**데이터구조 및 프로그래밍실습 3분반**

**설계 프로젝트**

학번 : 202211997

이름 : 정세연

본인의 Github 주소 : https://github.com/wjdtpdus9343/data-structure-lab-2025

(본 과제의 코드를 본인의 Github에 업로드 하세요)

**문제 1.**

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

이 문제에서는 다섯 개의 자연수로 이루어진 데이터를 처리하기 위해 리스트(List)와 정수형 변수(int)를 사용하였습니다.

우선, 주어진 숫자들을 저장하고 다루기 위해 리스트를 사용하였고, 리스트를 통해 입력값을 순서대로 저장하고 평균 및 중앙값 계산에 활용하였습니다. 특히 중앙값을 계산하기 위해 리스트를 오름차순으로 정렬(Sorting)한 후, 가운데 인덱스에 위치한 값을 선택하는 방식으로 구현하였습니다.

또한 평균과 중앙값이라는 두 개의 결과값을 저장하기 위해 각각 mean과 median이라는 정수형 변수를 사용하였고, 이 두 값을 다시 하나의 리스트에 담아 결과로 반환하도록 구성하였습니다.

리스트는 값을 순차적으로 저장하고 인덱스를 통해 접근할 수 있어 이번 문제처럼 수치 데이터를 처리하는 데 매우 적합한 자료 구조였습니다.

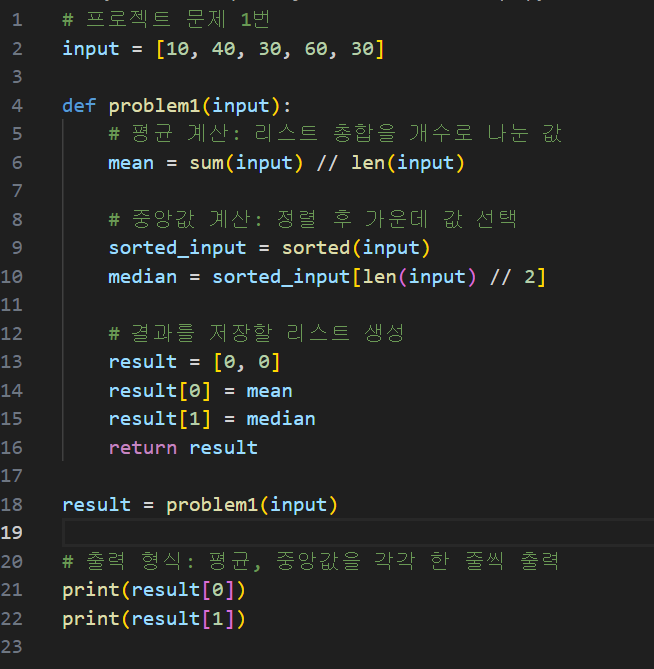
1. 본인이 작성한 파이썬 Code 캡처 이미지를 첨부하고 Algorithm Analysis를 수행하세요.

이 문제는 주어진 다섯 개의 자연수로부터 평균과 중앙값을 구하는 방식으로 구성되어 있습니다.

먼저 평균은 리스트에 저장된 값을 모두 더한 뒤, 그 총합을 리스트의 길이로 나누는 방식으로 계산하였습니다. 이때 리스트의 모든 요소를 한 번씩 확인하므로 시간 복잡도는 O(n)입니다.

그리고 중앙값을 구할 때는 리스트를 오름차순으로 정렬한 다음, 가운데 인덱스에 있는 값을 선택하여 계산하였습니다. 정렬에는 일반적으로 시간 복잡도 O(n log n)이 소요되기 때문에, 전체 알고리즘의 시간 복잡도는 O(n log n)이라고 볼 수 있습니다. 하지만 이 문제에서는 입력 개수가 5개로 고정되어 있기 때문에 실제 실행 시간은 매우 빠르고, 거의 상수 시간에 가깝다고 할 수 있습니다.

공간 복잡도의 경우에는 입력 리스트 외에 정렬된 리스트와 평균, 중앙값을 저장하기 위한 변수만 추가로 사용되므로 공간 복잡도는 O(n) 수준으로 분석할 수 있습니다.



**문제 2.**

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

이 문제에서는 괄호로 이루어진 문자열을 처리하기 위해 문자열(String)과 정수형 변수(int)를 사용하였습니다.

입력으로 주어진 문자열은 각 문자를 차례대로 탐색하기 위해 활용되었으며, 문자 하나하나를 순회하면서 괄호의 열림과 닫힘을 확인할 수 있도록 하였습니다. 문자열 내의 각 문자를 순서대로 확인하며 닫는 괄호가 나올 때마다, 열린 괄호와 짝을 이루는 경우를 찾아 괄호 덩어리의 개수를 계산하였습니다.

또한, 괄호 상태를 추적하기 위해 정수형 변수인 open\_needed를 사용하였습니다. 이 변수는 현재까지 닫히지 않은 열린 괄호의 개수를 의미하며, 괄호의 개수가 맞을 때마다 값을 조정하여 최종적으로 완성된 괄호 덩어리의 개수를 계산하는 데 활용하였습니다.

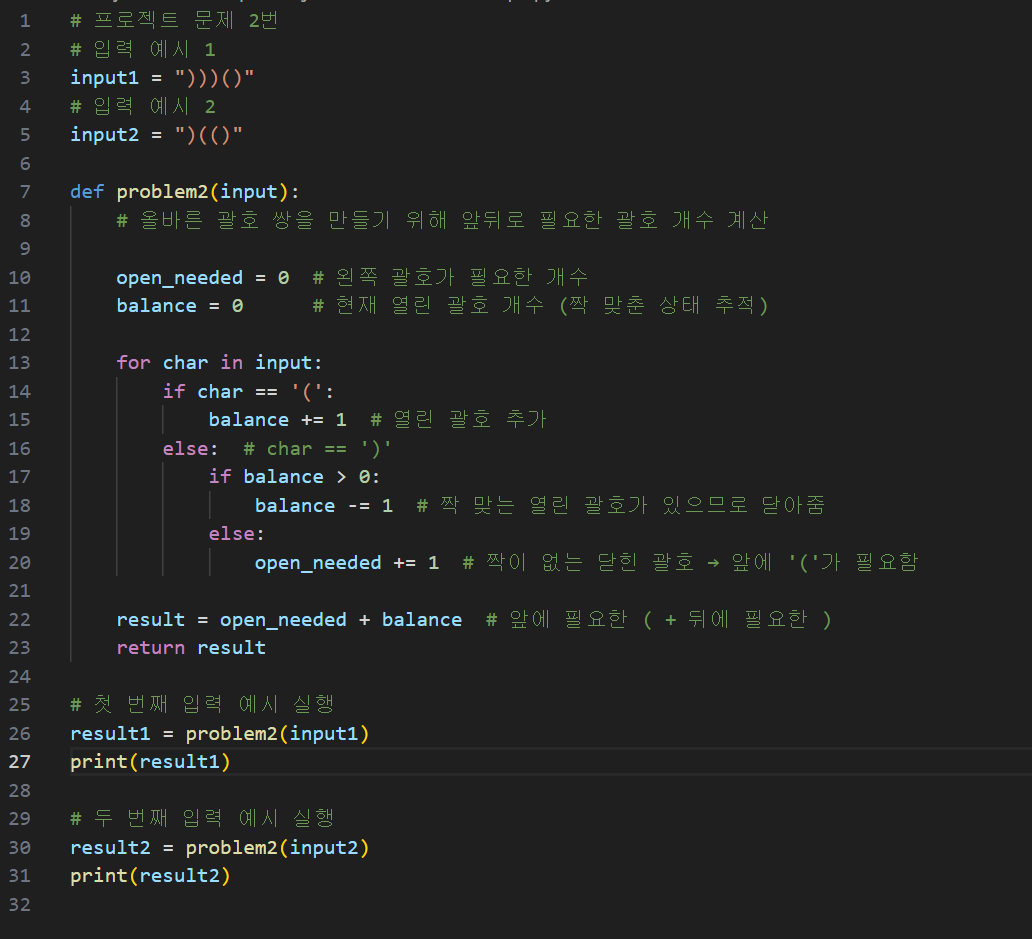
마지막으로, 완성된 괄호 덩어리의 총 개수는 별도의 result 변수에 저장하였습니다.

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

이 문제에서는 문자열을 한 글자씩 탐색하며 괄호의 짝을 맞추는 방식으로 문제를 해결하였습니다.

주어진 문자열의 길이를 n이라고 할 때, 모든 문자를 앞에서부터 한 번씩 확인하므로 전체 시간 복잡도는 O(n)입니다. 괄호의 상태를 추적하는 데에는 단 두 개의 정수형 변수만 사용하였기 때문에, 추가적인 공간 사용은 거의 없으며 공간 복잡도는 O(1)입니다.

문자열의 길이가 길어지더라도 시간과 메모리 사용이 크게 증가하지 않기 때문에, 이 알고리즘은 입력 크기에 관계없이 안정적으로 동작한다고 볼 수 있습니다.



**문제 3.**

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

이 문제에서는 곰이 숲을 돌아다니며 꿀벌집을 먹는 과정을 시뮬레이션하기 위해 2차원 리스트, 큐(Queue), 그리고 여러 개의 정수형 변수(int)를 사용하였습니다.

먼저, 입력으로 주어진 숲의 정보는 2차원 리스트 형태로 저장하였습니다. 숲의 각 칸에는 이동 가능 여부와 꿀벌집의 유무, 크기 등의 정보가 포함되어 있어 좌표 기반으로 데이터를 쉽게 탐색하고 수정하기 위해 2차원 리스트 구조가 적합하였습니다.

그리고, 곰이 꿀벌집을 찾아 이동하는 과정에서는 BFS 탐색을 활용하였습니다. 이 과정에서 탐색 순서를 효율적으로 관리하기 위해 큐(Queue) 자료구조를 사용하였으며, 파이썬의 collections 모듈에서 제공하는 deque를 이용하여 구현하였습니다. 이 구조는 양쪽 끝에서 데이터를 넣고 뺄 수 있기 때문에 BFS에 매우 적합합니다.

또한, 곰의 크기, 현재 위치, 꿀벌집을 먹은 개수, 전체 이동 시간 등을 저장하고 추적하기 위해

여러 개의 정수형 변수들이 사용되었습니다. 이러한 변수들은 시뮬레이션의 상태를 지속적으로 갱신해주는 역할을 하며, 곰의 성장 조건과 이동 가능 조건 등을 판단하는 데 필수적인 요소로 활용되었습니다.

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

이 문제는 곰이 숲 속을 돌아다니면서 꿀벌집을 먹는 과정을 구현하는 시뮬레이션 문제였습니다.

문제의 핵심은 가까운 거리 순으로 꿀벌집을 찾아가고 일정 개수를 먹으면 곰이 성장하며, 이 과정을 반복하는 흐름을 정확하게 구현하는 것이었습니다.

우선 꿀벌집을 찾아가는 과정에는 너비 우선 탐색(BFS) 알고리즘을 사용했습니다. 가장 가까운 위치부터 탐색할 수 있고, 같은 거리에서는 위쪽이나 왼쪽을 먼저 찾는 식으로 우선순위가 정해져 있었기 때문에 BFS가 가장 적절하다고 판단했습니다. 이때 큐는 deque 자료구조를 사용해서 구현했고, 이를 통해 탐색 속도를 높일 수 있었습니다.

시간 복잡도는 한 번의 탐색에 O(N²)이 걸리고, 꿀벌집을 하나 먹을 때마다 탐색을 반복하므로 전체적으로는 O(N² × K) 정도로 볼 수 있습니다. 다만 문제에서 N이 4로 고정되어 있고, 꿀벌집 개수도 많지 않아서 실행 시간은 매우 빠르게 나왔습니다.

공간 복잡도는 방문 여부를 저장하는 배열과 BFS에 필요한 큐 정도만 사용되기 때문에 대략 O(N²) 수준이며, 이 역시 고정된 입력 크기 덕분에 메모리 사용이 크지 않았습니다.

