MFC 미로찾기 설계보고서

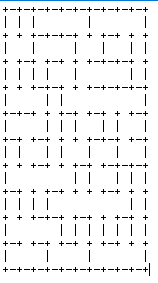
전공 : 컴퓨터공학 학년 : 2 학번 : 20151616 이름 : 최승환

1. 설계 문제 및 목표

미로찾기 프로젝트에서는 mfc를 사용해 미로를 그리고, 미로의 길을 찾아 최단경로를 찾아 이를 출력해줘야 한다. 이 과정에서 미로의 형태와 최단경로를 mfc GUI 프로그램으로 구현한다. 최단경로를 찾는 방법은 DFS,BFS 이 두 가지로 구분되어 있으며, 이 두 알고리즘으로 얻은 최단경로를 표시해주고, 방문했던 칸들을 다른 색으로 표시해준다.

우리가 길을 찾을 미로는 완전미로라는 점에 대해 먼저 생각해야 한다. 완전미로는 cycle이 존재하지 않는 미로로, 그 최단경로는 단 하나로, 정해져 있다. 따라서, DFS를 사용하던 BFS를 사용하던 최단경로는 동일하고, DFS를 이용해 미로의 끝을 찾은 경우, 그 경로는 반드시 최단경로이다.

일단 미로를 찾기 위해서는 미로에 대한 정보가 필요하다. 미로의 정보는 벽으로 그린 그림으로 주어지고, 그 형태는 ‘|’와 ‘-’ ‘+’의 조합으로, 다음과 같은 파일이 주어진다.



이 파일을 이용해 미로에 대한 정보를 저장하고 이를 이용해 길을 찾고 미로를 그리는 것이 이번 프로젝트의 할 일이다. 미로를 그려내기 위해 효율적인 자료구조를 선택해 활용하고, DFS,BFS 버튼을 활성화시켜 이 버튼들을 이용하도록 하는 것도 프로젝트의 일부이다. 또한, 프로그램들 중 확장자가 .maz 인 파일들만 보이도록 기본 설정해야 한다.

2. 요구사항

2.1 설계 목표 설정

미로 프로젝트의 각 주차별 프로그램을 설계하는 과정에서의 목표를 설정하고, 그 과정에서 요구되는 사항들을 정리한다.

1주차에는 미로를 생성하는 코드를 작성했다. 미로를 생성하는 방법에는 recursive division, kruskal algorithm 등 여러 가지가 존재하지만, 이 중에서 Eller’s algorithm 을 채택해 사용했다. Eller’s algorithm의 경우 한 줄씩 미로를 생성하며, 랜덤하게 벽을 뚫을지 안 뚫을지를 판별한다. 하지만 모든 벽이 뚫리지 않은 경우, 완전한 미로가 되지 않기 때문에 맨 마지막 벽을 그릴 때는 이 경우를 고려해주어야 한다. 마지막까지 벽이 뚫리지 않은 경우, 반드시 벽을 그려 완전 미로로 만들어준다. 마찬가지로 모든 벽을 뚫는 등의 경우에 대해서도 예외처리를 해줘야 한다.

2주차에서는 미로를 생성하는 프로그램을 작성하였다. 미로를 생성하는 방법은 maz파일을 입력받아 이 정보를 적절한 자료구조에 저장하는 것이다. 이 자료구조로는 구조체를 사용했고, 구조체에는 visit,up,down,left,right의 구성요소들이 있다. 이 구성요소들을 통해 해당하는 칸의 위, 아래, 왼쪽, 오른쪽에 벽이 있는지를 저장해주면 된다. 그리고 이 정보를 이용해 벽이 있는 경우 벽을 그리도록 하면 된다. 벽을 그릴 때는 오른쪽과 아래쪽 벽만 확인하며 그리고, 그려지지 않은 가장 윗줄과 가장 왼쪽은 마지막에 따로 그려주는 방법을 채택했다.

3주차에는 생성한 미로를 이용해 길을 찾는 알고리즘을 만들었다. 트리를 사용해 각 경우의 수를 구해내는 방법이 있고, 스택에 정보를 저장해 이를 출력하는 방법도 있다. 불완전 미로를 탐색하는 경우에는 반드시 트리를 사용해 최단경로를 구해줘야 하지만, 완전 미로의 경우, 굳이 트리를 사용해 이를 구할 필요가 없다. 구하는 최단 경로가 단 하나이기 때문이다. 특히 DFS의 경우, stack에 좌표를 저장해 이를 방문한 노드를 기록하며 그리면 된다.

2.2 합성

미로 프로젝트의 각 주차별 프로그램의 설계 및 구현을 위해 요구되는 이론, 자료구조, 알고리즘 등을 조사 분석하여 전체적인 설계를 수행한다.

1주차 미로 생성 알고리즘

Eller’s algorithm에 따라 한 줄씩 미로를 생성하고, 칸마다 숫자를 채워넣어 방의 구분을 해준다.

int main()

{

FILE \*fp = fopen("perfect.maz","w");

int rcnt,ccnt,input=1;

int row,col,limit=0,limit\_count=0;

int flag=0;

srand(time(NULL));

scanf("%d %d",&row,&col);

if(row>col)

{

limit = row/2;

}

else

{

limit = col/2;

}

arr = malloc(row\*sizeof(maze\*));

for(rcnt=0;rcnt<row;++rcnt)

{

arr[rcnt] = malloc(col\*sizeof(maze));

}

for(rcnt=0;rcnt<row;++rcnt)

{

for(ccnt=0;ccnt<col;++ccnt)

{

arr[rcnt][ccnt].key = input++;

arr[rcnt][ccnt].right=1;

arr[rcnt][ccnt].bottom=1;

}

}

for(rcnt=0;rcnt<row-1;++rcnt)

{

if(rcnt==0)

{

for(ccnt=0;ccnt<col-1;++ccnt)

{

//오른쪽 뚫기

if(rand()%2 == 0)

{

arr[0][ccnt].right=0;

arr[0][ccnt+1].key = arr[0][ccnt].key;

}

}

}

else

{

for(ccnt=0;ccnt<col-1;++ccnt)

{

if(arr[rcnt][ccnt].key != arr[rcnt][ccnt+1].key)

{

if(rand()%2 == 0)

{

arr[rcnt][ccnt].right=0;

//arr[rcnt][ccnt+1].key = arr[rcnt][ccnt].key;

change(rcnt,col,arr[rcnt][ccnt+1].key,arr[rcnt][ccnt].key);

}

}

}

}

for(ccnt=0;ccnt<col-1;++ccnt)

{

if((arr[rcnt][ccnt].key != arr[rcnt][ccnt+1].key)&&(flag == 0))

{

arr[rcnt][ccnt].bottom = 0;

arr[rcnt+1][ccnt].key = arr[rcnt][ccnt].key;

}

else if((arr[rcnt][ccnt].key != arr[rcnt][ccnt+1].key) && (flag == 1))

{

if(rand()%2 == 0)

{

arr[rcnt][ccnt].bottom = 0;

arr[rcnt+1][ccnt].key = arr[rcnt][ccnt].key;

}

flag=0;

}

else if(rand()%2 == 0)

{

arr[rcnt][ccnt].bottom = 0;

arr[rcnt+1][ccnt].key = arr[rcnt][ccnt].key;

flag = 1;

}

}

if(flag == 0)

{

arr[rcnt][col-1].bottom =0;

arr[rcnt+1][col-1].key = arr[rcnt][col-1].key;

}

flag=0;

}

for(ccnt=0;ccnt<col-1;++ccnt)

{

if(arr[row-1][ccnt].key != arr[row-1][ccnt+1].key)

{

arr[row-1][ccnt].right=0;

change(row-1,col,arr[row-1][ccnt+1].key,arr[row-1][ccnt].key);

}

}

--생략--

1. 처음 미로의 각 지점은 전부 막혀 있다.

2. i 번째와 i+1번째 열을 비교하여 서로 다른 집합이라면 1/2 확률(랜덤 자연수 생성)로 두 집합을 합친다.

3. i번째 열의 j번째와 j+1번째 행을 비교하여 서로 다른 집합이면 1/2 확률 로 두 집합을 합치고 수직 경로를 생성한다.

----3번을 반복------

4. 마지막 줄에 도달했을 때 단 하나의 수직 경로도 생성하지 않았다면 무조건 수직 경로를 생성한다.

2주차에는 MFC를 이용해 미로를 그리는 프로그램을 구현한다. 미로를 그리기 위해서는 먼저 .maz 확장자를 가진 파일이 보이도록 설정을 해야 한다. 그리고, setwindow를 이용해 그릴 미로의 범위를 선정해주며 시작을 해야 한다. setwindow로 범위를 선정해주기 위해서는 미로의 가로와 세로의 값을 먼저 알아야 하는데, 이 값은 maz 파이를을 끝까지 읽어본 뒤에 알 수 있다. 끝까지 읽어보며 지금까지 나온 개행의 개수를 센 후(마지막의 경우 EOF를 센다.) 이를 이용해 세로 값을 구해준다. 가로 값은 한 줄에서 공백과 벽의 개수를 세어 이용하면 된다. 이를 이용해, setwindow로 미로의 크기를 설정할 수 있다. 미로를 그릴 window의 크기를 설정한 후에는 직접적으로 미로를 그리기 시작하면 된다. 이 경우, upsidedown변수를 이용해, (좌표가 일반적으로 쓰는 x,y좌표와는 거꾸로 되어 있음) 좌표를 상식적인 x,y 좌표와 같게 만들어 준다. 이 이후, 자료구조에 저장된 각 칸에 저장된 정보를 사용하면서, 각 칸에 있는 벽들을 그려주면 된다. 벽을 그릴 때는 DrawLine을 이용하기 때문에 0,0 ->0,1 로 가는 형식으로 선을 그려줘야 한다. up down right left 를 각각 방문하기 때문에 벽의 모양은 규칙적이고, 규격에 맞춰 그릴 수 있다.

----파일을 읽어 자료구조에 미로 정보를 저장하는 함수 readFile----

bool readFile(const char\* filename){

//start of the user code

char ch;

int i=0,j=0;

FILE\* fp = fopen( filename, "r" );

if( fp == NULL ) return false;

while(fscanf(fp,"%c",&ch) != EOF&&(ch != '\n'))

{

if(ch == '-')

{

++ccnt;

}

}

rewind(fp);

while(fscanf(fp,"%c",&ch) != EOF)

{

if(ch == '\n')

{

++rcnt;

}

}

rcnt/=2;

rewind(fp);

maze = (room\*\*)malloc(sizeof(room\*)\*rcnt);

for(i=0;i<rcnt;++i)

{

maze[i] = (room\*)malloc(sizeof(room)\*ccnt);

}

while(fscanf(fp,"%c",&ch) != EOF)

{

if(ch == '+')

{

continue;

}

else if(ch == '\n')

{

break;

}

else if(ch == '-')

{

maze[0][j].up = 1;

}

}

i=0;

while(fscanf(fp,"%c",&ch) != EOF)

{

if(i == rcnt)

{

break;

}

if(ch == '\n')

{

j=0;

continue;

}

else if(ch == ' ')

{

fscanf(fp,"%c",&ch);

if(ch == ' ')

{

maze[i][j].right=0;

}

else

{

maze[i][j].right=1;

}

++j;

}

else if(ch == '+')

{

if(fscanf(fp,"%c",&ch) == EOF)

{

++i;

j=0;

break;

}

if( ch == '\n')

{

++i;

j=0;

continue;

}

if(ch == ' ')

{

maze[i][j].down=0;

}

else

{

maze[i][j].down=1;

}

++j;

}

}

for(j=1;j<ccnt;++j)

{

maze[0][j].left = maze[0][j-1].right;

}

for(j=0;j<ccnt;++j)

{

maze[rcnt-1][j].down = 1;

}

for(i=1;i<rcnt;++i)

{

maze[i][0].left = 1;

for(j=1;j<ccnt;++j)

{

maze[i][j].left = maze[i][j-1].right;

maze[i][j].up = maze[i-1][j].down;

}

}

fclose( fp );

setWindow(0,0,ccnt,rcnt,1);

return true; //edit after finish this function

//end of usercode

}

3주차에는 이 미로찾기 프로젝트의 메인이 되는 DFS와 BFS 알고리즘을 이용한다. 이는 분석에서 다루도록 하겠다.

2.3 분석

미로 프로젝트의 각 주차별 문제에 대해 기술하고, 상기 단계에서 설계한 알고리즘에 기반을 두어 목표로 하는 프로그램을 효과적으로 개발하는데 필요한 프로그램 기법들과 자료구조에 대하여 조사 분석하여 그것을 바탕으로 전체 프로그램 순서도를 작성한다. 또한 이러한 프로그램을 개발하는데 있어 고려해야할 모든 요소, 예를 들어, 입출력 양식, 관련 자료구조와 이론, 사용할 C언어 함수의 사용법 등 모든 가능한 고려 사항을 정리한다.

DFS의 경우 한 칸에서 위 아래 오른쪽 왼쪽을 방문하면서 벽이 없는 부분으로 이동하도록 해야 한다. 이 때 한 번에 이동할 수 있는 칸은 이 네 칸들 중 단 한칸이므로 이 부분의 구현은 if ~else if ~else로 되어 있다. 이 방식을 iterative하게 이용해, 한 칸씩 가능한 방으로 움직이면서 길을 찾게 되면 반복문 안에서 현 좌표인 i,j가 바뀌게 되고, 이 i,j가 마지막 칸이 되는 경우가 종료조건이다.

void DFS()

{

int i=0,j=0;

cycle=0;

branch=0;

visited = (history\*)malloc(sizeof(history)\*rcnt\*ccnt\*2);

best = (history\*)malloc(sizeof(history)\*rcnt\*ccnt);

visited[0].row = 0;

visited[0].col = 0;

best[0].row=0;

best[0].col=0;

for(i=0;i<rcnt;++i)

{

for(j=0;j<ccnt;++j)

{

maze[i][j].visit = 0;

}

}

maze[0][0].visit = 1;

i=0;

j=0;

while(i != rcnt-1 || j != ccnt-1)

{

cycle++;

if(maze[i][j].right == 0 && maze[i][j+1].visit == 0)

{

++j;

++branch;

maze[i][j].visit = 1;

visited[cycle].row = i;

visited[cycle].col = j;

best[branch].row = i;

best[branch].col = j;

continue;

}

else if(maze[i][j].down == 0 && maze[i+1][j].visit == 0)

{

++i;

++branch;

maze[i][j].visit = 1;

visited[cycle].row = i;

visited[cycle].col = j;

best[branch].row = i;

best[branch].col = j;

continue;

}

else if(maze[i][j].left == 0 && maze[i][j-1].visit == 0)

{

--j;

++branch;

maze[i][j].visit = 1;

visited[cycle].row = i;

visited[cycle].col = j;

best[branch].row = i;

best[branch].col = j;

continue;

}

else if(maze[i][j].up == 0 && maze[i-1][j].visit == 0)

{

--i;

++branch;

maze[i][j].visit = 1;

visited[cycle].row = i;

visited[cycle].col = j;

best[branch].row = i;

best[branch].col = j;

continue;

}

else

{

--branch;

visited[cycle].row = best[branch].row;

visited[cycle].col = best[branch].col;

i = best[branch].row;

j = best[branch].col;

continue;

}

}

}

BFS의 경우 DFS와 유사하게 위 아래 왼쪽 오른쪽이 방문 가능한지를 보아야 한다. 따라서 DFS와 마찬가지로 그 칸으로 가는 길에 벽이 있는지와, 그 칸을 방문한 적이 있는지를 항시 체크해줘야 한다. 하지만 DFS와 다르게 한 깊이에서 위 아래 왼쪽 오른쪽으로 가는 모든 칸을 queue에 푸시해야 하기 때문에 DFS와는 조금 다르게 if 문으로 구성이 되어 있다.

-----BFS의 각 노드 방문 부분-----

if (maze[x][y].right == 0 && visit[x][y + 1] == false)

{

visit[x][y + 1] = true;

que.push(ii(x, y + 1));

}

if (maze[x][y].down == 0 && visit[x + 1][y] == false)

{

visit[x+1][y] = true;

que.push(ii(x + 1, y));

}

if (maze[x][y].left == 0 && visit[x][y - 1] == false)

{

visit[x][y - 1] = true;

que.push(ii(x, y - 1));

}

if (maze[x][y].up == 0 && visit[x - 1][y] == false )

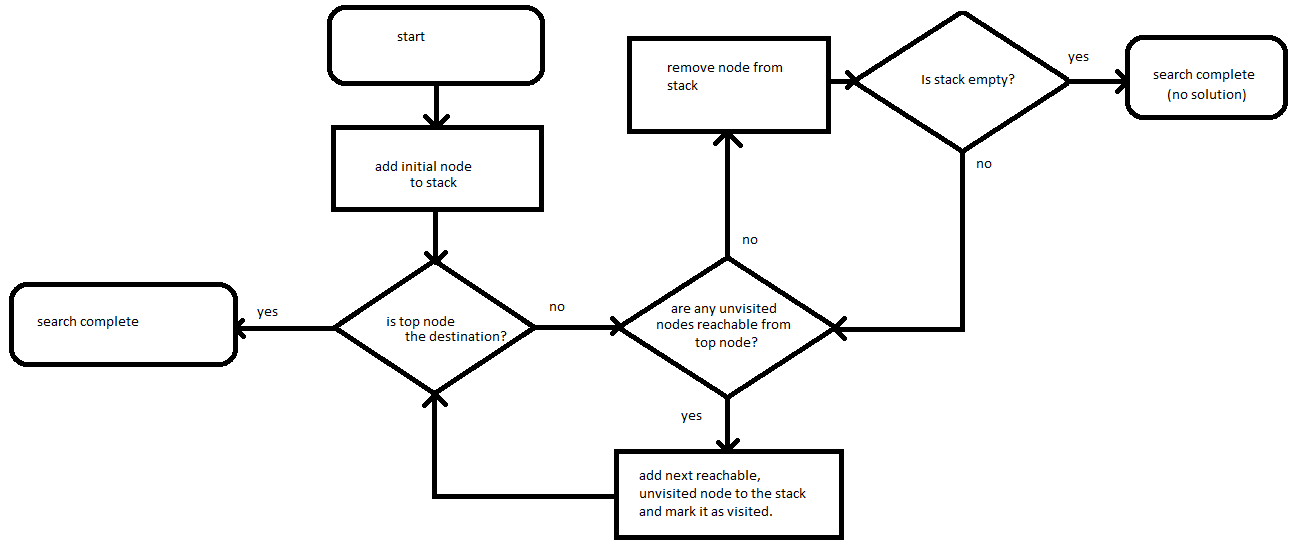
{

visit[x - 1][y] = true;

que.push(ii(x - 1, y));

}

-----------------------------------------



DFS 단순화 flowchart

2.4 제작

위에서 설계한 내용을 C언어를 사용해 구현한다. 구현 후 프로그램의 각 구성요소를 상세히 분석하여 구현방법을 프로그램 기법과 정리한 이론 등과 연관 지어 정리한다.

bool readFile(const char\* filename)

사용자가 파일 열기 dialog에서 선택한 파일 이름이 filename이 전달되면, 그 파일을 열고, 자료구조에 맞는 형태로 저장하는 함수. 파일 읽기가 성공하면 true, 실패하면 false를 return한다. 또, 파일 읽기가 끝나고, 그림을 그릴 영역을 설정해줘야 한다.

bool writeFIle(const char\* filename)

부른 파일을 저장하는 함수. 프로젝트에서 사용할 필요 없음.

void freeMemory()

readFile이나 Drawbuffered 후 등 사용자가 작성한 다른 함수에서 할당한 동적 메모리를 free시킨다. 보고 있던 파일에서 새 파일을 불러올 때 실행된다.

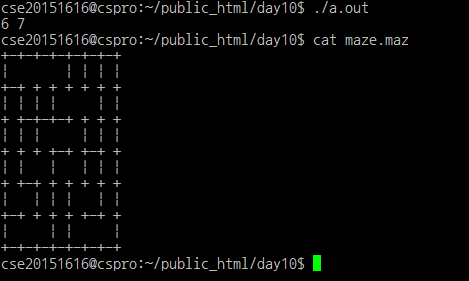
static void drawBuffered()

자료구조에 저장되어 있는 내용을 그리는 함수. 그리는 행위는 이 함수 내에서만 가능하기 때문에 DFS나 BFS 등에서 그리는 것을 생각하면서 짤 필요가 있다. 모든 것이 저장된 후 이 함수 내에서 저장된 내용을 바탕으로 한꺼번에 그리게 된다. 또한 그림이 겹치는 것을 막기 위해 visit 을 free시킨다.

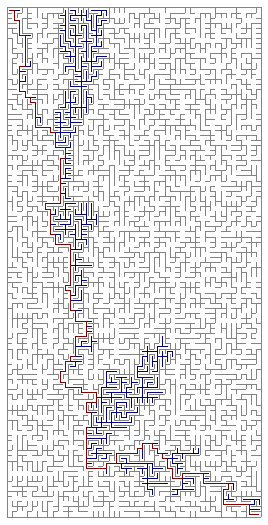
2.5 시험

위의 과정에서 수행한 문제 정의, 프로그램 순서도와 순서도 상의 각 부분 역할 및 구현 방법, 프로그램의 구현 방법, 구현 방법의 이론과의 연관성, 구현한 프로그램의 내용, 수행화면 등을 정리한다.

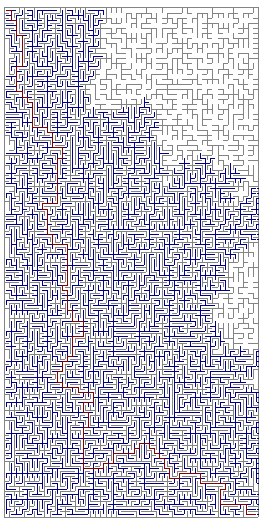
미로 생성



DFS)



BFS)



2.6 평가

문제와 이론의 연관성이 적절한지, 순서도와 실제 구현 사이에 차이는 없는지 프로그램은 잘 동작하는지 등을 평가한다.

주어진 알고리즘인 Eller’s algorithm을 이용해 미로를 짰다. 그리고, DFS와 BFS를 구현해 미로의 결로를 찾아내는데 성공했다. 이를 사용할 때 비슷한 형태를 띄도록 구현해, 자료구조의 사용을 용이하게 만들어 두었고, DFS에서 사용했던 자료구조를 BFS에서 다시 사용해 뽑도록 하고 있다는 점에서, 코드가 효율적으로 사용되었다고 할 수 있다. 하지만, BFS의 경우 그리는 방법에 문제가 발생했는데, 서로 인접한 칸들을 연속으로 방문하는 방식이 아니라 pop되는 칸들을 방문하는 방식이기 때문에 점이 왔다갔다 하며 대각선을 그리는 현상이 발견되었다. 이를 막기 위해서 새로운 배열 visit을 선언했고, 이는 BFS를 그릴 때에만 사용되었다. 이런 점은 DFS를 사용할 때부터 감안을 해 그리도록 했으면 더 활용도가 높았을 것이라고 생각한다. 마지막으로 종료 조건은 최단 경로를 구하는 것이 최종 목적이었기 때문에 마지막 점에 도달하면 종료하도록 만들어져 있다.

2.7 환경

window에서 제공하는 MFC를 사용했기 때문에 visual studio 2015 환경에서 제작되었다.

2.8 미학

프로젝트에서는 주어진 형식에 맞춰서 그림을 그려야 한다. 또, 최단경로와 방문경로의 색을 다르게 하여 구분을 쉽게 하여야 했다. 또한, 선을 굵게 그려 미로가 작은 경우에도 보기 쉽도록 만드는 대신에 선을 얇게 만들어 미로가 큰 경우에도 서로 뭉쳐 보이지 않도록 했다.

2.9 보건 및 안정

파일을 열기 전에는 DFS,BFS 버튼을 누를 수가 없다. 또한, DFS에서 BFS로 변할 때도 메모리를 프리해주기 때문에 그림이 겹치거나 잘못된 접근으로 인한 세그먼트 오류가 발생하지 않도록 제작되어 있다.

3. 기 타

3.1 환경 구성

프로젝트를 수행하기 위해 window 환경으로 Microsoft visual studio 2015를 사용하였으며, Windows 버전은 Windows 10

64bit 운영체제이다.

3.2 참고 사항

MFC는 윈도우 GUI프로그래밍을 위한 프로젝트를 실행시키기 위해서 윈도우 환경에서 파일을 실행해야 한다.

3.3 팀 구성

최승환 100%

3.4 수행기간

2016. 5. 26 ~ 2016. 6. 14