it is not necessary to repeat the proof of correctness of the dynamic programming algorithm presented in class, i.e. you can assume the correctness of the pseudopolynomial algorithm that computes an exact solution to the knapsack problem in time $O(nV) = O(n \sum_{i=1}^{n} v_i)$, when the values v_i are integers.