Intractability

Data Structures and Algorithms
Emory University
Jinho D. Choi
Polynomial Time
Polynomial Time

- Complexity
Polynomial Time

- Complexity
  - Analyze the complexity of an algorithm as a function of its input size.
Polynomial Time

• Complexity
  - Analyze the **complexity** of an algorithm as a function of its input size.
  - $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!).$
Polynomial Time

• Complexity
  - Analyze the complexity of an algorithm as a function of its input size.
  - $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!)$.

• Polynomial time
Polynomial Time

• Complexity
  - Analyze the complexity of an algorithm as a function of its input size.
  - $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!)$.

• Polynomial time
  - The complexity class $P$ is the set of all problems where the solutions can be found in polynomial time.
Polynomial Time

• Complexity
  - Analyze the complexity of an algorithm as a function of its input size.
  - \( O(\log n) \), \( O(n) \), \( O(n \log n) \), \( O(n^2) \), \( O(n^3) \), \( O(2^n) \), \( O(n!) \).

• Polynomial time
  - The complexity class \( P \) is the set of all problems where the solutions can be found in polynomial time.
  - A problem is intractable if there is no polynomial-time algorithm for the problem.
Polynomial Time

• Complexity
  - Analyze the complexity of an algorithm as a function of its input size.
  - $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!)$.

• Polynomial time
  - The complexity class $P$ is the set of all problems where the solutions can be found in polynomial time.
  - A problem is intractable if there is no polynomial-time algorithm for the problem.

<table>
<thead>
<tr>
<th></th>
<th>Prim</th>
<th>Kruskal</th>
<th>Chu–Liu-Edmonds</th>
<th>Topolo. Sort</th>
<th>Dijkstra</th>
<th>Ford Fulkerson</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Polynomial Time

- **Complexity**
  - Analyze the **complexity** of an algorithm as a function of its **input size**.
  - \( O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!) \).

- **Polynomial time**
  - The complexity class \( P \) is the set of all problems where the solutions can be **found** in **polynomial time**.
  - A problem is **intractable** if there is no **polynomial-time** algorithm for the problem.

<table>
<thead>
<tr>
<th></th>
<th>Prim</th>
<th>Kruskal</th>
<th>Chu–Liu-Edmonds</th>
<th>Topolo. Sort</th>
<th>Dijkstra</th>
<th>Ford Fulkerson</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( O )</td>
<td>( E \log V )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Polynomial Time

- **Complexity**
  - Analyze the complexity of an algorithm as a function of its input size.
  - $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!)$.

- **Polynomial time**
  - The complexity class $P$ is the set of all problems where the solutions can be found in polynomial time.
  - A problem is intractable if there is no polynomial-time algorithm for the problem.

<table>
<thead>
<tr>
<th></th>
<th>Prim</th>
<th>Kruskal</th>
<th>Chu–Liu-Edmonds</th>
<th>Topolo. Sort</th>
<th>Dijkstra</th>
<th>Ford Fulkerson</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O</td>
<td>E log $V$</td>
<td>E log $V$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Polynomial Time

- **Complexity**
  - Analyze the *complexity* of an algorithm as a function of its *input size*.
  - $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!)$. 

- **Polynomial time**
  - The complexity class $P$ is the set of all problems where the solutions can be *found* in polynomial time.
  - A problem is *intractable* if there is no polynomial-time algorithm for the problem.

<table>
<thead>
<tr>
<th></th>
<th>Prim</th>
<th>Kruskal</th>
<th>Chu–Liu-Edmonds</th>
<th>Topolo. Sort</th>
<th>Dijkstra</th>
<th>Ford Fulkerson</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O$</td>
<td>$\mathbf{E} \log V$</td>
<td>$\mathbf{E} \log V$</td>
<td>$V^3 = EV$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Polynomial Time

• Complexity
  - Analyze the complexity of an algorithm as a function of its input size.
  - \( O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!) \).

• Polynomial time
  - The complexity class \( P \) is the set of all problems where the solutions can be found in polynomial time.
  - A problem is intractable if there is no polynomial-time algorithm for the problem.

<table>
<thead>
<tr>
<th></th>
<th>Prim</th>
<th>Kruskal</th>
<th>Chu–Liu-Edmonds</th>
<th>Topolo. Sort</th>
<th>Dijkstra</th>
<th>Ford Fulkerson</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O )</td>
<td>( E \log V )</td>
<td>( E \log V )</td>
<td>( V^3 = EV )</td>
<td>( E + V )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Polynomial Time

- **Complexity**
  - Analyze the complexity of an algorithm as a function of its input size.
  - $O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!)$. 

- **Polynomial time**
  - The complexity class $P$ is the set of all problems where the solutions can be found in polynomial time.
  - A problem is intractable if there is no polynomial-time algorithm for the problem.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Prim</th>
<th>Kruskal</th>
<th>Chu–Liu-Edmonds</th>
<th>Topolo. Sort</th>
<th>Dijkstra</th>
<th>Ford Fulkerson</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O$</td>
<td>$E \log V$</td>
<td>$E \log V$</td>
<td>$V^3 = EV$</td>
<td>$E + V$</td>
<td>$V^2$</td>
<td></td>
</tr>
</tbody>
</table>
Polynomial Time

• Complexity
  - Analyze the **complexity** of an algorithm as a function of its input size.
  - \(O(\log n), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!)\).

• Polynomial time
  - The complexity class \(P\) is the set of all problems where the solutions can be **found** in polynomial time.
  - A problem is intractable if there is no polynomial-time algorithm for the problem.

<table>
<thead>
<tr>
<th></th>
<th>Prim</th>
<th>Kruskal</th>
<th>Chu–Liu-Edmonds</th>
<th>Topolo. Sort</th>
<th>Dijkstra</th>
<th>Ford Fulkerson</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O)</td>
<td>E log (V)</td>
<td>E log (V)</td>
<td>(V^3 = EV)</td>
<td>(E + V)</td>
<td>(V^2)</td>
<td>(E \cdot f)</td>
</tr>
</tbody>
</table>
Non-deterministic Polynomial Time
Non-deterministic Polynomial Time

- Nondeterministic polynomial time
Non-deterministic Polynomial Time

- Nondeterministic polynomial time
  - The complexity class NP is the set of problems where proposed solutions can be proved in polynomial time by a non-deterministic Turing machine.
Non-deterministic Polynomial Time

- Nondeterministic polynomial time
  
  The complexity class NP is the set of problems where proposed solutions can be proved in polynomial time by a non-deterministic Turing machine.

Deterministic Turing Machine
Non-deterministic Polynomial Time

- Nondeterministic polynomial time
  - The complexity class $\text{NP}$ is the set of problems where proposed solutions can be proved in polynomial time by a non-deterministic Turing machine.

**Deterministic Turing Machine**

![Deterministic Turing Machine Diagram]
Non-deterministic Polynomial Time

- Nondeterministic polynomial time
  - The complexity class NP is the set of problems where proposed solutions can be proved in polynomial time by a non-deterministic Turing machine.

Non-Deterministic Turing Machine

\[ S \rightarrow A \rightarrow E \]
Non-deterministic Polynomial Time

- Nondeterministic polynomial time
  - The complexity class NP is the set of problems where proposed solutions can be proved in polynomial time by a non-deterministic Turing machine.

Non-Deterministic Turing Machine

```
S ----> A ----> B ----> E
    |       |       |
    v       v       v
    C ----> A ----> B
```

Non-deterministic Polynomial Time
Non-deterministic Polynomial Time

- Nondeterministic polynomial time
  - The complexity class NP is the set of problems where proposed solutions can be proved in polynomial time by a non-deterministic Turing machine.

Non-Deterministic Turing Machine

Example?
Non-deterministic Polynomial Time

- Nondeterministic polynomial time

  - The complexity class NP is the set of problems where proposed solutions can be proved in polynomial time by a non-deterministic Turing machine.

Non-Deterministic Turing Machine

Example?

\[
\begin{align*}
  a_1 \cdot x + b_1 \cdot y + c_1 \cdot z &= d_1 \\
  a_2 \cdot x + b_2 \cdot y + c_2 \cdot z &= d_2 \\
  a_3 \cdot x + b_3 \cdot y + c_3 \cdot z &= d_3
\end{align*}
\]
NP Hard
NP Hard

- NP-hard
NP Hard

- NP-hard
  - The complexity class **NP-Hard** is the set of problems that are “at least as hard as” the **hardest problems** in **NP**.
NP Hard

• NP-hard
  - The complexity class NP-Hard is the set of problems that are “at least as hard as” the hardest problems in NP.

• Polynomial-time algorithms for NP-hard problems?
NP Hard

• NP-hard
  - The complexity class **NP-Hard** is the set of problems that are “at least as hard as” the **hardest problems** in **NP**.

• **Polynomial-time** algorithms for **NP-hard** problems?
  - People tend to think but no one has proved.
NP Hard

• NP-hard
  - The complexity class NP-Hard is the set of problems that are “at least as hard as” the hardest problems in NP.

• Polynomial-time algorithms for NP-hard problems?
  - People tend to think but no one has proved.

• What if ∃ polynomial algorithms for all NP-hard problems?
NP Hard

• **NP-hard**
  - The complexity class **NP-Hard** is the set of problems that are “at least as hard as” the *hardest problems* in **NP**.

• **Polynomial-time** algorithms for **NP-hard** problems?
  - People tend to think but no one has proved.

• **What if ∃ polynomial algorithms for all **NP-hard** problems?**
  - ∃ polynomial algorithms for all problems in **NP** →
NP Hard

• NP-hard
  – The complexity class **NP-Hard** is the set of problems that are “at least as hard as” the **hardest problems** in **NP**.

• **Polynomial-time** algorithms for **NP-hard** problems?
  – People tend to think but no one has proved.

• What if ∃ **polynomial** algorithms for all **NP-hard** problems?
  – ∃ polynomial algorithms for all problems in NP → P = NP
NP Hard

• NP-hard
  - The complexity class NP-Hard is the set of problems that are “at least as hard as” the hardest problems in NP.

• Polynomial-time algorithms for NP-hard problems?
  - People tend to think but no one has proved.

• What if ∃ polynomial algorithms for all NP-hard problems?
  - ∃ polynomial algorithms for all problems in NP → P = NP

• P ≠ NP?
NP Hard

- **NP-hard**
  - The complexity class **NP-Hard** is the set of problems that are “at least as hard as” the hardest problems in **NP**.

- **Polynomial-time algorithms for NP-hard problems?**
  - People tend to think but no one has proved.

- **What if ∃ polynomial algorithms for all NP-hard problems?**
  - ∃ polynomial algorithms for all problems in NP → P = NP

- **P ≠ NP?**
  - NP-hard problems are not solvable in polynomial time,
NP Hard

• NP-hard
  - The complexity class NP-Hard is the set of problems that are “at least as hard as” the hardest problems in NP.

• Polynomial-time algorithms for NP-hard problems?
  - People tend to think but no one has proved.

• What if $\exists$ polynomial algorithms for all NP-hard problems?
  - $\exists$ polynomial algorithms for all problems in NP $\rightarrow$ $P = NP$

• $P \neq NP$?
  - NP-hard problems are not solvable in polynomial time,

• $P = NP$?
NP Hard

- **NP-hard**
  - The complexity class **NP-Hard** is the set of problems that are “at least as hard as” the **hardest problems** in **NP**.

- **Polynomial-time algorithms for NP-hard problems?**
  - People tend to think but no one has proved.

- **What if \( \exists \) polynomial algorithms for all NP-hard problems?**
  - \( \exists \) polynomial algorithms for all problems in NP \( \rightarrow \) **P = NP**

- **P \( \neq \) NP?**
  - NP-hard problems are not solvable in polynomial time,

- **P = NP?**
  - Doesn’t mean the NP-hard problems are solvable in polynomial time.
NP Hard
NP Hard

- Subset sum problem
NP Hard

- Subset sum problem
  - Given a set of integers, does any non-empty subset add up to zero?
NP Hard

- Subset sum problem
  - Given a set of integers, does any non-empty subset add up to zero?
  - Solution?
NP Hard

• Subset sum problem
  - Given a set of integers, does any non-empty subset add up to zero?
  - Solution?

\[
\sum_{i=1}^{n}, \text{where } nC_i = \frac{n!}{i!(n-i)!}
\]
NP Hard

• Subset sum problem
  - Given a set of integers, does any non-empty subset add up to zero?
  - Solution?

\[
\sum_{i=1}^{n}, \text{where } nC_i = \frac{n!}{i!(n - i)!}
\]

• Traveling salesman problem
NP Hard

• Subset sum problem
  - Given a set of integers, does any non-empty subset add up to zero?
  - Solution?
    \[
    \sum_{i=1}^{n}, \text{where } nC_i = \frac{n!}{i!(n-i)!}
    \]

• Traveling salesman problem
  - Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?
NP Hard

- Subset sum problem
  - Given a set of integers, does any non-empty subset add up to zero?
  - Solution?

\[
\sum_{i=1}^{n}, \text{ where } nC_i = \frac{n!}{i!(n-i)!}
\]

- Traveling salesman problem
  - Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?
  - Solution?
NP Hard

• Subset sum problem
  - Given a set of integers, does any non-empty subset add up to zero?
  - Solution?
  \[
  \sum_{i=1}^{n}, \text{where } nC_i = \frac{n!}{i!(n-i)!}
  \]

• Traveling salesman problem
  - Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?
  - Solution?
  \[
  \min \sum_{i=0}^{n} \sum_{j \neq i, j=0}^{n} d_{ij} x_{ij} \quad x_{ij} = \begin{cases} 
  1 & \text{the path goes from city } i \text{ to } j; \\
  0 & \text{otherwise.}
\end{cases}
  \]
NP Complete
NP Complete

- NP complete
NP Complete

• NP complete
  - The complexity class NP-Complete is the set of problems that are both NP and NP-hard.
NP Complete

- NP complete
  - The complexity class **NP-Complete** is the set of problems that are both **NP** and **NP-hard**.

- Any solution to an NP-complete problem can be **verified in polynomial time**; however, there is no known polynomial algorithm to solve the problem.
NP Complete

• NP complete
  - The complexity class **NP-Complete** is the set of problems that are both **NP** and **NP-hard**.

• Any solution to an NP-complete problem can be **verified in polynomial time**; however, there is no known polynomial algorithm to solve the problem.

• If we can solve any NP-complete, then we can solve any problem in NP.
NP Complete

• NP complete
  - The complexity class **NP-Complete** is the set of problems that are both **NP** and **NP-hard**.

• Any solution to an NP-complete problem can be verified in polynomial time; however, there is no known polynomial algorithm to solve the problem.

• If we can solve any NP-complete, then we can solve any problem in NP.

• NP complete problems
NP Complete

• NP complete
  - The complexity class **NP-Complete** is the set of problems that are both **NP** and **NP-hard**.

• Any solution to an NP-complete problem can be verified in polynomial time; however, there is no known polynomial algorithm to solve the problem.

• If we can solve any NP-complete, then we can solve any problem in NP.

• NP complete problems
  - Hamiltonian path problem, clique problem, graph coloring, etc.
NP Complete

• NP complete
  - The complexity class **NP-Complete** is the set of problems that are both **NP** and **NP-hard**.

• Any solution to an NP-complete problem can be **verified in polynomial time**; however, there is no known polynomial algorithm to solve the problem.

• If we can solve any NP-complete, then we can solve any problem in NP.

• NP complete problems
  - Hamiltonian path problem, clique problem, graph coloring, etc.
NP Complete
NP Complete

- Hamiltonian path problem
NP Complete

- Hamiltonian path problem
  - Hamiltonian path: a path in a graph visiting each vertex exactly once.
NP Complete

- Hamiltonian path problem
  - Hamiltonian path: a path in a graph visiting each vertex exactly once.
  - Determine whether a Hamiltonian path exists in a given graph.
NP Complete

• Hamiltonian path problem
  - Hamiltonian path: a path in a graph visiting each vertex exactly once.
  - Determine whether a Hamiltonian path exists in a given graph.

• Clique problem
NP Complete

• Hamiltonian path problem
  - Hamiltonian path: a path in a graph visiting each vertex exactly once.
  - Determine whether a Hamiltonian path exists in a given graph.

• Clique problem
  - Clique: a subset of vertices in a graph that forms a complete graph.
NP Complete

• Hamiltonian path problem
  - Hamiltonian path: a path in a graph visiting each vertex exactly once.
  - Determine whether a Hamiltonian path exists in a given graph.

• Clique problem
  - Clique: a subset of vertices in a graph that forms a complete graph.
  - Find particular cliques in a graph.
NP Complete

• Hamiltonian path problem
  - Hamiltonian path: a path in a graph visiting each vertex exactly once.
  - Determine whether a Hamiltonian path exists in a given graph.

• Clique problem
  - Clique: a subset of vertices in a graph that forms a complete graph.
  - Find particular cliques in a graph.

• Graph coloring problem
NP Complete

- Hamiltonian path problem
  - Hamiltonian path: a path in a graph visiting each vertex exactly once.
  - Determine whether a Hamiltonian path exists in a given graph.

- Clique problem
  - Clique: a subset of vertices in a graph that forms a complete graph.
  - Find particular cliques in a graph.

- Graph coloring problem
  - Color the vertices of a graph such that no two adjacent vertices share the same color.
NP Complete

• Hamiltonian path problem
  - Hamiltonian path: a path in a graph visiting each vertex exactly once.
  - Determine whether a Hamiltonian path exists in a given graph.

• Clique problem
  - Clique: a subset of vertices in a graph that forms a complete graph.
  - Find particular cliques in a graph.

• Graph coloring problem
  - Color the vertices of a graph such that no two adjacent vertices share the same color.
  - Color the edges of a graph such that no two adjacent edges share the same color.
P vs. NP Problem
P vs. NP Problem

- P vs. NP problem
P vs. NP Problem

- P vs. NP problem
  - Whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time.
P vs. NP Problem

• P vs. NP problem
  - Whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time.

• Millennium prize problems
P vs. NP Problem

• P vs. NP problem
  - Whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time.

• Millennium prize problems
  - 7 problems in mathematics stated by the Clay Mathematics Institute.
P vs. NP Problem

• P vs. NP problem
  - Whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time.

• Millennium prize problems
  - 7 problems in mathematics stated by the Clay Mathematics Institute.
  - 6 of the problems remain unsolved.
P vs. NP Problem

• P vs. NP problem
  - Whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time.

• Millennium prize problems
  - 7 problems in mathematics stated by the Clay Mathematics Institute.
  - 6 of the problems remain unsolved.
  - P vs. NP problem is one of them!
P vs. NP Problem

- **P vs. NP problem**
  - Whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time.

- **Millennium prize problems**
  - 7 problems in mathematics stated by the Clay Mathematics Institute.
  - 6 of the problems remain unsolved.
  - **P vs. NP problem** is one of them!
  - A correct solution to any of the problems results in a **US$ 1M**.
P vs. NP Problem

- P vs. NP problem
  - Whether every problem whose solution can be verified in polynomial time can also be solved in polynomial time.

- Millennium prize problems
  - 7 problems in mathematics stated by the Clay Mathematics Institute.
  - 6 of the problems remain unsolved.
  - P vs. NP problem is one of them!
  - A correct solution to any of the problems results in a US$ 1M.
NP Hard but not NP Complete

- Halting problem
  - Given a program and its input, determine if the program will halt at some point.
NP Hard but not NP Complete

- Halting problem
  - Given a program and its input, determine if the program will halt at some point.

\[ P \neq NP \]
NP Hard but not NP Complete

- Halting problem
  - Given a program and its input, determine if the program will halt at some point.

  \[ P \neq NP \]
NP Hard but not NP Complete

- Halting problem
  - Given a program and its input, determine if the program will halt at some point.

\[ P \neq NP \]
NP Hard but not NP Complete

- Halting problem
  - Given a program and its input, determine if the program will halt at some point.

\[ P \neq \text{NP} \]

[Diagram showing the relationship between P, NP, and NP-Hard]
NP Hard but not NP Complete

- Halting problem
  - Given a program and its input, determine if the program will halt at some point.

\[ P \neq NP \]

NP-Hard

NP-C

NP

P
NP Hard but not NP Complete

- Halting problem
  - Given a program and its input, determine if the program will halt at some point.

\[ P \neq NP \quad \text{and} \quad P = NP \]
NP Hard but not NP Complete

- Halting problem
  - Given a program and its input, determine if the program will halt at some point.

\[ P \neq NP \quad \text{NP-Hard} \quad \text{NP-C} \quad \text{NP} \quad P = NP \]

\[ P = NP = \text{NP-C} \]
NP Hard but not NP Complete

- Halting problem
  
  Given a program and its input, determine if the program will halt at some point.

\[
P \neq \text{NP}
\]

\[
\text{NP-Hard}
\]

\[
\text{NP-C}
\]

\[
\text{NP}
\]

\[
P
\]

\[
P = \text{NP}
\]

\[
\text{NP-Hard}
\]

\[
P = \text{NP} = \text{NP-C}
\]