Existence of Hard Problems

We've seen a bunch of algorithms to solve various problems this semester. The runtimes of these algorithms have mostly been polynomial in the input size:

I.E: $O(n^c)$, where c is a constant

$= O(n^{O(1)})$

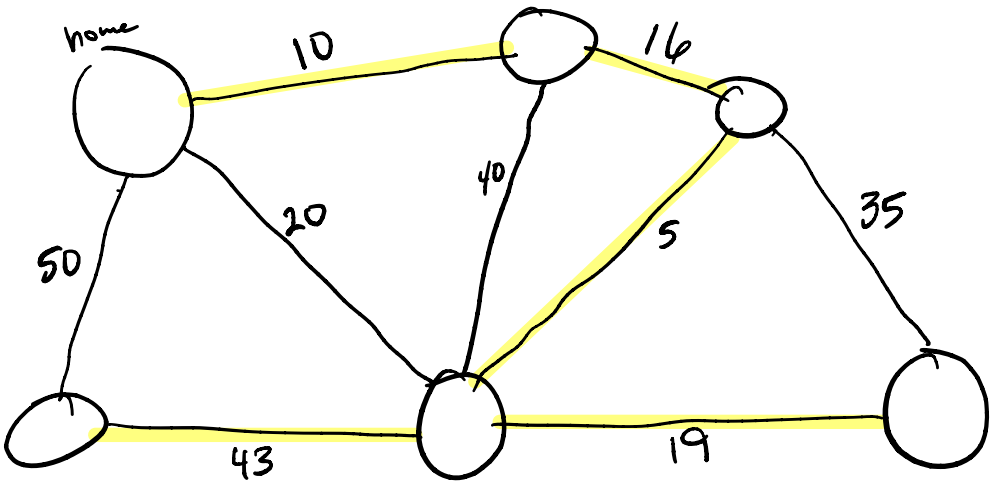
⇒ Can we solve all (or most of) interesting problems in polynomial time? // want to say yes, but I think there may be unsolvable problems ∴ 2^n = exponential, not polynomial

Traveling Salesperson Problem

Input: Undirected graph with weights on edges

Output: Shortest tour ("cycle") that visits each vertex exactly once
↳ "cheapest path that does round-trip"

Ex:



// want to go from home to all cities

= shortest cycle that visits all vertices, total cost = 93

Best known algorithm: $O(n2^n)$ time.
↳ Exponential time, very slow

What can we do?

- Spend more time designing algorithms for those problems
  - People tried for a few decades, no luck
- Prove there is no polynomial time algorithm for those problems
  - Would be great
  - Seems really difficult to prove
  - Best lower bounds are not strong enough to show this.

What else can we do?

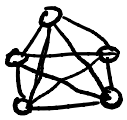
- Let's try to identify hard problems which are essentially equivalent. I.e., if we can solve one of them in polynomial time, then all others can be solved in polynomial time as well.

=> P vs. NP. (complexity classes)

Another difficult problem

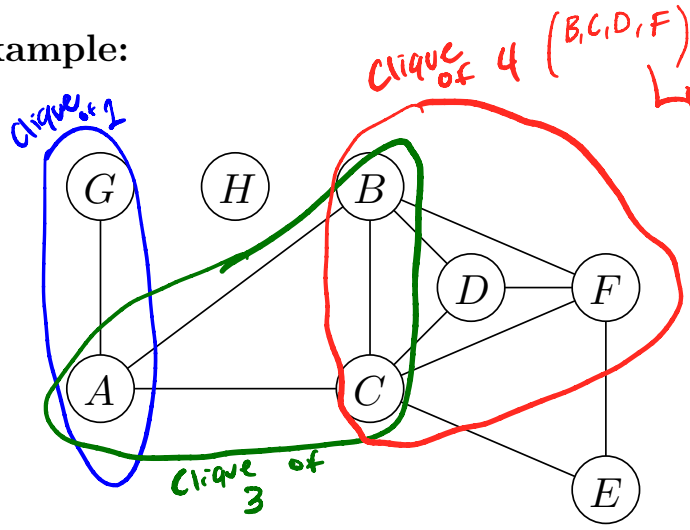
- Clique (optimization variant):
  - **Input:** Undirected graph $G = (V, E)$
  - **Output:** Largest subset C of V such that every pair of vertices in C has an edge between them (C is called a *clique*)

i.e.:



∴ every pair of vertices connected to eachother

Clique example:



Best known algorithm runtime

$\Rightarrow O(n \cdot 2^n)$

also exponential

Let's consider an even simpler variant of this *Clique* problem.

Decision problem vs. optimization problem

- 3 variants of Clique:

(1) **Input:** Undirected graph $G = (V, E)$, and an integer $k \geq 0$.
Output: Does G contain a clique C such that $|C| \geq k$?

returns boolean

(2) **Input:** Undirected graph $G = (V, E)$
Output: Largest integer k such that G contains a clique C with $|C| = k$. (removes ambiguity, as we are only solving for 1 #) returns integer

from previous page →

(3) **Input:** Undirected graph $G = (V, E)$
Output: Largest clique C of V. returns clique

- (3) is harder than (2) is harder than (1) So, if we reason about the decision problem (1), and can show that it is hard, then the others are hard as well. Also, every algorithm for (3) can solve (2) and (1) as well.

Theorem:

If (1) can be solved in polynomial time, then (2) can be solved in polynomial time. $O(n^c)$ where c is constant

Proof:

=> run (1) for values of $k = 1, 2, 3, 4, 5, \ldots, |V| = n$

=> return for (2) the largest possible k, where (1) returned true

=> runtime for (2):

$\in O(\text{runtime (1)} \cdot n) = O(n^c \cdot n)$

Theorem:

If (2) can be solved in polynomial time, then (3) can be solved in polynomial time.

Proof:

=> Run (2) to find size of largest Clique

=> Try removing a vertex from the graph

=> rerun (2) & see if clique size changes

=> repeat until we find all vertices in clique (restore vertices that change clique size, but remove others)

=> Runtime (3)

$\in O(\text{runtime}(2) \cdot n)$

↳ for every vertex

These techniques generalize to allow us to focus on solving the decision variant of these hard problems.

=> Optimization variant can be solved from decision variant

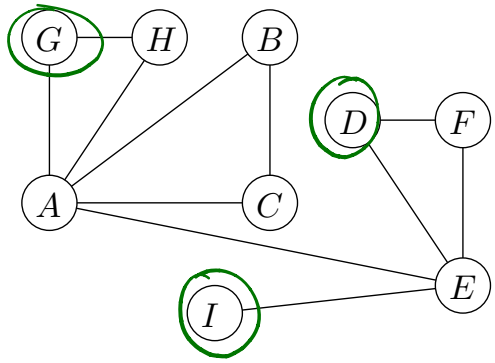
-Polytime Reductions

Here's another seemingly simple problem with no known polynomial time solution.

- Independent Set (usually abbreviated to IS):
  - **Input:** Undirected graph $G = (V, E)$, and an integer $k \geq 0$.
  - **Output:** Does G contain an independent set S such that $|S| \geq k$?

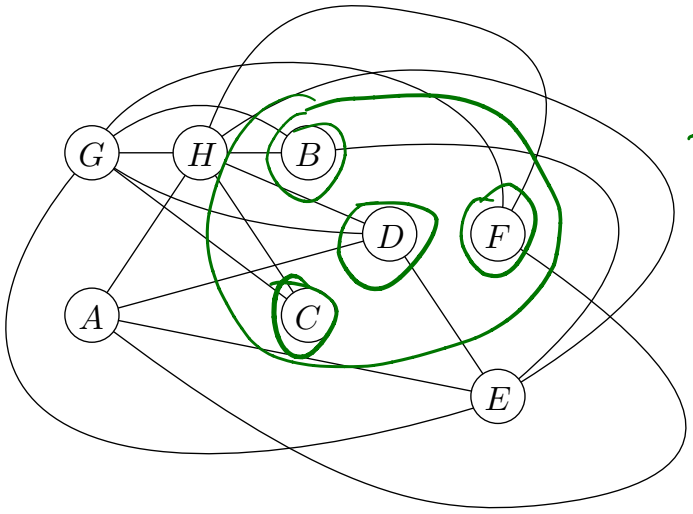
(An **independent set** in a graph is a set of vertices where no pair of them is connected.) - NO connections between vertices

IS **example 1:**



- largest IS set of size = 4
= {G, B, D, I}

IS **example 2:**



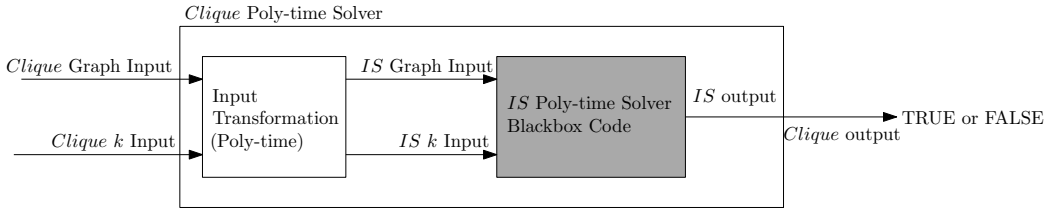
- same vertices of clique problem in the (IS)

=> they are just (reversed the edges connections)

Does this problem seem familiar?

=> Notice this problem is similar to the Clique problem. In the clique problem, we wanted to find set of vertices that are all connected. In the (IS) problem we are looking for a set of vertices that are not connected

Suppose a polynomial time algorithm for IS was found. What does this tell you about *Clique*?



Let the *Clique* graph input be denoted as $G_C = (V_C, E_C)$ and let the *Clique* k input be denoted as k_C.

Let the IS graph input be denoted as $G_{IS} = (V_{IS}, E_{IS})$ and let the IS k input be denoted as k_{IS}.

How does the input transformation change the inputs to *Clique* to get an equivalent IS problem?

$k_{IS} =$ k_C (size of clique became the IS, so this stays the same)

$V_{IS} =$ V_C

$E_{IS} =$ $\overline{E_C} = \{(a,b) \mid (a,b) \notin E_C\}$ (negation of E_C = reverse edges)

= "All edges, where that edge was not originally present"

- How long to reverse edges?

$\Rightarrow O(|V_C|^2)$ = # of possible edges

= Can turn any Clique problem into a IS problem in polynomial time

We just demonstrated that any *Clique* problem can reduced in polynomial time to an equivalent IS problem. This is denoted as $Clique \leq_p IS$. - reducible in poly-time to

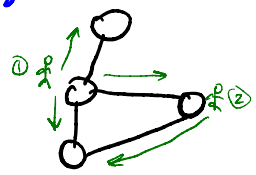
Using similar logic we could also show $IS \leq_p Clique$. Thus these problems are equivalent.

Let's try out another seemingly simple graph problem.

- Vertex cover (abbreviated to VC) = Vertices that cover all edges
  - **Input:** Undirected graph $G = (V, E)$, and an integer $k \geq 0$.
  - **Output:** Is there a subset S of V, $|S| \leq k$, such that each edge in E is incident to at least one vertex in S.

= Set of K locations to station a guard so every edge visible

I.E:

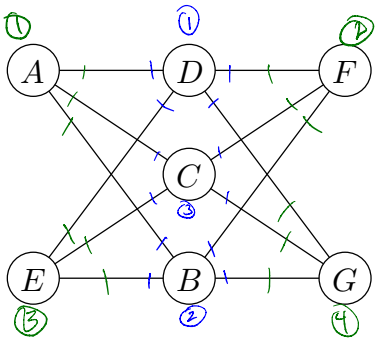


= adding K=2, lets us see all edges

VC example 1:

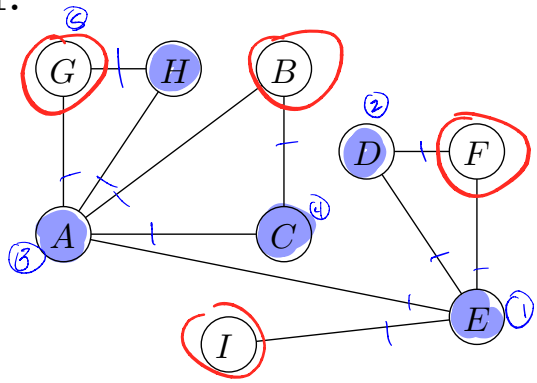
=> Every edge seen for size K = 4

=> Every edge seen for size K=3



VC example 1:

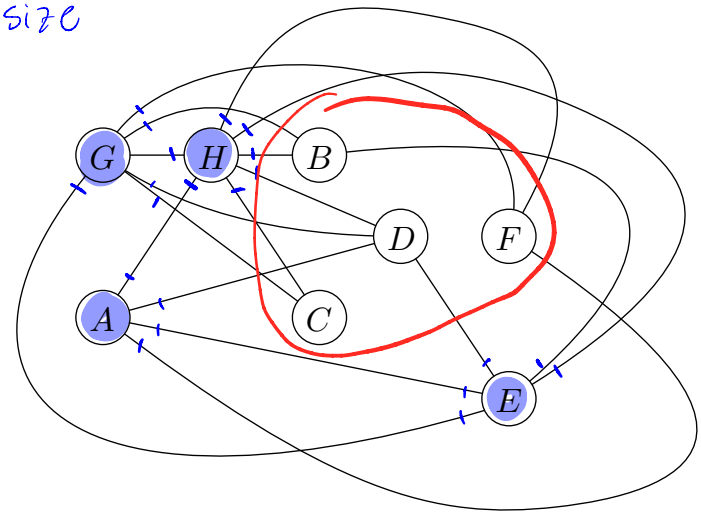
=> Vc of size K=5



∴ Notice non guarded vertices, create an IS of size K=4

VC example 2:

=> Vc of size K=4



is it harder to find VC of K, or K+1?

=> Unlike IS problem, it is harder to find VC of size K

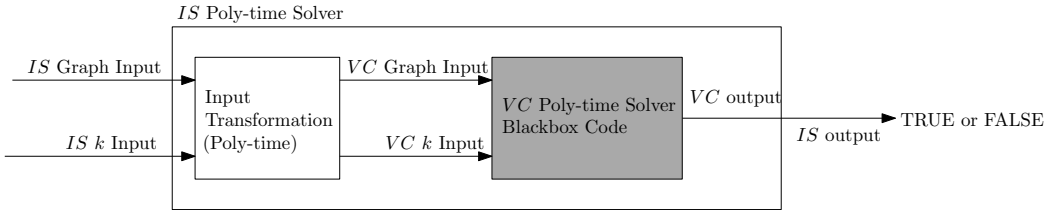
(K = small)
(K+1 = large)

What do notice about the vertices not in the vertex cover?

=> They form an IS

- Reduction example

Suppose a polynomial time algorithm for VC was found. Let's show that IS is now solvable in polynomial time. I.e. let's prove $IS \leq_p VC$.



Let the IS graph input be denoted as $G_{IS} = (V_{IS}, E_{IS})$ and let the IS k input be denoted as k_{IS}.

Let the VC graph input be denoted as $G_{VC} = (V_{VC}, E_{VC})$ and let the VC k input be denoted as k_{VC}.

$K_{IS} = 4$

$|V| = 8$

thus

$K_{VC} = |V| - K_{IS}$

How does the input transformation change the inputs to IS to get an equivalent VC problem?

- To answer: Did we change any vertices or Edges?
  = NO, the IS just showed itself

$k_{VC} = |V_{IS}| - K_{IS}$ - look at VC examples, notice Kvc is different from KIS

$V_{VC} = V_{IS}$

$E_{VC} = E_{IS}$

} = No changes

- runtime

O(1), just subtraction

$\Rightarrow IS \leq_p VC$ = ("IS is polynomial time reducible, to vertex cover")

$\Rightarrow IS =_p VC$, tells us

$\Rightarrow Clique =_p IS =_p VC$

Using similar logic we could also show $VC \leq_p IS$. Thus these problems are equivalent.

- reduction on non-graph problems

These reductions seem pretty straightforward so far. But what about hard problems that don't involve graphs directly? Can we reduce them to any of these previous problems?

- Satisfiability problem (abbreviated to SAT):
  - **Input:** a boolean formula f with m clauses over n variables.
  - **Output:** TRUE if there is an assignment to the variables that makes the formula true, FALSE otherwise.

(A clause consists of one or more literals connected with "or" operators. We then "and" together clauses to get our formula.)

SAT example

5 clauses over 4 variables $\{x_1, x_2, x_3, x_4\}$:

$$(\neg x_1 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_4)$$

- Making Satisfying Assignment (makes this true)

$X_1 = T$

$X_4 = T$

$X_3 = F$

$X_2 = F$

=> All clauses now evaluate to True & once we ($\wedge$) together we return True overall

∴ This isn't only assignments that satisfy problem

=> $X_1 = T$, $X_2 = T$, $X_3 = T$, $X_4 = T$ - works aswell, many options

We can prove that $SAT \leq_p Clique$. That is to say, a polynomial time solution to *Clique* also yields a polynomial time solution to *SAT*

clauses ↓ literals/variables ↓

- Given a *SAT* formula $f = C_1, \ldots, C_m$ over $x_1, \ldots, x_n$, we need to produce $G = (V, E)$ and k such that f is satisfiable $\Leftrightarrow$ G has a clique of size $\geq k$.

Reduction Idea:

- Reminder: a literal is either x_i or $\neg x_i$
- For each literal s occurring in a clause in f, create a vertex v_s // e/a $x_1, x_2, x_3 \ldots$ is a vertex
- Create an edge $\{v_s, v_t\} \Leftrightarrow$:
  - s and t are not in the same clause, and // if 2 variables not in same clause connect
  - s is not the negation of t // can't have x_1 & $\neg x_1$
- Let $k = m$ (i.e, k is equal to the number of clauses)

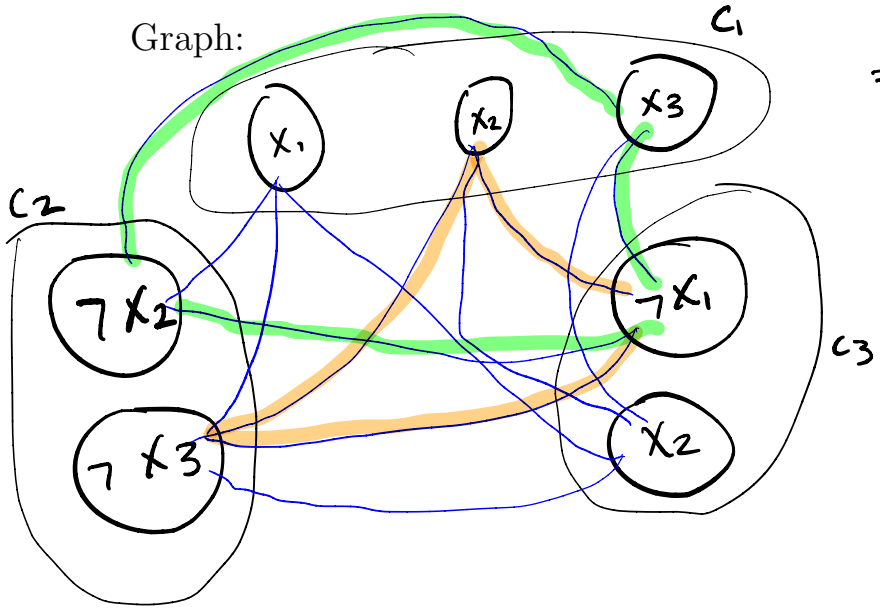
SAT $\leq_p$ *Clique* **example 1**

Formula: 3 clauses, thus $k = 3$

C_1 C_2 C_3

$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)$

Graph:



=> Satisfiable if we can find a clique ~~of size k~~ let $k = 3$

= another Clique of size 3

=> using this clique, setting $x_2 = T$, $x_3 = F$, $x_1 = F$ which gives a set of literals when $\wedge$ returns True which will satisfy SAT problem

=> 1 literal from e/a clause

$SAT \leq_p Clique$ **example 2**

Formula:

$(\neg x_1 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_4)$

Graph:

To recap, we have thus far demonstrated:

$SAT \leq_p Clique \leq_p IS \leq_p VC$; => SAT $\leq_p$ Clique $= p^{IS} = p^{Vc}$

=> SAT $=_p$ Clique $= p^{IS} = p^{Vc}$

Equivalence Classes

Benefits of equivalence:

- Combines research efforts:
  - If one problem has a polynomial time solution, then all of the equivalent problems do.
  - If an exponential lower bound is shown for one problem, it holds for all of them.
- If we show that a problem Π is equivalent to many other well studied problems without efficient algorithms, then we get very strong evidence that Π is hard.

Class of problems: P

This is the set of decision problems that can be correctly decided in polynomial time. Which of the following problems are in P?

- Sortedness: yes (walk through comparing elements) $\in P$
  - **Input:** an array of integers A
  - **Output:** TRUE if A is in (ascending) sorted order, FALSE otherwise

- Shortest Path: yes (Dijkstra's) $\in P$
  - **Input:** a graph G, a pair of vertices v_s and v_f, an integer k
  - **Output:** TRUE if G contains a path from v_s to v_f which uses k edges or fewer, FALSE otherwise

- Sorting: NO! (not a decision problem; can't output just a yes or no)
  - **Input:** an array of integers A
  - **Output:** Sorts the elements in A $\notin P$

- Independent Set: Maybe? (we don't know)
  - **Input:** Undirected graph $G = (V, E)$, and an integer $k \geq 0$.
  - **Output:** Does G contain an independent set S such that $|S| \geq k$?

Next, let's now try to characterize this equivalence class that we've been building.

Class of problems: NP

This is the set of decision problems which, when given a purported solution (a certificate), you can verify that solution in polynomial time.

Examples of what this means

- Clique:
  - **Input:** Undirected graph $G = (V, E)$, and an integer $k \geq 0$.
  - **Output:** Does G contain a clique S such that $|S| \geq k$?

Clique $\in NP$**:** The certificate is a set of vertices S in G which form a clique of size k (or greater). We can test whether these vertices do form a clique in polynomial time (test all $O(|S|^2)$ edges).

- Independent Set:
  - **Input:** Undirected graph $G = (V, E)$, and an integer $k \geq 0$.
  - **Output:** Does G contain an independent set S such that $|S| \geq k$?

$IS \in NP$**:** The certificate is a set of vertices S in G which form an independent set of size k (or greater). We can test whether these vertices do form an independent set in polynomial time (test all $O(|S|^2)$ edges).

- Vertex cover:
  - **Input:** Undirected graph $G = (V, E)$, and an integer $k \geq 0$.
  - **Output:** Is there a subset S of V, $|S| \leq k$, such that it is a vertex cover for G.

$VC \in NP$**:** The certificate is a set of vertices S in G which form a vertex cover of size k (or less). We can test whether these vertices do form a vertex cover in polynomial time (test whether every edge in E is covered).

- Is there a certificate for these?

- Satisfiability problem (abbreviated to SAT): Yes!
  - **Input:** a boolean formula f with m clauses over n variables.
  - **Output:** TRUE if there is an assignment to the variables that makes the formula true, FALSE otherwise.

$SAT \in NP$:

=> given assignment of variables, we can plug in & evaluate formula to see if satisfied

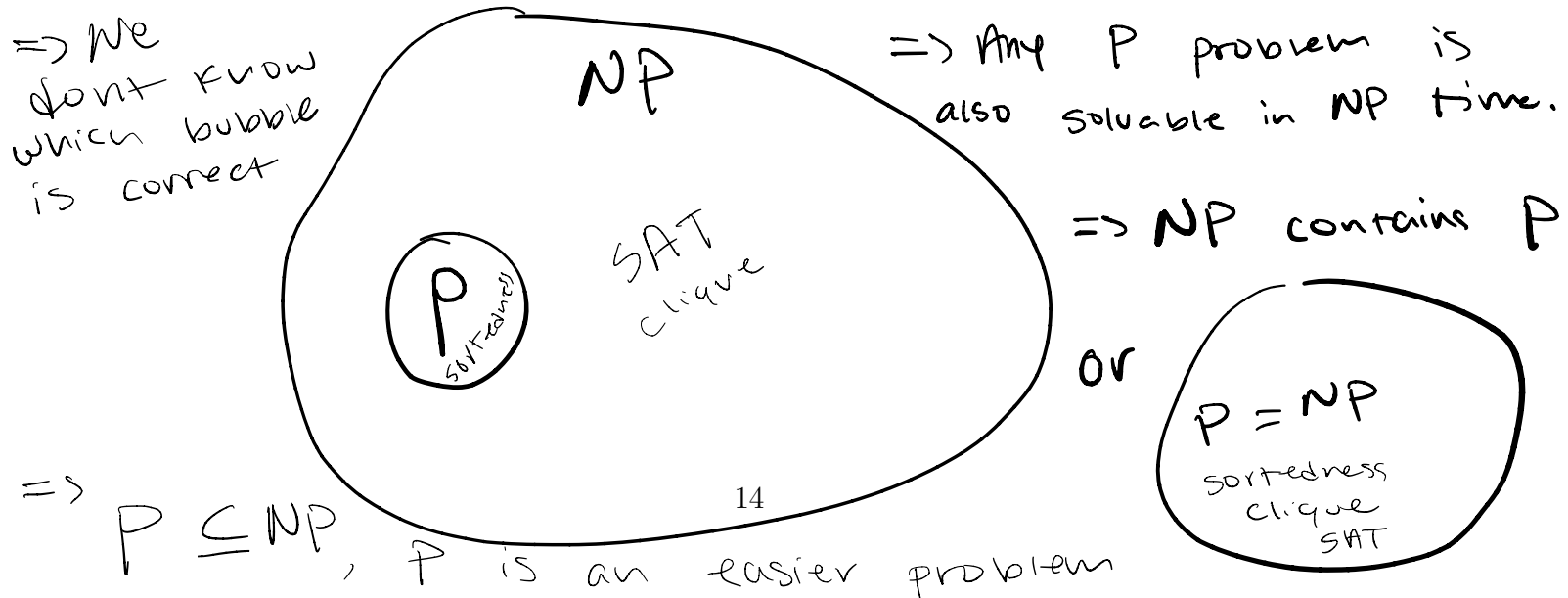
=> I.E. We are given $x_1 = T$, $x_2 = F$, ..., etc.

- Sortedness: Yes!
  - **Input:** an array of integers A
  - **Output:** TRUE if A is in (ascending) sorted order, FALSE otherwise

$Sortedness \in NP$: given Array A, we can just check if sorted in polynomial time

No certificate needed

What do you notice about any problem in P?



Alternate definition for NP

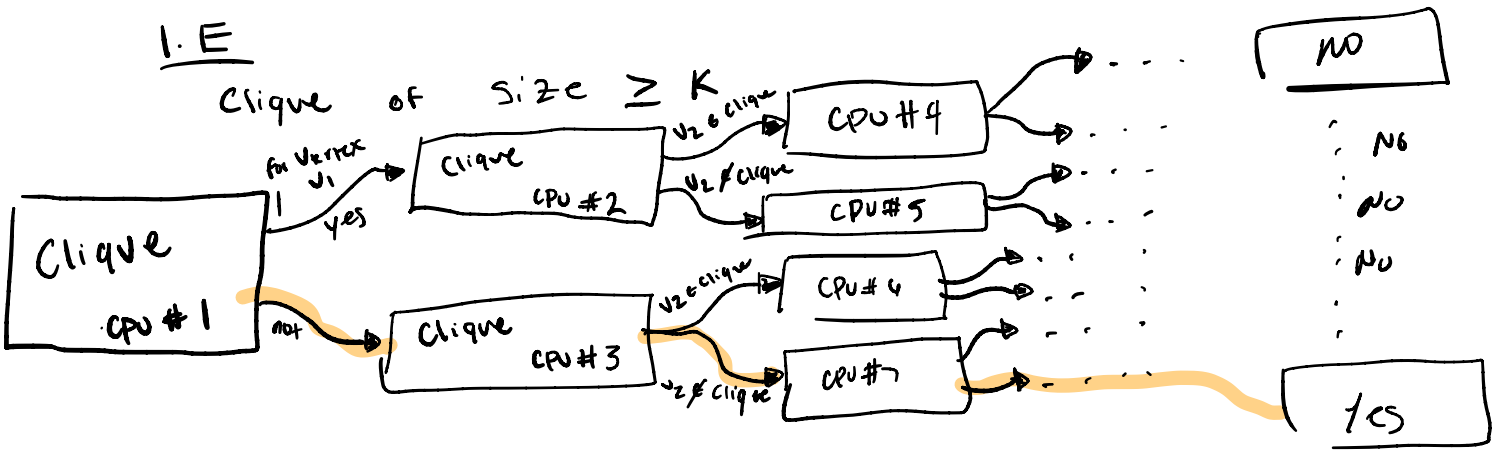
Alternatively this is the set of decision problems that can be correctly decided in nondeterministic polynomial time.

What does **nondeterministic polynomial time** mean?

↳ Imagine we have infinite # of Processors in computer

=> We could solve NP problems in Polynomial time

I.E



=> Eventually e/a vertex will have a decision made. All vertices will be choosen & we do a poly-time check of set of vertices, if they form Clique ≥ K

=> Upon yes we just return back getting all of the vertices

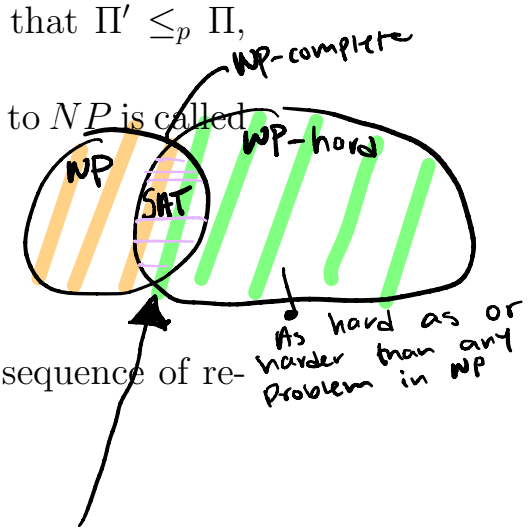
=> (height of Tree gives cost) + (time to verify certificate)

=> parralleization can help w/ these NP problems

SAT is NP-complete

↳ Hardest problem in NP

- **Fact:** $SAT \in NP$
- **Theorem [Cook71]:** For any $\Pi' \in NP$ ("problem"), we have $\Pi' \leq_p SAT$. // SAT can simulate other problems
- **Definition:** If for any $\Pi' \in NP$ it's the case that $\Pi' \leq_p \Pi$, then Π is called *NP-hard*
- **Definition:** An *NP-hard* problem that belongs to *NP* is called *NP-complete*
- **Corollary:** *SAT* is *NP-complete*.



Plan of attack:

Show that the problems are in *NP*, and show a sequence of reductions from other *NP-hard* problems:

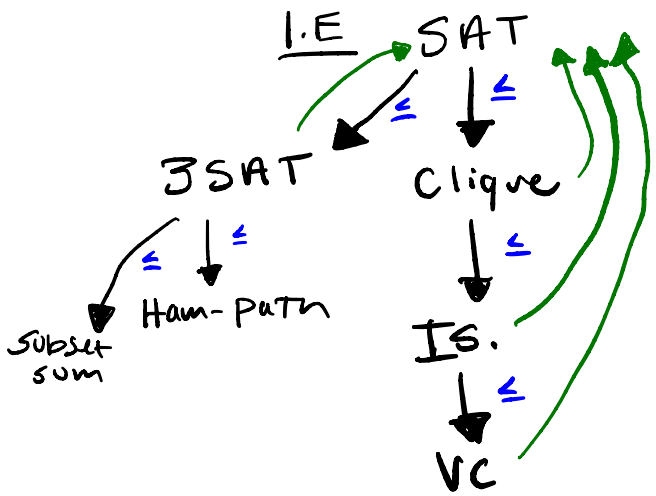
=> $SAT \leq_p$ Clique

($SAT =_p$ Clique $=_p$ IS $=_p$ VC = ...)

// Many problems are at the intersection

- Plan of Attack

- reduce solving known hard problem to solving that problem



// reduced solving SAT to solving Clique, IS, VC

=> Clique is NP-complete

// $\leq$ = polynomial time reductions

// All other are reduciable to SAT

Conclusion: all of the above problems are *NP-complete*

★ Given a new problem, not sure if NP-complete, you try to reduce it back to something known to be NP-complete (reciprocal direction implied) ★

$3SAT$

- Input: a boolean formula f with m clauses over n variables where every clause contains exactly 3 literals.
- Output: TRUE if there is an assignment to the variables that makes the formula true, FALSE otherwise.

3SAT example

$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_2 \vee \neg x_1) \wedge \ldots$

$\Rightarrow$ 3SAT $\in$ NP (since we can plug-in a given solution & evaluate formula(f) in polynomial time aka: literal T/F values are given)

$\Rightarrow$ $SAT \leq_p 3SAT$ **reduction idea**

- How would you turn SAT problem into 3SAT problem

1.) too few literals in a clause

I.E $(x_1 \vee \neg x_2)$ $\therefore x_1 = T$

↳ just add duplicate to reach 3SAT & maintain truth value $= (x_1 \vee \neg x_2 \vee x_1)$

2.) TOO many literals in a clause

I.E $(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_1)$ // 1 too many literals

↳ split clause & add a new literal to satisfy Truth value. Only 1 clause I.E; x_{new} needs to be True

// if you have 5 literals in a clause
=> split into 3 clauses
=> 2 new literals
- x_{new1}
- x_{new2}

$\equiv$

$(x_1 \vee \neg x_2 \vee x_{new}) \wedge (x_3 \vee \neg x_1 \vee \neg x_{new})$

= 1 of these clauses must be True while other doesn't matter

2SAT

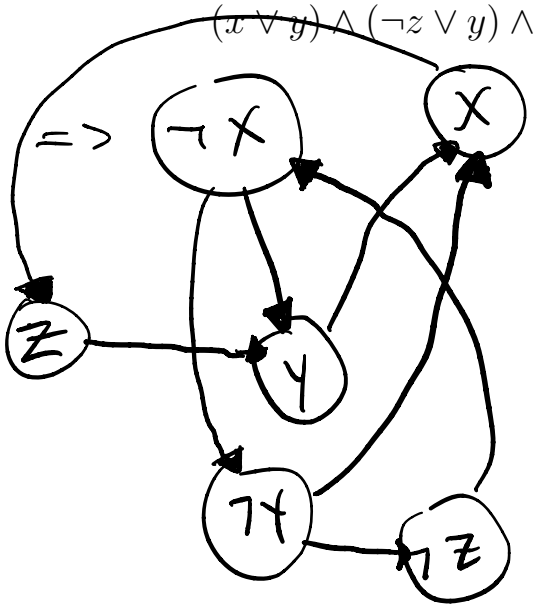
- **Input:** a boolean formula f with m clauses over n variables where every clause contains exactly 2 literals.
- **Output:** TRUE if there is an assignment to the variables that makes the formula true, FALSE otherwise.

Is 2SAT also *NP-complete*? NO

2SAT $\in$ P

Consider the following formula:

$(x \vee y) \wedge (\neg z \vee y) \wedge (\neg y \vee x) \wedge (z \vee \neg x)$



$\neg X \rightarrow X$

contradiction impossible

say $X = T$

$\therefore \neg X \rightarrow Y\ (T)$

$\neg Y \rightarrow X\ (T)$

$Z \rightarrow Y\ (T)$

(T)

$\neg Y \rightarrow \neg Z\ (T)$

$Y \rightarrow X$

$\neg X \rightarrow \neg Y$

$\neg Z \rightarrow \neg X$

$X \rightarrow Z$

$X = T$
$Y = T$
$Z = T$
) - satisfying assignment

- Use graph traversal alg. to check this

Issues when:

$A \rightarrow \neg A$
$\neg A \rightarrow A$
) = NO satisfy assignment possible

Here's another problem:

0-1 Knapsack

- **Input:** Given a set of n objects numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum carry capacity W, and an integer k.
- **Output:** TRUE if there a set of objects that you can carry such that the total value is $\geq k$ without exceeding the carry capacity W, FALSE otherwise.

Is 0-1 Knapsack *NP-complete*?

Let $T[n, W, k]$ denote the 0-1 knapsack solution for a given n, W, and k.

Base cases:
$T[n, W, k] =$
(where $W \leq 0$)

$T[0, W, k] =$
(where $k > 0$)

$T[n, W, k] =$
(where $k \leq 0$ and $W > 0$)

Recursive case:
$T[n, W, k] =$

Here's a simpler variant of the 0-1 knapsack problem:

Subset Sum Problem

Similar to 0-1 knapsack but where $v_i = w_i$ (i.e., weights and values are the same).

- **Input:** Given a collection of n objects numbered from 1 up to n, each with a value/weight v_i and a target W.
- **Output:** TRUE if there a set of objects that you can carry such that the total value is $= W$, FALSE otherwise.

Example is there subset that sums to 32: $\{02, 10, 11, 31, 20, 03\}$

Surprisingly you can prove that 3SAT $\leq_p$ Subset Sum. Below is an example reduction.

We have n variables x_i and m clauses. For a clause c_j for each variable x_i, construct numbers t_i and f_i of $n+m$ digits:

- The i-th digit of t_i and f_i is equal to 1
- For $n+1 \leq j \leq n+m$, the j-th digit of t_i is equal to 1 if x_i is in clause c_j
- For $n+1 \leq j \leq n+m$, the j-th digit of f_i is equal to 1 if $\neg x_i$ is in clause c_j
- All other digits of t_i and f_i are 0

Consider the following formula:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

	x_1	x_2	x_3	c_1	c_2	c_3	c_4
t_1							
f_1							
t_2							
f_2							
t_3							
f_3							

For each clause c_j, construct numbers x_j and y_j of $n + m$ digits:

- The $(n + j)$-th digit of x_j and y_j is equal to 1
- All other digits of xi and yj are 0

Again consider the following formula:

$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$

	x_1	x_2	x_3	c_1	c_2	c_3	c_4
x_1							
y_1							
x_2							
y_2							
x_3							
y_3							
x_4							
y_4							

Finally, construct a sum number s of $n+m$ digits:

- For $1 \leq j \leq n$, the j-th digit of s is equal to 1
- For $n+1 \leq j \leq n+m$, the j-th digit of s is equal to 3

	x_1	x_2	x_3	c_1	c_2	c_3	c_4
t_1	1	0	0	1	0	0	1
f_1	1	0	0	0	1	1	0
t_2	0	1	0	1	0	1	0
f_2	0	1	0	0	1	0	1
t_3	0	0	1	1	1	0	1
f_3	0	0	1	0	0	1	0
x_1	0	0	0	1	0	0	0
y_1	0	0	0	1	0	0	0
x_2	0	0	0	0	1	0	0
y_2	0	0	0	0	1	0	0
x_3	0	0	0	0	0	1	0
y_3	0	0	0	0	0	1	0
x_4	0	0	0	0	0	0	1
y_4	0	0	0	0	0	0	1
s	1	1	1	3	3	3	3

The 3SAT formula has a satisfying assignment iff the generated subset sum problem has a solution.

$\Rightarrow$ 3SAT $\leq_p$ Subset Sum

Likewise, we can prove Subset Sum $\leq_p$ 0-1 Knapsack