Complexity classes : P & NP.

Think of computational tasks as language recognition problem.

Eg. $\mathcal{L}_{CONN} = \{ x \in \{0,1\}^* : x$ is a connected graph $\}$

Given language $\mathcal{L}$, string x, is $x \in \mathcal{L}$?

Algo A solves this if : yes if $x \in \mathcal{L}$, no o.w.

Time-complexity of A : $t_A(n)$ = max. running time of A on any string x of length n.

Language $\mathcal{L}$ has a poly-time algorithm if $\exists$ A that solves / decides $\mathcal{L}$, and $t_A(n) = O(n^k)$ for some constant k. (independent of n)

Then $P = \{ \mathcal{L} : \exists$ a poly-time algo that decides $\mathcal{L} \}$

Eg. : $\mathcal{L}_{conn}$, $\mathcal{L}_{2COL}$, $\mathcal{L}_{2SAT}$, $\mathcal{L}_{Primes}$

Now consider the problems: is G 3-colorable ?

is G Hamiltonian ?

is ϕ satisfiable ?

These have a short certificate of membership.

i.e., $\exists$ an algo $A(x,y)$ that runs in poly-time, and:

$\forall x \in \mathcal{L} \ \exists y : |y| = \text{poly}(|x|)$ and $A(x,y) = \text{yes}$

$\forall x \notin \mathcal{L}, \ \forall y \qquad A(x,y) = \text{no}$

(what are certificates for above 3 languages ?)

$NP = \{ \mathcal{L} : \exists$ a poly-time algo $A(x,y)$ s.t.

$\forall x \in \mathcal{L}, \ \exists y : |y| = \text{poly}(|x|)$ and $A(x,y) = \text{yes}$

$\forall x \notin \mathcal{L}, \ \forall y \qquad A(x,y) = \text{no} \}$

i.e., $\exists$ a poly-time <u>verifier</u> for every language in $\mathcal{L}$.

co-NP is reverse : ...

(exercise : show that $P \subseteq NP \cap coNP$)

NP = non-deterministic polynomial -time.

Another way to think about NP : ~~allow our~~ extend our computer to make "guesses", or exist in multiple states simultaneously. This is known as non determinism. Then NP is the class of languages decidable by a non-deterministic Turing machine in <u>polynomial-time.</u>

Eg. : 3SAT ~~a~~ non-deterministic algo:

nondetermin. step → 1. make a guess for assignment of each variable
2. If ϕ satisfied, return yes, else return no.

3. If any guess returns yes, say yes, else say no.

Reductions: what does it mean for a problem to be at least as hard as another?

$\mathcal{L}_1$ is poly-time reducible to $\mathcal{L}_2$, written $\mathcal{L}_1 \leq_p \mathcal{L}_2$, if $\exists$ a poly-time computable f s.t. for any string x,

$$x \in \mathcal{L}_1 \iff f(x) \in \mathcal{L}_2.$$

So if $\mathcal{L}_2 \in P$, then $\mathcal{L}_1 \in P$.

If for every language $\mathcal{L}$ in NP, $\mathcal{L} \leq_p \mathcal{L}'$, then $\mathcal{L}'$ is NP-hard.
If $\mathcal{L}' \in NP$, then $\mathcal{L}'$ is NP-complete.

<u>Cook's Theorem</u>: 3-SAT is NP-complete.

Will show: 3-SAT $\leq_p$ IND-SET. (define IS!)

Hence k-IS is NP-hard.

Want to construct poly-time computable f:

$$\phi \text{ satisfiable} \iff f(\phi) \text{ has a } m\text{-IS}.$$

Given ϕ w/ m clauses, each clause of ϕ becomes a triangle. Add edges b/w $x_i, \bar{x}_i$. Then $f(\phi) = (\text{graph } G, m)$.

IS $\Rightarrow$ formula satisfiable (selected vertices set to true)
formula satisfiable $\Rightarrow$ IS (in each clause, choose one true vertex).

NP-hardness: Reductions & Approximation.

Last time:

$NP = \{ \mathcal{L} :$ $\exists$ a poly-time algo $A(x,y)$ s.t.
$\forall x \in \mathcal{L} : \exists y : |y| = \text{poly}(|x|)$ and $A(x,y) = \text{yes}$
$\forall x \notin \mathcal{L}, \forall y$ $A(x,y) = \text{no} \}$

i.e., NP is the class of languages for which there is a poly-time verifiable certificate of membership.

Language $\mathcal{L}^*$ is <u>NP-hard</u> if $\forall \mathcal{L} \in NP$, there is a poly-time reduction from $\mathcal{L}$ to $\mathcal{L}^*$

written $\mathcal{L} \leq_p \mathcal{L}^*$

i.e. $\exists$ a poly-time computable f s.t. for any string x,

$x \in \mathcal{L}$ iff $f(x) \in \mathcal{L}^*$

So if $\mathcal{L}^* \in P$, $\mathcal{L} \in P$ also.

(NP-complete: NP-hard + NP)

<u>Cook's Theorem</u>: 3-SAT is NP-complete.

Define $IS = \{ (G,k) : G$ has an IS of size $\geq k \}$

Theorem: IS (independent set) is NP-hard.

(NP-completeness should be easy).

Proof: Need to come up w/ poly-time computable f that takes input ϕ of 3-SAT, outputs (G, k) instance.

& ϕ satisfiable $\iff$ G has IS of size $\geq k$.

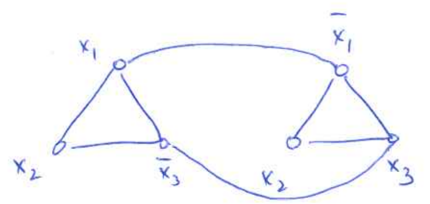
Construction: say ϕ has m clauses, n variables.

G has $3m$ vertices, with a triangle for each clause.

Each vertex hence corresponds to a literal.

Add edge between $x_i, \bar{x}_i$ for all variables x_i

Eg. $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$



~~add~~ and choose $k = m$. Note: 2 types of edges: clause edges & variable edges.

Now say G has an IS of size $\geq m$.

Claim: If any vertex for literal ~~$\bar{x}_i$~~ x_i is in IS, no vertex $\bar{x}_i$ is in IS.

easy, since there are $x_i, \bar{x}_i$ edges

Then each triangle / clause has $\geq$ one vertex in IS. Set these literals "on".

i.e., if x_i in IS, set x_i to T, If $\bar{x}_i$ in IS, set ~~or~~ x_i to F.

Set remaining variables arbitrarily.

Easy to see this gives satisfying assignment.

Can do reverse direction also: if ϕ satisfiable, G has IS of size m.

Choose one literal that evaluates to T in each clause, put corresponding vertex in IS.

Theorem: ~~k+~~ VC is NP-complete
(reduction from IS, do yourself)

Theorem: Subset-sum is NP-complete.

Problem: Given n integers $s_1, s_2, \ldots, s_n$ & T, does $\exists\, S \subseteq [n]$ s.t. $\sum_{i \in S} s_i = T$?

~~Proo~~ (note input-size; subset-sum is solvable in poly-time if $\sum_i s_i = O(\text{poly}(n))$) by dynamic programming.

Proof: By reduction from 3SAT. Given formula ϕ w/ n variables, m clauses:

- each s_i, T is given by $n + m$ bits.
- Each variable x_i corresponds to 2 integers

t_i

f_i $\underbrace{\qquad\qquad\qquad}_{n}\ \underbrace{\qquad\qquad\qquad}$

1 in i^{th} posn, 0 elsewhere — 1 for every clause the literal appears in.

- each clause C_j corresponds to 2 equal integers, C_{j1}, C_{j2}:

$$\begin{array}{lcc} & & \quad j \\ C_{j1} & \underbrace{0}_{n} & \underbrace{0 \quad 1 \quad 0}_{m} \\ C_{j2} & & \end{array}$$

$$T: \quad \underbrace{1 \; \cdots \; 1}_{n} \quad \underbrace{3 \; 3 \; \cdots \; 3}$$

example: $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$

	1	2	3	1	2	3
t_1	1	0	0	1	0	0
f_1	1	0	0	0	1	1
t_2	0	1	0	0	0	1
f_2	0	1	0	1	1	0
t_3	0	0	1	0	1	1
f_3	0	0	1	1	0	0
C_{j1}, C_{j2}	0	0	0	1	0	0
C_{21}, C_{22}	0	0	0	0	1	0
C_{31}, C_{32}	0	0	0	0	0	1
T	1	1	1	3	3	3

Let $f(\phi)$ be a yes instance of subset-sum.

Then: (i) exactly one of t_i, f_i is in S, $\forall i$

(ii) for each clause, at least one literal in the clause is in S.

Thus, <u>claim</u>: If $f(\phi)$ is in SUBSET-SUM, $\phi \in$ 3SAT.

<u>Proof</u>: Set each literal in S to be true. This is consistent, since for each variable, exactly one of t_i, f_i is in S.

Further, this is a satisfying assignment.

<u>Claim</u>: If $\phi \in$ 3SAT, $f(\phi)$ is in SUBSET-SUM. do yourself.

Set Cover: Greedy Algo.

Problem: given universe $\mathcal{U} = \{1, \ldots, n\}$ of n elements

$\mathcal{S} = \{S_1, \ldots, S_m\}$ are ~~m~~ subsets of $\mathcal{U}$

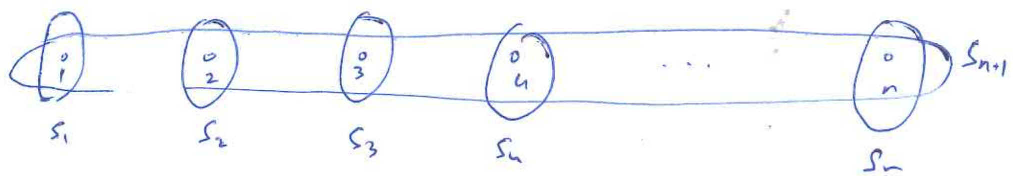
$\text{cost} : \mathcal{S} \to \mathbb{Q}_+$ are costs of the subsets

Find min-cost set of subsets $\mathcal{T}$ that covers $\mathcal{U}$, i.e.,

$\mathcal{T} \subseteq \mathcal{S}$ and $\bigcup_{S \in \mathcal{T}} \{S \in \mathcal{T}\} = \mathcal{U}$, $\mathcal{T}$ has minimum cost

subject to this.

E.g.:



$S_1 \ldots S_n$ have cost 1, S_{n+1} has cost $1+\varepsilon$.

Let OPT be cost of optimal set of subsets $\mathcal{T}^*$.

The problem is NP-hard, we will show an $O(\log n)$ approximation algo.

Algo:

$U' \leftarrow U, T \leftarrow \phi$ (set of uncovered element)

while $U' \neq \phi$

  choose $S \in \mathcal{S}$ that minimizes $\text{cost}(S) / |U' \cap S|$

  add S to $\mathcal{T}$

  $U' \leftarrow U' \setminus S$

at each step choose most cost-effective set. Hence greedy.

clearly, if $\exists$ a set-cover, when algorithm terminates Υ will be a set cover (hence algo is correct). We will show approximation guarantee.

Let's order elements by when they were covered: $e_1, \ldots, e_n$ breaking ties arbitrarily.

If e_i was covered by $S \in \Upsilon$, define $\text{price}(e_i) = \dfrac{\text{cost}(S)}{\#\text{ of uncovered elts. when } S \text{ was selected, covered by } S}$

We will prove: $\forall i \in [n]$, $\text{price}(e_i) \leq \dfrac{OPT}{n-i+1}$.

~~Note that~~ $\cancel{\text{cost}(\Upsilon) = \sum_{S \in \Upsilon} \text{cost}(S) = \sum_{S \in \Upsilon} \frac{\text{cost}(S)}{\#\text{ of uncovered elts when}} \times \#\text{ of uncovered elts when} \ldots}$

Let $S_1, \ldots, S_k$ be order of selected subsets

S_i cover elements $l_i \ldots h_i$

$$\underbrace{\overset{l_1}{e_1}, e_2, \overset{h_1}{e_3}}_{S_1}, \underbrace{\overset{l_2,h_2}{e_4}}_{S_2}, \underbrace{\overset{l_3}{e_5}, \overset{h_3}{\ldots}}_{S_3}, \ldots, \underbrace{e_{n-1}, e_n}_{S_k}.$$

Then $\text{cost}(\Upsilon) = \sum_{i=1}^{k} \text{cost}(S_i) = \sum_{i=1}^{k} \underbrace{\frac{\text{cost}(S_i)}{(h_i - l_i + 1)}}_{\text{price of elts. } e_{l_i}, e_{l_i+1}, \ldots e_{h_i}} (h_i - l_i + 1) = \sum_{i=1}^{k} \text{price}(e_{l_i}) + \ldots \text{price}(e_{h_i})$

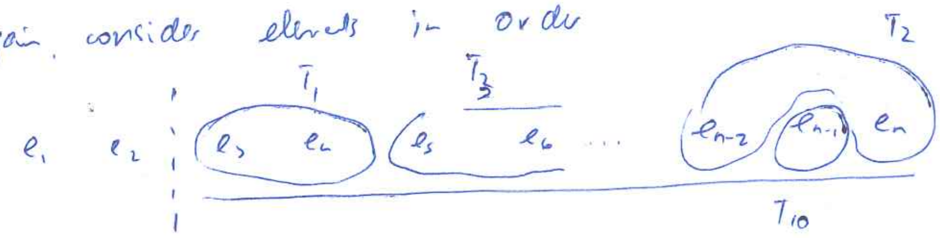
$$= \sum_{i=1}^{n} \text{price}(e_i)$$

$$\leq OPT \sum_{i=1}^{n} \frac{1}{n-i+1} = H_n \cdot OPT$$

We need to prove:

<u>Claim</u>: $\text{price}(e_i) \leq \dfrac{OPT}{n-i+1} \quad \forall i = 1 \dots n$

<u>Proof</u>: Again, consider elements in order



Fix i, consider elements $e_i, e_{i+1}, \dots e_n$.

Let $T_1, T_2, \dots, T_k$ ~~cover these~~ in $\mathcal{T}^*$ cover these element. Assign each element to a set that covers it arbitrarily.

Then $OPT \geq \sum_{j=1}^{k} \text{cost}(T_j) \cancel{= \sum_{j=i}^{n} \frac{\text{cost of set } T \text{ that covers } e_j}{\# \text{ of elts in } e_i \dots e_n \text{ that } T \text{ covers}}}$

$$= \sum_{j=1}^{k} \frac{\text{cost}(T_j)}{\# \text{ of elts. assigned to } T_j} \times \# \text{ of elts. assigned to } T_j$$

$$\Rightarrow \exists T_j : \frac{\text{cost}(T_j)}{\# \text{ of elts in } e_i, \dots, e_n \text{ assigned to } T_j} \leq \frac{OPT}{n-i+1}$$

$$\text{or, } \exists T_j : \frac{\text{cost}(T_j)}{\# \text{ of elts in } e_i, \dots, e_n \text{ in } T_j} \leq \frac{OPT}{n-i+1}, \quad T_j \in \mathcal{T}^*.$$

However, let S be set in $\mathcal{T}$ that covers e_i. Then S minimizes $\dfrac{\text{cost}(S)}{|\{e_i, \dots, e_n\} \cap S|}$. Hence $\text{Price}(e_i) = \dfrac{\text{cost}(S)}{|\{e_i, \dots, e_n\} \cap S|} \leq \dfrac{OPT}{n-i+1}$