MA421 Advanced Algorithms Lent 2019

Exercises 4

1. Implement Greedy 2 algorithm for Set Cover from the lecture in JAVA. Your program should receive the input from the command line as
   `>java SolveSC n w1 a11 a12 a13 a14 ... 0 w2 a21 a22 ... 0 ... 0 wm am1 am2 ... 0`
   where each argument is an integer, `n` is the size of the ground set $\{1, \ldots, n\}$, each `wj` is the integer weight of the corresponding subset S_j, for $j = 1, \ldots, m$, `aji`'s are the elements of S_j, and `0`-valued arguments are used to separate the descriptions of the consecutive sets. You can assume that the input is valid and, hence, no checks are required. You can also write an exhaustive search routine to find the optimal solution and compare it to your approximate solution.

2. Give a greedy algorithm that achieves an approximation factor of H_g for the set multicover problem, which is a generalisation of the set cover problem in which an integral coverage requirement is also specified for each element and sets can be picked multiple number of times to satisfy all coverage requirements. Here, g is the maximum cardinality of any set in the given collection. Assume that the cost of picking α copies of set S_j is α times the cost of picking set S_j.

3. By giving an appropriate tight example, show that the analysis of Greedy 2 algorithm cannot be improved by more than a constant factor even for the unweighted set cover problem: that is, the worst-case approximation factor on unweighted instances is $\Omega(\log n)$.

   [Hint: Consider a generalisation of the counter example from the textbook for the weighted case or running the greedy algorithm on a vertex cover instance based on a multigraph.]

4. Consider the following variant of the set cover problem. Given a universal set T of n elements, a collection of subsets of T, $S_1, S_2, \ldots, S_m$, and an integer k, pick k sets so as to maximise the number of the elements covered.

   In this question, you are asked to show that the obvious algorithm, of greedily picking the best set in each iteration until k sets are picked, achieves an approximation factor of $1 - \frac{1}{e}$.

   [Hint: The following quantities could be useful to think about. Let OPT denote the value of an optimal solution to the problem. Let x_i, for $i = 1, \ldots, k$, denote the number of new elements covered by the algorithm in the i-th set that it picks. Finally, $z_i = \text{OPT} - \sum_{j=1}^{i} x_j$ is the difference between the optimal value and the number of covered elements after the i-th set was picked.]

5. Given a graph $G(V, E)$, we want to partition the vertices of the graph into two (disjoint) sets A and B (i.e., $A \cup B = V$ and $A \cap B = \emptyset$) such that the number of edges with one endpoint in A and the other in B is maximised. In other words, we are looking for the maximum cut (A, B) of the graph. Consider the following algorithm. We start with an arbitrary partition (A, B). If there exists a vertex $v \in V$ such that moving it to the other set in the partition increases the number of edges in the cut, do so. Proceed until there is no such vertex.

   (a) Show that the algorithm stops.

(b) Show that when the algorithm stops, the size of the cut is at least half of the optimum. Hint: Consider the neighbours of each vertex after the algorithm stops.

Solutions to these exercises are to be handed in at the lecture on **12 February 2019**. The JAVA code for Question 1 should be emailed to `t.batu@lse.ac.uk`.

