CSC 349-07: Design and Analysis of Algorithms **Winter 2023**

P vs NP: March 7, 2023

Instructor: Rodrigo Canaan *Scribe: Ishaan Sathaye*

1 Announcements

- No new programming assignment
- Quiz 3 and other quizzes missed retake this Thursday (March 9, 2023)
- Look out for an activity next week

2 Complexity

2.1 Big O

Big O time complexity does not care about multiplicative constants. For example, consider a matrix multiplication algorithm. Reading a contiguous chunk of memory would be faster than reading columns in another matrix. However, Big O does not care about this.

2.2 Polynomial vs. Non-Polynomial

Time complexities such as $O(n^2)$, $O(n^3)$, $O(2^n)$, $O(n!)$, and $O(n^n)$ are all polynomial time complexities and bounded by a polynomial.

An example of a non-polynomial time complexity would be $O(2^n)$, which is significantly worse than something like $O(n^{100})$.

3 NP

3.1 Decision Problems[1]

Definition 1 *A problem for which any proposed solution can be quickly checked for correctness.*

Examples:

- **Finding x**: Is there an x that satisfies C some constraint?
- **Path Finding**: Is there a path from u to v in a graph G?
- **Knapsack**: Is there a knapsack that has a value of 100 while not going over the weight of 15?

[1] We will be focusing more on these types of NP Problems

- **Prime Number**: Does the number n have any factors f with $f \leq n$ and $f \neq 1$?
- **Sudoku**: Is there a solution to a given Sudoku board?

3.1.1 Path Finding Algorithm

FINDPATH()

---

1. Check Edges → **polynomial**
2. Solve Subproblems → **polynomial** → O($n + m$)

3.1.2 Sudoku Algorithm

Note 2 *An O(n^3) algorithm, which makes it a polynomial time algorithm.*

CheckSudoku and SolveSudoku **Input:** B = a Sudoku board, S = a set of Sudoku boards which are solutions to B **Goal:** Is there a solution to B?

CHECKSUDOKU(B, S)

---

1. For each cell → O(n^2) cells
   (a) Check row → O(n)
   (b) Check column → O(n)
   (c) Check 3x3 grid → O(n)

Checking edges is polynomial time.

SOLVESUDOKU(B)

---

1. Generate all possible solutions of B
2. Check each solution

Solving would be O(n^n) making it exponential not polynomial.

3.2 Search Problems

Definition 3 *A problem for which a solution is found by searching through a space of possible solutions. Sits in between decision problems and optimization problems.*

Examples:

- **Finding x**: Find an x subject to some constraint C.
- **Path Finding**: Find a path from u to v in a graph G.
- **Knapsack**: Find a knapsack that has a value of 100 while not going over the weight of 15.
- **Prime Number**: Find a factor or all factors f with $f \leq n$ and $f \neq 1$.
- **Sudoku**: Find a solution to a given Sudoku board.

3.3 Optimization Problems

Definition 4 *Harder problem in general since a solution to this problem would be a solution to a* Decision *problem.*

Examples:

- **Finding x**: Find an x^* that minimizes/maximizes $f(x)$ subject to some constraint C.
- **Path Finding**: Find a minimum length path from u to v in a graph G.
- **Knapsack**: Find a knapsack that $max(\sum_{i=1}^{n} v_i)$ subject to $\sum_{i=1}^{n} v_i \leq w$.
- **Maximum Increasing Subsequence**: Find a subsequence of a given sequence that is increasing and has the maximum length.

3.4 P vs NP

P stands for **polynomial** and **NP** stands for **non-deterministic polynomial**[2].

Class of **NP** Problems

- *Some decision problems that can be* CHECKED *in polynomial time.*
- Examples: Knapsack (convert from an optimization problem to a decision problem), Independent Set, Traveling Salesperson Problem, Minimum Spanning Tree, Matching, etc.

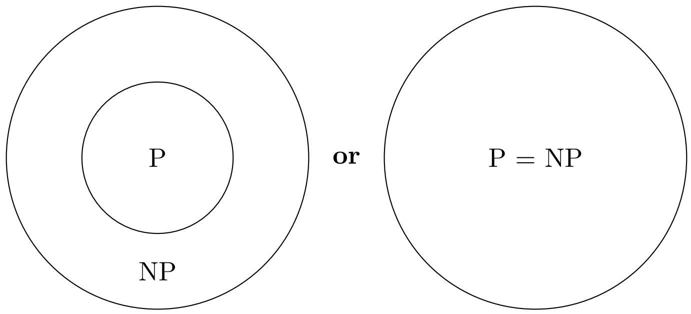
Class of **P** Problems

- *Some decision problems that can be* SOLVED *in polynomial time.*
- Examples: Path Finding, Minimum Spanning Tree, Longest Increasing Subsequence, Independent Set on Trees, Bipartite Matching, etc.

[2]NP problems would be be solvable in polynomial if we had a deterministic machine.

3.5 P = NP?

Does P = NP? If not, then P $\subseteq$ NP, essentially P $\neq$ NP. But does P = NP? We don't know. We can't prove or disprove it. This leads to the mathematical puzzle:



SOLVE(P)

---

1. Generate all potential solutions of P.
2. Check each solution to see if it is correct.

---

4 NP-Complete

Definition 5 (NP-Complete) *A problem X is NP-complete if:*

1. *X is in NP.*
2. *Every problem in NP is reducible to X in polynomial time.*

Note 6 *A problem satisfying condition 2 is NP-hard, whether or not it satisfies condition 2.*

5 Reductions

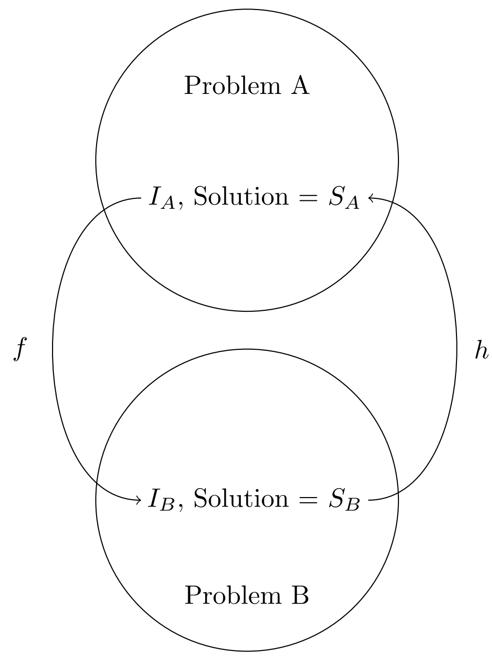
Definition 7 *A reduction from a decision problem A to a decision problem B ($A \to B$) is a polynomial time algorithm, f:*

- *That transforms an instance, I, of A into an instance, $f(I)$, of B.*

Together with another polynomial time algorithm, h:

- *That maps any solution S of $f(I)$ to a solution $h(S)$ of I.*

If $f(I)$ has no solution then neither does I.



If such a reduction exists, it implies that B is at least as hard as A.