**데이터구조 및 프로그래밍실습 3분반**

**설계 프로젝트**

학번 : 202211953

이름 : 이정은

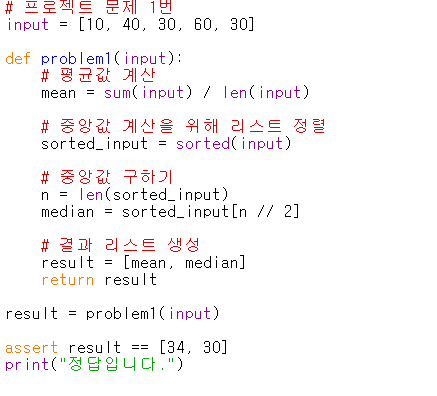
본인의 Github 주소 : https://github.com/jeongeunm

(본 과제의 코드를 본인의 Github에 업로드 하세요)

**문제 1.**

1. 본 문제를 풀기위해 사용한 데이터 구조를 제시하고 간단히 설명하세요.

이 코드에서 사용된 주요 데이터 구조는 리스트 (list)입니다. 정렬된 리스트에 관하여sorted\_input 변수는 input 리스트 [10, 40, 30, 60, 30] 를 정렬하여 얻은 리스트입니다. 정렬을 위해 파이썬 내장 함수 sorted()를 사용했습니다. 리스트 데이터 구조를 제외하고 함수 내에서는 간단한 산술 연산과 리스트 조작만을 사용해도 원하는 결과를 얻기에 충분했습니다.

1. 본인이 작성한 파이썬 Code 캡처 이미지를 첨부하고 Algorithm Analysis를 수행하세요. 

**\* 코드 동작 설명**

1) **평균값 계산**

mean = sum(input) / len(input): 주어진 리스트 input의 합계를 구하고 리스트의 길이로 나누어 평균값을 계산합니다.

**2) 중앙값 계산**:

- sorted\_input = sorted(input): input 리스트를 오름차순으로 정렬하여 sorted\_input에 저장합니다.

- n = len(sorted\_input): 정렬된 리스트의 길이를 n에 저장합니다.

- median = sorted\_input[n // 2]: 정렬된 리스트의 중간 인덱스를 구하여 중앙값을 결정합니다. 리스트의 길이가 홀수일 때는 중간값을 그대로 사용하고, 짝수일 때는 중간에 가까운 두 값 중 작은 값을 사용합니다.

**\* 알고리즘 분석**

**시간 복잡도**: 평균값을 계산하는 sum(input)은 O(n)이며, 정렬하는 sorted(input)은 O(n log n)입니다. 따라서 전체 시간 복잡도는 O(n log n)입니다.

**문제 2.**

1. 본 문제를 풀기위해 사용한 데이터 구조를 제시하고 간단히 설명하세요.

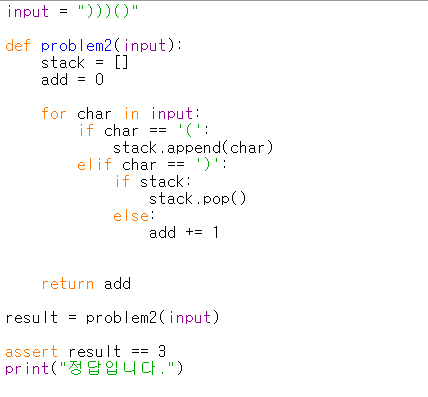
이 코드에서 사용된 주요 데이터 구조는 스택 (stack) 입니다. 스택은 데이터를 쌓아 올릴 수 있는 선형 자료구조로, 후입선출(LIFO, Last In First Out) 원칙을 따릅니다. 이 코드에서는 괄호 짝을 맞추기 위해 스택을 사용했습니다. 파이썬에서는 리스트를 사용하여 스택을 구현할 수 있습니다. 리스트의 append() 메소드로 데이터를 추가하고, pop() 메소드로 데이터를 제거할 수 있습니다.

\* 코드에서의 스택 활용

**스택 초기화**: stack = []로 스택을 초기화합니다

**여는 괄호 처리**: '('가 나오면 stack.append('(')로 스택에 추가합니다

**닫는 괄호 처리**: ')'가 나오면 스택이 비어있지 않으면 stack.pop()을 통해 마지막으로 추가된 '('를 제거합니다. 스택이 비어있으면 add 변수를 증가시켜야 할 '('의 개수를 세는 방식입니다.

1. 본인이 작성한 파이썬 Code 캡처 이미지를 첨부하고 Algorithm Analysis를 수행하세요.

**\* 코드 동작 설명**

- for char in input : 입력 문자열을 한 글자 씩 순회합니다.

- if char == '(' : '('를 발견하면 스택에 추가합니다.

; stack.append(char).

- elif char == ')':

')'를 발견하면 스택이 비어있지 않은지 확인합니다. 스택이 비어있지 않으면 짝이 맞는 '('가 있으므로 스택에서 '('를 제거합니다.

; stack.pop(). 스택이 비어있으면 '('의 짝이 없는 것으로 간주하고, add 변수를 1 증가시킵니다.

**\* 알고리즘 분석**

시간 복잡도 : 입력 문자열을 한 번 순회하면서 각 문자를 스택에 추가하거나 제거하므로 O(n)입니다.

**문제 3.**

1. 본 문제를 풀기위해 사용한 데이터 구조를 제시하고 간단히 설명하세요.

본 문제를 풀기 위해 BFS와 Deque가 사용되었습니다. BFS는 시작 노드에서 시작하여 현재 노드와 인접한 모든 미 방문 노드들을 먼저 탐색합니다. 따라서 같은 단계(거리)에 있는 노드들을 모두 탐색한 후 다음 단계로 이동합니다. 덱(Deque, Double-ended queue)은 양쪽에서 데이터의 추가와 삭제가 모두 가능한 자료구조 입니다. 즉, 큐(Queue)와 스택(Stack)의 특성을 모두 갖추고 있습니다. 양쪽 끝에서의 삽입과 삭제가 O(1)의 시간 복잡도를 가집니다. 이로 인해 큐나 스택의 특성을 모두 지원하며, 특정한 상황에서 유연하게 사용할 수 있습니다.

위 코드에서는 숲을 탐색하는 과정에서 곰의 위치에서 상하좌우로의 이동을 BFS 방식으로 처리하고 있습니다. 이 때 `deque`를 사용하여 BFS의 큐(Queue) 기능을 구현하였습니다. `queue` 변수는 곰의 현재 위치에서 탐색할 다음 위치들을 관리하며, BFS 탐색을 통해 숲을 너비 우선으로 탐색하면서 꿀을 모으는 과정을 구현합니다. 덱은 `append()` 메서드로 뒤쪽에 데이터를 추가하고, `popleft()` 메서드로 맨 앞의 데이터를 꺼내는 등의 작업을 통해 BFS에서 필요한 큐의 기능을 제공합니다. 이를 통해 곰이 이동하면서 수집한 꿀의 개수를 증가시키고, 곰의 크기를 증가시키는 등의 작업을 수행할 수 있습니다.

1. 본인이 작성한 파이썬 Code 캡처 이미지를 첨부하고 Algorithm Analysis를 수행하세요.

**\* 코드 동작 설명**

1) 상하좌우로 이동할 때 필요한 방향 (direction)과 BFS를 위한 deque (queue)를 초기화합니다. visited 리스트는 방문한 위치를 기록하는 데 사용됩니다.

2) queue에서 하나씩 꺼내어 해당 위치에서 상하좌우를 탐색합니다. 유효한 범위 내에 있고, 방문하지 않은 위치라면 방문 표시를 하고, 규칙에 따라 꿀을 모으거나 곰의 크기를 증가시키며 BFS를 계속 진행합니다. time 변수를 통해 경과 시간을 측정하며, BFS가 종료되면 최종적으로 경과 시간을 반환합니다

**\* 알고리즘 분석**

시간 복잡도 : BFS를 사용하여 숲을 탐색하므로 시간 복잡도는 O(N^2)입니다 (N은 숲의 크기를 의미).