인공지능 1차 과제 보고서

2013011640 정대한

1. 코드 설명

1-1.기타 함수 설명

Def is_inside(y,x,numbers) : x 좌표와 y 좌표가 미로 내에 있는지를 확인합니다. (y : 행, x : 열, numbers : 미로(2차원 배열))

Def print_maze(numbers,visited,visited_1) : 최단 경로를 숫자 5로 바꿔주고, length와 time을 계산하여 txt파일로 출력시킵니다. (numbers : 미로, visited : key를 찾기 전의 방문 경로, visited_1 : key를 찾은 후의 방문 경로)

Def key_end_location(numbers) : key의 위치와 도착점의 위치를 확인합니다.

(numbers : 미로)

Def heu(x,y,x1,y1) : (y,x) 와 (y1,x1) 까지의 대각선 거리를 구합니다.

(heuristic 함수는 (y,x) 에서 (y1,x1) 까지 가는 데 걸리는 대각선 최단 경로로 정했습니다.)

Def A_sup(x,y,sx,sy,x1,y1) : 미로 시작점에서 현재까지 움직인 위치까지의 heuristic 값과 현재까지 움직인 위치에서 key까지의 heuristic 값을 더해주거나, key에서 현재까지 움직인 위치까지의 heuristic 값과 현재까지 움직인 위치에서 도착점까지의 heuristic 값을 더해줍니다.

Def insert_heap(heap,i) : greedy best-first search 와 A* search 에서 이용되는 priority queue 에서, 경로가 append 될 때마다 min-heapify를 합니다.

Def delete_heap(heap) : 마찬가지로 priority queue에서 값이 pop 될 때 마다 min-heapify를 합니다.

Def dfs(depth) : dfs 알고리즘을 적용하여 미로를 찾아주는 함수입니다. IDS

알고리즘을 사용할 때 이용됩니다.

1-2. 알고리즘 코드 설명

Breadth-first search, Iterative deepening search, greedy best-first search, A* search 네 가지의 알고리즘을 이용하여 탐색 노드의 개수(time) 을 비교하여 가장 효율적인 알고리즘을 층마다 적용하였습니다.

(그러나, Greedy best-first search는 optimal 을 항상 보장하진 못하므로 time만 계산하였고 알고리즘을 그 층에 적용하진 않았습니다.)

2. 사용 알고리즘, 최단경로, 탐색한 노드의 개수

1층 – 알고리즘 : **A* search** 적용 **(최단경로 length : 3850)**

실행방법 : def first_floor() 함수 안의 f 변수에 파일경로와 파일명을 적은 뒤, first_floor() 을 main에 적으면 실행됩니다.

동작 : f 경로를 따라 나오는 파일을 numbers라는 2차원 배열에 파일 입력 합니다

-> 첫번째 행에 시작점의 위치가 어디 있는지를 확인합니다

-> 키를 찾기 전까지의 탐색 경로를 확인하기 위한 visited 2차원 배열, 키를 찾은 후 키 위치에서 도착점까지의 탐색 경로를 확인하기 위한 visited_1 2차원 배열을 생성합니다(0으로 초기화)

-> key의 위치와 도착점의 위치를 확인해둡니다

-> 튜플을 인자로 가지는 배열 Priority Queue 를 생성하여 시작점의 행, 열,f(n) 을 append 합니다

-> 반복문에 들어가서 A* search 알고리즘을 적용합니다

-> 키를 찾기 전에는 자식 노드로 방문할 때마다 숫자 1씩 증가시켜 visited 배열에 넣어줍니다. 탐색노드의 개수와 방문경로를 알 수 있습니다
-> 키를 찾은 후에는 동일한 적용을 visited_1 배열에 해줍니다.

-> 완성된 Visited_1 배열의 도착점에는 키 위치에서 도착점까지의 최단경로의 거리가 몇인지가 찍혀있고, 그 도착점의 값에서 1씩을 감소시켜가며 길을 거슬러 따라가면 그 길이 도착점에서 키 까지의 최단경로입니다. 또한 완성된 visited 배열의 키 위치에서 동일한 작업을 수행하면 그 길이 키 위치에서 시작점까지의 최단경로입니다. 따라서 Visited 배열 Visited_1 배열 두 개를 통해 최단경로의 거리, 최단경로, numbers의 숫자 2를 숫자

5로 바꾸기가 모두 가능합니다. 이 작업을 완료한 후 출력합니다.

BFS	IDS	A* search	Greedy best-first s
6752	X	6283	5840(X)

➔ 위의 표는 각 알고리즘을 적용했을 때 time을 계산한 것으로, A* search의 탐색 노드의 수가 가장 적은 것을 확인하여 A* search를 적용하였습니다. 미로의 행렬 크기가 너무 커서 IDS는 적용이 되지 않았습니다.

2층 – 알고리즘 : **Iterative deepening search** 적용**(최단경로 length : 758)**

실행방법 : def second_floor() 함수 안의 f 변수에 파일경로와 파일명을 적은 뒤, second_floor() 을 main에 적으면 실행됩니다.

동작 : **/***(f 경로를 따라 나오는 파일을 numbers라는 2차원 배열에 파일 입력 합니다 -> 첫번째 행에 시작점의 위치가 어디 있는지를 확인합니다 -> 키를 찾기 전까지의 탐색 경로를 확인하기 위한 visited 2차원 배열, 키를 찾은 후 키 위치에서 도착점까지의 탐색 경로를 확인하기 위한 visited_1 2차원 배열을 생성합니다(0으로 초기화) -> key의 위치와 도착점의 위치를 확인해둡니다) ***/** (1층과 동일한 과정입니다)

-> Stack 배열을 생성하여 현재위치의 행,열을 튜플 형식으로 append 합니다. -> iterative deepening search를 적용하여 제한 깊이를 1씩 늘려가며 동일한 작업을 수행합니다. (도착점을 못 찾을 시 second_floor() 함수 내에서 depth 를 1씩 늘려가며 dfs(depth) 함수를 계속적으로 호출합니다.)

-> **/*** (키를 찾기 전에는 자식 노드로 방문할 때마다 숫자 1씩 증가시켜 visited 배열에 넣어줍니다. 탐색노드의 개수와 방문경로를 알 수 있습니다 -> 키를 찾은 후에는 동일한 적용을 visited_1 배열에 해줍니다. -> 완성된 Visited_1 배열의 도착점에는 키 위치에서 도착점까지의 최단경로의 거리가 몇인지가 찍혀있고, 그 도착점의 값에서 1씩을 감소시켜가며 길을 거슬러 따라가면 그 길이 도착점에서 키 까지의 최단경로입니다. 또한 완성된 visited 배열의 키 위치에서 동일한 작업을 수행하면 그 길이 키 위

치에서 시작점까지의 최단경로입니다. 따라서 Visited 배열과 Visited_1 배열 두개를 통해 최단경로의 거리, 최단경로, numbers의 숫자 2를 숫자 5로 바꾸기가 모두 가능합니다. 이 작업을 완료한 후 출력합니다.) ***/** (1층과 동일한 과정입니다) 이하 주석은 생략합니다.

BFS	IDS	A* search	Greedy best-first s
1727	1150	1204	1053(X)

➔ 근소한 차이로 IDS의 time 더 적은 것을 확인하였습니다. 따라서 IDS 를 적용하였습니다.

3층 – 알고리즘 : **A* search** 적용**(최단경로 length : 554)**

실행방법 : def third_floor() 함수 안의 f 변수에 파일경로와 파일명을 적은 뒤, third_floor() 을 main에 적으면 실행됩니다.

동작 : 알고리즘이 1층과 같아 동작이 동일합니다. 미로 행렬의 크기만 다른 것이므로 처음 visited 배열과 visited_1 배열의 크기를 입력 받은 미로의 크기에 맞춰서 생성합니다.

BFS	IDS	A* search	Greedy best-first s
1008	982	773	669(X)

➔ 근소한 차이로 A* search의 time이 IDS time 보다 앞섰습니다. 따라서 A* search 알고리즘을 적용하였습니다.

4층 – 알고리즘 : **A* search** 적용**(최단경로 length : 334)**

실행방법 : def fourth_floor() 함수 안의 f 변수에 파일경로와 파일명을 적은 뒤, fourth_floor() 을 main에 적으면 실행됩니다.

동작 : 알고리즘이 1층과 같아 동작이 동일합니다.

BFS	IDS	A* search	Greedy best-first s
597	864	502	444(X)

➔ 근소한 차이로 A* search의 time이 BFS time 보다 앞섰습니다. 따라서 A*

search 를 적용하였습니다.

5층 – 알고리즘 : **A* search 적용(최단경로 length : 106)**

실행방법 : def fifth_floor() 함수 안의 f 변수에 파일경로와 파일명을 적은 뒤, fifth_floor() 을 main에 적으면 실행됩니다.

동작 : 알고리즘이 1층과 같아 동작이 동일합니다.

BFS	IDS	A* search	Greedy best-first s
236	313	136	127(X)

➔ 더욱 근소한 차이로 A* search의 time이 BFS time 보다 앞섰습니다. 따라서 A* search 를 적용하였습니다.

3. 부록 함수 설명

gbfs() -> f 의 파일 경로를 정해주어 main 함수에서 gbfs() 를 적어 실행하면 greedy best-first search 를 적용하여 미로를 찾아 gbfs_sample_output.txt로 파일 출력 해줍니다.

bfs() -> f의 파일 경로를 정해주어 main 함수에서 bfs()를 적어 실행하면 breadth first search 를 적용하여 미로를 찾아 bfs_sample_output.txt로 파일 출력 해줍니다.