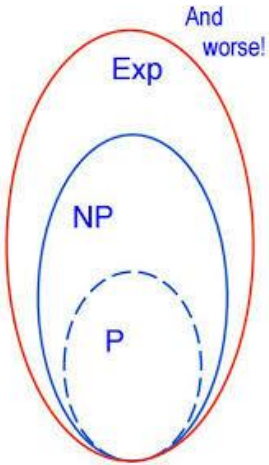
Lecture 21

Computational Complexity



Announcements

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

Algorithm Design

- Formulate our problem precisely
- Find a baseline algorithm
- Design an efficient algorithm
- Prove correctness
- Analyze running time

Algorithm Design

- Formulate our problem precisely
- Find a baseline algorithm
- Design an efficient algorithm
- Prove correctness
- Analyze running time

What if you cannot find an efficient algorithm?

MST Extension

Suppose you have a connected graph $G = (V, E)$ with edge weights

Minimum spanning tree (MST) problem

- Find minimum-cost subset of edges so that there is a path between any $u, v \in V$
- $O(m \log n)$ using a greedy algorithm

MST Extension

Suppose you have a connected graph $G = (V, E)$ with edge weights

Minimum spanning tree (MST) problem

- Find minimum-cost subset of edges so that there is a path between any $u, v \in V$
- $O(m \log n)$ using a greedy algorithm (we saw Prim's and Kruskal's algorithms)

MST Extension

Suppose you have a connected graph $G = (V, E)$ with edge weights

Minimum spanning tree (MST) problem

- Find minimum-cost subset of edges so that there is a path between any $u, v \in V$
- $O(m \log n)$ using a greedy algorithm (we saw Prim's and Kruskal's algorithms)

Minimum Steiner tree problem

- Find minimum-cost subset of edges so there is a path between any $u, v \in W$ for target $W \subseteq V$
- No polynomial-time algorithm is known

Subset Sum/Knapsack Problem

We looked at the subset sum problem in the second dynamic programming lecture

Input: n items with weights, capacity W

Goal: maximize total weight without exceeding capacity

- $O(nW)$ pseudo-polynomial time algorithm
- No polynomial time algorithm is known!

Tractability

Working definition of efficiency

- Polynomial time: $O(n^d)$ for some d
- Examples: linear, quadratic, $O(n^{1000})$ running times

Huge class of natural and interesting problems where:

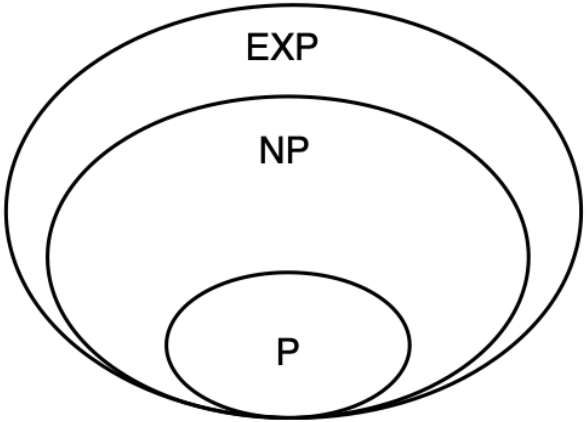
- We do not know any polynomial time algorithm
- We cannot prove that none exists

Goal: develop mathematical tools to say when a problem is hard or "intractable"

Complexity Classes

Nested classification of problems

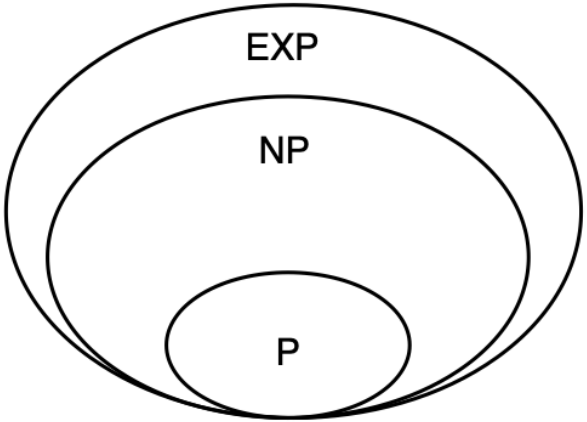
- **P**: solvable in polynomial time
- **NP**: includes most natural and interesting problems
- **EXP**: solvable in exponential time



Complexity Classes

Nested classification of problems

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



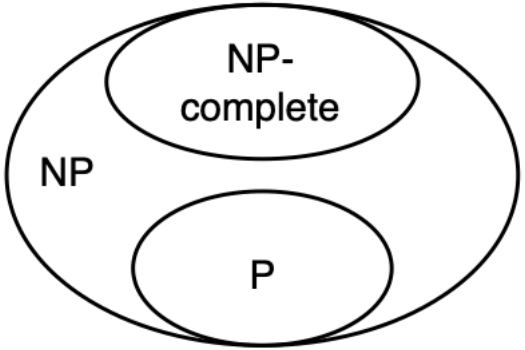
NP means nondeterministic polynomial

- Does not mean "not polynomial"

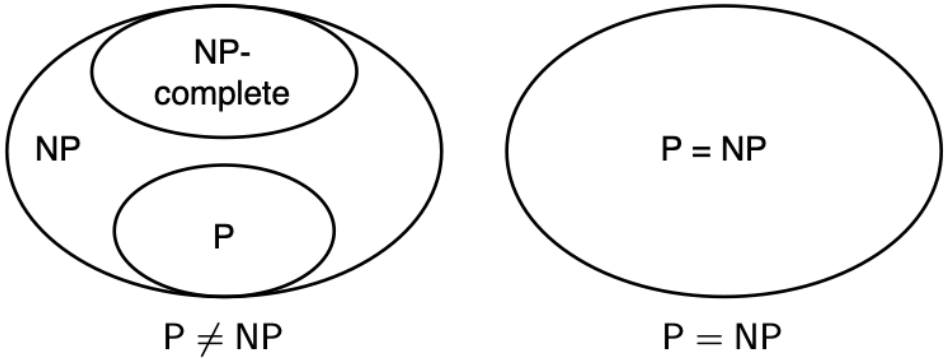
NP-Completeness

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**



P ≠ NP?



We don't know which world is true

- One of the Millennium Problems
- It's possible that neither is true (if you're interested in this answer, consider a class in mathematical logic)

Outline

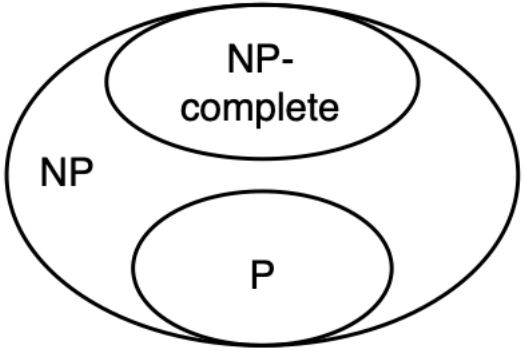
Goal: make sense of this picture

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

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
```

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: $Y \leq_P X$

Exercise

Suppose $Y \leq_P X$. Which of the following can we infer?

a. If X can be solved in polynomial time, then so can Y

b. If Y can be solved in polynomial time, then so can X

c. If Y cannot be solved in polynomial time, then neither can X

Exercise

Suppose $Y \leq_P X$. Which of the following can we infer?

a. If X can be solved in polynomial time, then so can Y

b. If Y can be solved in polynomial time, then so can X

c. If Y cannot be solved in polynomial time, then neither can X

Polynomial-Time Reduction

Suppose $Y \leq_P X$

- This is a statement at their relative difficulty or "hardness"
- If X is solvable in poly-time, then so is Y
- If Y is not solvable in poly-time, then neither is X

Polynomial-Time Reduction

Suppose $Y \leq_P X$

- This is a statement at their relative difficulty or "hardness"
- If X is solvable in poly-time, then so is Y
- If Y is not solvable in poly-time, then neither is X

What is a reductive we've seen recently?

Polynomial-Time Reduction

Suppose $Y \leq_P X$

- This is a statement at their relative difficulty or "hardness"
- If X is solvable in poly-time, then so is Y
- If Y is not solvable in poly-time, then neither is X

What is a reductive we've seen recently?

- We reduced the Perfect Matching problem to a Max Flow problem