RECAP OF LAST LECTURE

Computational model: $\boxed{\text{TURING MACHINE}}$

Q set of states

Σ finite-size alphabet (think ASCII/UTF-8)

Tapes: Read-only input tape

Read-write output & work tape

At each step

- Read symbols at current head positions
- Based on symbols & current state
  - write symbol on work tape
  - move tape heads (max 1 step)
  - go to new TM state

Use Turing machines to solve $\boxed{\text{DECISION PROBLEMS}}$

$$f : \Sigma^* \to \{0,1\}$$

0 = no
1 = yes

Ex Is there a path from s to t in graph G?
Is the propositional logic formula F satisfiable?

$\boxed{\text{Solve decision problem } f : \Sigma^* \to \{0,1\}} \Leftrightarrow \boxed{\text{DECIDE LANGUAGE } L = \{x \in \Sigma^* \mid f(x) = 1\}}$

$M(x) = 1$ "M accepts x"

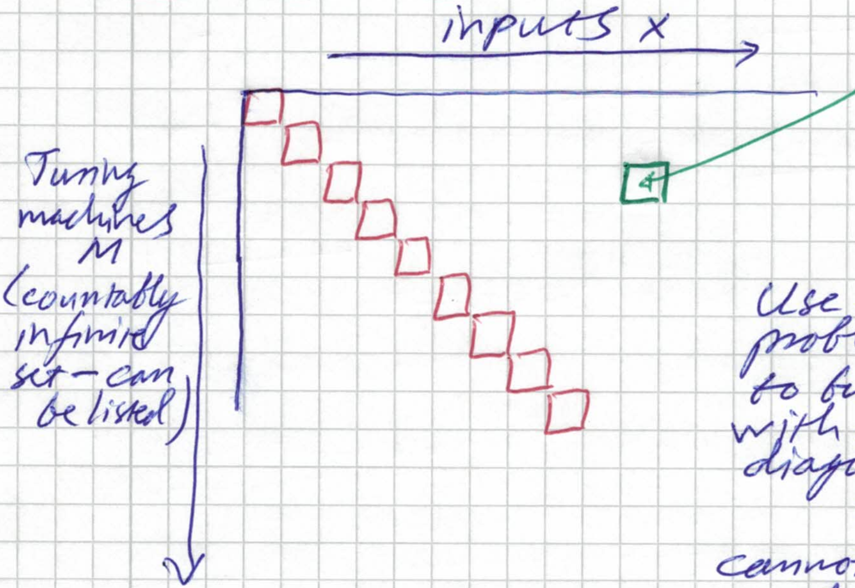
$M(x) = 0$ "M rejects x"

There is a $\boxed{\text{UNIVERSAL TURING MACHINE}}$ that can simulate any other TM efficiently given its description as a string

It is UNDECIDABLE whether a given Turing machine M halts on a given input x

DETOUR: What happened in the proof?

DIAGONALIZATION



Fill in whether M halts on x

Use assumption that halting problem can be decided to build TM that disagrees with all table cells on the diagonal.

⇓

cannot be one of the TMs in the table, but we are listing all of them ↯

Complexity class $\boxed{P}$

all languages L that can be decided in time $O(n^k)$ for constant k (depending on L)

EFFICIENT COMPUTATION

Defined as problems in P

Theoretical definition; can be debated

But on the whole very successful and fruitful definition

CHURCH-TURING THESIS [EXTENDED]

Anything computable by any computational device [efficiently] can be done by a Turing machine [efficiently]

Consistent with our experience so far

Extended version might break if/when quantum computers built.

REDUCTIONS

(polynomial time in $|x|$)

$L_1 \leq_p L_2$ if exists efficiently computable g such that

sometimes called KARP REDUCTION

$$x \in L_1 \implies g(x) \in L_2$$
$$x \notin L_2 \implies g(x) \notin L_2$$

L_1 is not harder than L_2 — use to solve problem

L_2 is not easier than L_1 — use to prove hardness

NP

Class of problems / languages with "efficiently verifiable solutions"

$L \in NP$ if exist
- polynomial p
- polynomial-time Turing machine $M(x,y)$

such that

$x \in L \iff$ Exist y ~~off~~ of length $\leq p(|x|)$ such that $M(x,y) = 1$

EXP

All languages L that can be decided in exponential time $O(2^{n^k})$ for some constant k (depending on L)

PROPOSITION 1 $P \subseteq NP \subseteq EXP$

Proof $P \subseteq NP$. Choose witness of length 0. Pick TM M that just solves the problem

$NP \subseteq EXP$ At most exponentially many witness candidates. Each can be checked in polynomial time

This simple proposition is state of the art (sadly)
One of the MILLENNIUM PRIZE PROBLEMS $P \stackrel{?}{=} NP$

Most researchers (but not all) believe $\boxed{P \neq NP}$

SOME EXAMPLE PROBLEMS AND WITNESSES

1. Is there a PATH from s to t in G?

2. Is given positive integer N COMPOSITE? (i.e., not prime)

3. Does integer N have PRIME FACTOR $\leq u$?

4. LINEAR PROGRAMMING

   m linear inequalities $a_1 \cdot u_1 + \ldots + a_n \cdot u_n \leq b$, $a_i, b \in \mathbb{Q}$. Is there an assignment to the u_i satisfying all inequalities?

5. 0-1 INTEGER LINEAR PROGRAMMING

   Same as (4), ~~by~~ but u_i have to be in $\{0, 1\}$

6. Is a given propositional logic formula SATISFIABLE?

   $(x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3)$

7. Is a given propositional logic formula a TAUTOLOGY?
   (i.e., always true)

   $(\neg x_1 \land \neg x_2 \land \neg x_3) \lor (x_1 \land x_2) \lor (x_1 \land x_3) \lor (x_2 \land x_3)$

8. Given integers $S = \{A_1, \ldots, A_n\}$ and target T is it possible to construct a SUBSET SUM $S' \subseteq S$ such that $\sum_{A_i \in S'} A_i = T$

   $\{2, 3, 5, 7\}$
   $T = 11$

(1) In NP Witness: vertices in path

In fact, in P Do, e.g., breadth first search

(2) In NP Witness: Factor f with $1 < f < N$

In fact, in P Efficient randomized algorithms known since [Miller '76] and [Rabin '80]

Poly-time algorithm (without randomness) in [AKS '04]

(3) In NP Witness: Prime factor

Not believed to be in P (RSA crypto would break)

But also not believed to be among hardest problems in NP

(4) In NP Witness: assignment

For long time, best LP algorithm had exponential worst-case complexity (Simplex) though very efficient in practice

Khachiyan '79: in P

Karmarkar '84: Practical algorithm

(5) In NP Witness: assignment

This problem is NP-complete — one of the hardest in NP

(6) In NP Witness: assignment

Also NP-complete

(7) ??? What would be a short witness?

Not believed to be in NP

But note that there are short counter-examples for non-members

(8) In NP Witness: subset S'

NP-complete

What does it mean to be "the hardest problem in NP"?

DEFINITION 2 (NP-HARD AND NP-COMPLETE)

The language $L \subseteq \Sigma^*$ is $\boxed{\text{NP-HARD}}$ if for every $L' \in NP$ it holds that $L' \leq_p L$ (i.e., there is a polynomial-time-computable function $g: \Sigma^* \to \Sigma^*$ such that

$$x \in L' \iff g(x) \in L$$

L is $\boxed{\text{NP-COMPLETE}}$ if in addition $L \in NP$

> L is as hard as any problem in NP, since any efficient algorithm for L can be used to decide any language in NP efficiently

LEMMA 3

1. If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$, then $L_1 \leq_p L_3$ (TRANSITIVITY)
2. If L is NP-hard and $L \in P$, then $P = NP$
3. If L is NP-complete, then $L \in P$ iff $P = NP$

Proof Exercise or see textbook

But do NP-complete problems exist?
Note that this is not clear from the definition...
But NP-complete problems do exist and turn out to be all over the place in mathematics, computer science, physics, chemistry, biology, economics, industry ...

> Learn more about all of this in course COMPUTABILITY AND COMPLEXITY (CoCo)

The most important ones (at least historically) are variants of $\boxed{\text{SATISFIABILITY}}$ (or SAT for short)

CNF formula (conjunctive normal form)

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

Variables x_i (or x, y, z) set to true = 1 or false = 0

Logical connectives AND $\wedge$, OR $\vee$, NOT $\neg$
(or sometimes write $\bar{x}$ for $\neg x$)

(Disjunctive) clause $C = x_1 \vee \neg x_2 \vee \neg x_3$
satisfied if one literal assigned to true

CNF formula conjunction of clauses
$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m = \bigwedge_{i=1}^{m} C_i$
satisfied if all clauses C_i are satisfied

$$\boxed{\text{SAT} = \{ F \mid F \text{ is a satisfiable CNF formula} \}}$$

k-CNF formula: Each clause has $\leq k$ literals

$$\boxed{k\text{-SAT} = \{ F \mid F \text{ is a satisfiable } k\text{-CNF formula} \}}$$

COOK-LEVIN THEOREM (1971 and 1973, respectively)
(1) SAT is an NP-complete problem
(2) 3-SAT is an NP-complete problem

Focus on (1) — (2) is easy corollary.

Clearly, SAT $\in NP$. A satisfying assignment to the variables is a short witness that is easy to verify

Need to show: For any $L \in NP$, exists efficient reduction g such that

$$x \in L \iff g(x) \text{ is a satisfiable CNF formula}$$

What can we do to prove this?

- Only thing we ~~know~~ is that exists Turing machine $M_L(x, y)$ such that

$$x \in L \iff \text{Exists } y \text{ of length} \leq p(|x|) \text{ such that } M_L(x, y) = 1$$

- Write computation of such TM M_L on x as a CNF formula
- Show that if $x \in L$, then can plug in witness y such that formula describing TM computation is satisfied

Think of alphabet Σ as $\{0, 1\}$ (can always re-encode symbols)

Think of input x as given.
Define Boolean function f_x by

$$\boxed{f_x^L(y) = \begin{cases} 1 & \text{if } M_L(x, y) = 1 \\ 0 & \text{otherwise} \end{cases}}$$

PROPOSITION 4

Any Boolean function $f : \{0,1\}^\ell \to \{0,1\}$ can be expressed as CNF formula of size $\leq \ell \cdot 2^\ell$

(size := total # literals in formula counted with repetitions)

Proof sketch

Consider assignment α s.t. $f(\alpha) = 0$

Write down clause C_α falsified exactly by this one assignment

Let the CNF formula be $F = \bigwedge_{\alpha \in f^{-1}(0)} C_\alpha$

EXAMPLE PARITY (x_1, x_2, x_3) = odd # variables true

Truth table

x_1	x_2	x_3	PARITY	Clauses
0	0	0	0	$(x_1 \vee x_2 \vee x_3)$
0	0	1	1	
0	1	0	1	
0	1	1	0	$\wedge \ (x_1 \vee \neg x_2 \vee \neg x_3)$
1	0	0	1	
1	0	1	0	$\wedge \ (\neg x_1 \vee x_2 \vee \neg x_3)$
1	1	0	0	$\wedge \ (\neg x_1 \vee \neg x_2 \vee x_3)$
1	1	1	1	

This CNF formula evaluates to true precisely when # true variables odd

<u>Proof attempt 1 for Cook-Levin Theorem:</u>

Given $L \in NP$, verifier $M_L(x,y)$, and input x

Consider $f_x^L(y)$

Use Proposition 4 to generate CNF formula F_x

F_x is satisfiable $\Longleftrightarrow$ Exists y s.t. $M_L(x,y) = 1$

This is our reduction g! QED ▨

OR IS IT? What is the size of F_x?

$p(|x|) \cdot 2^{p(|x|)}$ — exponential!

So g will run in exponential time — too slow!

Use that Turing machine computations are <u>LOCAL</u>

Only depend on current state and currently read symbols

Simplifying assumptions (but justified):

(1) For any $L \in NP$, fix polynomial p_L such that <u>witnesses</u> should have <u>length</u> <u>exactly</u> $p_L(|x|)$

(2) Turing machines have two tapes, input and work/output

(3) Turing machine is $\boxed{\text{OBLIVIOUS}}$: <u>head movements do not depend on tape contents</u>, only on input length (which is $|x| + |y|$)

Can run M_L on x and $0^{p(|x|)} = \overbrace{000\ldots0}^{p(|x|)}$

to determine head positions at every time step and build table

~~At most~~ quadratic loss in running time

(See Arora-Barak Ch 1 for details)

Notation

Q : Turing machine states (= "lines in program")

Σ : Alphabet (containing $0, 1, \sqcup$ = blank etc)

u : Input x and y concatenated

All symbols in Σ can be encoded in binary

If $|\Sigma| = s$, need $\lceil \log_2 s \rceil$ bits

$\boxed{\text{SNAPSHOT}}$ $z = \langle a, b, q \rangle \in \Sigma \times \Sigma \times Q$

captures what determines Turing machine action at given time step

- a: symbol read on input tape
- b: symbol read on work/output tape
- q: current TM state

Since Q is also finite, snapshot z can be encoded as binary string of fixed length, say c bits [choose $c \geq \lceil \log_2 s \rceil$]

Snapshot z_i at time i depends on:

(a) state at time $i-1$

(b) contents at current locations of tapes

Suppose we're given sequence of snapshots $z_1, z_2, z_3, \ldots, z_t$ claimed to describe computation by M_L. How to verify?

> INSIGHT: We can verify each z_i by only local checks

To check $z_i = \langle a_i, b_i, q_i \rangle$, only need to look at

(1) $z_{i-1} = \langle a_{i-1}, b_{i-1}, q_{i-1} \rangle$ — tells us if jump to state q_i correct

(2) $u_{inputpos(i)}$ — The input tape head is at position $inputpos(i)$ at time i (which we have computed in a table), so check $u_{inputpos(i)} = a_i$

(3) $z_{prev(i)}$ — Time $prev(i)$ is the last time the work tape head was at its current position (which we have also computed in a table), so check that symbol <u>written</u> to output tape at time $prev(i)$ as specified by $z_{prev(i)} = \langle a_{prev(i)}, b_{prev(i)}, q_{prev(i)} \rangle$ is the same as b_i.

$\boxed{\text{(1) – (3) \underline{uniquely} determine } z_i}$

So there is a Boolean function

<u>$step_i(z_i, z_{i-1}, u_{inputpos(i)}, z_{prev(i)})$</u>

that <u>evaluates to true precisely when transition at time step i correct</u>

Function of $\leq 4c$ bits = constant

Apply Proposition 4 $\Rightarrow$ constant-size formula

Running time of $M_L(x,y)$ is **exactly** $q(|x|+|y|)$ for some polynomial q, which is $q^*(|x|)$ for some other polynomial

$$q^*(|x|) = q(|x| + p_L|x|)$$

Let our reduction write down CNF formula F_x as follows

1. subformula $INPUT(x)$ saying that first $|x|$ symbols on input tape must match x.

2. subformula $START(z_1)$ encoding that the starting position of M_L is correct

3. subformulas $STEP(i)$ for $i = 2, 3, \ldots, q^*(|x|)$ saying that snapshot z_i is correct given z_{i-1}, $u_{inputpos(i)}$, and $z_{prev(i)}$ (where we can look up $inputpos(i)$ and $prev(i)$ in tables)

4. subformula ACCEPT saying that the final state is that of an accepting computation of M_L (e.g., 1 is written in first position of work tape and all other positions blank)

$$\boxed{F_x = INPUT(x) \wedge START(z_1) \wedge \bigwedge_{i=2}^{q^*(|x|)} STEP(i) \wedge ACCEPT}$$

Subformulas 1, 2, 4 easy — just fixing bits to values

"$v = w$" $(\neg v \vee w) \wedge (v \vee \neg w)$

Subformula 3

size $q^*(|x|) \cdot \overbrace{(\text{exponential in } c)}^{\text{constant}}$

Can be computed in polynomial time

- First run $M_L(x, 0^{p_L(|x|)})$ to compute tables "inputpos" and "prev"
- Then output CNF formula F_x, where subformula 3 is what takes time

F_x is satisfiable if and only if exists y such that $M_L(x,y) = 1$, i.e., if and only if $x \in L$ QED (for real) $\blacksquare$

Reducing from SAT to 3-SAT is straightforward exercise (or see textbook)

<u>Two observations</u>

1. If M_L runs in time $T(|x|)$, formula F_x can be made very small — $O(T \log T)$
2. From satisfying assignment to F_x, can read off witness y for x

This is called a LEVIN REDUCTION

To prove that a language L is NP-complete, we need to do two things

(i) Show $L \in NP$ (usually easy)

(ii) Reduce from SAT or 3-SAT (or from some other already known NP-complete problem) to L

We will now see some such reductions. But first one more definition

DEFINITION 5: COMPLEMENT CLASS

For a language $L \subseteq \Sigma^*$, the COMPLEMENT of L is $\overline{L} = \Sigma^* \setminus L$

DEFINITION 6: coNP

$$\boxed{coNP = \{ L \mid \overline{L} \in NP \}}$$

Aside: If strings in L encode objects such as formulas or graphs, then we usually think of $\overline{L}$ as only containing correctly encoded instances.
That is, L and $\overline{L}$ will both be sets of formulas or graphs satisfying or not satisfying some property, respectively, while "syntax error" strings are not contained in either L or $\overline{L}$.
This is just a technical convention that doesn't really matter much

Note that coNP is not the complement of NP — the intersection is non-empty! E.g.,

$$P \subseteq NP \cap coNP \quad \text{(why?)}$$

EXAMPLE $\boxed{\text{TAUTOLOGY}}$ (example 7 above) is in coNP. If F is not a tautology, then this is witnessed by an assignment falsifying the formula

$\boxed{\text{UNSAT}} = \{ F \mid F \text{ is an unsatisfiable CNF formula} \}$ is also in coNP

In fact, both of these languages are coNP-complete — any other language $L \in coNP$ can be efficiently reduced to them

Proof sketch for UNSAT: Given $L \in coNP$, run Cook-Levin reduction on $\overline{L}$

$$x \in L \iff x \notin \overline{L} \iff F_x \notin SAT$$
$$\iff F_x \in UNSAT$$

How is coNP related to NP?

Could there be short certificates for UNSAT that somehow "compress information about exponentially many assignments"?

Most researchers believe $\boxed{NP \neq coNP}$ but this is a wide-open question!

$$\boxed{\text{CLIQUE} = \left\{ \langle G,k \rangle \;\middle|\; \begin{array}{l} G \text{ is an undirected graph} \\ \text{containing a } k\text{-clique} \end{array} \right\}}$$

A $\boxed{k\text{-clique}}$ in G is a set of <u>k pairwise connected vertices</u> $v_1, v_2, \ldots, v_k \in V(G)$
That is, all edges (v_i, v_j), $1 \le i < j \le k$ are in $E(G)$

$$\boxed{\underline{\text{THEOREM}} \quad \text{CLIQUE is } NP\text{-complete}}$$

We will show
(i) $\text{CLIQUE} \in NP$
(ii) $3\text{-SAT} \le_p \text{CLIQUE}$ (CLIQUE is NP-hard)

For (i), let witness be the k vertices in a clique
Verifier checks that all edges between these vertices are in graph.
Can clearly be done in polynomial time on reasonable computer with reasonable programming language (which can be efficiently simulated by Turing machine).

For (ii), given a 3-CNF formula F, we need to <u>construct</u> a <u>graph G_F</u> and choose a <u>parameter</u> <u>k_F</u> such that

$$\boxed{F \text{ satisfiable} \iff G_F \text{ has } k_F\text{-clique}}$$

And construction should be <u>computable</u> in <u>polynomial time</u> (in size of F)

Construction of G_F from $F = \bigwedge_{i=1}^{m} C_i$

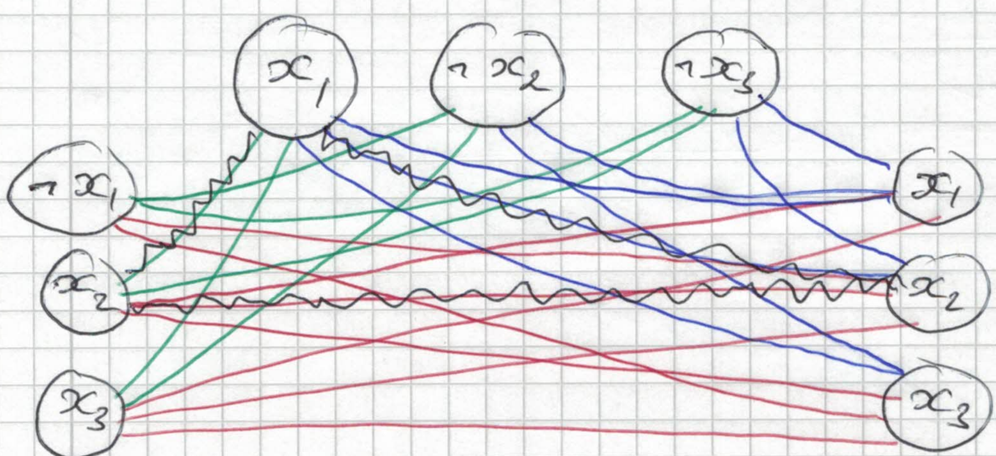
For every clause $C_r = l_{r,1} \vee l_{r,2} \vee l_{r,3}$
create 3 vertices $v_{r,1}, v_{r,2}, v_{r,3}$

Add edge $(v_{r,i}, v_{s,j})$ iff
a) $r \neq s$ (vertices come from different clauses)
b) $l_{r,i}$ and $l_{s,j}$ are <u>not</u> negations of each other

Set $k_F = m = \#$ clauses in F

<u>Example</u>

$$\boxed{\begin{aligned} F = \; & (x_1 \vee \neg x_2 \vee \neg x_3) & C_1 \\ \wedge\; & (\neg x_1 \vee x_2 \vee x_3) & C_2 \\ \wedge\; & (x_1 \vee x_2 \vee x_3) & C_3 \end{aligned}}$$



Clearly possible to build G_F & k_F from F in polynomial time

Need to show
($\Rightarrow$) F satisfiable $\Longrightarrow$ G_F has k_F-clique
($\Leftarrow$) F satisfiable $\Longleftarrow$ G_F has k_F-clique

($\Rightarrow$) Let α be satisfying assignment
α satisfies at least one literal $l_{r,i}$ per clause C_r
Pick one such literal per clause.
Since literals are all true, none is negation of other, so clique
One vertex per clause $\Rightarrow$ k_F - clique

Example $\alpha = \{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 0\}$

Pick x_1 from C_1, x_2 from C_2, x_2 from C_3

($\Leftarrow$) Suppose G_F has k_F - clique.
Then at least one (precisely one) vertex $v_{r,i}$ per clause C_r
All literals $l_{r,i}$ can be assigned true, since there are edges between them.
Yields partial truth value assignment satisfying all clauses.
(Assign any remaining variables arbitrarily)

Vertex cover of $G=(V,E)$ is subset $V' \subseteq V$ such that every edge $(u,v) \in E$ has an endpoint in V' (i.e., $\{u,v\} \cap V' \neq \emptyset$)

$$\boxed{\text{VERTEXCOVER} = \{\langle G, k\rangle \mid G \text{ has vertex cover of size } k\}}$$

THEOREM VERTEXCOVER is NP-complete

We will show

(i) VERTEXCOVER $\in$ NP

(ii) CLIQUE $\leq_p$ VERTEXCOVER (VERTEXCOVER is NP-hard)

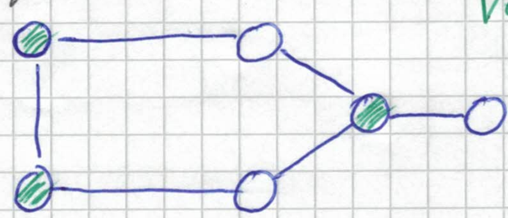
Already known to be NP-complete, so can reduce from it!

For (i), suitable witness consists of
$V' = (v_1, v_2, \ldots, v_k)$
Verifier checks for all edges (u,v) that
$\{u,v\} \cap \{v_1, \ldots, v_k\} \neq \emptyset$
Can clearly be done in polynomial time

Example

Vertex cover of size 3

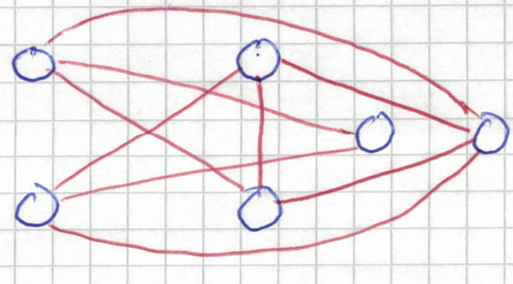
G



For (ii), translate $\langle G, k\rangle$ to $\langle \overline{G}, |V| - k\rangle$
where $\boxed{\overline{G}}$ is complement graph with same vertex set but edges $(V \times V) \setminus E$

(u,v) is edge in $\overline{G} \iff (u,v)$ is not edge in G

$\overline{G}$



$\overline{G}$ can clearly be constructed from G in polynomial time

Need to show:

$$\langle G, k \rangle \in \text{CLIQUE} \iff \langle \overline{G}, |V|-k \rangle \in \text{VERTEXCOVER}$$

($\Rightarrow$) Suppose $C \subseteq V$ is clique in G of size $|C| = k$

$u, v \in C$, $u \neq v$ $\Rightarrow (u,v)$ edge in G

$\Rightarrow (u,v)$ not edge in $\overline{G}$

So C contains no edges in $\overline{G}$

$\Rightarrow$ all edges in $\overline{G}$ have an endpoint in $V \setminus C$, which is a vertex cover of size $|V|-k$.

($\Leftarrow$) Suppose C^* vertex cover in $\overline{G}$

Then $V \setminus C^*$ contains no edges in $\overline{G}$

Means that all edges in $V \setminus C^*$ are present in G

Hence if C^* vertex cover in $\overline{G}$ of size $|V|-k$, then $V \setminus C^*$ clique in G of size $|V| - (|V|-k) = k$.

Subset sum

think of subset S' as indices of numbers

Given positive integers $A_1, A_2, \ldots, A_n, T$

Is there subset $S' \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S'} A_i = T \quad ?$$

$$\boxed{\text{SUBSETSUM} = \left\{ \langle A_1, \ldots, A_n, T \rangle \;\middle|\; \exists S' \text{ s.t. } \sum_{i \in S'} A_i = T \right\}}$$

$\boxed{\text{THEOREM} \quad \text{SUBSETSUM is } NP\text{-complete.}}$

We will show

(i) SUBSETSUM $\in NP$

(ii) 3-SAT $\leq_p$ SUBSETSUM (SUBSETSUM is NP-hard)

For (i), let witness be set of indices S'.

Verifier computes $\sum_{i \in S'} A_i$ and checks if this sum equals T

Can clearly be done in polynomial time

For (ii), suppose we have 3-CNF formula $F = \bigwedge_{i=1}^{m} C_i$ over variables $x_1, \ldots, x_n$

Create subset sum instance

$$\{ A_1^T, A_1^F, A_2^T, A_2^F, \ldots, A_n^T, A_n^F, B_1^1, B_1^2, \ldots, B_m^1, B_m^2 \}$$

and T as follows

- Each number has $n + m$ decimal digits
- n most significant digits associated with $x_1, \ldots, x_n$
- m least significant digits associated with $C_1, \ldots, C_m$

Assume w.l.o.g.

- No clause contains both a variable and its negation (if so, remove this clause)
- All variable appear in F (remove & renumber otherwise)

Construction of numbers

A_i^T: digit x_i is 1
digit C_j is 1 if $x_i \in C_j$
all other digits 0

A_i^F: digit x_i is 1
digit C_j is 1 if $\neg x_i \in C_j$
all other digits 0

B_j^1 digit C_j is 1
all other digits 0

B_j^2 digit C_j is 2
all other digits 0

T all digits x_i are 1
all digits C_j are 4

Clearly possible to construct such a subset sum instance in polynomial time in the size of F

$(\Rightarrow)$
Suppose F satisfied by assignment α
- If $\alpha(x_i) = 1$ pick A_i^T, otherwise A_i^F
- If α satisfies
  1 literal in C_j — pick B_j^1 and B_j^2
  2 literals in C_j — pick B_j^2
  3 literals in C_j — pick B_j^1

This sums to T

$(\Leftarrow)$
Given solution S' to subset sum instance
Let α set $x_i = 1$ iff A_i^T chosen in S'

Why is this a satisfying assignment?

Example

$$
\begin{aligned}
F = \; & (x_1 \vee \neg x_2 \vee \neg x_3) && C_1 \\
\wedge \; & (\neg x_1 \vee x_2 \vee x_3) && C_2 \\
\wedge \; & (x_1 \vee x_2 \vee x_3) && C_3
\end{aligned}
$$

Solution corresponding to $x_1 \mapsto 1,\ x_2 \mapsto 1,\ x_3 \mapsto 0$

$n = 3$

$m = 3$

A_1^T	=	100101	*	
A_1^F	=	100010		
A_2^T	=	010011	*	
A_2^F	=	010100		
A_3^T	=	001011		
A_3^F	=	001100	*	
B_1^1	=	000100		
B_1^2	=	000200	*	C_1
B_2^1	=	000010	*	C_2
B_2^2	=	000020	*	
B_3^1	=	000001		
B_3^2	=	000002	*	C_3
T	=	111444		

First n digits of T enforce that exactly one of x_i and $\neg x_i$ chosen

For last m digits, can only reach 4 if at least one literal in clause is chosen to be true.