15-251: Great Theoretical Ideas In Computer Science

Recitation 13 Solutions

Review

- The set **P** is the set of all languages L such that there exists a program P and a constant $c \in \mathbb{R}$ such that P decides L and $P(x)$ runs in at most $|x|^c$ steps for all x. Less formally, **P** is the set of all languages that are decidable in polynomial time.
- The set **NP** is the set of all languages L such that there exists a program V and a constant $c \in \mathbb{R}$ such that for all x,
  - If $x \in L$ then there exists a string y such that $V(x, y)$ returns "yes" in at most $|x|^c$ steps.
  - If $x \notin L$ then for all strings y, $V(x, y)$ returns "no" in at most $|x|^c$ steps.

  Less formally, **NP** is the set of all languages for which a proof of membership can be checked in polynomial time.
- A (many-one) reduction from a language L to a language L' is a function $f : L \to L'$ such that $x \in L \Leftrightarrow f(x) \in L'$. In this class we are interested only in polytime reductions, i.e., reductions that can be computed in polynomial time. Whenever we say reduction, we mean polytime reduction.
- A language L is **NP**-hard if for all languages $L' \in$ **NP**, there exists a polytime reduction from L' to L.
- A language L is **NP**-complete if $L \in$ **NP** and L is **NP**-hard.
- For your homework, you may assume without further proof (we've already shown this) that $CIRCUIT SAT, 3SAT, 3COLOR, CLIQUE, INDSET$ are all **NP**-complete.

Stupid Reductions...

Recall the definition of the halting problem:

HALT: Given a machine M and an input w, does M halt on w?

And the definition of the three-coloring problem:

3COLOR: Given a graph G, does G have a valid 3-coloring?

Now, define a new language ISSORTED, in the expected way:

ISSORTED: Given a list l of positive integers, are they sorted in non-decreasing order?

Show that 3COLOR reduces to HALT.

Show that ISSORTED reduces to 3COLOR.

What can you conclude from these two reductions?

Given an oracle for HALT, we can trivially solve 3COLOR. Construct a machine M which will try all possible 3-colorings of a graph. We make this machine halt if it finds a valid 3-coloring, and loop forever otherwise. Now, given an input G to 3COLOR, we have no more work to do, we simply pass the input into HALT(M, G) and see what the answer is. Note that constructing the machine M can be seen as constant time because it doesn't depend on our input G at all!

Given an oracle for 3COLOR, we can also solve ISSORTED easily. Remember that a reduction is just required to be in polytime. But checking if a list is sorted is easily done in polytime. In this case, when given an input for ISSORTED, we simply don't even use the oracle for 3COLOR, and just check if the list is sorted ourselves. This is polytime, so it is a valid reduction.

What did we just show? Not a whole lot! We showed that HALT is NP-hard, so every problem in NP reduces to it. Informally, this means that HALT is at least as hard as every problem in NP. This should be expected, since HALT is undecidable (and thus very, very hard). We didn't show that HALT is NP-complete because we can't show that HALT $\in$ NP.

We also showed that 3COLOR is harder than ISSORTED. Actually, our reduction for ISSORTED would have worked with any language. We can show that ISSORTED reduces to any other problem that we want. Informally, this means that ISSORTED is among the easiest problems that we can define. All problems in P are among the easiest by this definition, as solving them is just as easy as doing a polytime reduction.

How about you PROVE these are NP-complete instead of just taking our word for it?

Recall two problems from lecture:

CLIQUE: Given a graph G and a positive integer k, does G have a k-clique?
INDSET: Given a graph G and a positive integer k, does G have an independent set of size k?

Show that CLIQUE and INDSET are NP-complete by reducing 3SAT to one of these problems.
(Since we've already shown that CLIQUE and INDSET reduce to one another, it will suffice to show that either one is NP-complete.)

Clique is in NP because we can provide the k-clique and verify that it is a clique in polytime. Now, we will show that 3SAT reduces to CLIQUE, thus showing that CLIQUE is NP-hard.

Say that we are given an instance of 3SAT with n variables $x_1, \ldots, x_n$ and k clauses. For each clause k, with literals $x_1, x_2, \neg x_3$, we create three nodes $x_{1k}, x_{2k}, x_{\neg 3k}$.

Now, for every node x_{ik}, we connect it to all other nodes in the graph such that this node doesn't contradict x_{ik}. That is, we don't connect it to any node of the form $x_{\neg ij}$. Also, we don't connect it to the other two nodes for clause k.

Now, we ask if this graph has a k-clique. If so, then we must have one node from each clause selected. Further, we don't have any nodes representing opposite literals x_i and $\neg x_i$, so looking at the nodes in the k-clique will give us a satisfying assignment.

If there is no k-clique, then there is no way to choose one literal from each clause to be satisfied such that no literals are in conflict with one another. This means that there is no solution to our instance of 3SAT.

Therefore, CLIQUE is in NP and is NP-hard, so CLIQUE is NP-complete.

SUBSETSUM

Recall the following definition of problems:
SUBSETSUM: For a given finite $A \subseteq \mathbb{Z}$ and $k \in \mathbb{Z}$, determine if there is a set $S \subseteq A$ such that

$$\sum_{x \in S} x = k$$

.
PARTITION: For a given finite $A \subseteq \mathbb{Z}$ determine if there is a set $S \subseteq A$ such that

$$\sum_{x \in S} x = \sum_{k \in A \setminus S} x$$

Assume that SUBSETSUM is NP-complete. Prove that PARTITION is NP-complete.

The important part of this question is to remember exactly what we need to prove for NP-completeness. NP-complete means both NP-hard and NP. Let's start with NP-hard:

We need to show that PARTITION is NP-hard, that is, at least as hard as SUBSETSUM. This requires a reduction of SUBSETSUM to PARTITION: for an input A to SUBSETSUM, just add the element $2k - \sum_{x \in A} x$ as in recitation 7.

Next we need to show that PARTITION is in NP. Note that S in the definition of the problem is a certificate. One sentence answers are usually sufficient for proof that a problem is in NP because the problem is usually "Is there a y in x with a certain property?". Just make sure the certificate is of polynomial size, and takes polynomial time to verify.

Exactly 33% gives you an A+. All other scores get an F.

Recall the definition of 3-SAT: "Determine if a formula of the form $(\neg x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \ldots$ is satisfiable." Note that this is equivalent to each clause having at least one literal evaluate to true.

We define another problem, 1in3-SAT as "Is there a truth assignment where exactly one literal in each clause evaluates to true."
Assume 1in3SAT is NP-complete. Prove that 3-SAT is NP-complete.

3SAT is in NP, because a truth assignment is a certificate.
Take a clause in the input formula. If it has one or two literals, the proof is trivial. The core of this problem is to construct a set of clauses from three literals (a,b, and c) that evaluates to true if and only if exactly one of the literals evaluates to true. Let a', b', and c' be the negations of a, b, and c, respectively. If for some literal x, $a = x$, then $a' = x$, and if $a = \neg x$, then $a' = x$. Note that the negations are also literals. Note that the clause $(a \vee b \vee c)$ forces at least one of the three to be true. If both a and b are true, then we have a problem, so we force one of them to be false: $(a' \vee b')$. We do this with the other two pairs $(a' \vee c')$, and $(b' \vee c')$. The result is the four clauses:

$$(a \vee b \vee c) \wedge (a' \vee b') \wedge (a' \vee c') \wedge (b' \vee c')$$

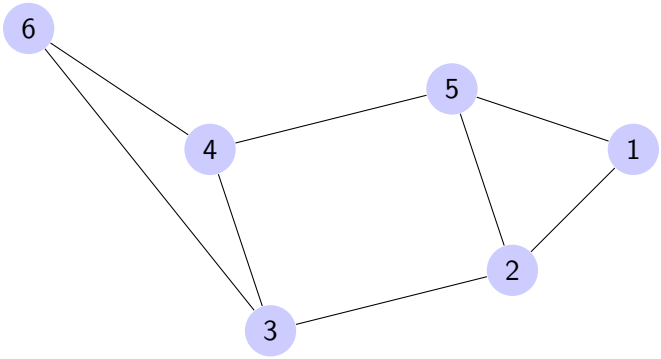
We construct these four clauses for each clause in the original formula, and pass this into the 3-SAT oracle.

Do not visit xkcd.com/230

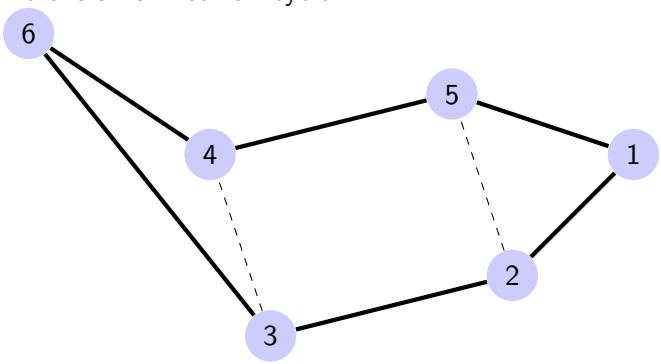
We define two very similar problems:

HAMILTONIAN-CYCLE: for a given graph G, is there a **cycle** that visits every vertex exactly once?
HAMILTONIAN-PATH: for a given graph G, is there a **path** that visits every vertex exactly once?

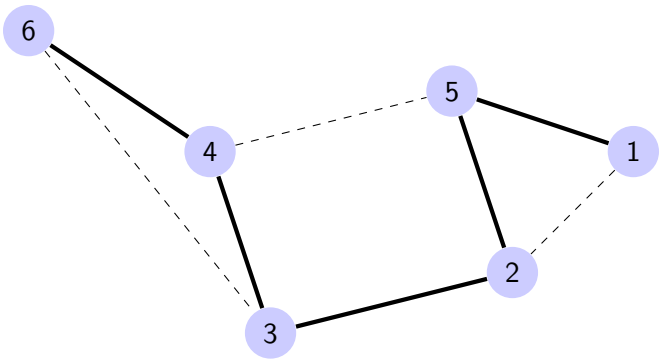
Consider an example graph:



Here is a hamiltonian cycle:



Here is a hamiltonian path:



Assume HAMILTONIAN-PATH is NP-complete. Prove that HAMILTONIAN-CYCLE is NP-complete.

HAMILTONIAN-CYCLE is in NP, because a cycle is a certificate.

We need to reduce HAMILTONIAN-PATH to HAMILTONIAN-CYCLE. Given a graph $G = (V, E)$, we want to construct a graph $G' = (V', E')$ such that G has a hamiltonian path if and only if G' has a hamiltonian cycle. Let v be a new vertex. Let $V' = V \cup \{v\}$. Let $E' = E \cup \{(v, x) \mid x \in V\}$.

Proof of correctness:

Assume G' has a hamiltonian cycle. Remove v from this cycle, and you have a hamiltonian path in G.

Assume G has a hamiltonian path. Connect both ends of this path to v, and you have a hamiltonian cycle in G'.

Assume HAMILTONIAN-CYCLE is NP-complete. Prove that HAMILTONIAN-PATH is NP-complete.

HAMILTONIAN-PATH is in NP, because a cycle is a certificate.

Given a graph $G = (V, E)$, we want to construct a graph $G' = (V', E')$ such that G has a hamiltonian cycle if and only if G' has a hamiltonian path. Attempt one: if we let $G' = G$, then it works in one direction. Specifically, if there is a cycle, then there will be a path. The problem is that it might not be possible to connect the endpoints in a path, so if there is a path, there might not be a cycle. So the intuition is to make sure that the path endpoints are connected. We don't know what the endpoints are though. Note that a vertex of degree one must be an endpoint, so if we introduce two new vertices of degree one, the path must go from one of them to the other.

Attempt two: Let x and y be two new vertices. Let a and b be two adjacent vertices in G. Let $V' = V \cup \{x, y\}$. Let $E' = E \cup \{\{x, a\}, \{y, b\}\}$. If there is an edge in G', then we can construct a cycle in G by removing x and y from the path, and connecting the endpoints. If there is a cycle in G that contains the edge $\{a, b\}$, then we can construct a path in G' starting at x, going to a, traveling the cycle until b, and then going to y. The problem is that if there is a cycle in G that does not contain the edge $\{a, b\}$, then it doesn't correspond to a path in G'. We need to change things a little. Note that for a given a (which we connect to x, we had many choices a neighbor b (which we connect to y). One of these neighbors must be adjacent to a in the cycle. What if we include every neighbor of a? Then the degree of y wouldn't be one anymore. So lets introduce another vertex z connected to y.

Attempt three: Let a be a vertex in V. Construct a graph G' from G with three new vertices x, y, and z. Connect x to a. Connect every neighbor of a to y. Connect y to z.
Proof of correctness:
Consider a path in G'. Note that the path must travel from x to a through every other vertex, to y and finally z. Remove x, y, and z from the path and connect the endpoints to form a cycle.
Consider a cycle in G. Note that the cycle must pass through a and then one of a's neighbors. Let b be this neighbor. Construct a path in G' by starting at x, going to a, traveling through the cycle (the long way) to b, to y, and finally z.