Sample Final: CSCI 6212 Algorithms

Policies: Test will be closed book, closed notes. You may bring in one sheet of paper. Tests will be taken individually, with no conversation, discussions with other classmates, or outside assistance.

1. Short Answer

   (a) Technically advanced aliens come to Earth and show us that some known NP-hard problem cannot be solved faster than $O(n^{100})$ time. Does this resolve the question of whether P = NP? (Explain briefly.)

   This does not change anything. While interesting, knowing that some problem cannot be solved with a polynomial with a small exponent does not prove that it cannot be be solved in polynomial time.

   (b) Suppose that you create a reduction from problem A to problem B, the reduction runs in $O(n^2)$ time, and problem B can be solved in $O(n^4)$ time. What can we infer about the time needed to solve A? (Explain briefly.)

   This proves that problem A can be solved in polynomial time, specifically, for a given input, in $O(n^2)$ times an instance of problem B with the same answer can be created. The size of that instance can't be more than $O(n^2)$, (because the reduction runs in $O(n^2)$ the output can't be bigger than that). so algorthm B can run on that in time $O((n^2)^4) = O(n^8)$.

   (c) (True or False): If a graph G has a vertex cover of size k, then it has a neighbor set (dominating set) of size k or smaller.

   False. Consider the graph of 5 nodes with 0 edges. In this case the vertex cover is size 0, but the NeighborSet is size 5

   (d) Suppose that you create a polynomial time reduction from problem A to problem B, and there is a factor-2 approximation to problem B, then which (0,1,2 or 3) of the following are true:

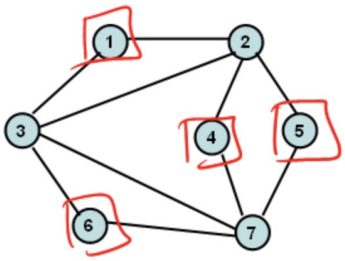
      i. There is a factor-2 approximation for A.
      ii. There is a constant factor approximation for A, but the factor might not be 2.
      iii. We cannot infer anything about our ability to approximate A.

   The third answer is true. Most reductions to not preserve the approximation. In particular, in our class, we mostly to use 3-SAT in our problems and it isn't clear what "approximation" means for a 3-SAT problem

2. Given an undirected, unweighted graph $G(V, E)$, and Independent Set is a set of nodes in the graph where none of the nodes are connected by an edge. For example, in the graph below with 7 nodes, the set with red squares around them are an independent set of size 4: The INDEPENDENT-SET problem is: given a graph G and an integer k, does G have an independent set of size $\geq k$?

   Prove INDEPENDENT-SET is NP-Complete by a reduction from the Vertex Cover Problem, following the following four steps.

   (a) (1) Prove INDEPENDENT-SET $\in NP$.
   tiny*Recall that this means that it is possible to present a certificate that proves when a given instance has an independent set. In this case, the certificate consists of the k vertices of V. In polynomial time we can verify that, for each pair of vertices $u, v \in V$, there is no edge between them.*



(b) Reduce VERTEX-COVER to INDEPENDENT-SET. *a great answer to the rest of this problem is here:* `http://www.cs.cornell.edu/courses/cs482/2005su/handouts/NPComplete.pdf`

- (2) Show how you translate the problem,
- (3) prove that the answer to your INDEPENDENT-SET question equivalent to the answer to the VERTEX-COVER question, and
- (4) argue that the INDEPENDENT-SET problem you have created is not more than a polynomial function of the size of the VERTEX-COVER problem.

3. On homework 5 we argued that the greedy algorithm for vertex cover may give a bad approximation ration. A similar argument applies to the NeighborSet (also called the Dominating Set) problem we talked about in the homework 5. Suppose you are trying to solve the NeighborSet problem on a graph where all nodes have 3 or fewer neighbors. Give an algorithm for this special version of the NeighborSet problem with an approximation bound of 3.

   Any sensible algorithm will give a 3 approximation in this case!

   Alg:
   (a) Choose any node. Remove it and all of its neighbors (at most 3 of these exist).
   (b) if any nodes are left, goto (a).

   We prove that if the optimal dominating set exist and has size k, this chooses at most $3k$ nodes.

   Consider the optimal dominating set, it consists of a set of nodes DS, and each of these nodes can have at most 3 neighbors.*

   If we consider one node in DS and its neighbors as a "cluster" our algorithm will pick at most 3 from each cluster. Why? If we pick the DS node first, we won't pick anything else in the cluster. If we don't pick the DS node first, then we will delete it (it is a neighbor of the node we pick), and there are only two nodes left in the cluster.

   So we pick up to 3 nodes per cluster (even though we don't know where the optimal clusters are!). If the size of the dominating set is k, there are k clusters and we pick at most $3k$ nodes, so this is a three approximation.

4. Suppose you have an oracle that can answer the $CLIQUE(G, k)$ problem, "Does graph G have a clique of size at least k". If a clique of size k exists, show how to use this oracle to return the set of nodes that are in the clique.

   `The idea is the throw out points that aren't needed,`

```
until you're left with just k-points

Let C*(G,k) be the oracle answering true if G has a
clique of size k

For i = 1...n
    Make G_tmp = G' without node $v(i)$
    If C*(G_tmp,k) is true
        G' = G_tmp     % Keep going without this node
    else
        G' = G'        % No change!
    end
end

When this ends, the graph G' is a clique of size k, and
all vertices in G' are vertices from the original graph G,
so listing those vertices out gives the Clique.

Argument for correctness:

Nodes are only deleted when the oracle reports that there
remains a clique of size $k$ in the graph without that
node, so if there is a clique in the original graph G,
there will be a clique in the resulting graph G'.

If there is no clique in the original graph G, we don't
create one, because nowhere in the algorithm do we add
nodes or edges.

Runtime: This construction loops over all nodes O(n)
loops, and removes the node and up to n-edges O(n) per
loop, and calls the oracle (treated as a constant) for a
total of O(n^2) time.
```