COMP20007 Design of Algorithms

Complexity Theory

Daniel Beck

Lecture 21

Semester 1, 2020

Complexity Theory

- So far, we have been concerned with the analysis of algorithms' running times.

Complexity Theory

- So far, we have been concerned with the analysis of algorithms' running times.
- Complexity theory asks a different question: "What is the inherent difficulty of the problem?"

Complexity Theory

- So far, we have been concerned with the analysis of algorithms' running times.
- Complexity theory asks a different question: "What is the inherent difficulty of the problem?"
- It does however also uses asymptotic notation, although usually more concerned with lower bounds (Ω notation).

Complexity Theory

- So far, we have been concerned with the analysis of algorithms' running times.
- Complexity theory asks a different question: "What is the inherent difficulty of the problem?"
- It does however also uses asymptotic notation, although usually more concerned with lower bounds (Ω notation).
- Tigther ("larger") lower bounds give us guarantees on best possible algorithms.

Complexity Theory - Examples

Comparison Sorting (worst case)

Complexity Theory - Examples

Comparison Sorting (worst case)

- A trivial lower bound: $\Omega(n)$.

if we gonna to sort array of n numbers,
we need to look at n elements

Complexity Theory - Examples

Comparison Sorting (worst case)

n ! possible remutation

- A trivial lower bound: $\Omega(n)$.
- A less trivial lower bound: $\Omega(\log n!) \approx \Omega(n \log n)$
  - Can be found by using a technique called Decision Trees.

$\boxed{?}$ BINAry tree

with n ! leaves

height = log(# leaves).

minimum compariasion.

$h = \log(n!) = \Omega(n\log n)$

any possiblealgorithm
based on comparison
sorting has to perform
at least nlogn

(cannot go better
than $\Omega(n\log n)$)
for worst case

Complexity Theory - Examples

Comparison Sorting (worst case)

- A trivial lower bound: $\Omega(n)$.
- A less trivial lower bound: $\Omega(\log n!) \approx \Omega(n \log n)$
  - Can be found by using a technique called Decision Trees.
- Bound is tight: we know $\Theta(n \log n)$ algorithms.

Complexity Theory - Examples

Comparison Sorting (worst case)

- A trivial lower bound: $\Omega(n)$.
- A less trivial lower bound: $\Omega(\log n!) \approx \Omega(n \log n)$
  - Can be found by using a technique called Decision Trees.
- Bound is tight: we know $\Theta(n \log n)$ algorithms.

Matrix Multiplication square $n \times n$

- A trivial lower bound: $\Omega(n^2)$

Complexity Theory - Examples

Comparison Sorting (worst case)

- A trivial lower bound: $\Omega(n)$.
- A less trivial lower bound: $\Omega(\log n!) \approx \Omega(n \log n)$
  - Can be found by using a technique called Decision Trees.
- Bound is tight: we know $\Theta(n \log n)$ algorithms.

Matrix Multiplication

- A trivial lower bound: $\Omega(n^2)$
- Unknown if bound is tight: best algorithms are $O(n^{2.37})$

STRASSEN's MATRIX ALGORITHMS $O(n^{2.187})$

Complexity Theory - Examples

Comparison Sorting (worst case)

- A trivial lower bound: $\Omega(n)$.
- A less trivial lower bound: $\Omega(\log n!) \approx \Omega(n \log n)$
  - Can be found by using a technique called Decision Trees.
- Bound is tight: we know $\Theta(n \log n)$ algorithms.

Matrix Multiplication

- A trivial lower bound: $\Omega(n^2)$
- Unknown if bound is tight: best algorithms are $O(n^{2.37})$

This lecture: discussion about “hardness” of problems.

Decision Problems

- A decision problem takes an input and generates a **YES** or a **NO** answer as the output.

Decision Problems

- A decision problem takes an input and generates a **YES** or a **NO** answer as the output.
  - "Is the integer n a prime number?"

Decision Problems

- A decision problem takes an input and generates a **YES** or a **NO** answer as the output.
    - "Is the integer n a prime number?"
    - "Is there a circuit that visits all nodes in a graph exactly once?" (Hamiltonian Circuit Problem)

Decision Problems

- A decision problem takes an input and generates a **YES** or a **NO** answer as the output.
  - "Is the integer n a prime number?"
  - "Is there a circuit that visits all nodes in a graph exactly once?" (Hamiltonian Circuit Problem)
- Optimisation problems can be framed as a sequence of decision problems:

Decision Problems

- A decision problem takes an input and generates a **YES** or a **NO** answer as the output.
  - "Is the integer n a prime number?"
  - "Is there a circuit that visits all nodes in a graph exactly once?" (Hamiltonian Circuit Problem)
- Optimisation problems can be framed as a sequence of decision problems:
  - Knapsack: "Is there a set of items of values at least i and weight at most j?"

Verification Problems

- A verification problem takes an input, a proposed solution and verifies if the solution satisfy the input.

Verification Problems

- A verification problem takes an input, a proposed solution and verifies if the solution satisfy the input.
  - "Given n and $\{i_1, i_2, \ldots, i_k\}$, check if $\prod i = n$"

Verification Problems

easier

- A verification problem takes an input, a proposed solution and verifies if the solution satisfy the input.
  - "Given n and $\{i_1, i_2, \ldots, i_k\}$, check if $\prod i = n$"
  - "Given a graph G and a sequence of nodes $\{v_1, v_2, \ldots, v_n\}$, verify if there is a path that follows that sequence.

True $\rightarrow$ n is not prime

False $\rightarrow$? ?

Verification Problems

- A verification problem takes an input, a proposed solution and verifies if the solution satisfy the input.
  - "Given n and $\{i_1, i_2, \ldots, i_k\}$, check if $\prod i = n$"
  - "Given a graph G and a sequence of nodes $\{v_1, v_2, \ldots, v_n\}$, verify if there is a ~~path~~ circuit that follows that sequence.

start from v1
keep check whether
next node is
connected
$\boxed{\text{linear}}$

Now we can define what is P and what is NP.

P and NP

- A problem is in *P* if its decision version has a solution which is polynomial in the input size.

eg. matrix multiplication

P and NP

- A problem is in *P* if its decision version has a solution which is polynomial in the input size.
- A problem is in *NP* if its verification version has a solution which is polynomial in the input size.

P and NP

- A problem is in *P* if its decision version has a solution which is polynomial in the input size.
- A problem is in *NP* if its verification version has a solution which is polynomial in the input size.
- We can turn a verification problem into a decision problem:

P and NP

- A problem is in *P* if its decision version has a solution which is polynomial in the input size.
- A problem is in *NP* if its verification version has a solution which is polynomial in the input size.
- We can turn a verification problem into a decision problem:

  will be able generate all candidates in parallel

  - 1) A non-deterministic "machine" generates a candidate.
  - 2) The verification algorithm verifies the solution.
  - 3) Repeat until verified.

P and NP

- A problem is in *P* if its decision version has a solution which is polynomial in the input size.
- A problem is in *NP* if its verification version has a solution which is polynomial in the input size.
- We can turn a verification problem into a decision problem:
  - 1) A non-deterministic "machine" generates a candidate.
  - 2) The verification algorithm verifies the solution.
  - 3) Repeat until verified.
- In other words, a problem is in *NP* if its decision version has a solution which is non-deterministically polynomial in the input size.

P and NP

- A problem is in *P* if its decision version has a solution which is polynomial in the input size.
- A problem is in *NP* if its verification version has a solution which is polynomial in the input size.
- We can turn a verification problem into a decision problem:
  - 1) A non-deterministic “machine” generates a candidate.
  - 2) The verification algorithm verifies the solution.
  - 3) Repeat until verified.
- In other words, a problem is in *NP* if its decision version has a solution which is non-deterministically polynomial in the input size.

That’s where the *N* in *NP* comes from. =)

P and NP

- Any problem in P is in NP ($P \subset NP$).

P and NP

- Any problem in P is in NP ($P \subset NP$).
  - One can verify a solution by solving the decision version and comparing the result.

any problem that can be decided in polynomial time

can also be verified in polynomial time

P and NP

- Any problem in P is in NP ($P \subset NP$).
  - One can verify a solution by solving the decision version and comparing the result.
- The reverse is unknown... ($P \stackrel{?}{\supset} NP$)

P and NP

- Any problem in P is in NP ($P \subset NP$).
  - One can verify a solution by solving the decision version and comparing the result.
- The reverse is unknown... ($P \stackrel{?}{\supset} NP$)
  - The non-deterministic step does not guarantee efficiency.

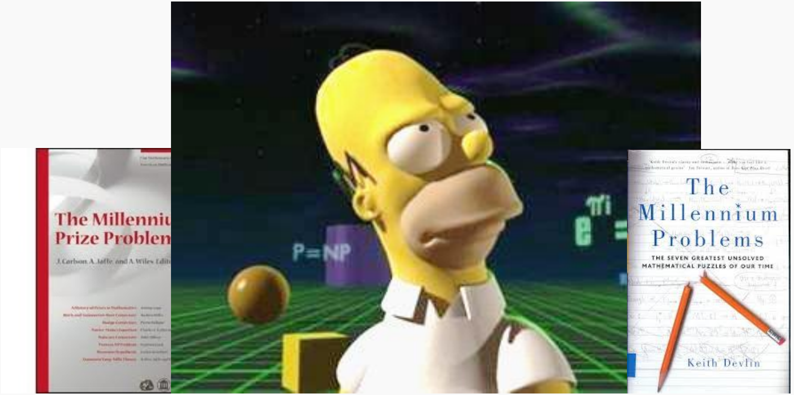
P and NP

- Any problem in P is in NP ($P \subset NP$).
  - One can verify a solution by solving the decision version and comparing the result.
- The reverse is unknown... ($P \stackrel{?}{\supset} NP$)
  - The non-deterministic step does not guarantee efficiency.

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$$

A Million Dollar Question: Is P = NP?

This is one of the seven “millennium problems”: The Clay Institute’s seven most important unsolved mathematical problems.



Reductions

- It's a daunting task to find and prove bounds for every new problem.

Reductions

- It's a daunting task to find and prove bounds for every new problem.
- Reductions allow us to ease this by, roughly, framing a problem as equivalent to another one we know the class.

Reductions

- It's a daunting task to find and prove bounds for every new problem.
- Reductions allow us to ease this by, roughly, framing a problem as equivalent to another one we know the class.
- For instance, the Hamiltonian Circuit (HAM) problem can be reduced to the decision version of TSP.

NP complete

Travelling salesman problem

Reductions

- It's a daunting task to find and prove bounds for every new problem.
- Reductions allow us to ease this by, roughly, framing a problem as equivalent to another one we know the class.
- For instance, the Hamiltonian Circuit (HAM) problem can be reduced to the decision version of TSP.
- The reduction function is polynomial. Therefore, since HAM is in *NP*, the decision version of TSP is also in *NP*.

From HAM to TSP

- Suppose we have a Hamiltonian Circuit in a graph G with n nodes.

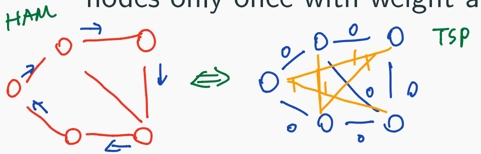
From HAM to TSP

- Suppose we have a Hamiltonian Circuit in a graph G with n nodes.
- Build a new graph G' where connected nodes in G have an edge of weight 0 and non-connected nodes have weight 1.
    - This can be done in polynomial time.

iterate all nodes

From HAM to TSP

- Suppose we have a Hamiltonian Circuit in a graph G with n nodes.
- Build a new graph G' where connected nodes in G have an edge of weight 0 and non-connected nodes have weight 1.
  - This can be done in polynomial time.
- Frame Decision-TSP as "Is there a circuit that visit all nodes only once with weight at most 0?"



NP-Completeness

A decision problem D is said to be *NP*-Complete if:

NP-Completeness

A decision problem D is said to be NP-Complete if:

- $D \in NP$ **and**

NP-Completeness

A decision problem D is said to be *NP*-Complete if:

- $D \in NP$ **and**
  - hard
  - Every problem in NP has a polynomial reduction to D **or**
  - A polynomial reduction from a known NP-complete problem to D exists.

NP-Completeness

A decision problem D is said to be *NP*-Complete if:

- $D \in NP$ **and**
  - Every problem in NP has a polynomial reduction to D **or**
  - A polynomial reduction from a known NP-complete problem to D exists.

Key property: if one finds a polynomial time algorithm to solve an NP-complete problem, then $P = NP$.

3-SAT

Proving that every problem in *NP* has a polynomial reduction to *D* is hard.

- This feat was accomplisehd in the 70's by Stephen Cook and Leonid Levin for the Boolean 3-satisfiability problem.

3-SAT

Proving that every problem in *NP* has a polynomial reduction to *D* is hard.

- This feat was accomplisehd in the 70's by Stephen Cook and Leonid Levin for the Boolean 3-satisfiability problem.
- "Given a boolean formula with a maximum of three literals, is there an assignment that results in TRUE?"

3-SAT

Proving that every problem in NP has a polynomial reduction to D is hard.

- This feat was accomplisehd in the 70's by Stephen Cook and Leonid Levin for the Boolean 3-satisfiability problem.
- "Given a boolean formula with a maximum of three literals, is there an assignment that results in TRUE?"
  - $(x_1 \overset{\text{or}}{\vee} \bar{x}_2 \vee \bar{x}_3) \overset{\text{and}}{\wedge} (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) = \text{true}$
  - $\{x_1 = \textit{true}, x_2 = \textit{true}, x_3 = \textit{false}\}$

3-SAT

From 3-SAT, one can reduce it to many other problems, all being *NP*-complete as a consequence:

3-SAT

From 3-SAT, one can reduce it to many other problems, all being *NP*-complete as a consequence:

- SAT
- Clique → find the maximum clique subgraph which is complete
- Vertex Cover → find the set of vertexes that cover all edges
- Hamiltonian Circuit
- Decision-TSP
- ...

3-SAT

From 3-SAT, one can reduce it to many other problems, all being *NP*-complete as a consequence:

- SAT
- Clique
- Vertex Cover
- Hamiltonian Circuit
- Decision-TSP
- ...

A polynomial time algorithm for any of these would imply that $P = NP$.

Summary

- Complexity Theory deals with bounds for problems.

Summary

- Complexity Theory deals with bounds for problems.
- Decision problems: P contains problems with polynomial time solutions.

Summary

- Complexity Theory deals with bounds for problems.
- Decision problems: *P* contains problems with polynomial time solutions.
- Verification problems: *NP* contains problems with polynomial time solutions.

Summary

- Complexity Theory deals with bounds for problems.
- Decision problems: P contains problems with polynomial time solutions.
- Verification problems: NP contains problems with polynomial time solutions.
- Reductions let us analyse new problems by framing them as existing ones.

Summary

- Complexity Theory deals with bounds for problems.
- Decision problems: P contains problems with polynomial time solutions.
- Verification problems: NP contains problems with polynomial time solutions.
- Reductions let us analyse new problems by framing them as existing ones.
- NP-completeness: solving one NP-complete problem implies in $P = NP$ due to reductions.

Last Words

- Some problems are undecidable: COMP30026

Last Words

- Some problems are undecidable: COMP30026
- Some scientists tried to prove that $P \stackrel{?}{=} NP$ is undecidable.

Last Words

- Some problems are undecidable: COMP30026
- Some scientists tried to prove that $P \stackrel{?}{=} NP$ is undecidable.
- Most scientists believe $P \neq NP$.

since NP complete problem

Last Words

- Some problems are undecidable: COMP30026
- Some scientists tried to prove that $P \stackrel{?}{=} NP$ is undecidable.
- Most scientists believe $P \neq NP$.
- While the problem itself still eludes computer scientists, proposed solutions led to advancements in theory, even though they were wrong.