HomeWork_3

과목 : :컴퓨터알고리즘
분반 : 061
학과 : 정보컴퓨터공학부
학번 : 201824520
이름 : 안주현

1. 방법

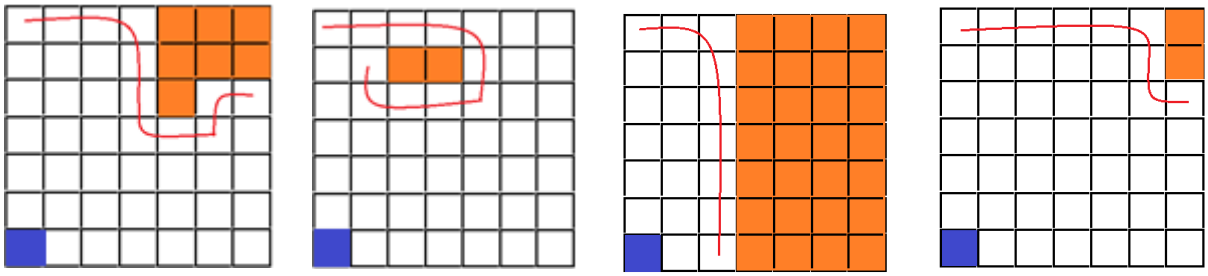
- 입력이 '?'인 경우
  U,R,D,L 순으로 유망한지 검사한 후에 유망하다면 이동한다.

- 입력이 U,R,D,L 중 하나인 경우
  입력된 문자가 유망한지 검사한 후에 유망하다면 이동한다.

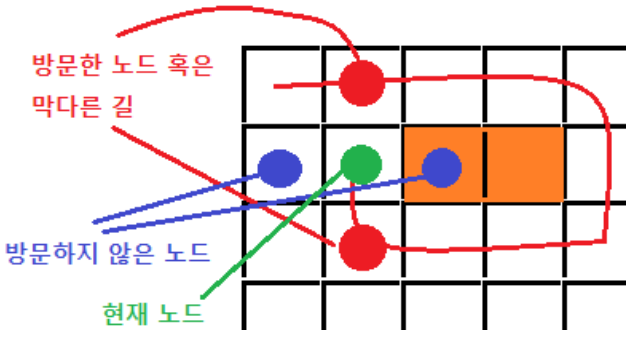
- 만약 현재 노드가 유망하지 않다면 진행을 멈춘다.

2. 가지치기

2-1. 방문하지 않은 곳이 둘러싸이는 경우(inner fragment와 비슷한 상태)



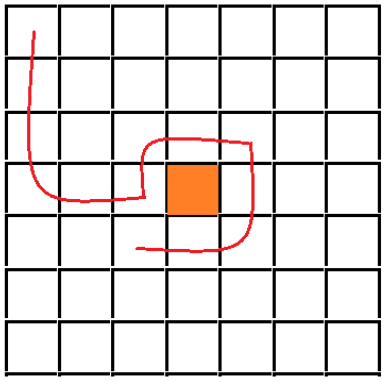
위 그림처럼 진행 중에 방문하지 않은 부분이 이미 방문한 길에 의해서 막히는 경우, 모든 칸을 방문하고 목적지에 도착하는 방법이 없으므로 멈춘다. 아래는 inner fragment의 탐지 방법에 대한 설명이다.

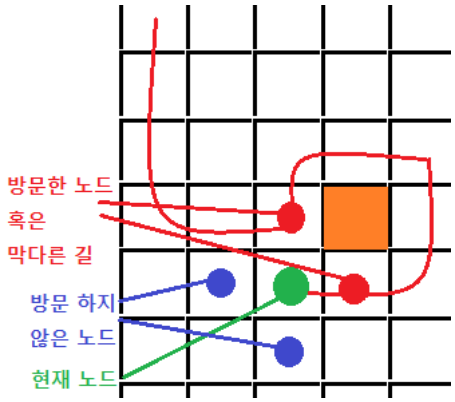


1. 현재 노드 기준, 상하 노드가 방문한 노드 혹은 막다른 길이고 좌우 노드가 방문하지 않은 노드
2. 현재 노드 기준, 좌우 노드가 방문한 노드 혹은 막다른 길이고 상하 노드가 방문하지 않은 노드

위의 두 경우 중에 하나라도 해당하면 inner fragment를 탐지할 수 있다.

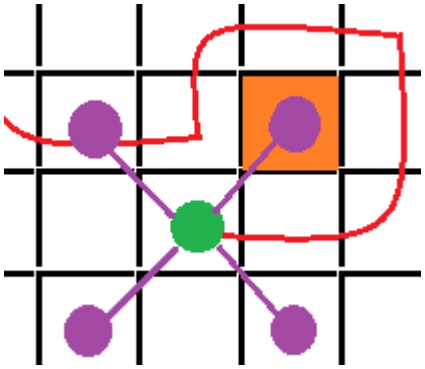
2-2. 2-1의 방법으로 탐지되지 않는 inner fragment





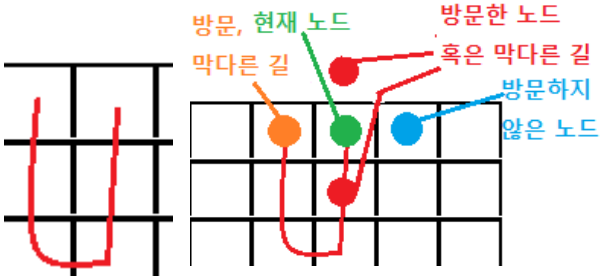
위 그림처럼 2-1의 방법으로 탐지되지 않는 경우가 있다.

아래는 이러한 경우를 탐지하는 방법에 대한 설명이다.



위 그림처럼 현재 노드에서 대각선 방향의 노드에 대하여 방문한 노드 혹은 막다른 길로 둘러싸였는지를 확인한다. 만약 둘러 싸였다면 진행을 멈춘다.

2-3. 2-1의 예외



위 그림처럼 둘러싸인 부분인 없지만 2-1의 방법에 의해서 inner fragment가 있다는 판단이 되는 경우가 생긴다. 아래는 이 경우를 탐지하는 방법에 대한 설명이다.

1. 현재 노드 기준, 2-1 방법으로 탐지된 경우 남은 두 노드 중 **한 노드**만 방문한 노드 혹은 막다른 길

이 경우에는 계속 진행해야 한다.

3. 코드

다음 입력의 유망성 검사

```
//현재 좌표 x,y 에서 입력 문자 ch 방향으로 이동이 유망한지 검사
bool promise(int x, int y, char ch, int (*matrix)[7])
{
        if(ch == 'U') { //위로 이동이 가능한지 검사
                if(matrix[x-1][y] || x-1<0) { return false; }
                else { return true; }
        }else if(ch == 'R') { //오른쪽으로 이동이 가능한지 검사
                if(matrix[x][y+1] || y+1>6) { return false; }
                else { return true; }
        }else if(ch == 'D') { //아래로 이동이 가능한지 검사
                if(matrix[x+1][y] || x+1>6) { return false; }
                else { return true; }
        }else if(ch == 'L') { //왼쪽으로 이동이 가능한지 검사
                if(matrix[x][y-1] || y-1<0) { return false; }
                else { return true; }
        }
}
```

가지치기

```
//2-1 의 1 번 경우
if((matrix[x-1][y] || x-1<0) && (matrix[x+1][y] || x+1>6)) {
        //2-3 의 예외가 아니면 진행 멈춤
        if(!(matrix[x][y-1]) && !(matrix[x][y+1])){
                index--; matrix[x][y]=0; return;
                }
        }
//2-1 의 2 번 경우
if((matrix[x][y-1] || y-1<0) && (matrix[x][y+1] || y+1>6)) {
        //2-3 의 예외가 아니면 진행 멈춤
        if(!(matrix[x-1][y]) && !(matrix[x+1][y])){
                index--; matrix[x][y]=0; return;
                }
        }
```

```
//2-2 의 방법
if(!(matrix[x-1][y+1]) && (x-1>=0 && y+1<=6)) { //우상대각 0 검사
        if((matrix[x-2][y+1] || (x-2<0)) && (matrix[x][y+1]) &&
            (matrix[x-1][y]) && (matrix[x-1][y+2] || (y+2>6))) {
               //해당되면 진행 멈춤
                index--; matrix[x][y]=0; return;
        }
}
if(!(matrix[x+1][y+1]) && (x-1>=0 && y+1<=6)) { //우하대각 0 검사
        if((matrix[x][y+1]) && (matrix[x+2][y+1] || (x+2>6)) &&
           (matrix[x+1][y]) && (matrix[x+1][y+2] || (y+2>6))) {
               //해당되면 진행 멈춤
                index--; matrix[x][y]=0; return;
        }
}
if(!(matrix[x+1][y-1]) && (x-1>=0 && y+1<=6)) { //좌하대각 0 검사
        if((matrix[x][y-1]) && (matrix[x+2][y-1] || (x+2>6)) &&
            (matrix[x+1][y-2] || (y-2<0)) && (matrix[x+1][y])) {
               //해당되면 진행 멈춤
                index--; matrix[x][y]=0; return;
        }
}
if(!(matrix[x-1][y-1]) && (x-1>=0 && y+1<=6)) { //좌상대각 0 검사
        if((matrix[x-2][y-1] || (x-2<0)) && (matrix[x][y-1]) &&
            (matrix[x-1][y-2] || (y-2<0)) && (matrix[x-1][y])) {
               //해당되면 진행 멈춤
                index--; matrix[x][y]=0; return;
        }
}



//모든 칸을 방문하고 목적지 도착(다음 입력 문자가 '#')
//cnt 를 1 증가시키고 그래프의 이전 단계로 back tracking
if(ch=='#' && x==6 && y==0) { cnt++; index--; matrix[x][y]=0; }

//모든 칸을 방문하지 않고 목적지 도착(다음 입력 문자가 '#'이 아님)
//그래프의 이전 단계로 back tracking
else if(ch!='#' && x==6 && y==0) { index--; matrix[x][y]=0; }
```

가지치기에서 걸리지 않은 경우, 탐색 진행

```
else if(ch == '?') { //입력 문자가 ? 인 경우
        //상,우,하,좌 순으로 유망성 검사 후에 이동
        for(int i=0; i<4; i++)
        {
                if(promise(x,y,dir[i], matrix)) {
                        index++;
                        switch(dir[i]) {
                                case 'U':
                                back_tracking(x-1,y,arr[index],matrix); break;
                                case 'R':
                                back_tracking(x,y+1,arr[index],matrix); break;
                                case 'D':
                                back_tracking(x+1,y,arr[index],matrix); break;
                                case 'L':
                                back_tracking(x,y-1,arr[index],matrix); break;
                        }
                }
        }
        //그래프의 이전 단계로 back tracking
        index--; matrix[x][y]=0;
}else { //입력 문자가 ?가 아닌 U,D,R,L 중에 하나인 경우
        if(promise(x,y,ch, matrix)) {
                index++;
                switch(ch) {
                        case 'U':
                                back_tracking(x-1,y,arr[index],matrix); break;
                        case 'R':
                                back_tracking(x,y+1,arr[index],matrix); break;
                        case 'D':
                                back_tracking(x+1,y,arr[index],matrix); break;
                        case 'L':
                                back_tracking(x,y-1,arr[index],matrix);break;
                }
        }
        //그래프의 이전 단계로 back tracking
        index--; matrix[x][y]=0;
}
```

main()

```
int main(void)
{
	cnt=0; index=0;

	//문자열 입력받기
	for(int i=0; i<48; i++)
	{
		cin >> arr[i];
	}
	arr[48]='#'; //문자열이 끝났음을 표시하는 문자 '#'

	int matrix[7][7] = {0}; // 7 X 7 행렬 선언

	//0,0 부터 입력 문자열에 따라 back tracking
	back_tracking(0,0,arr[index], matrix);

	cout << cnt;

	return 0;
}
```

4. 결과

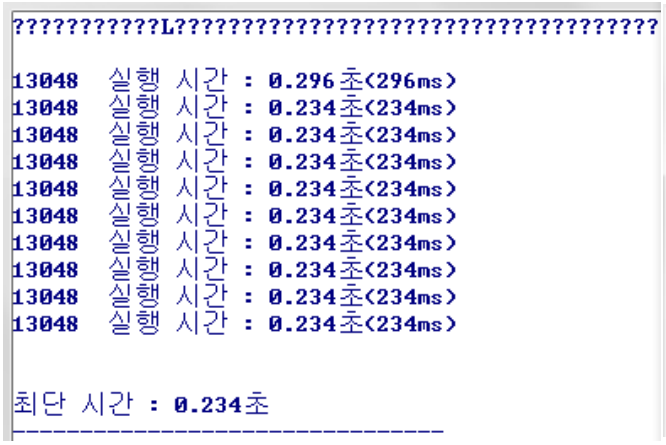
각 test_data 마다 10번의 back_tracking을 시행했고 각 시행마다 걸린 시간과 최단 시간을 출력했다.

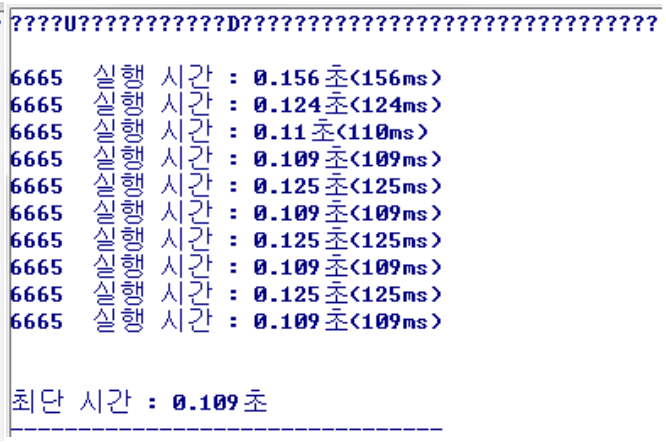
```
????D???L??D?L??????????????R??D???????????D??RD

0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)
0  실행 시간 : 0초(0ms)

최단 시간 : 0초
--------------------------------
```

```
????????????????????UL????U?L??L?D???RR??????????

1  실행 시간 : 0.078초(78ms)
1  실행 시간 : 0.047초(47ms)
1  실행 시간 : 0.046초(46ms)
1  실행 시간 : 0.032초(32ms)
1  실행 시간 : 0.046초(46ms)
1  실행 시간 : 0.047초(47ms)
1  실행 시간 : 0.031초(31ms)
1  실행 시간 : 0.047초(47ms)
1  실행 시간 : 0.031초(31ms)
1  실행 시간 : 0.031초(31ms)

최단 시간 : 0.031초
--------------------------------
```

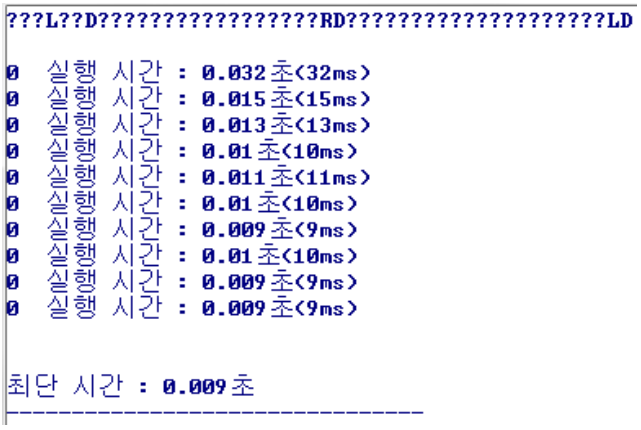
```
??????R??????U??????????????????????????LD????D?

201  실행 시간 : 0.109초(109ms)
201  실행 시간 : 0.094초(94ms)
201  실행 시간 : 0.078초(78ms)
201  실행 시간 : 0.078초(78ms)
201  실행 시간 : 0.078초(78ms)
201  실행 시간 : 0.078초(78ms)
201  실행 시간 : 0.078초(78ms)
201  실행 시간 : 0.078초(78ms)
201  실행 시간 : 0.078초(78ms)
201  실행 시간 : 0.078초(78ms)

최단 시간 : 0.078초
--------------------------------
```

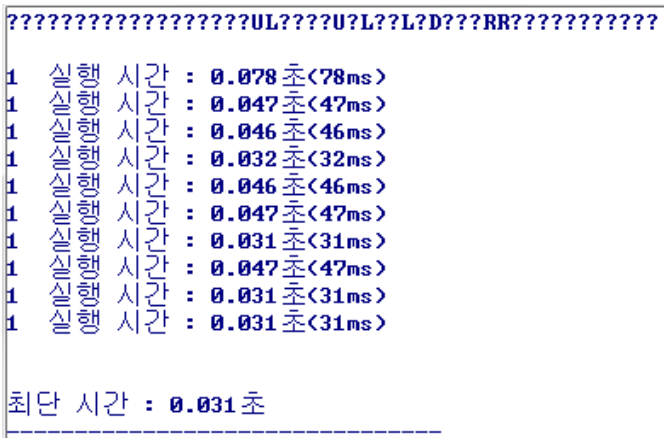
```
??????????????????????????????????????????????R?

0  실행 시간 : 1.763초(1763ms)
0  실행 시간 : 1.404초(1404ms)
0  실행 시간 : 1.404초(1404ms)
0  실행 시간 : 1.389초(1389ms)
0  실행 시간 : 1.435초(1435ms)
0  실행 시간 : 1.888초(1888ms)
0  실행 시간 : 1.903초(1903ms)
0  실행 시간 : 1.903초(1903ms)
0  실행 시간 : 1.903초(1903ms)
0  실행 시간 : 1.903초(1903ms)

최단 시간 : 1.389초
--------------------------------
```



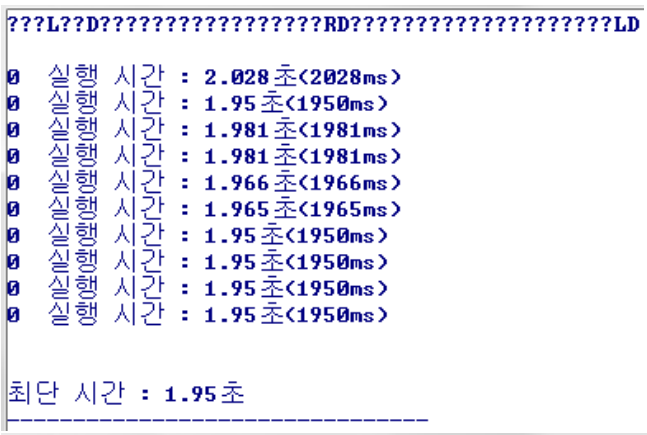


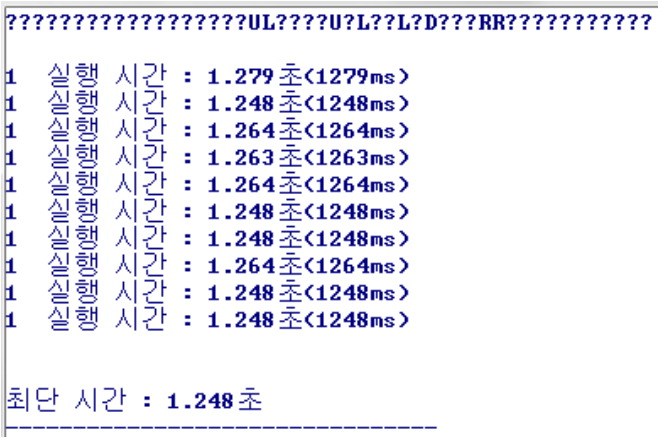
가지치기를 한 경우





가지치기를 안 한 경우





10 번의 시행이 거듭될수록 실행 시간이 단축되는 양상을 보여서 정확한 결과인지는 알 수 없지만 가지치기를 한 경우에 실행 시간이 비약적으로 단축됨은 알 수 있다.