The class NP:

A language is in NP if and only if it is decided by some non-deterministic polynomial time Turing machine.

A verifier for a language \( L \) is an algorithm \( V \), where
\[
L = \{ \ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \ (\text{certificate}) \}.
\]

A polynomial time verifier, is a verifier that runs in polynomial time in the length of \( w \).

NP - class of languages that have polynomial time verifiers. ("Easy to check")

\[
\text{HAMPATH} \quad \text{ verifier } V
\]

\[V \left( \langle G, s, k \rangle, P \right)\]

Checking possible path

Is \( P \) a Hamiltonian path in \( G \) that starts at \( s \) and goes to \( k \)?
CLIQUE - in an undir. graph is a subgraph, where every two nodes are connected by an edge.
CLIQUE = \{ (G, k) \mid G \text{ is an}\}
\text{undirected graph w/ an}\}
\text{clique of size k}\}
(\text{k-clique})

CLIQUE is in NP
2 methods of proof
I. give a polynomial time verifier.
II. give a non-det. polynomial time decider.

I. Construct a verifier \( V \) w/ a clique \( c \) as the certificate.

\[ V = \text{on input } \langle \langle G, k \rangle, c \rangle \]

\( O(k) \) 1. Test if \( c \) is a set of \( k \)-nodes

\( O(k^2) \) 2. Test if \( G \) contains all edges connecting nodes in \( c \).

3. If both tests pass, accept otherwise, reject
II. construct a Non det. TM that runs in polynomial time that decides CLIQUE

T= on input \( \langle G, k \rangle \) where

- \( G \) is an undirected Graph and
- \( k \) is a number.

1. non-deterministically select a subset \( C \) of \( k \) nodes in \( G \).
2. Test whether \( G \) contains all edges connecting nodes in \( C \).
3. if yes accept, if no reject.
\text{SUBSET-SUM} = \left\{ <S,t> \right\}

S = \{ x_1, \ldots, x_k \} \quad \text{and for}

\text{some} \quad \{ y_1, \ldots, y_e \} \subseteq S

\text{we have} \quad \sum_{i=1}^{e} y_i = t

\text{power set} \quad P(S) \text{ or } 2^S

I. Construct a verifier \( V \) on input \( \langle \langle S, t \rangle, c \rangle \)

1. Test if \( c \subseteq S \).
2. Test if \( c \) sums to \( t \).
3. If both pass, accept.
   Otherwise, reject.