COMP 250 - Homework #6
Due on Dec. 7 2015, at 23:59.

Question 1. (25 points) Greedy and Dynamic Programming Algorithms

You are leaving on a hiking expedition and you want to fill your backpack with useful objects. You have n different types of objects to choose from. Each object of type i has a integer weight W(i) and an integer value V(i). Your backpack can carry at most a total weight T. Your goal is to find the optimal total value Opt(T) you can put in your backpack subject to this constraint Assume that you have infinite supply of each type of object, so you can select as many copies of each object as you want. However, the items are not divisible; you have to take an integral number of each (either zero, one, two, etc.).

For example, for n = 4, we might have
W(0) = 20, V(0) = 9
W(1) = 15, V(1) = 7
W(2) = 11, V(2) = 5
W(3) = 8, V(3) = 1

Then if T = 34, the optimal solution is to take three copies of item2 and zero of all others, for a total weight of 33 ≤ T and a total value of Opt(T) = 15. If T = 35, the optimal solution is to choose one item0 and one item1, for a total weight of 35 ≤ T and a total value of Opt(T) = 16.

a) (8 points). Describe a greedy algorithm to try to solve the problem. Your algorithm will probably not be guaranteed to produce the optimal solution in all cases.
b) (4 points) Give an example of a situation where your greedy algorithm fails to produce the optimal solution.
c) (10 points) Write a dynamic programming algorithm to solve the problem. Your algorithm has to work for any choice of W and V. First write the recursive formula, and then give the pseudocode for the dynamic programming algorithm.
d) (3 points) Use your algorithm to calculate Opt(37).

Question 2. (15 points) Sorting, one last time!

a) (10 points) Consider the following sorting problem: You are given an array of n integers. The integers are all between 1 and 2n, but are not necessarily all distinct. Write an algorithm that sorts this array and that runs in time O(n) in the worst case. (Hint: this is really easy once you see it. I could have said that the numbers are all between 1 and 10 n, or between 1 and 9394923 n and it would make no difference, except that the constant hidden in the big-Oh notation would get larger.)

b) (5 points) Why can't your algorithm be used efficiently to sort an arbitrary set of integers?

Question 3. (20 points) Game trees

Consider the following two-player game that is a variant of the popular Nim game. The game starts with a stack of n matches. Players alternate in removing matches from the stack. Each player can remove from the stack either 2 or 3 matches (provided there are sufficiently many matches left). The player who removes the last match wins, except if there is only one match left, in which case the game is a draw. For example, if n=3, then the first player wins by removing all three matches. If n=5, then the first player necessarily loses because any move he makes leads to a situation where his opponent wins.

a) **(10 points)** Draw the game tree for a game starting with n=9 matches. If both players play as well as possible, who will win?

b) **(10 points)** In general, if the game starts with n matches, who will win the game? Express your answer as a function of n.

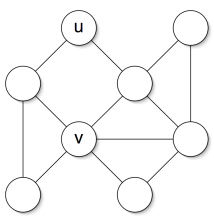
Question 4. (40 points) Graph algorithms

For this question, assume a graph has *n* vertices numbered 0,1, 2,...*n*-1.
In your algorithm, you can use the following graph ADT methods:
• *getNeighbors*(int *i*) returns the set of neighbors of vertex *i*. It is fine for you to write something like: for each vertex v in *getNeighbors*(17) do ...
• boolean *getVisited*(int *i*) returns TRUE if and only if vertex *i* has been marked as visited.
• *setVisited*(int *i*, boolean *b*) sets the visited status of vertex *i* to *b.*
• *getColor*(int *i*) returns the color of vertex *i*, either 0 or 1. If the color of vertex *i* has not been set previously, *getColor*(*i*) is undefined.
• *setColor*(int *i*, color *c*) sets the color of vertex *i* to *c*.

You may also want to associate to each vertex an integer called distance, which can be set and accessed through

- *setDistance*(vertex *v*, int *d*) sets the distance stored in *v* to *d*
- *getDistance*(vertex *v*) returns the distance stored in *v.* Undefined if that distance has not been set previously.

a) (20 points) In an undirected connected graph *G=(V,E),* the distance *d(a,b)* between vertices *a* and *b* is the number of edges in the shortest path between *a* and *b*. The excentricity of a vertex *a* is defined as the largest shortest-path distance between vertex *a* and any other vertex:
excentricity(*a*) = max { *d(a,b)* : $b \in V$ }



For example, in the graph to the right, excentricity(u) = 3 and excentricity(v) = 2.

Problem: Write an algorithm to compute the excentricity of a given vertex in a graph.

Algorithm excentricity(vertex u)
Input: a vertex u from the graph
Output: the excentricity of u
/* WRITE YOUR PSEUDOCODE HERE */

b) (20 points) Determining if an undirected graph can be colored with only three colors is an NP-complete problem. However, determining if it can be colored with only *two* colors is much easier. Write the pseudocode of an algorithm that determines if the vertices of a given connected undirected graph can be colored with only two colors (named 0 and 1) so that no two adjacent vertices have the same color.

Algorithm is2colorable(vertex u)
Input: a graph vertex u
Output: true if the graph to which u belongs is 2-colorable, and false otherwise
/* WRITE YOUR PSEUDOCODE HERE */