**데이터구조 및 프로그래밍실습 3분반**

**설계 프로젝트**

학번 : 202111287

이름 : 김범기

본인의 Github 주소 : https://github.com/Bumk1018/data-structure-lab.git

(본 과제의 코드를 본인의 Github에 업로드 하세요)

**문제 1.**

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

버전1

리스트의 평균 mean은 input 리스트를 모두 더하고 리스트의 길이(리스트에 주어진 수의 개수)로 나누어 구한다.

리스트의 중앙값 median은 input.sort()를 통해 주어진 input 리스트를 정렬시키고 리스트의 길이가 5, 홀수로 주어지기 때문에 리스트의 값 중 리스트의 길이를 2로 나눠서 나온 수번째 값이다.

버전2(파이썬의 statistics 모듈 사용)

파이썬의 statistics 모듈 사용하여 평균은 statistics.mean()을 이용하여, 중앙값은 statistics.median()으로 이용하여 구한다.

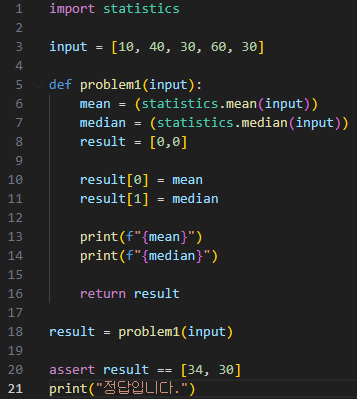
1. 본인이 작성한 파이썬 Code 캡처 이미지를 첨부하고 Algorithm Analysis를 수행하세요.

버전1, 버전2

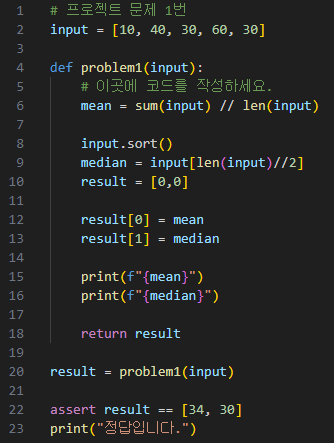
평균을 구하는 데 리스트의 모든 수를 더하므로 O(n)이고, 중앙값을 구할 때는 리스트를 정렬하는 데 O(n log n), 중앙값을 찾는 데 O(1)이므로 이 코드 전체 시간 복잡도는 O(n log n)이다.

+ 파이썬에서 sort() 함수는 Merge sort(병합 정렬)과 Insertion(삽입 정렬)을 결합한 Quick(하이브리드 방식)의 정렬 알고리즘을 사용한다.

버전1



버전2



**문제 2.**

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

버전1

‘(‘ 1개와 ‘)’ 1개가 짝을 이루어 올바른 괄호열을 만드므로 아래와 같은 반복문을 사용한다.

‘(‘가 필요한 개수를 변수 ‘open\_needed’, ‘)’가 필요한 개수를 변수 ‘closed\_needed’로 설정하고 둘다 초기값은 0으로 설정한다. 반복문을 사용하여 ‘(‘를 만날 때마다 ‘close\_needed’가 1씩 증가하고, ‘)’를 만날 때 ‘close\_needed’가 0보다 크다면 ‘close\_needed’를 1씩 감소시키고 ‘close\_needed’가 0보다 크지 않다면 ‘open\_needed’를 1씩 증가시킨다. 최종적으로 ‘open\_needed’와 ‘close\_needed’를 더하면 올바른 괄호열로 복원하기 위한 최소한의 괄호의 수가 나온다.

버전2

반복문 대신에 스택을 사용하여 표현할 수 있다. 빈 스택을 만들고 ‘(‘을 만나면 스택에 추가, ‘)’을 만났을 때는 스택의 맨 위에 ‘(‘가 있으면 이를 제거하고 스택이 비어 있으면 ‘)’을 추가한다. 괄호열을 모두 순회하고 난 후 스택에 남아 있는 괄호는 서로 짝을 이루지 못해 올바른 괄호열이 되지 못한 괄호들이므로 ‘(‘의 개수는 ‘(‘가 필요한 개수, ‘open\_needed’로 ‘)’의 개수는 ‘)’가 필요한 개수, ‘close\_needed’로 카운트해주고 이를 더하면 총 올바른 괄호열을 만들기 위한 최소한의 괄호의 수가 나온다.

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

버전1: 괄호열을 한 번 순회하면서 괄호의 개수를 카운트하기 때문에 시간 복잡도는 O(n)

버전2: 각 괄호를 처리하는 데 O(1)의 시간 복잡도가 소요되므로 전체 알고리즘의 시간 복잡도는 O(n)

버전1



버전2

**문제 3.**

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

우선 곰이 현재 위치에서 곰이 이동할 수 있는 위치를 탐색하기 위해 DFS를 사용한다. DFS를 사용하는 이유는 시작 노드에서 목표 노드까지의 최단 경로를 찾는 데 적합한 알고리즘이기 때문이다. 곰의 크기보다 작은 벌집을 발견하면 그 위치와 거리를 리스트에 추가하고 리스트를 거리, y좌표, x좌표 순으로 정렬하여 가장 가까운 벌집을 찾는다. Problem3 함수는 곰의 초기 위치를 찾고 BFS를 이용하여 먹을 수 있는 벌집을 찾고 가장 가까운 벌집으로 이동한다. 이동시간을 누적하고 벌집을 먹는 횟수를 증가시킨다. 또한 곰이 자신의 크기만큼 벌집을 먹으면 곰의 크기를 증가시킨다. 더 이상 먹을 벌집이 없을 때까지 이 과정을 반복한다. 마지막으로 출력은 곰의 초기 위치와 먹을 수 있는 벌집을 모두 먹었을 때의 총 시간을 출력한다.

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

bfs함수 일반적으로 O(노드의 수+간선의 수)의 시간 복잡도를 가지는데 이 문제에서 숲의 크기가 NxN이므로 노드의 수가 N2, 간선의 수는 각 칸에서 최대 4개의 방향으로 이동할 수 있으므로 대략 4N2이다. 따라서 bfs함수의 시간 복잡도는 O(N2)이다. Problem3 함수에서 ‘while’ 루프의 최악의 경우는 곰이 모든 벌집을 하나씩 먹을 때이고 이 경우 시간복잡도는 O(N2)인데 각 반복에서 bfs함수를 실행하므로 전체 시간 복잡도는 O(N4)이다.

