

Lecture 2: NP-completeness

Mika Göös



School of Computer and Communication Sciences

Lecture 2

Recap: NP

The class NP

Definition: A **verifier** for a language A is a TM V , where

$$A = \{x \mid \exists C \text{ s.t. } V \text{ accepts } \langle x, C \rangle\}.$$

A **polynomial time verifier** runs in polynomial time in $|x|$.

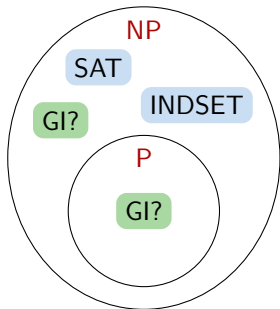
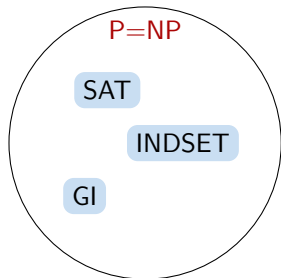
Definition: **NP** is the class of languages with **polynomial time verifiers**

P and NP

Is $P \subseteq NP$? Yes (obviously)

Is $P = NP$? Nobody knows ...

Find the answer and win USD 1,000,000!



Cook-Levin Theorem (informal): $SAT \in P$ iff $P = NP$.

(Also $INDSET \in P$ iff $P = NP$.)

Why is it called NP?

Non-deterministic Turing Machines

Recall: In a Turing machine, $\delta : (Q \times \Gamma) \longrightarrow Q \times \Gamma \times \{L, R\}$.

In a **Nondeterministic Turing Machine (NTM)**,

$$\delta : (Q \times \Gamma) \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

(several possible transitions for a given state and tape symbol)

Definition: A **nondeterministic decider** for language L is an NTM N such that for each $x \in \Sigma^*$, **every computation of N on x halts**, and moreover,

- ▶ If $x \in L$, then **some** computation of N on x **accepts**.
- ▶ If $x \notin L$, then **every** computation of N on x **rejects**.

An NTM is a **polynomial time** NTM if the running time its **longest** computation on x is **polynomial in $|x|$** .

Nondeterministic deciders \iff Verifiers

Theorem: For any language $L \subseteq \Sigma^*$,
 L has a nondeterministic poly-time decider $\iff L$ has a poly-time verifier.

Proof Sketch (\Leftarrow):

Let V be the verifier. NTM N on input x does the following:

- 1 Write a certificate C nondeterministically.
- 2 Run V on $\langle x, C \rangle$.

Proof Sketch (\Rightarrow):

Let N be the nondeterministic decider. Verifier V on $\langle x, C \rangle$ computes:

- ▶ Simulate N on x , choosing transitions given by C .

V accepts $\langle x, C \rangle$ iff C is the accepting path of N on x .

Non-deterministic Polynomial-time

Theorem: For any language $L \subseteq \Sigma^*$,

L has a nondeterministic poly-time decider $\iff L$ has a poly-time verifier.

Definition: **NP** is the class of languages which have poly-time nondeterministic deciders, or equivalently, have poly-time verifiers.

Definition:

$\text{NTIME}(t(n)) = \{L : L \text{ has a nondeterministic } O(t(n)) \text{ time decider}\}$

Then

$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k).$$

\therefore SAT, GI, INDSET are all in NP.

Reductions and NP-completeness

Reductions

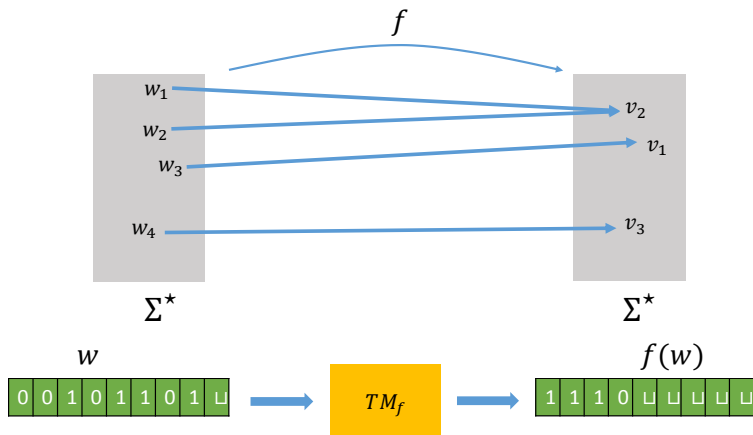
Reducibility Use knowledge about complexity of one language to reason about the complexity of other in an **easy** way

Reduction Way to show that to solve one problem (A), it is sufficient to solve another problem (B)

A reduces to B...

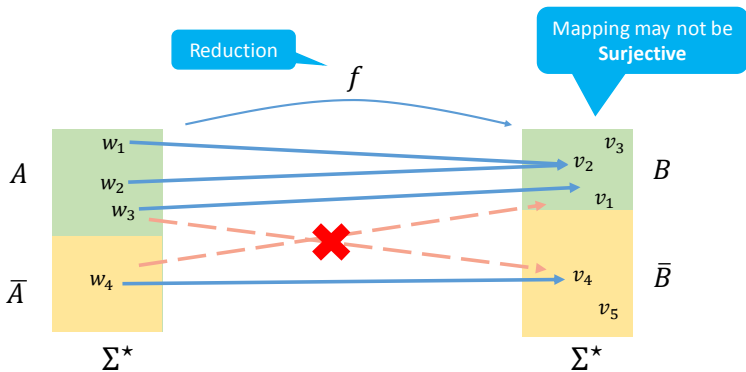
- ▶ Reductions quantify **relative** hardness of problems
 - ▶ If problem B is **easy** then problem A is **easy** too
 - ▶ If problem A is **hard** then problem B is **hard** too

Poly-time Reductions (1): Poly-time Computability



Definition: A function $f : \Sigma^* \rightarrow \Sigma^*$ is *polynomial time computable* if there is some *poly-time* TM that on *every* input w , outputs $f(w)$

Poly-time Reductions (2): Correctness



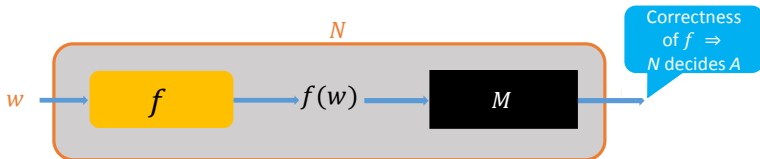
Definition: Language A is **poly-time reducible** to language B , written $A \leq_p B$, if there is a **poly-time** computable function $f : \Sigma^* \rightarrow \Sigma^*$, such that for **every** $w \in \Sigma^*$:

$$w \in A \Leftrightarrow f(w) \in B$$

Theorem: If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$

Proof:

- ▶ Let M be a poly-time algorithm for B and let f be a poly-time reduction from A to B
- ▶ Define algorithm N as follows:



- ▶ $N =$ "On input w :
 - 1 Compute $f(w)$
 - 2 Run M on input $f(w)$ and output whatever M outputs"

Corollary: If $A \leq_P B$ and $A \notin \mathbf{P}$, then $B \notin \mathbf{P}$

Definition: A language L is said to be **NP-complete** if

- ▶ L is in **NP**.
- ▶ For every language L' in **NP**, $L' \leq_P L$.

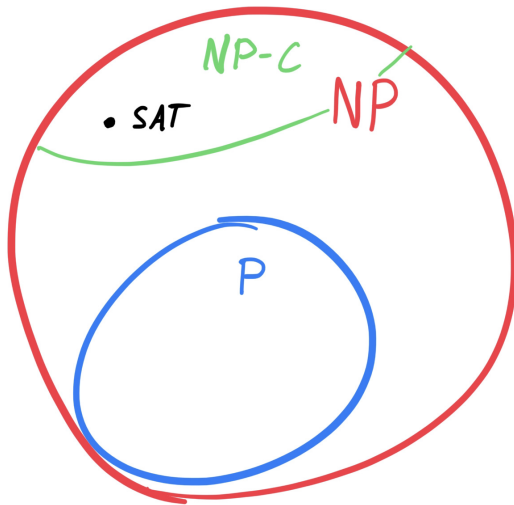
Observe: If **one NP-complete** language has a polynomial time decider, then **every** language in **NP** has a polynomial time decider, i.e. **P = NP**.

The Cook–Levin Theorem: SAT is **NP-complete**.

To show L is **NP-complete**:

- ▶ **[NP membership]** Give a poly-time verifier for L .
- ▶ **[NP hardness]** Show $C \leq_P L$ for some **NP-complete** language C
(not the other way around)

Big picture



Takehome message



“I can’t find an efficient algorithm, but neither can all these famous people.”

3SAT is NP-complete

$k\text{SAT} = \{\langle \varphi \rangle : \varphi \text{ is satisfiable and each clause of } \varphi \text{ contains } \leq k \text{ literals}\}$

Verifier for 3SAT: Just use the verifier for SAT.

Claim: $\text{SAT} \leq_P 3\text{SAT}$

Reduction: Given φ ,

- ▶ While φ contains a clause $K = (l_1 \vee l_2 \vee l_3 \vee \dots \vee l_m)$ with > 3 literals

Replace K with the following two clauses

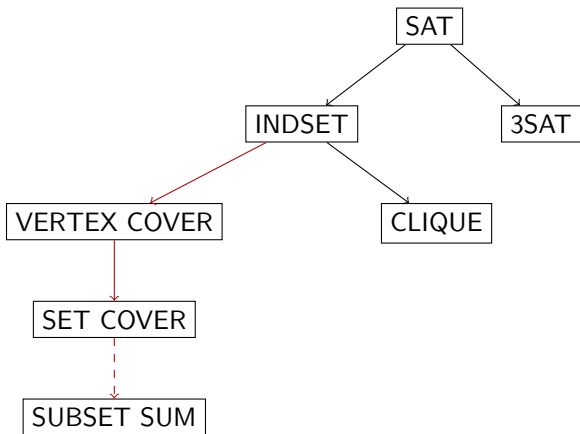
$$K_1 = (l_1 \vee l_2 \vee z)$$

$$K_2 = (\bar{z} \vee l_3 \vee \dots \vee l_m)$$

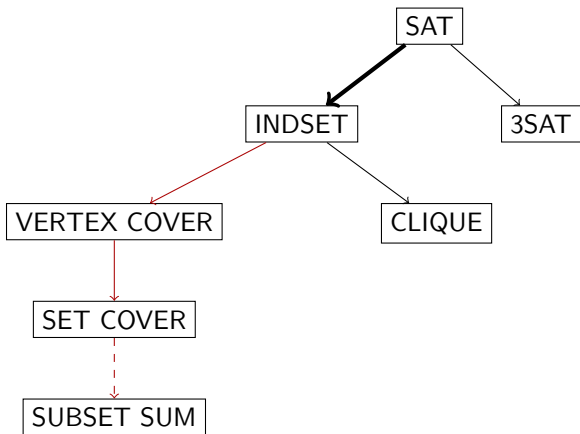
} Preserves satisfiability
(check!)

What is the runtime?

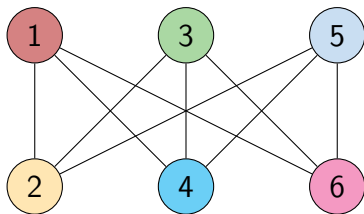
Does $\text{SAT} \leq_P 2\text{SAT}$ analogously?



and many more ...



INDSET is NP-complete



We have already seen that

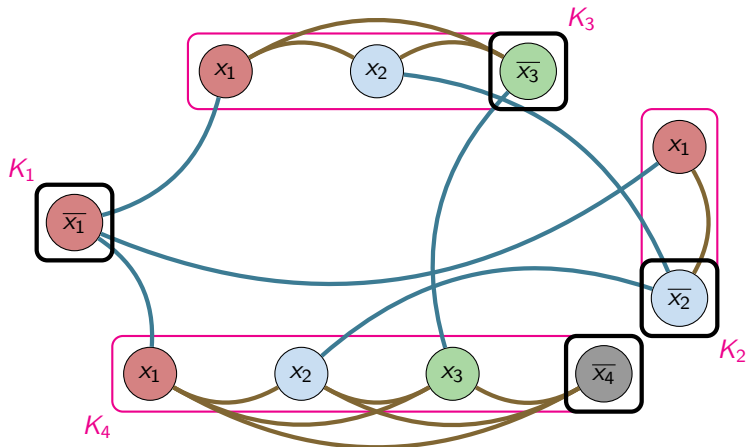
$$\text{INDSET} = \{ \langle G, k \rangle : G \text{ has an independent set of size } k \}$$

is in **NP** and so it remains to give a poly-time reduction from an **NP**-complete language

INDSET is NP-complete

Claim: $\text{SAT} \leq_P \text{INDSET}$

$$\varphi = \underbrace{\bar{x}_1}_{K_1} \wedge \underbrace{(x_1 \vee \bar{x}_2)}_{K_2} \wedge \underbrace{(x_1 \vee x_2 \vee \bar{x}_3)}_{K_3} \wedge \underbrace{(x_1 \vee x_2 \vee x_3 \vee \bar{x}_4)}_{K_4}$$



INDSET is NP-complete

Claim: $SAT \leq_P INDSET$

Reduction f : On input φ ,

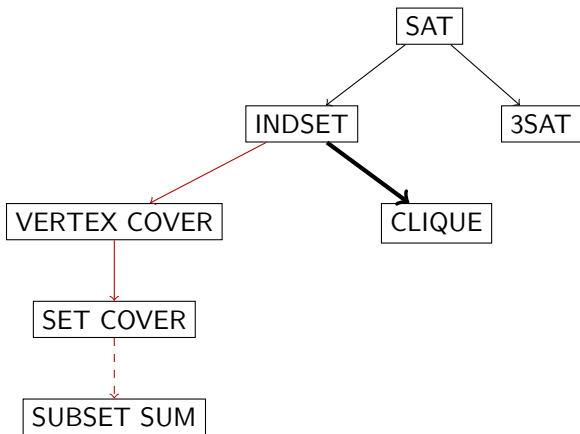
- 1 Let G be the graph generated as follows.
 - 1 Take a vertex for each literal of each clause.
 - 2 Add edges for pairs of conflicting literals.
 - 3 Add edges for pairs of literals from the same clause.
- 2 Let m be the number of clauses in φ .
- 3 Output (G, m) .

Claim: $\varphi \in SAT \implies f(\varphi) \in INDSET$

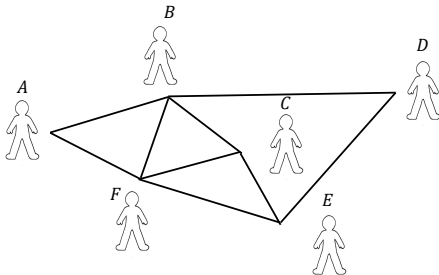
Proof: C : satisfying assignment of φ . Pick one true literal from each clause. The corresponding vertices form an independent set.

Claim: $f(\varphi) \in INDSET \implies \varphi \in SAT$

Proof: C : independent set in G , $|C| = m$. C contains one vertex from each group. Set the corresponding literals to true to get a satisfying assignment.



CLIQUE



A, B, F is a 3-clique

Definition: A k -clique is a subset of k pairwise connected vertices

$$\text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ has a clique of size } k \}$$

$\langle G, 3 \rangle \in \text{CLIQUE}$? Yes

$\langle G, 4 \rangle \in \text{CLIQUE}$? No

CLIQUE is NP-Complete

Theorem: $\text{INDSET} \leq_p \text{CLIQUE}$

Corollary: CLIQUE is **NP**-complete

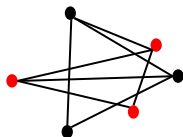
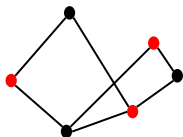
Goal find a poly-time reduction from INDSET to CLIQUE

INDSET vs CLIQUE

For a graph G :

Independent set: A subset of pairwise **non-adjacent** vertices

Clique: A subset of pairwise **adjacent** vertices



Def (Complement): The *complement* of a graph $G = (V, E)$ is a graph $\bar{G} = (V, \bar{E})$ with same vertex set and edge set \bar{E} s.t. $uv \in \bar{E}$ iff $uv \notin E$

Observation: If G is a graph and \bar{G} its complement, then a subset S of the vertices of G is an independent set iff S is a clique of \bar{G}

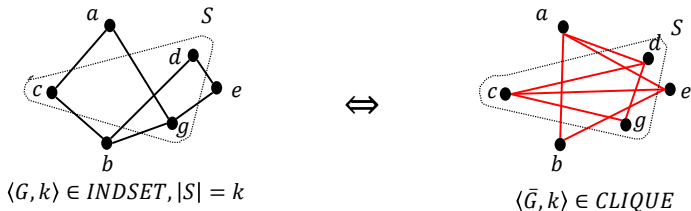
INDSET \leq_p CLIQUE

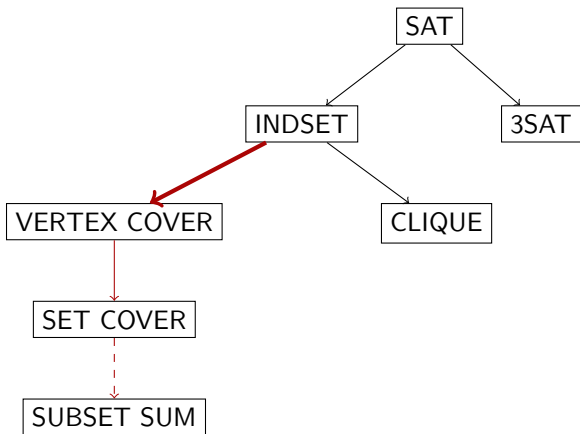
Reduction: $f(\langle G = (V, E), k \rangle) := \langle \bar{G} = (V, \bar{E}), k \rangle$

Efficiency: The reduction is polynomial time

Correctness: $\langle G, k \rangle \in \text{INDSET} \Leftrightarrow \langle \bar{G}, k \rangle \in \text{CLIQUE}$

Proof of correctness:

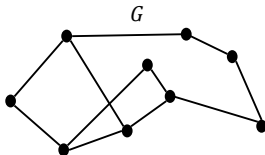




Vertex Cover

Definition: For a graph $G = (V, E)$, a *vertex cover* is a subset S of V such that every edge of G is incident to at least one vertex in S

Example



Definition:

VERTEX COVER = $\{ \langle G, k \rangle : G \text{ is a graph that has a vertex cover of size } k \}$

- ▶ $\langle G, 4 \rangle \in \text{VERTEX COVER? Yes}$
- ▶ $\langle G, 3 \rangle \in \text{VERTEX COVER? No}$

VERTEX COVER is NP-Complete

STEP 1: VERTEX COVER \in NP

TM V : "On input $\langle G, k, C \rangle$

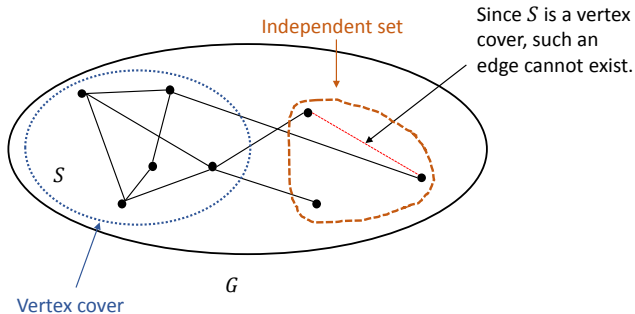
- 1 IF $|C| \neq k$, THEN REJECT
- 2 FOR every pair $u \neq v$ of vertices DO
 - ▶ IF uv is an edge AND $u \notin C$ AND $v \notin C$ THEN REJECT
- 3 ACCEPT"

STEP 2: INDSET \leq_p VERTEX COVER

How to reduce INDSET to VERTEX COVER?

INDSET vs VERTEX COVER

Lemma: For every graph G , a subset S of the vertices is a vertex cover if and only if \bar{S} is an independent set where $\bar{S} = V \setminus S$



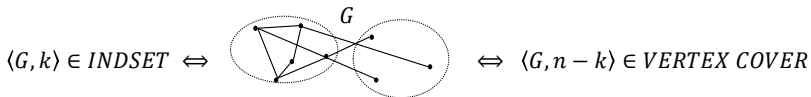
Corollary: For every graph G and a positive integer k , G has an independent set of size k iff G has a vertex cover of size $n - k$

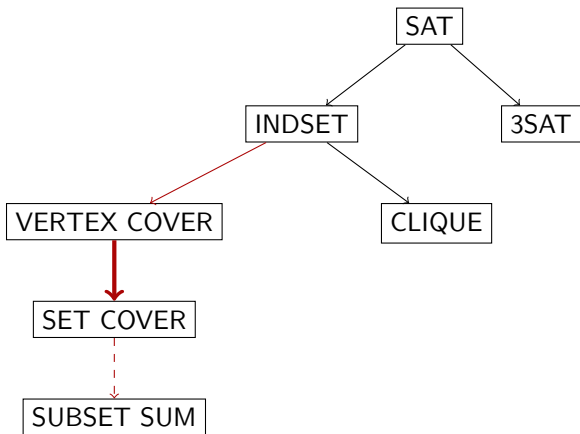
INDSET \leq_p VERTEX COVER

Reduction: $f(\langle G, k \rangle) := \langle G, n - k \rangle$, where n is the number of vertices of G

Efficiency: The reduction f is polynomial time

Correctness: Lemma/Corollary on previous slide!

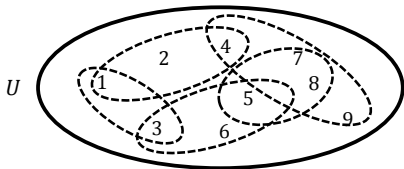




Set Cover

Definition: Let $U = \{1, \dots, n\}$ and $\mathcal{F} = \{T_1, \dots, T_m\}$ be a family of subsets $\forall i, T_i \subseteq U$

A subset $\{T_{i_1}, \dots, T_{i_k}\} \subseteq \mathcal{F}$ is called a **set cover** of size k if $\bigcup_{j=1}^k T_{i_j} = U$



SET COVER = $\{\langle U, \mathcal{F}, k \rangle \mid \mathcal{F} \text{ contains a set cover of size } k\}$

Example:

- ▶ $\langle U, \mathcal{F}, 3 \rangle \in \text{SET COVER}$? Yes $\{\{1, 2, 4\}, \{3, 6, 4\}, \{4, 7, 8, 9\}\}$
- ▶ $\langle U, \mathcal{F}, 2 \rangle \in \text{SET COVER}$? No

SET COVER is **NP**-Complete

Theorem: VERTEX COVER \leq_p SET COVER

Corollary: SET COVER is **NP**-complete

VERTEX COVER \leq_p SET COVER

Key idea: VERTEX COVER is a *special case* of SET COVER



Reduction:

- ▶ Let $\langle G = (V, E), k \rangle$ be an instance of VERTEX COVER
- ▶ Set $U := E$
- ▶ For every vertex $v \in V$ create a set S_v

$$S_v := \{e \in E \mid e \text{ is incident to } v\}$$

- ▶ Let $\mathcal{F} := \{S_v \mid v \in V\}$ and set $k' = k$

Efficiency: Obvious from construction

Correctness (\Rightarrow): If G has a vertex cover of cardinality k , then U can be covered by k sets

Proof:

- ▶ Suppose $C \subseteq V$ is a vertex cover of G and $|C| = k$
- ▶ Every edge e_i is adjacent to at least one vertex in C

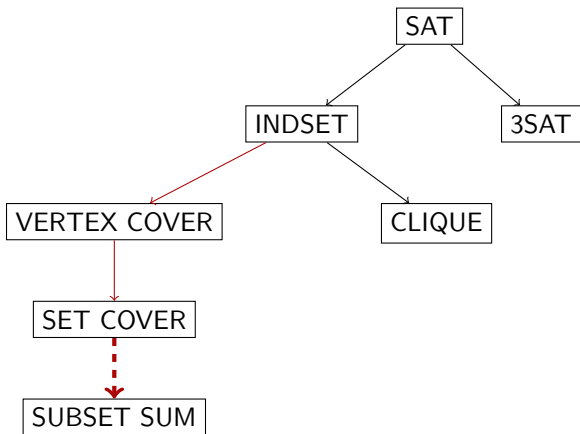
$$\bigcup_{v \in C} S_v = E$$

- ▶ Hence U can be covered by k sets

Correctness (\Leftarrow): If U can be covered by k sets, then G has a vertex cover of cardinality k

Proof:

- ▶ Let $S_{v_1}, S_{v_2}, \dots, S_{v_k}$ be a collection of sets which cover $U = E$
- ▶ We claim that $C = \{v_1, v_2, \dots, v_k\}$ is a vertex cover of G
- ▶ Indeed, every edge e in G belongs to S_{v_i} for some $i \in \{1, 2, \dots, k\}$
- ▶ Hence, every edge e in G is incident to some vertex $v_i \in C$



SUBSET-SUM

Let X denote a (multi) set of positive integers

Definition: SUBSET-SUM

$= \{ \langle X, s \rangle : X \text{ contains a subset whose elements sum to } s \}$

Example: $X = \{1, 3, 4, 6, 13, 13\}$

- ▶ $\langle X, 8 \rangle \in \text{SUBSET-SUM?}$ Yes $T = \{1, 3, 4\}$
- ▶ $\langle X, 12 \rangle \in \text{SUBSET-SUM?}$ No

Question: SUBSET-SUM $\in P$?

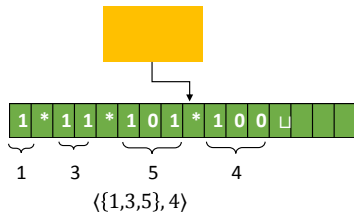
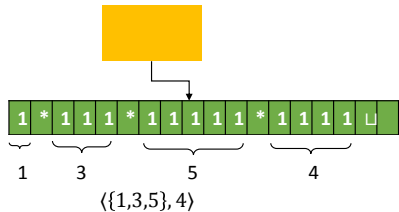
It depends on the **input length**

In *SUBSET-SUM*
we use binary
encoding

Unary

vs

Binary



Input length: $x_1 + x_2 + \dots + x_m + s$

>>

Input length: $\lceil \log(x_1) \rceil + \dots + \lceil \log(x_m) \rceil + \lceil \log(s) \rceil$

SUBSET-SUM: An Algorithm

- Let $X = \{x_1, x_2, \dots, x_m\}$, $sum := \sum_{j=1}^m x_j$
- FOR $i = 1 \dots m$ DO
- FOR $j = 0 \dots sum$ DO
- $A[i, j] := False$
- FOR $i = 0$ to m DO
- $A[i, 0] := True$
- FOR $i = 1 \dots m$ DO
- FOR $j = 1 \dots sum$ DO
- IF $A[i - 1, j - x_i] = True$ OR $A[i - 1, j] = True$ THEN $A[i, j] := True$
- IF $A[m, s] = True$, ACCEPT; ELSE, REJECT

SUBSET-SUM is **NP**-Complete

Theorem: SUBSET-SUM is **NP**-Complete

Proof: See bonus material on Moodle