# 고급문제해결 발표

컴퓨터학부 2019112543 김승준

# TEST THE COLLATZ CONJECTURE

## 콜라츠 추측이란?

어떤 자연수 n이 주어졌을 때,
n이 짝수이면 2로 나눕니다.
n이 홀수이면 3을 곱하고 1을 더합니다.
위의 과정을 n이 1이 될 때까지 반복

이 추측은 임의의 자연수에 대해 항상 성립한다는 가설이지만, 아직 증명되지 않은 문제로 남아있다.

# 콜라츠 추측 예

- 자연수 n = 11 이라고 가정

1. 11은 홀수 이기 때문에 11 *3 + 1 = 34
2. 34는 짝수이기 때문에 34 /2 = 17
3. 17은 홀수 이기 때문에 17 *3 + 1 = 52

4~ 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

결국 1로 수렴한다.

# 첫 n개의 자연수에 대해 콜라츠의 추측

1. n 이 충분히 작은 숫자라면 Naive 하게 하나씩 콜라츠 함수 적용

```
def collatz_conjecture(n):
    for num in range(1, n + 1):
        sequence = [num]
        while num != 1:
            if num % 2 == 0:
                num = num // 2
            else:
                num = num * 3 + 1
            sequence.append(num)

        print(f"Collatz sequence for {sequence[0]}: {sequence}")
```

# 좀더 효율적인 방법

1. 이미 1에 수렴한다고 증명된 모든 수를 저장하여 해당 수에 도달하면 바로 1에 도달할 것으로 가정
2. 짝수를 건너뛰기
3. k까지 모든 수를 테스트한 경우, k보다 작거나 같은 수에 도달하면 바로 1로 수렴한다 가정
4. 탐색 집합을 분할하고 병렬로 사용하여 집합 탐색

# 개선한 방식 1

```python
def test_collatz_conjecture(n):
    verified_numbers = set()
    for i in range(3, n+1, 2):
        sequence = set()
        test_num = i
        while test_num >= i:
            if test_num in sequence:
                return False
            if test_num % 2 != 0:
                if test_num in verified_numbers:
                    break
                next_test_num = 3 * test_num + 1
                if next_test_num <= test_num:
                    raise ArithmeticError(f"Collatz sequence overflow for {i
                test_num = next_test_num
            else:
                test_num //= 2
            sequence.add(test_num)
    return True
```

# 병렬 처리 방법

- 멀티스레드 프로그램 사용
- 스레딩 오버헤드를 최소화
- [1, N]을 n개의 동일한 크기의 범위로 나누고, 스레드 i가 k번째 하위 범위를 처리
- 주의 할점 : 스레드 오버헤드

```python
from concurrent.futures import ThreadPoolExecutor

def test_collatz_conjecture(n):
    verified_numbers = set()

    def collatz_worker(start, end):
        for i in range(start, end+1):
            sequence = set()
            test_num = i
            while test_num >= i:
                if test_num in sequence:
                    return False
                if test_num % 2 != 0:
                    if test_num in verified_numbers:
                        break
                    next_test_num = 3 * test_num + 1
                    if next_test_num <= test_num:
                        raise ArithmeticError(f"Collatz sequence overflow for
                    test_num = next_test_num
                else:
                    test_num //= 2
                sequence.add(test_num)
        return True

    with ThreadPoolExecutor(max_workers=n) as executor:
        chunk_size = (n + executor._max_workers - 1) // executor._max_workers
        futures = []
        for i in range(1, n+1, chunk_size):
            end = min(i + chunk_size - 1, n)
            futures.append(executor.submit(collatz_worker, i, end))
        for future in futures:
            if not future.result():
                return False

    return True
```