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 decision problems
(binary answer: YES or NO)

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

what is n?

$n = E \lg V$ ← represent graph [length $\le 2^w$]
$+ E \cdot w$ ← edge costs, w bit words
$+ 2 \lg V$ ← s, t
$+ (w + \lg V)$ ← representation of K
$= \Theta(E(w + \lg V))$ $[K \le V \cdot 2^w]$

Dijkstra $= O(E + V \lg V)$ word operations
$= O(Ew + wV \lg V)$ bit operations
$\le O(n \log n)$

"min cut" → "$\exists$ cut of size $\leq k$"
"min Spanning tree" → "$\exists$ spanning tree of size $\leq k$"
"max cut" → "$\exists$ cut of size $\geq k$"

If you can solve decision problem, can solve optimization ("what is shortest path length?")
by binary search on k.
[$O(\lg n)$ iterations]
⇒ can find solution ("what is the path?")
[remove edges sequentially & see if length changes]
$O(E) \leq O(n)$ 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 $\geq k$?)
seems hard to solve –
but can check a solution w/ proof
(if a long path exists, can check it)
(if doesn't 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 $y \in \{0,1\}^{\text{poly}(n)}$

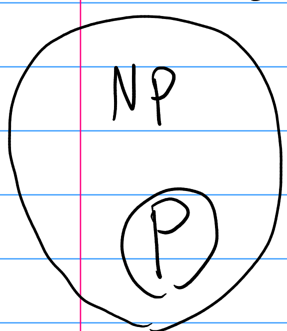
s.t. $A(X, y) = \text{YES}$

"Soundness" $\forall$ NO instances X

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

s.t. $A(X, y) = \text{YES}$

[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)

$\Rightarrow$ new problem no harder than old

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

$\Rightarrow$ new problem no easier than old

Cook's Theorem $\exists$ problem, called SAT, such that every problem $P \in NP$