알고리즘

2018 Term Project

Hidato Puzzle Generator/Solver

최종 보고서

13조

20132872 소프트웨어융합학부 박귀환

20091286 소프트웨어융합학부 홍일권

20133164 컴퓨터공학부 조경문

-목차-

1. 팀 구성 및 분담

1. 20132872 소프트웨어융합학부 박귀환 – Generator 및 Solver 개발
2. 20091286 소프트웨어융합학부 홍일권 – Generator 및 Solver 개발
3. 20133164 컴퓨터공학부 조경문 – Generator 개발 및 코드 디버깅

2. 프로젝트 계획

2-1. 개발 언어 및 개발 환경

개발언어: C++

개발환경: Linux Ubuntu, Window, visual studio2017

2-1. 프로젝트 추진 일정

프로젝트 일정	**2018.10.31 ~ 2018.12.14 (2개월)**																	
단계	**10월 31일 – 11월 8일**						**11월 8일 – 12월 10일**						**12월 10일 – 12월 14일**					
프로젝트 계획																		
분석/설계																		
개발																		
종합 기능 테스트 /디버깅																		

2-2. 단계별 세부 일정

일정	단계	작업
2018.10.31 ~2018.11.08	**프로젝트계획**	프로젝트 범위 확정 프로젝트 일정 확정 프로젝트 진행 방향 확정
2018.11.02 ~2018.11.12	**분석/설계**	라이브러리 및 알고리즘 구체적으로 정의
2018.11.08 ~2018.12.10	**개발**	시스템 구현
2018.12.10 ~2018.12.14	**종합 기능 테스트/디버깅**	통합테스트 미비점 보완 오류 해결

3. 개발 진행 결과 및 개발 순서

3-1. 개발 진행 결과 보고

개발 진행 결과: Hidato puzzle Generator 및 solver 개발 완료, 간단한 콘솔 디자인 추가

사용 라이브러리:

-iostream(전반적인 입출력에 사용)

-fstream(파일 입출력에 사용)

-vector(격자 생성 및 퍼즐을 다룰 때 사용)

-stdlib.h(난수 생성 등에 사용)

-time.h(srand() 함수에 시드로 사용)

-windows.h(콘솔 디자인에 사용)

3-2. 개발 순서 보고

1. Grid Generator로 격자 생성 후 난수 생성을 이용하여 격자 내 퍼즐 모형 생성

2. 랜덤으로 시작점을 정하고, 퍼즐 모형 생성에서 만든 모형의 개수를 퍼즐의 마지막 숫자로 정함

3. backtracking 알고리즘 이용하여, 퍼즐 모형에 해당하는 정답 path 생성

4. 정답 path에서 hint 수만큼을 제외한 나머지 퍼즐 모형 칸을 가려서 퍼즐을 생성함

5. 퍼즐 파일에서 데이터를 읽어와서 인덱스 값이 1이 인덱스를 시작점으로 설정하고, 이외에 0이 아닌 값은 이동 가능 블록으로 추가하고 시작 좌표, 배열, 이동 가능 블록, 현재 지점의 순서, 행, 열 변수를 argument로 go_solve 함수 실행

6. 현재 지점에서 8방향으로 루프를 돌면서 인덱스 값을 체크해서 이동가능한 블록과 힌트 블록에 대해 재귀적으로 함수 실행

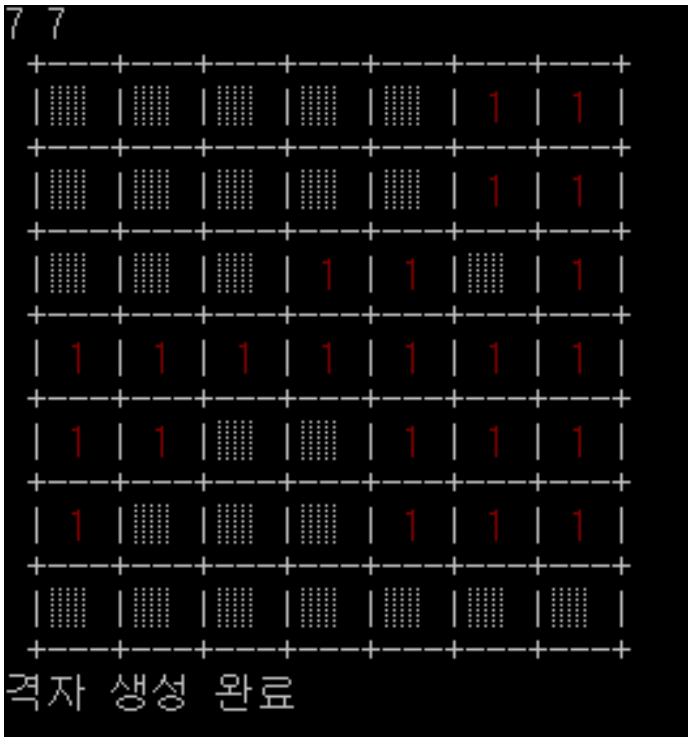
7. 함수 종료 후 풀이 완료된 퍼즐에 대해 출력 수행

8. windows.h 라이브러리 적용하여 간단한 콘솔 디자인 추가함

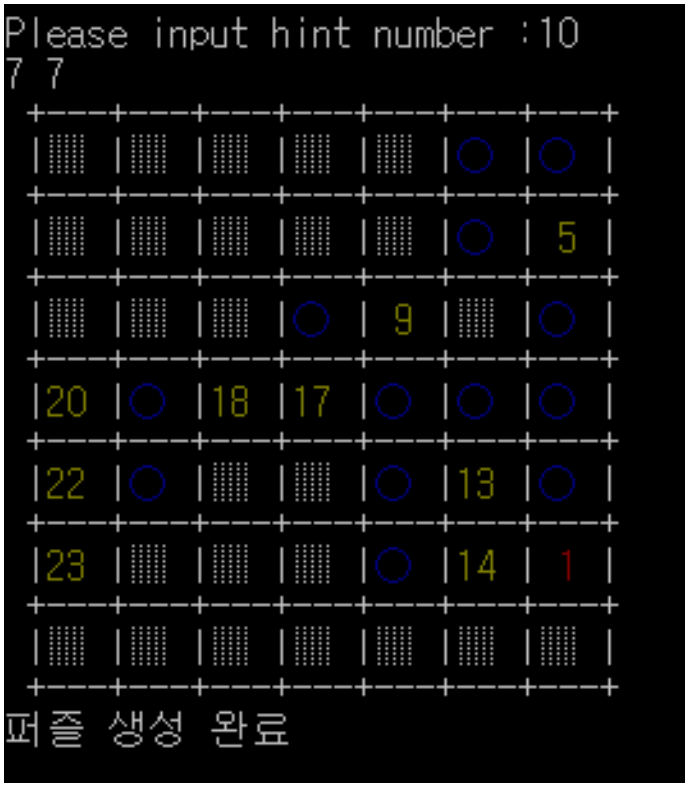
4. 프로그램 실행 화면 첨부

<실행 화면>

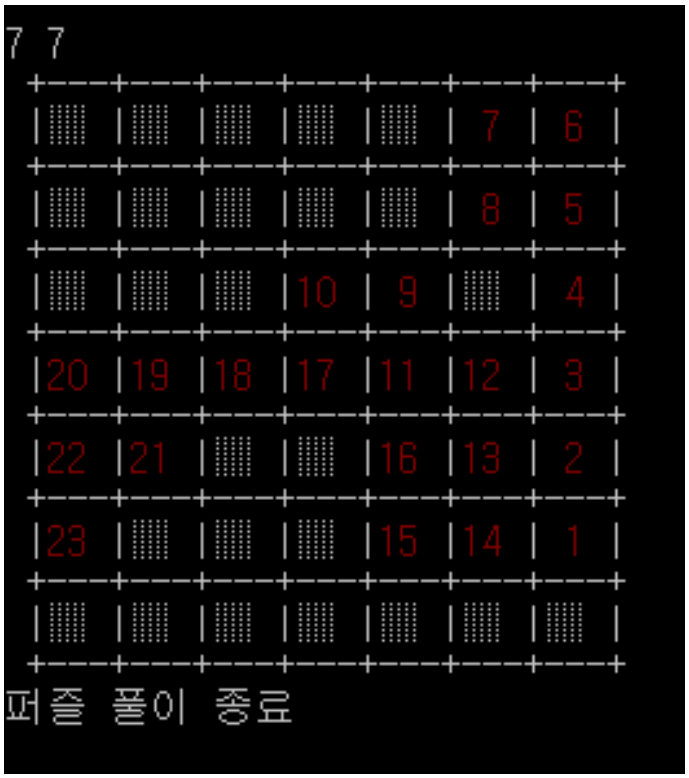
- 격자 생성 화면



-퍼즐 생성 화면



- 퍼즐 풀이 화면



5. 프로그램 코드 첨부

<코드 첨부>

```
#include <iostream>
#include <fstream>
#include <vector>
#include <stdlib.h>
#include <time.h>
#include<windows.h>

using namespace std;

//이동할 수 있는 8방향에 대한 순서쌍 배열 2개
int dx[8] = { -1, -1, -1, 0, 0, 1, 1, 1 };
int dy[8] = { 1, 0, -1, 1, -1, 1, 0, -1 };

void Generate_Grid();
void Generate_Puzzle(const char* filename);
void Hidato_Solver(const char* filename);

int main()
{
	int n;
	Generate_Grid();
	cout << "격자 생성 완료₩n₩n";
```

```
	Generate_Puzzle("grid.txt");
	cout << "퍼즐 생성 완료₩n₩n";
	Hidato_Solver("puzzle.txt");
	cout << "퍼즐 풀이 종료₩n₩n";
	cin >> n;
}

void Generate_Grid()
{

	vector<vector<bool>> map;
	ofstream fout;
	fout.open("grid.txt");


	srand(time(0));//난수 발생을 랜덤하게 하기 위해 srand로 시드값에 time 값 할당
	int row = rand() % 6 + 5, col = rand()%6 + 5; // 0부터 5사이 랜덤값 생성 후, 5 더해 5부터 10사이 난수 생성 후, 행 과 열 변수에 할당


	map.resize(row); //행의 개수를 생성한 난수 값으로 리사이즈함
	for (int i = 0; i < row; i++) map[i].resize(col); // 행마다 루프를 돌면서 열의 길이를 리사이즈함
```

```
for (int i = 0; i < row; i++) //행마다 루프
{
	for (int j = 0; j < col; j++) map[i][j] = false; //열마다 루프를 돌면서, 각 map 인덱스 값을 전부 false로 초기화
}

int x = rand() % row, y = rand() % col; // 0~row-1 사이의 x,  0~col-1 사이의 y로 난수 초기화 -> 시작점 x,y
map[x][y] = true; // 해당 x,y 인덱스를 true

int len = rand() % ((row*col)) + 1;   // at least length should be greater than 1,1부터 총 인덱스 개수 사이의 난수 생성

while (len < row*col/2) {
	len = rand() % ((row*col)) + 1;
	if (len >=row*col/2) break;
}

for (int i=0; i<len; i++) //0부터 len 까지 루프
{
	bool chk = false; //체크 변수 false로 할당
	for (int j = 0; j < 30; j++) //  임의의 수 30까지 루프를 돈다
	{
		int ran = rand() % 8; //방향을 정하기 위해 0~7사이 난수 생성
```

```
                int nx = x + dx[ran], ny = y + dy[ran]; //시작점 x,y 에 방향 배열의 값을 랜덤하게 호출해 더함


                if (0 <= nx && nx < row && 0 <= ny && ny < col)  // 각 nx,ny가 0부터 row, 0부터 col 사이에 있을때
                {
                    if (map[nx][ny] == true) continue; //해당 인덱스 값이 true면 루프 한번 건너뜀
                    map[nx][ny] = true; // 해당 인덱스 true 할당
                    chk = true; //bool true 할당
                    x = nx, y = ny; //기존의 x,y 값을 새로운 nx,ny 값으로 할당
                    break;
                }
            }
            if (!chk) break; //chk false 면 루프빠져나감
        }

        // 그리드 파일 생성 및 콘솔에 생성된 퍼즐 출력
        fout << map.size() << " " << map[0].size() << "₩n";
        cout << map.size() << " " << map[0].size() << "₩n";
        for (int i = 0; i < map.size(); i++)
        {
            for (int j = 0; j < (map[i].size() * 4); j++)
            {
```

```
            if (j == 0) cout << " +";
            else if (j > 0 && j % 4 == 0) cout << "+";
            else cout << "-";
        }
        cout << "+";
        cout << "₩n";
        for (int j = 0; j < map[i].size(); j++)
        {
            fout << map[i][j] << " ";
            if (map[i][j] == 1)
            {
                cout << " | ";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 4);
                cout << map[i][j];
            }
            else {
                cout << " |";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 7);
                cout << "▒";
            }

            SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 7);
```

```
            }
            cout << " |";
            cout << "₩n";
            fout << "₩n";
        }

        for (int j = 0; j < (map[0].size() * 4); j++)
        {
            if (j == 0) cout << " +";
            else if (j > 0 && j % 4 == 0) cout << "+";
            else cout << "-";
        }
        cout << "+";

        fout << "₩n";
        cout << "₩n";
        fout.close();
}

//(backtracking) 방식 적용해 길을 찾음
bool make_puzzle(
        int x, int y, int len, int avail_block,
        int row, int col, vector<vector<int>>& chk, vector<vector<int>>& ans)
{
```

if (len == avail_block) return true; // len 변수와 avail_block 값이 같으면 true 리턴, 재귀의 base case , 길을 다 찾았으니 리턴

for (int i = 0; i < 8; i++) //방향키 단위로 루프를 돈다

{

int nx = x + dx[i], ny = y + dy[i]; // x,y에 각 방향키 값을 더해본다.

if (0 <= nx && nx < row && 0 <= ny && ny < col) // 각 x,y가 배열 범위 내에 있을때

{

if (!chk[nx][ny]) continue; // chk 인덱스 값이 0이면,즉 길이 아니면 루프를 한번 건너 뛴다.

if (ans[nx][ny]) continue; //ans 인덱스 값이 0이 아니면, 즉 이미 길이면 루프를 한번 건너 뛴다.

ans[nx][ny] = ++len; // ans 인덱스에 len+1 값을 할당 (최초값은 1이므로 1씩 값을 더하면서 길을 찾는다)

if (make_puzzle(nx, ny, len, avail_block, row, col, chk, ans)) //재귀호출 시 true면 1리턴, 다음 길을 찾음

return 1 ;

len--; // len-1

ans[nx][ny] = 0; //ans 인덱스 값을 0으로 할당

}

}

return false; //루프를 끝날때까지 리턴이 없으면 false 리턴

}

```
//퍼즐 생성
void Generate_Puzzle(const char* filename)
{
	ifstream fin; // 파일을 읽어옴
	fin.open(filename); //파일 오픈

	int row, col; // 행 열 변수 선언
	fin >> row >> col; //행 열 값

	// gird 파일서 격자와 퍼즐의 형태 불러오기

	vector<vector<int>> chk; //2차원 벡터 생성, 기존의 그리드 파일 불러옴
	chk.resize(row); //행 크기로 배열의 행 리사이즈
	for (int i = 0; i < row; i++) chk[i].resize(col); //열 크기를 리사이즈

	int avail_block=0; // 가능한 블록을 0으로 할당
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++) //2차원배열을 모두 돌면서
		{
			fin >> chk[i][j]; // 각 인덱스 값을 읽어옴
			if (chk[i][j]) avail_block++;  // 만약 해당 인덱스에 값이 할당되어
```

있을 경우, avail_block 값 추가 ,길의 개수만큼 블록 추가

```
            }
        }
        fin.close(); //파일을 닫음

        vector<vector<int>> ans; // 정답을 담는 2차원 벡터 생성
        ans.resize(row); //행 크기로 배열의 행 리사이즈
        for (int i = 0; i < row; i++) ans[i].resize(col); // 열 크기로 배열의 열 리사이즈

        cout << "퍼즐의 시작점을 설정하고 정답을 만드는중입니다." << endl;
        while (1) // 루프
        {
            //vector 초기화
            for (int i = 0; i < row; i++)
            {
                for (int j = 0; j < col; j++) ans[i][j] = 0;
            }

            //랜덤한 시작점을 정함
            int ranx, rany;
            while (1)
            {
                ranx = rand() % row, rany = rand() % col; // 0~row-1, 0~col-1 의
```

난수로 랜덤한 좌표값 설정

```
                if (chk[ranx][rany]) break;// 그리드 파일에서 해당 인덱스가 1이어야 시작가능하기 때문에 1이라면 종료
                }
                // 시작점 필터링
                while (1)
                {
                        int count = 0;
                        int tmpx = 0, tmpy = 0;
                        for (int i = 0; i < 8; i++) {
                                tmpx = ranx + dx[i];
                                tmpy = rany + dy[i];
                                if (0 <= tmpx && tmpx < row && 0 <= tmpy && tmpy < col &&chk[tmpx][tmpy] == 1) // 각 x,y가 배열 범위 내에 있을때
                                {
                                        count++;
                                }
                        }
                        if (count <= 3) break;
                        else ranx = rand() % row, rany = rand() % col; // 0~row-1, 0~col-1 의 난수로 랜덤한 좌표값 설정
                        if (chk[ranx][rany]) break;// 그리드 파일에서 해당 인덱스가 1이어야 시작가능하기 때문에 1이라면 종료
                }

                int sx = ranx, sy = rany; //위에서 생성한 랜덤 좌표값을 시작점으로 설정
```

```
		ans[sx][sy] = 1; //해당 좌표 인덱스를 1로 설정



		if (make_puzzle(sx, sy, 1, avail_block, row, col, chk, ans)) //탐색으로 길을 찾는다.
		{
			cout << "완료₩n" << endl;
			break;
		}
		cout << "시작점을 재설정하고 정답을 만드는중입니다.₩n" << endl;
	}



	int count = 0;
	for (int i = 0; i < row; i++) {
		for (int j = 0; j < col; j++) {
			if (chk[i][j] == 1) count++;


		}
	}
	int hint; //힌트 카운트하는 변수


	while (1) // 루프
```

```
	{
		printf("힌트 개수는 %d~~%d정도의 적절합니다.₩n", count / 3, count / 2);
		printf("퍼즐에 힌트의 개수를 정해주세요 :");//힌트 입력받음
		scanf("%d", &hint); // 힌트 수 읽음
		printf("₩n₩n");
		if (hint >= avail_block) { // 힌트가 뚫린 블록 개수보다 많거나 같으면
			printf("hint number should be smaller than %d₩n", avail_block); // 경고문
		}
		else if (avail_block>=2 && hint<2) { // 가능한 블록이 2개 이상이고 힌트가 두개 미만이면
			printf("hint number should be greater than or equal to 2₩n"); // 경고문
		}
		else break; // 조건문 안걸리면 정지
	}

	int remove = avail_block - hint; // 지우는 칸수는 가능한 블록에서 힌트의 개수를 뺀 칸수

	for (int k = 0; k <= 10; k++)  // 퍼즐전체를 10번 반복
	{
		for (int i = 0; i < row; i++) // 행 루프
		{
			for (int j = 0; j < col; j++) //열 루프
```

```
{
    if (ans[i][j] > 0) // 정답 배열 인덱스 값이 0보다 클때,길일때
    {
        if (ans[i][j] == 1) continue; // 1이면 루프 한번 뜀
        if (ans[i][j] == avail_block) continue;// ans 배열 값이 뚫린 블록 값과 같다면, 즉 마지막 칸이면 루프 한번 뜀
        if (remove == 0) continue;// 지울 칸의 개수가 0이면 루프 뜀

        //지우는 칸수를 불규칙적으로 하기 위해 랜덤 값을 사용

        int ran = rand() % 9; //0-9 10%확률로 얻음.
        if (k == 0 && 2 > ran) {
            ans[i][j] = -1;
            remove--;
        }
        else if (k == 1 && 1 > ran) {
            ans[i][j] = -1;
            remove--;
        }
        else if (k == 2 && 1 > ran) {
            ans[i][j] = -1;
```

```
                remove--;
            }
            else if (k == 3 && 1 > ran) {
                ans[i][j] = -1;
                remove--;
            }
            else if (k == 4 && 1 > ran) {
                ans[i][j] = -1;
                remove--;
            }
            else if (k == 5 && 1 > ran) {
                ans[i][j] = -1;
                remove--;
            }
            else if (k == 6 && 1 > ran) {
                ans[i][j] = -1;
                remove--;
            }
            else if (k == 7 && 1 > ran) {
                ans[i][j] = -1;
                remove--;
            }
            else if (k == 8 && 1 > ran) {
                ans[i][j] = -1;
```

```
                                        remove--;
                                    }
                                    else if (k == 9 && 1 > ran) {
                                        ans[i][j] = -1;
                                        remove--;
                                    }
                                    else if (k == 10 && remove > 0) {// 무조
건지운다.

                                        ans[i][j] = -1;
                                        remove--;
                                    }
                                }
                            }
                        }
                    }

//  콘솔 출력 및 파일 쓰기
                    ofstream fout; //퍼즐 파일 생성
                    fout.open("puzzle.txt"); // 퍼즐 파일 생성
                    fout << row   << " " << col << "₩n"; //행 과 열 넣음
                    cout << row   << " " << col << "₩n";

                    for (int i = 0; i < row; i++)
```

```
{
                for (int j = 0; j < (ans[i].size() * 4); j++)
                {
                        if (j == 0) cout << " +";
                        else if (j > 0 && j % 4 == 0) cout << "+";
                        else cout << "-";
                }
                cout << "+";
                cout << "₩n";
                for (int j = 0; j < ans[i].size(); j++)
                {
                        fout << ans[i][j] << " ";
                        if (ans[i][j] == 1 || ans[i][j]==avail_block)
                        {
                                if (ans[i][j] > 9) {
                                        cout << " |";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 4);
                                        cout << ans[i][j];
                                }
                                else {
                                        cout << " | ";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 4);
```

```
                        cout << ans[i][j];
                    }
                }
                else if(ans[i][j] ==0) {
                    cout << " |";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 7);
                    cout << "▒";
                }

                else if (ans[i][j] > 1 && ans[i][j] <10) {
                    cout << " | ";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 6);
                    cout << ans[i][j];
                }
                else if (ans[i][j] > 9) {

                    cout << " |";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 6);
                    cout << ans[i][j];
                }
                else {
                    cout << " |";
```

```
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 2);
                                        cout << "○";
                                }

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 7);
                        }
                        cout << " |";
                        cout << "₩n";
                        fout << "₩n";




}

for (int j = 0; j < (ans[0].size() * 4); j++)
{
        if (j == 0) cout << " +";
        else if (j > 0 && j % 4 == 0) cout << "+";
        else cout << "-";
}
cout << "+";

fout << "₩n";
```

```
	cout << "₩n";
	fout.close();//파일 닫기
}

//퍼즐해결 알고리즘
bool go_solve(
	int x, int y, vector<vector<int>> &map, int len, int avail_block
	, int row, int col)
{
	if (len == avail_block) return true; //현재 지점과 가능한 길 블록의 개수와 같다면 종료
	for (int i = 0; i < 8; i++) // 8방향으로 루프
	{
		int nx = x + dx[i], ny = y + dy[i]; // 방향 배열 값을 기존 지점에서 추가해 새로운 좌표값 확보
		if (0 <= nx && nx < row && 0 <= ny && ny < col) // 새로운 좌표가 배열을 벗어나지 않을 때
		{

			if (map[nx][ny] == 0) continue; // 인덱스 값이 0이면 루프를 한번 건너뜀
			if (map[nx][ny] > 0 && (len + 1) == map[nx][ny]) // 인덱스값이 0보다높고 (벽이아니고), 새로운 좌표의 값이 기존 지점 값보다 하나 높을때(=힌트로 설정된 다음길)
			{
				if (go_solve(nx, ny, map, len + 1, avail_block, row, col)) //
```

해당 새로운 좌표가 다음 지점이므로 그 지점에 대해 재귀를 실행한다.

```
                                {
                                        return true;
                                }


                        }
                        else if (map[nx][ny] == -1) // 만약 가려진 점(힌트 없는 점)
                        {
                                map[nx][ny] = len + 1; // 일단 해당 좌표를 다음 지점으로 가정하여 값을 할당
                                if (go_solve(nx, ny, map, len+1, avail_block, row, col))  // 다음 지점에 대해 재귀 실행
                                {
                                        return true;
                                }
                                map[nx][ny] = -1; // 만약 잘못된 길이라면 false 를 리턴하여 -1로 다시 세팅함
                        }
                }
        }
        return false; // 위의 조건문을 다 빠져 나온 것은 잘못된 길로 들었다는 뜻이므로 false를 리턴해 백트래킹한다.
}
```

```
void Hidato_Solver(const char* filename)
{
	ifstream fin;
	fin.open(filename);//퍼즐 텍스트 파일을 오픈

	int row, col;
	fin >> row >> col; // 행과 열 수 가져옴

	vector<vector<int>> map; //2차원 배열선언
	map.resize(row); // 행의 크기로 배열 리사이즈
	for (int i = 0; i < row; i++) map[i].resize(col); //열의 크기로 배열 리사이즈

	int x, y; //좌표 2개
	int avail_block = 0; //이동가능한 길 블록
	//행렬 nested for
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++)
		{
			fin >> map[i][j]; //2차원 배열에 값을 입력받음
			if (map[i][j] == 1) x = i, y = j; // 만약 입력받은 값이 1이라면 해당 값을 시작 좌표로 설정
			if (map[i][j] != 0) avail_block++; // 입력받은 값이 0이 아니면 이동가능한 블록 개수 추가
```

```
		}
	}
	fin.close(); // 파일 종료
	go_solve(x, y, map, 1, avail_block, row, col);// 문제 풀이 함수에 시작 좌표, 배열, 블록, 행, 열 개수 넣는다.

	ofstream fout;
	fout.open("solution.txt"); //결과 출력 파일

	fout << row << " " << col << "₩n";
	cout << row << " " << col << "₩n";

	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < (map[i].size() * 4); j++)
		{
			if (j == 0) cout << " +";
			else if (j > 0 && j % 4 == 0) cout << "+";
			else cout << "-";
		}
		cout << "+";
		cout << "₩n";
		for (int j = 0; j < map[i].size(); j++)
		{
```

```
                fout << map[i][j] << " ";
                if (map[i][j] >= 1 && map[i][j] <10)
                {
                        cout << " | ";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 4);
                        cout << map[i][j];
                }
                else if(map[i][j] > 9) {
                        cout << " |";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 4);
                        cout << map[i][j];
                }
                else {
                        cout << " |";

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 7);
                        cout << "▒";
                }

                SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 7);
        }
        cout << " |";
        cout << "₩n";
```

```
            fout << "₩n";
        }
        for (int j = 0; j < (map[0].size() * 4); j++)
        {
            if (j == 0) cout << " +";
            else if (j > 0 && j % 4 == 0) cout << "+";
            else cout << "-";
        }
        cout << "+";

        fout << "₩n";
        cout << "₩n";
        fout.close();
}
```