Notes, 10(b)

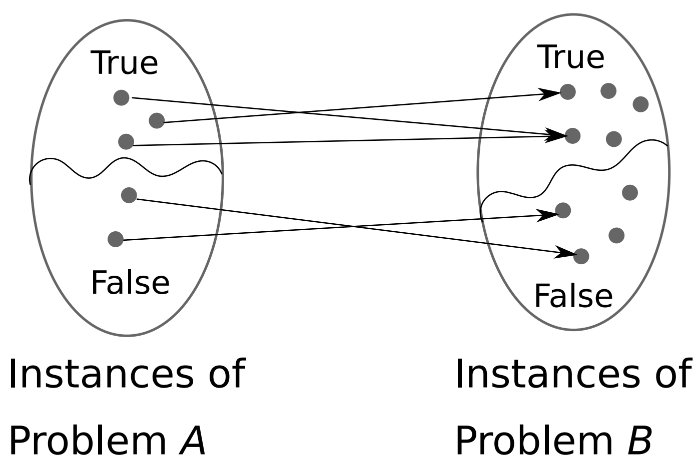
ECE 606

The Karp reduction

Given two decision problems A, B, suppose the set of all instances of problem A is denoted I_A and the set of all instances of problem B is denoted I_B. We say that A Karp-reduces to B, written $A \leq_k B$ if and only if:

> there exists a polynomial-time computable function $m : I_A \to I_B$ such that $i \in I_A$ is a `true` instance of problem A if and only if $m(i) \in I_B$ is a `true` instance of problem B.

Note: in the portions of your textbook that I have taken from CLRS, $\leq_k$ is written as $\leq_p$.



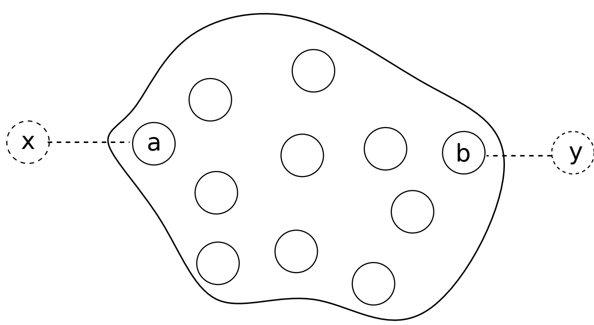
Claim 1. HamPathStartEnd $\leq_k$ HamPath.

HamPath: given as input a non-empty undirected graph G, is there a simple path in G of all its vertices?
HamPathStartEnd: given as input a non-empty undirected graph G and two distinct vertices in it, a, b, is there a simple path $a \rightsquigarrow b$ in G of all its vertices?

Our algorithm to show $\leq_c$ turns out to be a mapping that works.

$H_{SE}(G = \langle V, E \rangle, a, b)$

if $a = b$ *or* $|V| = 1$ **then return** false
Let $V' \leftarrow V \cup \{x, y\}$, where $x, y \notin V$
Let $E' \leftarrow E \cup \{\langle x, a \rangle, \langle b, y \rangle\}$
return $H(\langle V', E' \rangle)$



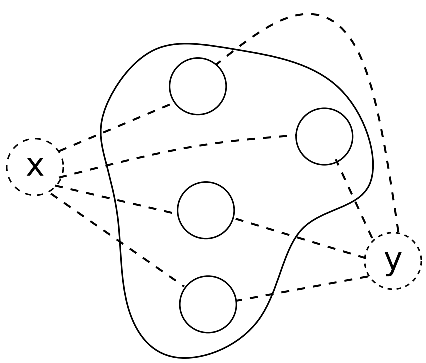
Claim 2. *HamPath* $\leq_k$ *HamPathStartEnd.*

For this direction, the algorithm we used to show $\leq_c$ does not work. We have to get a bit more creative.

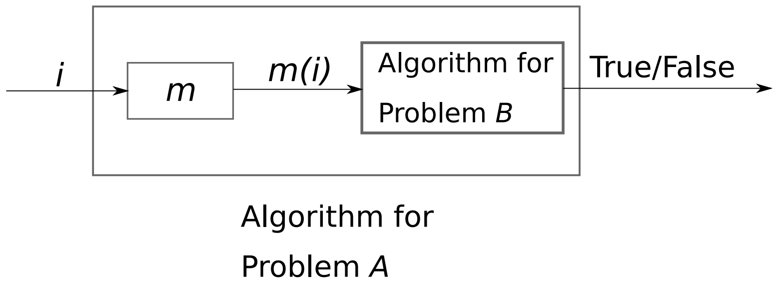
$H(G = \langle V, E \rangle)$

if $|V| = 1$ **then return** true
foreach $u \in V$ **do**
    foreach $v \in V \setminus \{u\}$ **do**
        if $H_{SE}(G, u, v) =$ true **then return** true
return false

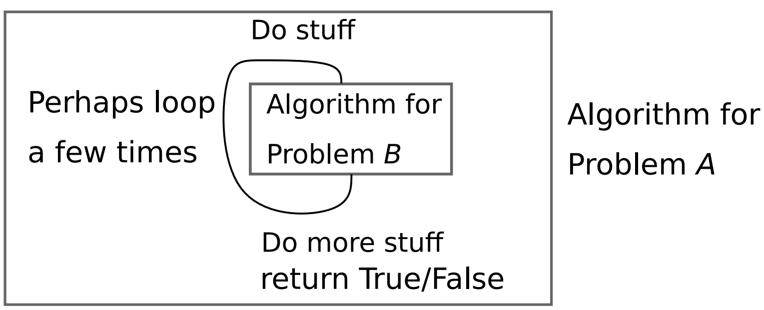
A mapping: introduce two new vertices, x, y. For every vertex u in the original graph, introduce edges $\langle x, u \rangle$ and $\langle y, u \rangle$. Let this new graph be G'. Output from the mapping: $\langle G', x, y \rangle$.



If $A \leq_k B$, then to construct an algorithm for A given an algorithm for B:



Compare to $A \leq_c B$:



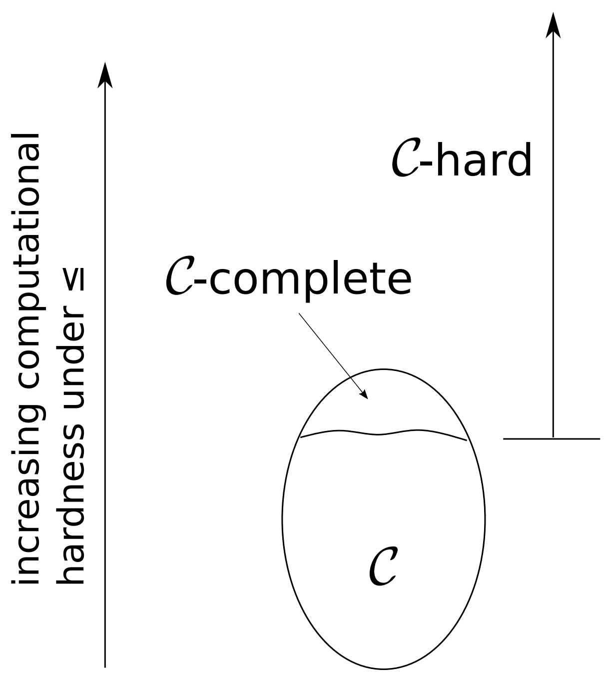
Claim 3. *$A \leq_k B \implies A \leq_c B$.*

Claim 4. *$\boldsymbol{NP}$ is closed under $\leq_k$. That is: if $A \leq_k B$ and $B \in \boldsymbol{NP}$, then $A \in \boldsymbol{NP}$.*

Now that we have a notion of a reduction to compare the computational hardness of two problems, we move on to comparing the hardness of a problem relative to an entire complexity class.

Definition: Given a complexity class $\mathcal{C}$, we say that a decision problem p is $\mathcal{C}$-hard under a reduction $\leq$ if: for all $q \in \mathcal{C}$, $q \leq p$.

Definition: Given a complexity class $\mathcal{C}$, we say that a decision problem p is $\mathcal{C}$-complete under a reduction $\leq$ if: (i) p is $\mathcal{C}$-hard under $\leq$, and, (ii) $p \in \mathcal{C}$.



We can instantiate $\leq$ with $\leq_k$, $\leq_c$ or any other notion of a reducion we think is meaningful.

In the context of **NP**, we usually adopt $\leq_k$.

NP-hard

A decision problem p is **NP**-hard if for all $q \in \mathbf{NP}$, $q \leq_k p$.

NP-complete

A decision problem p is **NP**-complete if: (i) p is **NP**-hard under $\leq_k$, and, (ii) $p \in \mathbf{NP}$.

Claim 5. *If $p \in$ **NP**-hard and $p \leq_k q$, then $q \in$ **NP**-hard.*

Claim 6. *If $q \in$ **NP**-complete and $q \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.*

So: if $\mathbf{P} \neq \mathbf{NP}$ and $q \in \mathbf{NP}$-complete, then $q \notin \mathbf{P}$.