COMP 5711: Advanced Algorithm
2014 Fall Semester
Written Assignment # 1

Chapter 10

Problem 1 (20pts) We claimed that the Hitting Set Problem was NP-complete. To recap the definitions, consider a set $A = \{a_l.....a_n\}$ and a collection $B_1, B_2, \ldots, B_m$ of subsets of A. We say that a set $H \subseteq A$ is a *hitting set* for the collection $B_1, B_2, \ldots, B_m$ if H contains at least one element from each B_i, that is, if $H \cap B_i$ is not empty for each i. (So H "hits" all the sets B_i.)

Now suppose we are given an instance of this problem, and we'd like to determine whether there is a hitting set for the collection of size at most k. Furthermore suppose that each set B_i has at most c elements, for a constant c. Give an algorithm that solves this problem with a running time of the form $O(f(c,k) \cdot p(n,m))$, where $p(\cdot)$ is a polynomial function, and $f(\cdot)$ is an arbitrary function that depends only on c and k, not on n or m.

Problem 9 (20pts) Give a polynomial-time algorithm for the following problem. We are given a binary tree $T = (V, E)$ with an even number of nodes, and a nonnegative weight on each edge. We wish to find a partition of the nodes V into two sets of equal size so that the weight of the cut between the two sets is as large as possible (i.e., the total weight of edges with one end in each set is as large as possible). Note that the restriction that the graph is a tree is crucial here, but the assumption that the tree is binary is not. The problem is NP-hard in general graphs.

Chapter 11

Problem 1 (20pts) Suppose you have n containers of weight $w_1, w_2, \ldots, w_n$, and you want to load them into trucks, each of which can hold K units of weight. (You can assume that K and each w_i is an integer.) You can stack multiple containers in each truck, subject to the weight restriction of K; the goal is to minimize the number of trucks that are needed in order to carry all the containers. This problem is NP-complete (you don't have to prove this).

A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers $1, 2, 3, \ldots$ into it until you get to a container that would overflow the weight limit. Now declare this truck "loaded" and continue the process with a fresh truck.

a) Give an example of a set of weights, and a value of K, where this algorithm does not use the minimum possible number of trucks.

b) Show, however, that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of K.

Problem 7 (20pts) You're consulting for an e-commerce site that receives a large number of visitors each day. For each visitor i, $i = 1, 2, \ldots, n$, the site has assigned a value v_i. Each visitor i is shown one of m possible ads $A_1, A_2, \ldots, A_m$ as they enter the site. Given a selection of one ad for each customer, we will define the *spread* of this selection to be the minimum, over $j = 1, 2, \ldots, m$, of the total value of all customers who were shown ad A_j.

Example. Suppose there are six customers with values 3, 4, 12, 2, 4, 6, and there are $m = 3$ ads. Then, in this instance, one could achieve a spread of 9 by showing ad A_1 to customers 1, 2, 4, ad A_2 to customer 3, and ad A_3 to customers 5 and 6.

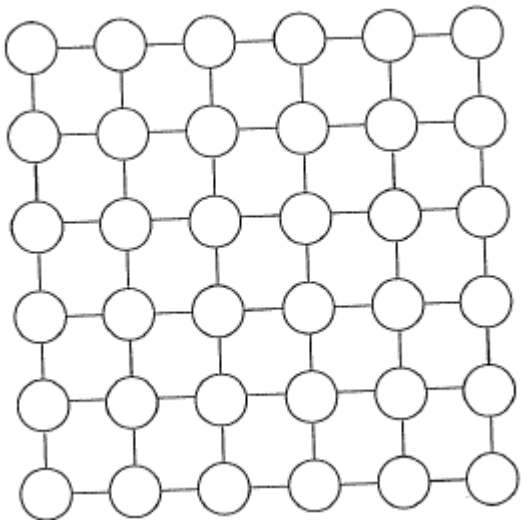
The ultimate goal is to find a selection of an ad for each customer that maximizes the spread. Unfortunately, this optimization problem is NP-hard (you don't have to prove this). So instead, we will try to approximate it.

a) Give a polynomial-time algorithm that approximates the maximum spread to within a factor of 2. That is, if the maximum spread is s, then your algorithm should produce a selection of one ad for each customer that has spread at least $s/2$. In designing your algorithm, you may assume that no single customer has a value that is significantly above the average; specifically, if $\bar{v} = \sum_{i=1}^{n} v_i$ denotes the total value of all customers, then you may assume that no single customer has a value exceecling $\bar{v}/(2m)$.

b) Give an example of an instance on which the algorithm you designed in part (a) does not find an optimal solution. Say what the optimal solution is in your sample instance, and what your algorithm finds.

Problem 10 (20pts) Suppose you are given an $n \times n$ grid graph G, as in the following figure. Associated with each node v is a weight $w(v)$, which is a nonnegative integer. You may assume that the weights of all nodes are distinct. Your goal is to choose an independent set S of nodes of the grid, so that the sum of the weights of the nodes in S is as large as possible.

Consider the following greedy algorithm for this problem.



Start with S equal to the empty set;
while *some node remains in* G **do**
    Pick a node v_i of maximum weight;
    Add v_i to S;
    Delete v_i and its neighbors from G;
end
Return S;

Algorithm 1: Heaviest-first algorithm

a) Let S be the independent set returned by the "heaviest-first" greedy algorithm, and let T be any other independent set in G. Show that, for each node $v \in T$, either $v \in S$, or there is a node $v' \in S$ so that $w(v) \leq w(v')$ and (v, v') is an edge of G.

b) Show that the "heaviest-first" greedy algorithm returns an independent set of total weight at least $\frac{1}{4}$ times the maximum total weight of any independent set in the grid graph G.