Complexity Theory

Seen lots of algorithms in class, lots of solvable problems.

Q: what can't we solve?

Unfortunate answer: almost everything.

Small variation of problem ⇒ probably intractable

Flows: Max $s \to t$ flow ✓

multicommodity flow:

K commodities, each w/ own $s_i \to t_i$

✗ can't even for $K=2$

Min cut: Smallest $(S, V \setminus S)$ cut ✓

Max cut: largest $(S, V \setminus S)$ cut ✗

min spanning tree: smallest tree connecting all nodes ✓

min Steiner tree: only need to connect subset $S \subseteq V$ ✗

Interval Packing: most disjoint intervals ✓

↳ on multiple machines: each interval works on $M_i \subseteq [K]$ machines ✗

Shortest path ✓

Longest path ✗

Complexity theory studies hard problems

- how hard are they?
- classify into "complexity classes"

Zooms out, for big picture view.

Now: n = size of input in bits

only consider <u>decision problems</u>
(binary answer: YES or NO)

Examples:
"Shortest Path" has input (G, s, t, K)
Q: does there exist $s \leadsto t$ path in G
of length $\leq K$

what is n?

$$
\begin{aligned}
n = \; & E \lg V && \leftarrow \text{represent graph} \\
& + E \cdot w && \leftarrow \text{edge costs, } w \text{ bit words} \quad [\text{length} \leq 2^w] \\
& + 2 \lg V && \leftarrow s, t \\
& + (w + \lg V) && \leftarrow \text{representation of } K \\
& = \Theta(E(w + \lg V)) && [K \leq V \cdot 2^w]
\end{aligned}
$$

$$
\begin{aligned}
\text{Dijkstra} &= O(E + V \lg V) && \underline{\text{word}} \text{ operations} \\
&= O(Ew + wV \lg V) && \underline{\text{bit}} \text{ operations} \\
&\leq O(n \log n)
\end{aligned}
$$

"min cut" $\to$ "$\exists$ cut of size $\le k$"
"min Spanning tree" $\to$ "$\exists$ spanning tree of size $\le K$"
"max cut" $\to$ "$\exists$ cut of size $\ge k$"

If you can solve decision problem, can solve optimization ("what is shortest path length?")
by binary search on k.
[$O(\lg n)$ iterations]

$\Rightarrow$ can find solution ("what is the path?")
[remove edges sequentially & see if length changes]
$O(E) \le O(n)$ times slower

Big Picture: such differences don't matter, & decision problems are simpler to study

P: Problems solvable in Polynomial time
[$= n^{O(1)}$ time]

Shortest path $\in$ P.

Longest Path ($\exists$ simple path of length $\ge k$?)
seems hard to solve —
but can <u>check</u> a solution w/ proof
(if a long path exists, can check it)
(if <u>doesn't</u> exist, can't prove it)

NP: Nondeterministic polynomial time

problems for which $\exists$ poly time <u>verifier</u> A.

"Completeness" $\forall$ YES instances x

(= input x s.t. answer = YES)

$\exists$ proof $y \in \{0,1\}^{poly(n)}$

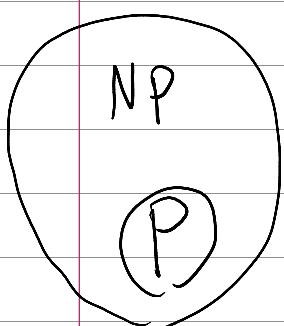
s.t. $A(x,y) = YES$

"Soundness" $\forall$ NO instances x

$\nexists$ proof $y \in \{0,1\}^{poly(n)}$

s.t. $A(x,y) = YES$

[anything true can be proven; nothing false can be]



NP still means <u>easy</u> (ish)

$P \subseteq NP$: $A(x,y) = A'(x)$.

$\exists$ much harder problems:

how many longest paths are there?

Does white win this chess position?

Does this program halt?

Million dollar Q: is $P = NP$?

finding proof harder than checking it?

(Max-cut, longest path, multicommodity flow, steiner tree, integer LP, ...) $\in NP$

Unknown if they lie in P.

Best unconditional lower bound for <u>any</u> problem:

$3.011\, n$

But we do know something:

all those problems equally hard: all in P or none.

How to show problem A easier than B if we don't know how hard either one is?

Reductions

A "reduces to" B

(= "Reduction from A to B")

if: can solve A using B

Cook Reduction:

given oracle to B, can solve A using $n^{O(1)}$ time + $n^{O(1)}$ calls to B.

Karp Reduction:

$$A(x) = B(f(x))$$

for polynomial time function f.

$A \preceq_p B \iff$ Karp reduction $A \to B$

We've seen linear time reductions:

Bipartite matching reduces to flow

fastest soln to game reduces to shortest path

In algorithms: (unknown, new problem) reduces to (known easy problem)

⇒ new problem no harder than old

In complexity: (known, "hard" problem) reduces to (unknown, new problem)

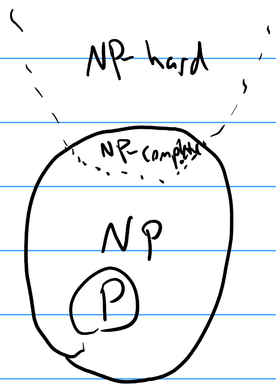
⇒ new problem no easier than old

Cook's Theorem ∃ problem, called SAT,

such that SAT ∈ NP and:

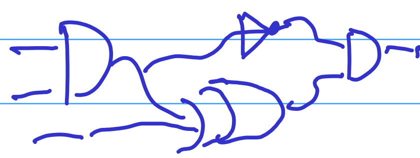
If SAT ∈ P,

then P = NP



equiv: every $q \in NP$ can be solved in a polynomial reduction to SAT

Circuit SAT Boolean Circuit Satisfiability



SAT

$$(a \vee b) \cap \overline{(c \cup d)} \cup \overline{(e \cap (f \cup a))}$$

3-SAT

$$(a \vee b \vee c) \cap (\_\_) \cap (\_\_) \cap (\_\_)$$

3 vars per clause

rest of Karp's 21 problems