CMSC 451 - Algorithm Design
Lecture 15 - NP-Completeness: General Definitions

So far...

- How to design and analyze efficient algorithms for combinatorial problems
- What do we mean by efficient?
  - worst-case (not average case)
  - asymptotic efficiency - is $10^{42} \cdot n$ efficient?
  - running times depending on numeric parameters? - $m^2 \cdot \log C$ (max flow)

- Important classification:
  - Polynomial time - $O(n^c)$, where c = constant (worst case)
  - Exponential time - $\Omega(c^n)$, $c > 1$

- Henceforth:
  - Polynomial = Efficient
  - Exponential = Inefficient
- Is this reasonable?
  - Not entirely, but it works most times
  - $n^{1,000}$ vs. 1.0001^n - Possible, but uncommon

- Polynomials are nice because they are closed under composition:

$f(n) = O(n^a)$ $g(n) = O(n^b)$

$\Rightarrow f(g(n)) = O((n^b)^a)$

$= O(n^{a \cdot b})$ = polynomial

- Upshot: If a poly-time function f calls a poly-time function g as a subroutine the result runs in poly time.

Hard Problems:

Near end of 1960's:

- Poly-time algorithms for many problems
- Many problems defied poly-time solutions
  - Best solutions based on brute-force
    $\Rightarrow$ Exponential time

- Researchers sought the crucial property that distinguishes between them

- Spoiler alert - They failed!

Enter Stephen Cook (1971)

- For many combinatorial problems, the best solution has the following form:
  - guess the answer + verify it is correct

- Guessing ≡ non-determinism (computer has many possible actions)
- Non-deterministic Turing machine (NDTM)

- Q: What can NDTM compute in poly-time?
  → called NP - Nondeterministic Poly-time

- NP-Complete - The "hardest problems" in NP

> If you can solve any of these efficiently - you can solve every problem in NP efficiently.

NO WAY!

- Okay - This is nice in theory, but who cares?

Enter Richard Karp (1972)

- Showed that 21 well-known problems are NP-Complete - This is major!
- Is P=NP? $1M prize for the answer

Many NP-complete problems are closely related to efficiently solvable problems.

Hard Problems (NP-complete)	Easy Problems (in P)
3SAT	2SAT
Traveling Salesman Problem (TSP)	Minimum Spanning Tree (MST)
Longest (Simple) Path	Shortest Path
Hypergraph Matching	Graph Matching
Knapsack	Unary Knapsack
Independent Set in Graphs	Independent Set in Trees
Integer Linear Programming	Linear Programming (weak poly time)
Hamiltonian Cycle	Eulerian Cycle
Balanced Cut	Minimum Cut

Input Size + Representations

- Efficiency is measured in terms of running time as a function of input size
- For this to be meaningful, inputs should be encoded efficiently

Not unary! 8 = 11111111

Numbers - integers given in binary or decimal
- rationals as ratio of integers: 7/13 or fixed point decimals: 17.3741
- arbitrary reals (e.g. π) + unbounded values (e.g. ∞) not allowed

Graph - Adjacency matrix or adj. list

Sets - List of elements or bit vector

Decision Problems + Language Recognition

- It will simplify the theory to assume all problems have same output - yes or no
- Called a decision problem
- Optimization problems have a decision problem variant. E.g.

Optimization: Find min. spanning tree for G

Decision: Given $G + z \geq 0$, does G have a spanning tree of weight $\leq z$?

Optimization $\Rightarrow$ Decision (trivial)

Decision $\overset{?}{\Rightarrow}$ Optimization

(binary search to find opt z)

Language Recognition

- For historical reasons (automata theory predates algorithm theory) NP problems are expressed as language recognition

- Assume we have function

serialize(I) - Encodes input I (e.g., graph, set, number) as a string

Decision problems	→	Language recognition
Does G have an MST of weight $\leq z$?	→	MST = $\{ G \mid$ MST weight of G $\leq z \}$
Does G have a Hamiltonian cycle?	→	HC = $\{ G \mid$ G has a Hamil. cycle $\}$

Complexity Class P:

Decision Problems

Def: P is the set of languages such that membership can be determined in polynomial time.

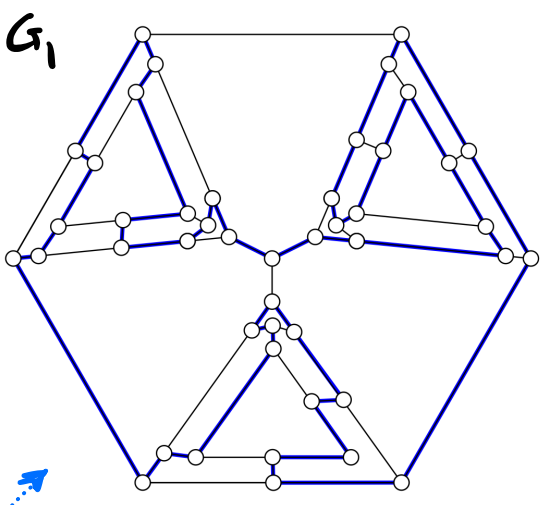
Eg. Can solve MSTs efficiently, so MST $\in$ P

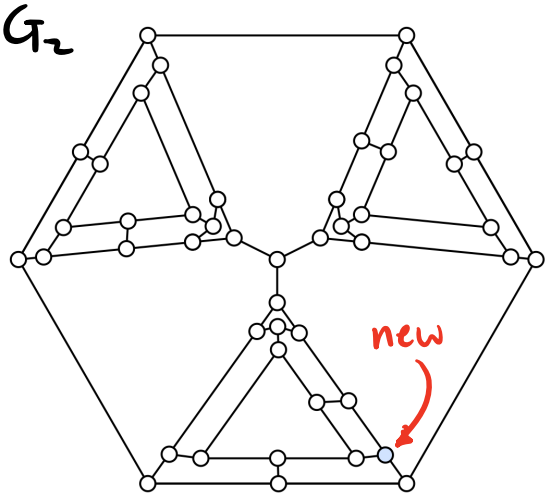
Are there problems (languages) not in P?

Hamiltonian Cycle: Given a graph G, a Hamiltonian cycle is a simple cycle that visits each vertex of G (exactly once)

Encoded as a string

$$HC = \{ G \mid G \text{ has a Ham. cycle} \}$$





$G_1 \in HC$ $\quad$ $G_2 \notin HC$? (don't know)

Is $HC \in P$? (Not known to science)

HC has a nice property - verifiability

- If $G \in HC$, it is easy to verify this
  (Show me the cycle, and I'll check)
- However, if $G \notin HC$, how to verify ???
  (No idea)

Complexity Class NP:

> Def: NP is the set of languages such that membership can be verified in poly time.

What do you mean by "verified"?

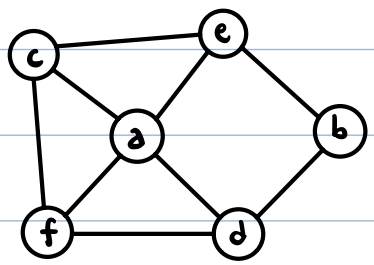
Given a **language L** (set of strings)

- Given **string x**, want to know whether **$x \in L$**.
- A **certificate** is a **"helper string" y**:
  - if $x \in L$, y can be used to **prove** this in **polynomial time**
  - if $x \notin L$, then **we don't care**

Examples:

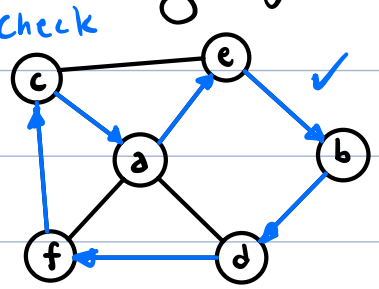
$HC = \{ G \mid G$ has a Hamil. Cycle$\}$

Certificate = List of vertices forming cycle

x: 

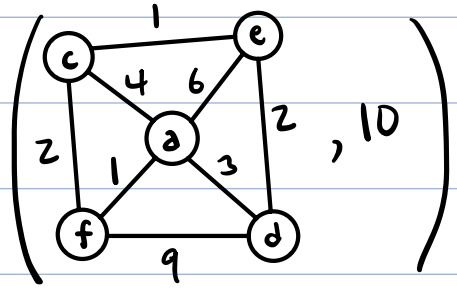
y: ⟨c, a, e, b, d, f⟩

Check



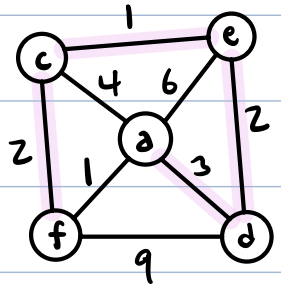
MST = {(G, z) | G has spanning tree of weight ≤ z}

Certificate = Edges of spanning tree

x: 

y:
(c, f)
(c, e)
(d, e)
(a, d)

Check



2
+1
+2
+3
8 ≤ 10 ✓

This is weirdly assymetric!

- if $x \in L$ - need to verify
- if $x \notin L$ - don't care

That is just how it is defined.

Are their (natural) languages that are not poly-time verifiable?

$UHC = \{G \mid G \text{ has a unique Ham. cycle}\}$

$\overline{HC} = \{G \mid G \text{ has no Hamil. cycle}\}$

If $x \in UHC$, you can show one Ham. cycle, but how do you prove there no other?

If $x \in \overline{HC}$, how do prove there is no Ham. cycle?

Good candidates, but don't know for sure (depends on whether P=NP)

Summary:

- How to show problems are hard?
- Poly-time ≡ easy Expon. time ≡ hard
- Decision probs. + Language recognition
- P, NP + verification
- Next - NP-Completeness