Lecture 22: Complexity 2

Task: factor 65941

ANNOUNCEMENTS

1) +2 points on final for filling out official course evals
   - due Sunday Aug 23

Task 2: verify $65941 = 23 \cdot 47 \cdot 61$

factoring $\in$ NP !

If P = NP:

- All emails immediately hacked
  encryption based on assumption that factoring is hard!
- You get $1,000,000 from Clay Math Institute.
- put mathematicians out of work.

If $P \neq NP$:

- Says something fundamental about our universe!
- For many questions, there is no clever alg/solution

Reminders:

Def **P** is the class of all problems solvable in time $O(n^c)$ where c is constant

"solvable" = $\exists$ an algorithm with this runtime

P stands for "polynomial time". It is a set of problems.

Def **EXP** is the class of problems that can be solved in $O(2^{n^c})$ time, c = constant.

Def **NP** is the class of decision problems for which YES instances have a "proof" or "certificate" that can be verified in polynomial time $O(n^c)$, c = const.

more formally: $f \in NP$ iff $\exists$ algorithm $V(x,y)$ and polynomials p and q, s.t.

1. $\forall x, y$, $V(x,y)$ runs in time $p(|x|)$
2. $\forall x \in f^{-1}(1)$ $\exists y$, $|y| \leq q(|x|)$, s.t. $V(x,y) = 1$.
3. $\forall x \in f^{-1}(0)$ and $\forall y$ of length $\leq q(|x|)$, $V(x,y) = 0$.

V = "verifier"

y = "proof/certificate"

I kept saying: "If Hamiltonian Path / 3-COLOR / .... has a polynomial time algorithm, then $P = NP$"

Why is this true? Why doesn't that simply mean that specific problem is in P, and nothing more?

To understand why this is, need to understand reductions

REDUCTIONS

Def We say that problem **A is polynomial time reducible to B** if there is an algorithm which, using a (hypothetical) polynomial time algorithm for B, solves A in polynomial time.

notation: $A \leq_p B$

intuition: A is no harder to solve than B, because if we can solve B, then the reduction tells us we can also solve A!

Warning: Don't get it backwards!

another example: On your homework, you showed a reduction

"shortest path w/ integer weights" $\leq_p$ "BFS"

(small)

another example: 3-COLOR $\leq_p$ 4-COLOR

ALG: on input G, add a new vertex, and connect it to all other vertices

call this graph G'

return 4-COLOR(G')

need to check: 1) if G 3-colorable, then G' 4-colorable

2) if G not 3-colorable, then G' not 4-colorable

} correctness

3) ALG runs in polynomial time, assuming 4-COLOR does

] poly-time

Def: A is NP-complete iff ① $\forall B \in NP$, $B \leq_p A$

② and $A \in NP$.

motivation: these are the "hardest" problems in NP

so if we want to show $P \neq NP$, these would be the best candidates! (most likely not to have a poly time algo)

Def A is NP-hard iff $\forall B \in NP$, $B \leq_p A$.

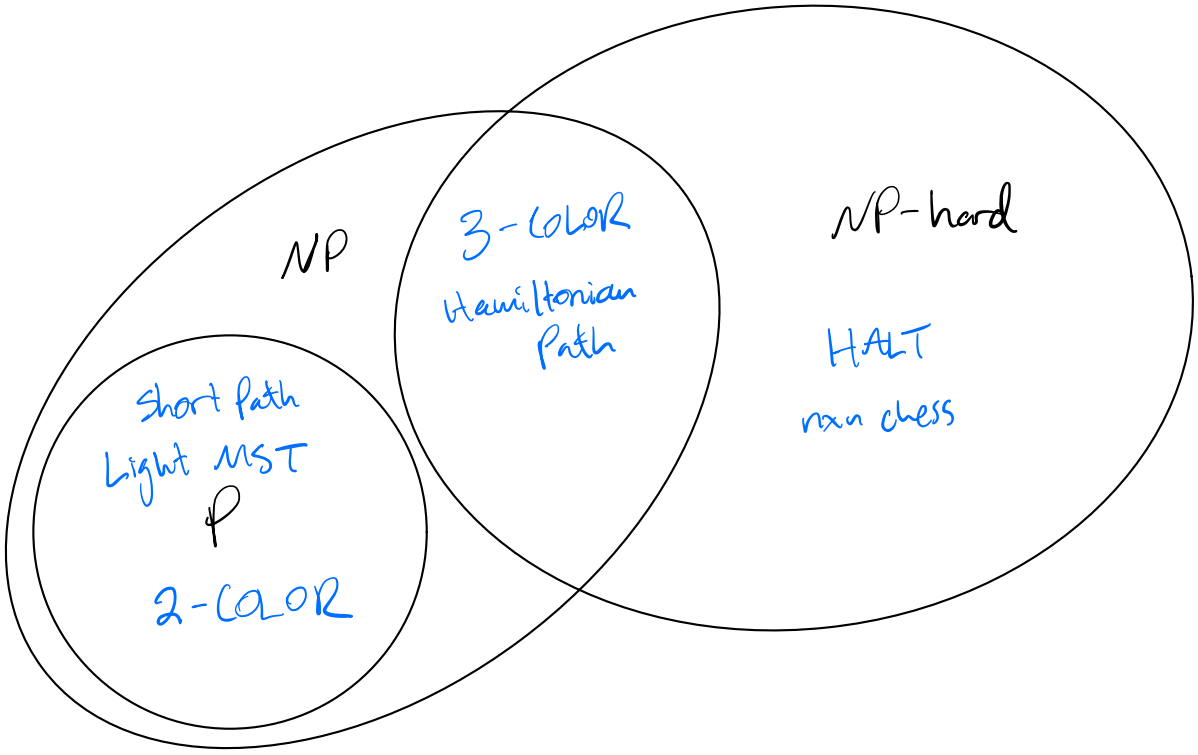
Remark: Many problems are NP hard, but not in NP

ex] halting problem is NP-hard (but certainly not NP-complete!)

$n \times n$ chess: is there an optimal strategy for white?

not easy to verify!

What we think the world looks like



if $P=NP$, then P "grows" and "consumes" all of NP.

Yet another example ... Boolean Satisfiability (SAT)

$$\Phi = (\overline{x_1} \vee x_3 \vee x_7) \wedge (x_2 \vee \overline{x_3} \vee x_1) \wedge (x_{12} \vee \overline{x_4} \vee x_1)$$
$$\wedge \quad \ldots \wedge (x_{77} \vee x_{13} \vee \overline{x_5})$$

SAT: "does Φ have a satisfying assignment"

3-SAT: CNF w/ ≤3 variables per clause

Theorem (Cook-Levin) SAT is NP-Complete

* actually, even 3SAT is too!

proof idea 1) $SAT \in NP$

2) goal: $\forall A \in NP, \quad A \leq_p SAT$

suppose $A \in NP$. fix instance x s.t. $A(x) = 1$.

we know: $\exists$ "proof" y and verifier V s.t. $V(x,y) = 1$

rough idea: using x, V, create a SAT instance Φ s.t.

Φ satisfiable $\iff \exists y, V(x,y) = 1$

inputs of Φ will be y!

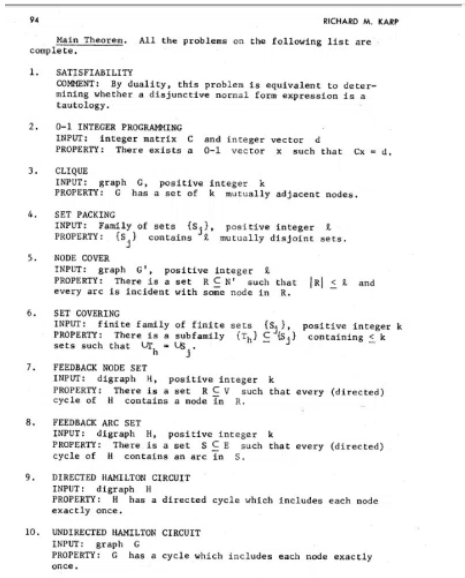
Corollary 1 if $SAT \in P$, then $P = NP$.

why?

Corollary 2 If $SAT \leq_p B$ and $B \in NP$, then B is NP-Complete.

proof $A_1 \leq_p A_2 \leq_p A_3 \Rightarrow A_1 \leq_p A_3$

why?



94 RICHARD M. KARP

Main Theorem. All the problems on the following list are complete.

1. SATISFIABILITY
COMMENT: By duality, this problem is equivalent to determining whether a disjunctive normal form expression is a tautology.

2. 0-1 INTEGER PROGRAMMING
INPUT: integer matrix C and integer vector d
PROPERTY: There exists a 0-1 vector x such that $Cx = d$.

3. CLIQUE
INPUT: graph G, positive integer k
PROPERTY: G has a set of k mutually adjacent nodes.

4. SET PACKING
INPUT: Family of sets $\{S_j\}$, positive integer ℓ
PROPERTY: $\{S_j\}$ contains ℓ mutually disjoint sets.

5. NODE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: There is a set $R \subseteq N'$ such that $|R| \leq \ell$ and every arc is incident with some node in R.

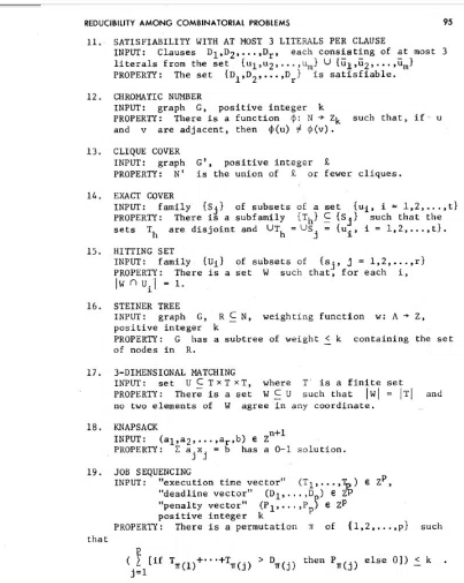
6. SET COVERING
INPUT: finite family of finite sets $\{S_i\}$, positive integer k
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ containing $\leq k$ sets such that $\cup T_h = \cup S_j$.

7. FEEDBACK NODE SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $R \subseteq V$ such that every (directed) cycle of H contains a node in R.

8. FEEDBACK ARC SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $S \subseteq E$ such that every (directed) cycle of H contains an arc in S.

9. DIRECTED HAMILTON CIRCUIT
INPUT: digraph H
PROPERTY: H has a directed cycle which includes each node exactly once.

10. UNDIRECTED HAMILTON CIRCUIT
INPUT: graph G
PROPERTY: G has a cycle which includes each node exactly once.



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS 95

11. SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
INPUT: Clauses $D_1, D_2, \ldots, D_r$, each consisting of at most 3 literals from the set $\{u_1, u_2, \ldots, u_n\} \cup \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n\}$
PROPERTY: The set $\{D_1, D_2, \ldots, D_r\}$ is satisfiable.

12. CHROMATIC NUMBER
INPUT: graph G, positive integer k
PROPERTY: There is a function $\phi: N \to Z_k$ such that, if u and v are adjacent, then $\phi(u) \neq \phi(v)$.

13. CLIQUE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: N' is the union of ℓ or fewer cliques.

14. EXACT COVER
INPUT: family $\{S_j\}$ of subsets of a set $\{u_i, i = 1,2,\ldots,t\}$
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that the sets T_h are disjoint and $\cup T_h = \cup S_j = \{u_i, i = 1,2,\ldots,t\}$.

15. HITTING SET
INPUT: family $\{U_j\}$ of subsets of $\{a_j, j = 1,2,\ldots,r\}$
PROPERTY: There is a set W such that, for each i, $|W \cap U_i| = 1$.

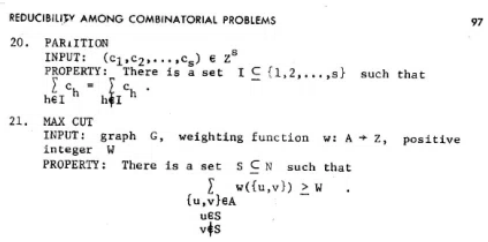
16. STEINER TREE
INPUT: graph G, $R \subseteq N$, weighting function $w: A \to Z$, positive integer k
PROPERTY: G has a subtree of weight $\leq k$ containing the set of nodes in R.

17. 3-DIMENSIONAL MATCHING
INPUT: set $U \subseteq T \times T \times T$, where T is a finite set
PROPERTY: There is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate.

18. KNAPSACK
INPUT: $(a_1, a_2, \ldots, a_r, b) \in Z^{n+1}$
PROPERTY: $\sum a_j x_j = b$ has a 0-1 solution.

19. JOB SEQUENCING
INPUT: "execution time vector" $(T_1, \ldots, T_p) \in Z^p$,
"deadline vector" $(D_1, \ldots, D_p) \in Z^p$
"penalty vector" $(P_1, \ldots, P_p) \in Z^p$
positive integer k
PROPERTY: There is a permutation π of $\{1,2,\ldots,p\}$ such that

$$\left(\sum_{j=1}^{p} [\text{if } T_{\pi(1)}+\cdots+T_{\pi(j)} > D_{\pi(j)} \text{ then } P_{\pi(j)} \text{ else } 0]\right) \leq k \ .$$



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS 97

20. PARTITION
INPUT: $(c_1, c_2, \ldots, c_s) \in Z^s$
PROPERTY: There is a set $I \subseteq \{1,2,\ldots,s\}$ such that

$$\sum_{h \in I} c_h = \sum_{h \notin I} c_h \ .$$

21. MAX CUT
INPUT: graph G, weighting function $w: A \to Z$, positive integer W
PROPERTY: There is a set $S \subseteq N$ such that

$$\sum_{\substack{\{u,v\} \in A \\ u \in S \\ v \notin S}} w(\{u,v\}) \geq W \ .$$

Karp's Theorem (1972)

A lot of problems people care about are NP-complete

3SAT
Hamiltonian Path
3-COLOR
K-CLIQUE
Knapsack
⋮

} All NP-complete!

Scenario: You have a problem in NP, but need to solve it. What to do??

options:

1. give up.

2. make sure it's not actually in P

3. Convert it to a SAT problem, use a SAT-solver

4. check and see if you are dealing with a special instance

   (some problems have fast algorithms for "nice" instances!)

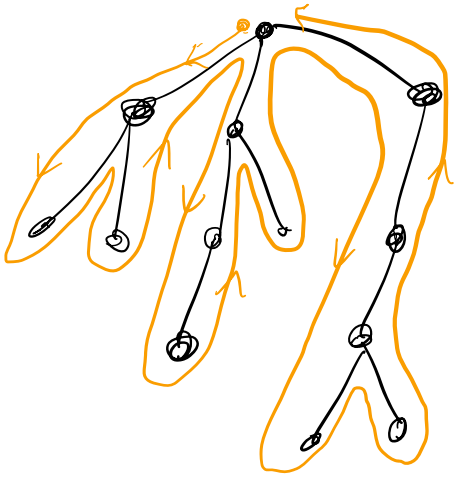
5. Use an approximation algorithm

Approximation Alg Example

TSP: given weighted graph G, find a tour (walk that visits every vertex and returns to start) of minimum total weight.

approximation algorithm:

① Compute a MST



② let your tour be the orange walk

claim: this produces a tour of weight at most twice as much as the best possible tour.

⟹ I can get a pretty good "approximation" of the answer!

why?

Aside: Why is it "nondeterministic" polynomial time?

a "nondeterministic" program can have multiple possible next states

(generalization of an NFA from 311!)

a nondeterministic (turing machine) program "computes" f if $\forall\ x \in f^{-1}(1)$, $\exists$ a valid sequence of states ending in "accept"

and $\forall\ x \in f^{-1}(0)$, all valid sequences of states end in "reject".

takeaway: this leads to an alternate (less intuitive) definition of NP, equivalent to our "verifier" one.