CS 610, Spring 2016, Prof. Calvin
Problem set #2
Due: Friday April 15, 5:00 pm.

Doe ONE of the following two problems, not both.

(1) Write a program that reads in a graph description and answers the following questions:
   (a) What is the size of the largest connected component of the graph?
   (b) What is the diameter of the largest connected component, and which two vertices are the furthest apart?
   (c) What is the most "important" subset of the largest connected component?

   The graph description is contained in a file with the following format. The first line contains a single integer n, which is the number of vertices of the graph. Each subsequent line contains two integers between 0 and $n-1$ representing an edge in the graph.

   Two example graph descriptions are in common/PROG2 on afs.

   The file relativity.txt contains a graph that describes collaboration among physicists, adapted from the file ca-GrQc available at

   `snap.stanford.edu/data/`

   The file actresses.txt describes a graph with vertices for each actress appearing in a movie since 2015, and an edge between two actresses if they appeared in the same movie. The file actressNames gives the actress names corresponding to the numerical vertex labels. The data are derived from actresses.list.gz, downloaded from the Internet Movie Database.

(2) Recall the $0-1$ knapsack problem with n items and weight bound W: Item i has weight w_i and value v_i and we want to choose the subset of items that weighs at most W and has maximal value. If the weights and values are integer, the problem can be solved using dynamic programming.

   Perhaps the simplest reasonable heuristic for the knapsack problem is the *decreasing density greedy* (DDG) algorithm: sort the items in decreasing order of "value per unit weight" v_i/w_i. Scanning down the sorted list, pack as many items as possible. The DDG algorithm then outputs the better of this solution and the one obtained by just taking the item with the largest value. The computational cost is $\mathcal{O}(n\log(n))$ (the cost of an efficient sort).

   How well would you expect this method to work on average? A popular probability model for addressing questions like this is to assume that the values and weights are drawn independently from a uniform distribution on $(0, 1]$. Note that the data are now continuous, and so our dynamic programming solution no longer applies.

   Let L denote the value of the linear programming relaxation of the knapsack problem (the "fractional" knapsack solution), and L_{DDG} the value of the DDG output (both random variables under our probability model).

   Consider a sequence of knapsack problems under the probability model introduced above and with the capacity W_n of the nth knapsack equal to $n/4$. Note that in our model the size and profit attributes are independent. Calvin and Leung (Calvin, J. and J. Y-T. Leung. 2003. Average-case

analysis of a greedy algorithm for the 0/1 knapsack problem. *Operations Research Letters* **31-3** pp. 202–210) show that for $x \geq 0$,

$$\lim_{n\to\infty} P\left(\left(\frac{4n}{3}\right)^{1/2}(P_L - P_{DDG}) \leq x\right) = F(x) \tag{0.1}$$

for a continuous distribution function F. Thus for large n, the gap between the lower bound and the greedy solution is of order $1/\sqrt{n}$.

Devise an algorithm that is fairly fast (say taking time $O(n^4)$) and has error smaller than DDG. You can characterize the error through numerical experiments or analytically. Your algorithm can be a variation of DDG or it may use it is a subroutine.

The multi-knapsack problem is the variation in which the items have more than one attribute with constraints on each attribute. Let's specialize to two attributes; call them weight and volume. So the ith item has weight w_i and volume y_i, and value v_i. We want to select the subset of items with maximal value that have total weight at most W and total volume at most Y.

Let's adopt the same probability model as above, with the weights, volumes, and values all independent and uniformly distributed between 0 and 1. Can you think of a good heuristic for this problem? Describe your algorithm and provide an error estimate through numerical experiments.