계명대학교
KEIMYUNG UNIVERSITY

Data Structure

계명대학교
KEIMYUNG UNIVERSITY

■ 과 목 명 자 료 구 조
■ 담 당 교 수 주 홍 택
■ 제 출 일 2024-03-27
■ 학 과 컴퓨터공학과
■ 학 번 5906945
■ 성 명 방 수 종

1) 하노이탑 문제를 순환방법이 아닌 반복적인 방법으로 구현 하는 방법

```c

// 원판 이동 함수
void moveDisk(char source, char target) {
    printf("Move disk from %c to %c\n", source, target);
}

// 하노이 탑 알고리즘 (반복적 방법)
void hanoiIterative(int numDisks, char source, char auxiliary, char target) {
    // 원판 이동 방향 결정
    char temp;
    if (numDisks % 2 == 0) {
        temp = auxiliary;
        auxiliary = target;
        target = temp;
    }

    // 총 이동 횟수 계산
    int totalMoves = (1 << numDisks) - 1; // 2^numDisks - 1

    // 각 이동을 반복적으로 수행
    for (int move = 1; move <= totalMoves; move++) {
        if (move % 3 == 1) {
            moveDisk(source, target);
        } else if (move % 3 == 2) {
            moveDisk(source, auxiliary);
        } else if (move % 3 == 0) {
            moveDisk(auxiliary, target);
        }
    }
}

int main() {
    int numDisks = 3;
    char source = 'A';
    char auxiliary = 'B';
    char target = 'C';

    // 하노이 탑 문제 해결
    hanoiIterative(numDisks, source, auxiliary, target);

    return 0;
}
```

-변수-

Source : 원판을 출발지로 하는 탑을 나타내는 변수
auxiliary : 원판을 보조로 사용하는 탑을 나타내는 변수
target : 원판을 목표로 하는 탑을 나타내는 변수
numDisks : 원판의 개수를 나타내는 변수
temp : 보조적인 탑 (auxiliary)과 목표탑(target)의 역할을 변경할 때 사용할 임시 변수

```
// 원판 이동 방향 결정
char temp;
if (numDisks % 2 == 0) {
    temp = auxiliary;
    auxiliary = target;
    target = temp;
}
```

- 원판 이동 방향 결정 -

원판의 개수가 홀수인지 짝수인지 확인. 만약 원판의 개수가 짝수 이면 보조적인 탑(auxiliary)과 목표탑 (target)의 역할을 서로 바꾼다. 즉, 보조적인 탑(auxiliary)의 역할을 목표 탑(target)의 역할을 보조적인 탑(auxiliary)으로 변경.

이렇게 함으로써 하노이탑 문제를 푸는과정에서 보조적인 탑의 역할을 변경함으로써 원판의 이동방향을 조절.

```
// 총 이동 횟수 계산
int totalMoves = (1 << numDisks) - 1; // 2^numDisks - 1
```

-총 이동 횟수 계산-

원판이 한 번 이동할 때마다 전체 이동 횟수가 1 씩 증가하므로, n 개의 원판이 있을 때의 전체 이동 횟수는 2^n-1 값과 같다.

```
// 각 이동을 반복적으로 수행
for (int move = 1; move <= totalMoves; move++) {
    if (move % 3 == 1) {
        moveDisk(source, target);
    } else if (move % 3 == 2) {
        moveDisk(source, auxiliary);
    } else if (move % 3 == 0) {
        moveDisk(auxiliary, target);
    }
}
```

-반복문-

'move'변수를 이용하여 3 으로 나눈 나머지를 계산. 나머지가 1 이면 첫 번째 원판으로 이동, 2 이면 두 번째 원판을 이동시키고, 0 이면 세 번째 원판을 이동한다.

즉, 각 이동은 3 개의 순환 패턴으로 반복되는데, 이를 나누어서 원판을 이동시키는 로직이다.

2) 재귀적인 해결 방법(알고리즘)을 반복적인 해결 방법(알고리즘)으로 변환하는 일반적인 방법

첫번째_꼬리 물기 최적화 : 함수 호출의 스택 프레임을 재사용하여 재귀적인 호출을 반복적인 형태로 최적화한다. 일반적으로 꼬리 물기 최적화는 함수의 마지막에서 자기 자신을 호출하는 재귀적인 호출에 적용한다. 이 때 함수가 반환할 때 추가적인 계산 없이 바로 자신을 호출한다.

두 번째_ 동적 프로그래밍 기법(Dynamic Programming_DP) 적용 : 중복 계산을 피하고 메모리에 중간 결과를 저장하여 효율적인 알고리즘을 설계하는 기법입니다. DP 가 적용되기 위해서는 2 가지 조건이 있는데 1) Overlapping Subproblems(겹치는 부분 문제), 2) Optimal Substructure(최적 부분 구조) 이렇게 있다 먼저 1)Overlapping Subproblems 에서 우선 DP 는 기본적으로 문제를 나누고 그 문제의 결과 값을 재활용해서 전체 답을 구한다. 그래서 동일한 문제들이 반복하여 나타나는 경우에 사용이 가능 하다. 즉, DP 는 부분 문제의 결과를 저장하여 재 계산하지 않을 수 있어야 하는데, 해당 부분 문제가 반복적으로 나타나지 않는다면 재사용이 불가능하니 부분 문제가 중복되지 않는 경우에는 사용 할 수가 없다.
2) Optimal Substructure(최적 부분 구조)에서 부분 문제의 최적 결과 값을 사용해 전체 문제의 최적 결과를 낼 수 있는 경우를 의미한다. 그래서 특정 문제의 정답은 문제의 크기에 상관없이 항상 동일하다. 이 조건에 만족을 하면 DP 를 적용 할 수 있다.

-마무리-

이 과제를 하며 순환 알고리즘에서 반복 알고리즘으로 변환 할 때 변수가 더 필요로 하고 순환 알고리즘이 반복 알고리즘으로 변환 할 때 더 코드가 복잡해진다는 것을 알게 되었습니다.