Recitation - NP

1 Definitions

- We say a language L is in NP if there exists a polynomial time verifier TM V and a constant $k > 0$ such that for all $x \in \Sigma^*$:

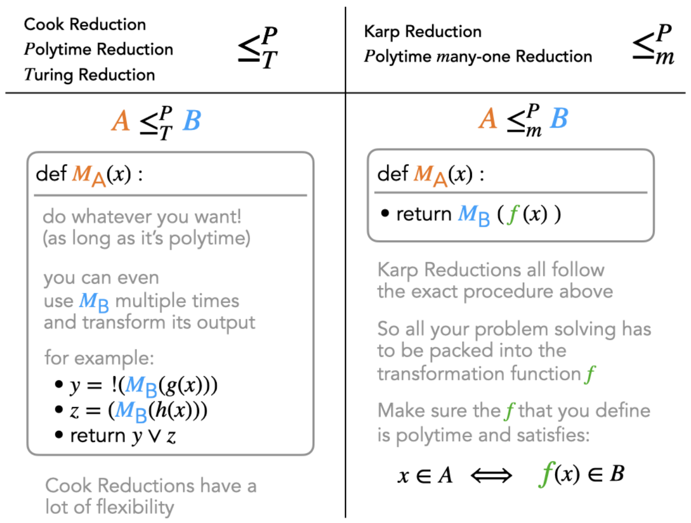
  If $x \in L$, then there exists a certificate u with $|u| \leq |x|^k$ such that $V(x, u)$ accepts.

  If $x \notin L$, then for all $u \in \Sigma^*$, $V(x, u)$ rejects.

- We say there is a **polynomial-time mapping reduction** from A to B if there is a **polynomial-time** computable function $f : \Sigma^* \to \Sigma^*$ such that $x \in A$ if and only if $f(x) \in B$. We write this as $A \leq_m^P B$. (We also refer to these reductions as Karp reductions.)

- A language Y is NP-hard if for every language $X \in$ NP, X polynomial-time reduces to Y.

- One can define hardness using either a Cook reduction (polynomial-time Turing reduction) or a Karp reduction. These lead to different definitions for NP-hardness. In this course, we use Karp reductions to define NP-hardness.


Cook Reduction
Polytime Reduction
Turing Reduction
$\leq_T^P$
Karp Reduction
Polytime many-one Reduction
$\leq_m^P$
$A \leq_T^P B$
def $M_A(x)$:
do whatever you want! (as long as it's polytime)
you can even use M_B multiple times and transform its output
for example:
• $y = !(M_B(g(x)))$
• $z = (M_B(h(x)))$
• return $y \vee z$
Cook Reductions have a lot of flexibility
$A \leq_m^P B$
def $M_A(x)$:
• return $M_B(f(x))$
Karp Reductions all follow the exact procedure above
So all your problem solving has to be packed into the transformation function f
Make sure the f that you define is polytime and satisfies:
$x \in A \iff f(x) \in B$

- A language is NP-complete if it is in NP and is NP-hard.

2 New Point of View

Imagine there existing an untrustworthy, omnipotent (computationally unbounded) **Prover** who likes to make claims about membership in a language L. On the other hand, you are a **Verifier** who can merely compute things that run in polynomial time. You are interested in verifying if a string is in L.

The **Prover** claims to you that a certain $x \in L$. In order to convince you, the **Prover** uses its unlimited computational power to provide a polynomial length (with respect to x) certificate/proof to you. You then use the certificate to verify whether x is truly in L. If $L \in$ NP then:

1. Can the **Prover** convince you for every $x \in L$ that x truly is a member of L?
2. Can the **Prover** ever fool you into thinking some $x \in L$ when really $x \notin L$?

Conversely, if L is such a language so that **Prover** can always provide you with polynomial length proofs for $x \in L$, and is never able to deceive you for $x \notin L$ then is $L \in$ NP?

Solution. `https://www.youtube.com/embed/UaMliSUGAfk`

If $L \in$ NP there exists some poly-time verifier TM V that the **Verifier** can use to verify whether a string $x \in L$. If $x \in L$, then there exists some poly-length certificate such that the verifier TM accepts, so **Prover** should be able to convince **Verifier** by providing this certificate/proof. On the other hand, if $x \notin L$, the verifier TM V is such that $\forall u \in \Sigma^*$, $V(x, u)$ rejects, so if the **Verifier** uses V then there is nothing **Prover** can do to produce a proof which convinces **Verifier** that $x \in L$ when really $x \notin L$.

For the converse note that by definition, the algorithm used by **Verifier** constitutes a poly-time verifier for L. If $x \in L$ then there exists a poly-length certificate, namely one that the **Prover** could use to convince the **Verifier**. If $x \notin L$ then regardless of what proof **Prover** sends over, the algorithm used by **Verifier** will not accept. ■

3 Network Pairs

Let $G = (V, E)$ be a graph. A clique in G is a set $S \subseteq V$ such that for any $u, v \in S$, $\{u, v\} \in E$ (so a clique is a set of vertices such that any two vertices in the set is connected by an edge). We are interested in the following problem that we call DCLIQUE (double clique): Given a graph $G = (V, E)$ and $k \in \mathbb{N}^+$, does G contain two vertex-disjoint cliques of size k each? As a language, this corresponds to:

DCLIQUE $= \{\langle G, k\rangle : G$ is a graph, $k \in \mathbb{N}^+$, G contains two vertex-disjoint cliques of size $k\}$

Show DCLIQUE is NP-complete.

Solution. https://www.youtube.com/embed/p3avwg2FhAM

This problem is in NP. Here is the description of a polynomial-time verifier TM.

```
def $A(\langle \text{graph } G = (V, E), \text{positive natural } k\rangle,\ u)$ :
  If $u$ does not correspond to two sets $S, T \subseteq V$ with $|S| = |T| = k$, reject.
  For each pair of vertices $u, v \in S$:
    If $\{u, v\} \notin E$, reject.
  For each pair of vertices $u, v \in T$:
    If $\{u, v\} \notin E$, reject.
  For each vertex $u \in S$:
    If $u \in T$, reject.
  Accept.
```

To prove its correctness, we have to argue the following.

1. If $x \in$ DCLIQUE, there is some proof string u of polynomial-length that makes A accept.

   If $x \in$ DCLIQUE, then $x = \langle G, k\rangle$ is a valid encoding and G contains 2 disjoint cliques of size k. When u is a valid encoding of a list of the two disjoint cliques, the verifier will accept (as in this case, none of the `reject` instructions will be reached).

2. If $x \notin$ DCLIQUE, no matter what u is, A REJECTS.

   If $x \notin$ DCLIQUE, then there are 2 options: x is not a valid encoding $\langle G, k\rangle$, or x is valid but G does not contain 2 disjoint cliques of size k. In the first case, A rejects (this is the implicit "type checker"). In the second case, A also rejects, because by design, the only way the verifier can get to the last line and accept is if the graph contains two disjoint cliques of size k each.

3. A is polynomial-time.

   It is relatively easy to see that each line of the algorithm can be carried out in polynomial time with respect to the number of vertices in the input graph.

To show DCLIQUE is NP-hard, we will Karp-reduce CLIQUE to DCLIQUE. So given an instance of CLIQUE, $\langle G, k\rangle$, we want to transform it to an instance of DCLIQUE, $\langle G', k'\rangle$, so that G has a clique of size k if and only if G' has two disjoint cliques of size k'. The main idea behind the reduction is to define G' as two disjoint copies of G. This way for any clique in G, there will be two disjoint copies of it in G'. We set $k' = k$.

Here is the definition of our Karp reduction $f : \Sigma^* \to \Sigma^*$.

```
def $f(\langle \text{graph } G = (V, E),\ \text{positive natural } k\rangle)$ :
  $V^1 = \{v^1 : v \in V\}$.
  $E^1 = \{\{u^1, v^1\} : \{u, v\} \in E\}$.
  $V^2 = \{v^2 : v \in V\}$.
  $E^2 = \{\{u^2, v^2\} : \{u, v\} \in E\}$.
  Return $\langle G' = (V^1 \cup V^2, E^1 \cup E^2), k\rangle$.
```

We now prove the correctness of the reduction.

- If $x \in$ CLIQUE, then $x = \langle G, k\rangle$ where G contains a clique of size k. Since G' contains two disjoint copies of G and each of those copies will contain a clique of size k, G' contains two disjoint cliques of size k. This means that $f(x) \in$ DCLIQUE.
- If $f(x) \in$ DCLIQUE, then $f(x) = \langle G', k\rangle$ and G' contains two disjoint cliques of size k each. We know by construction of f that $x = \langle G, k\rangle$ and G' consists of two disjoint copies $G^1 = (V^1, E^1)$ and $G^2 = (V^2, E^2)$ of G. Since the copies are disjoint, a clique in G' of size k is either completely contained in V^1 or is completely contained in V^2. This means the original graph G must have a clique of size k. That is, $x = \langle G, k\rangle \in$ CLIQUE.
- The function f is polynomial time computable since given a graph G, we can create two copies of it in polynomial time.

Since DCLIQUE is both in NP and NP-hard, it is NP-complete. ■

4 No Peeking

Let $G = (V, E)$ be a graph. A vertex cover in G is a set $C \subseteq V$ such that for every edge $\{x, y\} \in E$, either $x \in C$ or $y \in C$ (a set of vertices such that every edge is incident to at least one vertex in the set). An independent set in G is a set $S \subseteq V$ such that for any $u, v \in S$, $\{u, v\} \notin E$ (a set of vertices such that no edge connects two vertices in the set). Define the following languages:

VERTEX-COVER = $\{\langle G, k\rangle : G$ is a graph, $k \in \mathbb{N}$, G contains a vertex cover of size $k\}$

IND-SET = $\{\langle G, k\rangle : G$ is a graph, $k \in \mathbb{N}$, G contains an independent set of size $k\}$

Show that VERTEX-COVER $\leq_m^P$ IND-SET.

Solution. https://www.youtube.com/embed/dNJA2g2J1TQ

We will define a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that

$$x \in \text{VERTEX-COVER} \leftrightarrow f(x) \in \text{IND-SET}.$$

Here is the definition of our reduction:

```
def  $f(\langle \text{graph } G = (V, E),\ \text{positive natural } k\rangle)$ :
  Return $\langle G, |V| - k\rangle$.
```

To prove the correctness of this reduction, we need to argue that (i) if $x \in$ VERTEX-COVER, then $f(x) \in$ IND-SET; (ii) if $f(x) \in$ IND-SET, then $x \in$ VERTEX-COVER; (iii) f is polynomial-time computable. The first two points will follow from the observation that in a graph $G = (V, E)$, a set of vertices $S \subseteq V$ is a vertex cover if and only if $V \backslash S$ is an independent set.

- (i) If $x \in$ VERTEX-COVER then x is a valid encoding $\langle G = (V, E), k\rangle$. In addition, there exists a vertex cover of size k. Let S be the subset of vertices in the vertex cover and consider $V \setminus S$. Observe that $V \setminus S$ is an independent set. To see why, consider any pair of vertices $u, v \in V \setminus S$. Because neither u nor v are in S but S is a vertex cover, we know that $(u, v) \notin E$. In addition, $|V \setminus S| = |V| - k$, so there exists a independent set of size $|V| - k$. Hence $f(x) \in$ IND-SET
- (ii) If $f(x) \in$ IND-SET, then G must has an independent set of size $|V| - k$. Let S be the subset of vertices in the independent set. Similar to the previous direction, $V \setminus S$ is a vertex cover. To show this, consider an arbitrary edge $(u, v) \in E$. Because S is an independent set, we know one of $u, v \notin S$. Equivalently, one of $u, v \in V \setminus S$. Thus, all edges are covered by some vertex in $V \setminus S$ so G has a vertex cover of size $|V| - (|V| - k) = k$. So $x \in$ VERTEX-COVER.

- (iii) f is poly-time since we just need verify that the encoding is valid, count the number of vertices, and then compute $|V| - k$, which can all be done in poly-time.

■

5 (Extra) Never Pausing

Prove that HALTS is NP-hard.

Solution. https://www.youtube.com/embed/kaG4U8stl0E

Recall that HALTS $= \{\langle M, x\rangle : M$ is a TM, $x \in \Sigma^*$, and $M(x)$ halts$\}$.

To show HALTS is NP-hard, we will reduce 3SAT to HALTS. First note that 3SAT is decidable since it is in NP. In particular, we can do a brute-force search: Given a 3CNF formula with n variables, try all 2^n possibilities for the variable assignment, then check if that assignment satisfies all the clauses.

We now present a Karp reduction from 3SAT to HALTS.

```
def f(⟨3CNF formula $\varphi$⟩) :
  def HELP(y):
    brute force to check if $\varphi$ is satisfiable
    if $\varphi$ is satisfiable, then halt
    if $\varphi$ is not satisfiable, then loop
  return ⟨HELP, $\varepsilon$⟩.
```

This transformation can be carried out in polynomial time since the description of HELP can be created in polynomial time. This is because creating the description of HELP only requires φ to be hard-coded; the remainder of the description is just a constant.

Note: Notice that HELP itself is not a polynomial-time algorithm. This does not contradict the fact that f is polynomial time computable since f does not run HELP. It merely creates the string representation of it.

We now prove that the reduction is correct:

$$\begin{aligned} x \in \text{3SAT} &\iff x = \langle \varphi \rangle \text{ where } \varphi \text{ is a satisfiable 3CNF formula} \\ &\iff \text{HELP halts no matter what its input is} \\ &\iff \langle \text{HELP}, \varepsilon \rangle \in \text{HALTS}. \end{aligned}$$

■