COMP 360 - Fall 2019 - Sample Final Exam

1. Prove that the following problem belongs to P: Given a graph G, we want to know whether G has an independent set of size 100.

   Solution: Let n be the number of vertices of G. For every possible way of choosing 100 vertices, we check whether they form an independent set. Note that there are $\binom{n}{100} \leq n^{100}$ possible ways of choosing 100 vertices out of n vertices, and it takes on $O(1)$ to see whether they form an independent set. Thus the running time of this algorithm is $O(n^{100})$.

2. Give an example of a linear program that is infeasible and its dual is also infeasible.

   Solution:

   $$\begin{array}{ll} \max & x_1 + 2x_2 - x_3 \\ \text{s.t.} & x_1 \leq 1 \\ & -x_1 \leq -2 \\ & x_2 - x_3 \leq 0 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

   Dual:

   $$\begin{array}{ll} \max & y_1 - 2y_2 \\ \text{s.t.} & y_1 - y_2 \geq 1 \\ & y_3 \geq 2 \\ & -y_3 \geq -1 \\ & y_1, y_2, y_3 \geq 0 \end{array}$$

3. Prove that the following problem belongs to PSPACE: Given a graph G and an integer k, we want to know whether the number of independent sets in G is equal to k.

   Solution: Let n be the number of vertices of G. One can generate all the possible 2^n subsets of the vertices, one by one (reusing the memory). Each such set takes n bits of space. We will also have a variable I that is equal to the number of sets that form an independent set. Every time that a new subset is generated, we check whether it is an independent set, and then update the variable I accordingly. Note that I takes only $\log 2^n = n$ bits of memory. Hence in total the required space is going to be $O(n)$.

4. Show that the following problem is NP-complete:

   - Input: An undirected graph G and an edge e.
   - Question: Does G have a Hamiltonian cycle that passes through the edge e.

   Solution: We reduce the Hamiltonian cycle problem to this problem. For an input G for Hamiltonian cycle problem, we run the following oracle algorithm:

   - For every edge e in G.

- Run the oracle on $\langle G, e\rangle$ and if the output is YES, output YES and terminate.
- EndFor
- If not terminated yet, output NO.

5. Prove that the following algorithm is a 2-factor approximation algorithm for the minimum vertex cover problem:

   - While there is still an edge e left in G:
   - Delete all the two endpoints of e from G
   - EndWhile
   - Output the set of the deleted vertices

   Solution: Note that at every step the above algorithm finds a new edge e and deletes both its endpoints. Let M be the set of all such edges found by the algorithm. Note that the edges in M are completely disjoint (they do not share any vertices) as every time that the algorithm finds such an edge it deletes both its endpoints. The optimal solution must pick at least one vertex from each one of these edges, and thus the size of the optimal solution is at least $|M|$ (i.e. the number of the edges in M). On the other hand the algorithm deleted at most $2|M|$ vertices. Hence the output of the algorithm is at most $2 \times$ OPT.

6. Prove that the following algorithm is a $\frac{1}{2}$-factor approximation algorithm for the MAX-SAT problem, in which the goal is to maximize the number of satisfied clauses: Given a CNF ϕ on n variables $x_1, \ldots, x_n$:

   - For $i = 1, \ldots, n$ do
   - IF x_i appears in more clauses than $\overline{x_i}$ THEN
   - Set $x_i = T$
   - Else
   - Set $x_i = F$
   - Remove all True clauses from ϕ and remove x_i and $\overline{x_i}$ from all the other clauses
   - EndFor

   Solution: Let t_i be the number of clauses that become TRUE when we set the value of x_i, and similarly let f_i be the number of clauses that become FALSE (This means that at this point all the terms in the clause are already set to FALSE). Obviously the way the algorithm works guarantees $t_i \geq f_i$.

   On the other hand note that after the algorithm terminates, $\sum t_i$ is the total number of TRUE clauses in the formula and $\sum f_i$ is the total number of false clauses in the formula, and from the previous discussion we know that $\sum t_i \geq \sum f_i$. Thus the algorithm satisfies at least half of the clauses which is at least $\frac{1}{2}$OPT.

7. A *kite* is a graph on an even number of vertices, say $2k$, in which k of the vertices form a clique and the remaining k vertices are connected in a tail that consists of a path joined to one of the vertices of the clique. Prove that KITE problem defined as in the following is NP-complete.

   - Input: An undirected graph G, and a positive integer k.

- Question: Does G contain a kite on $2k$ vertices as a subgraph?

Solution: The problem is obviously in NP as given the vertices that form a kite as a subgraph one can easy check whether it is a valid kite or not. To prove completeness we reduce CLIQUE to this problem. On an input $\langle G, k \rangle$ for CLIQUE we attach a tail of length k to every vertex of G. Now we run the oracle for KITE on this new graph with the same number k. Note that if the oracle returns YES then the original contains a k-Clique, and if it returns NO then the original graph does not contain a k-Clique.