ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN
CS112.P11.KHTN

BTVN NHÓM 4

Nhóm 7:
Hoàng Đức Dũng - 23520328
Nguyễn Văn Hồng Thái - 23521418

Giảng viên :
Nguyễn Thanh Sơn

Ngày 5 tháng 12 năm 2024

Mục lục

1 Bài 1: Đối Kháng

- Trò chơi này thuộc loại trò chơi có tổng bằng 0. Vì có người thắng và có thua và tổng lợi ích luôn bằng 0.

1.1 $p <= 10$

- Chúng ta có thể quay lui tất cả các trường hợp có thể xảy ra.
- Phương pháp thiết kế thuật toán: Vét cạn
- Độ phức tạp: $O(2^p)$
- Mã giả

```
def backtracking_can_win(p):
def dfs(p):
    if p == 0:
        return False
    if p % 2 == 0:
        return not dfs(p // 2)
    else:
        return not dfs(p + 1) or not dfs(p - 1)
    return dfs(p)
```

1.2 $p <= 10^6$

- Phương pháp thiết kế thuật toán: Quy hoạch động
- Ý tưởng:
  - Tương tự như trên, sử dụng DP để lưu trữ các trạng thái thắng thua cho từng giá trị p từ 1 đến 10^6.
  - Cập nhật trạng thái cho mỗi p dựa trên các thao tác có thể thực hiện
- Độ phức tạp: $O(p)$
- Mã giả

```
def game_result(p):
    dp = [False] * (p + 1)
    dp[0] = False

for i in range(1, p + 1):
    if i % 2 == 0:
        dp[i] = not dp[i // 2]
    else:
        dp[i] = not dp[i - 1] or not dp[i + 1]
return "YES" if dp[p] else "NO"
```

1.3 $p <= 10^{18}$

- Ta có nhận xét:
  - Nếu p lẻ thì A luôn thắng
  - Nếu p chẵn thì sẽ có dạng $2^k.x$
    * Nếu k lẻ thì A luôn thắng
    * Nếu k chẵn thì A luôn thua
- Độ phức tạp: $O(log(p))$
- Mã giả:

```
def can_win(p):
cnt = 0
while p % 2 == 0:
    cnt += 1
    p //= 2
    if cnt % 2 == 1:
        return True
    else:
        return False
```

2 Bài 2: Trò chơi đồng xu

- Trò chơi này thuộc loại trò chơi có tổng bằng 0. Vì có người thắng và có thua và tổng lợi ích luôn bằng 0.

2.1 $n <= 1000$

- Phương pháp thiết kế thuật toán: Quy hoạch động
- Ý tưởng:
  - Khởi tạo một mảng dp với kích thước n + 1, trong đó dp[i] biểu thị liệu số đồng xu còn lại là i thì người chơi có thể thắng nếu chơi tối ưu hay không.
  - Duyệt từ 1 đến n và xác định liệu có thể đưa đối thủ vào trạng thái thua hay không
  - Đối với mỗi giá trị k, tính số lượng giá trị của k sao cho A có thể chắc chắn thắng.
- Độ phức tạp: $O(n.k)$
- Mã giả:

```
def count_win(n):
    dp = [False] * (n + 1)
    count = 0
    for k in range(1, n + 1):
```

```python
        for i in range(1, k + 1):
            if n - i >= 0 and not dp[n - i]:
                dp[n] = True
                break
        if dp[n]:
            count += 1
    return count
```

2.2 $n <= 10^{18}$

- Ta có nhận xét
  - Nếu n chia hết cho k + 1 thì người chơi đi sau luôn thắng.
  - Khi đó, chỉ cần đếm các k (k > 0) sao cho n mod (k + 1) = 0 sau đó lấy n - k.
  - Nhưng mà vì n quá lớn nên ta sẽ áp dụng định lí Fermat nhỏ để tính số lượng các ước cho nhanh.
- Độ phức tạp: $O(n^{1/3})$
- Mã giả:

```python
import math
import random

def indian_multiplication(a, b, mod):
    if b == 0:
        return 0
    half = indian_multiplication(a, b // 2, mod) % mod
    if b % 2 == 1:
        return (half + half + a) % mod
    else:
        return (half + half) % mod
def modular_exponentiation(a, b, mod):
    if b == 0:
        return 1
    half = modular_exponentiation(a, b // 2, mod) % mod
    product = indian_multiplication(half, half, mod)
    if b % 2 == 1:
        return indian_multiplication(product, a, mod)
    else:
        return product
def fermat_checking(n, k=50):
    if n < 4:
        return n == 2 or n == 3
    if n % 2 == 0:
        return False
    for _ in range(k):
        a = random.randint(2, n - 2)
        if modular_exponentiation(a, n - 1, n) != 1:
            return False
```

```
    return True
def eratosthenes_sieve(max_value):
    is_prime = [True] * (max_value + 1)
    is_prime[0] = is_prime[1] = False
    for i in range(2, int(math.sqrt(max_value)) + 1):
        if is_prime[i]:
            for j in range(i * i, max_value + 1, i):
                is_prime[j] = False
    primes = [i for i in range(2, max_value + 1) if is_prime[i]]
    return primes
def solution(n):
    primes = eratosthenes_sieve(1000000)
    res = 1
    for p in primes:
        if p * p * p > n:
            break
        cnt = 0
        while n % p == 0:
            n //= p
            cnt += 1

        res *= (cnt + 1)
    if fermat_checking(n):
        res *= 2
    else:
        squaroot = int(math.sqrt(n))
        if squaroot * squaroot == n and fermat_checking(squaroot):
            res *= 3
        elif n != 1:
            res *= 4
    print(n - (res - 1))
if __name__ == "__main__":
    n = int(input())
    solution(n)
```