**데이터구조 및 프로그래밍실습 3분반**

**설계 프로젝트**

학번 : 202117395

이름 : 김동훈

본인의 Github 주소 : [kdh044/DS2025](https://github.com/kdh044/DS2025)

(본 과제의 코드를 본인의 Github에 업로드 하세요)

**문제 1.**

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

본 문제를 풀기 위해서 사용한 데이터 구조는 리스트입니다. 그리고 중앙값을 구하기 위해 정렬 알고리즘을 사용하였습니다.

우선 리스트에 담긴 값들의 평균을 구하기 위해서 리스트의 값들을 모두 더하는 sum() 함수와 리스트의 길이(개수)를 반환하는 len() 함수를 사용한 다음, 총합을 길이로 나누어 평균을 간단히 구했습니다.

중앙값을 구하기 위해서는 일단 리스트의 값들을 정렬해야 했습니다. 정렬에는 퀵정렬, 버블정렬, 힙정렬 등 많은 정렬방법이 있지만, 저는 간단하게 파이썬의 내장함수인 sort() 함수를 이용하여 리스트 값들을 오름차순으로 정렬하였습니다. sort()는 Timsort라는 알고리즘을 사용하여 정렬합니다.

정렬한 후에 리스트의 요소가 5개이므로 중앙값은 3번째 요소입니다(인덱스 2). 따라서 정렬된 리스트의 3번째 요소를 median 변수로 저장하였습니다.

계산된 평균과 중앙값을 result 리스트에 넣고 반환하였습니다.

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

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**시간복잡도 분석**

N의 개수는 상수 = 5이기 때문에

T(N) = 5 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 5log5 = 13 + 5log5

Algorithm Analysis를 위해 Big-O Notation을 수행하면:

* 5log5 ≈ 11.6 (상수)
* 13 (상수)

모든 항이 상수이므로 \*\*시간복잡도는 O(1)\*\*입니다.

최선/평균/최악의 경우 분석

입력 크기가 5로 고정되어 있기 때문에:

* 최선의 경우 (이미 정렬된 입력): O(1)
* 평균의 경우 (무작위 입력): O(1)
* 최악의 경우 (역순 정렬된 입력): O(1)

**결론**

이 알고리즘은 고정된 입력 크기를 가져 입력 상태에 관계없이 항상 같은 시간복잡도 O(1)을 가지는 알고리즘입니다.

**문제 2.**

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

스택을 사용하여 푸는 것이 구현에 있어서 쉽지만, 알고리즘 효율을 위하여 여는괄호 "("와 닫는괄호 ")"를 카운팅하는 단순 정수형 변수 두 개를 사용하였습니다.

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

텍스트, 스크린샷, 멀티미디어, 소프트웨어이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

이 문제를 풀기 위해서 스택을 이용하여 푸는 것 대신 그리디 알고리즘을 사용하였습니다.

매 순간 최적이라고 생각되는 선택을 하는 알고리즘으로, 지역적 최적해가 전역적 최적해가 되는 경우에 사용할 수 있습니다.

이 문제에서는 각 괄호를 순회하면서 닫는 괄호 ")"를 만났을 때, 매칭 가능한 여는 괄호 "("가 있다면 즉시 매칭시키는 것이 항상 최적입니다. 나중을 위해 매칭을 미루는 것보다 지금 당장 매칭하는 것이 더 유리하기 때문입니다.

**스택 대신 그리디를 선택한 이유**

* 공간 복잡도: 스택 O(n) vs 그리디 O(1) 으로 그리디가 더 효율적임
* 시간 복잡도: 둘 다 O(n)이지만, 스택은 push/pop 연산을 사용하여 오버헤드 발생 가능함
* 연산 효율: 스택의 경우 리스트 자료구조를 이용하기 때문에 append(), pop() 메서드 호출이 필요하지만, 그리디는 단순 정수 덧셈 뺄셈으로만 구현이 가능함.

**코드 동작 원리**

각각 매칭 대기중인 "("와 ")"를 초기화합니다. 그리고 나서 주어진 괄호들의 문자열(문제에서는 경태가 깨트린 장난감입니다.)를 순회하면서 "("를 만나면 open\_count의 개수를 1개씩 늘려 매칭을 대기하고 ")"를 만나면 매칭을 성공하여 open\_count의 개수를 한 개 줄입니다. 단 open\_count가 0이라면 close\_count를 1개 늘립니다. 전부 다 순회하고 나서 나오는 open\_count와 close\_count가 문제에서 요구하는 고장난 장난감을 고치기 위한 괄호의 개수입니다. 단 문제에서는 앞에 붙이는 괄호의 개수와 뒤에 붙이는 괄호의 개수를 구분짓지 않고 총 합을 구하라 하였으므로 더한 뒤 반환하여 출력합니다.

**문제 3.**

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

본 문제를 풀기 위해 다음과 같은 데이터 구조들을 사용하였습니다:

**1) 큐(Queue) - deque 모듈**

* **목적**: BFS(너비 우선 탐색) 구현
* **구현**: collections.deque를 사용하여 FIFO 방식으로 동작
* **효율성**: 리스트의 pop(0)은 O(N)이지만, deque의 popleft()는 O(1)로 효율적
* **저장 내용**: (x좌표, y좌표, 이동거리) 튜플을 저장하여 현재 위치와 거리 정보 관리

**2) 2차원 리스트**

* **forest**: 게임 맵을 표현하는 N×N 2차원 배열로, 벌통의 크기와 위치 정보 저장
* **visited**: BFS 탐색 시 방문한 위치를 체크하는 불린 배열로, 중복 방문 방지

**3) 리스트(동적 배열)**

* **edible(변수)**: 현재 곰이 먹을 수 있는 벌통들의 정보를 임시 저장
* **저장 내용**: (거리, x좌표, y좌표) 튜플들을 저장
* **활용**: sort() 메서드로 정렬하여 가장 가까운 벌통을 우선 선택

이러한 데이터 구조들을 조합하여 곰의 현재 위치에서 먹을 수 있는 가장 가까운 벌통을 효율적으로 찾는 BFS 알고리즘을 구현하였습니다.

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

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**BFS 알고리즘 선택 이유**

본 문제에서는 곰이 "거리가 가장 가까운 벌집을 먹으러 간다"는 조건이 있기 때문에 DFS 대신 BFS를 사용하여 구현하였습니다. DFS는 깊이 우선 탐색으로 최단거리를 보장하지 않는 반면, BFS는 너비 우선 탐색을 통해 최단거리 탐색을 보장하기 때문입니다.

**시간복잡도 분석**

BFS 한 번 수행의 시간복잡도는 O(N²)입니다. N×N 크기의 2차원 배열에서 모든 셀을 최대 한 번씩 방문하며, 각 셀마다 상하좌우 4방향을 확인하지만 전체적으로는 O(N²)의 시간복잡도를 가집니다. 전체 알고리즘의 시간복잡도는 O(M × N²)이며, 여기서 M은 곰이 먹는 총 벌집의 개수로 각 벌집을 찾기 위해 BFS를 한 번씩 수행하기 때문입니다.

**알고리즘 동작 과정**

메인 루프에서는 먼저 곰의 현재 위치에서 BFS 탐색을 시작하여 큐를 사용해 시작 위치로부터의 거리를 함께 추적합니다. 먹을 수 있는 가장 가까운 벌집을 찾기 위해 BFS 함수를 반복 호출하며, 벌집이 발견되면 해당 위치로 이동하여 벌집을 먹고 시간과 벌집 수를 업데이트합니다. 이 과정에서 방문 배열을 통해 이미 방문한 노드의 재방문을 방지하고, 조건을 만족하는 벌집을 발견하면 후보 목록에 추가한 후 큐가 비워질 때까지 모든 도달 가능한 노드를 탐색합니다.

BFS 탐색 과정에서는 곰의 현재 위치를 큐에 추가하고 방문 체크를 한 후, 상하좌우 4방향으로 확산하며 탐색을 진행합니다. 각 위치에서 맵 경계 내부인지, 이미 방문했는지, 곰이 지나갈 수 있는지를 확인하며, 먹을 수 있는 벌집을 발견하면 거리 정보와 함께 후보 목록에 추가합니다. 최종적으로 거리가 가장 가까운 벌집 중에서 위쪽과 왼쪽을 우선하여 선택합니다. 이러한 과정을 더 이상 먹을 수 있는 벌집이 없을 때까지 반복하여 곰이 총 몇 초 동안 벌집을 털 수 있는지 계산합니다.

**최단거리 보장 원리**

BFS는 레벨별 탐색을 수행하여 자연스럽게 최단거리를 보장합니다. 시작점에서 거리 1, 2, 3 순서로 동심원 형태로 확산하며, FIFO 큐 구조를 통해 먼저 발견된 위치를 먼저 처리합니다. 이러한 특성으로 인해 특정 위치에 처음 도달한 경로가 반드시 최단경로가 되며, 이를 통해 문제에서 요구하는 "가장 가까운 벌집 우선 선택" 조건을 정확하게 구현할 수 있었습니다.