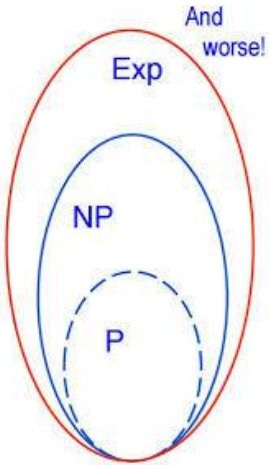
Lecture 22

Computational Complexity II



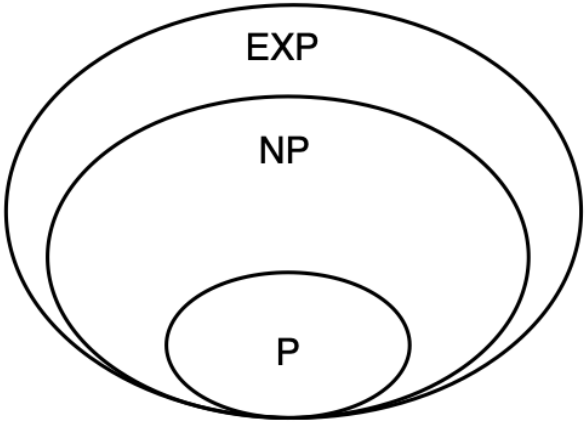
Announcements

- Quiz 2 releases Friday; due by end of semester (5/8)
- Group Project problems now up; choose one soon!

Complexity Classes

Nested classification of problems

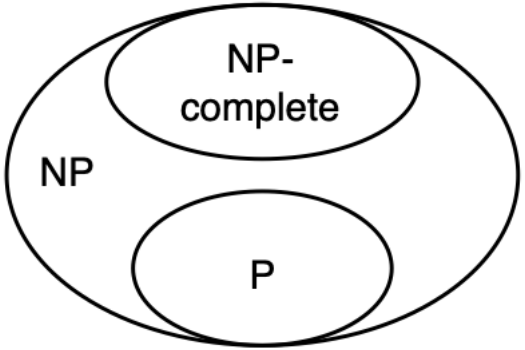
- **P**: solvable in polynomial time
- **NP**: includes most natural and interesting problems
- **EXP**: solvable in exponential time



NP-Completeness

NP-Complete

- Problems that are "as difficult" as every other problem in **NP**
- A polynomial time algorithm for any NP-Complete problem implies one for every problem in **NP**



Outline

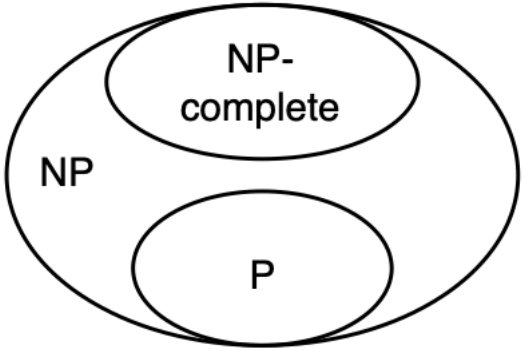
Goal: make sense of this picture

Polynomial-time reductions

- What it means for one problem to be "as difficult as" another problem

Define NP and NP-Complete

Work through examples



Polynomial-Time Reduction

Problem Y is **polynomial-time reducible** to Problem X if any instance of Problem Y can be solved with

- A polynomial number of standard computational steps
- A polynomial number of calls to a black box that solves problem X

```
solveY(yInput)
    Construct xInput                // poly-time
    foo = solveX(xInput)            // poly # of calls to solveX
    return yes/no based on foo      // poly-time
```

Notation: $Y \leq_P X$

Useful for discussing problems in **P** or complexity higher classes but erases distinctions between subclasses of **P**

Decision Problems

When thinking about the complexity of problems, we usually frame the problems in terms of answering a YES/NO question

Decision Problems

When thinking about the complexity of problems, we usually frame the problems in terms of answering a YES/NO question

- Given a bipartite graph, does there exist a perfect matching? (Perfect Matching problem)

Decision Problems

When thinking about the complexity of problems, we usually frame the problems in terms of answering a YES/NO question

- Given a bipartite graph, does there exist a perfect matching? (Perfect Matching problem)
- Given a capacity W and a set of items each with a weight, can we find a subset of items with combined weight less than W but greater than threshold T? (Knapsack problem)

Decision Problems

When thinking about the complexity of problems, we usually frame the problems in terms of answering a YES/NO question

- Given a bipartite graph, does there exist a perfect matching? (Perfect Matching problem)
- Given a capacity W and a set of items each with a weight, can we find a subset of items with combined weight less than W but greater than threshold T? (Knapsack problem)
- Given a set of integers, does there exist a subset that sums to zero? (Subset sum problem)

Decision Problems

When thinking about the complexity of problems, we usually frame the problems in terms of answering a YES/NO question

- Given a bipartite graph, does there exist a perfect matching? (Perfect Matching problem)
- Given a capacity W and a set of items each with a weight, can we find a subset of items with combined weight less than W but greater than threshold T? (Knapsack problem)
- Given a set of integers, does there exist a subset that sums to zero? (Subset sum problem)

All the problems we've looked at this semester
can be framed as decision problems

P and NP

Complexity Class **P**

- Decision problems for which there is a polynomial time algorithm

Complexity Class **NP**

- Decision problems for which there is a polynomial time certifier
- A solution can be "verified" in polynomial time
- NP means non-deterministic polynomial time

Solver vs Certifier

Let X be a decision problem and s be an input to the problem

- Example: Subset sum problem with input set S

Solver

- Takes as input a problem input and correctly returns YES or NO
- Types of algorithms we've been developing all semester

Certifier

- Takes as input a problem input and a "certificate" or "witness" and correctly returns YES

Certifier Example: Subset Sum

Input: set of integers $S = \{7, 3, -1, -24, 6, -2\}$

Problem: does S have a subset that sums to zero? $\text{Type equation here.}$

Witness: a witness w is a specific subset of S

Idea: we can check if w verifies that the answer is YES

Certifier Example: Subset Sum

Input: set of integers $S = \{7, 3, -1, -24, 6, -2\}$

Problem: does S have a subset that sums to zero? $\text{Type equation here.}$

Witness: a witness w is a specific subset of S

Idea: we can check if w verifies that the answer is YES

How would this work?

Certifier Example: Subset Sum

Input: set of integers $S = \{7, 3, -1, -24, 6, -2\}$

Problem: does S have a subset that sums to zero? $\text{Type equation here.}$

Witness: a witness w is a specific subset of S

Idea: we can check if w verifies that the answer is YES

How would this work?

- If w sums to zero, then return YES
- w is a certificate that YES is correct

Certifier Example: Subset Sum

Input: set of integers $S = \{7, 3, -1, -24, 6, -2\}$

Problem: does S have a subset that sums to zero? Type equation here.

Witness: a witness w is a specific subset of S

Idea: we can check if w verifies that the answer is YES

How would this work?

- If w sums to zero, then return YES
- w is a certificate that YES is correct

If w does not sum to zero, what does that tell us about the problem?

Certifier Example: Subset Sum

Input: set of integers $S = \{7, 3, -1, -24, 6, -2\}$

Problem: does S have a subset that sums to zero? Type equation here.

Witness: a witness w is a specific subset of S

Idea: we can check if w verifies that the answer is YES

How would this work?

- If w sums to zero, then return YES
- w is a certificate that YES is correct

If w does not sum to zero, what does that tell us about the problem?

- Likely, not much

Solver vs Certifier

There are problems with polynomial-time certifiers but may not have polynomial-time solvers

Subset Sum problem

- Baseline algorithm has running time:

Solver vs Certifier

There are problems with polynomial-time certifiers but may not have polynomial-time solvers

Subset Sum problem

- Baseline algorithm has running time: $O(2^n)$

Solver vs Certifier

There are problems with polynomial-time certifiers but may not have polynomial-time solvers

Subset Sum problem

- Baseline algorithm has running time: $O(2^n)$
- Best known dynamic programming algorithm: $O(nT)$ meaning pseudo-polynomial

Solver vs Certifier

There are problems with polynomial-time certifiers but may not have polynomial-time solvers

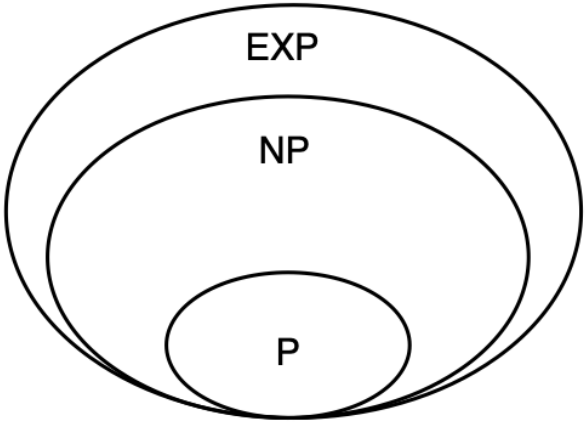
Subset Sum problem

- Baseline algorithm has running time: $O(2^n)$
- Best known dynamic programming algorithm: $O(nT)$ meaning pseudo-polynomial
- If a given subset sums to zero, we can verify it in running time $O(n)$

Therefore, the Subset Sum problem is in **NP**

P, NP, EXP

- **P**: easy to solve
- **NP**: easy to verify (may be hard to solve)
- **EXP**: may be hard to solve or verify

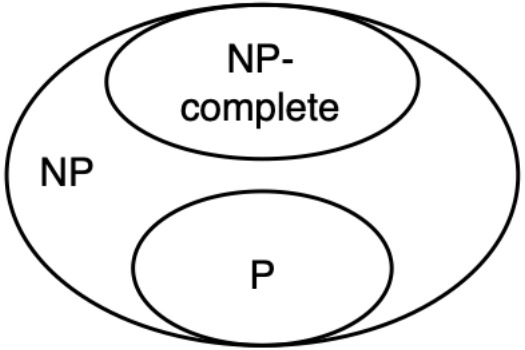


NP-Completeness

Problems that are "as difficult" as every other problem in **NP**

A problem X is NP-Complete if

- X is in **NP**
- If Y is in **NP**, then $Y \leq_P X$ (known as NP-hard)



The Subset Sum Problem is NP-Complete

More examples

title

blah

- blah

title

blah

- blah

title

blah

- blah

title

blah

- blah

title

blah

- blah