알고리즘_1~2일차 (1/29 ~ 1/30)

첫수업 설명

매일 과제 1개씩 나간다.
(가급적 1~2시간 남기고 과제를 올려줌, 가능하면 일과 중에 다 풀고 귀가하기)

swea learn 인터메디에트(주황색)에서 가져온 문제들임

A형 목표 : 완전검색을 사용해서 풀 수 있는가?

목차

배열(List) 1 (Array 1)

※ 여기서 List는 자료구조에서의 List이다.

---

1. 과정 소개

1-1. 학습 내용

> 핵심 : **자료구조와 알고리즘**

1-2. 학습 목표

> 논리적 사고력 향상

> 문제 해결 능력 향상

> 최종 목표는 연습 문제가 아닌 "현실 세계 문제"

1-3. 학습 도구

> Python

---

2. 알고리즘 (Algorithm)

> 유한한 단계를 통해 **문제를 해결하기 위한 절차나 방법**

> 주로 겸퓨터 용어로 쓰이며, 컴퓨터가 어떤 일을 수행하기 위한 단계적 방법
> (어떠한 문제를 해결하기 위한 절차)

> 예를 들어, 1부터 100까지의 합을 구하는 문제

2-1. 의사코드와 순서도

> 컴퓨터 분야에서 알고리즘 표현하는 방법은 2가지

1. 의사코드 (슈도코드, Pseudocode)
2. 순서도

2-2. 알고리즘의 성능

무엇이 좋은 알고리즘인가?

> **정확성** : 얼마나 정확하게 동작하는가
>
> **작업량** : 얼마나 적은 연산으로 원하는 결과를 얻어내는가
>
> **메모리 사용량** : 얼마나 적은 메모리를 사용하는가
>
> **단순성** : 얼마나 단순한가
>
> **최적성** : 더 이상 개선할 여지없이 최적화되었는가

알고리즘의 작업량 표현

> 시간 복잡도로 표현한다.

> **시간 복잡도 (Time Complexity)**
> 실제 걸리는 시간을 측정
> 실행되는 명령문의 개수를 계산

시간 복잡도 ≒ 빅-오(O) 표기법

> 빅-오 표기법 (Big-O Notation)

> 시간 복잡도 함수 중에서 가장 큰 영향력을 주는 n에 대한 항만을 표시

> 계수 (Coefficient)는 생략해서 표시

> 예시
>
> 1. O(3n + 2) = O(3n)
>   최고차항 (3n)만 선택
> 2. O(3n) = O(n)
>   계수 3 제거

> O($2n^2$ + 10n + 100) = O(n^2)
> O(4) = O(1)

> -> $4n^0$ 0제곱이라 n의 0차항이기 때문에 O(1)로 표시

---

3. 배열

> 일정한 자료형의 변수들을 하나의 이름으로 열거하여 사용하는 자료구조

> 6개의 변수를 사용해야 하는 경우, 배열로 바꾸어 사용하는 예
>
> num0 = 0, num1 = 1 ... num6 = 6 을 배열로 바꾸면
> num = [0, 1, 2, 3, 4, 5]

3-1. 배열의 필요성

> 프로그램 내에서 여러 개의 변수가 필요할 때, 일일이 다른 변수명을 이용하여 자료에 접근하는 것은 매우 비효율적일 수 있다.

> 배열을 사용하면 하나의 선언을 통해서 둘 이상의 변수를 선언할 수 있다.

> 단순히 다수의 변수 선언을 의미하는 것이 아니라, 다수의 변수로는 하기 힘든 작업을 배열을 활용해 쉽게 할 수 있다.

3-2. 1차원 배열

1차원 배열의 선언

> 별도의 선언 방법이 없으면 변수에 처음 값을 할당할 때 생성

> 이름 : 프로그램에서 사용할 배열의 이름
>
> 1차원 뜻 : 1개의 idx만을 가지고 접근 할 수 있다는 뜻

1차원 배열의 접근

> arr[0] = 10
> #배열 Arr의 0번 원소에 10을 저장하라

> arr[idx] = 20
> #배열 arr의 idx번 원소에 20을 저장하라

---

4. 버블 정렬 (Bubble Sort)

> 인접한 두 개의 원소를 비교하며 자리를 계속 교환하는 방식

4-1. 정렬

정렬

> 2개 이상의 자료를 특정 기준에 의해 작은 값부터 큰 값, 혹은 그 반대의 순서대로 재배열하는 것
> (오름차순, 내림차순)
>
> 키 : 자료를 정렬하는 기준이 되는 특정 값

정렬의 종류

> 버블 정렬 (Bubble Sort)
> 카운팅 정렬 (Counting Sort)
> 선택 정렬 (Selection Sort)
> 퀵 정렬 (Quick Sort)
> 삽입 정렬 (Insertion Sort)
> 병합 정렬 (Merge Sort)

4-2. 버블 정렬 과정

정렬 과정

> 교환하며 자리를 이동하는 모습이 물 위에 올라오는 거품 모양과 같다고 하여 버블 정렬이라고 한다.

1. 첫 번째 원소부터 인접한 원소끼리 자리를 계속 교환하면서 맨 마지막 자리까지 이동한다.
2. 한 단계가 끝나면 가장 큰 원소가 마지막 자리로 정렬된다.
   (한 단계마다 가장 큰 원소부터 마지막 자리부터 채워지면서 정렬해 나가는 것)
3. 다음 단계에서는 마지막 자리는 빼고 다시 1번부터 시작

> 시간 복잡도 : O(n^2)

---

5. 카운팅 정렬 (Counting Sort)

> 항목들의 순서를 결정하기 위해 집합에 각 항목이 몇 개씩 있는지 세는 작업을 하며, **선형 시간에 정렬하는 효율적인 알고리즘**

제한사항

> **정수나 정수로 표현할 수 있는 자료만 적용 가능**
> (각 항목의 발생 횟수를 기록하기 위해, 정수 항목으로 인덱스 되는 카운트들의 배열을 사용하기 때문에)

> 카운트들을 위한 충분한 공간을 할당하려면 집합 내의 가장 큰 정수를 알아야 한다.

시간 복잡도

> O(n + k) : n은 리스트 길이, k는 정수의 최대값

정렬 알고리즘 비교

메서드	설명	설명	설명	설명

메서드	설명	설명	설명	설명
알고리즘	평균 수행시간	최악 수행시간	알고리즘 기법	비고
버블 정렬	O(n^2)	O(n^2)	비교와 교환	코딩이 가장 쉽다
카운팅 정렬	O(n+k)	O(n+k)	비교환 방식	k가 비교적 작을 때만 가능
선택 정렬	O(n^2)	O(n^2)	비교와 교환	교환의 회수가 버블&삽입정렬보다 작다.
퀵 정렬	O(n log n)	O(n^2)	분할 정복	평균적으로 가장 빠르나 최악의 경우 O(n^2)
삽입 정렬	O(n^2)	O(n^2)	비교와 교환	n의 개수가 작을 때 효과적
병합 정렬	O(n log n)	O(n log n)	분할 정복	연결리스트의 경우 가장 효율적인 방식

6. 완전 검색 (Exaustive Search)

> 완전 검색 방법은 문제의 해법으로 생각할 수 있는 **모든 경우의 수를 나열해보고 확인하는 기법**

> 모든 경우의 수를 테스트한 후, 최종 해법을 도출
>
> 모든 경우의 수를 생성하고 테스트하기 때문에 수행속도가 느림
> (대신 해답을 찾지 못할 확률이 낮다.)
>
> 일반적으로 경우의 수가 상대적으로 작을 때 유용
>
> Brute-force 혹은 generate-and-test 기법 이라고도 불린다.

완전 검색 유용하게 사용하는 방법

> 주어진 문제를 풀 때 완전 검색으로 접근해서 해답을 도출한 후,
> 성능 개선을 위해 다른 알고리즘을 사용하고 해답을 확인하는 것이 바람직하다.

주의점

> 쓸데없는 중복을 피하자!

순열

> 서로 다른 것들 중 몇 개를 뽑아서 한 줄로 나열하는 것
> 서로 다른 n개 중 r개를 택하는 순열을 다음과 같다. (nPr)

순열 예시

```
# {1, 2, 3}을 포함하는 모든 순열을 생성하는 함수

for i1 in range(1,4):
    for i2 in range(1,4):
        if i2 != i1:
            for i3 in range(1,4):
                if i3 != i1 and i3 != i2:
                    print(i1, i2, i3)
```

6. 그리디 (Greedy Algorithm)

> 최적해를 구하는데 사용되는 **근시안적인 방법**

> 여러 경우 중 하나를 결정해야 할 때마다 **그 순간에 최적이라고 생각되는 것을 선택**해 나가는 방식으로 진행

> 각 선택시점에서는 결정이 지역적으로는 최적이지만,
> 그 선택들을 계속 수집하여 최종적인 해답을 만들었다고 하여,
> 그것이 최적이라는 보장은 없다.

> 일반적으로, 머릿속에 떠오르는 생각을 검증 없이 바로 구현하면 Greedy 접근이 된다.

> 단, 해답을 찾아내지 못하는 경우도 있으니 유의

6-1. 그리디 동작 과정

1. 해 선택

> 현재 상태에서 부분 문제의 최적 해를 구한 뒤, 이를 부분해 집합(Solution Set)에 추가한다.

2. 실행 가능성 검사

> 새로운 부분해 집합이 실행 가능한지 확인
> 곧, 문제의 제약 조건을 위반하지 않는지를 검사

3. 해 검사

> 새로운 부분해 집합이 문제의 해가 되는지 확인
> 아직 전체 문제의 해가 완성되지 않았으면 '해 선택'부터 다시 시작

그리디를 이용한 baby-gin 풀이

> 6개의 숫자는 6자리의 정수 값으로 입력된다.
> counts 배열의 각 원소를 체크하여 run과 triplet 및 baby-gin 여부를 판단한다.

```
num = 456789  # Baby Gin 확인할 6자리 수
c = [0] * 12
```

```
 # 6자리 수로부터 각 자리수를 추출하여 개수를 누적할 리스트

for i in range(6):
    c[num % 10] += 1
    num //= 10
# 위 for문 3줄 코드 자주 쓰이는 것이니 꼭 암기하기!!
# 자리수를 몰라서 range()를 못 쓰는 경우에는
# while num > 0을 for문 대신 쓰기

i = 0
tri = run = 0
while i < 10:
    if c[i] >= 3: # triplet 조사 후 데이터 삭제
        c[i] -= 3
        tri += 1
        continue
    if c[i] >= 1 and c[i+1] >= 1 and c[i+2] >= 1:
        # run 조사 후 데이터 삭제
        c[i] -= 1
        c[i+1] -= 1
        c[i+2] -= 1
        run += 1
        continue
    i += 1

if run + tri == 2:
    print("Baby Gin")
else:
    print("Lose")
```
