인공지능 과제 1

03분반

2021320122 김정우

Explanation (first of all))

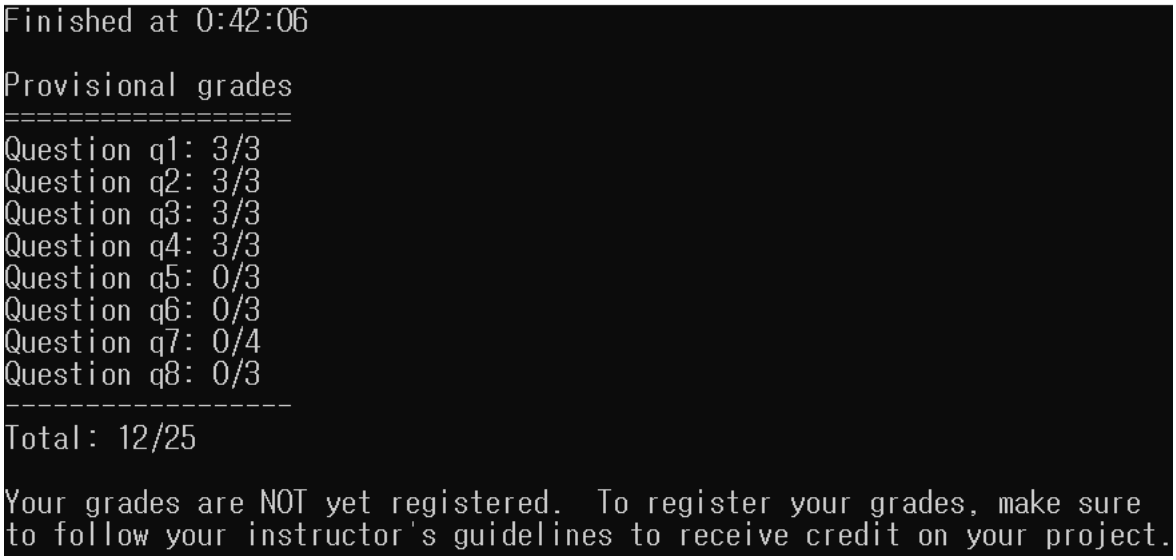
1. Dfs : start state을 stack에 넣고 시작한다. Stack 맨 위의 position(코드 내 변수명은 cur)을 pop(탐색; 탐색한 원소들은 모두 visit에 체크한다) 하고, 인접 노드들을 전부 stack에 push 한다. stack에는 그 position까지 진행한 direction들이 ans_dir로 전부 묶여 있다. 만약 현재 position이 goal이라면 ans_dir를 return한다.

2. Bfs : 너비 우선 탐색이기에 queue를 이용하는 것 이외엔 크게 달라지는 것이 없다. Dfs와 Bfs는 공통점이 있는데, cost를 알고리즘에 이용할 필요가 없다.

3. UCS : Dfs & Bfs와 다르게, 현재까지 진행하는데 필요한 cost를 저장하면서 cost 기준으로 min_priorityQueue 구조를 이용한다. Min_priorityQueue 구조 특성상 맨 위에 minimum cost가 오기에 매번 pop하면서 현재 position이 goal인지 확인해주면 된다. Goal이 아니라면 방문한 것이므로, visit에 추가하고 인접한 node들에 지금까지의 cost + 그 node로 가는데 필요한 cost를 해서 인자로 넘겨준다.

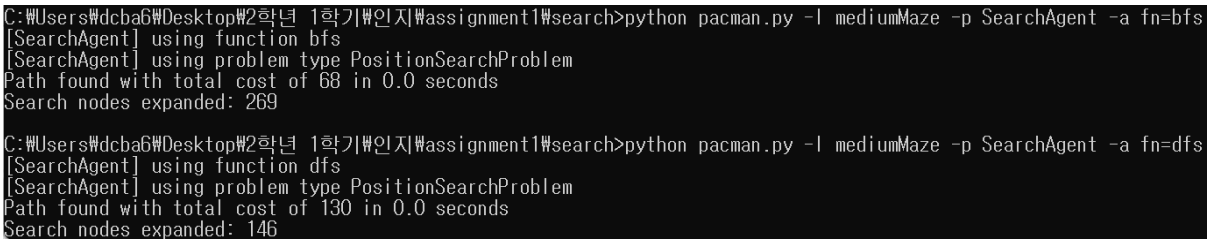
4. A* search : ucs의 cost에 heuristic이 추가된다. 기본적으로 주어진 heuristic은 Manhattan heuristic으로, 벽이 없다고 가정했을 때 goal까지의 manhattan distance를 더해 rough하게 search의 효율성을 높이는 방법이다.

B. autograder.py 결과 캡쳐



C. Three discussions on three different algorithms(DFS, BFS, A*) when playing Pacman

1. Discuss which algorithm is better between DFS or BFS. (In mediumMaze)



bfs를 보면 optimal solution을 찾기에 68의 cost를 사용하고 269개의 node search가 있었다.

반면 dfs에선 약 두 배 가량의 cost인 130을 소비했지만, 146개의 node search가 존재했다.

이처럼 mediumMaze에선 map의 크기 자체가 넓지 않기에, bfs와 dfs의 fringe space 차이가 크게 나지 않는다. 따라서 bfs가 더 효율적인 search 알고리즘이라고 보인다.

2. The proposed heuristic function is Manhattan. Is there any other heuristic function that is more efficient? Discuss specific cases where your algorithm works effectively.

매 position마다 goal까지의 실제 거리를 bfs를 통해 알아낸 뒤, 그것을 h값으로 두면 최소의 fringe space를 할당할 수 있다. 그러나 매 position마다 bfs를 구현하는 것은 시간복잡도 측면에서 매우 비효율적이 되기에, fringe space에서 얻을 수 있는 이득이 사실상 상쇄될 것 같아 다른 방법을 고민했다.

솔직하게, 번뜩이는 아이디어는 떠오르지 않았다. 그러나 고민 중 Manhattan heuristic의 영향력을 높이면 조금은 더 효율적인 알고리즘이 되지 않을까 생각했고, 이에 myHeuristic 함수에서 manhattanHeuristic에 가중치를 두어 return해봤다.

```
def myHeuristic(position, problem, info = {}):

    return 10 * manhattanHeuristic(position, problem, info)
```

가중치가 없을 때(1일 때)는 549개의 Search nodes expanded가 발생하는데,

가중치가 2, 3, 4, 5, 6 ... 올라감에 따라 510, 510, 503, 495, 482, ... 하고 서서히 줄어들었고, 어느 순간부터는 466을 마지막으로 줄어들지 않았다. 끝에는 제곱까지 시켜봤지만 466이 마지막이었다.

그럼에도 549개에서 466개로 줄어든 것은 꽤나 유의미한 발전이라고 볼 수 있다.

3. Ask yourself one question and answer.

   왜 일정 수준 이상의 가중치부터는 search nodes가 줄어들지 않았을까?

   : f = g + h 에서, h의 영향력이 커지면 결국 f 가 h와 근사한 값이 되기 때문에, h 자체가 변하지 않는 이상 결국 fringe는 특정 하한선 아래로 떨어질 수가 없는 것 같다.

D. Capture the result of A* algorithm with your implemented heuristic function.

```
def myHeuristic(position, problem, info = {}):

    return manhattanHeuristic(position, problem, info) ** 2
```

적당한 Heuristic을 생각해내지 못해 매우 아쉽다.