



## 4) 고급 알고리즘



매 강의 강의자료 시작에 PDF파일을 올려두었어요!

### ▼ PDF 파일

#### [수업 목표]

- 이 강의에서는 동적 프로그래밍, 그리디 알고리즘, 분할 정복 등 고급 알고리즘에 대해 배우게 됩니다.
- 각 알고리즘의 시간 복잡도와 공간 복잡도를 분석하고, 실제 코드 작성과 분석 연습을 진행합니다.

#### [목차]

01. 동적 프로그래밍 ( Dynamic Programming ).

02. 그리디 알고리즘 ( Greedy Algorithm ).

03. 분할 정복 ( Divide and Conquer ).



모든 토글을 열고 닫는 단축키

Windows : **Ctrl** + **alt** + **t**

Mac : **⌘** + **⌥** + **t**

### 01. 동적 프로그래밍 ( Dynamic Programming )



작은 문제 단위로 쪼개서 반복하여 푸는 방식

### ▼ 1) 동적 프로그래밍이란?

- 동적 프로그래밍은 큰 문제를 작은 하위 문제로 분할하여 푸는 접근 방식입니다.
- 작은 하위 문제의 해결 방법을 계산하여 저장하고, 이를 이용하여 큰 문제의 해결 방법을 도출합니다. 이러한 저장 과정을 "메모이제이션(Memoization)"이라고 합니다.
- 동적 프로그래밍은 중복되는 하위 문제들을 효율적으로 해결하기 위해 사용됩니다.
- 일반적으로 재귀적인 구조를 가지며, 하향식(Top-down)과 상향식(Bottom-up) 두 가지 방법으로 구현할 수 있습니다.
- 동적 프로그래밍은 최적 부분 구조와 중복 부분 문제의 특징을 가진 문제들을 효과적으로 해결할 수 있습니다.

### ▼ 2) 동적 프로그래밍 실습 예제

```
// 문제: 피보나치 수열의 n번째 항을 구하는 함수를 작성하세요.
public int Fibonacci(int n)
{
    int[] dp = new int[n+1];
    dp[0] = 0;
    dp[1] = 1;

    for(int i = 2; i <= n; i++)
    {
        dp[i] = dp[i-1] + dp[i-2];
    }

    return dp[n];
}
```

#### 해설

- 위 코드는 피보나치 수열의 n번째 항을 구하는 문제를 해결하는 C# 코드입니다.
- 이 문제를 해결하는 데는 동적 프로그래밍 방법을 사용하였으며, 시간 복잡도는  $O(n)$ 이고 공간 복잡도는  $O(n)$ 입니다.

## 02. 그리디 알고리즘 ( Greedy Algorithm )



## 현재에 집중하는 알고리즘

### ▼ 1) 그리디 알고리즘이란?

- 그리디 알고리즘은 각 단계에서 가장 최적의 선택을 하는 알고리즘입니다.
- 각 단계에서 가장 이익이 큰 선택을 하여 결과적으로 최적해에 도달하려는 전략을 따릅니다.
- 그리디 알고리즘은 지역적인 최적해를 찾는데 초점을 맞추기 때문에 항상 전역적인 최적해를 보장하지는 않습니다.
- 그리디 알고리즘은 간단하고 직관적인 설계가 가능하며, 실행 시간이 매우 빠른 편입니다.
- 그리디 알고리즘은 최적해를 찾는 문제에 사용되는 경우가 많지만, 일부 문제에서는 최적해를 보장하지 않을 수 있습니다.

### ▼ 2) 그리디 알고리즘 실습 예제

```
// 문제: 주어진 동전들로 특정 금액을 만드는데 필요한 최소 동전 수를 구하는 함수를 작성하세요.
public int MinCoins(int[] coins, int amount)
{
    Array.Sort(coins);
    int count = 0;

    for(int i = coins.Length - 1; i >= 0; i--)
    {
        while(amount >= coins[i])
        {
            amount -= coins[i];
            count++;
        }
    }

    if(amount > 0) return -1;

    return count;
}
```

#### 해설

- 위 코드는 주어진 동전들로 특정 금액을 만드는데 필요한 최소 동전 수를 구하는 문제를 해결하는 C# 코드입니다.

- 이 문제를 해결하는 데는 그리디 알고리즘을 사용하였으며, 시간 복잡도는 동전의 개수에 비례하는  $O(n)$ 이고, 공간 복잡도는  $O(1)$ 입니다.

### 03. 분할 정복 ( Divide and Conquer )



작은 부분부터 착실히 해결해 나가자

#### ▼ 1) 분할 정복 이란?

- 큰 문제를 작은 부분 문제로 분할하므로 문제의 크기에 따른 성능 향상이 가능합니다.
- 재귀적인 구조를 가지므로 문제 해결 방법의 설계가 단순하고 직관적입니다.
- 분할된 부분 문제들은 독립적으로 해결되므로 병렬 처리에 유리합니다.
- 하지만 문제를 분할할 때 발생하는 부분 문제들 간의 중복 계산이 발생할 수 있습니다. 이런 경우에는 중복 계산을 피하기 위한 메모이제이션 등의 기법을 활용하여 성능을 향상시킬 수 있습니다.

#### ▼ 2) 분할 정복 실습 예제

```
// 문제: 주어진 배열을 정렬하는 함수를 작성하세요. (퀵 정렬 사용)
public void QuickSort(int[] arr, int low, int high)
{
    if(low < high)
    {
        int pi = Partition(arr, low, high);

        QuickSort(arr, low, pi - 1); // Before pi
        QuickSort(arr, pi + 1, high); // After pi
    }
}

int Partition(int[] arr, int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for(int j = low; j <= high - 1; j++)
    {
        if(arr[j] < pivot)
        {
            i++;
            Swap(arr, i, j);
        }
    }
}
```

```

        Swap(arr, i + 1, high);
        return (i + 1);
    }

    void Swap(int[] arr, int a, int b)
    {
        int temp = arr[a];
        arr[a] = arr[b];
        arr[b] = temp;
    }


```

## 해설

- 위 코드는 주어진 배열을 정렬하는 문제를 해결하는 C# 코드입니다. 여기서는 퀵 정렬 알고리즘을 사용하였습니다.
- 이 문제를 해결하는 데는 분할 정복 방법을 사용하였으며, 시간 복잡도는 평균적으로  $O(n \log n)$ 이고, 최악의 경우(이미 정렬된 배열 등)에는  $O(n^2)$ 가 됩니다. 공간 복잡도는  $O(\log n)$ 입니다. (재귀 호출 스택의 크기를 고려)

이전 강의

다음 강의

 3) 탐색 알고리즘

19강 문제 해결 전략과 실전 연습