**데이터구조 및 프로그래밍실습 3분반**

**설계 프로젝트**

학번 : 202211770

이름 : 김왕진

본인의 Github 주소 : https://github.com/2011kwj/data-structure-lab-2025.git

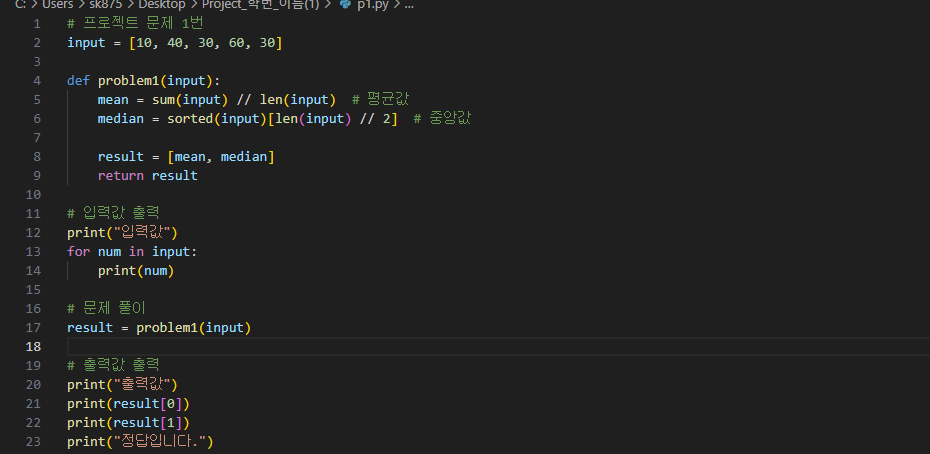
(본 과제의 코드를 본인의 Github에 업로드 하세요)

**문제 1.**

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

- 사용한 데이터 구조: 리스트 (List)  
 주어진 자연수 5개를 저장하고 처리하기 위해 리스트를 사용하였습니다. 리스트는 여러 값을 순서대로 저장할 수 있어, 평균과 중앙값 계산 시 반복 처리와 정렬이 용이하다고 할 수 있습니다. sum(), len(), sorted() 등의 내장 함수와 함께 사용하여 평균과 중앙값을 쉽게 계산할 수 있습니다.

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



이 프로그램의 시간복잡도는 O(n log n), 공간복잡도는 O(n)입니다.  
sum()함수로 평균을 계산할 때는 리스트를 한 번 순회하므로 O(n)이 걸리고, 중앙값을 구하기 위해 sorted()를 사용하는 과정에서 O(n log n)의 시간이 소요됩니다.  
공간 측면에서는 정렬 결과가 새로운 리스트로 생성되므로 추가로 O(n)의 공간이 사용됩니다.  
입력 데이터가 5개로 고정되어 있어 실제 실행 시간은 매우 빠르지만, 이 분석은 입력 개수가 커질 경우를 고려한 이론적 복잡도 분석이라 할 수 있겠습니다.

**문제 2.**

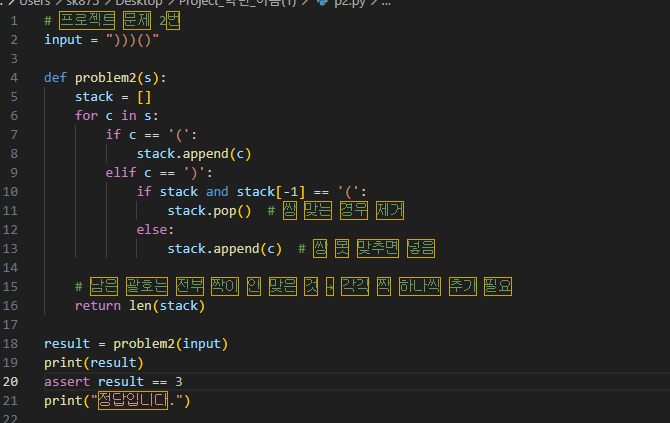
1. 본 문제를 풀기위해 사용한 데이터 구조를 제시하고 간단히 설명하세요.

이 문제에서는 주어진 괄호열을 순차적으로 탐색하며 괄호 쌍의 유효성을 확인하기 위해, 정수형 변수(int)를 이용한 균형 추적 방식을 사용하였습니다.

구체적으로는, 열리는 괄호 '('의 개수를 추적하는 balance변수와, 짝을 이루지 못한 닫는 괄호 ')'의 개수를 세는 add\_front변수를 활용하였습니다. 이 두 변수는 스택을 직접 사용하는 대신, 괄호의 짝 맞춤 상태를 정수의 누적/감소 연산을 통해 간단하게 모델링할 수 있도록 해 줍니다.

이러한 방식은 괄호열을 선형적으로 한 번만 순회하면서도 유효성을 검사할 수 있는 효율적인 방법이며, 시간 복잡도 O(n), 공간 복잡도 O(1)로 문제를 해결할 수 있다는 점에서 매우 효과적이라고 생각을 하였고 그래서 이렇게 구조를 만들게 되었습니다..

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



이 문제는 주어진 괄호열에서 짝이 맞지 않는 괄호의 개수를 찾아내고, 그것을 완전한 괄호열로 만들기 위해 앞이나 뒤에 추가로 붙여야 하는 최소 괄호 수를 구하는 문제입니다. 저는 이 문제를 해결하기 위해 수업 시간에 배운 스택(Stack) 자료구조를 활용했습니다.

문자열의 왼쪽부터 오른쪽으로 하나씩 문자를 확인하면서 여는 괄호 '('는 스택에 넣고, 닫는 괄호 ')'가 나왔을 때 스택의 맨 위에 여는 괄호가 있으면 그것과 짝을 지어 pop 해주었습니다. 만약 스택이 비어 있거나 짝이 없는 상태에서 ')'가 들어오면 그 괄호는 짝을 이룰 수 없으므로 그대로 스택에 넣습니다. 이런 과정을 반복한 뒤 최종적으로 스택에 남아 있는 괄호는 전부 쌍을 이루지 못한 괄호들이고, 그만큼 짝을 붙여줘야 완전한 괄호열이 되기 때문에 이 개수가 바로 정답이 됩니다.

이 코드는 주어진 괄호열을 한 번만 순회하면서 문제를 해결하기 때문에, 입력 길이를 n이라 할 때 시간복잡도는 O(n)입니다. 즉, 입력이 길어져도 성능이 급격히 떨어지지 않는 효율적인 알고리즘이라고 생각을 하였고 그래서 이렇게 구조를 만들게 되었습니다.

**문제 3.**

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

본 문제를 풀기 위해 사용한 주요 데이터 구조는 다음과 같습니다.

1. 2차원 리스트 (forest)  
숲의 지형 정보를 저장하는 데 사용됩니다. 각 요소는 0(빈칸), 1~6(벌집 크기), 9(곰의 초기 위치)를 의미합니다. 이 리스트를 통해 곰의 위치, 벌집의 크기, 이동 가능한 칸을 판단할 수 있습니다.

2. 튜플 ((x, y, 거리)형태)  
BFS에서 탐색 중인 좌표와 현재까지 이동한 거리를 함께 저장하기 위해 사용합니다. 각 상태는 큐에 (현재 위치 x, y, 이동 거리) 형식으로 저장됩니다.

3. deque(collections 모듈의 큐 자료구조)  
곰이 꿀을 찾기 위해 너비 우선 탐색(BFS)을 할 때 사용됩니다. 일반적인 리스트보다 양 끝에서의 삽입과 삭제가 빠르기 때문에, BFS의 성능을 위해 적합합니다.

4. 리스트 (edible)  
곰이 먹을 수 있는 벌집들의 위치와 거리를 저장한 리스트입니다. (거리, x, y)형태로 저장하여, 정렬 시 거리가 가장 가까우면서 가장 위/왼쪽에 있는 벌집을 우선순위로 선택하게 합니다.

5. 불린형 2차원 리스트 (visited)  
탐색 중 중복 방문을 방지하기 위해 사용됩니다. 특정 좌표에 대해 방문 여부를 기록하여 무한 루프를 막고, 효율적인 BFS를 수행하게 합니다.

이와 같은 자료구조를 통해 곰의 이동 경로 탐색, 조건 판단, 상태 업데이트를 효율적으로 구현할 수 있었습니다.

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

이 코드는 곰이 숲을 돌아다니며 먹을 수 있는 벌집을 찾아가 꿀을 최대한 많이 먹고, 그 과정에서 걸린 총 시간을 계산하는 프로그램입니다. 곰은 자신의 크기보다 작은 벌집만 먹을 수 있고, 상하좌우로 1초에 한 칸씩 이동할 수 있습니다. 꿀을 먹으면 해당 칸은 빈칸이 되고, 곰은 자신의 크기만큼 벌집을 먹을 때마다 크기가 1씩 증가합니다.

곰은 너비 우선 탐색(BFS)을 통해 숲 전체를 탐색하며 먹을 수 있는 벌집을 찾습니다. 여러 개의 벌집이 선택지로 있을 경우, 가장 가까운 벌집을 우선 선택하고, 거리가 같다면 위쪽, 위쪽도 같다면 왼쪽에 있는 벌집을 우선적으로 선택합니다.

핵심 알고리즘 구조

-곰의 현재 위치에서 BFS를 이용해 먹을 수 있는 벌집들을 모두 탐색합니다.

-조건에 맞는 벌집이 하나뿐이라면 곰은 그 벌집으로 이동해 꿀을 먹습니다.

-여러 개일 경우 정해진 우선순위에 따라 하나를 골라 이동하고 꿀을 먹습니다.

-꿀을 먹을 때마다 곰의 위치, 크기, 먹은 횟수를 갱신하고, 위 과정을 반복합니다.

-더 이상 먹을 수 있는 벌집이 없으면 탐색을 종료하고, 총 이동 시간(초)을 출력합니다.

시간 복잡도

-숲의 크기가 N x N 이고 최대 N = 20 이므로 전체 칸 수는 최대 400개입니다.

-BFS는 한 번 수행할 때 O(N^2) 의 시간 복잡도를 가집니다.

-꿀을 먹을 수 있는 벌집이 여러 개일 경우, BFS를 반복적으로 수행하므로  
 최악의 경우 전체 시간 복잡도는 O(K⋅N2) 입니다.  
 여기서 K는 곰이 먹을 수 있는 벌집의 개수입니다.

-하지만 N이 작고 입력 제한도 엄격하므로, 실제 실행 시간은 매우 빠릅니다.

기타 사항

-먹을 수 있는 벌집 후보들을 리스트에 저장한 후 우선순위에 따라 정렬하는데,  
이때 정렬의 시간 복잡도는 O(KlogK) 정도로 충분히 빠르게 처리됩니다.

-입력의 크기가 작고, 반복 횟수가 제한되어 있기 때문에  
BFS 기반의 반복 탐색 구조가 문제 해결에 매우 적절합니다.

결론  
- 본 알고리즘은 너비 우선 탐색(BFS)을 통해 곰의 행동을 시뮬레이션하며, 문제에서 요구하는 모든 조건을 정확하게 반영합니다. 입력 제한이 작기 때문에 시간과 공간 효율성 모두 우수하며, 실제 실행 시에도 빠르고 안정적으로 동작하는 코드입니다.

