

WINK Algorithm Study

윙크 알고리즘 1주차 스터디

- 발표자 : WINK 기획차장 김태훈 -



발표자 소개



김태훈



@iamfinetangyou

WINK 7대 기획 차장

소프트웨어융합대학 제5대

학생회장 정후보

활동분야 : App Front-end

관심분야 : 다랑이 / 앱 / 웹 / 인공지능 / 디자인 / 독서

혈액형 : O형

키 / 몸무게 : 183 / 78

MBTI : ENFJ (정의로운 사회운동가) / 전세계 인구 2% ㅋㅋ

연락처 : 010-2383-6161

rlaxogns4504@kookmin.ac.kr

rlaxogns4504@gmail.com

참고자료



Algorithm

SAMSUNG



kakao

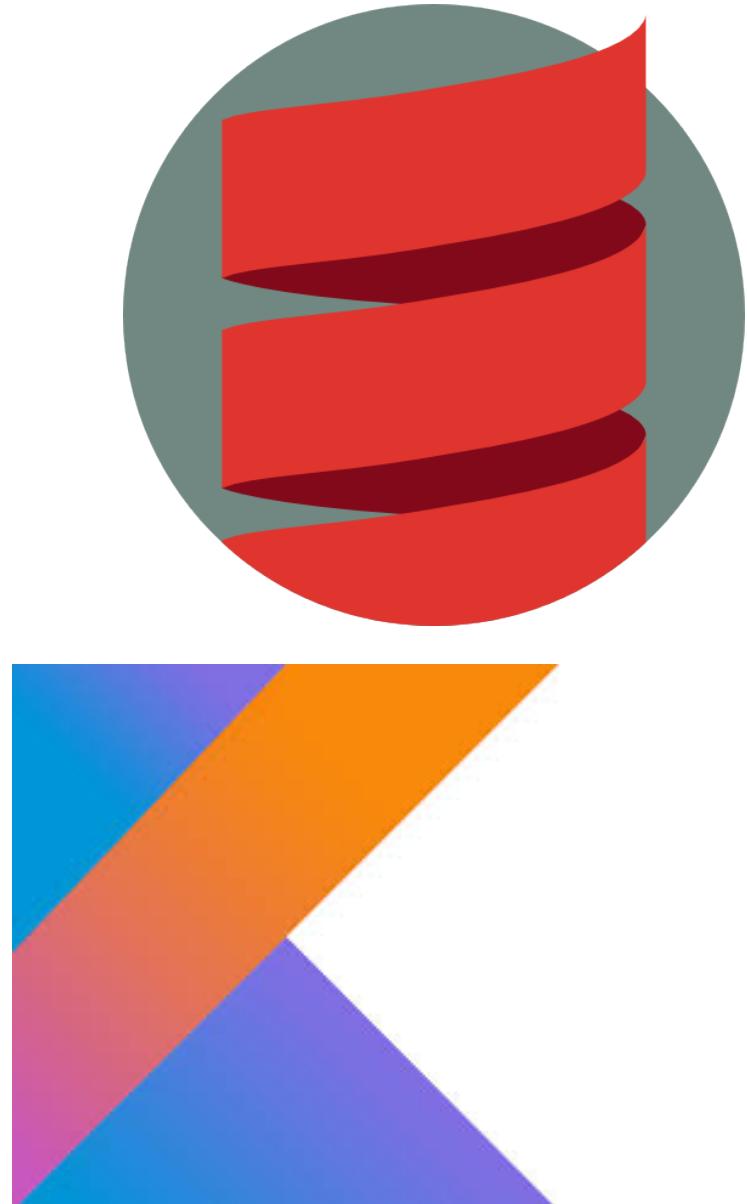


어디에 쓰일까요?

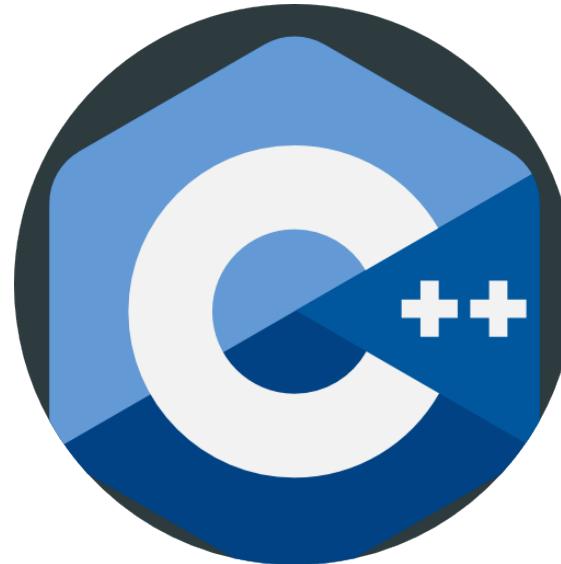
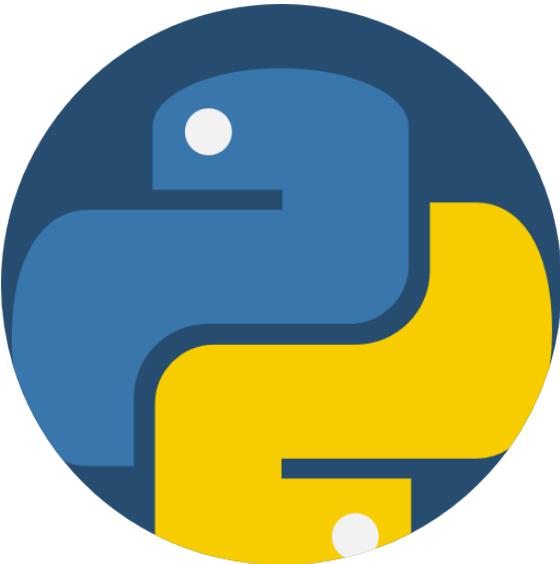
Language



무슨 언어를 쓸까?



코딩 테스트에 유리한 언어



양대산맥

Python



- 배우기 쉽다
- 풍부한 라이브러리
- 변칙적인 유형에 대응이 쉬움
- 코드가 짧고 직관적으로 문제를 풀이할 수 있음
- 특히 문자열 처리가 매우 간결하고 쉬움
- 통신모듈 코드도 매우 간결 (카카오는 특정 서버와 데이터를 주고 받는 문제를 꾸준히 출제함)
- 채점 시스템이 Python3 뿐만 아니라 PyPy3를 지원할 경우 C/C++ 보다 실행시간이 빠를 수도 있음

C++



- 실행시간이 빠르다 (삼성전자는 SW 역량 테스트 B형부터는 파이썬을 배제함)
- C++을 공부하는 과정에서 컴퓨터의 내부 동작 구조를 파악할 수 있음
- C++이 여러 산업에서 오래 사용이 되어왔기 때문에 인터넷에 참고 할 자료가 매우 많음

진행방향

그리디, 구현, DFS/BFS, 정렬, 바이너리 서치, 다이나믹 프로그래밍, 최단 경로

욕심 Greedy

그리디 알고리즘

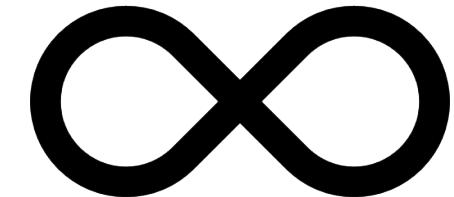
욕심 Greedy

- 현재 상황에서 지금 당장 좋은 것만 고르는 알고리즘
(Dijkstra Algorithm과 같은 특수한 경우 제외)
- 사전에 외우고 있지 않아도 풀 수 있는 가능성이 높은 문제 유형
(정렬, 최단 경로 등의 알고리즘 유형은 사전에 사용 방법을 완전히 숙지하고 있어야 함)
- 창의력, 즉 문제를 풀기 위한 최소한의 아이디어를 떠올릴 수 있는 능력을 요구
- 기준에 따라 좋은 것을 선택하는 알고리즘이므로 정렬 알고리즘과 자주 짹을 이뤄서 문제가 출제됨

그리디 알고리즘 - 거스름돈



x



당신은 카운터 알바생입니다. 카운터에는 거스름돈으로 사용한 500 원, 100원, 50원, 10원 동전이 무한하게 존재한다고 가정합니다.

그리디 알고리즘 - 거스름돈

가장 큰 화폐 단위부터 돈을 거슬러 주면 되는 것 아닙니까?



N원을 거슬러 줘야 할 때, 가장 먼저 500원으로 거슬러 줄 수 있을 만큼 거슬러 준다. 그 다음 차례대로 100원, 50원, 10원 동전을 거슬러 주면 최소의 동전 개수로 거슬러 줄 수 있다.

그리디 알고리즘 - 거스름돈

STEP 1 - 거스름돈 1,260원



X 2

줘야할 남은 돈 - 260원

일단 1,260원을 넘지 않는 선을 지켜서 500원 짜리 두 개를 줍니다. 그럼 남은 거스름돈은 260원이 됩니다.

그리디 알고리즘 - 거스름돈

STEP 2 - 거스름돈 260원



X 2

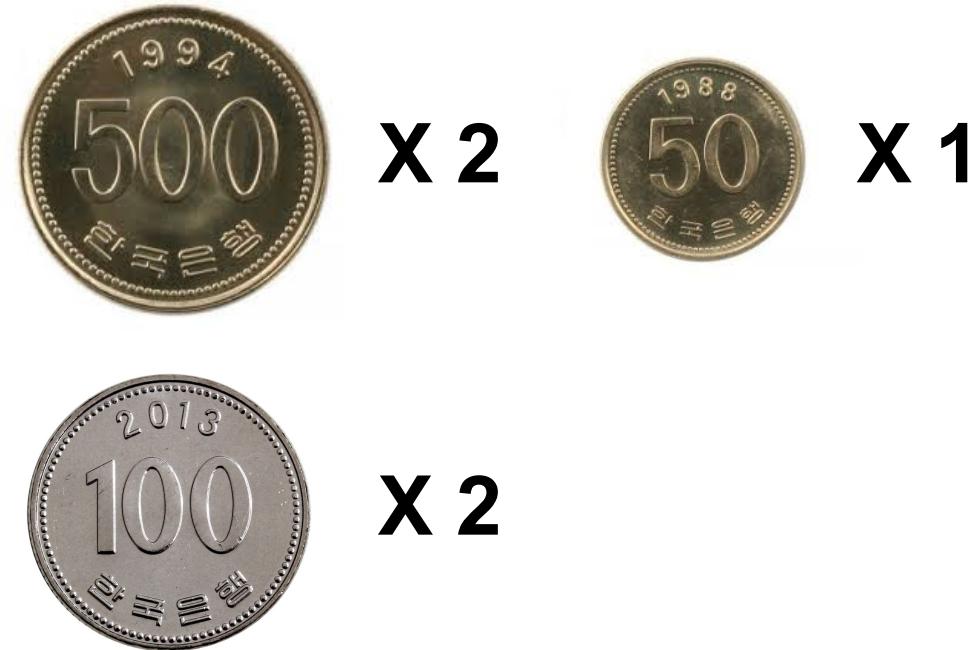


X 2

줘야할 남은 돈 - 60원

그리디 알고리즘 - 거스름돈

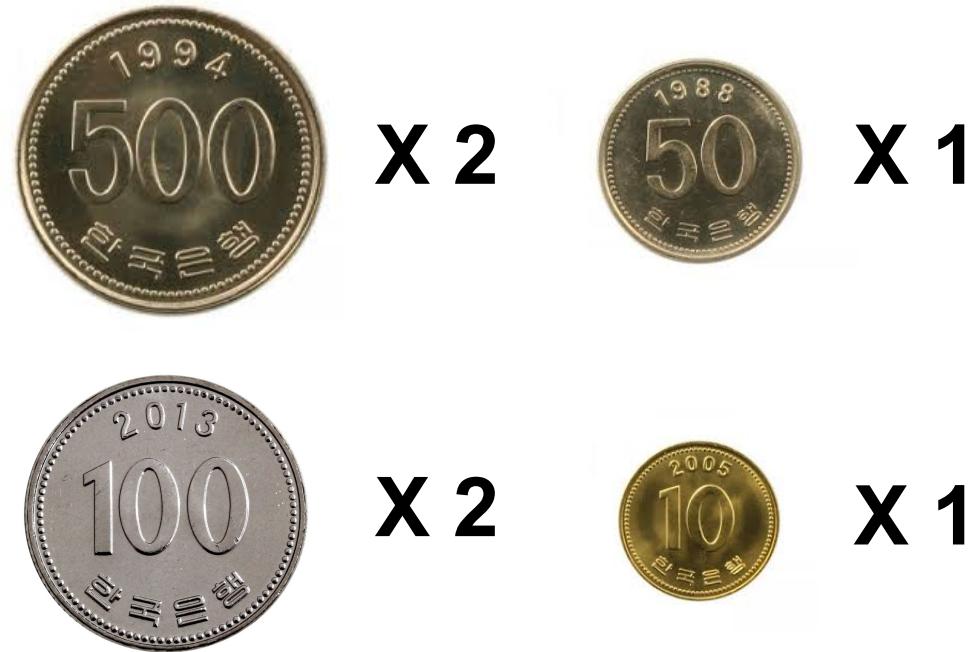
STEP 3 - 거스름돈 60원



줘야할 남은 돈 - 10원

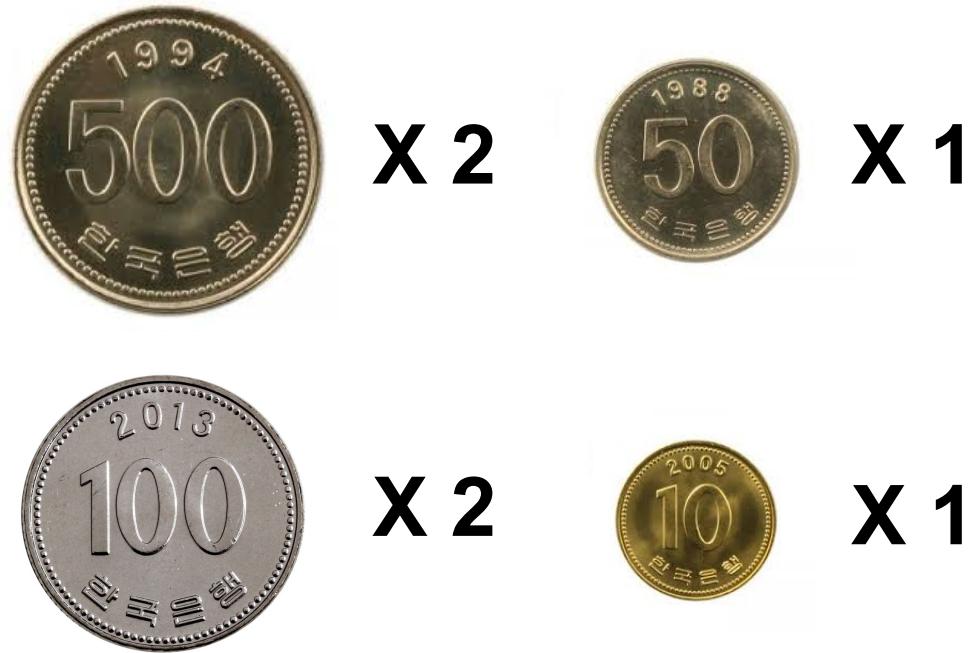
그리디 알고리즘 - 거스름돈

STEP 4 - 거스름돈 10원



줘야할 남은 돈 - 0원 끝!

그리디 알고리즘 - 거스름돈



N이 1,260일 때 손님이 받을 수 있는 동전의 최소 개수는 6개

그리디 알고리즘 - 거스름돈

```
1  n = 1260
2  count = 0
3
4  coin_types = [500, 100, 50, 10]
5
6  for coin in coin_types:
7      count += n // coin
8      n %= coin
9
10 print(count) # 6
```

```
1  #include <iostream>
2
3  ▶ int main() {
4      int n = 1260, count = 0;
5      count += n / 500;
6      n %= 500;
7      count += n / 100;
8      n %= 100;
9      count += n / 50;
10     n %= 50;
11     count += n / 10;
12     std::cout << count; // 6
13
14 }
```

화폐의 종류만큼 반복을 수행해야 합니다. 화폐의 종류를 K개라고 가정할 때 이 소스의 시간 복잡도는 O(K)가 되겠군요

동전의 종류에만 영향을 받고 거스름돈의 크기와는 영향이 없음



Web · IN · Kookmin

국민대학교 | 소프트웨어융합대학

그리디 알고리즘의 정당성

거스름돈 800원

화폐 단위 500원, 400원, 100원



그리디 알고리즘의 정당성

400원 짜리 두 개가 최적의 해다.

그리디 알고리즘으로 해결할 수 있었던 이유는 가지고 있는 동전 중에서 큰 단위가 항상 작은 단위의 배수이므로 작은 단위의 동전들을 종합해 다른 해가 나올 수가 없기 때문입니다.

그리디 알고리즘 문제를 풀 때 이것이 정당한지 검토할 수 있어야 답을 도출할 수 있습니다.

화폐의 단위가 무작위로 주어진 경우는 추후에 Dynamic Programming으로 풀 수 있습니다.

구현 Implementation

머릿속에 있는 알고리즘을 소스코드로 바꾸는 과정

어떤 문제를 풀던 간에 소스코드를 작성하는 과정은 필수이므로 구현 문제 유형은 모든 범위의 코딩 테스트 문제 유형을 포함하는 개념입니다.

취업을 목표로 하는 코테에서 구현이 중심이 되는 문제가 자주 등장

문법을 잘 알아야하고 경험이 많아야 한다

DFS / BFS 그래프 탐색

DFS/BFS 알고리즘

탐색

많은 양의 데이터 중에서 원하는 데이터를 찾는 과정

대표적인 탐색 알고리즘 : DFS / BFS

선행 지식 : STACK, QUEUE, RECURSIVE

자료구조

데이터를 표현하고 관리하고 처리하기 위한 구조

EX) STACK, QUEUE

DFS/BFS 알고리즘

**파이썬은 스택을 이용할 때에 별도의 라이브러리를 사용하지
않아도 됨**

**기본 리스트에서 `append()`, `pop()` 메서드를 지원해서 스택
자료구조와 동일하게 동작함**

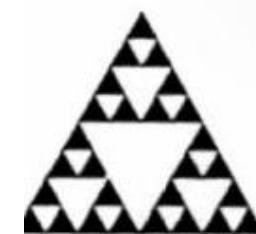
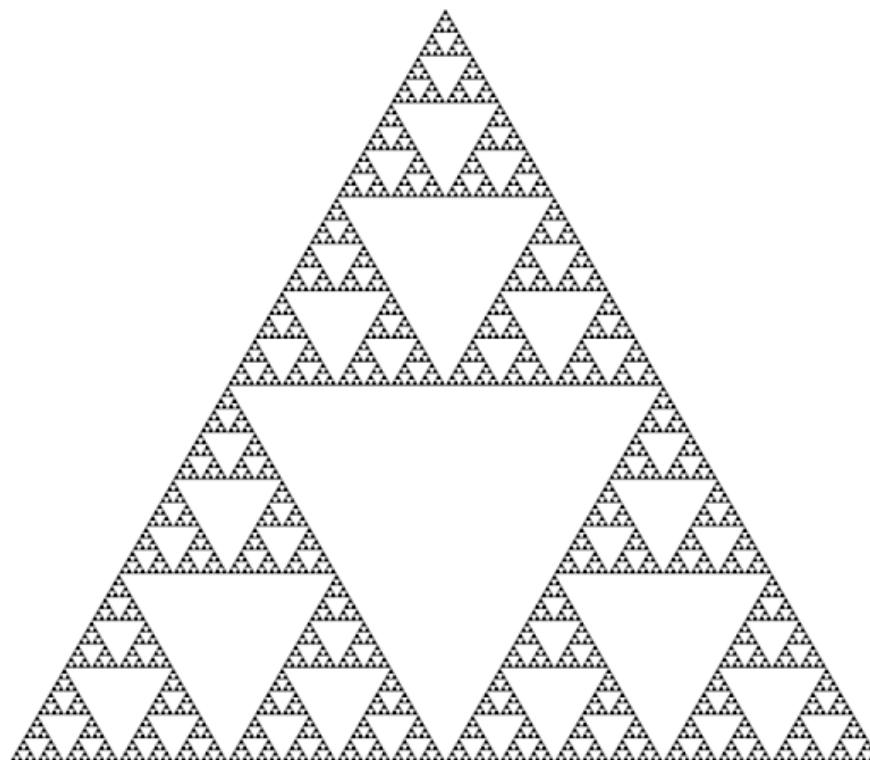
**큐는 `queue` 라이브러리 말고 `collections` 모듈에서 제공하
는 `deque` 자료구조를 활용하는 것이 좋음**

DFS/BFS 알고리즘

```
1  from collections import deque
2
3  # Queue 구현을 위해 deque 라이브러리 사용
4  queue = deque()
5
6  queue.append(5)
7  queue.append(2)
8  queue.append(6)
9  queue.append(8)
10 queue.append(7)
11 queue.popleft()
12 queue.append(5)
13 queue.append(4)
14 queue.popleft()
15
16 print(list(queue)) # [6, 8, 7, 5, 4]
```

재귀 함수

DFS/BFS를 구현하기 위해선 재귀함수를 알아야합니다



재귀 함수

```
1 def factorial_iterative(n):
2     result = 1
3     for i in range(1, n + 1):
4         result *= i
5     return result
6
7
8 def factorial_recursive(n):
9     if n <= 1:
10        return 1
11    return n * factorial_recursive(n - 1)
12
13
14 print('반복적으로 구현 :', factorial_iterative(5)) # 120
15 print('재귀적으로 구현 :', factorial_recursive(5)) # 120
```

소스코드를 반복적으로 구현했다는 말은 반복문을 이용했다는 말입니다. 흔히 재귀적으로 구현했다는 말과 대비됩니다.

재귀 함수를 쓴 코드가 더 간결함

재귀 함수가 수학의 점화식(재귀식)을 그대로 소스코드로 옮겼기 때문

수학의 점화식

특정한 함수를 자신보다 더 작은 변수에 대한 함수와의 관계로 표현한 것

factorial을 수학적 점화식으로 표현

$n \geq 0$ or 1 : $\text{factorial}(n) = 1$

$n > 1$: $\text{factorial}(n) = n * \text{factorial}(n-1)$