

2018 학년도 1 학기

자료구조

- 미로찾기 -



과목 | 자료구조

담당교수 | 최윤정 교수님

팀 | Team9

이름 | 박상희, 김제헌, 정주원, 임수연

제출일 | 2018.05.27

Index

0. 미로찾기란.....	3
1. 미로찾기 분석.....	3
1.1. 미로의 성질.....	3
1.2. 미로의 유형.....	4
1.3. 단순히 미로를 푸는 방법.....	5
2. 탐색 알고리즘 조사 및 분석.....	6
2.1. 탐색 알고리즘이란.....	6
2.2. Uninformed Search.....	6
2.3. Informed Search.....	10
3. 탐색 알고리즘 구현.....	14
3.1. BFS(너비우선탐색) - 정주원.....	14
3.2. DFS(깊이우선탐색) - 김제현.....	19
3.3. 다익스트라 및 A* - 김제현.....	24
3.4. A* - 박상희.....	29
4. 번외.....	39
4.1. AVL 트리 - 임수연.....	39
5. 일정관리.....	43
6. 역할분담.....	43

0. 미로찾기란

미로찾기를 말하기 앞서 미로란 복잡한 길을 찾아 출발점부터 시작해 도착점까지 도달하는 퍼즐이다. 따라서 미로찾기란 출발점에서 도착점으로 가는 경로를 찾는 것이다. 이는 굉장히 유명한 퍼즐로, 일반적으로 위에서 미로를 보는 구조와 정면에서 미로를 보는 구조로 나뉜다. 또한, 종이와 컴퓨터 등 미로찾기를 다양한 방식으로 할 수 있다.

1. 미로찾기 분석

1.1. 미로의 성질

이동	실제로 이동	가상으로 이동
정보	Uninformed	Informed
순환구조	있음	없음
도착지	있음	없음
크기	유한	무한
가중치	있음 [불균일]	없음 [균일]

기본적으로 미로의 성질을 위의 6가지[이동, 정보, 순환구조, 도착지, 크기, 가중치]로 나눠보았다.

- 이동 상태 : 만약 실제로 이동한다면, 알고리즘을 적용하는데 있어서 점프를 할 수가 없고 무조건 직접 가야 한다. 이 경우, 좌선법/우선법, DFS 알고리즘 등을 이용할 수밖에 없다. 만약 가상으로 이동한다면, 추후 다른 미로의 성질[정보, 순환구조, 도착지, 크기, 가중치]에 따라서 다양한 미로 알고리즘[좌선법/우선법, DFS에 추가로 BFS, 다익스트라 알고리즘, A* 알고리즘]도 적용 가능하다.
- 정보의 제공 : 만약 정보가 주어지지 않는다면, 이 역시 무조건 가보면서 탐색해보는 수 밖에 없다. 이 경우, Uninformed Search 알고리즘으로 BFS, Uniform-cost Search, DFS, Depth-limited Search, Iterative Deepening Search, Bidirectional search의 방법을 적용할 수 있다. 만약 정보가 주어졌다면 바로 확인가능 하며, 목적지 방향으로 가중치를 주면서 진행할 수도 있다. 이 경우, Informed Search 알고리즘으로 Best-first Search, A* 알고리즘 등의 방법을 적용할 수 있다.
- 순환구조 유무 : 만약 미로 내부에 내벽이 존재하는 경우 순환구조가 존재하며 이 경우, 좌선법/우선법과 같은 방법으로는 풀 수가 없다. 그 외에 DFS같은 방식을 적용하더라도 자신이 갔던 길임을 분명히 표시해야한다. 만약 미로 내부에 순환구조가 존재하지 않은 1자형 통로로만 구성되어 있다면, 이 경우 단순히 좌선법/우선법과 같은 방법을 통해서도 풀 수가 있다.
- 도착지 유무 : 일반적으로 미로의 성질은 도착지가 있는 것으로 가정하지만, 만약 도착지가 없을 경우 무한루프에 빠지는 것이 아니라 이 미로에는 도착지로 가는 경로가 없다는 것을 올바르게 출력할 수 있어야 한다.
- 맵의 크기 : 보통 미로의 크기는 유한하다고 가정한다. 하지만, 현실세계처럼 그 크기가 엄청나게 큰 경우 등도 거의 무한이라고 가정할 수 있다.
- 가중치 유무 : 일반적으로 DFS, BFS 등의 알고리즘을 적용할 때에는 미로에 가중치가 없는 평면상태라고 가정한다. 하지만, 미로가 울퉁불퉁하는 등의 가중치가 존재할 경우, 다익스트라, A*알고리즘 등의 알고리즘을 적용해서 최소 비용의 경로를 탐색해야 한다.

1.2. 미로의 유형

앞서 소개한 6가지의 성질을 통해 미로의 유형을 나눠보았다. 단, 실험의 한계 때문에 맵의 크기는 제한했다.

미로 이미지	미로에 대한 설명
	Case1) 벽에 구멍이 뚫려 있지 않은 경우. 미로가 오직 1직선 상으로만 이루어져 있어 가장 쉬운 유형이다
	Case2) 벽에 구멍이 뚫려 있는 경우. 미로 내에 벽에 구멍이 뚫려 있어 경로를 찾는 도중 빙글빙글 돌 수도 있다.
	Case3) 벽에 구멍이 뚫려 있고, 목적지가 벽으로 둘러 쌓인 경우 Case2에 추가로 목적지가 벽으로 둘러 쌓여 목적지에 도착할 수 없다.

	<p>Case4) 목적지가 최외곽 벽이 아닌 중앙에 있을 경우 단순히 벽만을 따라서 간다면 중앙에 있는 목적지에 도착할 수 없다.</p>
	<p>Case5) 목적지가 중앙에 있고, 목적지가 벽으로 둘러 쌓였을 경우 목적지에 도착할 수가 없다.</p>
	<p>Case6) 목적지가 최외곽 벽이 아닌 중앙에 있으며, 맵이 균일하지 않고 가중치가 있을 경우, 최소 비용의 경로를 찾아서 목적지에 도달해야 한다.</p>

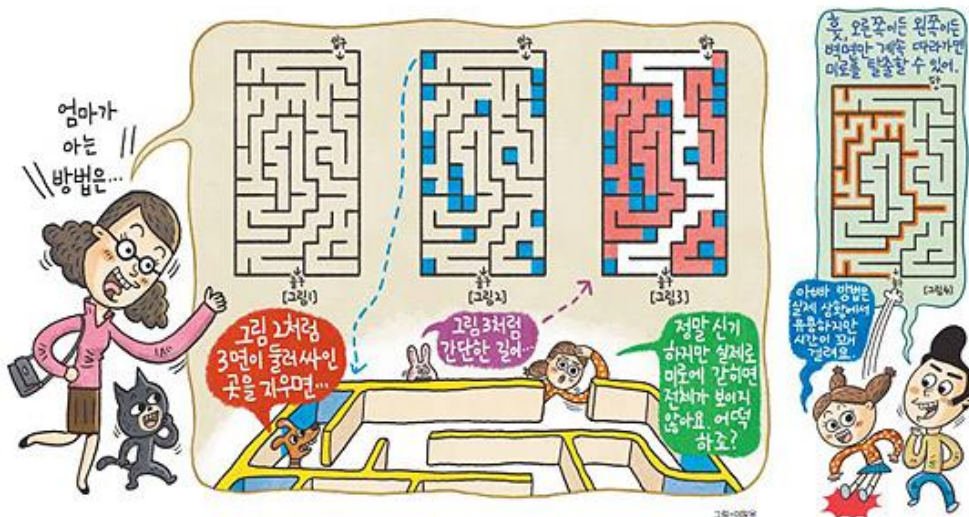
1.3. 단순하게 미로를 푸는 방법

1) 벽을 따라서 탐색하는 방법

가장 고전적인 방법으로써, 좌측 또는 우측에 있는 벽면에 손을 짚고 목적지에 도착할 때까지 계속 걷는 방법이다. 만일, 걷는 도중 막다른 길에 도달하면 막다른 길에 도착하기 전에 갈림길에서 가지 않았던 길을 간다. 이 방법은 미로에 대한 정보가 없을 경우 사용하는 방법으로 정보가 없어 효율성이 떨어진다. 또한, 미로의 목적지가 벽으로 연결되지 않고 중앙에 있을 경우 도착할 수 없다. 또한 목적지가 막혀 있을 경우, 자신의 위치로 다시 돌아오게 된다. 이때, 목적지가 없다고 판단한다. 또한, 도착한 경로가 최단 경로임을 보장하지 않는다.

2) 막다른 공간을 칠하는 방법

이 방법은 미로에 대한 정보가 있다는 가정하에 출발하는 방법으로, 미로의 막다른 공간을 벽으로 칠하는 방법으로 계속 칠하다 보면 막다른 공간이 다 사라지고 목적지로 가는 경로 하나만 남는다. 다만 이 경우는 case1일 때 적합하고, case 2부터는 적합하지 않다. 왜냐하면 막다른 공간을 칠한다음 이 역시 벽면에 손을 짚고 걷기 때문이다.



2. 탐색 알고리즘 조사 및 분석

2.1. 탐색 알고리즘이란

어떤 순서리스트 또는 비순서 리스트에서 목적지의 존재 및 그 위치를 찾는 방법을 말한다. 미로찾기 역시 미로에서 목적지로 가는 방법을 찾는 것이기에 탐색 알고리즘을 적용할 수 있다. 일반적으로 탐색 알고리즘은 Uninformed Search와 Informed Search로 나뉜다.

2.2. Uninformed Search

도메인에 대한 정보 없이 탐색을 하는 것으로 다만 알고있는 것은 이 곳이 목적지인지 아닌지 만을 알 수 있다. 그래서 주먹구구식으로 하는 방법인 brute-force방법이다. 하지만 정보가 없기에 정보가 있는 경우보다 더 좋은 탐색을 할 수는 없다.

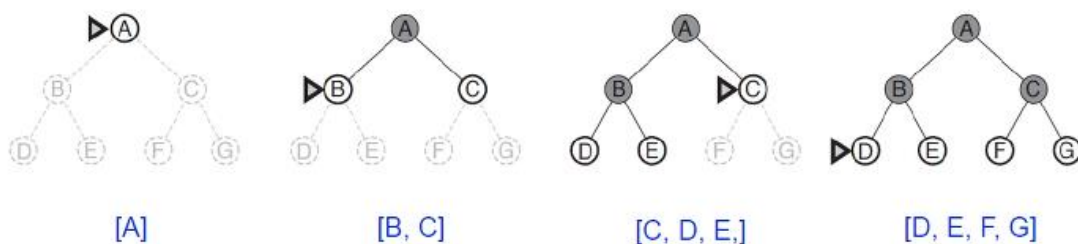
1) 알고리즘 평가요소

- Completeness: 답이 존재하면 항상 찾을 수 있는가?
- Optimality: 찾아낸 답이 최소 비용인가? 더 좋은 답은 없는가?
- Time Complexity: 시간 복잡도
- Space Complexity: 공간 복잡도

2) BFS(Breadth First Search, 너비 우선 탐색)

출발지 노드에서 인접한 모든 노드를 우선 방문하는 방법이다. 더 이상 방문하지 않은 노드가 없을 때까지 방문하지 않은 노드에 대해 너비 우선 검색을 한다. 간단히 말하자면 출발지 노드를 루트로 가정하고 목적지 노드를 찾기 위해 Depth 또는 Level단위로 노드들을 검사하면서 진행하는 방식이다. 저장된 순서대로 탐색해야 하므로 선입선출 방식인 Queue에 저장한다.

2.1) 과정



2.2) 알고리즘 평가

- Completeness: 노드가 많더라도 영점의 시간동안 탐색하면 목적지 노드를 찾을 수 있으므로 Complete하다.
- Optimality: 노드 간의 이동 비용이 동일하다는 가정하에는 Optimal하다. 다만, 비용이 다른 경우에는 보장할 수 없다.

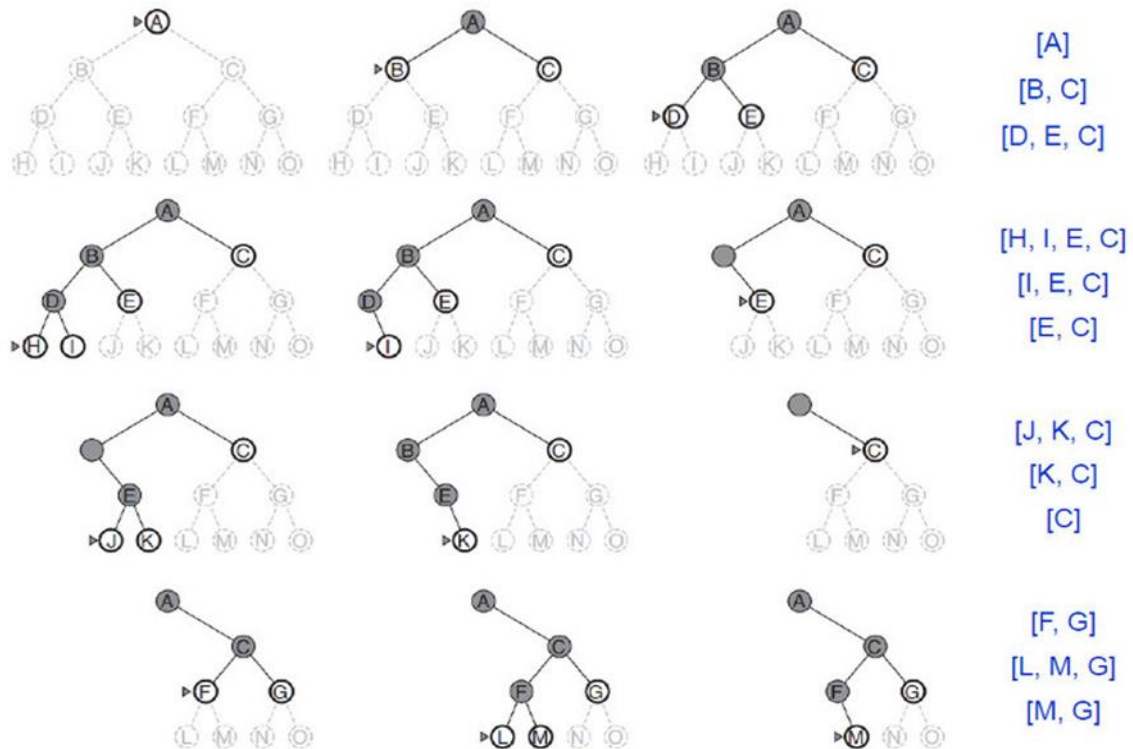
-Time & Space Complexity: b 를 평균 자식 노드의 개수, d 를 depth라 하면, 시공간 복잡도는 $O(b^{(d+1)})$ 이다.

-결론: 목적 노드에 도착할 때까지 노드들을 모두 저장하므로 저장공간을 굉장히 많이 차지하고 복잡도가 지수적으로 증가하여 사용하기 어렵다.

3) DFS(Depth First Search, 깊이 우선 탐색)

말 그대로 깊이를 우선적으로 하여 탐색하는 방법으로, 목적지 노드에 도착할 때까지 계속 전진하다가 자식 노드가 없으면 그 전의 갈림길에서 다른 노드를 선택하여 또 전진하는 방법이다. 1개의 노드에서 자식 노드를 저장하고 또 자식 노드에서 자식 노드를 저장하는 방식으로 후입선출의 구조인 Stack을 사용한다.

3.1) 과정



3.2) 알고리즘 평가

-Complete: 노드 간의 루프구조가 있다면 루프에 빠질 가능성이 있어 반드시 목적지 노드를 찾는다라는 보장을 할 수 없다.

-Optimal: 깊이를 우선적으로 탐색하기에 목적지 노드까지 가는 경로가 하나가 아니라면 보장할 수 없다.

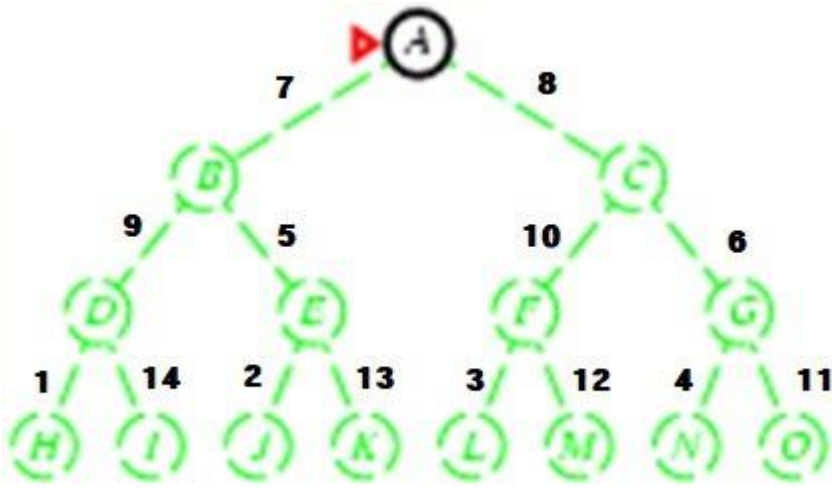
-Time & Space Complexity: 가장 깊은 부분까지 탐색을 하므로 시간 복잡도는 $O(b^m)$ 이고 공간 복잡도는 $O(bm)$ 이다. m 은 트리의 최대 깊이이다.

-결론: 현재 탐색하고 있는 경로 상의 노드들만 기억하면 되기 때문에 BFS와 비교시 공간 복잡도가 작고, depth가 클 경우 BFS보다 빨리 찾을 수 있다. 다만, 목적 노드가 없는 depth가 큰 경로에 빠지면 시간을 굉장히 낭비할 수 있다. 또한, 목적 노드까지 여러 경로가 있을 경우 먼저 발견된 경로가 최단 경로라는 보장을 할 수 없다.

4) Uniform-cost Search, Lowest-cost Search

노드 간의 경로 비용을 비교하여 최단 경로를 탐색하는 방식이다. 자식 노드로 가는데 드는 총 비용들을 검사해서 비교한 후 가장 작은 비용이 드는 노드를 저장소에 추가하면서 데이터를 확장한다. 노드를 저장하고, 비용을 기준으로 정렬을 하기에 Priority Queue가 적합하다.

4.1) 과정



1. [(A,0)]
 2. [(B,7), (C,8)]
 3. [(C,8), (E,12), (D,16)]
 4. [(E,12), (G,14), (D,16), (F,18)]
 -
- [A → B → C → E → G → J → D → H → F → N → L → K → O → I → M] 순서로 진행한다.

4.2) 알고리즘 평가

- Complete & Optimal: BFS의 특성을 따르기에 Complete와 Optimal을 보장한다.
- Time & Space Complexity: 시공간 복잡도는 $O(b^{\frac{C}{\epsilon}})$ C는 최단 경로에서의 총 비용이고, ϵ 은 탐색하는데 소요되는 비용의 최소값을 의미한다.
- 결론: 출발 노드에서 목적지 노드까지 최단 경로를 보장해주고, 무한 루프를 피할 수 있게 해주지만, 역시 깊이가 커질수록 엄청난 메모리 공간이 필요하다.

5) Depth-limited Search

DFS는 무한 루프에 빠질 수가 있다는 문제점이 있다. 이를 해결하기 위해 탐색 깊이에 제한을 뒤 문제를 해결한 방법이다. 현재 진행중인 노드의 depth가 limit 값인 l이면 자식 노드를 더 추가하지 않는 방식을 추가하면 된다. DFS의 방식을 따르기에 Stack 구조를 사용한다.

5.1) 과정



Depth Bound = 3

Depth limit값을 3으로 잡았을 때의 과정으로, level이 3을 초과하지 않고 그 안에서 DFS 알고리즘을 수행한다.

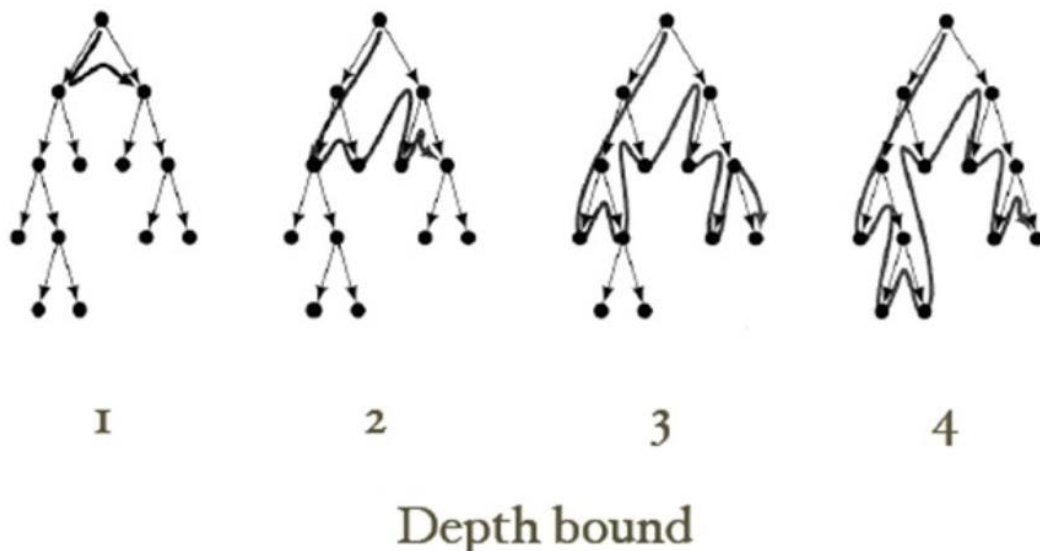
5.2) 알고리즘 평가

- Complete: 목적지 노드의 depth가 l보다 작을 경우는 Complete하나, 클 경우 탐색을 하지 않으므로 Complete하지 않다.
- Optimal: DFS의 속성을 띄기에 Optimal을 보장하지 않는다.
- Time & Space Complexity: 최대 깊이인 m에서 l로 변했으므로 시간 복잡도는 $O(b^l)$ 고, 공간 복잡도는 $O(b \cdot l)$ 다.

6) Iterative Deepening Search

DLS는 Optimal을 보장하지 못하는 문제점이 있다. 그래서 limit인 l를 1부터 증가시키면서 DLS를 반복해 문제를 개선한 방법이다. DFS의 방식을 따르므로 Stack 구조를 사용한다.

6.1) 과정



Depth limit값을 1, 2, 3, 4로 점점 늘려가면서, 그 안에서 DFS 알고리즘을 수행한다.

6.2) 알고리즘 평가

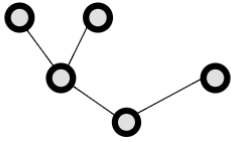
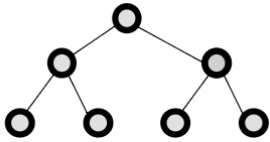
- Complete: DLS를 반복하면서 진행하므로 Complete를 만족한다.
- Optimal: l를 1씩 증가하고, 노드간 이동 비용이 같다면 Optimal을 보장한다.
- Time & Space Complexity: 시간 복잡도는 $O(b^d)$ 고, 공간 복잡도는 $O(bd)$ 다.

7) Bidirectional search

Bidirectional search는 출발지 및 목적지에서 각각 출발해서 진행해 나가면서, 서로 만날 때까지 진행하는 방식이다.

양 끝단에서는 기본적으로 BFS 계통의 방법을 사용하는 것 같다.

7.1) 과정



7.2) 알고리즘 평가

- Complete & Optimal: BFS의 특성을 따르기에 Complete와 Optimal을 보장한다.
- Time & Space Complexity: 시간 복잡도는 $O(b^{(d/2)})$ 고, 공간 복잡도는 $O(b^{(d/2)})$ 다.

8) 알고리즘간 비교

- ◇ b : branching factor
- ◇ d : depth of solution
- ◇ m : maximum depth of the search tree
- ◇ l : depth limit
- ◇ C^* : path cost of the optimal solution

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

2.3. Informed Search

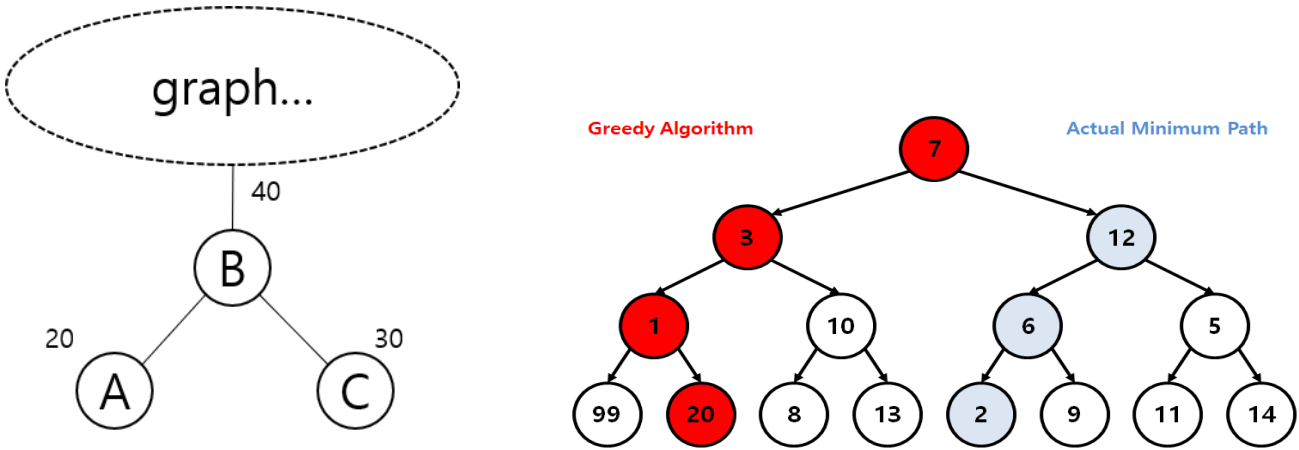
도메인에 대한 정보가 주어진 상태에서 목적지 노드를 찾는 것으로 모든 경우의 수를 고려할 필요없이 평가 함수에 따른 경로로 이동한다. 기본적으로 평가 함수 $f(n)$ 은 휴리스틱 함수 $h(n)$ 과 비용 함수 $g(n)$ 의 합으로 이루어진다. $f(n)=h(n)+g(n)$. 평가 함수는 일반적으로 빠른 결정이 나도록 설계하여 정확성은 주요 관심사가 아니다. 평가 함수의 목적은 확장 시킬 노드들에게 순위를 매김으로써 어떤 것이 목적지 노드까지 최상의 경로에 있는지를 결정함에 있다. 그래서 $f(n)$ 은 출발지 노드로부터 노드 n 을 통하여 목적지 노드까지 가는 최소 비용 경로에 대한 비용의 추정치다. $g(n)$ 은 출발지 노드로부터 노드 n 까지의 최단 경로의 실제 비용에 대한 추정치고, $h(n)$ 은 노드 n 부터 목적지 노드까지의 최적경로의 비용에 대한 추정치다. 휴리스틱 함수는 수학적, 논리적으로 구한 함수일 수도 있고, 경험과 실험을 통해 구한 함수일 수도 있다. 적절한 휴리스틱 함수를 통해 Uninformed Search보다 훨씬 빠르게 해결할 수 있다.

1) Best-first Search

목적지 노드로 가기 위해 현재 목표에 가장 가까워 보이는 노드로 이동하는 방법을 반복해 해결하는 방법이다. 현재 상태의 정보만을 토대로 최적의 경로를 선택하기 때문에 Greedy 알고리즘으로 분류된다.

1.1) 알고리즘 평가

-Complete and Optimal: B에서 출발하면 목적지 노드까지 A가 20으로 가장 가까워 A로 이동하면 다른 노드로 갈 수 없어 다시 B로 돌아온다. 또 B에서 탐색을 하면 A가 가장 가까워 A로 간다. 이렇게 무한반복에 빠져 Complete를 보장하지 못한다. Greedy 알고리즘이기에 지역적으로는 최적을 보장하지만 그 지역적인 최적들의 합이 최적이라는 보장은 못한다. 따라서 Optimal을 보장하지 못한다.

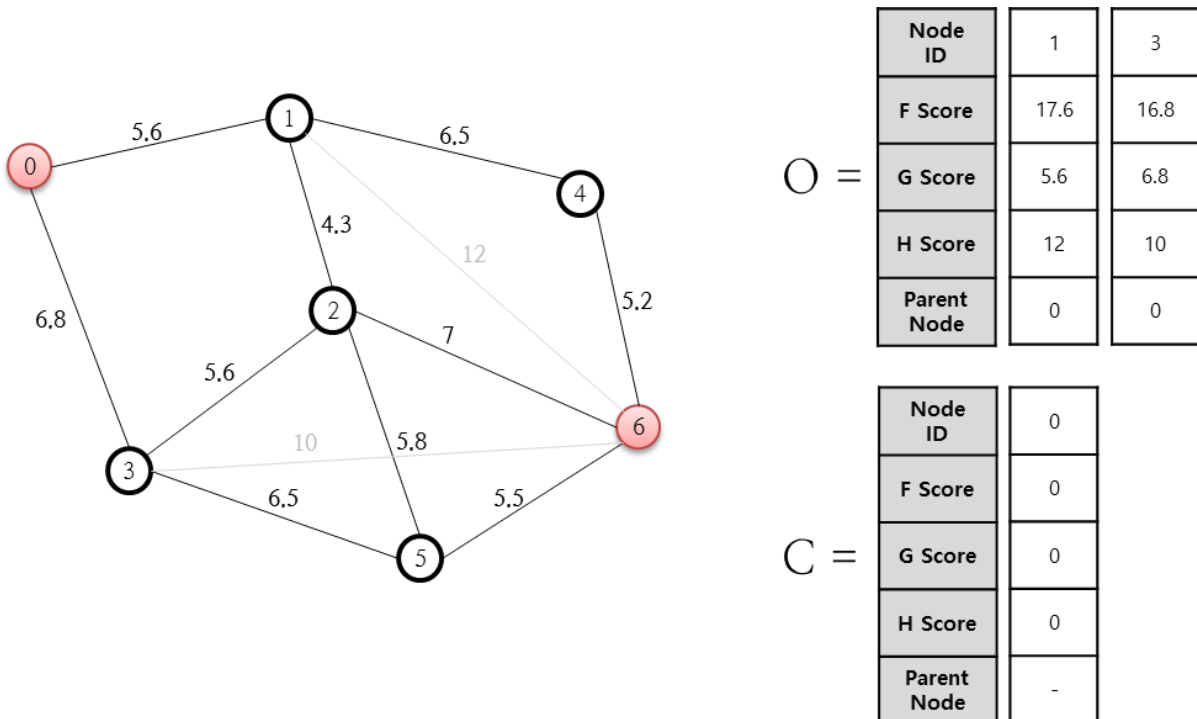


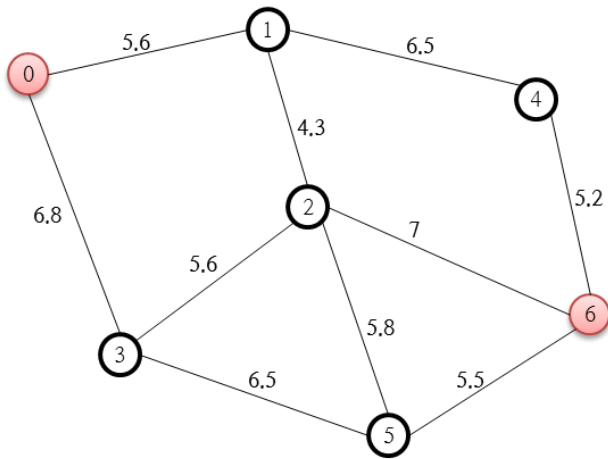
-결론: 휴리스틱 함수를 통해 최단 경로를 선택하는 것 자체는 좋으나, 목적지 노드를 찾는 과정에서 실제로 지난 경로에 대한 비용을 계산하지 않기 때문에 사용하기에 부적합하다.

2) A* 알고리즘

Best-first Search을 보완한 것으로, 다음 노드를 선택할 때 현재 노드까지 오는데 사용한 비용도 반영하는 알고리즘이다. $f(n)$ 은 출발지에서 노드 n 을 거쳐 목적지 노드에 도착할 때 사용될 비용으로 추정치다. $g(n)$ 은 출발지 노드에서 노드 n 까지 도착하는데 사용된 비용이다. $h(n)$ 은 노드 n 에서 목적지 노드까지 가는데 사용될 비용으로 추정치다. 그래서 평가함수 $f(n)=g(n)+h(n)$ 이다.

2.1) 과정





O =

Node ID	1	2	5
F Score	17.6	19.4	18.8
G Score	5.6	12.4	13.3
H Score	12	7	5.5
Parent Node	0	3	3

C =

Node ID	0	3
F Score	0	16.8
G Score	0	6.8
H Score	0	10
Parent Node	-	0

2
19.4
12.4
7
3

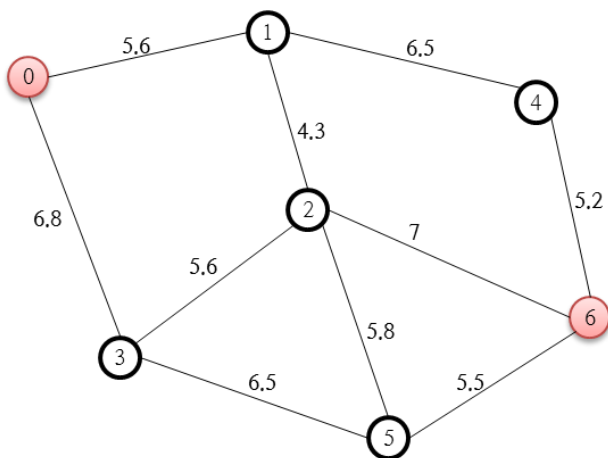


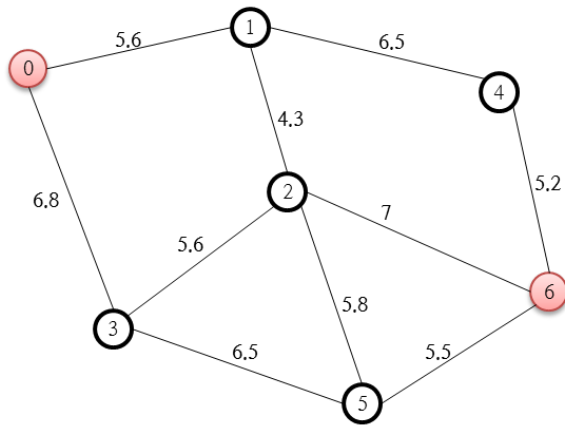
O =

Node ID	2	5	4
F Score	16.9	18.8	17.3
G Score	9.9	13.3	12.1
H Score	7	5.5	5.2
Parent Node	1	3	1

C =

Node ID	0	3	1
F Score	0	16.8	17.6
G Score	0	6.8	5.6
H Score	0	10	12
Parent Node	-	0	0



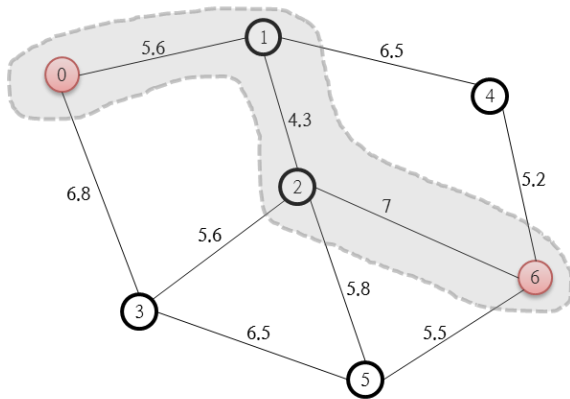


O =

Node ID	5	4	6
F Score	18.8	17.3	16.9
G Score	13.3	12.1	16.9
H Score	5.5	5.2	0
Parent Node	3	1	2

C =

Node ID	0	3	1	2
F Score	0	16.8	17.6	16.9
G Score	0	6.8	5.6	9.9
H Score	0	10	12	7
Parent Node	-	0	0	1



O =

Node ID	5	4
F Score	18.8	17.3
G Score	13.3	12.1
H Score	5.5	5.2
Parent Node	3	1

C =

Node ID	0	3	1	2	6
F Score	0	16.8	17.6	16.9	16.9
G Score	0	6.8	5.6	9.9	16.9
H Score	0	10	12	7	0
Parent Node	-	0	0	1	2

2.2) 알고리즘 평가

- Complete: Best-first Search에 현재 지나온 경로의 비용을 계산하므로 무한 루프에 빠지지 않아 complete를 보장한다.
- Optimal: 휴리스틱 함수가 실제 비용보다 작으면 Optimal을 보장하지만, 크면 보장하지 않는다.
- 결론: A* 알고리즘은 현존하는 최고의 검색 알고리즘이지만, 맵의 크기가 크면 열린 목록과 닫힌 목록에 많은 노드가 들어갈 수 있기 때문에 메모리와 수행속도에 문제가 있을 수 있다. A* 알고리즘의 가장 근본적인 약점은 열린 목록과 닫힌 목록을 관리하는 비용이다.

3. 탐색 알고리즘 구현

3.1. BFS(너비우선탐색) - 정주원

1) 개요

일반적인 경우 미로에 대한 지식 없이 탐색을 하기에 Uniformed Search 방법 중 하나인 BFS를 적용한다. BFS에 대한 설명은 위에서 했으니 생략한다. BFS는 특성상 자료구조 Queue를 사용하는 것이 적합하여, 큐를 구현하고 사용한다. 입구에서 가까운 쪽부터 여러 갈림길에서 각각의 마지막 탐색 좌표에서 한 칸씩 전진하는 방식으로 탐색한다. 미로 탐색시 이동간 가중치가 존재하지 않고 비용은 동일하므로 가장 먼저 출구에 도달한 경로가 최단경로이다.

2)구조체

BFS를 이용한 미로 탐색에는 좌표를 저장하는 Location과 Location들을 저장하는 queue가 사용된다. Location 구조체는 미로의 특정 위치의 좌표를 나타내며 main.c의 QueueType* path에 현위치에서 갈 수 있는 공간의 좌표가 Location 구조체로 들어간다.

구조체명		Location		Struct 기술서	작성일	2018/05/26	Page
Header					작성자	정주원	1/3
구조체 설명		특정 위치의 좌표를 저장한다.					
No	Variable name	Variable type	Variable Description				
1	C	int	좌표가 몇 번째 열에 존재하는지 즉, x좌표를 저장한다.				
2	R	int	좌표가 몇 번째 행에 존재하는지 즉, y좌표를 저장한다.				

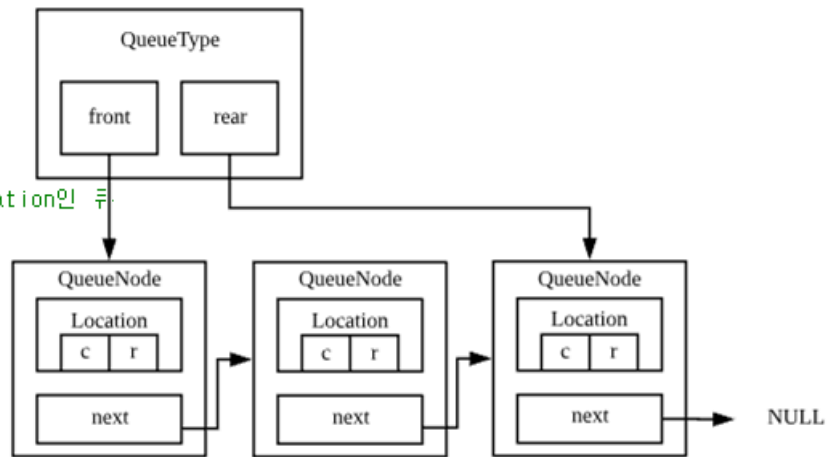
구조체명		QueueNode	Struct 기술서	작성일	2018/05/26	Page
Header				작성자	정주원	2/3
구조체 설명		큐의 노드에 대한 자료형으로 갈 수 있는 길의 좌표를 저장한다.				
No	Variable name	Variable type	Variable Description			
1	Loc	Location*	갈 수 있는 길의 좌표를 저장한다.			
2	Next	QueueNode*	다음 노드의 주소를 저장한다.			

구조체명		QueueType	Struct 기술서	작성일	2018/05/26	Page
Header				작성자	정주원	3/3
구조체 설명		큐에 대한 자료형으로 큐의 처음과 끝을 저장한다.				
No	Variable name	Variable type	Variable Description			
1	front	QueueNode*	큐의 처음 노드의 주소를 저장한다.			
2	rear	QueueNode*	큐의 마지막 노드의 주소를 저장한다.			

```
typedef struct location {
    int c; // 열 ...x
    int r; // 행 ...y
} Location;
```

```
typedef struct node {
    Location* loc; // item이 location인 링크
    struct node* next;
} QueueNode;
```

```
typedef struct queue {
    QueueNode* front;
    QueueNode* rear;
} QueueType;
```



4. 미로

```
#define SIZE 9
```

```
int maze[SIZE][SIZE] = {
    {-1, -1, -1, -1, -1, -1, -1, -1, -1},
    {-2, 0, -1, 0, 0, 0, 0, 0, -1},
    {-1, 0, -1, 0, -1, -1, 0, -1, -1},
    {-1, 0, 0, 0, 0, -1, 0, 0, -1},
    {-1, 0, -1, -1, -1, -1, -1, 0, -1},
    {-1, 0, 0, -1, 0, 0, 0, 0, -1},
    {-1, 0, -1, -1, 0, -1, -1, 0, -3},
    {-1, -1, -1, 0, 0, 0, -1, 0, -1},
    {-1, -1, -1, -1, -1, -1, -1, -1, -1}
};

// -3 : Exit
// -2 : Entrance
// -1 : Wall
// 0 : Road
// 1 : Visited Road
```


1	지나간 길	☆ (★)
0	지나가지 않은 길	□
-1	벽	■
-2	입구	▶
-3	출구	▷

3) 함수

3.1) Queue.h

3.1.1) 함수 선언

3.1.2) 함수 기술서

함수명	is_empty			Function 기술서	작성일	2018/05/2 6	Page 1/9
Header					작성자	정주원	
함수 설명	큐의 주소를 인자로 받아 큐가 공백 여부를 반환한다.						
Category	No.	Name	Data type	Description			
Parameter	1	q	QueueType*	비어 있는지 확인할 큐의 주소를 인자로 받는다.			
Return Value			int	큐가 비어 있다면 1, 그렇지 않다면 0을 반환한다.			

함수명	q_init		Function 기술서	작성일	2018/05/2 6	Page 2/9
Header				작성자	정주원	
함수 설명	큐를 사용하기 위해 초기화된 큐를 반환한다.					
Category	No.	Name	Data type	Description		
Return Value			QueueType*	초기화된 큐를 반환한다.		
Local Variable	1	q	QueueType*	동적으로 생성하여 초기화되는 큐로 구조체의 멤버가 초기화된 후 반환된다.		

함수명	enqueue			Function 기술서	작성일	2018/05/2	Page 3/9
Header					작성자	정주원	
함수 설명	큐의 주소와 갈 수 있는 공간의 좌표, 미로의 상태를 나타내는 이차원 배열을 인자로 받아 해당 좌표로 이동 가능하다면 그 좌표를 큐에 삽입한다.						
Category	No.	Name	Data type	Description			
Parameter	1	pQ	QueueType**	노드를 삽입할 큐의 주소를 인자로 받는다.			
	2	l	Location*	큐에 삽입할 공간의 좌표를 인자로 받는다.			
	3	maze	int[] []	미로의 상태를 나타내는 이차원 배열을 인자로 받는다.			
Return Value			void				
Local Variable	1	newQ	QueueNode*	동적으로 생성하여 큐에 삽입되는 노드로 인자로 받은 l을 저장하고 해당 좌표가 갈 수 있는 공간이라면 인자로 받은 pQ가 가리키는 큐에 저장된다.			

함수명	dequeue			Function 기술서	작성일	2018/05/2 6	Page 4/9
Header					작성자	정주원	
함수 설명	큐의 주소를 인자로 받아 맨 앞에 있는 노드를 삭제하고 그 값을 반환한다.						
Category	No.	Name	Data type	Description			
Parameter	1	pQ	QueueType**	노드를 삭제할 큐의 주소를 인자로 받는다.			
Return Value			Location*	삭제한 좌표 노드의 값을 반환한다.			
Local Variable	1	temp	Location*	맨 앞 노드를 삭제하기 전에 그 노드의 값을 저장해 놓았다가 노드 삭제 후 반환된다.			

함수명	init_loc			Function 기술서	작성일	2018/05/26	Page 5/9
Header					작성자	정주원	
함수 설명	특정 위치의 x값과 y값을 인자로 받아 좌표에 저장하고 이를 반환한다.						
Category	No.	Name	Data type	Description			
Parameter	1	x	int	특정 위치의 x값을 인자로 받는다.			
	2	y	int	특정 위치의 y값을 인자로 받는다.			
Return Value			Location*	x와 y가 정해진 좌표를 반환한다.			
Local Variable	1	l	Location*	동적으로 생성하여 인자로 받은 x와 y를 저장하고 반환된다.			

2.2) Maze.h

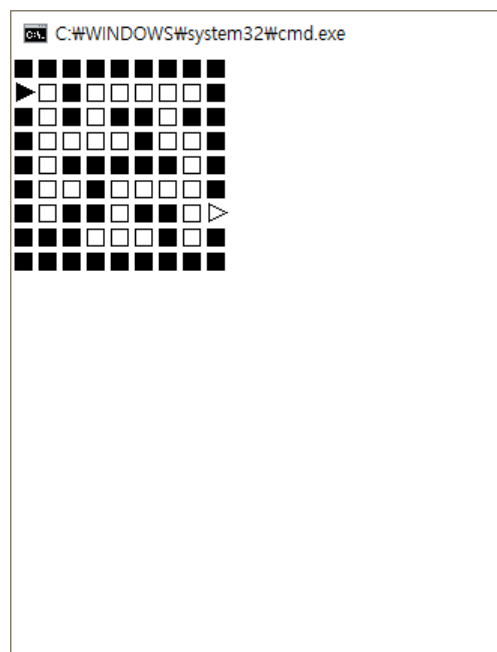
함수명	bfs			Function 기술서	작성일	2018/05/26	Page 6/9
Header					작성자	정주원	
함수 설명	미로의 상태를 나타내는 이차원 배열을 인자로 받아 이를 인자로 받은 큐의 주소가 가리키는 큐를 사용하여 BFS의 방식으로 탐색하고 성공 여부를 반환한다.						
Category	No.	Name	Data type	Description			
Parameter	1	maze	int[] []	미로의 상태를 나타내는 이차원 배열을 인자로 받는다.			
	2	pQ	QueueType**	갈 수 있는 공간의 좌표를 쌓아 둘 큐의 주소를 인자로 받는다.			
Return Value			int	미로 탐색 성공 여부를 반환한다. (성공 0 / 실패 -1)			
Local Variable	1	now	Location*	현재 위치의 좌표를 저장한다.			
	2	i	int	이차원 배열을 탐색하는 첫번째 인덱스로 사용된다.			
	3	j	int	이차원 배열을 탐색하는 두번째 인덱스로 사용된다.			

본격적으로 미로를 탐색하는 bfs() 함수는 미로의 상태를 나타내는 이차원배열 maze와 갈 수 있는 곳의 좌표를 쌓아 놓을 QueueType포인터 pQ를 인자로 받고 이차원배열 maze에서 입구(값: -2)를 찾아내 그 인덱스를 좌표로 하여 현 위치를 나타내는 Location 구조체 now를 그 좌표로 초기화하도록 하였다. 이는 탐색하고자 하는 미로의 입구의 좌표를 따로 입력해주지 않아도 미로의 정보만 있다면 정상적으로 입구에서 출발할 수 있도록 하기 위함이다.

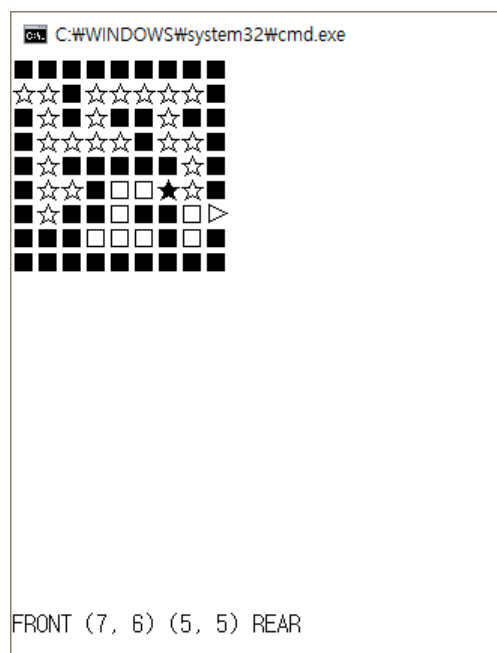
현위치가 입구의 좌표로 설정된 후에는 while문을 통해 출구(값: -3)에 도달하기 전까지 작업을 반복한다. 다음은 while문 내부에서 반복하는 내용이다. 이차원배열 maze에서 현위치를 지나간 길(값: 1)로 설정하고 상하좌우의 좌표를 enqueue한다. 현위치에서 갈 수 있는 모든 길이 queue에 들어가면 print_queue() 함수를 이용하여 queue에 들어있는 좌표들을 출력하고 print_maze() 함수를 이용하여 미로의 상태를 보여준다. 만약 출구를 찾지 못했는데 queue에 좌표가 남지 않는다면 탐색에 실패했음을 알리고 return한다. queue에 좌표가 존재한다면 현위치를 dequeue한 좌표로 설정한다.

while문을 마친 후 아직 출구는 지나가지 않은 상태이므로 현위치(=출구)를 지나간 길로 설정하고 queue와 maze를 한번 더 출력해준 뒤 탐색에 성공했음을 알리고 return한다

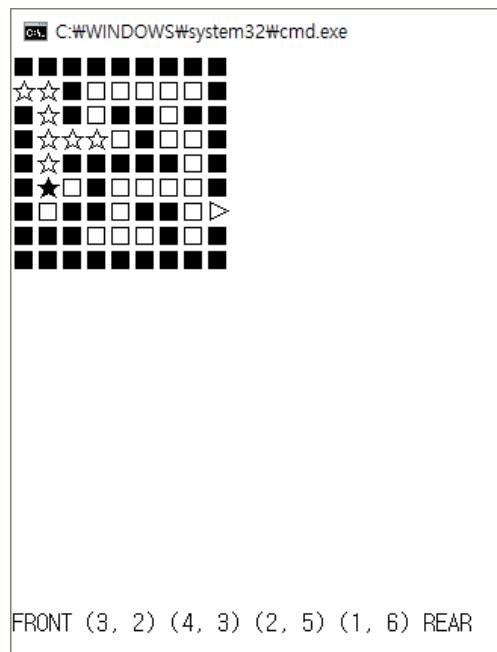
4) 실행결과



[탐색 시작하기 전]



[탐색 중 - 후반]



[탐색 중 - 초반]



[탐색 완료]

3.2. DFS(깊이우선탐색) - 김제현

1) 개요

일반적인 경우 미로에 대한 지식 없이 탐색을 하기에 Uniformed Search 방법 중 하나인 DFS를 적용한다. DFS는 탐색을 진행함에 있어서 가장 막다른상태까지 탐색하면서 진행하는 방식이다. 좀 더 풀어 설명하면, 매번 자식노드가 있을 때 마다, LIFO stack에 자식노드들을 넣으면서 탐색하다가 특정 상황에서 최대한 깊숙이 들어가서 확인한 뒤, 스택에서 가장 최근 상태를 꺼내어 체크하는 것을 반복하는 형태이다. DFS를 2가지 방식으로 구현했다.

첫 번째는 막다른 길에 도착했을 때, 스택에 저장된 가장 최근 갈림길로 점프하는 방식이고, 두 번째는 막다른 길에 도착했을 때, 스택에 저장된 가장 최근 갈림길까지 다시 돌아가는 방식이다.

2) 구조체

구조체명		Point	Struct 기술서	작성일	2018/05/26	Page 1/3
Header		Maze_DFS_BFS.c		작성자	김제현	
구조체 설명		지도상의 위치를 표현하는 구조체다.				
No	Variable name	Variable type	Variable Description			
1	x	int	지도상의 x위치를 표현하는 변수다.			
2	y	int	지도상의 x위치를 표현하는 변수다.			
3	d	int	방금 전의 위치에서, 자신이 왔던 방향을 의미하는 변수다. 동, 남, 서, 북 순서대로 1, 2, -1, -2에 대응한다.			
4	n	int	처음 출발로부터 자신이 있는 위치까지의 거리를 의미하는 변수다. 이 경우에는 보여주기식으로, 알고리즘상 필요하진 않다.			

구조체명		Node		Struct 기술서	작성일	2018/05/26	Page
Header		Maze_DFS_BFS.c			작성자	김제현	2/3
구조체 설명		스택에 대한 자료형으로, 지도상의 위치를 저장하기 위해 사용된다.					
No	Variable name	Variable type	Variable Description				
1	P	int	지도상의 위치가 저장되는 변수다.				
2	Next	Node*	다음 노드의 주소를 저장하는 변수다.				

구조체명		LinkedListType	Struct 기술서	작성일	2018/05/26	Page 3/3
Header		Maze_DFS_BFS.c		작성자	김제현	
구조체 설명		스택에 대한 자료형으로, 리스트의 끝과, 개수를 저장한다. 단일 연결리스트로 구현된다.				
No	Variable name	Variable type	Variable Description			
1	top	Node*	스택의 마지막 노드의 주소를 저장한다.			
2	length	int	스택에 저장된 노드의 개수를 저장한다.			

```
// 배열의 요소는 element타입으로 선언
typedef struct _point {
    int x;
    int y;
    int d; //direction : 자신이 왔던 방향을 의미함. 동, 남, 서, 북 순서대로 1, 2, -1, -2 매칭함 [서로 매칭함]
    int n; //number : 처음 출발부터 자신이 있는 위치까지 거리, 숫자
    //기존의 Maze_Ver2.c에서는 n의 값이 실질적으로 필요하지만, 지금의 경우는 그냥 보여주기 식으로 사용되며 알고리즘상으로 꼭 필요하지는 않다.
} Point;
```

```
//Point 구조체를 담는 연결 리스트의 노드
typedef struct _node Node;
typedef struct _node { //노드 타입
    Point p;
    Node *next;
} Node;

//DFS 스택 관련
typedef struct _LinkedListType{ //연결 리스트 스택의 타입
    Node *top; //의미상 tail에 해당
    int length; //현재 연결 리스트의 사용량
} LinkedListType;
```

3) 함수


3.1) Maze_DFS_BFS.c

함수명	init_S			Function 기술서	작성일	2018/05/26	Page 1/4
Header	Maze_DFS_BFS.c				작성자	김제헌	
함수 설명	스택의 주소를 받고, 이를 초기화한다.						
Category	No.	Name	Data type	Description			
Parameter	1	s	LinkedListType *	초기화할 스택의 주소			
Return Value			void				

함수명	is_empty_S			Function 기술서	작성일	2018/05/26	Page 2/4
Header	Maze_DFS_BFS.c				작성자	김제헌	
함수 설명	스택의 주소를 받고, 이게 비어있는지 검사한다.						
Category	No.	Name	Data type	Description			
Parameter	1	s	LinkedListType *	비어있는지 검사할 스택의 주소			
Return Value			int	스택이 비어있는지 여부 [1이면 비어있고, 0이면 안 비어있음]			


함수명	push			Function 기술서	작성일	2018/05/26	Page 3/4
Header	Maze_DFS_BFS.c				작성자	김제헌	
함수 설명	스택의 주소 및 지도상의 위치 값을 받아서, 스택에 저장한다.						
Category	No.	Name	Data type	Description			
Parameter	1	s	LinkedListType *	값을 추가할 스택의 주소			
	2	item	Point	스택에 추가될 지도상의 위치 값			
Return Value			void				
Local Variable	1	temp	Node*	동적으로 생성되는 노드로, 인자로 받은 item를 저장하고, 스택에 추가된다.			

함수명	pop			Function 기술서	작성일	2018/05/26	Page 4/4
Header	Maze_DFS_BFS.c				작성자	김제헌	
함수 설명	스택의 주소를 받아서, 스택에서 가장 마지막에 저장되었던 위치 값을 반환한다.						
Category	No.	Name	Data type	Description			
Parameter	1	s	LinkedListType *	값을 반환 및 제거할 스택의 주소			
Return Value			Point	스택에서 가장 마지막에 저장되었던 위치 값			
Local Variable	1	temp	Node*	스택에서 가장 마지막 노드의 주소를 받는 임시 변수로, p에 자기 자신의 값을 넘겨준 후, 할당해제된다.			
	2	p	Point	스택에 저장된 가장 마지막 위치 값을 받는 변수			

함수명	GetDirectionNum			Function 기술서	작성일	2018/05/26	Page 1/5
Header	Maze_DFS_BFS.c				작성자	김제현	
함수 설명	위치의 주소를 받고, 해당 위치에서 동서남북의 방향을 확인하고, 갈 수 있는 방향의 개수를 구한다.						
Cateogry	No.	Name	Data type	Description			
Parameter	1	p	Point*	갈 수 있는 방향을 확인할 위치의 주소			
Return Value			int	현재 위치에서 갈 수 있는 방향의 개수			
Local Variable	1	dn	int	현재 위치에서 갈 수 있는 방향의 개수 Direction Number의 약자			

함수명	GetDirection			Function 기술서	작성일	2018/05/26	Page 2/5
Header	Maze_DFS_BFS.c				작성자	김제현	
함수 설명	위치의 주소를 받고, 해당 위치에서 동서남북의 방향을 확인하고, 갈 수 있는 방향을 얻는다.						
Category	No.	Name	Data type	Description			
Parameter	1	p	Point*	갈 수 있는 방향을 확인할 위치의 주소			
Return Value			int	현재 위치에서 갈 수 있는 방향			
Local Variable	1	d	int	현재 위치에서 갈 수 있는 방향 Direction 의 약자			

함수명	isEqual			Function 기술서	작성일	2018/05/26	Page 3/5
Header	Maze_DFS_BFS.c				작성자	김제현	
함수 설명	두 위치의 주소를 받고, 두 위치 값이 같은지 확인하는 함수						
Category	No.	Name	Data type	Description			
Parameter	1	p1	Point*	비교할 첫 번째 위치의 주소			
	2	p2	Point*	비교할 두 번째 위치의 주소			
Return Value			int	현재 위치에서 갈 수 있는 방향			
Local Variable	1	isEqual	int	두 위치 값이 같은지를 나타내는 변수			

함수명	ChangeBoard			Function 기술서	작성일	2018/05/26	Page 4/5
Header	Maze_DFS_BFS.c				작성자	김제현	
함수 설명	위치의 주소와 숫자값을 받고, 지도상에 해당 위치의 값을 입력받은 숫자값으로 바꾼다.						
Category	No.	Name	Data type	Description			
Parameter	1	p	Point*	지도상에 표시될 위치의 주소			
	2	num	int	지도의 특정 위치에 표시될 숫자 값			
Return Value			void				

함수명	MovePoint			Function 기술서	작성일	2018/05/26	Page 5/5
Header	Maze_DFS_BFS.c				작성자	김제현	
함수 설명		위치의 주소를 받고, d의 방향으로 1칸 이동한다.					
Category	No.	Name	Data type	Description			
Parameter	1	p	Point*	이동할 위치의 주소			
Return Value			void				

함수명	DFS			Function 기술서	작성일	2018/05/2 6	Page 1/1
Header	Maze_DFS_BFS.c				작성자	김제현	
함수 설명	dfs알고리즘을 통해서, 미로를 탐색한다. 갈림길이 있을 때에만, 스택에 값을 넣으며, 막다른 길에 도달했을 경우, 가장 최근 값을 꺼내서 이동하며, 목적지에 도달할 때까지 반복하는 구조이다. 만일, 스택이 비어 있는 경우는 미로의 모든 경우를 살펴보았지만 목적지를 못 찾은 경우로, 해당 메시지를 출력하고 종료된다.						
Category	No.	Name	Data type	Description			
Parameter			void				
Return Value			void				
Local Variable	1	stack	LinkedStackType	미로 탐색을 할 때 도움을 주는 스택 [단일 연결리스트 형태로 구현되었다]			
	2	current_p	Point	현재 위치를 나타내는 변수			
	3	dn	int	현재 위치에서 갈 수 있는 방향의 개수 Direction Number의 약자			

3.2) Maze_Ver2[DFS].c

함수명	GetDirection2			Function 기술서	작성일	2018/05/2 6	Page 1/1
Header	Maze_Ver2[DFS].c				작성자	김제현	
함수 설명	위치의 주소를 받고, 해당 위치에서 동서남북의 방향을 확인하고, 자신이 왔던 길[=자기 자신의 위치의 숫자 - 1]로 가는 방향을 얻는다.						
Category	No.	Name	Data type	Description			
Parameter	1	p	Point*	갈 수 있는 방향을 확인할 위치의 주소			
Return Value			int	현재 위치에서 돌아갈 방향			
Local Variable	1	d	int	현재 위치에서 돌아갈 방향 Direction 의 약자			

함수명	ReturnSavePoint			Function 기술서	작성일	2018/05/2 6	Page 1/1
Header	Maze_Ver2[DFS].c				작성자	김제현	
함수 설명	갈 수 있는 위치의 개수가 0일 때, 호출되는 함수로, 현재위치에서 가장 최근 갈림길까지 돌아오는 함수이다.						
Category	No.	Name	Data type	Description			
Parameter	stack		LinkedStackType *	스택의 주소			
	current_p		Point	막다른 길에 도달했을 당시의 현재 위치 값			
Return Value			Point	가장 최근 갈림길의 위치 값			
Local Variable	1	save_p	Point	스택에서 꺼낸 가장 최근 갈림길의 위치 값을 저장하는 변수			
	2	dn	int	현재 위치에서 갈 수 있는 방향의 개수 Direction Number의 약자			

함수명	DFS2	Function 기술서	작성일	2018/05/26	Page 1/1
Header	Maze_Ver2[DFS].c		작성자	김제현	
함수 설명	dfs알고리즘을 통해서, 미로를 탐색한다.				

Category	No.	Name	Data type	Description
Parameter			void	
Return Value			void	
Local Variable	1	stack	LinkedListType	미로 탐색을 할 때 도움을 주는 스택 [단일 연결리스트 형태로 구현되었다]
	2	current_p	Point	현재 위치를 나타내는 변수
	3	dn	int	현재 위치에서 갈 수 있는 방향의 개수 Direction Number의 약자

4) 실행결과

4.1) Maze_DFS_BFS.c

DFS 미로 탐색 알고리즘 동작

current_p.n : 5, s->length = 0

미로 탐색 초반

DFS 미로 탐색 알고리즘 동작

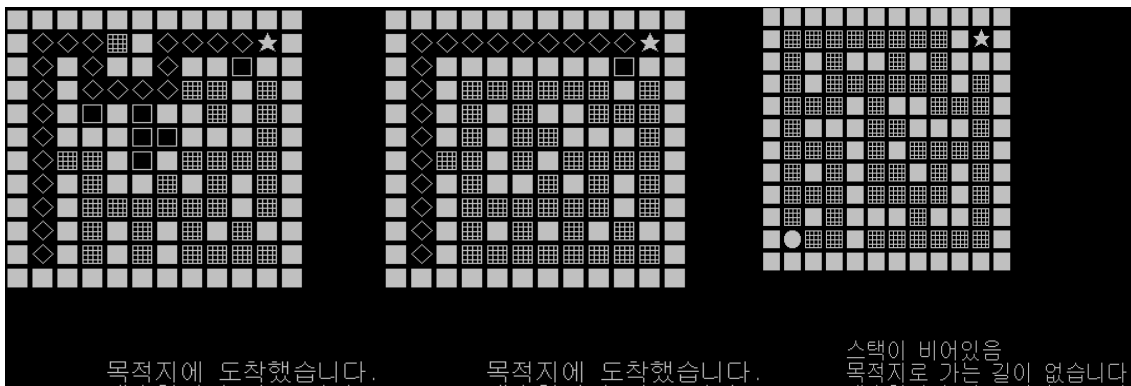
current_p.n : 23, s->length = 3
스택에 넣는 값 : [x,y,n] : [8, 0, 22]
목적지에 도착했습니다.
계속하려면 아무 키나 누르십시오 . . .

미로 탐색 끝난 후

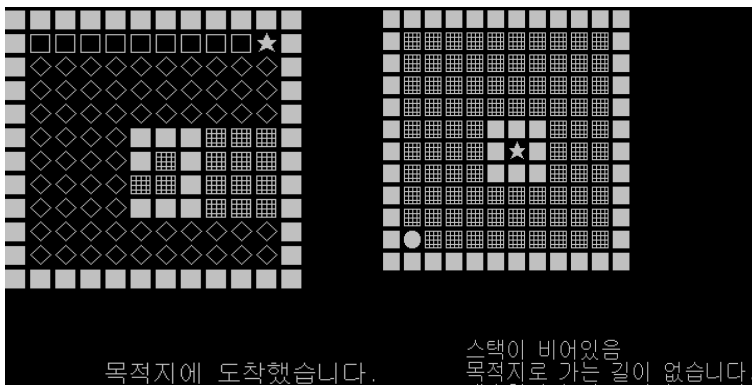
1이상	지나간 길	◇
0	지나가지 않은 길	□
-1	벽	■
-2	지나간 + 막다른 길	▣

4.2) Maze_Ver2[DFS].c

Case 1, 2, 3



Case 4, 5



1이상	지나간 길	◇
0	지나가지 않은 길	□
-1	벽	■
-2	지나간 + 막다른 길	▣

3.3. 다익스트라 및 A* - 김제현

1) 개요

앞선 알고리즘들은 가중치가 없다고 가정한 균일한 미로에서의 탐색이었고, 가중치가 있을 경우, 최소 비용의 경로를 탐색하기 위해서는 Uninformed Search의 다익스트라 알고리즘 및 Informed Search의 A* 알고리즘이 필요하다. 기존의 DFS 코드와 수업 PPT 자료에 나와있는 힛트리를 이용해서 간단하게 구현해 보았다.

이 코드 역시 기존 DFS 코드의 틀을 이용했으므로 변경된 내용만 언급하겠다.

2) 구조체

구조체명	HeapType	Struct 기술서	작성일	2018/06/05	Page
Header	Maze_Dijkstra_A_Star.c		작성자	김제현	1/1
구조체 설명	스택에 대한 자료형으로, 지도상의 위치를 저장하기 위해 사용된다.				

No	Variable name	Variable type	Variable Description
1	heap	Point	힛트리에 저장된 위치를 담는 배열이다.
2	Heap_size	int	힛트리에 저장된 위치의 개수를 저장한다.

```
//힛트리 관련
#define MAX_ELEMENT 200


// Point들을 담고 있는 힛트리
typedef struct {
    Point heap[MAX_ELEMENT];
    int heap_size;
} HeapType;
```

3) 함수

3.1) Maze_Dijkstra_A_Star.c 에서 다익스트라 알고리즘 관련 함수

함수명	init			Function 기술서	작성일	2018/06/05	Page 1/4
Header	Maze_Dijkstra_A_Star.c				작성자	김제현	
함수 설명	힛트리의 주소를 받고, 이를 초기화한다.						
Category	No.	Name	Data type	Description			
Parameter	1	h	HeapType*	초기화할 힛트리의 주소			
Return Value			void				

함수명	insert_min_heap		Function 기술서	작성일	2018/06/05	Page 2/4
Header	Maze_Dijkstra_A_Star.c			작성자	김제현	
함수 설명	힛트리의 주소 및 지도상의 위치 값을 받아서, 힛트리에 저장한다. [삽입될 당시, x,y의 위치가 미로 밖이 아니면서, 벽도 아니고, 이동한 값이 기존 값보다 더 적은 3 가지 조건을 만족한 경우에만 힛트리에 추가한다]					
Category	No.	Name	Data type	Description		
Parameter	1	h	HeapType*	값을 추가할 힛트리의 주소		
	2	item	Point	스택에 추가될 지도상의 위치 값		
	3	x	int	지도상의 위치의 x좌표 값		
	4	y	int	지도상의 위치의 y좌표 값		
	5	n	int	지도상의 위치의 가중치 값		
Return Value			void			
Local Variable	1	p	Point	정적으로 생성되는 위치 변수로, 인자로 받은 x,y,n을 저장하고, 힛트리에 추가된다.		
	2	i	int	힛트리 배열의 인덱스로 사용됨		

함수명	delete_min_heap			Function 기술서	작성일	2018/06/05	Page 3/4
Header	Maze_Dijkstra_A_Star.c				작성자	김제현	
함수 설명	힛트리의 주소를 받아서, 힛트리에서 가장 작은 n값의 위치 값을 반환한다.						
Category	No.	Name	Data type	Description			
Parameter	1	h	HeapType*	값을 반환 및 제거할 힛트리의 주소			
Return Value			Point	힛트리에서 가장 작은 n값의 위치 값			
Local Variable	1	p	Point	스택에 저장된 가장 작은 n의 위치 값을 받는 변수			
	2	i	int	힛트리 배열의 인덱스로 사용됨			
함수명	Dijkstra			Function 기술서	작성일	2018/06/05	Page

Header	Maze_Dijkstra_A_Star.c		작성자	김제현	4/4
함수 설명	Dijkstra 알고리즘을 통해서, 미로를 탐색한다. 현재 위치 기준으로 동서남북을 검사하여, 3가지 조건[x,y의 위치가 미로 밖이 아니면서, 벽도 아니고, 이동한 값이 기존 값보다 더 적은 조건]을 만족한 경우에만, 힛트리에 값을 넣는다. 동남서북 검사를 마친 후 힛트리에서 가장 작은 n의 위치값을 꺼내서 이동하며, 목적지에 도달할 때까지 반복하는 구조이다. 만일, 힛트리가 비어 있는 경우는 미로의 모든 경우를 살펴보았지만 목적지를 못 찾은 경우로, 해당 메시지를 출력하고 종료된다.				
Category	No.	Name	Data type	Description	
Parameter			void		
Return Value			void		
Local Variable	1	heap	HeapType	미로 탐색을 할 때 도움을 주는 힛트리 [배열 형태로 구현되었다]	
	2	current_p	Point	현재 위치를 나타내는 변수	

3.2) Maze_Dijkstra_A_Star.c 에서 A* 알고리즘 관련 함수

함수명	heuristic			Function 기술서	작성일	2018/06/05	Page
Header	Maze_Dijkstra_A_Star.c				작성자	김제현	1/3
함수 설명	현재 위치로부터 목적지까지의 휴리스틱 비용을 구한다.						
Category	No.	Name	Data type	Description			
Parameter	1	x	int	현재위치의 x좌표			
	2	y	Int	현재위치의 y좌표			
Return Value			void				

함수명	insert_min_heap_A			Function 기술서	작성일	2018/06/05	Page
Header	Maze_Dijkstra_A_Star.c				작성자	김제현	2/3
함수 설명	힛트리의 주소 및 지도상의 위치 값을 받아서, 힛트리에 저장한다. 이때 휴리스틱 함수의 값도 같이 더한 값을 저장한다. [삽입될 당시, x,y의 위치가 미로 밖이 아니면서, 벽도 아니고, 이동한 값이 기존 값보다 더 적은 3가지 조건을 만족한 경우에만 힛트리에 추가한다]						
Category	No.	Name	Data type	Description			
Parameter	1	h	HeapType*	값을 추가할 힛트리의 주소			
	2	item	Point	스택에 추가될 지도상의 위치 값			
	3	x	int	지도상의 위치의 x좌표 값			
	4	y	int	지도상의 위치의 y좌표 값			
	5	n	int	지도상의 위치의 가중치 값			
Return Value			void				
Local Variable	1	p	Point	정적으로 생성되는 위치 변수로, 인자로 받은 x,y,n을 저장하고, 힛트리에 추가된다.			
	2	i	int	힛트리 배열의 인덱스로 사용됨			

함수명	A_Star	Function 기술서	작성일	2018/06/05	Page
Header	Maze_Dijkstra_A_Star.c		작성자	김제현	3/3
함수 설명	A_Star 알고리즘을 통해서, 미로를 탐색한다. 현재 위치 기준으로 동서남북을 검사하여, 3가지 조건[x,y의 위치가 미로 밖이 아니면서, 벽도 아니고, 이동한 값이 기존 값보다 더 적은 조건]을 만족한 경우에만, 힛트리에 값을 넣는다. [이때 휴리스틱 함수의 값도 같이 더한 값을 저장]. 동남서북 검사를 마친 후 힛트리에서 가장 작은 n의 위치값을 꺼내서 이동하며, 목적지에 도달할 때까지 반복하는 구조이다. 만일, 힛트리가 비어 있는 경우는 미로의 모든 경우를 살펴보았지만 목적지를 못 찾은 경우로, 해당 메시지를 출력하고 종료된다.				

Category	No.	Name	Data type	Description
Parameter			void	
Return Value			void	
Local Variable	1	heap	HeapType	미로 탐색을 할 때 도움을 주는 힙트리 [배열 형태로 구현되었다]
	2	current_p	Point	현재 위치를 나타내는 변수

4) 실행결과

4.1) Dijkstra 알고리즘

원하는 미로 탐색 알고리즘을 입력하십시오 [1:Dijkstra, 그외 : A *] : 1
Dijkstra 알고리즘 미로 탐색 알고리즘 동작



```

current_p.n : 44, h.heap_size = 11
힙트리에 넣는 값 : [x,y,n]: [ 0, 0, 45]

```

미로 탐색 초반

원하는 미로 탐색 알고리즘을 입력하십시오 [1:Dijkstra, 그외 : A *] : 1
Dijkstra 알고리즘 미로 탐색 알고리즘 동작



```

current_p.n : 63, h.heap_size = 4
힙트리에 넣는 값 : [x,y,n]: [ 9, 0, 64]
목적지에 도착했습니다.
계속하려면 아무 키나 누르십시오 . . .

```

미로 탐색 끝난 후

1이상	지나간 길	◇
9999	지나가지 않은 길	□
-1	벽	■

4.2) A* 알고리즘

원하는 미로 탐색 알고리즘을 입력하십시오 [1:Dijkstra, 그외 : A *] : 2
A * 알고리즘 미로 탐색 알고리즘 동작



current_p.n : 1596, h.heap_size = 10
힙트리에 넣는 값 : [x,y,n] : [1, 4, 1854]
휴리스틱 비용 : 200

미로 탐색 초반

원하는 미로 탐색 알고리즘을 입력하십시오 [1:Dijkstra, 그외 : A *] : 2
A * 알고리즘 미로 탐색 알고리즘 동작



current_p.n : 2409, h.heap_size = 10
힙트리에 넣는 값 : [x,y,n] : [9, 5, 2517]
목적지에 도착했습니다.
계속하려면 아무 키나 누르십시오 . . .

미로 탐색 끝난 후

1이상	지나간 길	◇
9999	지나가지 않은 길	□
-1	벽	■

3.4. A* - 박상희

1) 개요

Informed Search에서 가장 기본적인 탐색 알고리즘이다. A*알고리즘에 대한 설명은 위에서 했으므로 생략한다. 블록을 좌표로 표현하고, 비용함수는 블록을 이동한 거리이고, 휴리스틱 비용은 노드와 목적지 노드 사이의 직선거리다. 또한 현재 비용과, 휴리스틱 비용의 합을 통해 최단 경로를 결정하므로 우선순위 큐를 구현하여, 사용한다. 또한, 노드(갈림길 지점)와 간선(갈림길)을 저장하기 위해 그래프 자료형을 구현하여 사용한다.

2) 구조체

2.1) MyHeap.h

구조체명		HeapData		Struct 기술서	작성일	2018/05/26	Page
Header		MyHeap.h			작성자	박상희	3/3
구조체 설명		우선순위 큐에 저장될 자료형으로, key와 value로 이루어져 있다.					
No	Variable name	Variable type	Variable Description				
1	key	int	우선순위 큐에서 정렬 기준이 되는 값으로 A* 알고리즘에서 value에 저장된 노드로 이동하는 비용이 저장되어 비용을 기준으로 정렬된다.				
2	value	int	key에 따른 value로 A* 알고리즘에서 노드 번호가 저장된다.				

구조체명		HeapType	Struct 기술서	작성일	2018/05/26	Page
Header		MyHeap.h		작성자	박상희	3/3
구조체 설명		HeapData에 대한 1차원 배열과 힙의 원소 개수를 저장한다.				
No	Variable name	Variable type	Variable Description			
1	heap	HeapData []	HeapData를 저장하는 변수다.			
2	heap_size	int	저장된 힙의 개수이다.			

```
typedef struct _HeapData {
    int key;
    int value;
} HeapData;

typedef struct _HeapType {
    HeapData heap[MAX_ELEMENT];
    int heap_size;
} HeapType;
```

2.2) MyAStar.h

구조체명		GraphNode	Struct 기술서	작성일	2018/05/26	Page
Header		MyAStar.h		작성자	박상희	3/3
구조체 설명		GraphType에서 사용되는 구조체로 특정 노드에서 갈 수 있는 노드를 저장하기 위해 사용된다.				
No	Variable name	Variable type	Variable Description			
1	vertex	int	특정 노드에서 갈 수 있는 노드의 번호다.			
2	weight	int	특정 노드에서 vertex 노드로 가는데 드는 비용이다.			
3	next	GraphNode*	특정 노드에서 갈 수 있는 다른 노드의 주소를 저장한다.			

구조체명		GraphType	Struct 기술서	작성일	2018/05/26	Page
Header		MyAStar.h		작성자	박상희	3/3
구조체 설명		그래프의 간선이 저장되는 구조체다.				
No	Variable name	Variable type	Variable Description			
1	n	int	그래프의 노드 개수다.			
2	sour	int	그래프에서 출발지 노드의 번호다.			
3	dest	int	그래프에서 목적지 노드의 번호다.			
4	adjList_H	GraphNode* []	인덱스는 노드의 번호로, 인덱스에 대한 값은 인덱스 번호인 노드에서 갈 수 있는 노드들의 주소를 가리킨다. 예를 들어 [4]의 경우 4번 노드에서 갈 수 있는 노드가 저장되어 있다. 만일 없으면 NULL이 저장되어 있고, 여러 개 있으면 Node의 next를 통해 리스트 구조를 띄고 있다.			
5	heuri	float []	인덱스는 노드의 번호, 인덱스에 대한 값은 해당 인덱스 번호인 노드의 휴리스틱 비용이다. 예를 들어 [4]는 4번 노드의 휴리스틱 비용이다.			

```

typedef struct _GraphNode {
    int vertex;
    int weight;
    struct _GraphNode* next;
} GraphNode;

typedef struct _GraphType {
    int n;
    int sour;
    int dest;
    GraphNode* adjList_H[MAX_VERTEX];
    float heuri[MAX_VERTEX];
} GraphType;

```

3) 함수


3.1) MyHeah.h

```

HeapType* createHeap();
void insertHeap(HeapType *heap, int key, int value);
HeapData* deleteHeap(HeapType *heap, HeapData* data);
void printHeap(HeapType *heap);

```

함수명	createHeap			Function 기술서	작성일	2018/05/26	Page
Header	MyHeap.h				작성자	박상희	1/5
함수 설명	힙을 생성하고 초기화한 후 반환하는 함수다.						
Category	No.	Name	Data type	Description			
Parameter			void				
Return Value			HeapType*	동적 할당을 통해 생성한 HeapType의 주소를 반환한다.			
Local Variable							

함수명	insertHeap			Function 기술서	작성일	2018/05/26	Page 1/5
Header	MyHeap.h				작성자	박상희	
함수 설명	힙에 HeapData를 추가하는 함수다.						
Category	No.	Name	Data type	Description			
Parameter	1	heap	HeapType*	HeapData를 저장할 힙의 주소를 인자로 받는다.			
	2	key	int	HeapData에 들어갈 key값을 인자로 받는다.			
	3	value	int	HeapData에 들어갈 value값을 인자로 받는다.			
Return Value			void				
Local Variable							

함수명	deleteHeap			Function 기술서	작성일	2018/05/26	Page 1/5
Header	MyHeap.h				작성자	박상희	
함수 설명	힙에 있는 가장 key값이 작은 HeapData를 힙에서 제거하고 반환하는 함수다.						
Category	No.	Name	Data type	Description			
Parameter	1	heap	HeapType*	HeapData를 제거해 반환할 힙을 인자로 받는다.			
	2	data	HeapData*	key값이 가장 작은 HeapData를 저장하기 위해 인자로 받는다.			
Return Value			HeapData*	key값이 가장 작은 HeapData를 저장하기 위해 인자로 받은 변수를 반환한다.			
Local Variable	1	parent	int				
	2	child	Int				
	3	key	int				
	4	value	int				

3.2) MyAStar.h

```

GraphType* createGraph(GraphType* graph);
void insertVertex(GraphType* graph, int v, int x, int y, int dest_x, int dest_y);
GraphNode* insertEdge(GraphType* graph, int u, int v, int w);
void insertEdgeByside(GraphType* graph, int u, int v, int w);
void deleteVertex(GraphType* graph, int v);
void deleteEndNode(GraphType* graph);
int getDegree(GraphNode* node);
void explorGraph(GraphType* graph);
void createMazeGraph(GraphType* graph);
void print_adjList(GraphType* graph);


```

함수명	createGraph			Function 기술서	작성일	2018/05/26	Page 1/5
Header	MyAStar.h				작성자	박상희	
함수 설명	그래프를 동적으로 생성하여 초기화한 후 반환하는 함수다.						
Category	No.	Name	Data type	Description			
Parameter	1	hraph	GraphType*	생성한 GraphType의 주소를 저장하기 위해 인자로 받는다.			
Return Value			GraphType*	생성한 GraphType의 주소를 반환한다.			
Local Variable							

함수명	insertVertex	Function 기술서	작성일	2018/05/26	Page
Header	MyAStar.h		작성자	박상희	1/5

함수 설명	그래프에 노드를 추가하는 함수다. 노드와 목적지 노드 사이의 직선거리를 휴리스틱 비용으로 계산해 저장한다.			
Category	No.	Name	Data type	Description
Parameter	1	graph	GraphType*	노드를 저장할 그래프를 인자로 받는다.
	2	v	int	노드의 번호다.
	3	x	int	노드의 x좌표 값이다.
	4	y	int	노드의 y좌표 값이다.
	5	dest_x	int	목적지 노드의 x좌표 값이다.
	6	dest_y	int	목적지 노드의 y좌표 값이다.
Return Value			GraphType*	생성한 GraphType의 주소를 반환한다.
Local Variable				

함수명	insertEdge			Function 기술서	작성일	2018/05/26	Page
Header	MyAStar.h				작성자	박상희	1/5
함수 설명	그래프에 간선을 추가하는 함수다. GraphNode를 동적으로 생성하여 u노드에서 v노드로의 간선과 거리를 저장하고 반환한다. 만일 u노드에 다른 간선들이 있으면 GraphNode의 next 변수를 사용해 연결한다.						
Category	No.	Name	Data type	Description			
Parameter	1	graph	GraphType*	간선을 저장할 그래프를 인자로 받는다.			
	2	u	int	노드의 번호다.			
	3	v	int	노드의 번호다.			
	4	w	int	간선의 거리다.			
Return Value			GraphNode*	저장한 간선을 반환한다.			
Local Variable							


함수명	insertEdgeByside		Function 기술서	작성일	2018/05/26	Page
Header	MyAStar.h			작성자	박상희	1/5
함수 설명	그래프에 간선을 양방향으로 추가하는 함수다. u->v, v->u 방향으로 간선을 저장한다.					
Category	No.	Name	Data type	Description		
Parameter	1	graph	GraphType*	간선을 저장할 그래프를 인자로 받는다.		
	2	u	int	노드의 번호다.		
	3	v	int	노드의 번호다.		
	4	w	int	간선의 거리다.		
Return Value			void			
Local Variable						

함수명	deleteVertex			Function 기술서	작성일	2018/05/26	Page
Header	MyAStar.h				작성자	박상희	1/5
함수 설명	그래프에서 노드를 삭제하는 함수다. 그래프에서 노드를 삭제할 때 노드에 연결되어 있는 간선들도 제거한다.						
Category	No.	Name	Data type	Description			
Parameter	1	graph	GraphType*	삭제할 노드를 저장한 그래프를 인자로 받는다.			
	2	v	int	삭제할 노드의 번호다.			
Return Value			void				
Local Variable							

함수명	createMazeGraph		Function 기술서	작성일	2018/05/26	Page
-----	-----------------	--	--------------	-----	------------	------

Header	MyAStar.h			작성자	박상희	1/5
함수 설명	미로를 노드와 간선으로 이루어진 그래프로 만드는 함수다. 단말노드의 경우 휴리스틱 비용을 큰 값으로 설정한다.					
Category	No.	Name	Data type	Description		
Parameter	1	graph	GraphType*	미로 정보를 저장할 그래프를 인자로 받는다.		
Return Value			void			
Local Variable						
함수명	exploreGraph		Function 기술서	작성일	2018/05/26	Page
Header	MyAStar.h			작성자	박상희	1/5
함수 설명	A* 알고리즘을 기반으로 그래프를 탐색하는 함수다. 갈 수 있는 노드를 우선순위 큐에 저장하고 빼는 방식을 통해 목적지 노드를 탐색한다. 다만, 방문했던 노드는 다시 방문하지 않는다. Heap의 key 값에 이전까지 이동 비용과 간선의 길이, 휴리스틱 비용을 합하고, value에는 노드의 번호를 넣고 저장한다. 그리고 그 중 가장 작은 값을 큐에서 빼내 다시 탐색한다.					
Category	No.	Name	Data type	Description		
Parameter	1	graph	GraphType*	탐색할 그래프를 인자로 받는다.		
Return Value			void			
Local Variable	1	heap	HeapType*	노드와 노드의 비용을 저장할 우선순위 큐이다.		
	2	data	HeapData*	큐에서 빼낸 데이터를 저장하기 위한 변수다.		
	3	node	GraphNode*	노드의 연결된 노드를 탐색하기 위한 변수다.		
	4	visited	int[]	배열의 값을 0으로 초기화하고, 방문하면 방문한 노드 번호의 인덱스 값을 1로 변경한다. 예를들어 4번 노드 방문 시 visited[4]=1로 한다. 또한, 방문했던 노드는 다시 큐에 저장하지 않는다.		

함수명	deleteEndNode			Function 기술서	작성일	2018/05/26	Page
Header	MyAStar.h				작성자	박상희	1/5
함수 설명	그래프에서 단말 노드를 삭제하는 함수다.						
Category	No.	Name	Data type	Description			
Parameter	1	graph	GraphType*	단말 노드를 삭제할 그래프를 인자로 받는다.			
Return Value			void				
Local Variable							

함수명	getDegree			Function 기술서	작성일	2018/05/26	Page
Header	MyAStar.h				작성자	박상희	1/5
함수 설명	노드에 연결된 노드(간선)의 수를 반환하는 함수다.						
Category	No.	Name	Data type	Description			
Parameter	1	node	GraphNode*	연결된 간선의 수가 궁금한 노드를 인자로 받는다.			
Return Value			Int	노드에 연결된 간선의 수를 반환한다.			
Local Variable	1	count	int	노드의 next를 따라 순회하며 count를 1씩 증가시킨 후 끝나면 반환된다.			

4) 실행결과

4.1) 예제 1

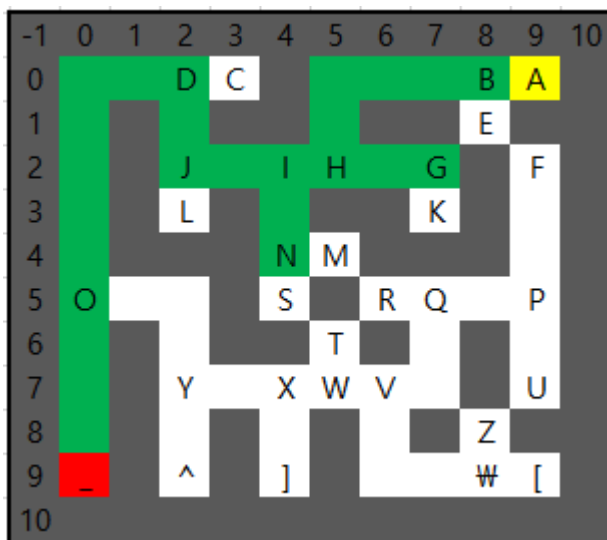
4.1.1)미로와 미로를 그래프로 변환


```

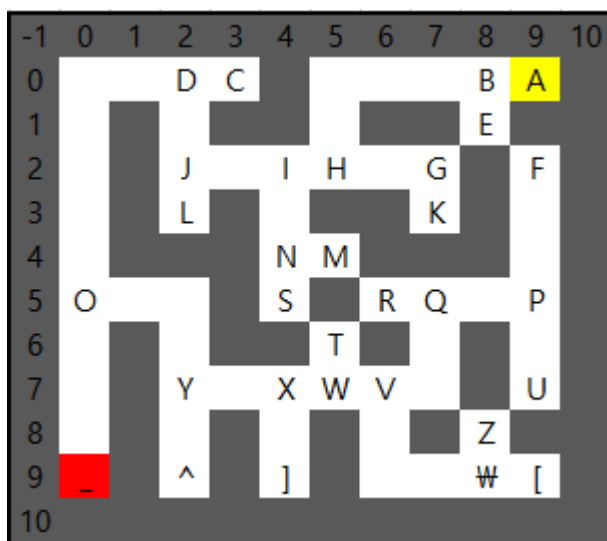
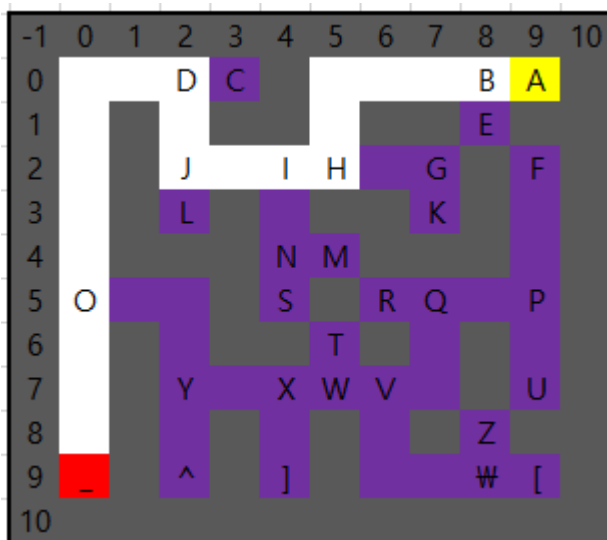
현재 A노드 도착
Heap : [13.04, 1]
현재 B노드 도착
Heap : [14.60, 7] [102.00, 4]
현재 H노드 도착
Heap : [15.06, 8] [102.00, 4] [17.90, 6]
현재 I노드 도착
Heap : [15.40, 13] [16.28, 9] [17.90, 6] [102.00, 4]
현재 N노드 도착
Heap : [16.28, 9] [102.00, 4] [17.90, 6] [110.00, 18] [110.00, 12]
현재 J노드 도착
Heap : [17.90, 6] [20.22, 3] [110.00, 12] [110.00, 18] [102.00, 4] [110.00, 11]
현재 G노드 도착
Heap : [20.22, 3] [102.00, 4] [109.00, 10] [110.00, 18] [110.00, 11] [110.00, 12]
현재 D노드 도착
Heap : [27.85, 14] [110.00, 12] [102.00, 4] [110.00, 18] [110.00, 11] [109.00, 10] [112.00, 2]
현재 O노드 도착
Heap : [22.00, 30] [24.83, 24] [102.00, 4] [110.00, 12] [110.00, 11] [112.00, 2] [109.00, 10] [110.00, 18]
현재 _노드 도착

목적지 도착

```



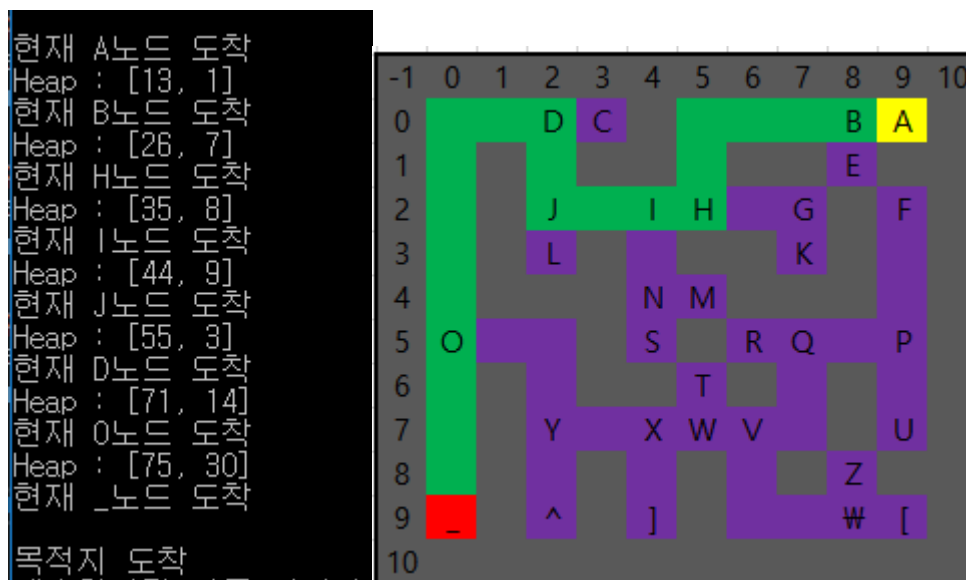
4.1.4.1)미로 이미지

 $\mathbb{I} = \mathbb{J}$ 

4.1.4.2) 그래프 출력

```
graph의 인접리스트
A의 인접리스트 -> B
B의 인접리스트 -> H -> A
C의 인접리스트
D의 인접리스트 -> O -> J
E의 인접리스트
F의 인접리스트
G의 인접리스트
H의 인접리스트 -> I -> B
I의 인접리스트 -> J -> H
J의 인접리스트 -> D -> I
K의 인접리스트
L의 인접리스트
M의 인접리스트
N의 인접리스트
O의 인접리스트 -> _ -> D
P의 인접리스트
Q의 인접리스트
R의 인접리스트
S의 인접리스트
T의 인접리스트
U의 인접리스트
V의 인접리스트
W의 인접리스트
X의 인접리스트
Y의 인접리스트
Z의 인접리스트
[_]의 인접리스트
[^]의 인접리스트
]의 인접리스트
-> 0
```

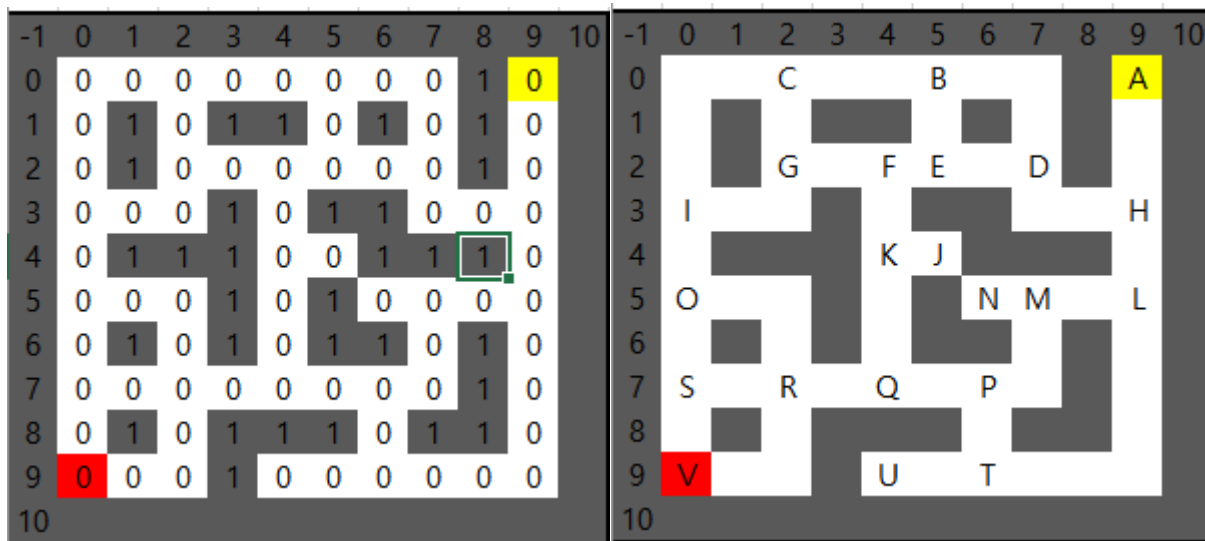
4.1.1) A* 알고리즘으로 탐색한 결과



단일 간선으로 이루어져 있음을 알 수 있다.

4.2) 예제 2

4.2.1)미로와 미로를 그래프로 변환



4.2.2)그래프 출력

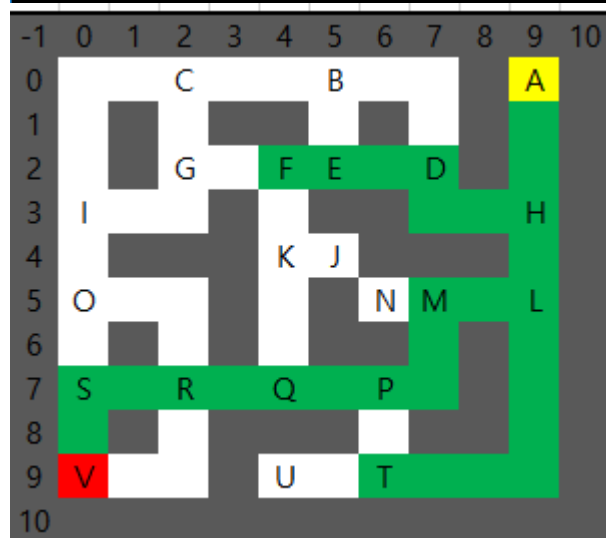
```
graph2의 인접리스트
A의 인접리스트: B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
B의 인접리스트: A, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
C의 인접리스트: A, B, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
D의 인접리스트: A, B, C, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
E의 인접리스트: A, B, C, D, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
F의 인접리스트: A, B, C, D, E, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
G의 인접리스트: A, B, C, D, E, F, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
H의 인접리스트: A, B, C, D, E, F, G, I, J, K, L, M, N, O, P, Q, R, S, T, U, V
I의 인접리스트: A, B, C, D, E, F, G, H, K, L, M, N, O, P, Q, R, S, T, U, V
J의 인접리스트: A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, U, V
K의 인접리스트: A, B, C, D, E, F, G, H, I, J, L, M, N, O, P, Q, R, S, T, U, V
L의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, M, N, O, P, Q, R, S, T, U, V
M의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, N, O, P, Q, R, S, T, U, V
N의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, O, P, Q, R, S, T, U, V
O의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q, R, S, T, U, V
P의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, Q, R, S, T, U, V
Q의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, R, S, T, U, V
R의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, S, T, U, V
S의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, T, U, V
T의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, U, V
U의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, V
V의 인접리스트: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U
-> H
-> E
-> I
-> E
-> F
-> K
-> I
-> L
-> O
-> K
-> Q
-> T
-> P
-> M
-> S
-> T
-> R
-> S
-> V
-> U
-> T
-> S
-> C
-> G
-> B
-> B
-> B
-> G
-> C
-> D
-> D
-> G
-> J
-> M
-> N
-> R
-> Q
-> P
-> R
-> B
-> R
-> P
-> Q
-> R
-> 0
```

4.2.3) A* 알고리즘으로 탐색한 결과

```

현재 A노드 도착
Heap : [13.82, 7]
현재 H노드 도착
Heap : [14.85, 11] [15.90, 3]
현재 L노드 도착
Heap : [15.06, 12] [18.00, 19] [15.90, 3]
현재 M노드 도착
Heap : [15.90, 3] [18.00, 19] [16.32, 15] [108.00, 13]
현재 D노드 도착
Heap : [16.32, 15] [16.60, 4] [108.00, 13] [18.00, 19] [20.30, 1]
현재 P노드 도착
Heap : [16.47, 16] [16.60, 4] [108.00, 13] [20.30, 1] [18.00, 19]
현재 Q노드 도착
Heap : [16.60, 4] [16.83, 17] [21.40, 10] [20.30, 1] [18.00, 19] [108.00, 13]
현재 E노드 도착
Heap : [16.83, 17] [18.00, 19] [17.06, 5] [20.30, 1] [108.00, 13] [21.40, 10]
현재 R노드 도착
Heap : [17.06, 5] [18.00, 19] [18.00, 18] [20.30, 1] [108.00, 13] [21.40, 10] [18.00, 21] [22.00, 14]
현재 F노드 도착
Heap : [18.00, 19] [18.28, 6] [18.00, 18] [20.30, 1] [108.00, 13] [21.40, 10] [18.00, 21] [22.00, 14]
현재 T노드 도착
Heap : [18.00, 18] [18.28, 6] [18.00, 21] [20.30, 1] [108.00, 13] [21.40, 10] [22.00, 14] [114.00, 20]
현재 S노드 도착
Heap : [18.00, 21] [18.28, 6] [21.40, 10] [20.30, 1] [108.00, 13] [114.00, 20] [22.00, 14]
현재 V노드 도착

```



4. 변외

4.1. AVL 트리 - 임수연

1) AVL 트리란?

AVL 트리는 Adelson_Velskii와 Landis에 의해 1962년에 제안된 트리로서 각 노드에서 왼쪽 서브 트리의 높이와 오른쪽 서브 트리의 높이 차이가 1이하인 이진 탐색 트리를 말한다. AVL트리는 트리가 비균형 상태로 되면 스스로 노드들을 재배치하여 균형 상태로 만든다. 따라서 AVL트리는 균형 트리가 항상 보장되기 때문에 탐색이 $O(\log_2 n)$ 시간 안에 끝나게 된다. 또한 삽입과 삭제 연산도 $O(\log_2 n)$ 시간 안에 할 수 있다. (왼쪽 서브 트리의 높이 - 오른쪽 서브 트리의 높이)로 균형인수가 정의 되는데, 모든 노드의 균형 인수가 ± 1 이하이면 AVL트리이다.

2) AVL 트리의 균형 맞추기

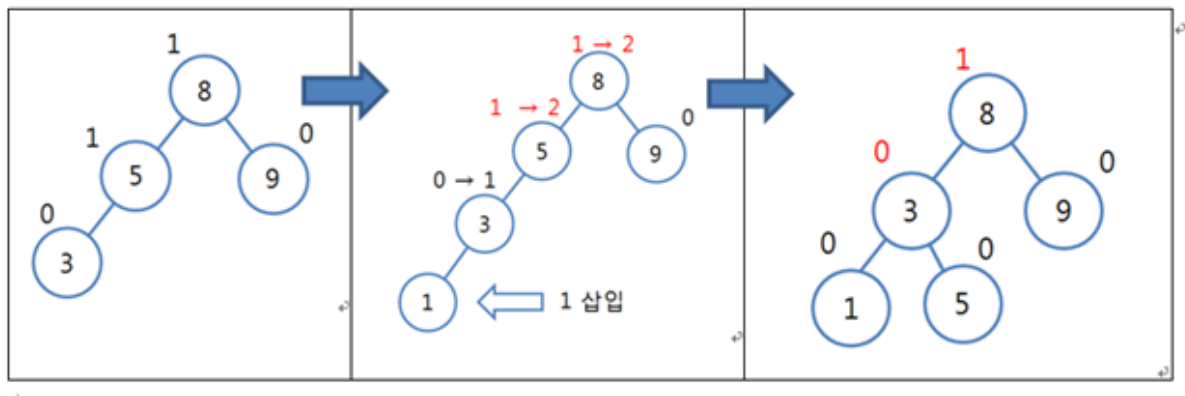
균형을 이룬 이진 탐색 트리에서는 삽입 연산과 삭제 연산 때문에 균형 상태가 깨질 수 있다. 따라서 새로운 노드의 삽입 후에 불균형 상태로 변한 가장 가까운 조상 노드, 즉 균형 인수가 ± 2 가 된 가장 가까운 조상 노드의 서브 트리들에 대하여 다시 균형을 잡아야 한다.

AVL트리에서 균형이 깨지는 경우에는 다음의 4가지 경우가 있다. 새로 삽입된 노드 N으로부터 가장 가까우면서 균형 인수가 ± 2 가 된 조상 노드를 A라고 하자.

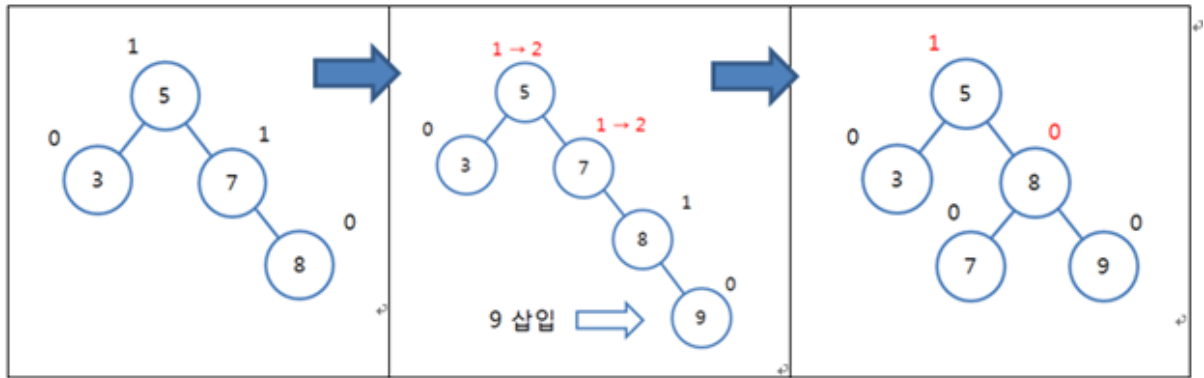
- LL타입 : N이 A의 왼쪽 서브 트리의 왼쪽 서브 트리에 삽입된다.
- RR타입 : N이 A의 오른쪽 서브 트리의 오른쪽 서브 트리에 삽입된다.
- RL타입 : N이 A의 오른쪽 서브 트리의 왼쪽 서브 트리에 삽입된다.
- LR타입 : N이 A의 왼쪽 서브 트리의 오른쪽 서브 트리에 삽입된다.

다음은 각각의 경우에 대해 균형 트리로 다시 만드는 방법이다.

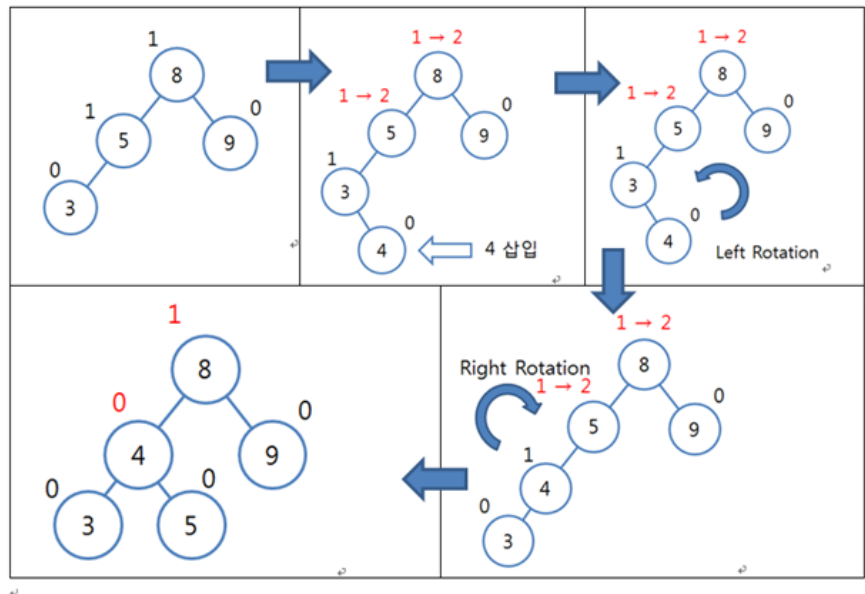
- LL회전: A부터 N까지의 경로상의 노드들을 오른쪽으로 회전시킨다.



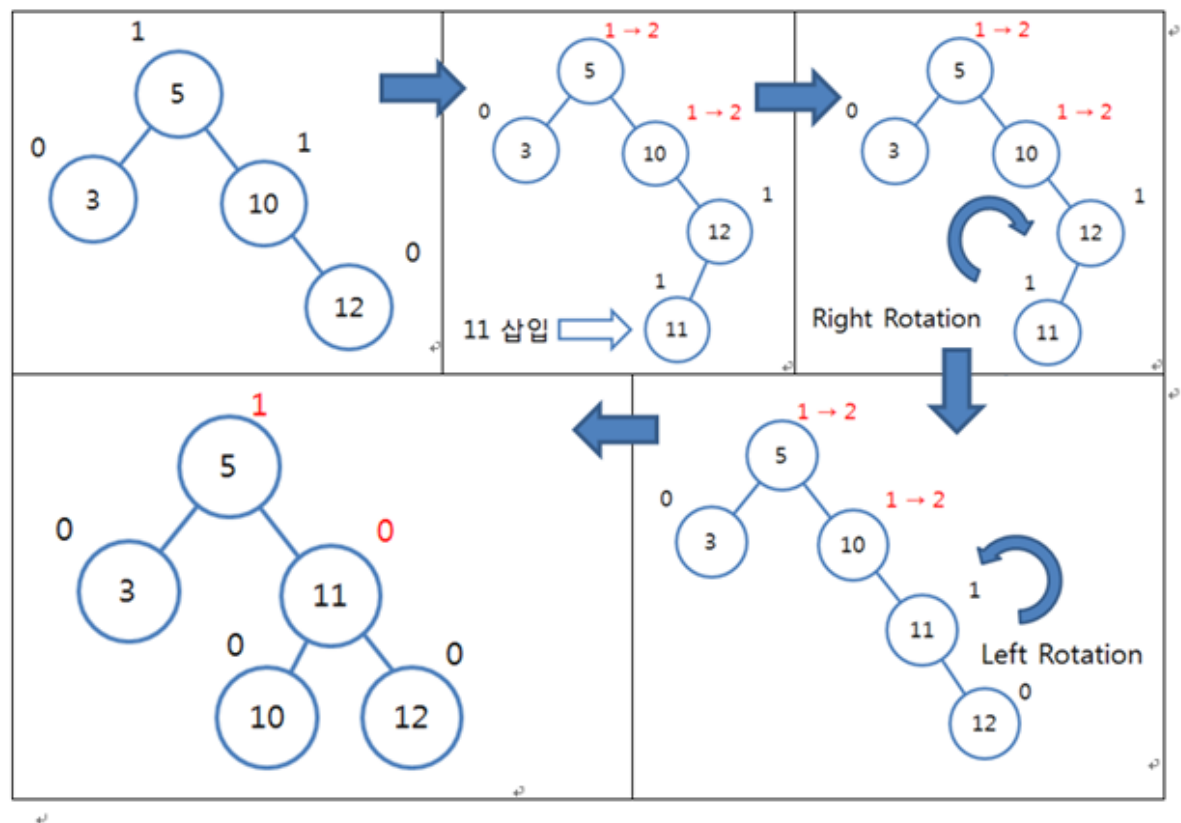
- RR회전 : A부터 N까지의 경로상의 노드들을 왼쪽으로 회전시킨다.



- RL회전 : A부터 N까지의 경로상의 노드들을 오른쪽-왼쪽으로 회전시킨다.



- LR회전 : A부터 N까지의 경로상의 노드들을 왼쪽-오른쪽으로 회전시킨다.



한번만 회전시키는 것을 단순회전(Single rotation)이라고 하며 LL회전, RR회전이 여기에 속한다. 이 경우, 탐색 순서를 유지하면서 부모와 자식 원소의 위치를 교환하면 된다. 두번의 회전이 필요한 것을 이중 회전 (double rotation) 이라고 하며 LR회전, RL회전이 여기에 속한다.

3) 구현

3.1) 구조체

1.1.

구조체명		AvlNode		Struct 기술서	작성일	2018/05/28	Page 1/1
Header		AVL_TREE.h			작성자	임수연	
구조체 설명		AVL 트리의 노드에 대한 자료형으로 key 값과 자식노드에 대한 정보를 저장한다.					
No	Variable name	Variable type	Variable Description				
1	data	int	Key가 되는 data값을 저장하는 변수다.				
2	left_child	AvlNode*	왼쪽 자식노드의 주소를 저장하는 변수다.				
3	right_child	AvlNode*	오른쪽 자식노드의 주소를 저장하는 변수다				

3.2) 함수


함수명	rotate_LL			Function 기술서	작성일	2018/05/28	Page 1/
Header	AVL_TREE.h				작성자	임수연	
함수 설명	부모노드를 인자로 받아 오른쪽으로 회전시킨다.						
Category	No.	Name	Data type	Description			
Parameter	1	parent	AvlNode*	부모노드의 주소를 인자로 받는다..			
Return Value			AvlNode*	자식노드의 주소를 반환한다.			
Local Variable	1	child	AvlNode*	부모노드의 왼쪽 자식노드의 주소를 저장하고 부모노드를 오른쪽 자식노드로 받는다.			

함수명	rotate_RR			Function 기술서	작성일	2018/05/28	Page 1/
Header	AVL_TREE.h				작성자	임수연	
함수 설명	부모노드를 인자로 받아 왼쪽으로 회전시킨다.						
Category	No.	Name	Data type	Description			
Parameter	1	parent	AvlNode*	부모노드의 주소를 인자로 받는다..			
Return Value			AvlNode*	자식노드의 주소를 반환한다.			
Local Variable	1	child	AvlNode*	부모노드의 오른쪽 자식노드의 주소를 저장하고 부모노드를 왼쪽 자식노드로 받는다.			

함수명	rotate_RL	Function 기술서	작성일	2018/05/28	Page 1/
Header	AVL_TREE.h		작성자	임수연	
함수 설명	부모노드를 인자로 받아 오른쪽 자식노드를 LL회전시킨 뒤 RR회전을 해준다.				

Category	No.	Name	Data type	Description
Parameter	1	parent	AvlNode*	부모노드의 주소를 인자로 받는다..
Return Value			AvlNode*	자식노드의 주소를 반환한다.
Local Variable	1	child	AvlNode*	부모노드의 오른쪽 자식노드의 주소를 저장하고 LL회전한다.

함수명	rotate_LR			Function 기술서	작성일	2018/05/28	Page 1/
Header	AVL_TREE.h				작성자	임수연	
함수 설명	부모노드를 인자로 받아 오른쪽 자식노드를 RR회전시킨 뒤 LL회전을 해준다.						
Category	No.	Name	Data type	Description			
Parameter	1	parent	AvlNode*	부모노드의 주소를 인자로 받는다..			
Return Value			AvlNode*	자식노드의 주소를 반환한다.			
Local Variable	1	child	AvlNode*	부모노드의 오른쪽 자식노드의 주소를 저장하고 RR회전한다.			

함수명	get_height		Function 기술서		작성일	2018/05/28	Page 1/
Header	AVL_TREE.h				작성자	임수연	
함수 설명	순환 호출을 이용하여 왼쪽 서브 트리과 오른쪽 서브 트리에 대하여 각각 높이를 구한다음 더 큰 값에 1을 더해서 트리의 높이를 구해준다.						
Category	No.	Name	Data type	Description			
Parameter	1	node	AvlNode*	높이를 잴 노드의 주소를 인자로 받는다.			
Return Value			int	트리의 높이를 반환한다.			
Local Variable	1	height	int	왼쪽 서브트리의 높이와 오른쪽 서브트리의 높이 중 큰 값에 1을 더한 값을 저장한다.			

함수명	get_balance			Function 기술서	작성일	2018/05/28	Page 1/
Header	AVL_TREE.h				작성자	임수연	
함수 설명	노드 주소를 인자로 받아 노드의 균형인수를 재 준다.						
Category	No.	Name	Data type	Description			
Parameter	1	node	AvlNode*	균형인수를 잴 노드의 주소를 인자로 받는다.			
Return Value			int	트리의 균형인수를 반환한다.			


함수명	blance_tree			Function 기술서	작성일	2018/05/28	Page 1/
Header	AVL_TREE.h				작성자	임수연	
함수 설명	양쪽 서브 트리의 높이 차이를 구한 다음, 어떤 회전이 필요한지 결정한다.						
Category	No.	Name	Data type	Description			
Parameter	1	node	AvlNode**	균형을 맞추출 노드를 가리키는 포인터의 주소를 인자로 받는다			
Return Value			AvlNode*	균형이 맞춰진 노드를 반환한다.			
Local Variable	1	height_diff	AvlNode*	노드의 균형인수를 저장한다.			

함수명	avl_add		Function 기술서		작성일	2018/05/28	Page
-----	---------	--	--------------	--	-----	------------	------

				8	1/4
Header	AVL_TREE.h			작성자	임수연
함수 설명	key에 대해 순환호출을 반복함으로써 트리에 삽입 한 후 균형화 함수를 호출한다.				
Category	No.	Name	Data type	Description	
Parameter	1	root	AvlNode **	루트를 가리키는 포인터의 주소를 인자로 받는다.	
	2	key	int	삽입할 data를 인자로 받는다.	
Return Value			AvlNode *	루트 노드의 주소를 반환한다.	

함수명	Avl_search			Function 기술서	작성일	2018/05/28	Page 1/4
Header	AVL_TREE.h				작성자	임수연	
함수 설명	key에 대해 순환호출을 반복함으로써 탐색해준다.						
Category	No.	Name	Data type	Description			
Parameter	1	node	AvlNode *	탐색할 노드의 주소를 인자로 받는다.			
	2	key	int	탐색할 data를 인자로 받는다.			
Return Value			AvlNode *	찾은 노드의 주소를 반환한다.			

4) 실행 결과


C:\Windows\system32\cmd.exe

7->9->10->11->계속하려면 아무 키나 누르십시오 . . .

11값을 찾기까지 탐색한 노드들의 data값들을 볼 수 있다.

5. 일정관리

- 5/11: 간단한 내용 설명 및 역할분담과 일정 정리(미팅).
- 5/16: 각자 조사한 내용 공유.
- 5/18: 취합한 내용 제출.
- 5/22: 구현 내용 공유.(제현, 주원)
- 5/25: 구현 내용 공유(상희), 1차 보고서 공유
- 5/28: 구현 내용 공유(수연)
- 5/29: 2차 보고서 공유.
- 5/31: 프로젝트 발표(제현)
- 6/8: 보고서 제출(상희)

6. 역할분담

- 박상희: 미로 찾기/탐색 알고리즘 조사 및 취합하여 문서 작성, A* 알고리즘으로 미로 탐색 구현.
- 김제현: 미로 찾기/탐색 알고리즘 조사, DFS 등의 알고리즘으로 미로 탐색 구현. 발표자료 준비 및 발표.
추가로 다익스트라 알고리즘 및 A* 알고리즘 구현.
- 정주원: 미로 찾기/ 탐색 알고리즘 조사, BFS 알고리즘으로 미로 탐색 구현.
- 임수연: 미로 찾기/탐색 알고리즘 조사, AVL 트리 구현.