P, NP, NP-Completeness

- Want to characterize a problem independent of computational hardware.

- What problems are solvable in poly-time? $O(\log n)$
  i.e. in time $2^\log n$
  where $n$ is size of input.

- We need to develop a way to encode a problem with strings, to be entered into a computational device.

Ex: Given a graph $G = (V, E)$ of vertices $u, v \in V$, determine shortest path from $u$ to $v$.

Encoding: if $G = (\{1, 2, 3, 4\}, \{(1, 3), (2, 4), (1, 2), (1, 4)\})$
We can use binary to encode the numbers indexing vertices & parents of brackets as needed.

Full input:

```
("{1, 10, 11, 100}, {0, 1, 11, 10, 100}, (1, 10), (1, 100)}
```

- We want the runtime to depend on the length of the string.

When we use BFS, it takes $O(|V| + |E|)$ operations. We start at $u$ & then $v$.d after it finishes.

- Machine has to parse the input string into graph structure.

- For $G = (V, E)$, how long will encoding be?
(Assume \( V \subseteq \mathbb{N} \))

- We need \( \log |V| \) bits for each vertex.

- \( 2 \cdot |E| \cdot \log |V| \) for edge labels.

- \( 2 \cdot |E| \) for parentheses.

- \( 2 \cdot |E| \) for commas.

- \( 2 \cdot |E| \) for \( "\{\cdot\}" \)

- \( |V| \cdot \log |V| \) for encoding of vertex labels.

- \( |V| \) for commas.

- \( 2 \) for brackets.

- \( 2 \) parentheses outside all.

Total: \( 6 + |V| + |V| \log |V| + 4|E| + 2 \left( \log |V| \right) |E| \)

\[ = O \left( |V| \log |V| + |E| \log |V| \right) \]

\[ = O \left( (|V| + |E|) \log |V| \right) \]
TYPES OF ALGORITHMS:

DECIDER:
Given a string, outputs "yes" or "no".

TRANSFORMER:
Given a string, output a string.

\[ \text{given } G = (V, E), \text{ decide whether } G \text{ is a DAG.} \]

We call the abstract version of this problem

\[ \text{DAG} = \{ <G> : G \text{ is a DAG} \} \]

- \(<.>\) denoted string encoding.
The decision problem above can be restated as:

Determine if \( G \) is a DAG

**Example**

**K-clique** = \( \{ G : G \text{ is an undirected graph and contains a set of vertices that is a } k\text{-clique} \} \)

**Definition**

A k-clique is a set \( S \subseteq V \) such that \( \forall u, v \in S, (u, v) \in E \).

- We want ideas of algorithmic reduction: making one problem look like another which we can solve and translate the solution back to the original.
DEF THE SET \( \mathcal{P} \) IS THE SET OF ALL PROBLEMS FOR WHICH THERE EXISTS A DECIDER THAT RUNS IN POLYNOMIAL TIME.

DEF NP IS THE SET OF PROBLEMS FOR WHICH THERE EXISTS A DECIDER TO VERIFY THAT A PROBLEM HAS BEEN SOLVED CORRECTLY, WHEN GIVEN A PROBLEM INSTANCE OR CERTIFICATE. IT MUST ALSO RUN IN POLY TIME.
\[ \text{HAMPATH} = \{ G : G \text{ is a graph and } G \text{ contains a Hamiltonian path} \} \]

Recall: A Hamiltonian path is a path through a graph which passes through each vertex exactly once.

Def: A cycle is Hamiltonian if the path forming it is Hamiltonian.

\[ \text{EULEPATH} = \{ G : G \text{ is a graph that has a path passing through every edge exactly once} \} \]

Def: Eulerian path contains each edge exactly once.
FACT: EULER-PATH ∈ P

B/C HAS E-P IFF ALL VECTORS ARE EVEN DEGREE.
(SEE DISCRETE NOTES)

IS HAM-PATH ∈ P?
OR HAMCYCLE ∈ P?

WE CAN KNOW

HAM-PATH ∉ HAMCYCLE ∈ NP
B/C IF GIVEN \( G=(V,E) \) & A PATH \( P \in V \)
WE CAN CHECK TO SEE

1) \( P \) IS A VALID PATH
   i.e. \( (P_i, P_{i+1}) \in E \) \( \forall i \)

2) \( V = P \) & EACH VERTEX APPEARS ONLY ONCE

-TAKES ONLY POLY(\( |V| \)) TIME
FACT \( P \subseteq NP \)

If we can solve \( NP \) in poly time, we can verify in poly-time.

\[ \text{HAM-CYCLE} \subseteq \text{NP} \]

\[ \text{P} \]

- \text{RED-CUT}
- \text{EULER-PATH}
- \text{DAG}

NOTE: NOT KNOWN IF \( \text{HAM-CYCLE} \in \text{P} \).
MILLION-DOLLAR QUESTION:

DOES P = NP?

MOST BELIEVE NO.

ALGORITHMIC REDUCTION

DEF For languages/problems

L₁ and L₂, we say that

L₁ is polynomial-time reducible to L₂, denoted

L₁ ≤ₚ L₂, if there exists a function f such that

for any x, we have

f(x) ∈ L₂ iff x ∈ L₁,

and we can compute f in poly-time.
FACT: If \( A \leq_p B \) \& \( B \leq_p \), then \( A \leq_p \).

Proof:

An Alg. for "A" is:
- Given instance \( x \)
- Compute \( f(x) \) where
  is reduction from \( A \) to \( B \)
- Run Alg. for \( B \) on \( f(x) \)
- Return true if \( f(x) \in B \)
A problem $P$ is NP-complete if

1) $P \in \text{NP}$

2) Every problem $L \in \text{NP}$, $L \leq_p P$ (we call this NP-hard).

Because

$f(x) \in B$ iff $x \in A$.

So if $P$ is false otherwise.

$A \leq_p B$. 

DEF: A problem $P$ is NP-complete if
TSP IS NP-HARD BUT NOT KNOWN TO BE IN NP.

Fact using above, if L is NP-complete then prove that L is in P, then P = NP.

Examples

VERTEX-COVER $\leq_P$ HAM-CYCLE

CLIQUE $\leq_P$ VERTEX-COVER

$\exists$-SAT $\Sigma \{ \phi \} : \phi$ is a $\exists$-CNF boolean formula with a satisfying solution

Cook-Levin Theorem

$\exists$-SAT IS NP-COMPLETE.
3-CNF

\((x_1 \lor x_2 \lor \overline{x}_1) \land (\ldots) \land (\ldots)\)

Find satisfying assignment of variables.

Fact: \(3\text{-SAT} \leq_P \text{CLIQUE}\)

Fact: If \(L_1\) is \(NP\)-hard and \(L_1 \leq_P L_2\), then \(L_2\) is \(NP\)-hard.