13주차 결과보고서

전공: 경영학과 학년: 3학년 학번: 20190963 이름: 한다현

**1.**

**DFS**

13주차 실습에서는 DFS 알고리즘을 사용하였다. DFS 알고리즘은 stack을 사용하였다. 12주차에서 노드 사이의 연결 관계를 저장하기 위해 graph라는 vector 배열을 사용해 방의 번호를 저장하였고, DFS 알고리즘을 실행하면서 거쳐간 모든 경로를 저장하기 위해 path라는 int형 배열과 모든 경로 중 최적 경로를 저장하기 위해 best라는 int형 배열, 그리고 방의 방문 여부를 저장하는 vistited라는 int형 배열을 사용하였다. DFS() 함수에서는 먼저 stack<int> stack이라는 코드로 이번 실습에서 사용할 stack을 정의하였고, maze\_row\*maze\_col 크기의 visited 배열의 모든 원소 값을 0으로 초기화하였으며, maze\_row\*maze\_col\*5 크기의 path 배열과 maze\_row\*maze\_col 크기의 best 배열의 모든 원소 값을 -1로 초기화하였다. 이 때 path와 best의 원소 값을 0으로 초기화하지 않은 이유는 초기값과 0번 방과의 혼동을 방지하기 위해서이다. 목적지인 target 변수는 maze\_row\*maze\_col -1 이고, 시작 노드 v는 0이다. 자료 구조와 변수들을 모두 초기화 한 후에는 먼저 stack에 v를 추가하고, visited[v]를 1로, path[num++]은 v로 변경한다. 이 때, num과 best\_num은 각각 path의 순서, best의 순서를 의미한다. 즉 현재 시점에서 path[num++]은 path[0]을 의미한다. 첫 번째 노드를 저장한 후에는 while문이 시작된다. while문은 stack이 비어있지 않은 경우에 계속 반복된다. while문 내에서는 먼저 stack의 top이 target, 즉 목적 노드와 동일한지 검사하고, 동일하다면 실행을 중단한다. stack의 top이 target이 아니라면, graph[top]의 사이즈가 0인지 검사한다. 즉, 인접한 노드가 있는지 검사하는 것이다. 인접한 노드가 더 있다면 그 인접한 노드가 방문된 적이 있는지 검사한다. 인접한 노드를 모두 검사하기 위해서는 graph[top]의 사이즈만큼 for loop이 반복되어야 한다. 인접 노드 u가 방문된 적이 없다면 u를 stack에 추가하고, visited[u] = 1, path[num++] = u로 저장한다. 모든 인접 노드를 검사한 후에 인접한 노드들이 모두 이미 방문된 적이 있는 노드들이라면 stack의 top을 pop하고, 그 상태에서의 top을 path[num++]에 저장한다. 만약 graph[top]의 사이즈가 0이면 인접한 노드가 없다는 의미이므로 stack의 top을 제거한다. 이런식으로 stack의 top이 target일 때까지 반복한 뒤에 whlie문이 종료되면 stack에는 아직 데이터가 남아있다. 첫 번쨰 while문이 종료되면 두 번째 while문을 실행하는데, 이 반복문은 stack이 비어 있지 않은 경우에 계속 반복하면서 stack의 top을 반환하여 best[best\_num++]에 저장하고, stack의 top을 제거한다. 이 while문까지 종료되고 나면 isdfs의 true로 설정하고 함수를 종료한다. DFS() 함수에서는 stack이 비어 있지 않은 동안에 while loop을 통해 반복하였고, 그 while loop 안에서 다시 graph[top]의 사이즈만큼 for loop을 통해 반복하였기 때문에 DFS 함수의 시간 복잡도는 O(n^2)이고, vector 배열을 사용하였기 때문에 공간 복잡도는 O(n^2)이다.

dfsdraw

13주차 실습에서 작성한 dfsdraw 함수는 DFS 함수에서 구한 path와 best를 사용해 경로에 해당하는 선을 그리는 함수이다. 먼저 i가 0부터 num-2까지 rooms[path[i]]와 rooms[path[i+1]] 사이의 선을 그린다. 이 때 rooms는 방의 번호에 해당하는 x, y 좌표를 저장한 Dots 구조체 배열이다. path에 대한 선을 그린 후에 선의 색깔과 굵기를 바꾸고, j가 0부터 best\_num-2까지 rooms[best[j]]와 rooms[best[j+1]] 사이의 선을 그린다. path를 그리는 것이 전체 경로를 그리는 과정이고, best를 그리는 것이 최적 경로를 그리는 과정이다. 이 함수에서는 for loop이 사용되므로 시간 복잡도는 O(n)이고, 1차원 배열을 사용하므로 공간 복잡도는 O(n)이다.

BFS

13주차 과제에서 작성한 BFS 알고리즘은 queue를 사용한다. 이 함수에서는 3가지 자료구조를 사용한다. 탐색한 경로의 부모 노드를 저장하는 vector 배열인 parent, 노드의 방문여부를 저장하는 int형 배열인 visited, 그리고 경로를 저장하는 int형 배열인 q\_path이다. 먼저 시작 노드인 v를 queue에 추가하고 visited[v]=1, q\_path[num2++]=v로 저장한다. 그리고 while loop를 실행하는데, 이 while문은 queue가 비어 있지 않은 동안 반복한다. while문 내에서는 먼저 queue의 back이 target, 즉 목적지와 동일한지 검사하고, 동일하다면 while문을 종료한다. 동일하지 않다면 queue의 front 값을 w에 저장하고 front를 제거한다. 이렇게 구한 w를 사용하여 k=0부터 graph[w]의 사이즈까지 for loop으로 반복한다. graph[w][k], 즉 w와 연결된 노드를 u에 저장하고 u를 방문한 적이 없다면 visited[u]=1, q\_path[num2++] = u를 저장하고, parent[u]에 w를 추가하고 u를 queue에 추가한다. while loop이 종료되면 isbfs를 true로 저장하고 함수를 종료한다. BFS() 함수에서는 queue가 비어 있지 않은 동안에 while loop을 통해 반복하였고, 그 while loop 안에서 다시 graph[w]의 사이즈만큼 for loop을 통해 반복하였기 때문에 BFS 함수의 시간 복잡도는 O(n^2)이고, vector 배열을 사용하였기 때문에 공간 복잡도는 O(n^2)이다.

bfsdraw

13주차 실습에서 작성한 bfsdraw 함수는 BFS 함수에서 구한 q\_path를 사용해 경로에 해당하는 선을 그리는 함수이다. 먼저 j가 num2-2부터 2까지 j를 1씩 감소시키며 반복하는 for loop을 실행한다. for loop 내에서는 node라는 정수형 변수에 parent[q\_path[j]].back()을 저장한다. 이는 q\_path[j]의 parent를 node에 저장하는 것이다. 그 후 rooms[q\_path[j]]와 rooms[node] 사이의 선을 그리고 for loop 안에서 다시 while loop을 실행하는데, 이 while 문은 node가 0이 아닌 동안 반복한다. while 문 내에서는 parent[node].back()을 node2에 저장하는데, 이는 node의 부모 노드를 저장하는 것이다. node와 node2를 구하면 rooms[node]와 rooms[node2] 사이의 선을 그리고 node에 node2를 저장한다. 이렇게 하면 방문한 모든 경로를 그릴 수 있다. 모든 경로를 그리는 작업이 종료되면 선의 색깔과 굵기를 바꾸고 parent[q\_path[num2-1]].back()을 node에 저장한다. q\_path[num2-1]은 목적 노드를 의미하고, 최적 경로를 구하기 위해서는 목적 노드의 부모 노드를 따라가는 방식으로 구할 수 있다. 그래서 먼저 rooms[q\_path[num2-1]]과 rooms[node] 사이의 선을 그리고 while loop를 실행하여 node가 0이 아닌 동안 반복한다. while 문 내에서는 node2에 node의 parent를 저장하고 rooms[node]와 rooms[node2] 사이의 선을 그리고 node에 node2를 저장한다. 이렇게 하면 bfs의 최적 경로를 그릴 수 있다. 이 함수에서는 for loop과 while loop으로 이루어진 이중 반복문이 사용되므로 시간 복잡도는 O(n^2)이고, vector 배열인 parent을 사용하므로 공간 복잡도는 O(n^2)이다.

다른 함수는 모두 실험 전과 동일하게 작성되었지만, BFS 함수에서는 DFS와 마찬가지로 path와 best 두 개의 배열이 사용될 것이라고 생각했지만, 모든 경로를 q\_path라는 배열에 저장하고, 최적 경로는 best라는 새로운 배열에 저장하지 않고, q\_path의 마지막 노드의 parent를 따라가면서 선을 출력하는 방식으로 수정되었다.

**2.**

DFS는 stack을 사용하였고, BFS는 queue를 사용하였다. DFS에서는 모든 경로를 저장하는 배열과 최적 경로를 저장하는 배열을 따로 선언하여 사용하지만 BFS에서는 경로를 하나의 배열에 모두 저장하고, 각각의 노드의 부모 노드를 저장하는 vector 배열을 선언하여 q\_path 배열의 마지막 원소의 parent부터 시작하여 시작점까지 선을 그리며 최적 경로를 출력하였다. BFS는 vector 배열을 사용하였지만 DFS는 1차원 배열을 사용하였기 때문에 공간 복잡도 차원에서는 DFS가 장점을 가지고 있다. 반면 BFS는 DFS에 비해 인접한 노드를 방문하는 알고리즘이 더 간단하다는 장점이 있다. 하지만 DFS 알고리즘을 통해 얻은 path와 best 배열을 출력하는 알고리즘이 BFS 알고리즘을 통해 얻은 q\_path를 출력하는 알고리즘에 비해 시간 복잡도와 공간 복잡도가 낮기 때문에 이 프로그램의 자료구조에는 DFS가 더 적합하다고 생각한다. 다음은 DFS와 BFS의 출력이다.

텍스트, 소프트웨어, 스크린샷, 컴퓨터이(가) 표시된 사진

자동 생성된 설명 dfs

텍스트, 스크린샷, 소프트웨어, 컴퓨터이(가) 표시된 사진

자동 생성된 설명bfs