**데이터구조 및 프로그래밍실습 3분반**

**설계 프로젝트**

학번 : 202011967

이름 : 홍영우

본인의 Github 주소 : <https://github.com/youngwoo1206/-data-structure-lab-2024.git>

(본 과제의 코드를 본인의 Github에 업로드 하세요)

**문제 1.**

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

사용한 데이터 구조: list

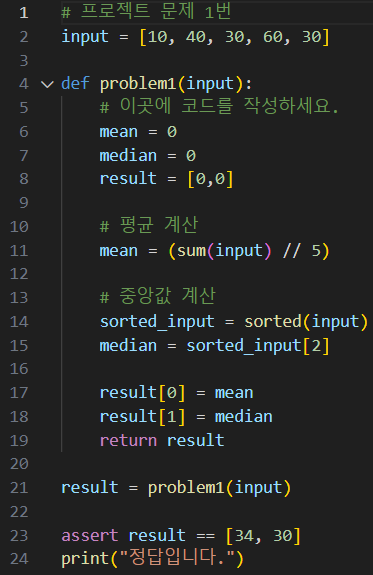
list는 순차적인 데이터를 나타내는데 사용되는 데이터 구조입니다.

주어진 문제1에서 한 줄에 하나씩 주어지는 자연수를 list에 차례대로 저장합니다.

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

**- 핵심 코드**

다섯 개의 자연수라고 했으므로 평균은 ((주어진 값의 합)/5) 입니다.

또한, 중간값은 sorted\_list의 3번째 요소인 sorted\_list[2]로 구합니다.

**- 시간 복잡도(Big-O notation)**

평균 계산: O(n)

sum(input)은 O(n)이고, //는 O(1)입니다.

중앙값 계산: O(n logn)

파이썬 정렬 함수의 시간복잡도는 O(n logn)이므로, sorted(input)은 O(n logn)입니다.

중앙값 찾기 sorted\_input[2]은 O(1)입니다.

정리

평균 계산: O(n)

중앙값 계산: O(n logn)

결과 반환, 대입 연산: O(1)

전체 시간 복잡도: O(n logn)

**문제 2.**

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

사용한 데이터 구조: stack

stack은 데이터를 LIFO 방식으로 저장하고 꺼내는 데이터 구조입니다.

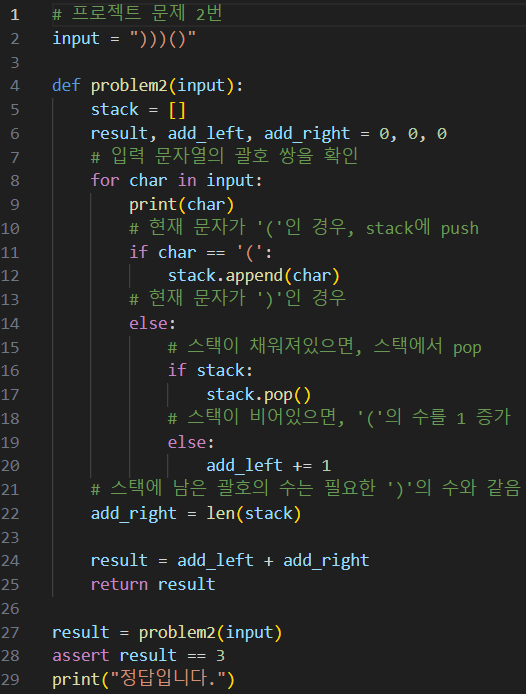
stack을 사용하면 시간 복잡도가 O(1)인 장점이 있습니다.

주어진 문제에서 '('와 ')'가 올바를 쌍을 이루도록 하기 위해서 둘 중 하나를 stack을 이용하여 push, pop하며 괄호가 올바른 쌍을 이루도록 합니다.

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

**- 핵심 코드**

'(”이면 stack에 push하고, ')'이면 stack에서 pop하거나 '('을 추가하고 stack에 '('이 남아있으면 ')'을 추가합니다. 이때 추가한 '('을 add\_left, ')'을 add\_right라고 합니다.

결과적으로, 필요한 괄호의 최소 갯수는 add\_left + add\_right 입니다.

**- 시간 복잡도(Big-O notation)**

반복문: O(n)

입력 문자열 input에 대해 n번 반복해야 하므로 O(n) 입니다.

스택: O(1)

스택의 push, pop연산은 각각 O(1)입니다.

정리

반복문: O(n)

스택: O(1)

결과 반환, 대입 연산: O(1)

전체 시간 복잡도: O(n)

**문제 3.**

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

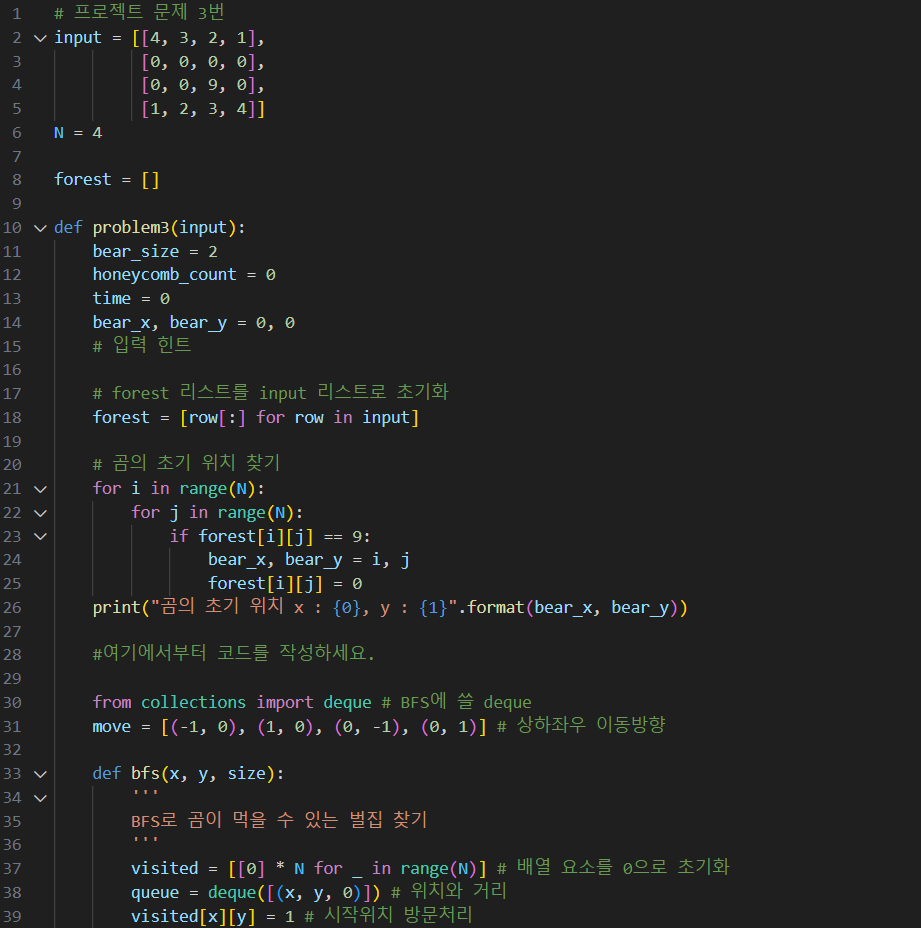
사용한 데이터 구조: list, deque

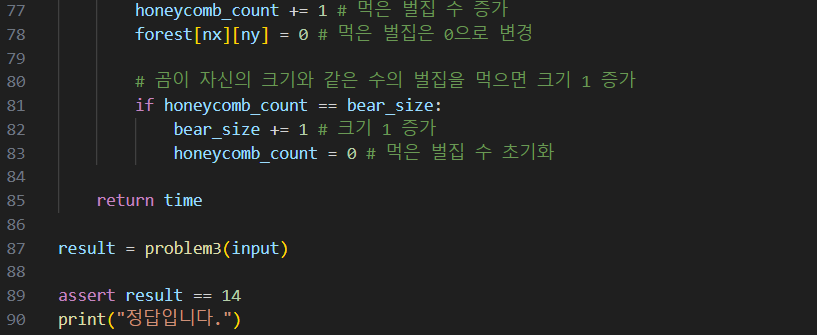
list, 2D list를 여러번 사용하여 데이터를 저장합니다.

list인 move, honeycombs와 2D list인 forest, visited를 사용합니다.

deque는 front와 rear에서 모두 입출력이 가능한 큐인 데이터 구조입니다.

BFS 탐색에 사용하며 시간복잡도가 O(1)인 장점이 있습니다. 위치와 거리를 저장합니다.

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



**- 핵심 코드**

input 리스트로 forest 리스트를 초기화하고, 초기 곰의 위치를 찾은 것을 바탕으로 코드를 작성했습니다.

우선, BFS에 쓸 deque를 불러오고, 상하좌우 이동방향 move를 설정했습니다.

**1.** BFS

곰이 먹을 수 있는 벌집을 찾습니다.

크기가 N인 공간에 대해 visited 배열과 큐(deque)를 초기화 합니다.

반복문에서 큐에서 위치와 거리 정보를 하나씩 받아서 상하좌우로 이동을 시도합니다.

위치가 유효하고 방문하지 않았으면 곰의 크기와 비교하여 방문하고 큐에 추가합니다.

만약 곰의 크기보다 작다면, 먹을 수 있는 벌집인 honeycombs 리스트에 추가합니다.

단, honneycombs 리스트에 추가할 때 (거리, 행, 열) 순서로 저장합니다.

**2.** sort

위 BFS에서 찾은 honneycomb 리스트를 정렬합니다.

honneycombs에서 (거리, 행, 열) 순서로 저장했기에 honneycombs.sort()로 정렬합니다.

python의 sort는 기본적으로 앞에 있는 인자를 우선으로 하기에 이를 활용했습니다.

정렬한 후 가장 가까운 벌집인 honeycombs[0]을 반환합니다.

**3.** 메인루프

BFS를 호출하여 가장 가까운 벌집을 반환받습니다. 함수의 반환값이 None이면 먹을 수 있는 벌집이 없으므로 메인루프를 종료합니다.

honey\_n의 값을 바탕으로 곰의 위치와 이동거리(시간)을 업데이트합니다. 또한, 먹은 벌집 수를 증가시키고 먹을 벌집은 fores 리스트에서 0으로 바꿔줍니다.

마지막으로, 먹은 벌집수인 honeycomb\_count가 곰의 크기 bear\_size와 같으면 곰의 크기를 1증가시키고 먹은 벌집의 수를 0으로 초기화합니다.

**- 시간 복잡도**

forest 리스트: O(n^2)

곰의 초기 위치 찾기: O(n^2)

BFS: visited 배열은 O(n^2), queue는 O(1)이고

while 루프는 최악의 경우 모든 위치에서 상하좌우를 탐색하므로 O(4N^2)입니다.

따라서 O(n^2) 입니다.

sort: 정렬의 시간복잡도는 O(m logm), m은 먹을 수 있는 벌집의 수로 n^2보다 작으므로 O(n^2)인 BFS보다는 영향이 적다고 볼 수 있습니다.

메인루프: BFS는 O(n^2)이고 최악의 경우 이를 n^2번 반복해야합니다.

따라서 O(n^4) 입니다.

전체 시간복잡도: O(n^4)