Q01：The difference between pseudo-polynomial and polynomial

- prompt：**What is the difference between pseudo-polynomial and polynomial?**
- model_name：**ChatGPT**
- model_version：**ChatGPT-4o**
- response：

The main difference between **pseudo-polynomial time** and **polynomial time** lies in how the complexity depends on the size of the input:

1. **Polynomial Time**
   If the time complexity of an algorithm can be expressed as a polynomial function of the size of the input (usually represented by log(n) , the number of bits needed to represent the input), such as O(poly(log(n))), the algorithm is said to have polynomial time complexity. It depends on the length of the input, not the numerical value itself.
2. **Pseudo-Polynomial Time**
   If the time complexity depends on the numerical value of the input (n) rather than the size of its representation log(n), it is called pseudo-polynomial. For example, an algorithm with a complexity of $O(n^k)$ is pseudo-polynomial because it depends on the value of n instead of log(n).

Key Difference:

- **Polynomial time** complexity grows as a polynomial of the input size, making it scalable.
- **Pseudo-polynomial time** complexity grows as a polynomial of the input value, which becomes inefficient as the input size increases.

Examples:

- Polynomial time: The time complexity of fast exponentiation is O(log(n)), as it depends on the input's bit-length.
- Pseudo-polynomial time: The time complexity of trial division is $O(\sqrt{n})$, as it depends on the numerical value of n.

Q02：Pseudo polynomial judgment algorithm

- prompt：**How to determine whether the complexity of an algorithm is pseudo-polynomial?**
- model_name：**ChatGPT**
- model_version：**ChatGPT-4o**
- response：

To determine whether the complexity of an algorithm is pseudo-polynomial, consider the following aspects:

1. **Representation of Input Size**
   First, check if the input size is represented in terms of the number of bits log(n). A pseudo-polynomial algorithm's complexity depends on the numerical value n of the input rather than its bit-length log(n).
2. **Form of Time Complexity**
   If the algorithm's time complexity can be expressed as a polynomial function of the input value n, such as $O(n^k)$ (where n is the actual value of the input), rather than as a polynomial of log(n), such as $O((log(n))^k)$, the algorithm is likely pseudo-polynomial.
3. **Comparison of Input Size and Actual Complexity**
   As the input value increases, the complexity of a pseudo-polynomial algorithm grows much faster, but it may appear similar to polynomial algorithms for small input values. Analyzing the growth pattern helps distinguish pseudo-polynomial complexity from true polynomial complexity.
4. **Specific Dependency on Input**
   Check if the algorithm depends on the numerical value of the input. For example, trial division has a complexity of $O(\sqrt{n})$, which depends on the numerical value, making it pseudo-polynomial. In contrast, fast exponentiation has a complexity of O(log(n)), depending on the input bit-length, classifying it as polynomial time.

Q03：0-1 backpack problem and P problem

- prompt：**Is the 0-1 knapsack problem a P problem? Why?**
- model_name：**ChatGPT**

- model_version：**ChatGPT-4o**
- response：

The 0-1 knapsack problem is **not** a P problem because it belongs to the class of NP problems. Specifically, the 0-1 knapsack problem is a combinatorial optimization problem, and its complexity is NP-complete in general. This means that while a given solution can be verified in polynomial time, no algorithm has been found that can solve all instances of the problem in polynomial time. However, when treated as a weighted problem with integer weights and values, there are dynamic programming algorithms that solve it in pseudo-polynomial time, but this does not change its classification as an NP problem.

Q04：Prime number testing problem and pseudo polynomial problem

- prompt：**Primality Testing Problem: Given any positive integer n, check whether n has an integer factor greater than 1. Please provide an algorithm with pseudo-polynomial complexity for this problem and explain the reason behind its complexity.**
- model_name：**ChatGPT**
- model_version：**ChatGPT-4o**
- response：

The **primality testing problem** can be solved using trial division within pseudo-polynomial time. Below is the pseudo-polynomial algorithm and its explanation:

Pseudo-Polynomial Algorithm:

1. Given a positive integer n.
2. Check whether n is divisible by any integer in the range from 2 to $\sqrt{n}$:

- If $n\%d$==0(where d is the divisor being tested), then n is not a prime.
- If no divisor is found, n is a prime number.

3. Algorithm Complexity:

- The range of testing is $\sqrt{n}$, and each division operation takes constant time. Thus, the time complexity is O($\sqrt{n}$).

Reason for Pseudo-Polynomial Nature:

The definition of pseudo-polynomial time refers to algorithms whose complexity depends on the **numerical value of the problem** size rather than the number of digits (input size). In the case of trial division:

1. When a positive integer n is given as input, its input size is log(n), i.e., the number of bits required to represent n.
2. he time complexity of trial division is O($\sqrt{n}$), which depends on the actualnumerical value of n, not its bit-length log(n).
3. If the complexity were a polynomial function of log(n) (e.g., O(poly(log(n)))), the algorithm would be in strictly polynomial time. However, in trial division, the dependence on $\sqrt{n}$ leads to exponential growth relative to the input size.
4. Thus, while trial division is efficient for small numerical values of n, its complexity scales exponentially with the input size log(n), categorizing it as pseudo-polynomial time.

Q05：Research on whether P problem is equal to NP problem

- prompt：**What are the latest and most cutting-edge developments in research on whether P equals NP?**
- model_name：**ChatGPT**
- model_version：**ChatGPT-4o**
- response：

The P vs. NP problem is one of the most significant unsolved questions in computer science, addressing whether every problem whose solution can be verified in polynomial time (NP) can also be solved in polynomial time (P). Despite being proposed over 50 years ago, the question remains unanswered. In recent years, researchers have made subtle but important progress in this field.

The Rise of Meta-Complexity Theory

Researchers have found that proving the computational difficulty of certain problems is itself an extremely challenging task. This has led to the study of "meta-complexity," which focuses on understanding the complexity of computational problems by analyzing computational

complexity itself. While not directly solving the P vs. NP problem, meta-complexity provides new insights and lays a foundation for future research.

Advancements in Quantum Computing

Quantum computing is considered a potential game-changer for the P vs. NP problem. In 2023, researchers made significant upgrades to Shor’s algorithm, nearly three decades after its invention. This algorithm allows for efficient integer factorization on quantum computers, posing a potential threat to current encryption systems. However, practical quantum computers are still not available, and their ability to solve NP-complete problems remains unproven.

AI Applications in Theoretical Proofs

Recently, researchers from institutions such as Microsoft Research Asia and Peking University employed GPT-4 in a rigorous Socratic-style reasoning process, resulting in the conclusion that "P≠NP." While not a formal mathematical proof, this experiment showcased the potential of large language models to assist in research and problem-solving.

Other Advances in Computational Complexity

In 2023, computer scientists achieved breakthroughs in algorithm development. For example, they devised new algorithms for finding the shortest paths in graphs with negative weights at speeds comparable to those for positive weights. Progress has also been made in the group isomorphism problem, with faster algorithms to determine when two mathematical structures are identical. These advances, while not directly addressing the P vs. NP question, provide valuable tools and methods for understanding computational complexity.

Conclusion

While the P vs. NP problem remains unsolved, recent research has made significant strides in areas such as meta-complexity theory, quantum computing, AI applications, and algorithm design. These developments offer new perspectives and approaches to tackling this fundamental question. With continued exploration and innovation, further breakthroughs in this field are anticipated in the future.
