Documentation For Convert a Number to Hexadecimal

Problem Statement

The problem requires converting a given **32-bit signed integer** into its **hexadecimal representation**. The conversion should follow the **two's complement method** for negative numbers, ensuring that all letters in the result are lowercase (0-9 and a-f). Additionally, leading zeros should be avoided, except when the number itself is zero. Built-in library functions like hex() are not allowed, meaning the solution must manually perform the conversion.

Intuition

Since hexadecimal is a **base-16** number system, each digit corresponds to **4 bits**. The key idea is to extract **4-bit chunks** from the number, map them to their hexadecimal equivalent, and construct the result iteratively. For negative numbers, the **two's complement representation** ensures proper conversion by treating them as **32-bit unsigned integers**.

Key Observations

To convert an integer to hexadecimal, we repeatedly extract the last **4 bits** using the bitwise operation num & 15. This isolates the least significant hex digit, which is then mapped to its corresponding character. The number is then right-shifted by **4 bits** to remove the processed portion. For negative numbers, we first convert them into their **32-bit unsigned equivalent** by adding 2^{32}. The extracted hex digits are collected in **reverse order**, so the final output is obtained by reversing the collected characters.

Approach

First, we handle the **special case** where the input is zero, returning "0" immediately. Next, we define a **hexadecimal mapping** containing the characters 0-9 and a-f. If the input number is **negative**, we convert it into an unsigned **32-bit representation** by adding 2^{32}. We then iteratively extract **4-bit chunks**, map them to hexadecimal characters, and accumulate the result. The process continues until all significant bits have been processed. Since the digits are collected in reverse order, we return the final string as is.

Edge Cases

There are several important edge cases to consider. If the input is **zero**, the output must be "0". For the **smallest negative number** -2^{31}, the result should be "80000000", as it follows the **two's complement rule**. The **largest positive number** 2^{31} - 1 should convert to "7fffffff". Handling **negative numbers** like -1 is critical since its **two's complement representation** results in "ffffffff". Special cases like **powers of 16** (e.g., 16 → "10", 256 → "100") should also be considered.

Complexity Analysis

The approach operates in **constant time**, making it **O(1)**. Since a **32-bit integer** can be fully represented in at most **8 hexadecimal digits**, the loop runs at most **8 times**, making it highly efficient. The space complexity is also **O(1)** because only a fixed amount of extra space is used for storing the hexadecimal mapping and result string.

Alternative Approaches

An alternative method involves using **recursion** instead of iteration. This approach recursively extracts the last **4 bits**, converts them to hexadecimal, and appends the result in reverse order. However, recursion introduces **function call overhead** and increases **stack space usage**, making it **less optimal** than the iterative method.

Final Thoughts

This approach efficiently converts integers to hexadecimal using **bitwise operations** and **manual mapping** without relying on built-in functions. The method correctly handles **negative numbers** using **two's complement representation** and ensures **leading zeros** are avoided. The **constant time complexity** and **low space requirements** make it an optimal solution for this problem. 🚀