2020년도 2학기 컴퓨터공학설계및실험I

14주차 결과 보고서

20170175 김태안

1. 실습 목적

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.

1. 실습 및 과제 구현 내용

DFS와 BFS 구현의 편의를 위해 다음의 자료구조와 멤버함수를 추가하였다.

|  |  |
| --- | --- |
| class Stack:public SLList {  public:  node\* pop();  void push(int i);  bool isEmpty();  }; | class Queue:public SLList {  public:  node\* pop();  void push(int i);  bool isEmpty();  }; |

Stack과 Queue는 연결리스트 SLList를 부모 클래스로 갖는 클래스로, Stack은 LIFO(Last In First Out), Queue는 FIFO(First In First Out) 형식의 연결 리스트이다. 각 멤버 함수는 다음의 기능을 가진다.

node\* pop();

연결 리스트 맨 앞의 값을 제거하고, 그 값을 반환한다.

void push(int i);

Stack은 연결 리스트의 맨 앞에, Queue는 연결 리스트의 맨 뒤에 새 원소 i를 삽입한다.

bool isEmpty();

연결 리스트에 원소가 존재하면 거짓을, 존재하지 않으면 거짓을 반환한다.

DFS와 BFS를 통한 미로 탐색을 구현하기 위해 수정/생성한 함수는 다음과 같다.

bool ofApp::readFile(const char\* fileName);

파일을 읽고 인접 리스트를 생성한 뒤, DFS 함수와 BFS 함수를 호출해 미로의 해답을 찾는다. DFS와 BFS에 사용되는 WIDTH\*HEIGHT 크기의 visited, parent 배열을 생성한다.

void ofApp::DFS(int i);

|  |
| --- |
| void DFS(v) { init a stack S;  S.push ( v );  mark v as visited;  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);  mark u as visited;  }  else {  S.pop( );  }  }  } |

함수 DFS의 수도코드

DFS를 통해 미로의 해답을 찾는다. 시작 지점을 Stack에 삽입한 뒤, 인접하고 방문하지 않은 칸이 있다면 Stack에 집어 넣고 방문했음을 표시한다. 새로 삽입한 칸에 대해 위 과정을 반복하고, 인접하고 방문하지 않은 칸이 없다면 Stack을 pop한다. 삽입한 칸이 목표 지점인 WIDTH \* HEIGHT면 함수를 종료하는데, 종료 시점에 Stack에 저장된 칸은 시작 지점부터 목표 지점까지의 경로를 담고 있다.

이때, DFS는 최악의 경우 모든 칸을 탐색한다. 즉. O(HEIGHT\*WIDTH)의 시간 복잡도를 지니며, 추가로 사용한 배열 visitedDFS와 최대 Stack의 크기의 합인 O(2\*HEIGHT\*WIDTH) 만큼의 공간 복잡도를 가진다.

void ofApp::BFS();

|  |
| --- |
| 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  Q.enqueue(u);  }  }  } |

함수 BFS의 수도 코드

BFS를 통해 미로를 탐색한다. 방문한 칸 i는 visitedBFS[i]에 표시한다. 칸과 인접한 칸을 Queue에 저장한 뒤, Queue의 맨 앞 원소부터 제거해가며 해당 원소와 인접하고 방문하지 않은 칸을 Queue에 집어넣는다. 이때, 인접하지 않고 방문하지 않은 칸 v의 부모인 원소 i를 parentBFS[i]에 저장해 경로를 역추적할 수 있게 한다. 목표 칸은 WIDTH\*HEIGHT-1 칸에 도달하면 Queue를 해제하고 함수를 종료한다.

BFS는 최악의 경우 모든 칸을 탐색하며 O(HEIGHT\*WIDTH)의 시간 복잡도를 가진다. 그리고, 추가로 사용한 visitedBFS와 parentBFS, Queue의 크기인 O(3\*HEIGHT\*WIDTH)의 공간 복잡도를 가진다.

void ofApp::keyPressed(int key);

d키에 DFS 경로 출력, b키에 BFS 경로 출력, e 키에 출력한 경로 지우기의 기능을 할당하였다.

void ofApp::draw();

DFS와 BFS 탐색 도중 방문한 칸은 회색으로, 찾은 해답 경로는 초록색으로 출력한다. visited 배열을 처음부터 읽으며 방문을 한 적이 있는 번호의 칸은 회색 사각형을 그려 채운다.

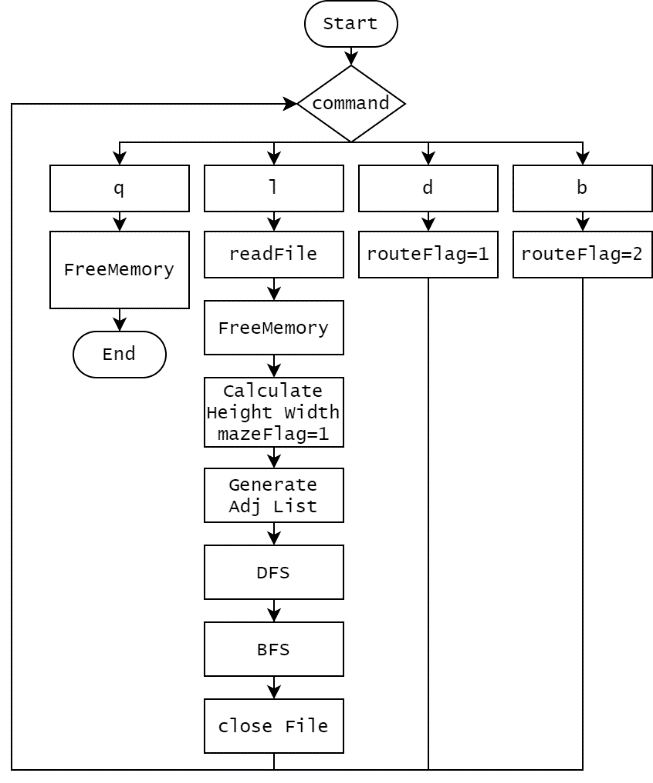
DFS의 경로의 경우, 경로를 저장한 Stack을 읽으며 그려 나간다. 현재 칸과 다음 칸의 차이가 WIDTH인지. -WIDTH인지, 1인지 -1인지를 통해 칸이 인접한 위치 (위, 아래, 왼쪽, 오른쪽)을 파악한 뒤, 두 칸을 잇는 직사각형을 그린다.

BFS의 경우, parentBFS를 역추적하며 경로를 그려 나간다. WIDTH\*HEIGHT-1에서 시작해 parentBFS에 저장된 parent를 찾는다. DFS때와 마찬가지로 child와 parent의 차이를 통해 인접한 위치를 파악한 뒤, 두 칸을 잇는 직사각형을 그린다. parent의 parent를 찾아가며 0에 도달할 때까지 탐색한다.

void ofApp::freeMemory();

기존의 변수들에 더해 DFS와 BFS에서 추가로 생성한 배열과 Stack, 변수를 해제/초기화한다.

구현한 전체 프로그램의 순서도는 다음과 같다.



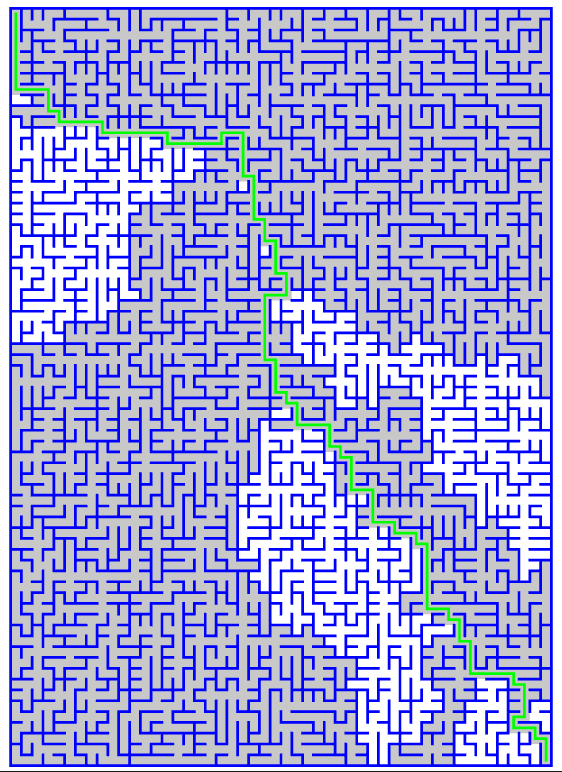
draw 함수는 매초 15번 호출되며 routeFlag와 mazeFlag에 따라 미로와 경로를 출력한다. FreeMemory와 GenAdjList, DFS, BFS 함수는 readFile 함수 안에서 순서대로 실행된다. 따라서 복잡도는 각 함수의 복잡도의 합으로, 시간 복잡도는 O(HEIGHT\*WIDTH)이고, 공간 복잡도는 O(HEIGHT\*WIDTH)이다.

1. 실습 환경

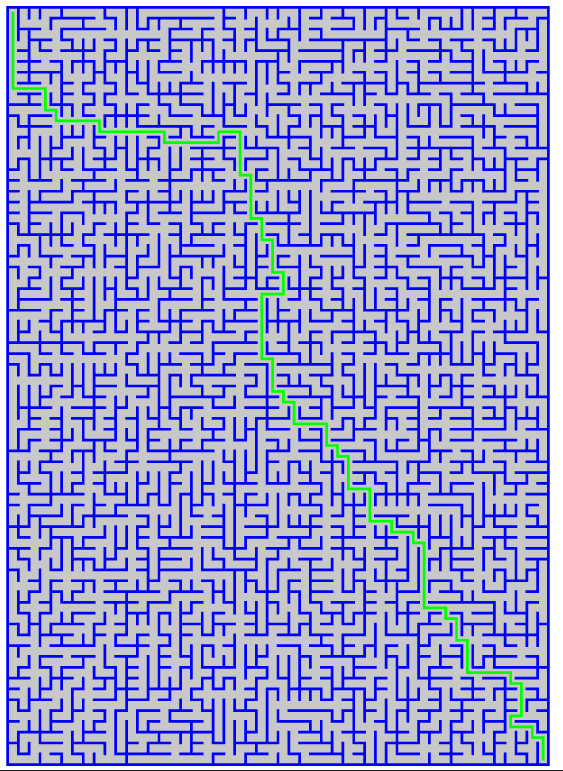
Microsoft Visual Studio Community 2019 버전 16.7.7

OpenFrameworks v0.11.0 vs2017 Release

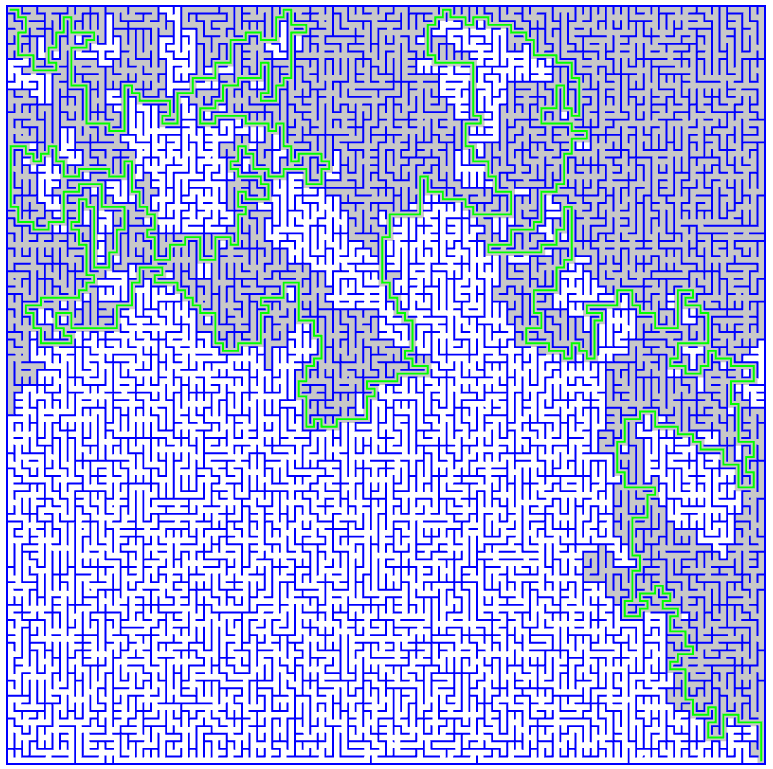
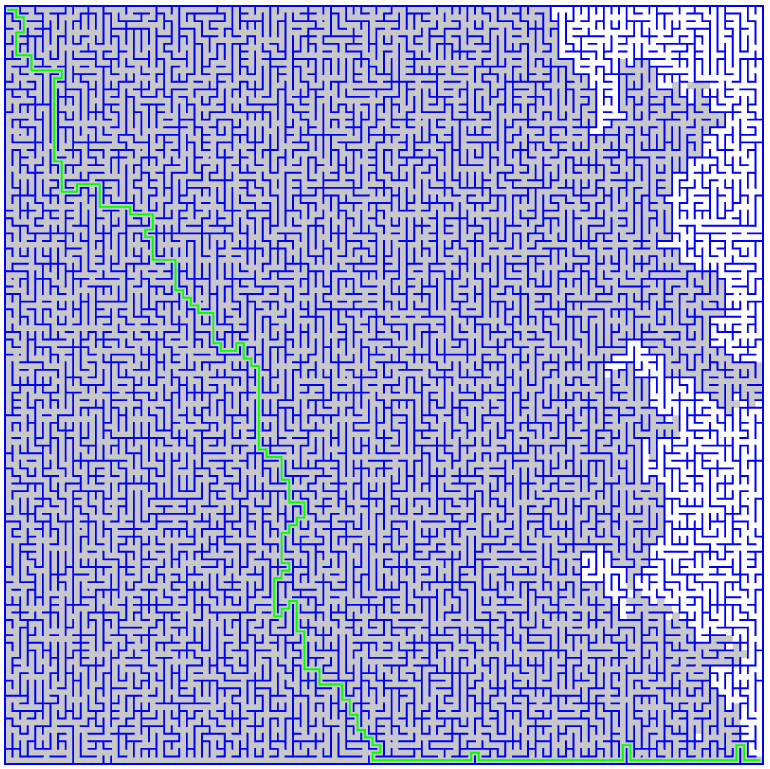
1. 실습 결과 및 분석



50\*70의 완전 미로를 불러와 DFS로 탐색한 결과는 위와 같다.



같은 미로를 BFS로 탐색한 결과는 다음과 같다. 완전 미로는 경로가 하나이기 때문에 같은 경로를 보인다. 이때, BFS는 모든 칸을 탐색한 반면, DFS는 일부 칸을 탐색하지 않은 것을 확인할 수 있다. DFS는 BFS에 비해 적은 칸을 탐색할 확률이 높으므로 더 효율적인 방법이라고 할 수 있다.

위 그림은 100\*100의 불완전 미로를 DFS와 BFS로 탐색한 결과이다. 불완전 미로의 경우, 경로가 여러 개가 나올 수 있다. 이때, BFS는 더 짧은 깊이를 탐색한 상태에서 찾은 해답을 먼저 찾기 때문에 DFS에 비해 거리가 짧다. DFS의 경우 더 적은 칸을 탐색해 경로를 찾을 수 있다.

1. 결론

이번 실습에서는 DFS와 BFS를 활용해 미로의 경로를 탐색한 뒤, OpenFrameWorks를 사용해 화면에 출력하는 프로그램을 제작하였다. DFS와 BFS 모두 시간 복잡도 O는 O(WIDTH \* HEIGHT)로 같다. 그러나 BFS의 탐색이 평균적으로 더 많은 칸을 탐색하며, BFS가 더 많은 공간을 필요로 한다. 따라서 완전 미로의 경우, DFS를 통한 탐색이 유리하다. 그러나 불완전 미로의 경우, BFS는 DFS보다 짧은 경로를 찾을 확률이 더 높다는 장점이 존재한다.