15-251

Great Theoretical Ideas in Computer Science

A quick recap on reductions and NP-hardness

Lecture 29 (December 3, 2009)

In the previous lecture, we saw two problem classes: P and NP

The Class P

We say a set $L \subseteq \Sigma^*$ is in P if there is
a program A and
a polynomial p( )

such that for any x in Σ^*,

A(x) runs for at most p(|x|) time
and answers question "is x in L?" correctly.

The Class P

The class of all sets L that can be recognized in polynomial time.

The class of all decision problems that can be decided in polynomial time.

P

contains many useful problems:

- graph connectivity
- minimum spanning tree
- matchings in graphs
- shortest paths
- solving linear systems Ax = b
- linear programming
- maximum flows

Many of this we will (re)visit in 15-451.

NP

A set $L \in NP$

if there exists an algorithm A and a polynomial p( ) such that

For all $x \in L$	For all $x' \notin L$
there exists y with $\|y\| \le p(\|x\|)$	For all y' with $\|y'\| \le p(\|x'\|)$
such that A(x,y) = YES	such that $A(x',y')$ = NO
in $p(\|x\|)$ time	in $p(\|x\|)$ time
"exists a quickly-verifiable proof"	"all non-proofs rejected"

The Class NP

The class of sets L for which there exist "short" proofs of membership
(of polynomial length)
that can be "quickly" verified
(in polynomial time).

Recall: A doesn't have to find these proofs y; it just needs to be able to verify that y is a "correct" proof.

$P \subseteq NP$

For any L in P, we can just take y to be the empty string and satisfy the requirements.

Hence, every language in P is also in NP.

Summary: P versus NP

Set L is in P if membership in L can be decided in poly-time.

Set L is in NP if each x in L has a short "proof of membership" that can be verified in poly-time.

Fact: $P \subseteq NP$

Million (Billion) $ question: Does $NP \subseteq P$?

NP Contains Lots of Problems We Don't Know to be in P

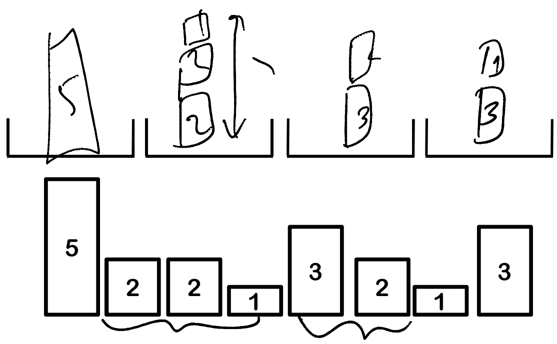
Classroom Scheduling
Packing objects into bins
Scheduling jobs on machines
Finding cheap tours visiting a subset of cities
Allocating variables to registers
Finding good packet routings in networks
Decryption
…

E.g. Scheduling Jobs on Machines

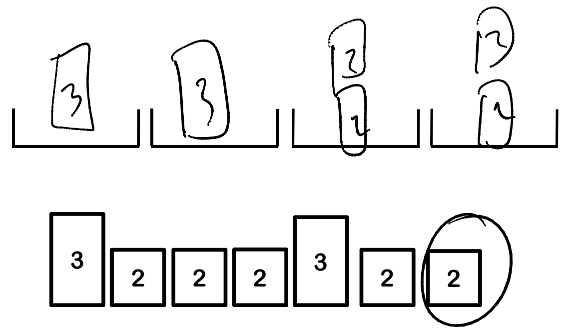
Input:
A set of n jobs, each job j has processing time p_j
A set of m identical machines
A value T

Can you allocate these n jobs to these m machines such that the latest ending time over all jobs $\le$ T?

E.g.: m=4 machines, n=8 jobs, T = 5



E.g.: m=4 machines, n=7 jobs, T = 4



E.g. Scheduling Jobs on Machines

Input:
A set of n jobs, each job j has processing time p_j
A set of m identical machines
A value T

Can you allocate these n jobs to these m machines such that the latest ending time over all jobs $\leq$ T?

(This latest ending time is called the "makespan")

we think it is NP hard…

NP hardness proof

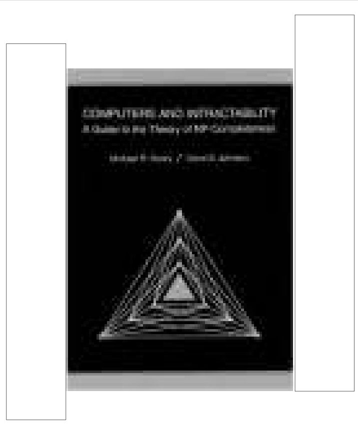
To prove NP hardness, find a problem such that

a) that problem is itself NP hard

b) show that if you can solve Makespan minimization quickly, you can solve that problem quickly

"reduce a hard problem to Makespan-minimization"

Can you suggest such a problem?



The (NP hard) Partition Problem

Given a set A = $\{a_1, a_2, \ldots, a_n\}$ of n naturals which sum up to 2B (B is a natural), find a subset of these that sums to exactly B.

so we've found a problem such that:
a) the problem is itself NP hard

Garey & Johnson prove that Partition is NP-hard

b) show that if you can solve Makespan minimization quickly, you can solve Partition quickly

Take any instance ($A = \{a_1, \ldots, a_n\}$, B) of Partition

Each natural number in A corresponds to a job.
The processing time $p_j = a_j$

We have m = 2 machines.

Easy Theorem: there is a solution with makespan = B iff there is a partition of A into two equal parts.

$\Rightarrow$ if you solve Makespan fast, you solve Partition fast.

$\Rightarrow$ if Partition is hard, Makespan is hard.

E.g. Scheduling Jobs on Machines

Input:
A set of n jobs, each job j has processing time p_j
A set of m identical machines

Allocate these n jobs to these m machines to minimize the ending time of the last job to finish.

(We call this objective function the "makespan")

NP hard!

I now know that Makespan Minimization is NP-hard.

How do I show that Makespan Minimization is NP-complete?

Show that the problem is itself in NP.

Being in NP

If someone claims the answer is YES, can they convince you in polynomial time that the answer is YES?

I.e., is there a poly-time checkable "proof" that truly YES instances should pass
and
none of the NO instances should pass?

Makespan $\in$ NP

Makespan is NP-hard

$\Rightarrow$ Makespan is NP-complete