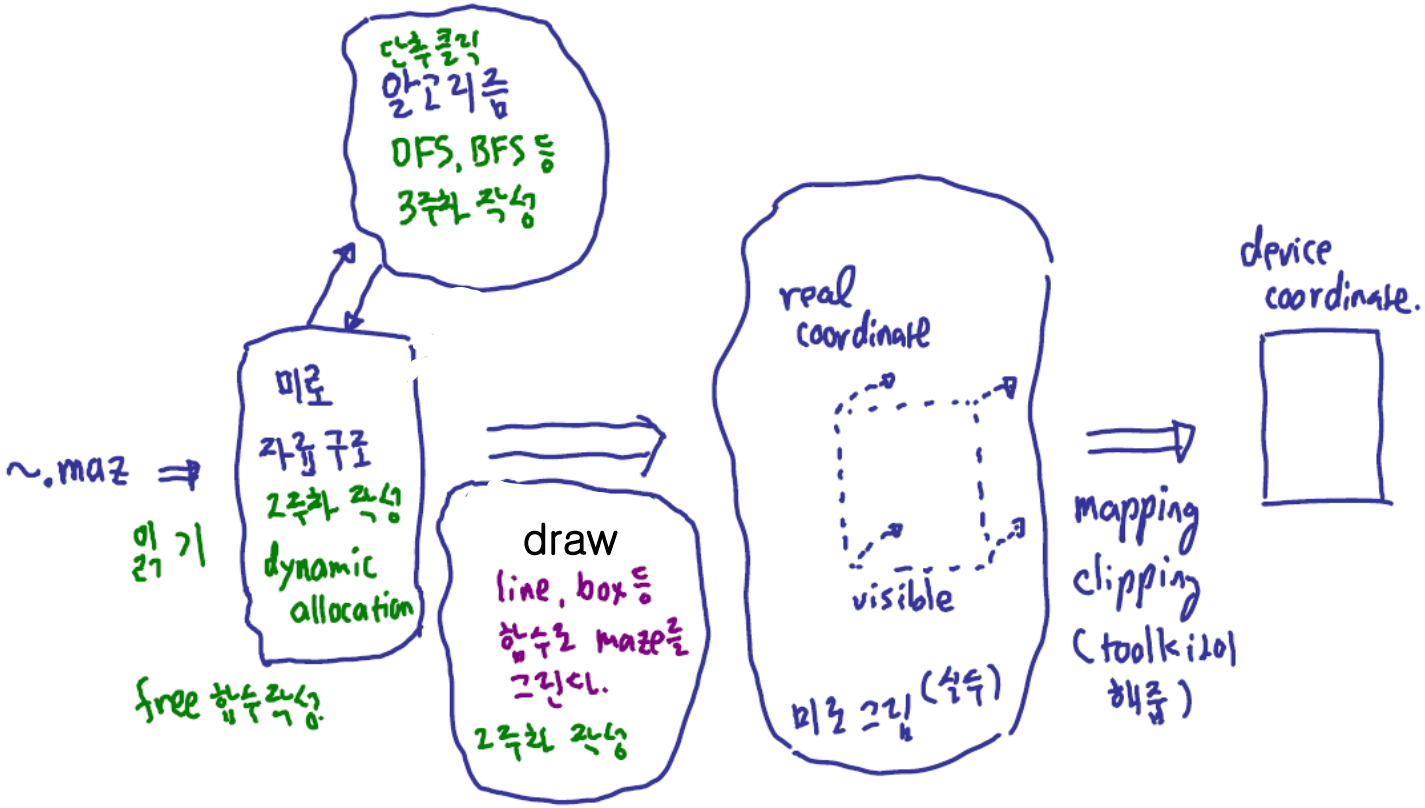
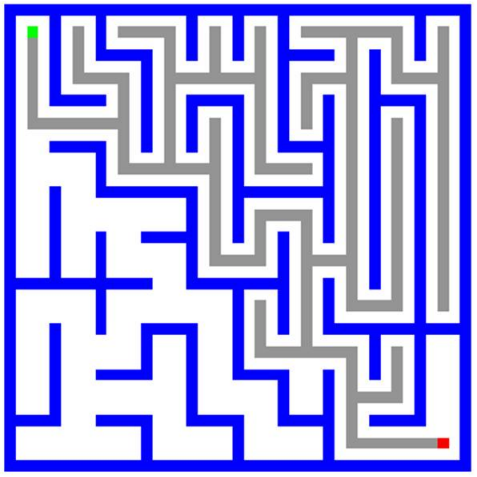
PRJ-2 미로 (Maze) 3주차

- 프로젝트 정리
  - 1주차 : 미로를 텍스트 파일로 만들어 출력하는 프로그램 작성.
  - 2주차 : 미로를 Openframeworks를 이용하여 그리는 프로그램 작성.
  - 3주차 : 미로에서 길 찾는 프로그램 작성.

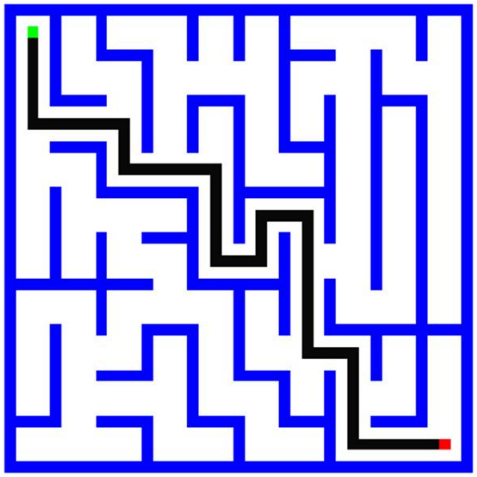


PRJ-2 미로 (Maze) 3주차

- DFS를 이용한 미로 길 찾기 문제
  - DFS를 이용한 미로 길 찾기 문제는 2주차까지 설계한 미로의 출발점과 도착점을 연결하는 가장 짧은 경로를 탐색해 출력하는 문제이다. NxM 미로의 출발점과 도착점은 각각 1행1열과 M행N열의 방으로 설정한다.
  - 실험에서는 [도구]-[DFS] 메뉴를 선택하거나 DFS 버튼을 선택하면 아래 그림의 (a)에 나타난 탐색했던 경로와 (b)에 나타난 탈출경로를 모두 표시한다. 두 경로는 서로 구분이 가능하도록 색을 달리해서 표시하거나 hatch등을 이용하도록 하자.



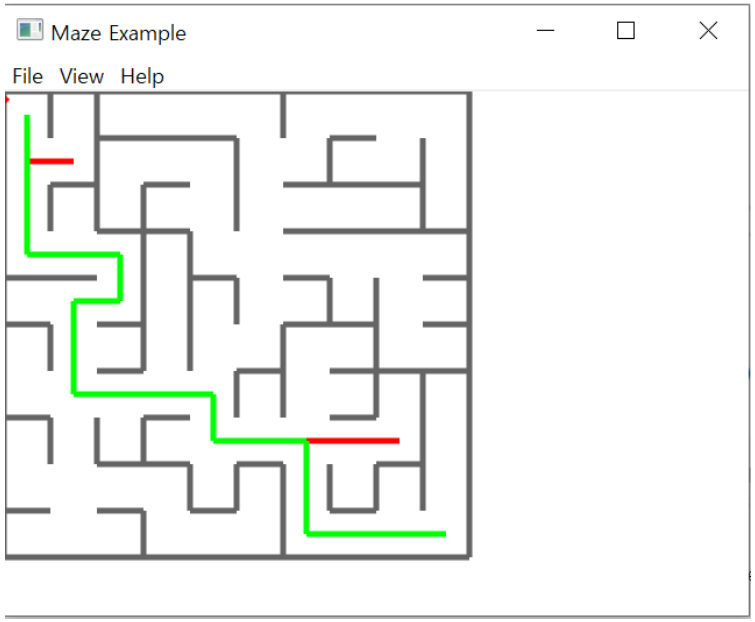
(a) 탐색 중 방문한 모든 경로



(b) 출발점과 도착점을 잇는 최단 경로

PRJ-2 미로 (Maze) 3주차

- 입력
  - 미로 텍스트 파일(~.maz)
- 출력
  - 미로 텍스트 파일(~.maz)을 읽어 들이면 아래 그림과 같이 화면에 미로를 출력한다. [View]-[Show DFS] 메뉴를 선택하면 DFS 탐색을 수행한다. 아래 그림과 같이 출발점과 도착점을 연결하는 탈출경로와 경로 탐색 수행 중 방문한 모든 길을 서로 구분이 갈 수 있도록 출력한다.



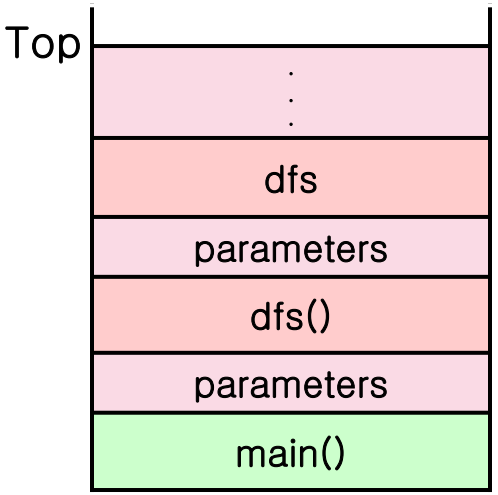
PRJ-2 미로 (Maze) 3주차

- 문제해결방법
  - 미로 길찾기 문제 (1) : DFS

일반적으로 알려진 DFS 알고리즘은 아래 (a)와 같이 재귀적 형태이다. 재귀함수의 경우 함수가 자기 자신을 호출할 때 parameter를 넘겨주기 위해 (b)와 같이 컴퓨터 내부의 stack memory를 사용한다. 만약 매우 큰 사이즈의 미로가 입력될 경우 DFS과정에서 stack overflow가 발생할 가능성이 있으므로 반복(iterative)함수의 형태로 구현할 필요가 있다.

```
void dfs (int v)
{/* depth first search of a graph beginning at v*/
      nodePointer w;
      visited[v] = TRUE;
      printf("%5d", v);
      for (w = graph[v]; w; w = w->link)
          if(!visited[w->vertex])
              dfs(w->vertex);
}
```

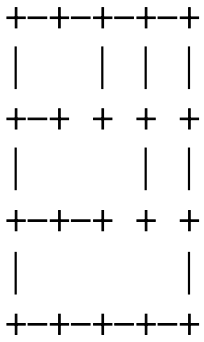
(a) 재귀적 형태의 DFS(Depth First Search)

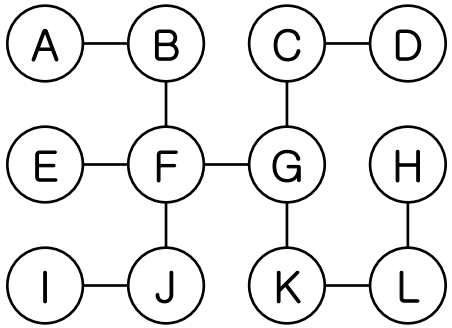


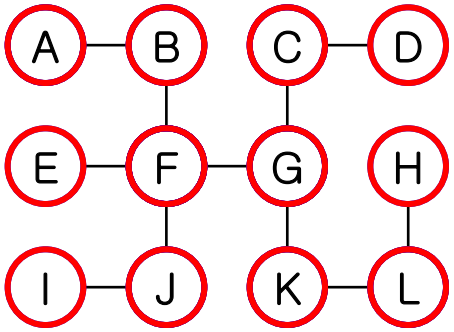
(b) Stack memory

PRJ-2 미로 (Maze) 3주차

- Recursive DFS

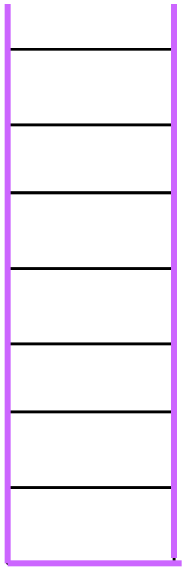







```
DFS(v) {
  mark v as visited;
  printf( v ); // do somthing
  for (each unvisited vertex  u  adjacent  to  v) {
      mark edge ( v – u ); // crossing edge
      DFS(u);
  }
}
```

DFS(A)
DFS(B)
DFS(F)
DFS(G)
DFS(C)
DFS(D)
DFS(K)
DFS(L)
DFS(H)
DFS(E)
DFS(J)
DFS(I)



system stack
(ret addr 제외)

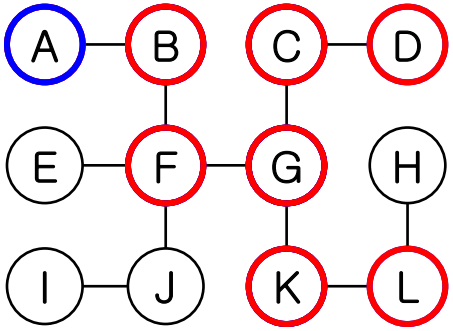
PRINT(visiting order)

A
B
F
G
C
D
K
L
H
E
J
I

PRJ-2 미로 (Maze) 3주차

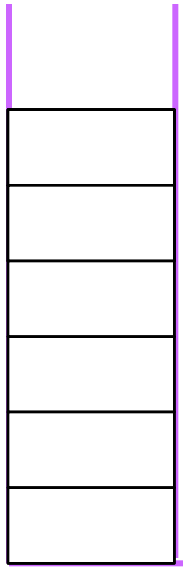
- Recursive DFS with TARGET

```
int DFS(v) {
  mark v as visited;
  if ( v == target ) {
      printf( v );
      return  TARGET_MEET;
  }
  for (each unvisited vertex  u  adjacent  to  v) {
       if ( DFS(u) == TARGET_MEET ) {
            mark edge ( v – u );
            printf( v );
            return TARGET_MEET;
       }
  }
   return  STILL_FINDING;
}
```



PRINT
(backward)

DFS(A)
DFS(B)
DFS(F)
DFS(G)
DFS(C)
DFS(D)
DFS(K)
DFS(L)



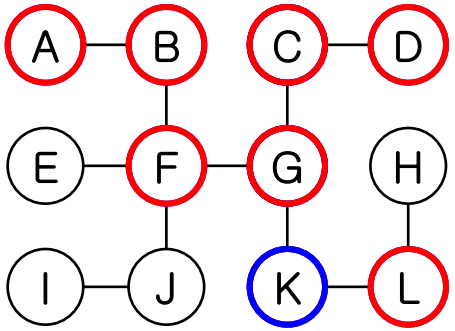
system stack
(ret addr 제외)

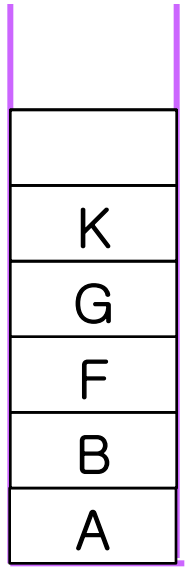
L
K
G
F
B
A

PRJ-2 미로 (Maze) 3주차

- Iterative DFS

```
void  DFS(v) {
   init a stack S;
   S.push ( v );
   mark v as visited;  // may print v
   while ( S != empty ) {
        if ( S.top has an unvisited adjacent node ) {
           u = an unvisited, adjacent node to S.top;
           S.push(u); // may print v, and mark (v,u)
           mark u as visited;
        }
        else {
           S.pop ( );
        }
   }
}
```





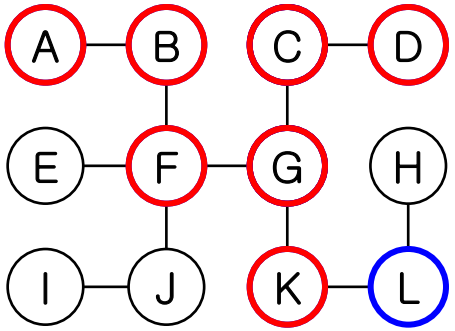
PRINT
(backward)

A
B
F
G
C
D
K
L

PRJ-2 미로 (Maze) 3주차

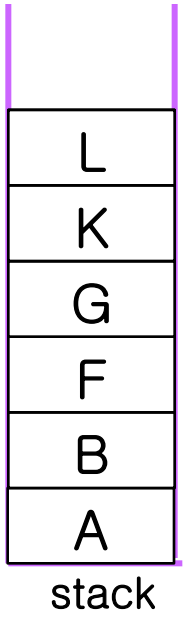
- Iterative DFS with Target

```
void   DFS(v) {
   init a stack S;
   S.push ( v );
   mark v as visited;  // may print v
   while ( S != empty ) {
        if ( S.top == target ) return;
        if ( S.top has an unvisited adjacent node ) {
           u = an unvisited, adjacent node to S.top;
           S.push(u); // may print u, and mark (v,u)
           mark u as visited;
        }
        else {
           S.pop ( );
        }
   }
}
```



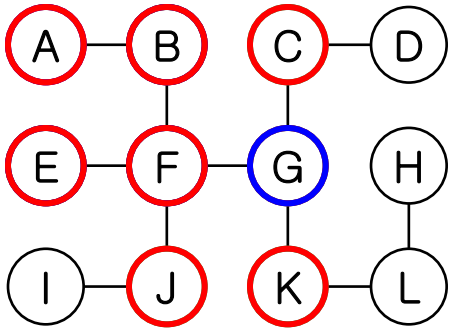
PRINT
(backward)

A
B
F
G
C
D
K
L



PRJ-2 미로 (Maze) 3주차

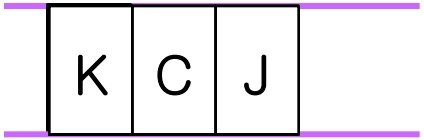
- BFS (Breath First Search)




```
Bfs(v) {
    Init  a  queue  Q;

    Q.enqueue(v);
    mark v as visited;
    while ( Q != EMPTY) {
        w = Q.dequeue();   // may print w
        for ( each unvisited node u adjacent to w ) {
            mark u as visited  //  set u.parent = w
            //  later  u.parent  will  be  used  to  find  a
path
            Q.enqueue(u);
        }
    }
}
```

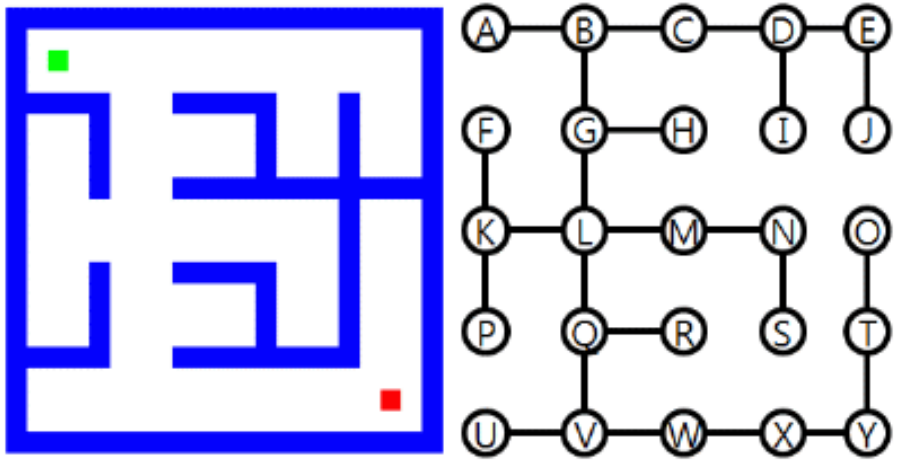
queue



PRJ-2 미로 (Maze) 3주차

- 문제해결방법

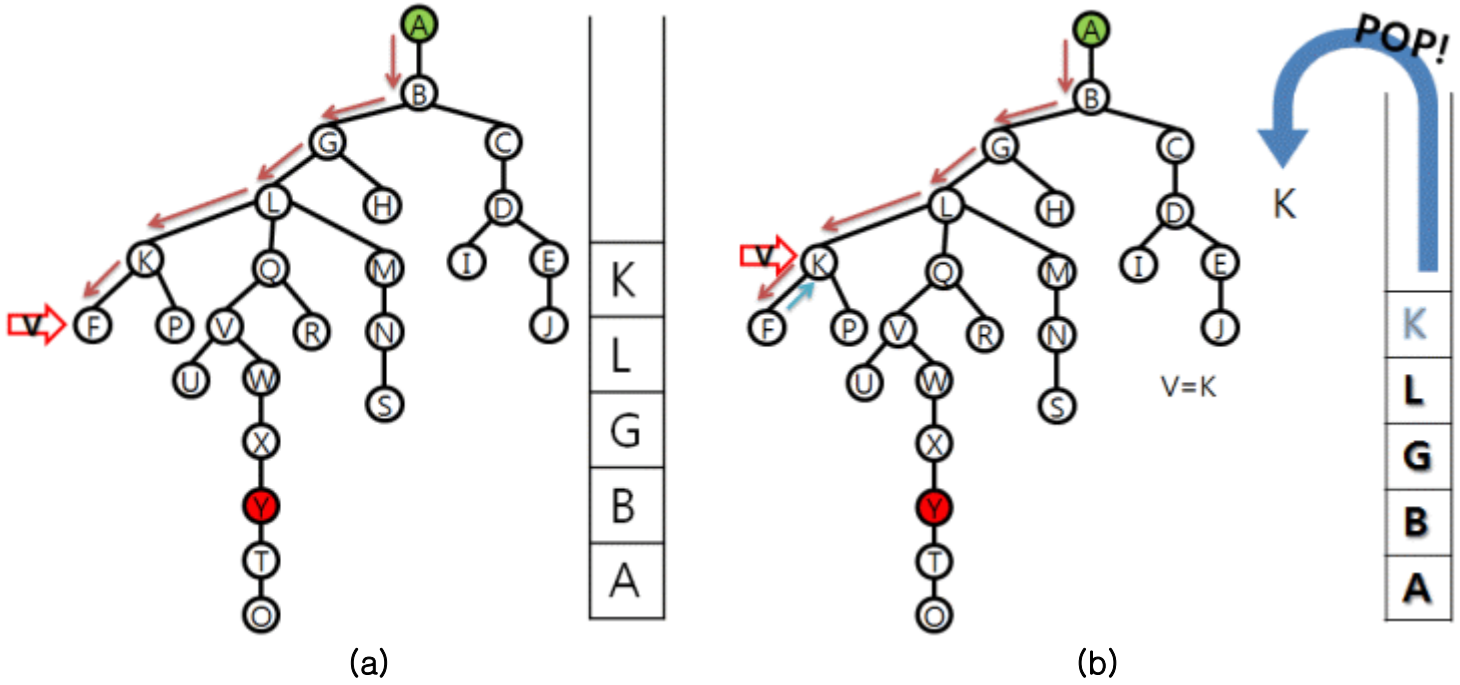
아래 그림처럼 미로는 그래프(node A를 root로 갖는 spanning tree)로 표현 가능하다. DFS 알고리즘을 반복함수 형태로 구현하기 위해 더 이상 방문할 수 있는 자식 node가 존재하지 않는 node에서 이전 node로 되돌아가기 위하여 현재까지 거쳐 왔던 node들을 저장할 stack이 필요하다.



PRJ-2 미로 (Maze) 3주차

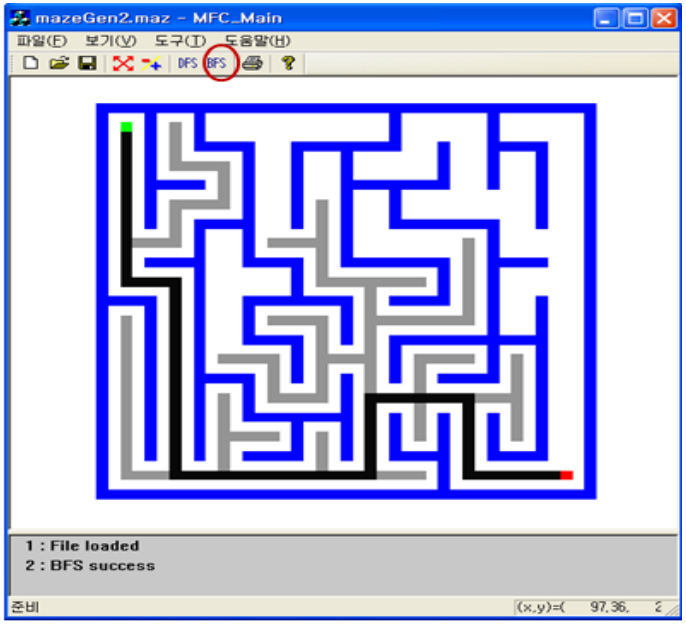
문제해결방법

node A부터 시작하여 그래프로 변환된 미로를 DFS 탐색방법으로 계속 탐색하면서 출구인 node Y를 찾을 때, 아래그림 (a)의 F처럼 방문한 적이 없었던 자식 node가 더 이상 존재하지 않는 vertex에 도달하게 되면 (b)처럼 stack에 저장해둔 node를 pop시켜서 이전 node로 되돌아 갈 수 있을 것이다. 자신이 구현한 미로의 자료구조를 어떻게 stack에 저장할 것인지 생각해보자.



PRJ-2 미로 (Maze) 3주차 – 숙제

- 미로 길찾기 문제 (2) : BFS
  - 3주차 실습에서 완성한 프로그램을 바탕으로 BFS버튼을 누르면 BFS 방법을 통한 탈출 경로 및 탈출과정에서 방문했던 모든 경로를 표시하는 프로그램을 작성한다.
  - 입출력 형식은 실험 시간에 해결한 완전 미로 생성 문제와 동일하다.
    - 입력 : 미로 텍스트 파일(~.maz)
    - 출력 : BFS 탐색을 수행한다. 아래 그림과 같이 출발점과 도착점을 연결하는 탈출경로와 경로 탐색 수행 중 방문한 모든 길을 서로 구분이 갈 수 있도록 출력한다.



PRJ-2 미로 (Maze) 3주차 -보고서 작성

- 예비 보고서
  - DFS와 BFS의 시간 복잡도를 계산하고 그 과정을 설명한다.
  - 자신이 구현한 자료구조 상에서 DFS와 BFS 방법으로 실제 경로를 어떻게 찾는지 설명한다. 특히 DFS 알고리즘을 iterative한 방법으로 구현하기 위한 방법을 생각해보고 제시한다.
  - 작성한 예비보고서는 실험 시작 전 제출하여야 한다.

- 결과 보고서
  - 실습 및 숙제로 작성한 프로그램의 알고리즘과 자료구조를 요약하여 기술한다. 완성한 알고리즘의 시간 및 공간 복잡도를 보이고 실험 전에 생각한 방법과 어떻게 다른지 아울러 기술한다.
  - 자신이 설계한 프로그램을 실행하여 보고 DFS, BFS 알고리즘을 서로 비교한다. 각각의 알고리즘은 어떤 장단점을 가지고 있는지, 자신의 자료구조에는 어떤 알고리즘이 더 적합한지 등에 대해 관찰하고 설명한다.
  - 작성한 숙제 프로그램을 공지한 제출요령에 의거하여 기한 내 제출하시오. 요청이 있을 경우 실험시간에 작성한 프로그램도 아울러 제출한다.