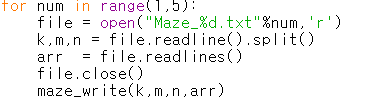
Assignment1 보고서

컴퓨터 소프트웨어 학부

2015004439

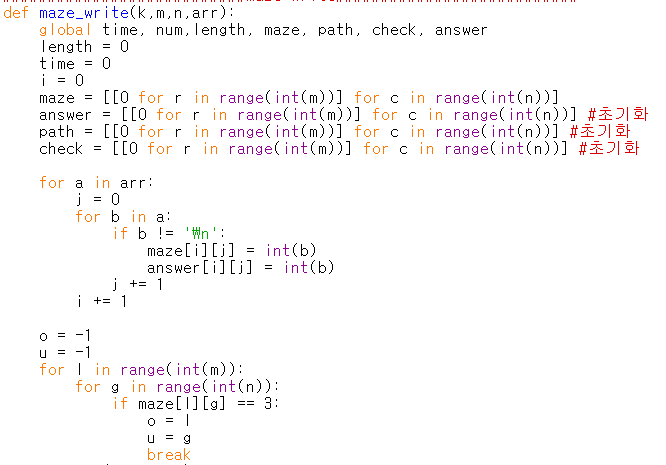
김재홍

* 코드 설명 및 함수 설명

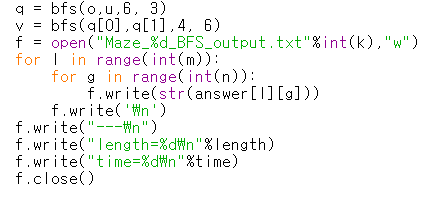


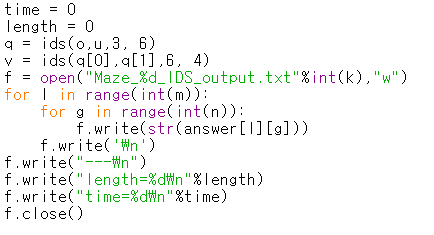
현재 미로가 Maze\_1 부터 4까지 있기 때문에 반복문을 통해 읽어 옵니다.(Maze\_4는 pdf에 있는 test 파일) k는 미로의 번호, m과 n은 미로의 크기를 나타냅니다.

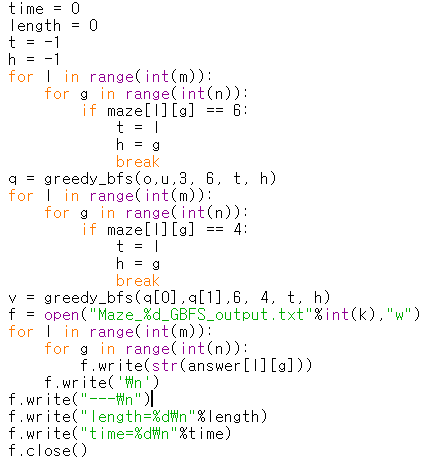
1. maze\_write 함수



maze\_write 함수는 읽어온 미로 파일을 저장하여 각각의 알고리즘을 실행하는 함수 입니다. 위 그림은 입력 받은 미로 파일을 읽어 각각 maze 리스트와 output파일에 기록될 answer리스트에 각각 저장하고 시작점이 어디인가를 찾는 코드 입니다.

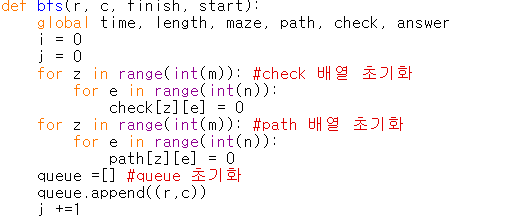




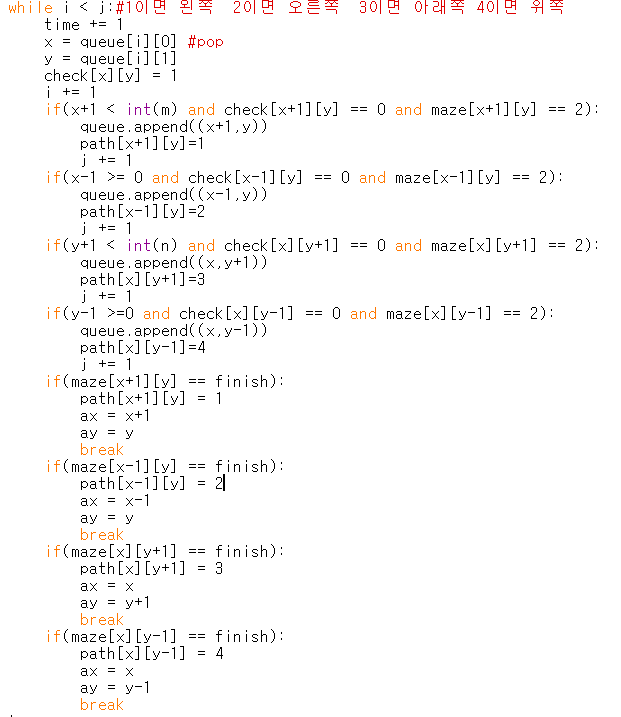


이후 각각의 알고리즘에 맞춰 도착지점 정보가 필요하다면 도착지점을 찾아 함수를 실행시킵니다. 과제의 경우 열쇠를 먼저 찾고 그 후 도착지점으로 가야 하므로 각각의 알고리즘에 대해 첫 시작지점에서 열쇠까지 한 번, 열쇠부터 도착지점까지 총 두 번 함수를 호출 합니다. 이때 각각의 함수 별 return 값을 도착지점의 좌표로 설정하여 반환 받습니다. 그래서 열쇠까지 찾고 열쇠의 위치를 반환 받은 후 그 좌표를 시작지점으로 설정하여 도착지점까지 다시 돌려 줍니다.

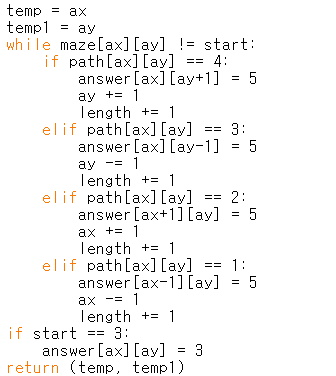
1. bfs 함수



Bfs 함수는 bfs 알고리즘을 시행하는 함수 입니다. 먼저 방문했는 지 여부를 파악하는 check 배열을 초기화 해주고, 경로 역 추적을 위해 현재 위치가 어디로부터 왔는지를 알 수 있도록 path 배열 역시 초기화 합니다. 이때 bfs 알고리즘을 queue를 통해 구현하였기 때문에 queue를 선언해주고 넘겨 받은 시작위치를 queue에 넣어 줍니다. 이때 j가 queue의 end, i가 queue의 front 입니다.

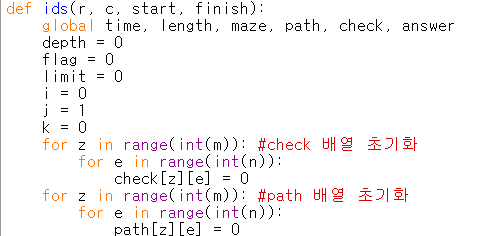


넘겨 받은 도착지점이 나올 때까지 미로를 탐색해 나갑니다. 이때 도착지점이 나오게 된다면 break를 통해 while문 밖을 빠져나가고 도착 지점의 좌표를 ax, ay에 저장하여 줍니다. 이후, 역 추적을 통해 최단 경로를 5로 바꾸어 주는 작업을 진행합니다.

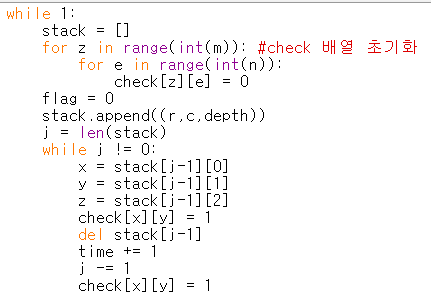


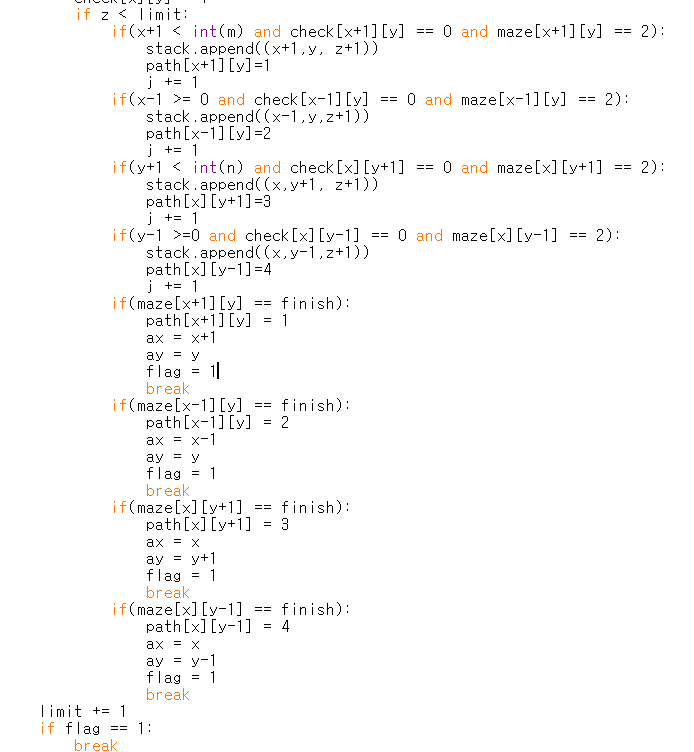
마지막 도착지점 좌표를 temp와 temp1에 저장하고 return 값으로 설정하여 줍니다. 그리고 path 배열에 저장해온 정보를 토대로 경로 역 추적을 시행합니다.

1. ids 함수

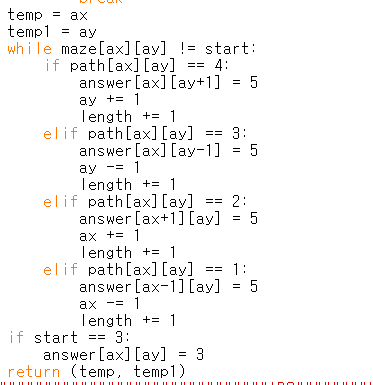


Ids 함수는 ids 알고리즘을 실행시켜주는 함수입니다. Bfs 함수와 마찬가지로 check 배열과 path 배열을 초기화 합니다.



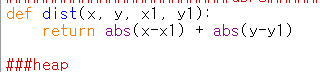


Ids 알고리즘은 stack을 리스트로 구현하여 코딩하였습니다. 먼저 전체적으로 limit라는 값을 설정하여 무한 루프를 돌려 주며 한 사이클이 끝나면 limit를 증가시켜 줍니다. 이때 stack에 들어갈 인자에 depth를 추가 시켜주어, stack에서 pop했더라도 limit보다 큰 depth를 가진 노드가 pop되었으면 그 아래 경로를 탐색하지 않도록 구현하였습니다. 따라서 limit 보다 작은 depth를 가진 것들만 조건을 따져 그 다음 경로에 finish가 있는지를 파악합니다.



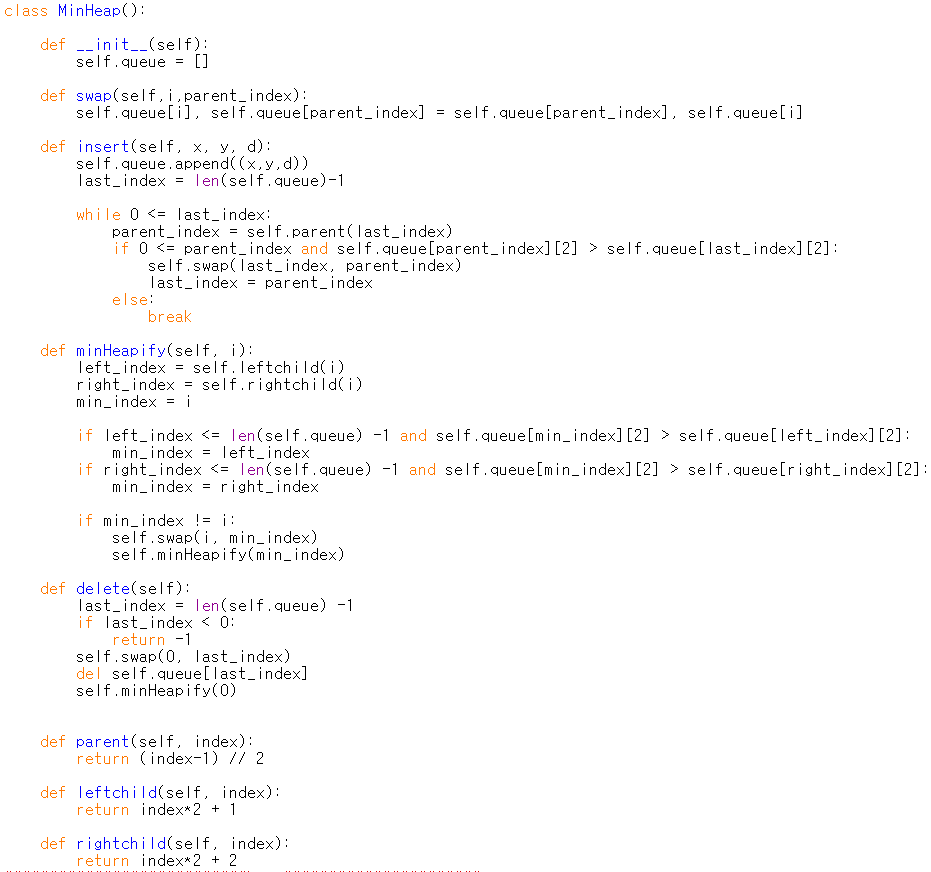
이후 과정은 마찬가지로 경로 역추적을 통해 최단 경로를 5로 바꾸어주는 작업을 진행합니다.

1. dist 함수



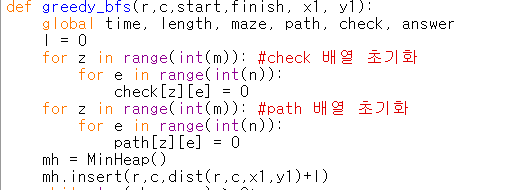
Dist 함수는 맨해튼 거리 공식을 사용하여 좌표간의 거리를 반환해주는 함수 입니다.

1. Heap

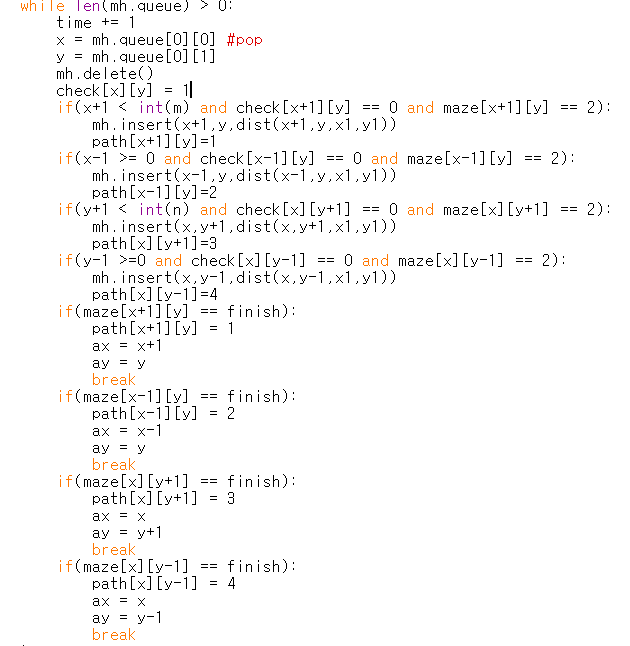


이제부터 설명할 greedy\_bfs 알고리즘과 a\*알고리즘은 최솟값을 뽑아내야 하므로 내장 되어 있는 priority\_queue가 아닌 min heap을 구현하여 사용하였습니다.

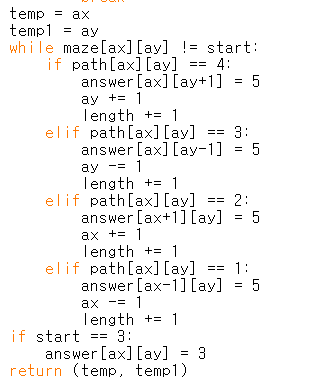
1. greedy\_bfs 함수



Greedy\_bfs 함수는 greedy bfs 알고리즘을 실행하는 함수입니다. 이때 도착지점까지의 거리가 있어야 하므로 이전 알고리즘들과 달리 도착지점의 좌표를 미리 구해 넘겨받습니다. 그리고 구현한 MinHeap의 객체를 만들어 주고, 이번엔 힙에 시작지점의 좌표와 도착지점까지의 거리를 구현해 놓은 dist함수를 통해 넘겨 받은 값을 같이 추가합니다.

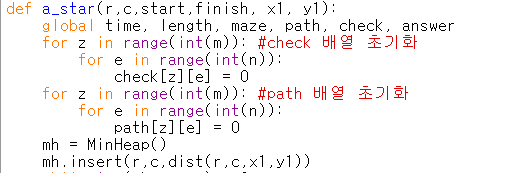


while문을 mh의 queue가 빌 때까지 돌려주면서 minheap의 꼭대기에 있는 노드를 pop하여 좌표를 저장하여 그 노드 주변 경로를 탐색합니다. 노드 주변 중 조건에 맞는 경로만 탐색하여 각각에 대한 path배열을 저장하고 check배열을 1로 만들어줍니다. 그리고 조건이 맞는 경로를 queue에 그 좌표와 그 지점부터 도착지점까지의 거리를 dist함수를 통해 반환받은 값을 함께 저장합니다.

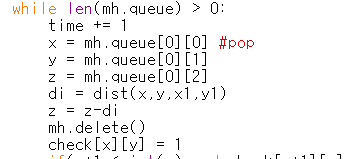


이 후 나온 finish 좌표를 temp과 temp1에 저장하여 return 합니다. 그 아래 반복문은 경로 역 추적을 통해 5로 변환하여 주는 작업입니다.

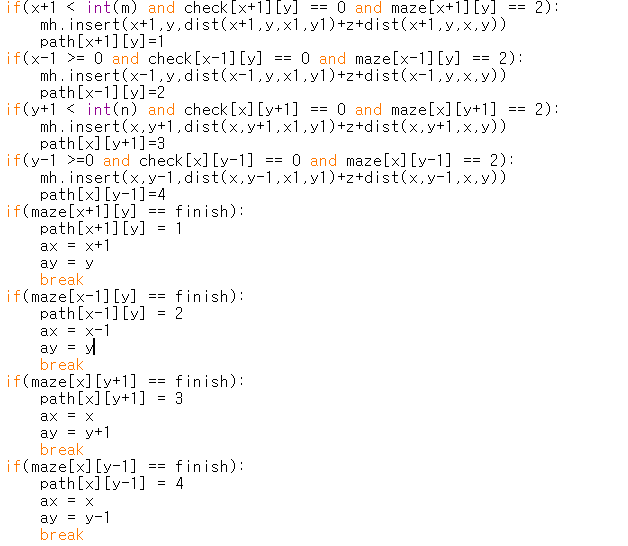
1. a\_star 함수



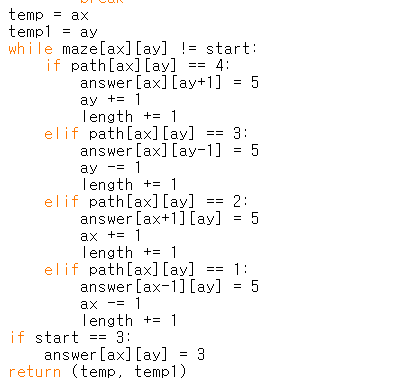
A\_star 알고리즘은 구조적으로 greedy\_bfs와 굉장히 유사합니다. 하지만 차이가 있다면 greedy\_bfs는 현재지점부터 도착지점까지의 거리만을 넣어줬다면, a\_star알고리즘은 현재까지 온 거리+현재 지점부터 도착지점까지 온 거리를 넣어줍니다.



이때 현재 까지 온 거리를 추가해주기 위해 현재 좌표의 거리를 z값에 저장합니다. 그럼 지금까지 온 거리+현재 좌표부터 도착지점 까지 거리가 나오게 됩니다. 그리고 di에 현재 지점부터 도착지점까지 거리를 구해 z값에서 빼줍니다. 그렇게 되면 지금까지 온 거리만 z값에 저장되게 됩니다. 그래서 이후 조건에 맞는 경로가 발견되어 넣어줄 때, z값+그 다음 경로 좌표부터 도착지점까지의 거리+현재 좌표와 다음 경로 좌표의 거리 차이를 넣어주게 된다면 a\_star알고리즘에 맞는 거리 값을 넣어줄 수 있습니다.

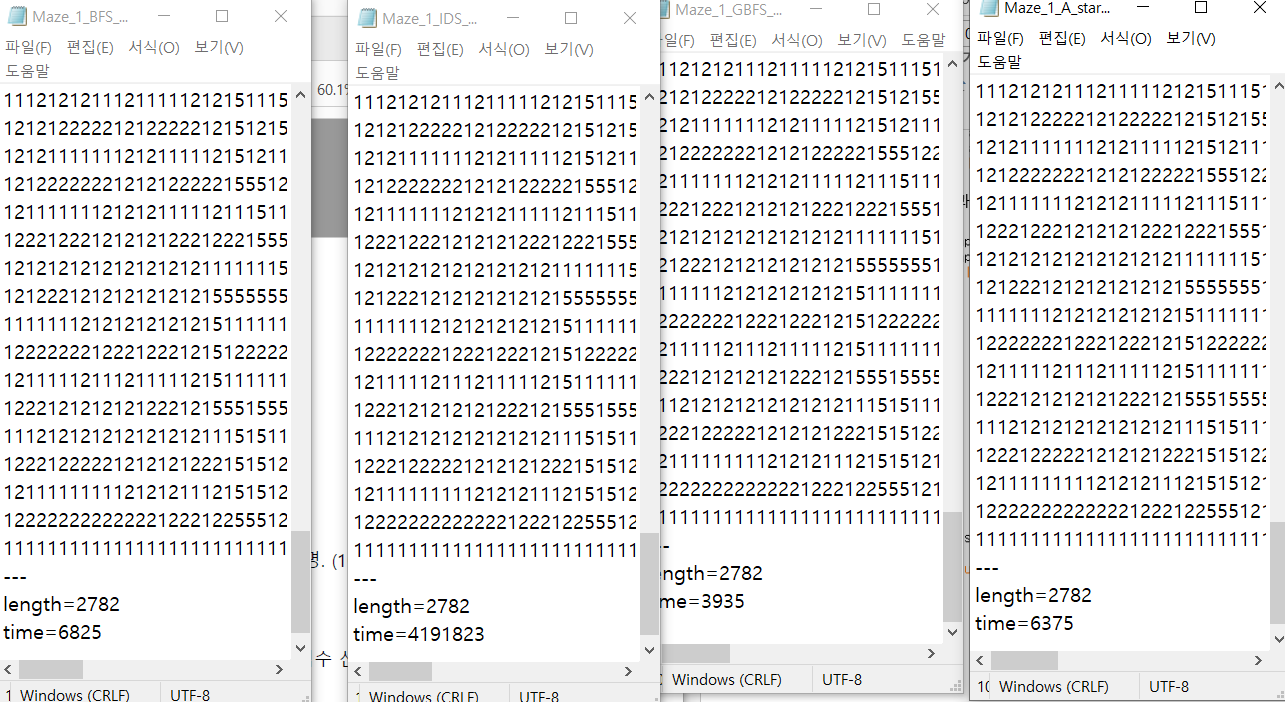


이 과정을 거쳐 도착지점을 찾게 된다면 경로 역추적을 진행합니다.

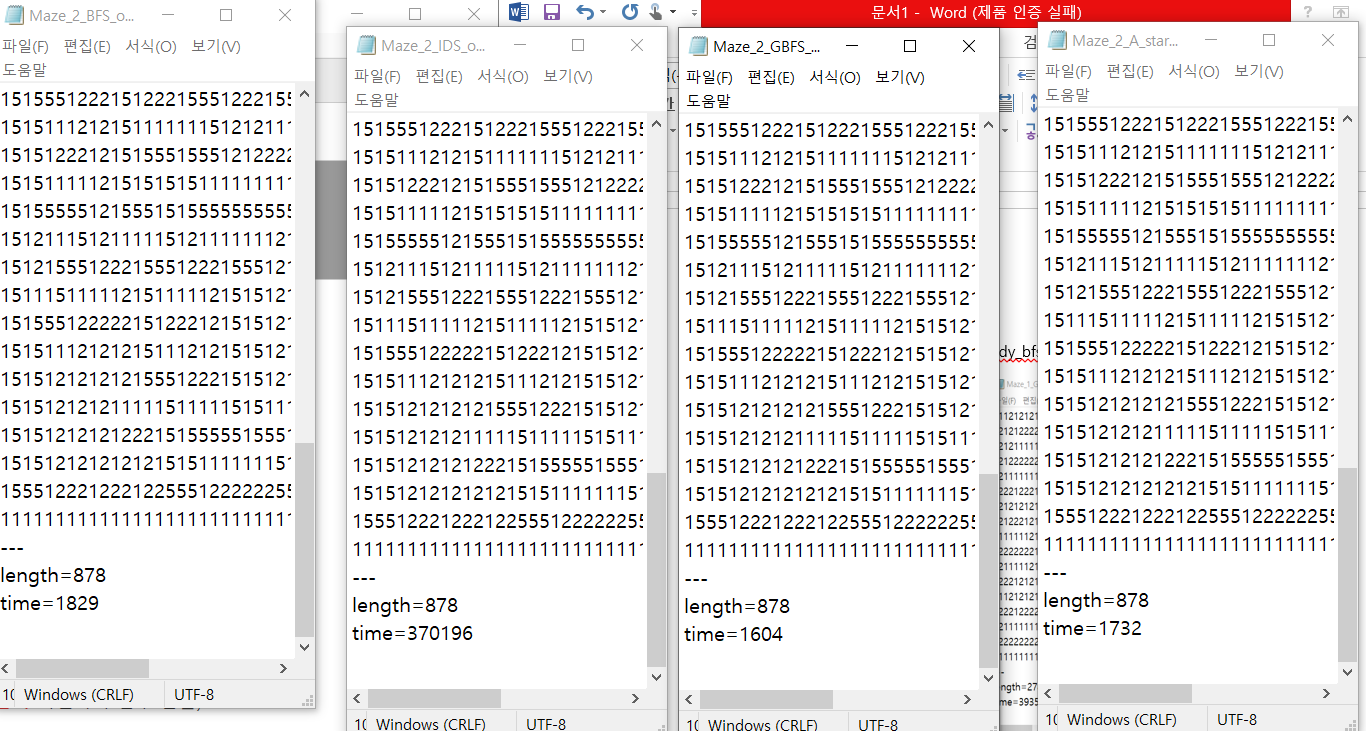


* 실험결과

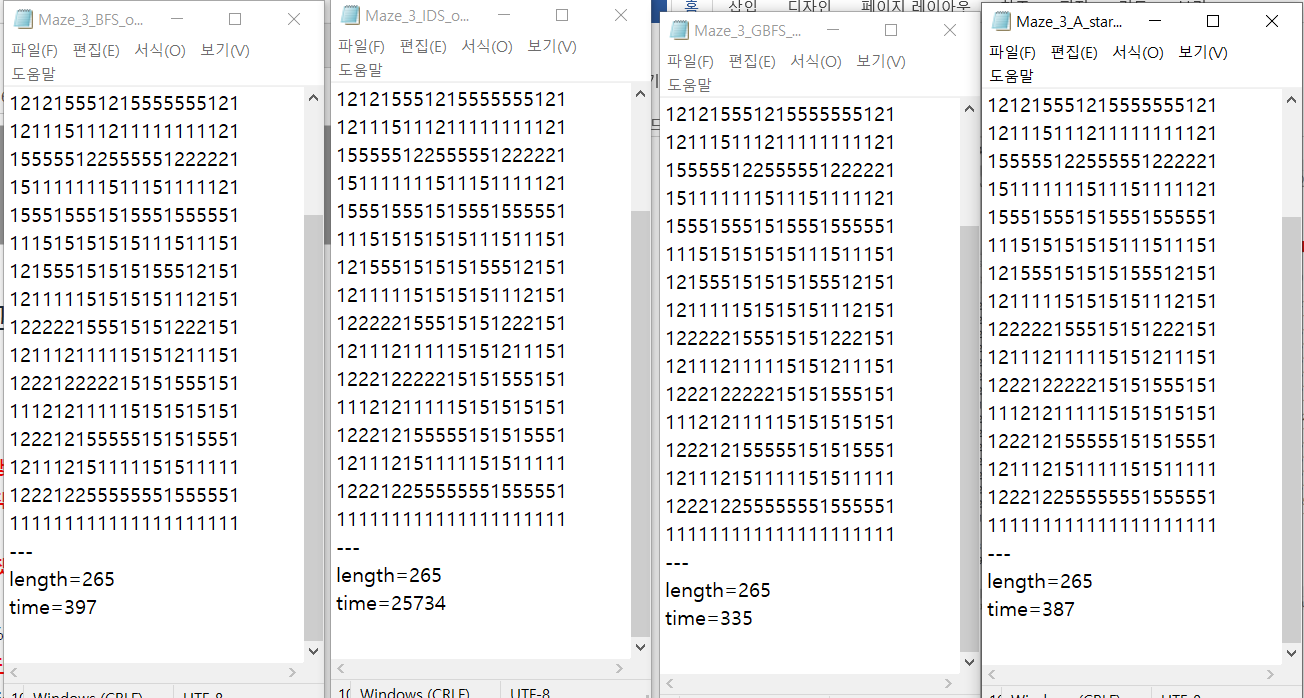
Maze\_1에 대한 결과(순서대로, bfs, ids, greedy\_bfs, a\_star)



Maze\_2에 대한 결과(순서대로, bfs, ids, greedy\_bfs, a\_star)



Maze\_3에 대한 결과(순서대로, bfs, ids, greedy\_bfs, a\_star)



Maze\_4에 대한 결과(순서대로, bfs, ids, greedy\_bfs, a\_star)

