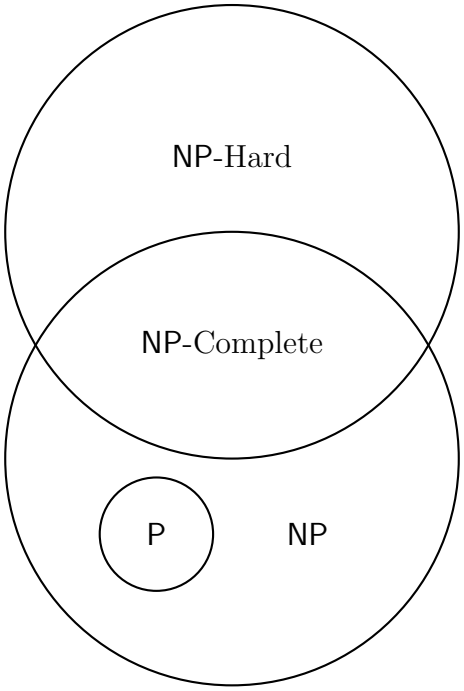
CS 3510: Design & Analysis of Algorithms 11/09/2023

Practice Exam 4: Intro to Complexity

Abrahim Ladha

- This is the CS 3510 practice exam for exam 4.
- Topics include: Algorithmic Complexity
- The number of questions on this practice exam is roughly what you could expect on the real exam.



1. For certain, this graph shows a necessarily correct depiction of the relationship between the sets P, NP, NP-Hard, and NP-Complete (this is a trick question).

   ◯ True

   √ **False**

   √ **Maybe**

   Frankly I'd say false because this question is essentially asking, does $\mathsf{P} \neq \mathsf{NP}$ and as a matter of fact, we do not know. However maybe is also a correct answer as well to provide more leniency.

2. A algorithms researcher has discovered that the running time for an algorithm solving the 3-Coloring problem has a lower bound of $\Omega(2^n)$. If the researcher's discovery is correct, what does that mean in terms of the relationship between P and NP?

   ◯ $\mathsf{P} = \mathsf{NP}$

   √ $\mathsf{P} \neq \mathsf{NP}$

   ◯ There is nothing additional that could be said about the relationship between P and NP

   With this researcher's discovery, we have found that the fastest algorithm to an NP-Complete problem (3-Coloring) is exponential. This means that this problem cannot be in P, and since we found that there exists a problem in NP that isn't in P, $\mathsf{P} \neq \mathsf{NP}$.

3. Let's define the SHORTEST-PATH decision problem to be the following:

   $\texttt{SHORTEST-PATH} = \{\langle G, s, Wt, k\rangle \mid G \text{ with weights } W \text{ has a path from vertices } s \text{ to } t \text{ of length} \leq k\}$

   If Alice has found a polynomial time reduction from 3SAT to SHORTEST-PATH, what does that mean in terms of the relationship between P and NP?

   - √ P = NP
   - ◯ P ≠ NP
   - ◯ There is nothing additional that could be said about the relationship between P and NP

   SHORTEST-PATH is in P since it can be solved by Dijkstra's algorithm, which runs in polynomial time. Since 3SAT is an NP-Complete problem, all problems NP reduce to 3SAT. Since Alice has found a reduction from 3SAT to SHORTEST-PATH, a composition of reductions can be used to reduce all NP problems to SHORTEST-PATH. As a result, all NP problem are solvable in polynomial by doing a reduction to SHORTEST-PATH and applying Dijkstra's algorithm, meaning that P = NP.

4. Every problem in NP\P is NP-complete

   - ◯ True
   - √ **False**

   If P ≠ NP then there do exist problems which are in NP, not in P, and not NP-complete. We have some useful candidates of NP intermediate problems to be factoring and graph isomorphism.

5. You've just made a break through and found a polynomial solution to the `LimitSAT`! The issue is that it's not exactly the same as normal `SAT` because each variable can only appear in at most three clauses, and there can only be at most three literals per clauses. Can you show that this is NP-Complete, thus showing P = NP (Obviously this is a made up scenario because we don't actually know if P = NP, but nonetheless, show that `LimitSAT` is in NP-Complete)?

Solution:

First we'll prove that this is in the class NP. To do this, we could take the assignments as a witness and check that it evaluates to true like a standard `SAT` problem.

Now we must show that the problem is in the class NP-hard; we will reduce `3SAT` with exactly three clauses to `LimitSAT`. In pour transformation we'll take every instance of a variable which appears more than once and replace it with some element in a sat of related variables; for instance if we were to replace instances of x, we'd replace it with $x_0, x_1, ..., x_n$. We need to create implication's between the clauses in order to make them all equal, so we'll add $(\overline{x}_1 \vee x_2) \wedge (\overline{x}_2 \vee x_3) \wedge ... \wedge (\overline{x}_n \vee x_1)$

This is trivially correct because if there's a satisfying assignment for normal `3SAT`, then the same must exist for `LimitSAT`. If there's not a satisfying assignment for `3SAT`, then there must not be one for `LimitSAT`. This is because all the variables corresponding to the original variables are equal by the implications, so this will map correctly, meaning the `LimitSAT` problem is in NP-complete.

6. As a student, you're trying to determine your schedule for the upcoming semester. For a particular day, there's some set of classes $s \in S$, each class has some meeting times (a class can meet multiple times in the day i.e. lecture and recitation). You need to determine whether you can schedule k non-overlapping classes throughout the day. Show this problem is NP-Complete.

Solution:

We can verify that a witness is valid by simply checking that there are no overlapping classes.

We can show that this problem is NP-hard by doing a reduction from independent set. We will take every node in independent set as a class and take the edges as individual meeting times for that class. Each node which shares an edge will have an overlapping time. This will not allow two nodes that share an edge to be chosen together. We will map k to k in independent set.

This is correct because it correctly maps the rules from independent set to the scheduling problem. If a problem is valid in independent set, the classes corresponding to the chosen nodes can also be chosen because they do not have overlapping times. If a problem is invalid in independent set, no more classes than the number for independent set can be chosen because the possible sets of classes which can be chosen is the same as the possible sets of nodes. Then this problem is in NP-complete.

7. Show that the following CYCLE-FREE problem is NP-Complete.

Input: a directed graph $G = (V, E)$ and an integer k.
Output: Returns a set $M \subseteq E$ of size at most k such that $|M| \leq k$ and a subgraph $G' = (V, E \backslash M)$ is cycle-free.

Solution:
Problem is in NP: Recall that topological sort takes linear time in the size of the graph, i.e. $\mathcal{O}(|V| + |E|)$ time. Given a subset of edges, we can thus check in linear time if the size of the subset M is at most k, and if the $G(V, E \setminus M)$ is acyclic. Thus, CYCLE-FREE problem is in NP.

Choose NP-Complete problem Vertex-Cover. To show that CYCLE-FREE is NP-hard, we perform a reduction from Vertex-Cover. Let $G(V, E), k$ be an instance of Vertex-Cover, which we transform to an instance of CYCLE-FREE (G', k') in linear time as follows:
For every vertex $v \in V$, we add two corresponding vertices $v_{in}, v_{out} \in V'$ that we connect by a directed edge (v_{in}, v_{out}), and for every edge $(u, v) \in E$, we add the directed edges (u_{out}, v_{in}) and (v_{out}, u_{in}) to E'. Finally we set $k' = k$.

$\phi \in$ Vertex-Cover $\implies f(\phi) \in$ CYCLE-FREE
I has a solution in Vertex-Cover i.e. G has a vertex cover S where $|S| \leq k$. For every $v \in S$, consider removing the edge (v_{in}, v_{out}) in G'. The resulting graph is acyclic. If a cycle in G' enters a vertex in v_{in}, it can only leave through the edge (v_{in}, v_{out}) given our construction. Similarly, a cycle can only enter v_{out} through the edge (v_{in}, v_{out}). Thus, if a cycle in G' uses the edge (u_{out}, v_{in}), it must use the edge (u_{in}, u_{out}) and the edge (v_{in}, v_{out}). At least one of these two edges would have been removed since at least one of u or v must be in the vertex cover to cover the edge $(u, v) \in E$. Therefore, if G has a vertex cover of size at most k, then G' has a CYCLE-FREE set of size at most $k' = k$ as well.

$\phi \in$ CYCLE-FREE $\implies f(\phi) \in$ Vertex-Cover
Conversely, suppose G' has a cycle-free set S' of size at most k'. Without loss of generality, we assume that the only edges removed are of the form (u_{in}, u_{out}) because if some other edge (u_{out}, v_{in}) is removed, then all cycles in which this edge participated would have included the edge (v_{in}, v_{out}) too and we could remove this edge instead.

We claim that the set of vertices $v \in V$ for which the corresponding edge (v_{in}, v_{out}) is removed in E' forms a vertex cover for G. Suppose not, and suppose there is some edge $(u, v) \in E$ not covered. Then in G' the cycle (u_{in}, u_{out}), (u_{out}, v_{out}), (v_{in}, v_{out}), (v_{out}, u_{in}) is unbroken by the removal of the edges contradicting the assumption. Thus (G, k) must have a vertex cover of size at most $k = k'$.

This proves CYCLE-FREE problem is NP-Hard.

Since CYCLE-FREE problem is both in NP and is NP-hard, it is NP-complete.

8. Show that the `Subgraph-Isomorphism` is in the class NP-complete. The problem is as follows: given as input two undirected graphs G and H, determine whether G is a subgraph of H.

Solution: We know this problems in NP by using the witness as the set of chosen nodes in the H which is equal to G. The witness is not only the set of nodes, but the order they are in. It specifies exactly where the subgraph is.

We will reduce from Clique. Note that if a graph has a clique, thats really just a complete subgraph. Given a graph for clique and a size c, construct a new complete graph with c nodes. Plug the provided graph into H and the constructed graph into G.

If there is a clique in the original graph the `Subgraph Isomorphism` problem must find it or decide it can't because the clique has been passed in. It can not find another clique because that would imply it had found a subgraph not present in the actual graph. If there is not a clique in the original graph, it could not find say there is one because that would imply that a clique does exist even though there isn't.

9. Show that the `6-Coloring` problem is in the class NP-complete. The problem is as followings: given as input a graph G, determine whether if there exists an assignment of the vertices to 6 colors such that no edge has both ends the same color.

Solution: We know this problem is in NP by using the witness as the assignment of each vertex in G to a color. The verifier will check that 6 colors are used and each edge has both ends different colors.

We will reduce from `3-Coloring`. We construct a triangle of three new vertices that are all connected to each other. For each new vertex in the triangle, we add an edge between this new vertex and all the original vertices in G. Lets call this new graph G'.

If there is exists a 3 coloring in the original G, then the same coloring can be applied to the corresponding original vertices in G'. The new vertices in the constructed triangle would then be assigned the remaining 3 colors. If the G' has a valid 6 coloring, then by construction of G', the colors used in the triangle are different from the original vertices. Removing these 3 vertices would result in 3 remaining colors for the original vertices. Therefore, a valid 3-coloring exists for the original graph G.

10. The Frog Software Foundation has recently come out with Knapsack as a service, allowing you to solve the Knapsack decision problem, wherein you input your knapsack problem and a value, and the service will tell you if the best value in Knapsack is above the provided value. How could you use this to efficiently solve the Knapsack optimization problem wherein you're given the Knapsack parameters and you want to find the best possible value. Then discuss your algorithms runtime.

Solution: You're trying to find the maximum number while still being able to satisfy Knapsack. Luckily since the problem is formulated by giving a lower bound, the function provided by the Frog Software Foundation is Monotonic Decreasing with respect to a given lower bound. This means you could just count up until you find a lower bound which does not work, but this will take exponential time with respect to the number of bits. Luckily, we know that we can do a binary search on a monotonic function in logarithmic time. This means we can find the highest lower bound in linear time.

For example, for some knapsack instance, fix the items and capacity and try values $V = 1, 10, 100, 1000, 10000...$ in binary. The Frogs will say "higher, higher, higher, higher, lower". When they say lower, you know the first bit of V, and its length. Then for the rest of the bits you infer them one at a time. For example if you try 110000, and the Frogs say lower, you know it is at most 101111, or that that second bit was a zero. If it said higher, then you know it was at most 111111, or that the second bit was a one. You proceed bit by bit to reconstruct the maximum value V. This takes $\log V$ steps.