Intro to P and NP: poly-time and reductions

Recall from Lecture 1

I Design of Algorithms
II Analysis of Algorithms

III Lower Bounds: do we have the best algorithm?

Lower bounds, Problem P, Algorithm A. Run-time $t(n)$.

When is Algorithm A good enough?

e.g. branch and bound can be $\Theta(2^n)$

Is that the best algorithm?

Would need to show that any algorithm for problem P has worst case run-time $\geq 2^n$ asymptotically

i.e. $\Omega(2^n)$

Such lower bounds are hard to prove.

State of the Art in Lower Bounds / Impossibility Results

* Some problems don't have algorithms
  
  Turing 1930's. We'll cover this at end of course.

  Also in CS 245 and CS 360.

* Some problems can only be solved in exponential-time.
* Some problems have like $\Omega(n \log n)$ lower bounds on restricted model of computing, e.g. sorting.
Major Open Question
There are many problems e.g., Travelling Salesman, 0-1 Knapsack, where no one knows a poly-time alg., and no one can prove there's no poly-time alg.
The best we can do is prove that a large set of problems are equivalent in the sense that a poly-time alg. for one yields a poly-time alg. for all.

Our focus is on polynomial time
Our main tool is reductions

The class of equivalent problems are the NP-complete problems
Polynomial Time = Efficient

Polynomial time means [worst case] running time is $O(n^k)$ for some constant $k$. $n = \text{input size}$
e.g., $\Theta(2^n)$, $\Theta(n!)$ are NOT poly. time.
Most of the alg's we've studied have been poly. time,
except backtracking, branch-and-bound, pseudo-poly.
time alg. for 0-1 knapsack $O(n \cdot W)$.

Polynomial time = "good"
quote from Edmonds [from Schrijver's book]
in his 1963 paper "Paths, Trees and Flowers".

2. Digression. An explanation is due on the use of the words "efficient algorithm." First, what I present is a conceptual description of an algorithm and not a particular formalized algorithm or "code."

For practical purposes computational details are vital. However, my purpose is only to show as attractively as I can that there is an efficient algorithm. According to the dictionary, "efficient" means "adequate in operation or performance." This is roughly the meaning I want—in the sense that it is conceivable for maximum matching to have no efficient algorithm. Perhaps a better word is "good."

I am claiming, as a mathematical result, the existence of a good algorithm for finding a maximum cardinality matching in a graph.

There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially with the size of the graph. It is by no means obvious whether or not there exists an algorithm whose difficulty increases only algebraically with the size of the graph.

The mathematical significance of this paper rests largely on the assumption that the two preceding sentences have mathematical meaning. I am not prepared to set up the machinery necessary to give them formal meaning, nor is the present context appropriate for doing this, but I should like to explain the idea a little further informally. It may be that since one is customarily concerned with existence, convergence, finiteness, and so forth, one is not inclined to take seriously the question of the existence of a better-than-finite algorithm.
Reduction

Problem $A$ reduces to problem $B$, written $A \leq_p B$, if an algorithm for $B$ can be used to make an algorithm for $A$. “$A$ is easier than $B$.”

Problem $A$ reduces in poly-time to problem $B$, written $A \leq_{p} B$, if a poly-time algorithm for $B$ can be used to make a poly-time algorithm for $A$.

Consequences of $A \leq_{p} B$:

- A lower bound for $A$ [A cannot be solved in poly. time] yields a lower bound for $B$ [B cannot …]

Even if we don’t have an alg. for $B$ or a lower bound for $A$, we can still use reductions to show that problems are equivalently hard (show $A \leq_{p} B, B \leq_{p} A$)
Example of Reductions

Hamiltonian cycle/path = cycle/path that visits every vertex exactly once

This graph has a Hamiltonian path but not a Hamiltonian cycle.

Lemma: Hamiltonian path $\leq_p$ Hamiltonian cycle.

Proof: Suppose we have a poly. time algorithm for Hamiltonian cycle.

We want to design a poly. time algorithm for Hamiltonian path.

Input: graph $G$.

Algorithm:
- construct graph $G'$ by adding one new vertex adjacent to all vertices of $G$
- send $G'$ to the algorithm to test for Hamiltonian cycle
- return the YES/NO answer

This alg. runs in poly. time.
Correctness: must prove

Claim: $G$ has a Ham. path iff $G'$ has a Ham. cycle

Proof:
- Suppose $G$ has a Ham. path $x$ to $y$.
  - Adding $v$ and edges $(x, v), (v, y)$ gives a Ham. cycle in $G'$.
- Suppose $G'$ has a Ham. cycle. Removing $v$ gives a Ham. path in $G$.

Lemma: Ham. cycle $\leq_{p}$ Ham. path

Exercise: prove this.

FACT: no one knows a poly. time alg. for either problem.
Decision Problems
- problems where output is YES/NO

Theory of NP-completeness focuses on decision problems
- it’s easier that way
- optimization and decision are usually equivalent wrt poly. time.

Examples
- given a number, is it prime?
- given a graph, does it have a Hamiltonian cycle?
- given an edge weighted graph and number k, does it have a TSP tour of length \( \leq k \)?

Equivalence of optimization and decision
- no general proof, but usually OK
- e.g., max ind. set - an ind. set in graph \( G \) is a set of vertices \( U \subseteq V \), no two joined by an edge
- optimization - find size of max ind. set
- decision - given \( k \), is max ind. set \( \geq k \)?

Decision \( \leq_p \) optimization
- just see if the max ind. set is \( \geq k \)
- find max size by asking for each \( k = 1 \ldots n \)
- find actual max ind. set by throwing away vertices one by one and repeatedly asking decision
Equivalence is not always known
e.g. test if a number is prime or composite
seems easier than finding its prime factorization

$P = \exists$ decision problems that have polynomial
time algorithms?

what model? bit complexity