

목 차

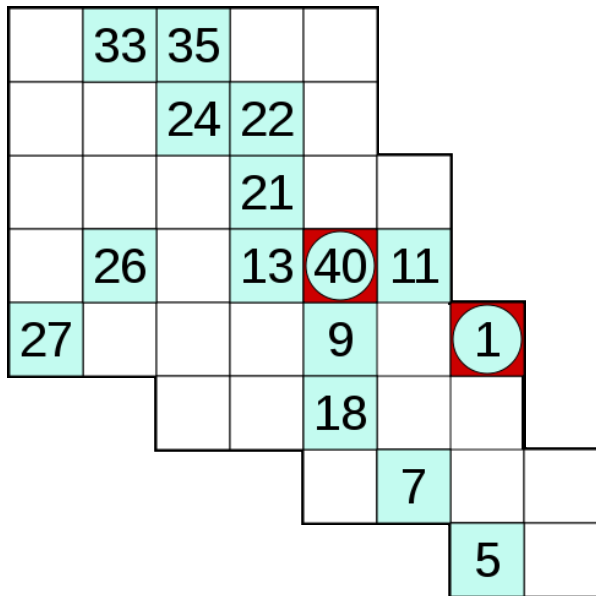
1. 프로젝트 개요
2. 핵심 알고리즘
3. Generator
4. Solver
5. 실행 결과 및 평가
6. 참조 및 코드

1. 프로젝트 개요

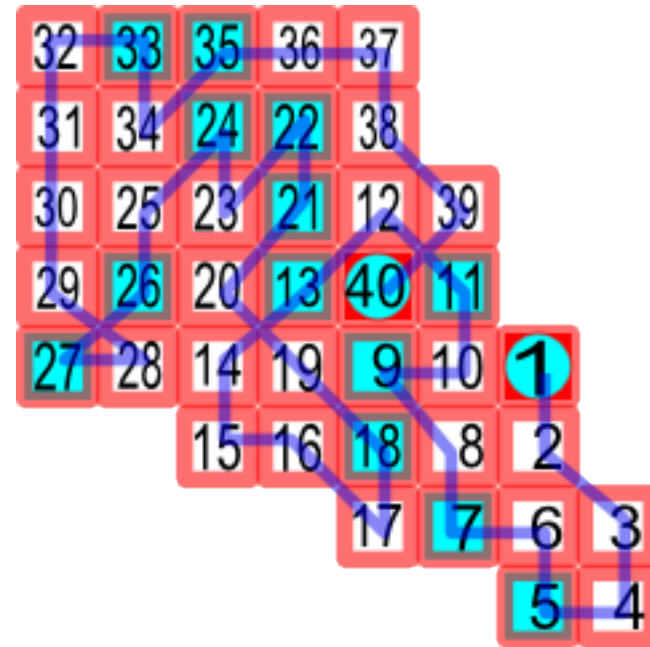
- 프로젝트 실행환경
 - 사용언어
 - Microsoft Visual C/C++(GNU C/C++는 포함되지 않음)
 - 프로젝트 PC 환경
 - OS: Windows 10 Pro ver.1809(64-bits)
 - Editors:
 - Microsoft Visual Studio 2017 Community

히다토 퍼즐이란?

puzzle



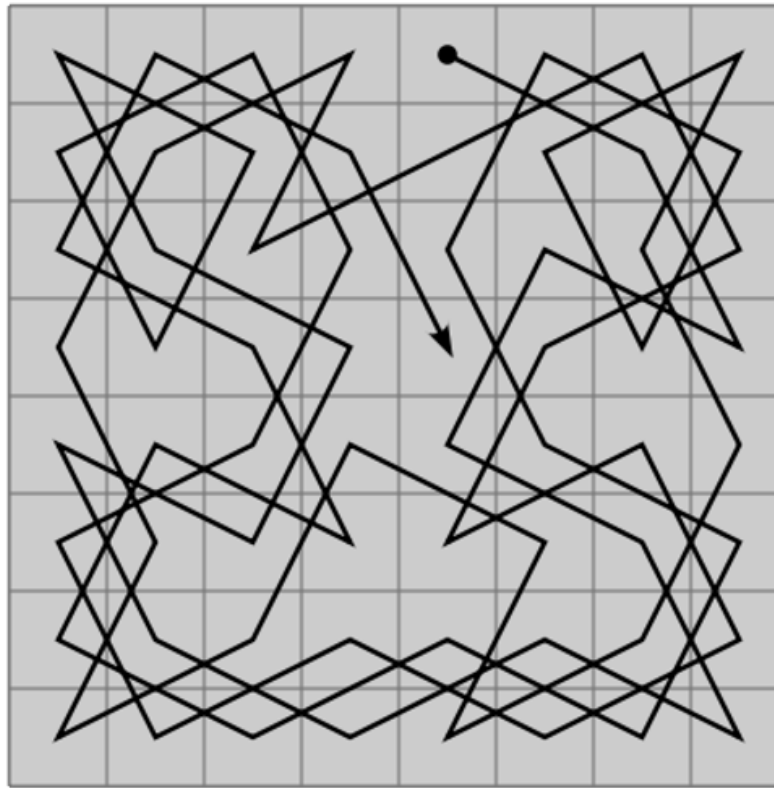
solution



- 주어진 격자에 가로, 세로, 대각선으로 연결되는 연속된 숫자를 채우는 퍼즐

2. 핵심 알고리즘

- Back Tracking Algorithm
 - Ex) Knight's tour Algorithm



Main

- Main 함수

- Generator

```
#include "HidatoSol.h"
#include "HidatoGen.h"
int main() {

    // Generator
    ifstream input;
    input.open("input.txt");
    ofstream output;
    output.open("output.txt");

    MapIn(input);
    GeneratePuzzle();
    MapOut(output);
}
```

- Solver

```
// Solver
vector<string> inputtext;
int row, col;
string line;

ifstream fin("output.txt");

getline(fin, line);
istringstream is(line);
is >> row >> col;

for (int i = 0; i < row; ++i)
{
    getline(fin, line);
    inputtext.push_back(line);
}

initHidato(inputtext, row, col);

cout << ("Puzzle Input:") << endl;
printPuzzle();

solveHidato(startPoint.first, startPoint.second, 1, 0);

cout << ("\nSolution Found:") << endl;
printPuzzle();
return 0;
```

3. Generator

- 핵심 함수
 - MapIn(ifstream in)
 - SetStartingPoint()

```
// 파일스트림을 통해 너비, 길이 및 벽 데이터 입력받습니다.
void MapIn(ifstream& input)
{
    input >> width >> height;

    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < height; j++)
        {
            input >> puzzle[i][j];
            //만약 1(숫자가 들어갈 수 있는 자리)이라면, 숫자 카운트 N 증가시킵니다.
            if (puzzle[i][j] == 1)
                N++;
            //방문했는지 아닌지를 표시하는 path 2차원 배열은 0으로 초기화 합니다.
            path[i][j] = 0;
        }
    }

    //arr은 입력해야하는 수의 sequence 범위를 저장하는 배열입니다.
    for (int i = 0; i <= N; i++)
        arr[i] = 1;

    //임의의 시작점을 설정하는 메소드를 호출합니다.
    SetStartingPoint();
}
```

3. Generator

- 핵심 함수

- GeneratePuzzle()
- Generate(int w, int h, point p, int count ,int num)

```
91 int Generate(int width, int height, point pos, int count, int N)
92 {
93     int i;
94     point next;
95
96     //N까지의 모든숫자가 퍼즐판에 들어갔다면 return 1
97     if (count == N)
98         return 1;
99
100     for (i = 0; i < 8; i++)
101     {
102         //next point는 주위 8방향으로 이동하였을 때 좌표입니다.
103         next.x = pos.x + direction[i].x;
104         next.y = pos.y + direction[i].y;
105
106         //next point는 width와 height를 벗어나서는 안되고, 해당 좌표의 path가 1이면 안되며(이미 방문한 칸이 아니며), 해당 좌표에는 벽이 없어야 합니다.
107         if (next.x >= 0 && next.x < width &&
108             next.y >= 0 && next.y < height &&
109             path[next.x][next.y] != 1 &&
110             puzzle[next.x][next.y] != 0)
111         {
112             //그렇다면 path에 방문했음을 표시합니다.
113             path[next.x][next.y] = 1;
114
115             // 다음에 들어가야 할 숫자의 arr 값이 0이라면 해당 카운트 숫자는 퍼즐판에 넣지않고 숫자가 들어가는 빈칸으로 남겨둡니다 (-1)
116             if (arr[count + 1] == 0)
117                 puzzle[next.x][next.y] = -1;
118
119             // 만약 1이라면 , 그 자리에는 해당하는 숫자를 채워넣어줍니다.
120             else
121                 puzzle[next.x][next.y] = count + 1;
122
123             //다음 next point에 대해 recursive하게 generate를 호출하고, true가 리턴되었다면 count == N이라는 뜻이므로 return 후 함수 종료합니다.
124             if (Generate(width, height, next, count + 1, N))
125                 return 1;
126
127             //위의 식에서 true가 리턴되지 않았다면 false라는 뜻이므로 이 path에 다시 방문할 수 있게 방문표시를 지워줍니다.
128             path[next.x][next.y] = 0;
129         }
130     }
131     return 0;
132 }
```

3. Generator

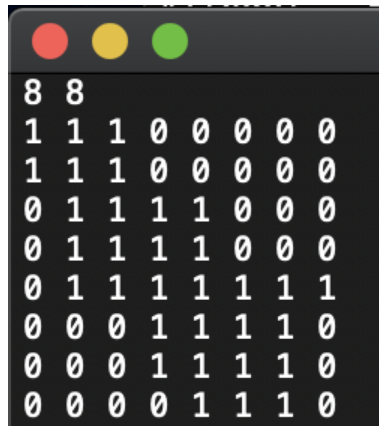
- 핵심 함수
 - MapOut(ofstream out)

```
134 void MapOut(ofstream& output)
135 {
136     // 완성된 맵 형식을 파일로 출력합니다.
137     output << width << ' ' << height << endl;
138     for (int i = 0; i < width; i++)
139     {
140         for (int j = 0; j < height; j++)
141         {
142             output << puzzle[i][j] << " ";
143         }
144         output << endl;
145     }
146 }
```


3. Generator

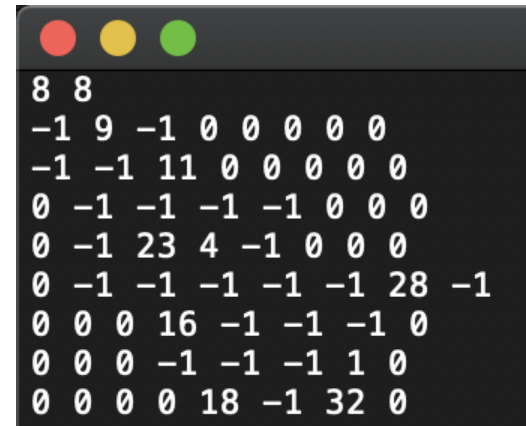
- 입력과 중간 결과물

- Generator에 왼쪽과 같은 형식의 input.txt를 넣는다.
- 오른쪽과 같은 output.txt를 만든다.
- output.txt는 Solver의 input이 된다.



```
8 8
1 1 1 0 0 0 0 0
1 1 1 0 0 0 0 0
0 1 1 1 1 0 0 0
0 1 1 1 1 0 0 0
0 1 1 1 1 1 1 1
0 0 0 1 1 1 1 0
0 0 0 1 1 1 1 0
0 0 0 0 1 1 1 0
```

- 8*8 input



```
8 8
-1 9 -1 0 0 0 0 0
-1 -1 11 0 0 0 0 0
0 -1 -1 -1 -1 0 0 0
0 -1 23 4 -1 0 0 0
0 -1 -1 -1 -1 -1 28 -1
0 0 0 16 -1 -1 -1 0
0 0 0 -1 -1 -1 1 0
0 0 0 0 18 -1 32 0
```

- 8*8 output

Main

- Main 함수
 - Generator

```
#include "HidatoSol.h"
#include "HidatoGen.h"
int main() {

    // Generator
    ifstream input;
    input.open("input.txt");
    ofstream output;
    output.open("output.txt");

    MapIn(input);
    GeneratePuzzle();
    MapOut(output);
```

- Solver

```
// Solver
vector<string> inputtext;
int row, col;
string line;

ifstream fin("output.txt");

getline(fin, line);
istringstream is(line);
is >> row >> col;

for (int i = 0; i < row; ++i)
{
    getline(fin, line);
    inputtext.push_back(line);
}

initHidato(inputtext, row, col);

cout << ("Puzzle Input:") << endl;
printPuzzle();

solveHidato(startPoint.first, startPoint.second, 1, 0);

cout << ("\nSolution Found:") << endl;
printPuzzle();
return 0;
```

4. Solver

- 핵심 함수

- `initHidato(vector<string>& input, int row, int col)`
- `PrintPuzzle()`
- `solveHidato(int r, int c, int n, int next)`

4. Solver

- solveHidato(int r, int c, int n, int next)

```
81 bool solveHidato(int r, int c, int n, int next)
82 {
83     //현재 숫자가 주어진 숫자의 max값보다 크면 탐색 종료
84     if (n > inputNum[inputNum.size() - 1])
85         return true;
86     //탐색하려는 곳이 0(빈칸)이 아니고 현재의 숫자가 아니라면 해당 칸으로는
87     //탐색 불가
88     if (hidatoBoard[r][c] != 0 && hidatoBoard[r][c] != n)
89         return false;
90     //탐색하려는 곳이 0(빈칸)이고, 주어진 숫자 벡터의 next번째가 현재 숫자라면
91     //탐색불가
92     if (hidatoBoard[r][c] == 0 && inputNum[next] == n)
93         return false;
94
95     //탐색하려는 곳을 lastPoint에 저장해둡니다
96     int lastPoint = hidatoBoard[r][c];
97     //탐색하려는 곳과 현재 숫자가 같다면 다음 탐색지점을 탐색.
98     if (lastPoint == n)
99         next++;
100    //해당 점에 숫자 표시
101    hidatoBoard[r][c] = n;
102    //8방향으로 위의 과정을 recursive하게 호출해서 확인합니다.
103    for (int i = -1; i < 2; i++)
104        for (int j = -1; j < 2; j++)
105            //퍼즐이 끝났다면 끝
106            if (solveHidato(r + i, c + j, n + 1, next))
107                return true;
108    //8방향중 아무곳으로도 갈 수가 없을때는 저장해둔 lastPoint로 돌아갑니다
109    hidatoBoard[r][c] = lastPoint;
110
111    return false;
112 }
```

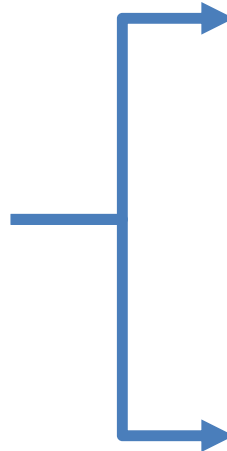
5. 실행 결과 및 평가

- Project Example

- Generator

- 난수 생성을 이용하므로,
매번 다른 퍼즐이 생성됨

- 5*5 input



	21		4	
18			7	1
		13		11

- 5*5 output(1)

	20	21	1	
18				
16		6	11	

- 5*5 output(2)

5. 실행 결과 및 평가

– Solver

	21		4	
18			7	1
		13		11

- 5*5 input(1)



20	21	3	4	5
19			2	6
18			7	1
17	14	8	9	10
15	16	13	12	11

- 5*5 output(1)

	20	21	1	
18				
16		6	11	

- 5*5 input(2)



19	20	21	1	2
18			3	4
17			5	10
16	13	6	11	9
14	15	12	7	8

- 5*5 output(2)

7x7 Puzzle Example



4						
		7				
		10	1	26	27	
12				20	23	
			15			



4	6					
5	3	7	8			
		2	9			
		10	1	26	27	
	11	18	19		25	
12	13	14	17	20	23	24
			15	16	21	22

Grid file:

```
7 7
1 1 0 0 0 0 0
1 1 1 1 0 0 0
0 0 1 1 0 0 0
0 0 1 1 1 1 0
0 1 1 1 0 1 0
1 1 1 1 1 1 1
0 0 0 1 1 1 1
```

Puzzle file:

```
7 7
4 -1 0 0 0 0 0
-1 -1 7 -1 0 0 0
0 0 -1 -1 0 0 0
0 0 10 1 26 27 0
0 -1 -1 -1 0 -1 0
12 -1 -1 -1 20 23 -1
0 0 0 15 -1 -1 -1
```

Solution file:

```
7 7
4 6 0 0 0 0 0
5 3 7 8 0 0 0
0 0 2 9 0 0 0
0 0 10 1 26 27 0
0 11 18 19 0 25 0
12 13 14 17 20 23 24
0 0 0 15 16 21 22
```

6. 참조 및 코드

- 다음 링크를 참조했습니다.
 - https://rosettacode.org/wiki/Solve_a_Hidato_puzzle