Decision Problems

Why formulate problems as decision problems ?

1. Decision problems are in essence easier than the corresponding optimization problems. So by proving the decision problem is hard, the optimization problem must be as least as hard
2. In many cases, both the decision problem and its corresponding optimization problem are equivalent

$$\mathcal{P} = \{ \text{ Problems that can be solved in polynomial time } \}$$

$$\mathcal{NP} = \{ \text{ Problems that can be verified in polynomial time } \}$$

Proposition. ***Naive algorithm for $\mathcal{NP}$ problems***
*All problems in $\mathcal{NP}$ can be solved by a **generate-and-verify** algorithm with the following structure*

```
Function Generate-and-Verify(x)
    Generate all certificates for each c ∈ certificates do
        if Verify (x, c) then
            return True
    return False
```

Example. Composite
Given positive integer x, does x have any factor (i.e. a composite number)

```
Function Composite(x)
    Generate all certificates for for all integer c ∈ 2 to x − 1 do
        if c divides x then
            return True
    return False
```

Definition. *B is an efficient **certifier** for a problem X, if the following properties hold:*

1. *B is a polynomial time algorithm that takes two input s, the input, and t, the certificate and returns either yes or no to the problem X*
2. *There is a polynomial time function p such that for every string $s = \{0, 1\}*$, we have $s \in X$, i.e. s is a yes solution to X, if and only if there exists a string t such that $|t| \leq p(|s|)$ (i.e. $|t| \leq |s|^2$ say) and $B(s, t)$ is yes (implies the loop is not over infinite number of times, i.e. there is an upper bound on the loop)*

$s \in X$ means s is an yes instance of X

Theorem.

$$\mathcal{P} \subseteq \mathcal{NP}$$

Proof. Consider a decision problem $X \in \mathcal{P}$. This implies there exists a polynomial time algorithm A that solves X. To show $X \in \mathcal{NP}$, we want to find an efficient verifier $B(s, t)$ for X, such that $B(s, t) = A(s)$ for any t. Dont know... □

Definition. *A problem X is called $\mathcal{NP}$-complete (NPC) if*

1. $x \in \mathcal{NP}$
2. *$\mathcal{NP}$-hard, i.e. for every $Y \in \mathcal{NP}$, $Y \leq_p X$*

Example. Prove NP

1. Independent Set $\in \mathcal{NP}$. For the set $S = \{v_1, \cdots, v_n\}$ The verification steps takes every $v_i \in S$ and check $G.Adj[v_i]$ (totals to $|E|$) takes $k(|V| + |E|)$ steps at most. But $k \leq |V|$ so takes $|V|(|E| + |V|)$
2.

Definition. *X in $\mathcal{NP}$ means there is a certifier $B(s, t)$ running in polynomial time such that*

$$B(s,t) = \begin{cases} true & \text{for some } t \text{ where } s \text{ is a yes instance of } X \\ false & \text{for all } t \text{ where } s \text{ is a no instance of } X \end{cases}$$

co$\mathcal{NP}$ is the complement of problems in $\mathcal{NP}$, i.e. problems whose no-instances are easy to verify

$$B(s,t) = \begin{cases} true & \text{for all } t \text{ where } s \text{ is a yes instance of } X \\ false & \text{for some } t \text{ where } s \text{ is a no instance of } X \end{cases}$$

Example. Examples of $co\mathcal{NP}$ algorithms

1. **Prime**

```
Function Prime ∈ coNP
    For input x
    for c = 2, ··· , x − 1 do
        if c divides x then
            return False
    return True
```

2. **Dense set** $\in co\mathcal{NP}$

Proposition.

$$\mathcal{P} \subseteq co\mathcal{NP}$$

NP-completeness problems are hardest problems in NP

Example. **SAT** (Satisfiability problems)

1. **Circuit-SAT** Given a circuit with AND, OR, NOT gates and input set I and a single output x. The question asks if there is a set of I such that $x = T$ (satisfiability) If the answer is *yet*, the circuit is **satisfiable** otherwise **unsatisfiable**.

2. **SAT** Since any circuit can be transformed into a boolean expression, is an equivalent question for such boolean formula, i.e. if the value of formula yields $true$ or $false$

3. **CNF-SAT** Conjunctive normal form.

$$\phi = c_1 \wedge \cdots c_i \cdots \wedge c_k \qquad c_i = (t_{i_1} \vee \cdots \vee t_{i_j})$$

   where $t_{i_j} = x_j$ or $\neg x_j$. Note every boolean expression can be converted to CNF

4. **3-SAT** A special form of CNF-SAT where each clause c_i has exactly 3 literals. Again can convert from every CNF.

Theorem. *SAT is NPC. Given a boolean formula ϕ, ask the question if ϕ is satisfisable. Let X be SAT-family of problems*

Proof. Prove by definition of NPC (2-part)

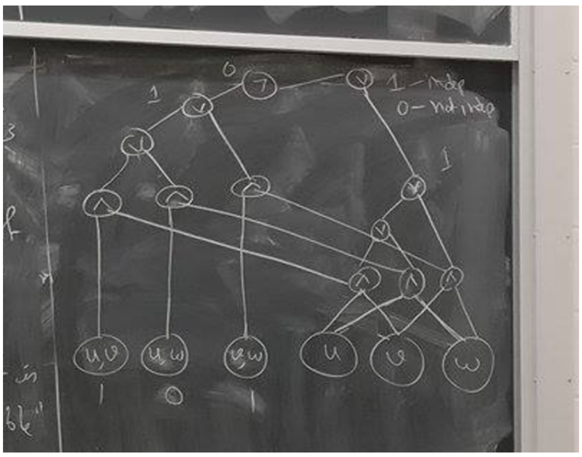
1. Prove $X \in \mathcal{NP}$. Given ϕ where t is the truth assignment of the variables in ϕ. Given t, verify ϕ is true is easy, since can just substitute variable in and evaluate using boolean expression. Hence can be verified easily

2. Prove $Y \leq_q X$ for all $Y \in NP$. Basic idea, every NP problem can be reduced to circuits (Circuit-SAT).

□

Example. Given a graph $G = (V, E)$ Does it contain a 2 node independent set.

Solution. □



Definition. ***Techniques for prooving NPC***
To prove X is NP-hard, use a known NP-hard problem Y and show that $Y \leq_q X$

Proof. Note If $A \leq_p B$ and $B \leq_p C$ then $A \leq_q C$. So if Y is NP-hard, then $\forall Z \in \mathcal{NP}$, $Z \leq_p Y$. And since $Y \leq_p X$, so $\forall Z \in \mathcal{NP}$, $Z \leq_p X$, so X is NP-hard □

Proposition. ***3-SAT*** *is NPC*

Proof. Idea: Reduce it to SAT, which was shown to be NP-hard

1. 3-SAT $\in NP$, true...

2. Now we prove $CNF - SAT \leq_p 3 - SAT$. Given a formula ϕ in CNF, obtain an formula $\phi^{\texttt{Prime}}$ in 3-SAT, such that ϕ is satisfiable if and only if $\phi^{\texttt{Prime}}$ is satisfiable. Let $\phi = c_1 \wedge \cdots \wedge c_r$ where $c_i = (x_{j_i} \vee \cdots \vee x_{j_k})$ Want to size of each c_i to 3. For each c_i in ϕ, if

   (a) If $c = (a_1)$ then replace c with $(a_1 \vee a_1 \vee a_1)$.

   (b) If $c = (a_1 \vee a_2)$, then replace c with $(a_1 \vee a_1 \vee a_2)$.

   (c) If $c = (a_1 \vee a_2 \vee a_3)$, then leave it as is.

   (d) If $c = (a_1 \vee a_2 \vee \cdots \vee a_s)$ where $s > 3$, then replace c with $c^{\texttt{Prime}} = (a_1 \vee a_2 \vee a_3) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \wedge \cdots \wedge (\neg z_{s-4} \vee a_{s-2} \vee z_{s-3}) \wedge (\neg z_{s-3} \vee a_{s-1} \vee a_s)$ where $(z_1, \cdots, z_{s-3})$ are new variables. Now we prove c is satisfiable if and only if $c^{\texttt{Prime}}$ is satisfiable

      i. (=>) There is a truth assignment $a_1, \cdots a_s$ that makes c true. This implies that there is some $a_i = T$ the first i such that $a_i = T$.

         A. If $i = 1$ or 2, let $z_1, \cdots, z_{s-3} = F$, then every clause is true.

         B. If $i = s-1$ or s, let $z_1, \cdots, z_{s-3} = T$, then every clause is true

         C. If $2 < i < s-1$, let $z_1, \cdots, z_{i-2} = T$ and let $z_{i-1}, \cdots, z_{s-3} = F$, then every clause is true

      ii. (<=) If there is a truth assignment that makes $c^{\texttt{Prime}}$ true, we want to show that there is a truth assignment that makes c is true. Obvious, the same assignment works

□