

2020년도 2학기 컴퓨터공학설계및실험I  
14주차 Maze Project 결과보고서

20160641 조보현

## 1. 실습목적

본 실험에서는 2주차의 실습에서 만든 OpenFrameWork 응용 프로그램을 수정하여 DFS 방법 (실습)과 BFS 방법 (과제)으로 미로를 탐색하여 그 결과를 화면에 표시하는 프로그램을 작성한다.

## 2. 실습 구현 내용

### 1) void DFS()

DFS알고리즘을 통해 미로의 최적 경로를 계산하는 함수. Iterative한 DFS구현을 위해 구현한 스택을 이용한다. 먼저 스택을 초기화한 후 스택에 head노드를 넣는다. 이후 while문에 들어가 목표노드를 만날 때까지 다음과 같은 과정을 반복한다. 스택의 top원소를 뽑아내 방문하지 않은 자식이 있다면 방문하고, 방문 정보를 저장한다. 방문한 자식을 다시 스택에 넣는다. 방문하지 않은 자식이 없다면 스택을 pop 한다. 이 과정을 반복하면 목표노드에 도달하고 while문이 종료된다.

### 2) void resetTree()

트리의 방문정보를 초기화하는 함수. 미로의 경로를 추적하며 변경된 트리의 방문정보를 초기화시킨다. BFS알고리즘을 통해 모든 노드를 방문하며 visited 멤버를 false로 초기화시킨다. 이는 DFS알고리즘을 통해 미로의 최적경로를 계산한 후 바로 BFS알고리즘을 통해 미로의 최적경로를 계산할 수 있게 하기 위해 구현한다.

### 3) void draw()

계산한 미로의 최적경로 및 방문 경로를 모두 화면상에 그리는 함수. BFS알고리즘을 통해 모든 노드를 조사해 visited가 true인 노드(방문한 노드)의 경로는 초록색으로 그리고, path가 true인 노드(최적 경로 노드)의 경로는 빨간색으로 그린다.

### 4) DFS 알고리즘의 동작 과정

```
void DFS()
{
    initialize stack S;
    head->visited = true;
    S.push(head);
    while(!S.isEmpty())
    {
```

```

        if(S->Top() == target)    // 목표 노드에 도달한 경우
            break;
        if(S->Top()->child1 != NULL && S->Top()->child1->visited == false)

        {
            // child1 이 방문하지 않은 노드일 경우
            S->Top()->child1->visited = true;
            S.push(S->Top()->child1);
            continue;
        }
        else if(S->Top()->child2 != NULL && S->Top()->child2->visited ==
false)

        {
            // child2 이 방문하지 않은 노드일 경우
            S->Top()->child2->visited = true;
            S.push(S->Top()->child2);
            continue;
        }
        else if(S->Top()->child3 != NULL && S->Top()->child3->visited ==
false)

        {
            // child3 이 방문하지 않은 노드일 경우
            S->Top()->child3->visited = true;
            S.push(S->Top()->child3);
            continue;
        }
        else    // 방문하지 않은 노드가 없거나 자식이 없는 경우
            S->pop();
    }
    while(!S->isEmpty())    // 스택의 최적 경로 계산
    {
        S->pop()->path = true;
    }
}

```

##### 5) DFS 알고리즘의 시간 및 공간 복잡도

위와 같은 과정의 DFS 알고리즘은 미로의 높이를  $N$ , 가로 길이를  $M$ 이라 하면, 방의 개수는  $N \times M$ 개이다. 이 때 최악의 경우 모든 방을 탐색하고 목표타겟을 찾아서 종료될 수 있다. 따라서 시간복잡도는  $O(N \times M)$ 이다. 하지만 최악의 경우는 거의 발생하지 않을 확률이 높다. 공간복잡도는 최악의 경우 모든 방을 탐색할 때 스택에  $N \times M$  개의 방이 쌓이게 되므로 마찬가지로  $O(N \times M)$ 이다.

#### 6) 스택 및 큐

스택은 가장 윗 원소를 가리키는 `TreeNode* top`을 멤버변수로 갖고, `top`을 초기화하는 생성자, 원소를 스택에 넣는 `push()`, 가장 윗 원소를 반환하는 `Top()`, 가장 윗 원소를 반환하고 스택에서 제거하는 `pop()`, 스택이 비었는지를 판단해주는 `isEmpty()` 메소드들로 이루어져있다.

큐는 가장 앞 원소와 뒤 원소를 가리키는 `TreeNode* front, rear`를 멤버 변수로 갖고, `front`와 `rear`를 초기화하는 생성자, 큐에 원소를 넣는 `enqueue()`, 큐의 가장 앞부분 원소를 반환하고 제거하는 `dequeue()`, 큐가 비었는지 판단해주는 `isEmpty()` 메소드들로 이루어져있다.

### 3. 실습 환경

SAMSUNG Notebook9

CPU : Intel Core i5-2450M

RAM : 8GB

GPU : Intel 내장 그래픽

OS : Windows 7 Home Premium K

### 4. 과제

#### 1) void BFS()

BFS 알고리즘으로 미로의 최적 경로를 찾는 함수. 먼저 큐를 초기화 시킨 후, `head`를 큐에 넣는다. 이후 `while`문에 들어가 목표노드를 만날때까지 다음의 과정을 반복한다. 큐에서 원소를 뽑아 낸후 방문정보를 기록한다. 이후 노드의 인접노드를 모두 큐에 넣는다. 이 과정을 반복하면 목표노드를 만나고 최적경로를 찾게 된다.

#### 2) BFS 알고리즘의 동작 과정

```
void BFS()
{
    initialize queue Q;
    Q.enqueue(head);
    while(!Q.isEmpty())
    {
        TreeNode* ptr = Q.dequeue(); // 큐에서 원소 반환
        ptr->visited = true; // 방문정보 기록
        if(ptr == target) // 목표노드 방문시 break
            break
        for(ptr에 연결된 자식 노드들 모두 조사)
```

```

    {
        u = ptr의 자식노드
        if( u.visited == false)    // 방문하지 않은 노드라면
            Q.enqueue(u);    / 큐에 삽입
    }
}
}

```

### 3) BFS 알고리즘의 시간 및 공간 복잡도

BFS 알고리즘은 미로의 가로 길이가  $N$ , 세로 길이가  $M$ 이라고 했을 때 방(노드)의 개수는  $N \times M$ 이다. 이 때 최악의 경우 모든 방을 탐색해야 할 수 있다. 따라서 시간 복잡도는  $O(N \times M)$ 이다. 공간복잡도는 마찬가지로 큐에 모든 노드를 삽입해야 할 수도 있으므로 공간 복잡도는  $O(N \times M)$ 이다.

## 5. 실습 결과 및 분석

DFS 방식과 BFS 방식을 7가지 input 파일을 통해 비교해본다.

### 1) maze0.maz

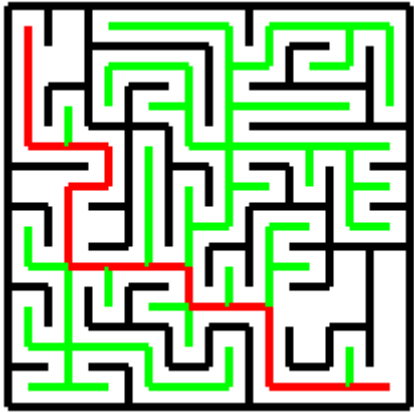
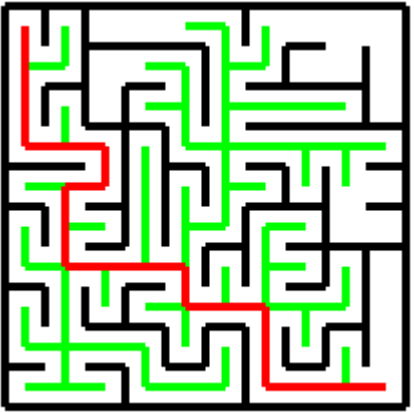
	
DFS 알고리즘	BFS 알고리즘

표 1 DFS 알고리즘과 BFS 알고리즘 성능 비교(1)

미로의 크기가 비교적 작을 때는 둘의 성능이 비슷한 것처럼 보인다. 우선 두 알고리즘 모두 최적경로와 방문한 모든 경로를 적절히 출력하는 것을 확인 할 수 있다. BFS 방식에서 모든

방을 방문하지 않는 것은 큐에 원소가 남아있더라도 목표노드를 만나면 종료되기 때문이다.

2) maze1.maz

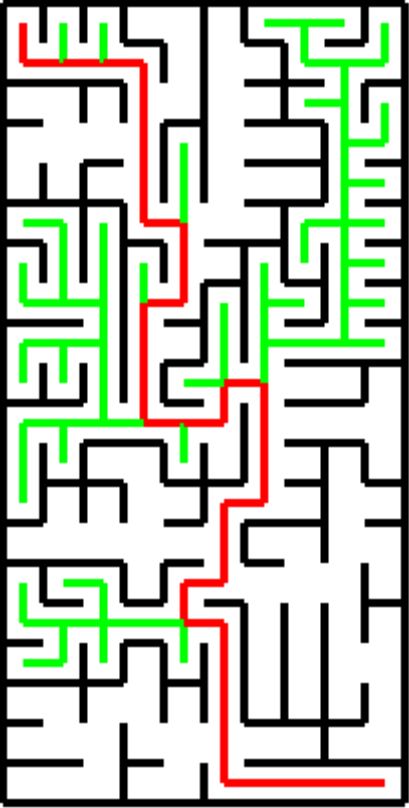
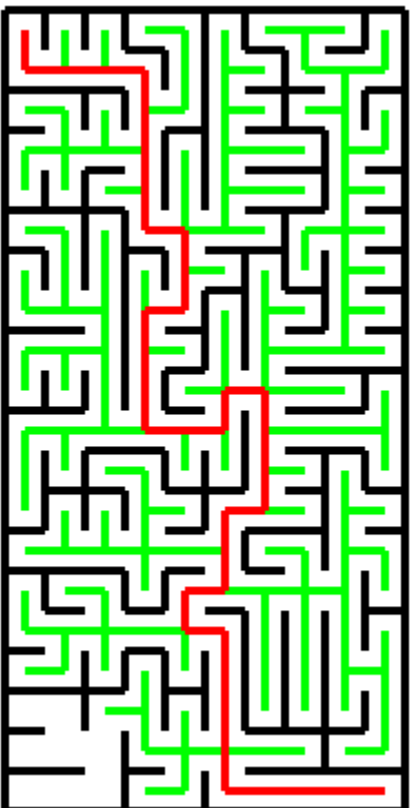
	
DFS 알고리즘	BFS 알고리즘

표 2 DFS 알고리즘과 BFS 알고리즘 성능 비교(2)

미로의 크기가 커지자 성능의 차이가 보이기 시작한다. DFS알고리즘이 더 빨리 최적경로에 도달하는 것을 확인 할 수 있다.

3) maze2.maz

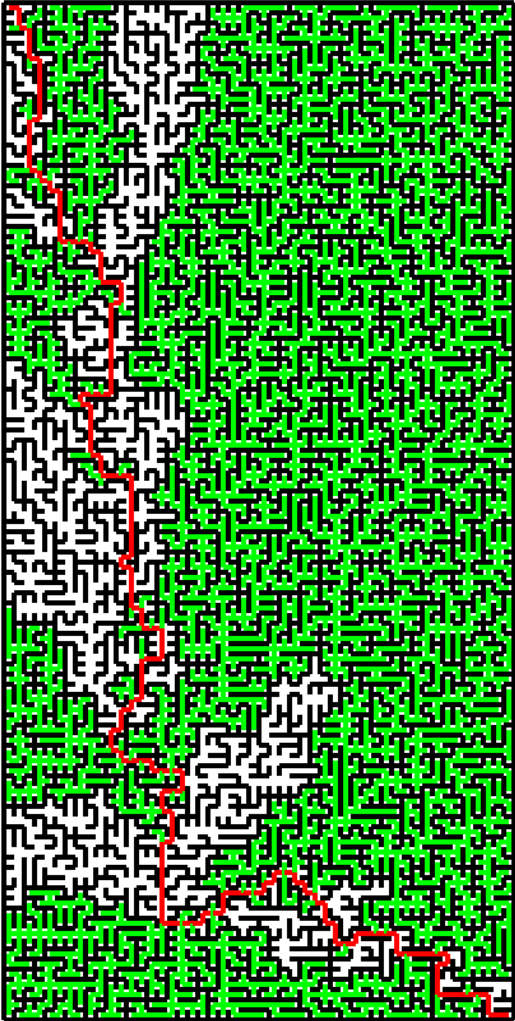

	
DFS 알고리즘	BFS 알고리즘

표 3 DFS 알고리즘과 BFS 알고리즘 성능 비교(3)

이 경우에는 둘의 성능이 비슷한 것처럼 보인다.

4) maze3.maz

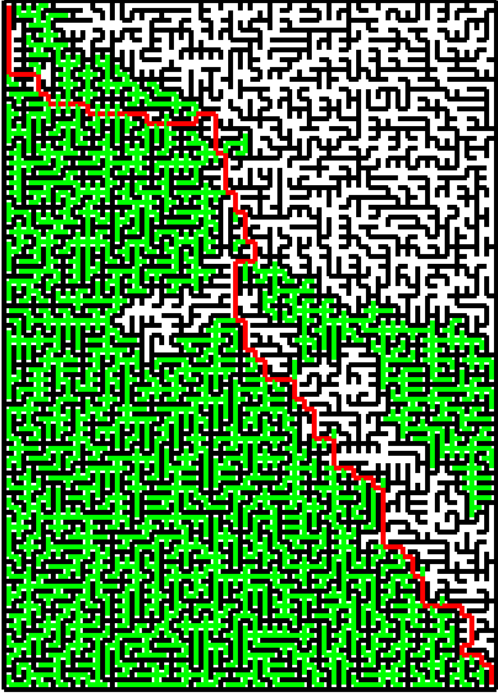
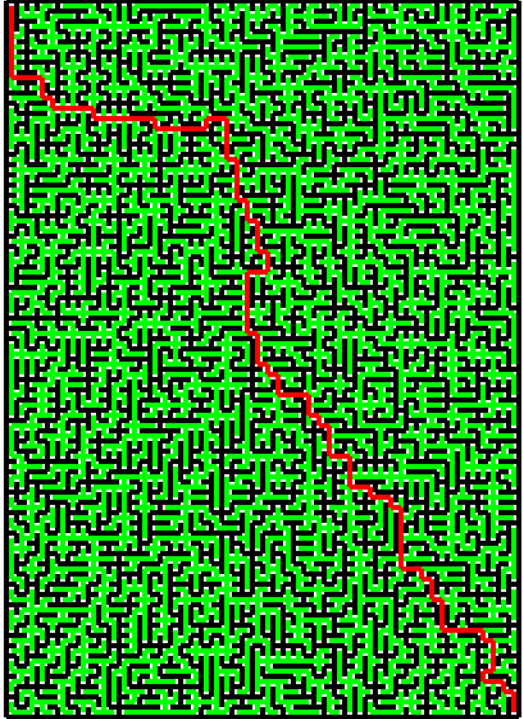
	
DFS 알고리즘	BFS 알고리즘

표 4 DFS 알고리즘과 BFS 알고리즘 성능 비교(4)

이 경우에는 DFS 알고리즘의 성능이 훨씬 좋은 것을 확인 할 수 있다. BFS 알고리즘은 거의 모든 방을 방문하는 것처럼 보인다. DFS 알고리즘은 비교적 적은 방을 방문한 뒤 최적경로를 찾아 탐색이 종료됐다.

5) maze4.maz

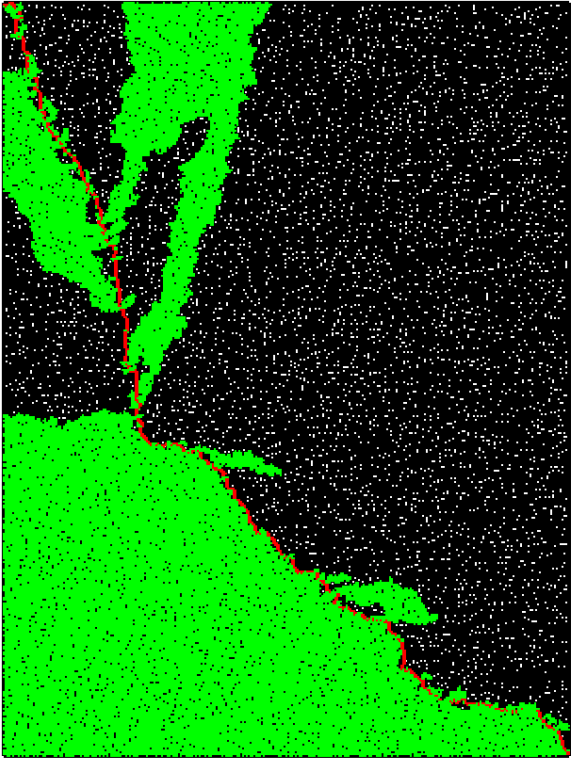
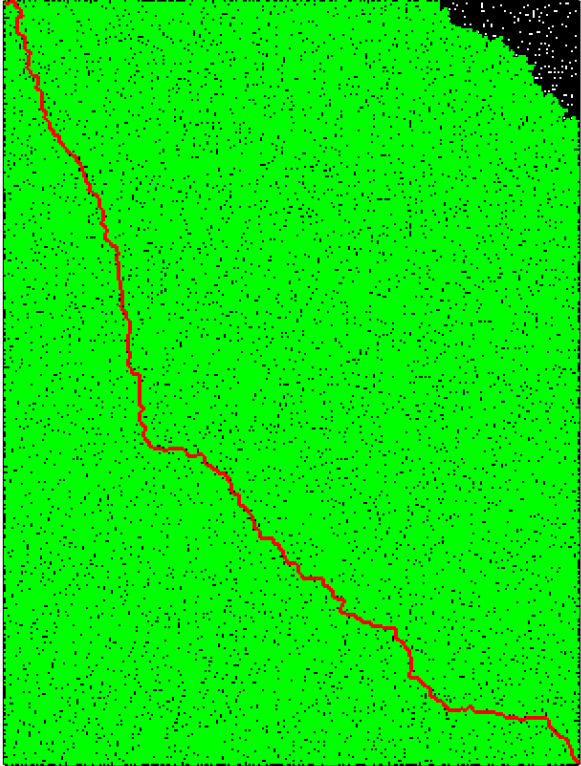
	
DFS 알고리즘	BFS 알고리즘

표 5 DFS 알고리즘과 BFS 알고리즘 성능 비교(5)

미로의 크기가 매우 클 때 성능의 차이가 확실히 보인다. DFS 알고리즘이 BFS 알고리즘보다 훨씬 적은 수의 방을 방문하고 최적경로를 찾게 된다. 그러나 BFS 알고리즘은 매우 많은 수의 방을 방문하고 최적경로를 찾게 된다.



6) maze5.maz

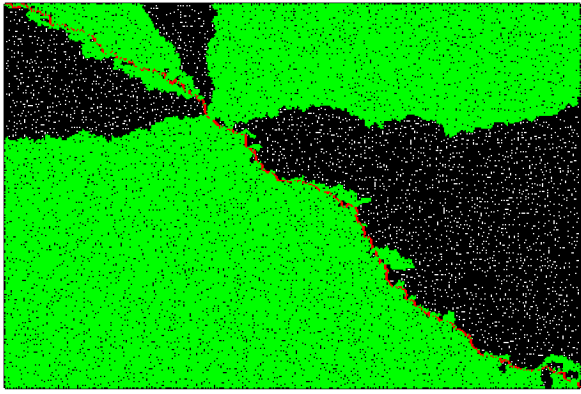
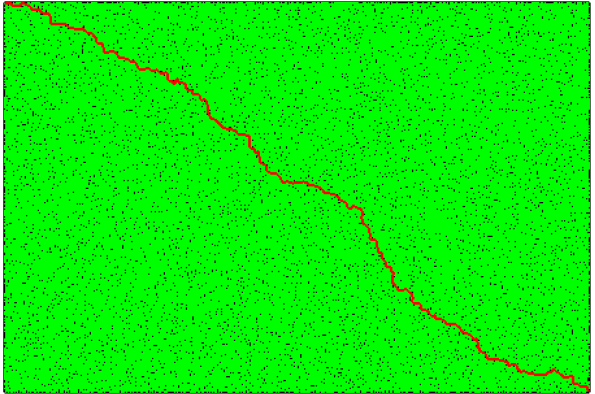
	
DFS 알고리즘	BFS 알고리즘

표 6 DFS 알고리즘과 BFS 알고리즘 성능 비교(6)

이 경우도 미로의 크기가 매우 큰 경우이다. 이때도 마찬가지로 DFS 알고리즘이 훨씬 좋은 성능을 보이는 것을 확인 할 수 있다. BFS 알고리즘은 거의 모든 방을 방문하는 것처럼 보인다.

7) maze6.maz

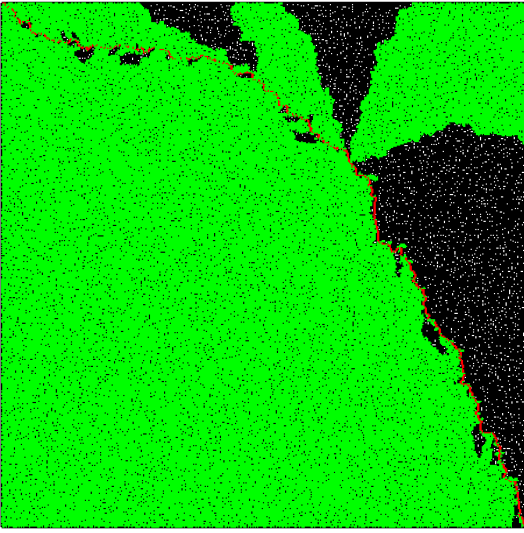
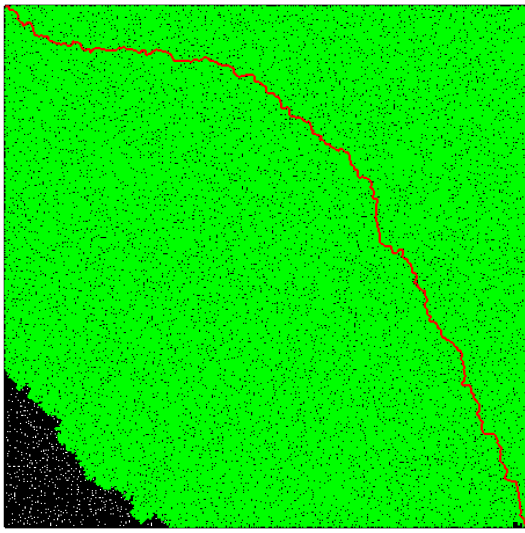
	
DFS 알고리즘	BFS 알고리즘

표 7 DFS 알고리즘과 BFS 알고리즘 성능 비교(7)

이 경우에도 DFS 알고리즘의 성능이 BFS 알고리즘보다 훨씬 좋은 것을 확인 할 수 있다.

위 결과들을 통해 DFS알고리즘의 성능이 BFS 알고리즘보다 좋음을 확인 할 수 있다. 이는 DFS와 BFS의 탐색방법이 다르기 때문이다. DFS는 더 이상 자식노드가 없을때까지 계속해서 경로를 추적한다. 이것은 한 경로의 끝까지 가보는 것을 의미한다. 완전미로는 한 노드에서 다른 노드까지 가는 길이 하나 뿐이므로, 길만 잘든다면 한번에 목표노드까지 갈 가능성도 있다. 반면 BFS 는 인접 노드를 차례차례 방문하는 방식이다. 즉, 모든 경로를 하나의 방씩 조금씩 조금씩 찾아나가는 식이다. 이는 불완전미로에서 최적경로를 찾을 때에는 DFS방식보다 더 좋은 경로를 찾을 수 있겠지만, 완전미로에서는 불필요한 탐색을 너무 많이 하게 될 가능성이 커진다. 따라서 완전미로에서의 탐색은 DFS방식이 우수하다는 것을 확인 할 수 있다.

## 6. 결론

이번 미로 프로젝트를 진행하며 DFS 알고리즘과 BFS 알고리즘에 대해 익숙해지고 직접 구현해보며 이에 대해 이해할 수 있었다. 특히 DFS와 BFS를 구현하면서 스택과 큐를 직접 구현해보며 자료구조 수업 때 배운 스택 및 큐 자료구조에 대한 이해도가 높아졌으며 익숙해질 수 있었다. 또한 미로를 구현할 때 트리 구조체를 이용해 트리 자료구조에 대한 이해도 높아졌으며 직접 구현해보며 이를 응용하는 능력도 가질 수 있었다.

완전미로에서 최적경로를 찾을 때는 BFS 알고리즘보다 DFS 알고리즘이 더 우수한 성능을 보여줄 가능성이 높다. 그러나 불완전 미로의 경우에는 DFS 알고리즘보다 BFS 알고리즘이 시간은 더 오래 걸릴지 몰라도 더 최적(비용이 적은)경로를 찾을 가능성이 높다.

BFS 알고리즘에서 방문한 모든 방의 경로는 최적경로에 가까운 것으로 보인다. 그 이유는 최적경로를 탐색하면서 그 주위의 방을 우선 탐색하기 때문이다.