5) 문제 해결 전략과 실전 연습

매 강의 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- 이 강의에서는 알고리즘 문제를 해결하는 전략을 배우고, 실전 연습을 진행하게 됩니다.
- 실제 코딩 테스트에서 나올 수 있는 문제들 확인

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 문제 해결 전략

1. **문제 이해**: 문제를 정확히 이해하고 요구사항을 파악하는 것이 중요합니다. 문제 설명을 꼼꼼히 읽고, 입력과 출력의 형식을 이해하고 분석해야 합니다.

2. **예제와 테스트 케이스**: 문제의 예제와 추가적인 테스트 케이스를 활용하여 문제를 이해하고 해결 방법을 검증해야 합니다. 예제와 테스트 케이스는 문제의 조건과 제약을 이해하는 데 도움을 줄 수 있습니다.
3. **알고리즘 설계**: 문제를 해결하기 위한 알고리즘을 설계해야 합니다. 문제의 특성에 맞는 알고리즘을 선택하고, 알고리즘의 구현 방법과 시간 복잡도를 고려해야 합니다.
4. **코드 작성**: 알고리즘을 기반으로 코드를 작성해야 합니다. 코드는 가독성이 좋고, 문제의 요구사항을 정확히 반영해야 합니다. 변수명을 명확하게 지어 가독성을 높이고, 주석을 추가하여 코드를 설명하는 것도 좋은 습관입니다.
5. **효율성**: 문제의 제약 조건과 입력 크기에 따라 알고리즘의 효율성을 고려해야 합니다. 최적화 기법을 사용하고, 시간 복잡도와 공간 복잡도를 최대한 줄이는 방향으로 코드를 작성해야 합니다.
6. **디버깅과 테스트**: 코드를 작성한 후에는 디버깅을 통해 오류를 찾고 수정해야 합니다. 테스트 케이스를 활용하여 코드의 정확성을 검증하고, 예외 상황을 고려하여 코드를 완성해야 합니다.
7. **시간 관리**: 코딩 테스트는 제한된 시간 안에 문제를 해결해야 하는 것이 특징입니다. 따라서 시간을 효과적으로 관리하고, 문제에 맞는 효율적인 접근 방법을 선택하는 능력이 필요합니다.
8. **연습과 경험**: 코딩 테스트는 많은 연습과 경험이 필요한 분야입니다. 다양한 유형의 문제에 노출되고, 해결 방법을 익히며 자신의 실력을 향상시켜야 합니다. 코딩 테스트 관련 문제를 많이 풀고 다른 사람들의 풀이를 학습하는 것도 좋은 방법입니다.

02. 실전 연습 문제

1. **백준 온라인 저지** (**https://www.acmicpc.net/**): 다양한 난이도의 알고리즘 문제를 제공하며, 많은 사용자들과 소통할 수 있는 커뮤니티도 제공합니다.
2. **프로그래머스** (**https://programmers.co.kr/**): 코딩 테스트 연습을 위한 문제들을 다양한 난이도로 제공하고 있습니다. 실제 취업 시험에서도 출제되는 문제들을 포함하고 있습니다.
3. **바킹독 (**https://blog.encrypted.gg/): 알고리즘 강의 다수 제공하고 있습니다.
4. **LeetCode** (**https://leetcode.com/**): 알고리즘 문제와 코딩 테스트 문제를 다양한 난이도로 제공하고 있으며, 실제 기업 코딩 인터뷰에서 출제되는 문제들을 포함하고 있습니다.

03. 코딩 테스트나 알고리즘 문제의 종류

1. **탐색과 정렬**: 배열, 리스트, 문자열 등의 데이터에서 특정 값을 찾거나 정렬하는 문제입니다. 선형 탐색, 이진 탐색, 퀵 정렬, 병합 정렬 등의 알고리즘이 주로 활용됩니다.
2. **그래프**: 그래프 구조를 활용하여 문제를 해결하는 문제입니다. 최단 경로, 신장 트리, 네트워크 연결 등의 문제가 있을 수 있으며, 대표적인 알고리즘으로는 DFS(Depth-First Search), BFS(Breadth-First Search), Dijkstra, 크루스칼, 프림 등이 있습니다.
3. **동적 프로그래밍**: 큰 문제를 작은 하위 문제로 분할하여 해결하는 동적 프로그래밍 알고리즘을 활용하는 문제입니다. 피보나치 수열, 최장 공통 부분 수열, 0-1 배낭 문제 등이 있습니다.
4. **그리디 알고리즘**: 각 단계에서 가장 최적인 선택을 하는 알고리즘으로, 지역적 최적해를 찾는 문제입니다. 거스름돈 문제, 회의실 배정, 작업 스케줄링 등이 있습니다.
5. **분할 정복**: 문제를 작은 부분으로 분할하여 해결하는 분할 정복 알고리즘을 사용하는 문제입니다. 퀵 정렬, 병합 정렬, 이진 탐색 등이 있습니다.
6. **동적 그래프**: 그래프 구조에서 동적인 변화를 다루는 문제입니다. 플로이드-와샬 알고리즘, 벨만-포드 알고리즘 등이 있습니다.
7. **문자열 처리**: 문자열에 대한 다양한 처리를 다루는 문제입니다. 문자열 압축, 회문 판별, 문자열 매칭 등이 있을 수 있습니다.
8. **기타**: 그 외에도 수학적인 문제, 비트 연산 문제, 시뮬레이션 문제, 동적 계획법과 그래프의 결합 문제 등 다양한 유형의 문제가 출제될 수 있습니다.

04. 실전 문제 체험하기

▼ 물품의 무게 정렬할기 (Quick Sort)

문제: 물품의 무게를 나타내는 정수 배열이 주어집니다. 이 물품들을 퀵정렬을 사용하여 무게순으로 오름차순으로 정렬하세요.

입력:

- 정수 배열 `weights`: 물품의 무게를 나타내는 원소가 포함된 배열입니다. 배열의 길이는 1 이상 100 이하입니다. 각 원소는 -100 이상 100 이하의 정수입니다.

출력:

- 정렬된 정수 배열 `sortedWeights`: 주어진 `weights` 배열을 퀵정렬을 사용하여 오름차순으로 정렬한 배열입니다.

예시:

```
입력: [5, 3, 9, 1, 7]
출력: [1, 3, 5, 7, 9]
```

주의사항:

- 퀵정렬은 분할 정복 기법을 사용하는 정렬 알고리즘입니다. 재귀적인 방식으로 구현해야 합니다.
- 퀵정렬을 구현할 때는 배열을 직접 분할하고 정렬하는 방식을 사용해야 합니다. 내장된 정렬 함수는 사용하지 마세요.
- 정렬된 배열의 순서는 오름차순이어야 합니다.
- 동일한 값이 여러 번 나타날 수 있으며, 이러한 값들은 정렬된 배열에서 동일한 값끼리 인접하게 위치해야 합니다.
- 추가적인 배열을 사용하지 않고 주어진 배열 내에서 정렬을 수행해야 합니다.

▼ 정답

```
def quicksort(weights, start, end):
    if start >= end:
        return

    pivot = partition(weights, start, end)
    quicksort(weights, start, pivot - 1)
    quicksort(weights, pivot + 1, end)

def partition(weights, start, end):
    pivot = weights[end]
    i = start - 1

    for j in range(start, end):
        if weights[j] < pivot:
            i += 1
            weights[i], weights[j] = weights[j], weights[i]

    weights[i + 1], weights[end] = weights[end], weights[i + 1]
    return i + 1

weights = [5, 3, 9, 1, 7]
quicksort(weights, 0, len(weights) - 1)
print(weights)
```

▼ 그리디 알고리즘 - 작업 스케줄링

문제:
작업 스케줄링
주어진 일정과 작업에 대한 정보를 바탕으로 작업을 최적으로 배치하는 문제입니다. 각

작업은 시작 시간과 종료 시간이 주어지며, 하나의 작업을 동시에 수행할 수 없습니다. 작업 스케줄링 알고리즘을 사용하여 최대한 많은 작업을 완료할 수 있는 방법을 찾아보세요.

입력:

- jobs: 작업의 정보를 담은 리스트. 각 작업은 시작 시간과 종료 시간으로 구성되며, 작업의 수는 N입니다. (1 <= N <= 100)
- 예시: [(1, 4), (3, 6), (2, 8), (5, 7), (4, 9)]

출력:

- 최대로 완료할 수 있는 작업의 개수를 반환하세요.

예제:
Input: [(1, 4), (3, 6), (2, 8), (5, 7), (4, 9)]
Output: 3

▼ 정답

```
using System;
using System.Collections.Generic;

public class Job
{
    public int StartTime { get; set; }
    public int EndTime { get; set; }

    public Job(int startTime, int endTime)
    {
        StartTime = startTime;
        EndTime = endTime;
    }
}

public class JobScheduling
{
    public int MaxJobScheduling(List<Job> jobs)
    {
        jobs.Sort((x, y) => x.EndTime.CompareTo(y.EndTime));

        int maxJobs = 0;
        int prevEndTime = 0;

        foreach (var job in jobs)
        {
            if (job.StartTime >= prevEndTime)
            {
                maxJobs++;
                prevEndTime = job.EndTime;
```

```
            }
        }

        return maxJobs;
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        List<Job> jobs = new List<Job>
        {
            new Job(1, 4),
            new Job(3, 6),
            new Job(2, 8),
            new Job(5, 7),
            new Job(4, 9)
        };

        JobScheduling scheduler = new JobScheduling();
        int maxJobs = scheduler.MaxJobScheduling(jobs);

        Console.WriteLine($"Maximum Jobs: {maxJobs}");
    }
}
```

이전 강의

📕 4) 고급 알고리즘