Colorado CSCI 5454: Algorithms
Homework 5

Instructor: Bo Waggoner
Due: October 3, 2019 at 11:59pm
Turn in electronically via Gradescope.

Remember to list people you worked with and any outside sources used.

Problem 1 (8 points)

In this problem, the input is an unweighted, undirected graph $G = (V, E)$. Note G is not necessarily bipartite. Recall that a *matching* is a set of edges of the graph such that each vertex $v \in V$ appears as an endpoint at most once.

Hint for this problem: revisit lecture notes on bipartite matching.

Part a (2 points) A matching is called *maximal* if it cannot be made larger, because adding any more edges to the set will violate the condition that all vertices appear at most once. In contrast, a matching is *maximum* if it has the largest number of edges possible, out of all matchings in the graph.

Give an example of a graph and a matching that is maximal, but not maximum.

Part b (2 points) Give a greedy algorithm for finding a maximal matching in a graph. (You do not need to argue correctness and efficiency.)

Part c (4 points) Give an approximation algorithm for the *maximum* matching in a general graph. Argue that it has approximation ratio at least 0.5.

Problem 2 (10 points)

A *vertex cover* of a graph is a set of vertices such that, for every edge $(u, v) \in E$, at least one of $\{u, v\}$ is in the set. An example of a vertex cover is the entire set V.

The *minimum vertex cover problem* asks to find the smallest possible vertex cover. This problem is NP-complete. In fact, it has been proven that unless P=NP, the best possible polynomial-time algorithm has an approximation factor at least 1.36, and it is conjectured that the best possible factor is 2.

Part a (2 points) Prove the following lemma: If S is a maximal matching in a graph, then the set of all endpoints of all edges in S is a vertex cover.

Part b (2 points) Now, prove this lemma: If S is a maximal matching, then *every* vertex cover must contain at least $|S|$ vertices.

Part c (2 points) Use the previous parts to give an approximation algorithm for the minimum vertex cover problem and prove that it is a 2-approximation.

Part d (4 points) Finally, consider the even harder problem of *online* vertex cover.

- Initially, nothing is known by an algorithm and it begins with an empty set W, its vertex cover.
- Each round, a new vertex v arrives. The algorithm learns all of its edges *to vertices that have already arrived*.
- The algorithm may choose to add any vertices to W, with the constraint that it must always maintain a vertex cover of the graph so far.
- After the algorithm decides, the next round begins.

The algorithm's performance is the size of its final vertex cover W, compared to the offline optimal minimum vertex cover of the graph. Note the algorithm can never remove vertices from W once they are added.

Question: Give an online algorithm and prove it has a competitive ratio of 2.

Problem 3 (16 points)

Recall the knapsack problem (without repeats): Given a list of n items' values $v_1, \ldots, v_n$ and weights $w_1, \ldots, w_n$, along with a weight limit W, find a subset of items with total weight at most W. The performance of an algorithm is the sum of the values of the items, and the goal is to maximize performance.

For this problem, assume that $w_i \leq W$ for all i; in other words, each item can at least fit on its own. Also assume $\sum_{i=1}^{n} w_i > W$, otherwise we could just include all the items.

We saw a dynamic programming algorithm to find OPT (the maximum performance), but it could be quite slow for large W. In this problem, you'll show a fast 0.5-approximation.

To start, consider the following algorithm, Greedy, that tries to add items in order of the most "bang per buck" (value per weight):

- For each item i, let $a_i = \frac{v_i}{w_i}$.
- Sort the items from largest a_i to smallest.
- Add items in this order until the next one does not fit; then stop.

Part a (2 points) Consider the following input: $v_1 = 1$, $w_1 = 1$; $v_2 = 5, w_2 = 10$; $v_3 = 9, w_3 = 10$; $W = 10.1$.
What is the value of the optimal solution and which items are in it?
What is the value of Greedy's solution and which items does it choose?
What is the ratio of Greedy to optimal in this example?

Part b (2 points) Explain how to modify the previous example to get an arbitrarily low approximation ratio. In other words, if you are given any $\epsilon > 0$, construct an instance of knapsack where $\frac{\text{Greedy}}{\text{Opt}} \leq \epsilon$.

Part c (2 points) Now consider the algorithm CheatingGreedy. This is the same as Greedy, but it gets to include the last item in the loop, the one that does not fit, in its solution. So CheatingGreedy actually will violate the weight constraint W, but analyzing it will help us find a good non-cheating algorithm.

First, what is the performance of CheatingGreedy on the example in part (a)?

Part d (4 points) Next, argue that CheatingGreedy's performance is at least as large as Opt. *Hint: if we let W' be the total weight used by CheatingGreedy, first argue that CheatingGreedy achieves the optimal performance for constraint W'.*

Part e (2 points) Our final algorithm is called CarefulGreedy. First, it runs Greedy. Let i be the first item that Greedy cannot fit. Let V be the total value of the items taken by Greedy. Then CarefulGreedy outputs:

- just item i, if $v_i \geq V$;
- the Greedy solution, otherwise.

What is the performance of CarefulGreedy on the instance in part (a)?

Part f (4 points) Prove that CarefulGreedy has an approximation ratio of 0.5.

Hint 1: first show that CarefulGreedy ≥ 0.5 CheatingGreedy.

Hint 2: what is the performance of CheatingGreedy in terms of V and v_i?

Problem 4 (Bonus: 1 point)

For the Load Balancing problem (lecture 9), give an algorithm and prove it guarantees a $\frac{3}{2}$-approximation.

Hint: Make the greedy algorithm just a little bit smarter.

Problem 5 **(Bonus: 1 point)**

Recall that in Problem 2, we mentioned that exactly solving min vertex cover is NP-complete — on general graphs. Give a polynomial-time algorithm for min vertex cover on *bipartite* graphs.

Hint: start from a maximum matching, then try for the best possible.