Examples of decision problems:

**SUBGRAPH ISOMORPHISM**

**INSTANCE**: Two graphs \( G_1 = (V_1, E_1) \), \( G_2 = (V_2, E_2) \).

**QUESTION**: Does \( G_1 \) contain a subgraph isomorphic to \( G_2 \)?

**TRAVELING SALESMAN**

**INSTANCE**: A finite set \( C = \{c_1, \ldots, c_m\} \) of "cities", a set of distances \( d(c_i, c_j) \) for all \( c_i, c_j \in C \) and a bound \( B \leq \infty \).

**QUESTION**: Is there a tour of all the cities in \( C \) having total length no more than \( B \), i.e.

an ordering \( < c_{\pi(1)}, \ldots, c_{\pi(m)} > \) of \( C \) such that:

\[
\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B.
\]

**Def**: An alphabet \( \Sigma \) is any finite set of symbols.

A language over \( \Sigma \) is any subset of \( \Sigma^* \) (all finite words over \( \Sigma \)).

**Def**: An encoding scheme \( e \) for a decision problem \( \Pi \) is a way of describing each instance of \( \Pi \) by an appropriate string of symbols over some fixed alphabet \( \Sigma \).

**Def**: \( L(\Pi, e) = \{ x \in \Sigma^* \mid \exists b \in \Sigma^* \text{ the alphabet used by } e \text{ and } x \text{ is the encoding under } e \text{ of an instance } I \in \text{Yes}(\Pi) \} \)

**Note**: As long as we restrict ourselves to "reasonable" encodings, "significant" properties of \( L(\Pi, e) \)...
are independent of e. In the similar way, we can associate with a problem \( \Pi \) an encoding-independent function \( \text{Length}: D_e \rightarrow \mathbb{R} \) st. for any "reasonable" encoding \( e \), for any instance \( I \) of \( \Pi \):

\[
\text{Length}(I) \leq p(|x|) \quad (1)
\]

\[
|x| \leq p'(\text{Length}(I)) \quad (2)
\]

where \( |x| \) is the length of the string \( x \), \( p \) and \( p' \) polynomials, \( x \) the string corresponding to \( I \) under \( e \).

**Example:** If the instance \( I \) of a graph \( G = (V, E) \), then

\[
\text{Length}(I) = |V| \text{ is o.k. (Note: } L(1) \leq 12 \text{)}
\]

Thus (2) is also necessary.

What do we mean by a "reasonable coding"?

i) (conciseness) - natural gravity with no padding

ii) (decodability) - given any particular instance we have a polynomial time algorithm that can extract a description of the "essential" components of the encoded instance.

For convenience we accept in DTM larger alphabet \( \Pi \) for the contents of the tape squares instead of \( \{0,1\} \), and a subset \( \Sigma \subset \Pi \) as the alphabet for input strings.
Def: A DTM program $M$ with input alphabet $\Sigma$ accepts $x \in \Sigma^*$ iff $M$ halts in state $\text{Yes}$ when applied to $x$.

Def: The language $L_M$ recognized by the program $M$ is given by

$$L_M = \{ x \in \Sigma^* \mid M \text{ accepts } x \}$$

"If we want that a DTM program corresponds to our notion of an algorithm, it must halt on all possible strings over its input alphabet."

Def: A DTM program $M$ solves the decision problem $\Pi$ under encoding scheme $e$ if $M$ halts for all possible strings of the input alphabet and $L_M = L(\Pi, e)$.

Def: The time used in the computation of a DTM program $M$ on an input $x$ is the number of steps occurring in that computation up until a halt state is entered.

Def: For a DTM program $M$ that halts for all inputs $x \in \Sigma^*$, its time complexity function $T_M : \omega \rightarrow \omega$ is given by

$$T_M(n) = \max \{ m \mid \exists x \in \Sigma^* \ (|x| = n \land \text{time used in computing with input } x \leq m) \}$$
Theorem: A program is called a polynomial time DTM program if there exists a polynomial $p$ such that for all new
\[ T_M(n) \leq p(n) \]

\textbf{Def.} $P = \{ L |$ there is a polynomial time DTM program $M$ for which $L = L_M \}$.

\textbf{Def.} We will say that a decision problem $P$ belongs to $P$ iff for some (any) reasonable encoding $e$, $L(P, e) \in P$.

\textbf{Note.} Each such DTM program has its informal counterpart — "polynomial time algorithm".

\textbf{Note.} Because of "polynomial" equivalence between "realistic" computers, we will not tie ourselves to DTM's.
Nondeterministic algorithm that solves a decision problem \( \Pi \) consists of two stages: a guessing stage and a checking stage; in the guessing stage we produce a "structure" \( S \) and then the checking stage proceeds in a deterministic manner with input \( I,S \). A nondeterministic algorithm "solves" a decision problem \( \Pi \) if for all instances \( I \in \text{D}_\Pi \):

i) \( I \in \text{YES}_\Pi \) then for some \( S \) when guessed at input \( I \) the checking stage leads to yes.

ii) \( I \notin \text{YES}_\Pi \) no such \( S \) exists.

A nondeterministic algorithm that solves a decision problem \( \Pi \) is said to operate in polynomial time if for some polynomial \( p \) for any \( I \in \text{Y}_\Pi \) there exists a guess \( S \) such the deterministic checking stage responds yes within time \( p(\text{length}(I)) \).

If we want "essential" help from \( S \), we must examine it entirely: since only a polynomially bounded amount of time can be spent examining guess \( S \), we also have a polynomial bound on the size of \( S \).

Obviously \( TS \) and \( SB6150 \) are NP-computable (by guessing the route, subgraph and the mapping).
6 Formalizing NDP-computability

**Def:** A nondeterministic Turing machine \( NDTM = DTM + \text{guessing module with write-only head} \)

It works in two stages: guessing stage first where it guesses and writes down starting from \(-1\) any word \( \in \Sigma^* \) and arbitrarily decides when to stop (if even guessing and start the second, checking stage which works identically as DTM.

If it stops in a state \( Y \) we call it an accepting computation (guessing included).

If it stops in a state \( N \) or doesn't stop at all we call it a non-accepting computation.

Thus, for any input string \( x \) any NDTM program \( M \) will have an infinite number of possible computations, one for each possible guessed string from \( \Gamma^* \).

**Def:** The language recognized by \( M \) is

\[
L_M = \{ x \in \Sigma^* : M \text{ accepts } x \}
\]
The time required by an NDTM program $M$ to accept the string $x \in \mathcal{L}_M$ is defined to be the minimum over all accepting computations of $M$ for $x$ of the number of steps occurring in the guessing and deciding stages up until the halt state $y^*$ is entered.

**Def.** The time complexity function $T_M : \omega \rightarrow \omega$

for $M$ is

$$T_M(n) = \max\left\{ t \mid \exists x \in \mathcal{L}_M (|x| = n \text{ s.t. the time to accept } x \text{ by } M = t) \right\}.$$ 

**Def.** The NDTM program $M$ is a polynomial time program if there exists a polynomial $p$ s.t. $T_M(n) \leq p(n)$ for all $n \geq 1$.

**Def.** $NP = \{ L \mid \text{there is a polynomial time NDTM program } M \text{ s.t. } L_M = L \}$.

**Def.** A decision problem $P$ belongs to $NP$ iff for some reasonable encoding $e$, $L(P,e) \in NP$.

**Remark.** The above model of NDTM is "polynomially" mutually replaceable by many others.

**Remark.** Obviously $P \subseteq NP$. 
Theorem: If $\Pi \in \text{NP}$, then there exists a polynomial \( p \) s.t. \( \Pi \) can be solved by a deterministic algorithm having time complexity \( O(2^{p(n)}) \). \((n = |\Pi|)\).

Proof: Let \( A \) be a polynomial time nondeterministic algorithm for solving \( \Pi \) with \( 2(n) \) as a bound on the time complexity of \( A \). For any \( n \)
\( 2(n) \) can be evaluated in a polynomial time; then we can repeat \( 2^{2(n)} \) times checking procedure, for each possible guess \( \gamma \) of length \( \leq 2(n) \), which would take
\( c \cdot 2(n) \cdot 2^{2(n)} \) steps which is \( O(2^{p(n)}) \) for a suitable \( p(n) \). \((|\Pi| = k)\).
**Def:** A polynomial transformation from a language \( L_1 \subseteq \Sigma_1^* \) to a language \( L_2 \subseteq \Sigma_2^* \) is a function \( f: \Sigma_1^* \rightarrow \Sigma_2^* \) s.t.

(i) There is a polynomial time DTM program that computes \( f \).

(ii) For all \( x \in \Sigma_1^* \), \( x \in L_1 \) iff \( f(x) \in L_2 \)

We note it by \( L_1 \preceq L_2 \); \((\mathcal{M}-1 \text{ red})\)

**Lemma:** \( \text{LEP} \) and \( L_1 \preceq L_2 \) then \( L_1 \in \text{EP} \)

**Proof:** Let \( f \) polynomially transforms \( L_1 \) into \( L_2 \), \( M_f \) DTM computing \( f(x) \) in less than \( \text{P}_f(\|x\|) \) steps, and \( M \) DTM program recognizing \( L \) in less than \( \text{P}(\|x\|) \) steps. Then the following program recognizes \( L_1 \): compute \( f(x) \) by \( M_f \) and then check by \( M \) if \( f(x) \in L \); number of steps needed is less than \( O(\text{P}_f(\|x\|) + \text{P}(\text{P}_f(\|x\|))) \) since \( \text{length} f(x) \leq \text{P}_f(\|x\|) \).

**Example:** \( \text{HC} \preceq \text{TS} \), where \( \text{TS} \) is the Traveling salesman and \( \text{HC} \) the Hamiltonian circuit problem.

**HC**

**Instance:** A graph \( G = (V,E) \)

**Question:** Does \( G \) contain a Hamiltonian circuit?

Where a Hamiltonian circuit is a path through the graph visiting each node exactly once and coming back into the starting pt.
**Proof:** Define $f : (V,E) \rightarrow (C,D,B)$ by: $C = V$, $D = \{ d_{ij}(c_i,c_j) \mid i,j \leq |V| \}$, where
\[
\begin{align*}
d_{ij}(c_i,c_j) &= 1 & \text{if} & & (c_i,c_j) \in E \\
d_{ij}(c_i,c_j) &= 2 & \text{if} & & (c_i,c_j) \notin E
\end{align*}
\]
$B = |V|$. Obviously, $\exists$ path $\leq B \iff \exists C$.

**Lemma:** $\exists$ is transitive.

**Proof:** Assume $\Sigma_1, \Sigma_2, \Sigma_3$ are alphabets of the languages $L_1, L_2, L_3$ and $f_1 : \Sigma_1^* \rightarrow \Sigma_2^*$, $f_2 : \Sigma_2^* \rightarrow \Sigma_3^*$ such that $f_1(f_2)$ is polynomially computable in less than $P_1(P_2)$ steps. Then $f_3(f_1(x))$ is a transformation from $L_1$ into $L_3$, which is polynomial:

- $f_1(x)$ can be computed in $P_1(|x|)$ steps.
- $f_2(f_1(x))$ can be computed in $P_2(|f_1(x)|)$ steps.
- Since $|f_1(x)| \leq P_1(|x|)$ (worst case: program only writes to the front of $\Sigma_2$), $f_3(f_1(x))$ can be computed in $P_2(P_1(|x|))$ steps.

**Def:** $L_1$ and $L_2$ are polynomially equivalent iff $L_1 \subset L_2$ and $L_2 \subset L_1$.

**Def:** A language $L$ is defined to be **NP-complete** iff for any other NP-language $L_1$, $L_1 \subset L_1$.
Lemma: If $L_1$ and $L_2$ belong to NP, $L_1$ is NP-complete and $L_1 \times L_2$, then $L_2$ is NP-complete.

Proof: If $L_3$ is NP then $L_3 \times L_1 \times L_2 \Rightarrow L_3 \times L_2$.

Corollary: General strategy: to show that $\Pi$ is NP complete we have to:

(i) show that $\Pi$ is NP
(ii) find a NP complete problem $\Pi'$ s.t. $\Pi' \times \Pi$. 


SATISFIABILITY (SAT)

INSTANCE: A set \( U \) of variables and a propositional formula \( C \) which is a conjunction of disjunctions of \( \pm \) the variables.

QUESTION: Is there a satisfying truth assignment for \( C \)? (i.e., is \( C \) satisfiable?)

THEOREM SAT is NP-complete.

Proof: SAT is obviously NP, since having a guess for \( \phi \), we can check if \( \phi(U) = T \) or \( F \) in polynomial time with respect to \( \ell \phi(U, C) \).

Let \( L \) be any NP language accepted by \( M \) specified by \( \Gamma, \Sigma, \beta \) (= blank), \( \emptyset \) (= all states), \( Z_0 \) (= start), \( Z_Y \) (= final yes), \( Z_N \), \( S \) (= mapping), with \( 3 \)-tuple \( p(1 \times 1) \).

We have to construct \( f \) s.t. for any \( x \), \( f(x) \) is a formula \( \forall x \) and \( x \in L \) iff \( \forall x \) is satisfiable, and \( f \) is \( P \)-computable.

Since any \( x \in L \) is accepted within \( p(1 \times 1) \) steps any such computation has used \((p(1 \times 1), p(1 \times 1) + \) tape cells.

Let 
\[
\begin{align*}
\Theta &= \{ q_0, q_1 (= q_Y), q_2 (= q_N), q_3, \ldots, q_5 \} \\
\Gamma &= \{ s_0 (= \emptyset), s_1, \ldots, s_5 \}
\end{align*}
\]

we will define three types of variables each with a clear intended meaning; we let \( n = 1 \times 1 \).
<table>
<thead>
<tr>
<th>Variable</th>
<th>Domain</th>
<th>Intended Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q(i,k) )</td>
<td>( 0 \leq i \leq P(n) ), ( 0 \leq k \leq 2 )</td>
<td>At ( i )th step ( M ) is in state ( 2k )</td>
</tr>
<tr>
<td>( H(i,j) )</td>
<td>( 0 \leq i \leq P(n) ), ( P(n) \leq j \leq P(n) + 1 )</td>
<td>At ( i )th step the read-write head is scanning the ( j )th tape cell</td>
</tr>
<tr>
<td>( S(i,j,k) )</td>
<td>( 0 \leq i \leq P(n) ), ( P(n) \leq j \leq P(n) + 1 ), ( 0 \leq k \leq V )</td>
<td>At ( i )th step the content of the ( j )th cell on the tape is ( k ).</td>
</tr>
</tbody>
</table>

Therefore, each computation that halts is of \( p(n) \) states. A DTM induces a valuation of our Boolean variables, but not vice versa. Our goal is to build a formula \( \varphi \) s.t. \( \varphi \models \exists x \exists y \exists z \) is satisfiable, s.t. each conjunct from \( \varphi \) takes care of a specific property of a DTM program's execution. We have six groups of properties:

\begin{align*}
\text{G}_1 & \text{ At each time } M \text{ is exactly at one state } \forall 0 \leq i \leq P(n) \exists k \leq 2 \left[ Q(i,k) \land \bigwedge_{0 \leq l \leq P(n)} \bigwedge_{0 \leq k \leq V} Q(l,k) \right] \\
\text{G}_2 & \text{ At each time the head is scanning exactly one tape cell } \forall 0 \leq i \leq P(n) \forall 0 \leq j \leq P(n) + 1 \forall 0 \leq k \leq V \left[ H(i,j) \land \bigwedge_{0 \leq l \leq P(n) \land l \neq i} \neg H(l,j) \right] \\
\text{G}_3 & \text{ At each time a tape square contains exactly one symbol from } \Sigma \text{ } \forall 0 \leq i \leq P(n) \forall 0 \leq j \leq P(n) + 1 \left[ S(i,j,k) \right] \\
\text{G}_4 & \text{ At time } 0, \text{ the computation is in the initial configuration of its cheating stage for input } x \left[ Q(0,0) \land H(0,1) \land S(0,0,0) \land S(0,1,k_1) \land \cdots \land S(0,1,k_n) \land S(0,1,i_0) \land \cdots \land S(0,1,i_n) \right] \\
\text{where } x = k_1, \ldots, k_n
\end{align*}
By time $p(n)$ $N$ has entered state $2r$

For each time $i$, $0 \leq i \leq p(n)$, the configuration of $M$ at time $i+1$ follows by a single application of the transition function $\delta'$ from the configuration at time $i$.

$\psi$: $M \leftarrow S(i, s, e) \rightarrow VH(i, \delta) \rightarrow SC(i+1, \delta, e)$ which ensures that unscanned cells won't change their content, plus:

$$M \leftarrow H(c(i, \delta) \leftarrow Q(i, e) \rightarrow SC(i, \delta, e) \rightarrow H(i+1, \delta, e+\Delta) \leftarrow S(i+1, i, e) \leftarrow Q(i+1, i, e')$$

which ensures that NDTM works according to $\delta'$.

Obviously $\psi_x$ holds iff $M$ accepts $x$ within $p(n)$ steps, assuming $M$ halts at $2r$.

It is easy to see that $\text{length}(\psi_x) \approx |M| \log |M| \cdot |C|$

where $|M|$ is the number of variables and $|C|$ cardinality of each conjunct and that $|M| = \Theta((p(n))^2) = |C|$; consequently $\text{length}(\psi_x) = \Theta(p(n)/p(n))$ and program that fills this length is obviously polynomial. Thus SAT is NP com
A complete description at step \( t \) must specify:

- \( Q: i \) state of the machine \( q_0, \ldots, q_k \)
- \( H: 2 \) position of the head \( \{ - \cdots, i + 1 \} \leq \{ - P(n), P(n + 1) \} \)
- \( S: 3 \) content of the \( j \)th cell

Correct rules are according to the table and general rules for TMs:

\( D^i \rightarrow D^{i+1} \) via \( G_1 - G_6 \)
**3-SAT**

**INSTANCE**: A finite set of boolean variables and a conjunction of disjunctions \( C \) s.t. for each \( c \in C \), \( |c| = 3 \).

**Question**: Is \( M \in C \) satisfiable?

**Prop**: 3-SAT is NP-complete.

**Proof**: 3-SAT is obviously NP, to show completeness we show \( \text{SAT} \subseteq \text{3-SAT} \); assume \( u \)'s are literals over \( \{x_1, x_2, \ldots, x_n\} \) and \( \overline{u} \)'s are literals over \( \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\} \).

(i) \( |c| = 1 \) \( \rightarrow \) replace \( u \) by \( \{u_{1,2} | x_1, x_2 \in \{0,1\}\} \).

(ii) \( |c| = 2 \) \( \rightarrow \) replace \( u_1 \lor u_2 \) by \( \{u_{1,2} | x_1, x_2 \in \{0,1\}\} \).

(iii) \( |c| > 3 \) \( \rightarrow \) replace \( u_1 \lor u_2 \lor \ldots \lor u_k \) \( \rightarrow \) \( u_1 \lor \overline{u}_2 \lor \overline{u}_3 \lor \ldots \lor \overline{u}_{k-3} \lor u_{k-1} \lor u_k \).

This is obviously a polynomial transformation. The above method is called **local replacement**.

**3DM (3-Dimensional Matching)**

**INSTANCE**: A set \( M \subseteq W \times X \times Y \) where \( W, X, Y \) are disjoint having the same number \( I \) of elements.

**Question**: Does \( M \) contain a matching subset \( M' \) s.t. all coordinates of two pts from \( M' \) are pairwise different and \( |M'| = I \)?

**Prop**: 3DM is NP-complete.

**Proof**: We reduce any instance of SAT to an instance of 3DM by a p-time fcn.

Let \( C = \bigvee_{i=1}^m C_i \), each \( C_i \) a disjunction of literals \( a \) \( a \) or a negated \( a \) variable over \( \{u_1, \ldots, u_n\} \).
We proceed by the method **COMPONENT DESIGN**

We define $M$ as follows:

$$M = T \cup C \cup G$$

where: $T = \bigcup_{i=1}^{k} T_i$ and $T_i = T_i^+ \cup T_i^-.$

The $T$ component sets value of $U_i$'s by synchronizing the values of $U_i(t)$ where $U_i(t)$ is $U_i$ version for $c_j$.

$C_j$ is satisfaction tester for $c_j$’s, $C = UC_j.$

$G$ is the garbage collector.

(i) Let $T_i$ be the collection of the following triples:

$$\{U_i(1), a_i(2), b_i(1)\}, \{\overline{U_i}(1), a_i(1), b_i(2)\}, \ldots$$

For example, for $M = 4$, we have

Where

$$T_i^+ = \{ (\overline{U_i}(1), a_i(1), b_i(2)) \mid i \in \mathbb{N} \}$$

$$T_i^- = \{(U_i(1), a_i(2), b_i(1)) \mid i \in \mathbb{N} \}$$

$T_i = \{ \bigcup_{i=1}^{k} (U_i(1), a_i(1), b_i(2)) \}.$

\[ U \times \{ U_i(m), a_i(1), b_i(m) \} \]
(ii) \( C_1 = \{ (u_i(d), s_1(d), s_2(d)), (\overline{u}_i(d), s_1(d), s_2(d)) \} \) if \( u_i \) or \( \overline{u}_i \) appear in \( C_2 \).

(iii) \( C = \{ (u_i(d), s_1(k), s_2(k)), (\overline{u}_i(d), s_1(k), s_2(k)) \} \)
\[ 1 \leq K \leq m(n-1), \quad 1 \leq i \leq h, \quad 1 \leq j \leq m^2 \]

The above is obviously a poly-time. Now, \( m \leq h \) is true iff in each \( C \) we can find a true literal — this corresponds to \( u_i^\wedge(d), s_1(d), s_2(d) \)

Once one \( u_i^\wedge(d) \) bound by \( s_1(d), s_2(d) \), \( u_i^\wedge(d) \) cannot be (if we have \( H^1 \cap H \) a matching bound by \( a_i(d), b_i(d) \)) and so must be \( u_i^\wedge(d) \)

But then, by \( T \), all \( u_i^\wedge(d) \) are set equal — all bound by \( a_i, b_i \); 'Garbage' is bound all unused \( u_i(d) \)'s and \( m-1 \) of them since for any \( C \), we have chosen only one \( u_i^\wedge(d) \).

Note that \( M \) contains only \( 2m + m + 2m(n-1) \) entries which helps us see that \( F \) is poly computable.
**EXACT COVER BY 3-SETS (X3C)**

**INSTANCE:** A finite set $X$ with $|X| = 3q$ and a collection of 3-element subsets of $X$.

**QUESTION:** Does $C$ contain an exact cover for $X$ i.e. does $C$ contain a subset $C'$ which is a 3-partition of $X$.

We show $X3C$ is NP complete by **RESTRCITION** method.


Obviously $M$ has a match $M'$ iff $M^* = \{ \{ W_i, X_i, Y_i \} \mid (W_i, X_i, Y_i) \in M \}$ has a 3-partition of $Z$.

**VERTEX COVER (VC)**

**INSTANCE:** A graph $G = (V, E)$ and a positive integer $K \leq |V|$.

**QUESTION:** Is there a vertex cover of size $\leq K$ for $G$, i.e. a subset $V' \subseteq V$ s.t. $|V'| \leq K$ and for each edge $\{u, v\} \in E$ at least one of $u, v$ belongs to $V'$.

**CLIQUE**

**INSTANCE:** A graph $G = (V, E)$ and a positive integer $J \leq |V|$.

**QUESTION:** Does $G$ contain a clique of size $J$ or more i.e., a subset $V' \subseteq V$ s.t. $|V'| \geq J$ and for any $u, v \in V' \quad (u, v) \in E$ (i.e., $(V', E[V'])$ is complete graph).
**Definition:** Let $(V,E)$ be a graph, then $V' \subseteq V$ is an independent set of $G$ iff $V'$ is a clique in $(V,E^c)$.

**Lemma:** For any graph $G = (V,E)$ and $V' \subseteq V$

**TFAE:**
1. $V'$ is a vertex cover of $V$
2. $V - V'$ is an independent set of $(V,E)$

**Theorem:** VERTEX COVER is NP complete.

**Proof:** (Again component design.) VC is obviously NP complete. We reduce 3SAT to VC. Let $U = \{ u_1, \ldots, u_m \}$, $C = \{ c_1, \ldots, c_n \}$, Consider the graph

![Graph Diagram]

Corresponds to $c_i = \{ u_j, \overline{u}_j \} \cup \{ a_{i1}, a_{i2}, a_{i3}, a_{i2}', a_{i3}' \}$

$E = \{ \{ u_i, \overline{u}_i \} \mid u_i \in U \} \cup \{ a_{i1}, a_{i2}, a_{i3}, a_{i2}', a_{i3}' \}$

$E \cup \{ u_j, \overline{u}_j, a_{1k}, a_{2k}, a_{3k}, a_{1k}', a_{2k}', a_{3k}' \}, K = 2m + n$.

Again if $P$-computable and $(V,E)$ has a $K$-covering iff $C$ is satisfiable.

any $V'$ sat. $|V'| \leq K$ must, in order to be a vertex covering, be $|V'| = K$ and in each triangle two vertices picked and one of each $u_i$ or $\overline{u}_i$. 
But then for each $i$, since the appropriate triangle has a vertex uncovered at least the opposite literal must be chosen, so $c_i$ is true. Opposite if $C$ is true it is easy to build a vertex covering.

As we saw $HC \propto TS$, so by proving NP-completeness of $HC$ we also get the NP-completeness of $TS$.

**Theorem:** $HC$ is NP-complete.

**Proof:** Enough to see $VC \propto HC$. For any $(u,v) \in E$ define the following graph $G$:

Any Hamiltonian path containing $G$ that is connected through some of the 4 end points $u_0$ must go in one of the following ways:

We build our graph in the following way:

where $c_1, \ldots, c_k$ are new vertices and $K$ is from $VC$ problem ($\leq K$ points to be chosen).
Obviously any HC gives a K-covering of the graph. For (1) and (2) only one vertex is taken, for (3) both.

Theorem: Directed HC is also NP-complete.
Proof: (By restriction) Consider only graphs of the form \((u,v) \in E \Rightarrow (v,w) \in E\) to get HC.

**PARTITION**

**INSTANCE:** A finite set \(A\) and a "size" \(s(a) \in \mathbb{N}\) for each \(a \in A\).

**QUESTION:** Is there a subset \(A' \subseteq A\) s.t.
\[
\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a) ?
\]

Theorem: PARTITION is NP-complete.
Proof (component design) We transform 3DM to partition.

Let \(W = \{w_1, \ldots, w_k\}\)
\(X = \{x_1, \ldots, x_k\}\)
\(Y = \{y_1, \ldots, y_k\}\)
\(M = \{m_1, \ldots, m_k\}\)
Thus \(|M| = k\).

Let \(p = \lceil \log_2 (k+1) \rceil\)

Consider \(k\) "registers" as above: for each \(m_i \in M\) put 1 in the rightmost places of \(w_{s_i}, x_{s_i}, y_{s_i}, z_{s_i} \in M\), 0's in the rest to get all
Note: If we sum all the binary numbers corresponding to members from M, there is no "carries" from one zone to another, since we have enough room for "binary" in \( \log_2(k+1) \) places. Thus \( M' \) is a matching iff the sum of all numbers from \( M' \) is of the form

\[
\begin{array}{c}
\begin{array}{c}
0001 \\
0001 \\
0001 \\
\vdots \\
0001
\end{array}
\end{array}
\Rightarrow
\begin{array}{c}
\begin{array}{c}
0001 \\
0001 \\
0001 \\
\vdots \\
0001
\end{array}
\end{array}
\]

Now consider two extra members, \( b_1 \) and \( b_2 \) s.t.

\[
\begin{align*}
\sum_{i=0}^{k} s(a_i) = 2S - B &= 2S - B \\
\sum_{i=0}^{k} s(b_i) &= 2S + B
\end{align*}
\]

which can be specified in \( 3p_1^2 + 1 \) bits each, and consequently our transition \( (W, X, Y, M) \rightarrow a_1, \ldots, a_k, b_1, b_2 \) is polynomially computable.

Claim: \( M' \) corresponding to \( M' \) is a matching.

Proof: In any partition \( b_1 \) and \( b_2 \) must be in different pieces obviously, since \( b_1 + b_2 = 3 \sum_{i=0}^{k} s(a_i) \).

But

\[
\begin{align*}
\sum_{i=\in A} s(a_i) + 2S - B &= \sum_{i=\in A - A'} s(a_i) + S + B \\
\sum_{i=\in A} s(a_i) + 2S - B &= S - \sum_{i=\in A'} s(a_i) + S + B \\
2 \sum_{i=\in A} s(a_i) &= 2B \\
\sum_{i=\in A} s(a_i) &= B \Rightarrow M' \text{ is a matching.}
\end{align*}
\]
**SHORTEST PATH**

**INST.** Graph $G=(V,E)$, length $l(e)\in \mathbb{W}$ for each $e\in E$, specified vertices $a,b \in V$, a positive integer $B$.

**QUEST.** Is there a simple path from $a$ to $b$ in $G$ having total length $\leq B$?

**P-time computable**

---

**LONGEST PATH**

**INST.** Same

**QUESTION.** Is there a simple path from $a$ to $b$ in $G$ having total length $\geq B$?

---

**EDGE COVER**

**INST.** Graph $G=(V,E)$, positive integer $K$.

**QUEST.** Is there an $E' \subseteq E$ with $|E'| \leq K$ s.t., for each $v \in V$ at least one edge starting at $v$ belongs to $E'$?

---

**VERTEX COVER**

**INST.** Same

**QUEST.** Is there a $V' \subseteq V$ s.t., $|V'| \leq K$ and for all $e \in E$ at least one vertex of $e$ belongs to $V'$?

---

NP-complete
**Example:** Consider PARTITION PROBLEM, with the instance \( A = \{ a_1, \ldots, a_3 \} \) \( S = \{ s(a_1), \ldots, s(a_3) \} \). Let \( B = \sum S(a_i) \). If \( B \) is odd - answer is NO; if even we can construct a table \( n \times B \)

s.t. at the place \( (i, j) \) \( i \in n \), \( j \in B/2 \) we have \( T \) iff among \( \{ a_1, \ldots, a_3 \} \) it is possible to choose some with the sum \( \geq \) \( t \) \( \text{deciding (i+1, j)} \). This is possible to do recursively, consulting the

value of \( (i, j - S(a_{i+1}) \) and takes \( O(nB) \) steps. Since the input length is \( O(n \log B) \)

this certainly doesn't give a P-time algorithm (we have proved the NP-completeness of PARTITION)

for PARTITION, but whenever the input numbers are bounded by a polynomial of the

length of the input \( I \) we have \( B \leq n \cdot p(\text{length}(I)) \) and so \( T \leq p \cdot p(\text{length}(I)) = p^2(\text{length}(I)) \),

i.e. this restriction of the PART \( B \) is P-solvable.

For many others this is false, e.g. for

problems with no numerical input (except for

the names which are always polynomially (linearly)

bounded.

Besides \( \text{length}(I) : D_n \to \mathbb{N} \) we define now

another encoding independent function \( \text{in the sense that different encodings produce polynomially}

related functions} \) \( \text{Max}(I) \) with intended meaning to be the magnitude of the largest number of \( I \).
Def. Two length functions \( \text{Length}(I), \text{Length}'(I) \) are \underline{polynomially related} if there are polynomials \( p \) and \( p' \), st. for all instances \( I \in D_n \)

\[
\text{Length}[I] \leq p'(\text{Length}'[I])
\]

\[
\text{Length}'[I] \leq p(\text{Length}[I])
\]

(Note: this is always the case, whenever lengths are polynomially related to the input length \( |I| \).

Def. The pair of functions \( (\text{Length}, \text{Max}) \) is \underline{polynomially related} to the pair of functions \( (\text{Length}', \text{Max}') \) iff \( \text{Length}, \text{Length}' \) are polynomially related as above and there are two variable polynomials \( q, q' \), st. for all \( I \in D_n \)

\[
\text{Max}[I] \leq q'(\text{Max}'[I], \text{Length}'[I])
\]

\[
\text{Max}'[I] \leq q(\text{Max}[I], \text{Length}[I])
\]

All our results are invariant for polynomially related pairs. We also require that the binary notation of \( \text{Max}[I] \) and \( \text{Length}[I] \) are computable by DTM's in polynomial time.

In our present case this is important; the above inequalities are not sufficient any more only for the simple reason that we will be considering restrictions on instances defined in terms of \( \text{Length}[I] \) and \( \text{Max}[I] \) and we need to be able to check whether or not a given string encodes an instance
Def: An algorithm that solves a problem \( \Pi \) is called a pseudo-polynomial time algorithm for \( \Pi \) if its time complexity function is bounded by a polynomial function of two variables \( \text{Length}(I), \text{Max}(I) \).

Def: A problem \( \Pi \) is a number problem if there exists no polynomial \( p \) such that \( \text{Max}[I] \leq p(\text{Length}[I]) \) for all \( I \in \Pi \).

Obviously, unless \( P=NP \), if \( \Pi \) is NP-complete and \( \Pi \) is not a number problem, then \( \Pi \) cannot be solved by a pseudo-polynomial time algorithm.

Def: For any decision problem \( \Pi \) and any polynomial \( p \) over \( \omega \), let \( \Pi_p \) denote the subproblem of \( \Pi \) obtained by restricting \( \Pi \) to only those instances \( I \) that satisfy \( \text{Max}[I] \leq p(\text{Length}(I)) \).

Note: \( \Pi_p \) is never a number problem and if \( \Pi \) is solvable by a pseudo-polynomial algorithm, then \( \Pi_p \) is polynomially solvable! Namely, for any \( x \) we can compute \( \text{Max}[I], \text{Length}[I] \) and the value \( p(\text{Length}(I)) \) to see if \( x \) is an instance of \( \Pi_p \). If so apply our pseudo-
polynomial algorithm which is polynomial for \( P \) instances.

**Def**: A decision problem \( \Pi \) is \( \text{NP-complete in the strong sense} \) if \( \Pi \) belongs to \( \text{NP} \) and there exists a polynomial \( p \) over \( \omega \) for which \( \Pi_p \) is \( \text{NP-complete} \).

Obviously any problem \( \Pi \) which is not number problem is \text{NP-complete in the strong sense}.

**Corollary**: If \( \Pi \) is \text{NP complete in the strong sense}, then \( \Pi \) cannot be solved by a pseudo-polynomial time algorithm unless \( \text{P=NP} \).

Thus \text{PARTITION is not NP-complete in the strong sense}.

**Proposition**: TRAVELING SALESMAN is \text{NP-complete in the strong sense}.

**Proof**: Consider \( T_S \) with distance limit \( 2 \) and \( R = m \) (m-number of cities). To this \( \Pi_p \text{HC} \) is transformable and consequently \( T_S \) is \text{NP-complete in the strong sense}.
**NP - HARDNESS**

**Def.:** A search problem $\Pi$ consists of a set $D_\Pi$ of finite objects - instances and for each $I \in D_\Pi$ a set $S_\Pi (I)$ of finite objects - solutions for $I$. An algorithm is said to solve a search problem $\Pi$ if, given as input any instance $I \in D_\Pi$ it returns the answer "No" whenever $S_\Pi (I)$ is empty and otherwise returns some solution $s \in S_\Pi (I)$. (Search for a solution)

**Note:** A decision problem can be seen as a search problem with $S_\Pi (I) = \emptyset$ if $I \notin Y_\Pi$ and $S_\Pi (I) = \{ \text{yes} \}$ if $I \in Y_\Pi$.

Formal counterpart of a search problem:

**Def.:** For a finite alphabet $\Sigma$ a string relation $R \subseteq \Sigma^+ \times \Sigma^+$ over $\Sigma$ is a binary relation $(\Sigma^+ = \Sigma^* \setminus \{ \varepsilon \}, \varepsilon$ empty string).

We can identify a language $L$ over $\Sigma$ with the string relation $R = \{(x, s) | x \in \Sigma^+ \cap L \}$ where $s$ is any fixed symbol from $\Sigma$. ("yes" yes)

This ignores whether or not $x \in L$ but this is for our purposes unimportant.
**Def:** A function \( f : \Sigma^* \to \Sigma^* \) realizes the shyness relation \( R \) iff for each \( x \in \Sigma^+ \), \( f(x) = e \) whenever there is no \( y \in \Sigma^+ \) s.t. \((x,y) \in R\) and \( f(x) \) equals some \( y \) s.t. \((x,y) \in R\) whenever such a \( y \) exists.

**Def:** A DTM program \( M \) solves the shyness relation \( R \) iff it computed by \( M \) realizes \( R \).

Again we fix an encoding scheme \( e \), this time encoding not only instances \( I \) of \( \Pi \) but also candidates for solution \( s \in S_{\Pi} (I) \). Again we define

\[
R(\Pi, e) = \{ (x, y) : x \in \Sigma^+ \text{ is the encoding under } e \text{ of an instance } I \in D_{\Pi} \text{ and } y \text{ is the encoding under } e \text{ of a solution } s \in S_{\Pi} (I) \}.
\]

Again, \( \Pi \) (under \( e \)) is solvable by a polynomial time algorithm if there is a polynomial time DTM program that "solves" \( R(\Pi, e) \).
Def: An oracle Turing machine (OTM) consists of two tapes and two heads.

![Diagram of OTM with oracle tape and working tape]

A program for an OTM specifies:

1. A finite set $\Gamma$ of tape symbols, including $\Sigma \subseteq \Gamma$ of input symbols and a distinguished blank symbol $b \in \Gamma - \Sigma$.

2. A finite set $Q$ of states including a distinguished start state $q_s$, a halt state $q_h$, an oracle consultation state $q_c$ and a resume-computation state $q_r$.

3. A transition function $\delta$:

$$\delta : (Q - \{q_h, q_c\}) \times \Gamma \times \Sigma \rightarrow Q \times \Gamma \times \Sigma \times \{0, 1, 1\}^2$$

(Note: on the oracle tape only symbols from $\Sigma$ are allowed.)

The computation of an OTM program on an input $x \in \Sigma^*$ begins with the symbols of $x$ written in the squares 1 to $|x|$ of the primary tape, with the rest of that tape and all of the oracle tape being blank. Each tape scanning square 1 of its
tape, and the finite state control is state $q_0$. Computation proceeds with the following three possibilities occurring at each step:

(a) If the current state is $q_0$, then the computation has ended and no further steps take place.

(b) If the current state is $q \in \mathcal{Q} - \{q_0, q_3\}$, then the action taken depends on the symbols $s_1, s_2$ being scanned by the heads and $\delta$: if

$$\delta(q, s_1, s_2) = (q', s_1', s_2', \Delta_1, \Delta_2)$$

then the finite state control changes to state $q'$, $s_1, s_2$ are replaced by $s_1', s_2'$ and heads are moved left or right for one cell depending if $\Delta_1 (\Delta_2)$ is -1 or +1.

(c) If the current state is $q_3$ then, for $y \in \Sigma^*$ being the string appearing in squares 1 through $|y| + 1$ with $|y| + 1$ being the first blank cell on the positive part of the oracle tape, with the oracle head scanning the box 1 through $|y|$, let $z \in \Sigma^*$ be $g(y)$. Then, in one step the oracle tape is changed to contain the string $z$ in squares 1 through $|z|$, with blanks everywhere else, the oracle head is set to scan square 1, and the finite state control is
changed from state \( I_c \) to state \( I_x \), leaving the content of the working tape and its head unchanged.

We denote by \( M_g \) the "relativized" OTM program obtained by combining \( M \) with oracle \( f \).

**Def.** If \( M_g \) halts for all inputs \( x \in \Sigma^* \), then it can be viewed as computing a function \( f_{M}^{g}: \Sigma^* \to \Sigma^* \).

**Def.** \( M_g \) is a polynomial time OTM program if there exists a polynomial \( p \) s.t. \( M_g \) halts within \( p(|x|) \) steps for every input \( x \in \Sigma^* \).

(Turing reducible)

**Def.** Let \( R \) and \( R' \) be any two binary relations over \( \Sigma \). A polynomial time reduction from \( R \) to \( R' \) is an OTM program \( M \) with input alphabet \( \Sigma \) s.t. for every function \( g: \Sigma^* \to \Sigma^* \) that realizes \( R' \), the relativized program \( M_g \) is a polynomial time OTM, and the function \( f_{M}^{g} \) computed by \( M_g \) realizes \( R \).

If there is such a reduction from \( R \) to \( R' \), we write \( R \leq_{R} R' \) (\( R \) Turing reduces to \( R' \)).
Before, \( \alpha \vdash \beta \) is transitive.

**Def:** A string relation \( R \) is **NP-hard** if there is some **NP-complete** language \( L \) (itself stated as a string relation) such that \( L \not\subset R \).

A search problem \( \Pi \) (under \( e \)) is **NP-hard** if the string relation \( R(\Pi, e) \) is **NP-hard**.

**Note:** By the transitivity of \( \alpha \vdash \beta \) if \( R \) is NP hard and it reduces to \( R' \) then \( R' \) must be also NP hard.

**Lemma:** If a string relation \( R \) is NP-hard then it cannot be solved in polynomial time unless \( P = NP \).

**Proof:** By our requirement about the position of the oracle head, if \( M_K \) solves an NP-complete problem \( \Pi \) in \( p(1|x|) \) steps, then \( \Pi \) is restricted to the values of the argument \( \leq p(1|x|) \) and so if \( M_K \) were polynomially computable in \( g(1|x|) \) steps, then \( \Pi \) would be P-computable within \( p(1|x|) + g(1|x|) \) steps.

**Note:** If \( \Pi \) \( \not\subset \Pi_e \) is NP-complete, then \( \Pi_e \) is NP-hard, with \( M_{Kn} \) just inverting yes–no answers.
Example: TRAVELING SALESMAN OPTIMIZATION

PROBLEM is NP-hard.
All we need to solve TS instance is
to compute the length of the optimal
route and compare it with bound of TS.
Here we used the TSO subroutine only once.

\( K^{\text{fs}} \) - LARGEST SUBSET

INSTANCE: A finite set \( A \), \( |A| \leq n \)
for each \( a \in A \) and two nonnegative integers \( B \leq \sum_{a \in A} s(a) \) and \( K \leq 2^{|A|} \).

QUESTION: Are there at least \( K \) distinct subsets \( A' \subseteq A \) s.t. \( \sum_{a \in A'} s(a) \leq B \) ( \( \sum_{a \in A'} s(a) = \sum_{a \in A} s(a) \) )?

(i.e. we order \( S(A) \) in a nondecreasing sequence
with respect to \( S(A') \), and look if \( K \)th subset
\( 3 \leq B \)).

1° \( K \)-LDS can be solved in pseudo-polynomial
time \( \leq P (|A| \cdot K \cdot \log \log (S(A)) \cdot \log S(A)) \).
Thus, for each fixed \( K \), it is P-solvable.

2° This problem not only appears not to be
in \( P \) but also not to be in \( NP \),
since writing down a guess of \( K \) subsets
doesn't seem possible in \( P (|A| \cdot \log \log S(A) +
\log \log (K+1)) \) steps.
3. No transformation of a known NP-complete problem to $K^hLS$ is known.

4. Nevertheless $K^hLS$ is NP-hard:

Proposition: PARTITION PROBLEM can be Turing reduced to $K^hLS$ PROBLEM.

Proof: Consider the following DTM program with oracle:

1. Firstly compute (in polynomial time)

$$\sum_{a \in A} s(a) ; \text{ if odd - answer no; if even set } b = \frac{\sum_{a \in A} s(a)}{2}.$$

2. By computing each digit separately:

- find the number of subsets $\binom{n}{k}$ such that

$$\sum_{a \in X} s(a) \leq b$$

- ask

- is the number of such sets more than $2^{n-1}$
- if yes, ask is number of sets more than $2^{n-1} + 2^{n-2}$ if not ask
- if it is more than $2^{n-2}$ etc.

3. After determining all the digits of $\binom{n}{k}$

- ask: is the number of sets having

$$\# \text{units} \leq 2^{n-1}$$

$$\# \text{?} \geq \# ?$$

- if yes, answer yes; if no, answer no.

Used the fact: one can determine a number $X$ with at most $2^{1 \times 1}$ questions.
Thus K^th LS is NP-hard, and so if we could solve it in P-time we would have P = NP, but vice versa need not be true unlike for NP-complete problems. But if A ∩ B and B ∩ A and A is NP-complete then B is solvable in P-time iff P = NP. Namely, if P = NP then A is P-solvable and so is B, since B ∩ A.

TRAVELING SALESMAN EXTENSION (TSE).

INSTANCE: A finite set C = {c_1, ..., c_m} of cities, \{d(c_i, c_j) | i ≠ j\} and a partial tour Θ = \langle c_{x(1)}, ..., c_{x(K)} \rangle of K distinct cities from C, 1 ≤ K ≤ m.

QUESTION: Can Θ be extended to a full tour having total length ≤ B.

Trivially TSE is NP and so TSE ∩ TS

TRAVELING SALESMAN OPTIMIZATION (TSO)

INSTANCE: C, \{d(c_i, c_j) | i ≠ j, c_i, c_j ∈ C\}

QUESTION: What is the optimal tour?

Proposition: TSO ∩ TSE

Let B^* be the length of the optimal tour. Since m ≤ B^* ≤ m, \max \{d(c_i, c_j) | i ≠ j\} we can otherwise B^* using TSE for Θ = \langle c_1 \rangle
with $O\left(\log_2 m \cdot \max_i Ed(c_i, c_j) \mid i \neq j \right) = O\left(\text{P}(x, y)\right)$, 
questions for TSE, and so in polynomial time.

Now, clearly $\langle c_1 \rangle$ is an extendible tour, 
and after at most $m-2$ questions of TSE for $\langle c_1, c_k \rangle$, 
we find the right $c_{k+1}$. Continuing like this we get the optimal tour with 
polynomially many questions for TSE and 
polynomially much of other work. Thus 
TSE $\leq TSE \leq TS \leq TSO$ and the 
previous remarks apply: TSO is solvable 
in P-TIME iff P=NP. (FORMULA: BINARY SEARCH 
FOR B* + STEPS-PRISE BUILDING OF THE SOLUTION).

DEF: A search problem $\Pi$ is \underline{NP - easy} iff 
there is an \underline{NP - problem} A s.t. $\Pi \leq A$. 
Thus $\Pi$ is not harder than any NP 
complete problem: knowing how to solve A 
enables us to solve $\Pi$ with P-time 
extra work.

It is easy to see that all the 
search problems corresponding to SAT (find a 
satisf. evaluation) VC, HC etc. are \underline{NP - easy} 
by the previous procedure.

Thus, the restriction of the basic theory to 
decision problems only lies caused 
in substantial loss of generality, 
since most often the search problems
whose decision counterparts are proven to be NP-complete (and so search problems NP-hard) are also NP-easy and thus of equivalent complexity.

Thus we have a large class of NP-equivalent (both NP-hard and NP-easy) search problems with NP-complete decision counterparts.
let B be a recursive language s.t. B \notin P. Then there exists a P-time recognizable language D \in P s.t. D \notin P. D \ominus B \neq B \ominus D \ominus B

Corollary: If P \neq NP then there is an NP-language which is neither in P nor it is complete.

Proof: Take B = any NP-complete problem, D as above and get D \ominus B. Since \epsilon \in \text{DEF} and B \ominus \text{NP} \rightarrow D \ominus B is NP but (by \text{BND} implies bad) \ominus B is not NP-complete. Since D \ominus B \notin P we are done.

Applying the above theorem to B \ominus D from the corollary we get still simple language yet not in P.

Def: A problem \Pi is in \text{co-NP} if \Pi^c is in \text{NP} i.e. 
\text{co-NP} = \{ \Sigma^* - L | L is a language over the alphabet \Sigma \text{ and } L \in \text{NP} \}.
Theorem: If there is an NP-complete problem \( \Pi \) such that \( \Pi^c \) is also NP, then NP = co-NP.

Proof: Trivial using completeness.

A candidate for an NP (incomplete) problem:

**COMPOSITE NUMBERS**:

**Instance**: A positive integer \( K \)

**Question**: Are there integers \( M, N > 1 \) such that \( K = M \times N \)?

The CN problem is obviously NP, one can show that \( CN^c \) is also NP by giving a short guess proof that a number is prime. Thus, if \( NP \neq \text{co-NP} \), then CN cannot be complete. Miller (1976 J. Comp Syst Sci 13,317) gives an algorithm for \( CN^c \) (prime number test) that runs in poly-time if the "Extended Riemann Hypothesis" of number theory is true.

Hypotheses:

1) \( P \neq NP \)
2) \( NP \neq \text{co-NP} \)

We also don't know even with 1) and 2):

i) \( P = NP \cap \text{co-NP} \),

\( P \neq \text{co-NP} \).
**Def:** We say that the relation $R_M$ is computed by an NDTM program $M$ iff

$$R_M = \{ <x,y> | \text{there is a string } z \text{ st. an input } x \text{ and guess } z \text{ M has output } y \}$$

**Def:** We say that a language $L_1$ over $\Sigma_1$ is $\delta$-reducible to a language $L_2$ over $\Sigma_2$ ($L_1 \leq \delta L_2$) iff there is a polynomial time NDTM program $M$ st.

for all $x \in \Sigma_1^*$ there is $y \in \Sigma_2^*$ for which $<x,y> \in R_M$ (i.e. $M$ halts for all $x$ for some guess $z$) and for all $xy$ $<x,y> \in R_M \iff x \in L_1 \iff y \in L_2$.

**Note:** $L_1 \leq \delta L_2 \rightarrow L_1 \leq \delta L_2$ but we don't know if the opposite holds.

**Ex:** LINEAR DIVISIBILITY

**INST:** $a, c > 0$

**GUESS:** $\exists x : ax + 1 | c$?

$LD$ is $\delta$-complete

**Theorem:** If $L$ is $\delta$-complete and $L \in \text{NP} \cap \text{co-NP}$ then $NP = \text{co-NP}$.

**Def:** Two languages $L_1, L_2$ are polynomial time isomorphic iff there is a 1-1 function...
CONJECTURE: All NP-complete languages are P-time isomorphic.

\[ P \neq \text{NP} \] since \( P \) contains both finite and infinite languages so not all of them can be isomorphic (\( P = \text{NP} \rightarrow \) all \( P \)-languages are \( \text{NP} \)-complete and thus isomorphic).

**Def.:** A \( \text{NOM} \) (non-deterministic oracle Turing machine) is a \( \text{NDTM} \) augmented with an oracle tape, with the same conventions about the running times.

**Def.:** Let \( Y \) be a class of languages. The classes \( P^Y \) and \( \text{NP}^Y \) are defined as follows.

\[ P^Y = \{ L \mid \text{there is a language } L' \in Y \text{ s.t. } L \leq_T^P L' \} \]

\[ \text{NP}^Y = \{ L \mid \text{there is a language } L' \in Y \text{ s.t. there is a polynomial time non-deterministic reduction from } L \text{ to } L' \} \]

**Note:** \( P^{\text{NP}} = \text{all } \text{NP} \)-easy languages.
POLYNOMIAL HIERARCHY

\[ \Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathcal{P} \]

\[ \Delta_{k+1}^p = \mathcal{P} \Sigma_k^p \]

\[ \Sigma_{k+1}^p = \mathcal{N} \Sigma_k^p \]

\[ \Pi_{k+1}^p = \text{co} - \Sigma_{k+1}^p \]

**Theorem:** [W. R. J. Mitchell; complete sets and 𝓔, TIME HIER, Theor, 33 (233)]

Let \( L \subseteq \Gamma^* \) be a language, \( |\Gamma| \geq 2 \). For any \( k \geq 1 \), \( L \subseteq \Sigma_k^p \) iff there are polynomials \( p_1, p_2, \ldots, p_k \) and a polynomial-time recognizable relation \( R \) of dimension \( k+1 \) over \( \Gamma^* \) (i.e., a relation for which there is a \( P \)-time DTM that recognizes the language of exactly those \( k+1 \) tuples that are in \( R \)) s.t. for all \( x \in \Gamma^* \)

\[ x \in L \iff (\exists y_1 \in \Gamma^* \mid |y_1| \leq p_1(1|x|)) \]

\[ (\forall y_2 \in \Gamma^* \mid |y_2| \leq p_2(1|x|)) \]

\[ (\exists y_k \in \Gamma^* \mid |y_k| \leq p_k(1|x|)) \]

\[ R(x, y_1, \ldots, y_k) \]

**Def:** A language \( L \) is \( \Sigma_k^p \) complete if for every \( L' \subseteq \Sigma_k^p \), \( L' \alpha L \).
Proposition: \( \Pi \in PH \Rightarrow PH \) can be solved by exhaustive search in deterministic time \( O(2^{\text{length}(I)}) \).

Proof: Corollary of [Wadhwani].

**Definition:** An enumeration problem is based on the search problem \( \Pi \) as: "Given \( I \), what is the cardinality of \( S_n(I) \), i.e., how many solutions are there?

**Note:** We only ask "how many" and don't ask to display all of them; their number, even if exponential on length \( (I) \), might be still possible to write in polynomially many digits.

**Def:** An enumeration problem belongs to \( \#P \) if there is a nondeterministic algorithm that for each \( I \in D \Pi \) the number of guesses that lead to acceptance of \( I \) is exactly \( |S_n(I)| \) (\( S_n(I) \) = the numbers of solutions of the enumeration problem \( \Pi \)) and the length of the longest accepting computation is bounded by a polynomial in length \( (I) \).

**Example:** \( HC \), \( SAT \), etc.

**Def:** An enumeration problem \( \Pi \) will be called \( \#P- \)
complete iff. $\Pi \in \#P$ and for all $\Pi' \in \#P$
$\Pi' \preceq \Pi$. ($\Pi$ is a number output problem so $\preceq$ instead of $\preceq$).

**Def.** Given two search problems $\Pi$ and $\Pi'$, a polynomial time **parsimonious transformation** from $\Pi$ to $\Pi'$ is a function $f : D_\Pi \rightarrow D_{\Pi'}$ that can be computed in polynomial time and that satisfies for all $I \in D_\Pi$,

$$|S_\Pi(I)| = |S_{\Pi'}(f(I))|.$$

**Note.** Obviously a parsimonious transformation from $\Pi$ to $\Pi'$ automatically gives rise to a Turing reduction between the associated enumeration problems.

The generic transformation in the proof of Cook's theorem can be made parsimonious and so we get:

**Theorem (Simon 1977)** The problem of counting the number of satisfying truth assignments for an instance of SAT is $\#P$-complete.

**Note.** Some search problems that can be solved in $P$-time while the corresponding enumeration problems cannot be solved in $P$-time unless $P = NP$.
DEF: The space used in the computation is a Turing machine with input $x$ is the number of distinct tape squares visited by the read-write head.

Note: We do not know if there are problems solvable in P-space but not in P-time.

Note: All problems in PH can be solved in P-space (obviously by induction)

DEF: PSPACE is the class of all languages recognizable by polynomial space bounded DTM programs that halt on all inputs (note: we again restrict ourselves to decision problems only).

We analogously define PSPACE-complete (polynomial transformability) problems.

Lemma: If $L \in \text{PSPACE-complete}$, then $L \in \text{P} \iff P = \text{PSPACE}$, also $L \in \Sigma^P_k \iff \text{PH} \text{ collapses at } k$, and $\Sigma^P_{k+m} = \text{PSPACE}$, $m \geq 0$.

Proof: Obvious.
**Def**: \( B_k \)

**INSTANCE**: A set of variables \( X = \{ x[i,j] : 1 \leq i \leq k, 1 \leq j \leq m_i \} \) and a well-formed Boolean expression \( E \) over \( X \).

**QUESTION**: Is the following expression true:

\[
\exists x[1,1] \exists x[1,2] \ldots \exists x[1,m_1] \exists x[2,1] \exists x[2,2] \ldots \exists x[2,m_2] \exists x[k,1] \exists x[k,2] \ldots \exists x[k,m_k] E.
\]

**Proposition**: \( B_k \) is a \( \Sigma^p_k \) complete problem.

(Woodall 76).

A \text{ PSPACE-complete problem} is \( \cup B_k \).

**Def**: \text{QUANTIFIED BOOLEAN FORMULAS (QBF)}

**INSTANCE**: A well-formed quantified Boolean formula \( F = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n E \) where \( E \) is a Boolean expression involving the variables \( x_1, \ldots, x_n \) and each \( Q_i \) \( F \) or \( T \).

**QUESTION**: Is \( F \) true?

Obviously \( QBF \) is in \text{PSPACE} since we can check \( F \) by evaluating \( F \) under all assignments using for each the same space, keeping trace where we use the inductive argument on \(
We say that a DTM program $M$ is linear bounded if no computation with input $x$ uses more than $1|x|+1$ space.

**Def.** **LINEAR SPACE ACCEPTANCE**

**INSTANCE:** A linear bounded DTM program $M$ and a finite string $x$, over its input alphabet.

**QUESTION:** Does $M$ accepts $x$?

**Prop.** LSA $\equiv$ PSPACE-complete.

**Corollary.** Linear space is contained in $P$ if PSPACE $\subseteq P$.

We change (inevitably) our definition of NDTM allowing that $n+1$ guess can be asked at any time instead of requiring that all the guesses should be done in the beginning of the computation. But all the guesses are written on the same square and if the program decides to keep the $n^{th}$ guess when $n+1^{st}$ is also needed it uses space to write it down and preserve it in this way. It is obvious that in this way the TIME complexity remains unchanged.

**Def.** NSPACE languages are those recognized by NDTM's (with above definition of a NDTM) with
polynomial bound on space used.

Theorem: [Savitch] If $L$ can be recognized by an NDTM program in space bounded by $T(n)$, $T(n) = \log^2(n)$ for all $n \geq 1$, then it can be recognized by a DTM program with space bound $T^2(n)$.

Corollary: $\text{PSPACE} = \text{NPSPACE}$
LOGARITHMIC SPACE

We again change our Turing machine model by allowing 3 tapes for a basic DTM:

- Input tape with read-only head
- Output tape with write-only head
- Working tape with read-write head

The ends of the input and the output tapes are recognized by having a blank box left in the beginning. It is easy to see that PH and PSPACE are unchanged; only now Space can be counted in the working tape only and thus, have \( S(n) < n \).

**Definition:** DLOGSPACE is the class of all languages recognizable by DTM programs that obey a space bound of \( \log_2 n + 1 \).

One can show that the above class is the same as the class of all languages recognizable in space \( c \cdot \log_2 n + 1 \), any \( c > 0 \).

**Proposition:** DLOGSPACE \( \leq P \) (open question: DLOGSPACE = P)

**Proof:** We can have \( \leq P(1x1) \) instantaneous descriptions ( \( K^{\log_2 n} \cdot |1| \)) and so if the machine hasn't halted within \( p(1x1) \) steps there are two identical instantaneous descriptions and so the machine is in a loop.
Def: Let $L_1, L_2$ be languages over alphabets \( \Sigma_1 \) and \( \Sigma_2 \) respectively. We say that a function \( f: \Sigma_1^* \to \Sigma_2^* \) is a \textit{log-space transformation} from \( L_1 \) to \( L_2 \) iff

(i) \( f \) can be computed by a DLOGSPACE bounded DTM program,

(ii) \( x \in L_1 \iff f(x) \in L_2 \)

We denote the existence of such an \( f \) by \( L_1 \preceq_{\text{log}} L_2 \).

Note: By the previous proposition \( L_1 \preceq_{\text{log}} L_2 \Rightarrow \quad L_1 \preceq L_2 \)

Prop: \( L_1 \preceq_{\text{log}} L_2 \), \( L_2 \preceq_{\text{log}} L_3 \), then

1) \( L_1 \preceq_{\text{log}} L_3 \)

2) \( L_2 \in \text{DLOGSPACE} \Rightarrow L_1 \in \text{DLOGSPACE} \)

Proof: Nontrivial (SN-M)

Def: A language \( L \in \text{EP} \) is \textit{log-space complete for} \( \text{P} \) if for all \( L' \in \text{EP} \),

\( L \preceq_{\text{log}} L' \).

By the above proposition, a log-space complete language is in DLOGSPACE iff \( \text{DLOGSPACE} = \text{P} \).
Def: NLOGSPACE is the set of all languages recognizable using space bounded by \( \log_2 n + 17 \) by NDTMs (space for input, output and the guesses are not counted).

We don’t know if NLOGSPACE = LOGSPACE

Def: DLOG^k-SPACE is defined as DLOG-space but with a bound of \( (\log_2 n + 17)^k \).

POLYLOGSPACE = \bigcup_{k \in \omega} DLOG^k-SPACE

As we saw DLOGSPACE \subseteq P but also (coded)

NLOGSPACE \subseteq P but \textit{POLYLOGSPACE} \neq P \neq NP

Why is so difficult to resolve P=NP problem? All known techniques for proving \( A = B \) as well as those for proving \( A \neq B \) relative to oracles and consequently yield \( A^c = B^c \) and \( A^c \neq B^c \) results for any oracle \( C \). On the other hand, it is possible to construct oracles \( C, D \) s.t.

\( p^c = NP^c \) and \( p^D \neq NP^D \). (Similar technique also shows that NP = coNP etc are also difficult in the same sense).

Note: we even don’t know if \( \text{EXPTIME} \leq \text{NP} \).