2014312516 김형준

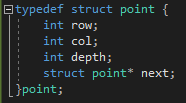
김형식 교수님

자료구조개론

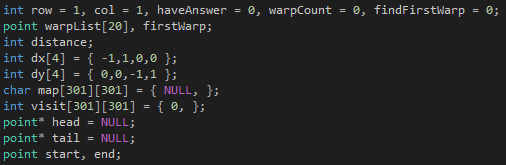
Assignment2



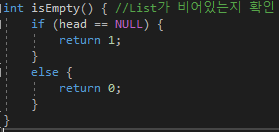
표준입출력의 stdio.h 와 malloc과 free함수를 위한 stdlib.h를 사용하였다.



사용한 구조체이다. 가로, 세로를 표한하기위한 row, col변수와, end까지 도달하기 위한 distance를 계산하기 위한 depth, linked list로써의 사용을 위한 구조체 포인터 next로 이루어져있다.

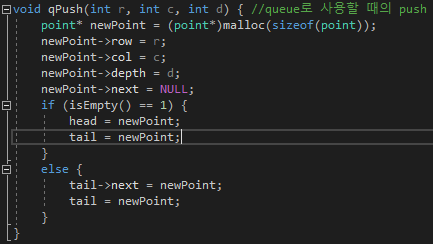


사용한 전역변수이다. main함수가 아닌 함수들에서도 이용하기 위하여 전역으로 선언해주었다. 길찾기 문제에서 dx[4], dy[4]의 사용은, 방학 때 수강한 삼성sds 알고리즘 특강에서 배웠었다. 제한된 n,과 m의 크기가 300으로 작기에 미리 크기를 설정해주었다.



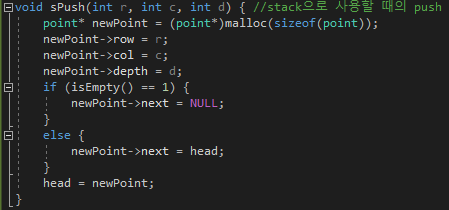
List가 비어있는지 확인하는 함수이다. Queue나 stack이라고 적지 않은 이유는, 이 파일에서는 list를 stack으로도, queue로도 사용하기 때문이다.

Performance analysis : O(1)



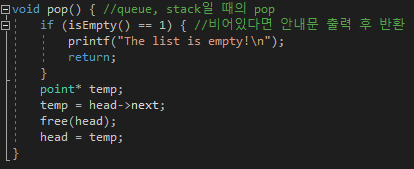
Queue로 사용할 때에 list를 삽입하는 push함수이다. 비어있을 때의 경우와, 그렇지 않을 때의 경우로 나누어 설정하였다. 새로운 원소는 tail로 설정된다. 이는 FIFO를 위함이다.

Performance analysis : O(1)



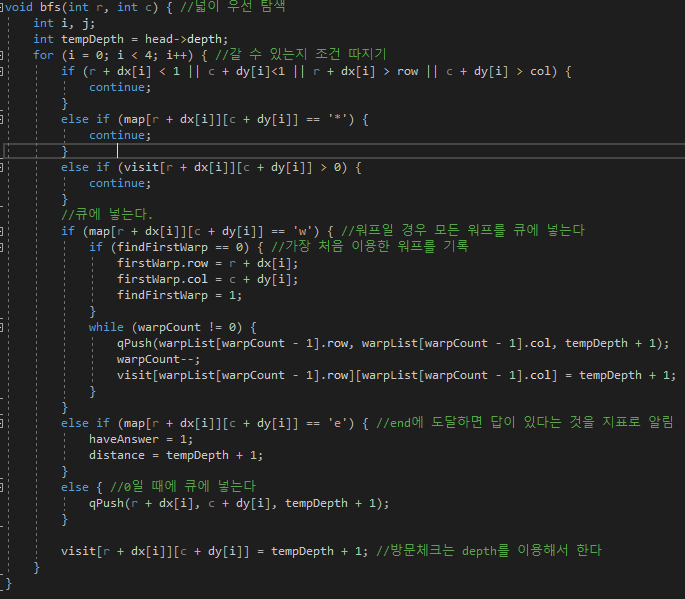
stack으로 사용할 때에 list를 삽입하는 push함수이다. 마찬가지로 비어있을 때와, 비어있지 않을 때로 나누어 삽입 과정을 진행하였다. Queue와 달리 새로운 원소는 head로 설정된다. 이는 LIFO를 위함이다.

Performance analysis : O(1)



가장 앞 원소를 제거하는 pop함수이다. stack일 때나 queue일 때나 똑같이 작용하여, push와는 달리 나누지 않아도 되었다. head쪽의 원소를 제거하게 된다.

Performance analysis : O(1)



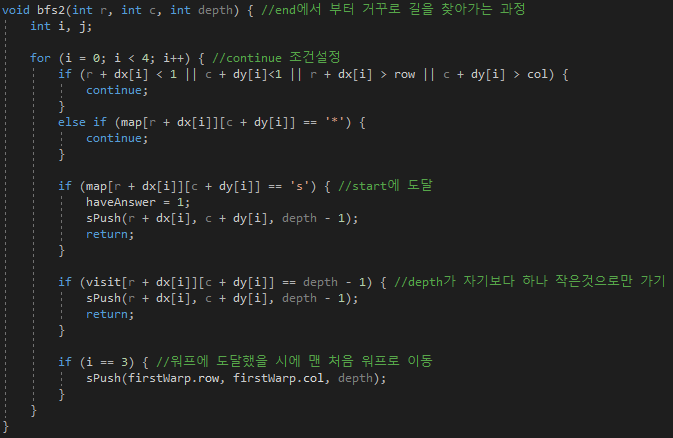
넓이 우선 탐색법인 bfs함수이다. 우선 갈 수 있는 길인지 조건을 먼저 따져준다. ‘\*’이거나, map의 범위를 초과하거나 방문기록이 있다면 갈 수 없으므로 continue를 진행한다.

그 후의 경우는 e,w,0의 경우이다. 우선 w로, 워프에 도달한다면, 가장 첫 워프인 것을 기록한 후에 모든 워프를 큐에 넣는다. 가능한 모든 워프의 경우를 따져주어야 하기 때문이다.

e일 때에는 끝 점에 도달한 것이므로 haveAnswer변수를 통해 답을 구했음을 체크한다. 또한 답을 나타내는 distance에 현재 깊이+1을 설정한다.

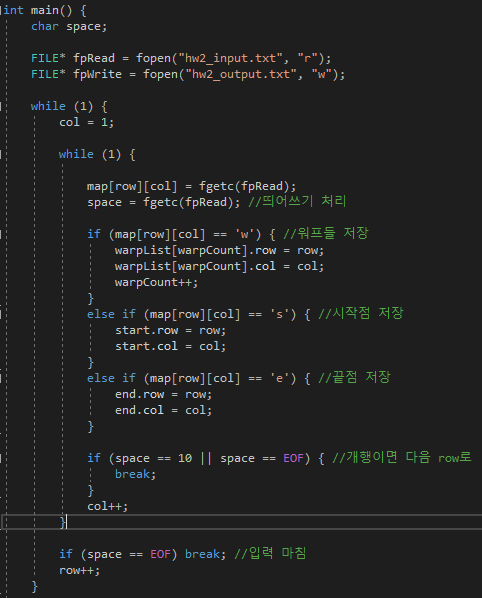
e나 w가 아닌 0일 때에는 평범하게 queue에 넣으면 된다. 방문체크는 공통된 과정이기에 if문 밖에 설정하였다. 방문체크를 할 때에는 단순히 0과 1로만 하는 것이 아니라, depth 자체를 기록하였다. 후에 길을 거꾸로 찾아오기 위함이다.

Performance analysis : O(4)



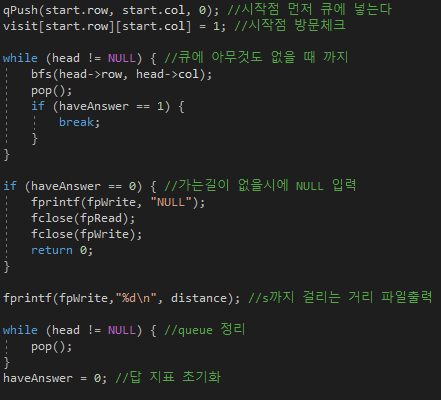
거꾸로 e에서부터 s로 찾아올 때를 위한 함수이다. bfs와는 크게 관계가 없지만 이름을 간단하게 bfs2로 설정하였다. s에 도달한다면 마찬가지로 haveAnswer를 통해 답을 찾았음을 체크해준다. bfs와는 달리 visit 2차원 배열에서 depth 가 하나 더 작은 곳으로만 이동한다. 만약 4방향을 전부 다 둘러봐도 depth-1 인 점이 존재하지 않는다면 워프를 이용한 것이므로 맨 처음 워프로 이동한다.

Performance analysis : O(4)



main함수이다. 입출력 할 파일들에 파일포인터를 연결한다. 입력파일에서 문자들을 하나 씩 읽어오면서 map에 저장한다. 띄어쓰기를 처리하기 위해 space 변수를 도입하였다. input파일을 분석해 본 결과 개행 문자가 ‘10’(\n) 으로만 주어져 있어(window의 개행\r\n과 달리) 10일 때에만 개행문자로 인식해주면 되었다. EOF에 도달하면 입력을 마친다. 입력 받을 때에 시작점, 끝점, 워프들의 위치를 저장한다.

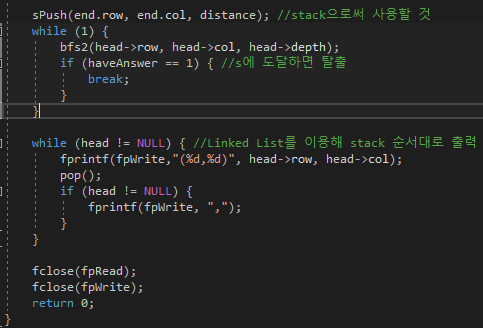
Performance analysis : O(row\*col)



그 후 bfs를 진행하기 위해 시작점을 먼저 큐에 넣는다. 방문체크 또한 해준 후에 queue에 아무 것도 존재하지 않을 때 까지 bfs를 진행해준다. 만약 그 과정에서 답을 찾으면 while문을 탈출한다. Queue에 아무것도 없을 때 까지 진행하여도 답을 찾지 못했다면 파일에 NULL을 출력하고 main함수를 마친다.

답을 찾았다면 파일에 우선 그 distance를 출력하여준다. 그 후 list를 stack으로 다시 사용하기 위해 모든 원소를 pop해 준다.

Performance analysis : O(row\*col)



그 후, stack으로써 끝 점을 삽입하여 준 후에, 위에서 만든 bfs2함수를 이용하여 e에 도달하기 까지 지나온 길을 stack에 삽입한다. 전부 탐색하고나면 파일에 나중에 들어온 순서대로 row,col을 출력하여준다. 전부 마친 후에 파일포인터를 닫고 함수를 마친다.

Performance analysis : O(distance)

전체 Performance analysis : O(row\*col)

List List의 push와 pop의 time efficiency가 매우 좋아, 결국 입력받은 파일의 가로행의 개수와 세로행의 개수에 따라 performance가 결정된다. 하지만 최대가 300인 것으로 미루어 보았을 때에

채 1millisecond도 걸리지 않는 빠른 performance를 가지는 프로그램이 된다.

공간복잡도 : char 2차원 배열이 row\*col 만큼 있기에 O(row\*col)byte가 필요하다.

또한 col과 row를 저장하여야 하기에 4 \* 2 = 8byte가 필요하다.

stack 과 queue의 worst case로는 벽이 전체 map 의 1/2씩 존재하여 가장 긴 길을 지나는 것이므로 각각 1/2 \* sizeof(map크기) \* sizeof(struct point)가 되어, 2 \* 1/2 \* row\*col \* (int 3개, 포인터 1개 = 20byte) = O(20\*row\*col) 가 된다.

이를 모두 더하면 O(row\*col)로 공간복잡도의 표현이 가능하다.

