스마트센서와 엑츄에이터

TEAM : 7조

이수빈
팀장
2019050982

노나현
2019073502

송영욱
2017012988

조용우 교수님
김민곤 조교님
2020.12.12

과제 1

알고리즘 설명

과제 1

알고리즘

1. **지도 행 우선 탐색**
   [출발점] → [도착점]
2. **지도 출력**
3. **최단 경로 복귀**
   [도착점] → [출발점]
4. **점수 출력**

1. 행 우선 탐색을 이용해 패치가 저장된 배열 s를 채운다.
2. 배열 s를 이용해 지도를 출력한다.
3. 최고 점수를 저장하는 배열 dt를 만든다.
4. 배열 dt를 바탕으로 최단 경로로 복귀한다.
5. 4번 과정 중 계산된 점수를 출력한다.

과제 1

코드 설명

completeSearch()

printgraph()

1. 행 우선 탐색을 이용해 패치가 저장된 배열 s를 채운다.
2. 배열 s를 이용해 지도를 출력한다.

S[5][5]

0	1	0	1	0
0	-1	0	-1	-1
0	1	0	0	0
0	0	-1	-1	1
0	0	0	1	0

-1 이면 x

1 이면 o

0 패치가 없으면 +

```
void printgraph()
{
    char c;
    for (int y = 0; y<5; y++) {
        for(int x = 0; x<5; x++){
            if (S[y][x] == -1) c = 'X';
            else if(S[y][x] == +1 ) c = 'O';
            else c = '+';

            displayStringAt( (x+1) * 10 , 100 - (y+1)*10, "%c", c);
        }
    }
    sleep(10000);
}
```

행 단위로 읽어 나간다.

한 행의 열 전체 탐색

Y=0

0	1	0	1	0
0	-1	0	-1	-1
0	1	0	0	0
0	0	-1	-1	1
0	0	0	1	0

y좌표

+

디스플레이

이

왼쪽 상단부터 0행을 출력해준다.

(0,0)

(x,y)

x좌표

+

```
task main()
{
    while(getButtonPress(1) == 0) sleep(10);

    completeSearch();
    printgraph();

    setMotorSpeed(lm, 0);
    setMotorSpeed(rm, 0);
    sleep(1000);

    setMotorSpeed(lm, 20);
    setMotorSpeed(rm, 20);
    sleep(1000);
    setMotorSpeed(lm, 20);
    setMotorSpeed(rm, -20);
    sleep(400);
    turnRight();

    count = row = 0;

    for(int i=0;i<5;i++)
       for(int j=0;j<5;j++)
       {
          if(i==0 && j==0)dt[i][j]=S[i][j];
          else if(i==0) dt[i][j]=dt[i][j-1]+S[i][j];
          else if(j==0) dt[i][j]=dt[i-1][j]+S[i][j];
          else dt[i][j]=max(dt[i-1][j], dt[i][j-1])+S[i][j];
       }

    row=0; r=c=4;
    while(r!=0 || c!=0)
    {
       if(r==0) goLeft();
       else if(c==0) goUp();
       else if (dt[r-1][c] > dt[r][c-1]) goUp();
       else goLeft();
      result += S[r][c];
    }
```

탐색 끝

탐색을 끝낸 후, 잠시 정지한다.

돌아가기 위해 유턴한다.

dt배열 완성

dt배열 채우기

3. 최고 점수를 저장하는 배열 dt를 만든다.

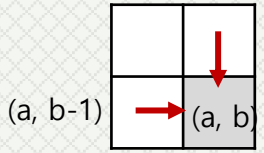
dt[a][b] = (0, 0)에서 출발하여 (a, b)까지 이동하여 얻을 수 있는 최대 패치의 수

0	1	1	2	2
0	0	1	1	1
0	1	1	1	1
0	1	0	0	2
0	1	1	2	2

if dt[0][0] = 시작점. 해당 위치 패치 수 S[0][0]

else if 시작점(0,0)이 아니고 0행이거나 0열 일 때,
이전 위치는 이전 열 or 이전 행이다.

else dt[a][b] = max(dt[a-1][b] , dt[a][b-1]) + S[a][b]



; 최단 경로 이동이므로 위치 (a,b)의 이전 위치는 (a-1, b), (a, b-1) 두 곳 뿐
최대 패치를 얻기 위해 큰 두 위치 중 큰 값에서 오도록 한다.

```
    displayBigTextLine(5,"final score is %d",result);
    stopMotor();
    while(getButtonPress(1)==0) sleep(10);
}
```

```
task main()
{
    while(getButtonPress(1) == 0) sleep(10);

    completeSearch();
    printgraph();

    setMotorSpeed(lm, 0);
    setMotorSpeed(rm, 0);
    sleep(1000);

    setMotorSpeed(lm, 20);
    setMotorSpeed(rm, 20);
    sleep(1000);
    setMotorSpeed(lm, 20);
    setMotorSpeed(rm, -20);
    sleep(400);
    turnRight();

    count = row = 0;

    for(int i=0;i<5;i++)
       for(int j=0;j<5;j++)
       {
          if(i==0 && j==0)dt[i][j]=S[i][j];
          else if(i==0) dt[i][j]=dt[i][j-1]+S[i][j];
          else if(j==0) dt[i][j]=dt[i-1][j]+S[i][j];
          else dt[i][j]=max(dt[i-1][j], dt[i][j-1])+S[i][j];
       }

    row=0; r=c=4;
    while(r!=0 || c!=0)
    {
       if(r==0) goLeft();
       else if(c==0) goUp();
       else if (dt[r-1][c] > dt[r][c-1]) goUp();
       else goLeft();
      result += S[r][c];
    }
```

최단경로로 복귀 및 점수 계산

최단 경로 복귀

4. 배열 dt를 바탕으로 최대 점수를 얻는 최단 경로로 주행하여 복귀한다.

dt[5][5]

0	1	1	2	2
0	0	1	1	1
0	1	1	1	1
0	1	0	0	2
0	1	1	2	2

행번호가 0일 때, 행 방향 이동 불가 goLeft()

열번호가 0일 때, 열 방향 이동 불가 goUp()

최단 경로로 가기 위해,

행 또는 열이 감소하는 방향으로만 이동한다.

현재 위치 (r, c) 에서 갈 수 있는 위치는 (r-1, c), (r, c-1) 뿐

두 위치 중, 최대 점수가 높은 위치로 이동

dt[r-1][c]가 더 클 때, goUp()

dt[r][c-1]가 더 클 때, goLeft()

result 이동한 위치의 패치 점수를 계속 갱신해준다.

점수 출력

5. 4번 과정으로 최종 계산된 점수를 출력한 result 다.

```
    displayBigTextLine(5,"final score is %d",result);
    stopMotor();
    while(getButtonPress(1)==0) sleep(10);
}
```

점수 출력 후 멈춘다

과제 1 -> 과제 2

0	1	0	1	0
0	-1	0	-1	-1
0	1	0	0	0
0	0	-1	-1	1
0	0	0	1	0

0	5	0	5	0
0	-3	0	-3	-3
0	5	0	0	0
0	0	-3	-3	5
0	0	0	5	0

과제 2

알고리즘 설명

과제 2

알고리즘

1. **지도 행 우선 탐색**
   [출발점] → [도착점]
2. **지도 출력**
3. **최고 득점 경로 복귀**
   [도착점] → [출발점]
4. **점수 출력**

1. 행 우선 탐색을 이용해 패치가 저장된 배열 s를 채운다.
2. 배열 s를 이용해 지도를 출력한다.
3. 최고 점수 경로를 따라 복귀한다.
4. 4번 과정 중 계산된 점수를 출력한다.

알고리즘 설명

	0	1	2	3	4
0		(0,1)		(0,3)	
1		(1,1)		(1,3)	(1,4)
2		(2,1)			
3			(3,2)	(3,3)	(3,4)
4				(4,3)	

1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

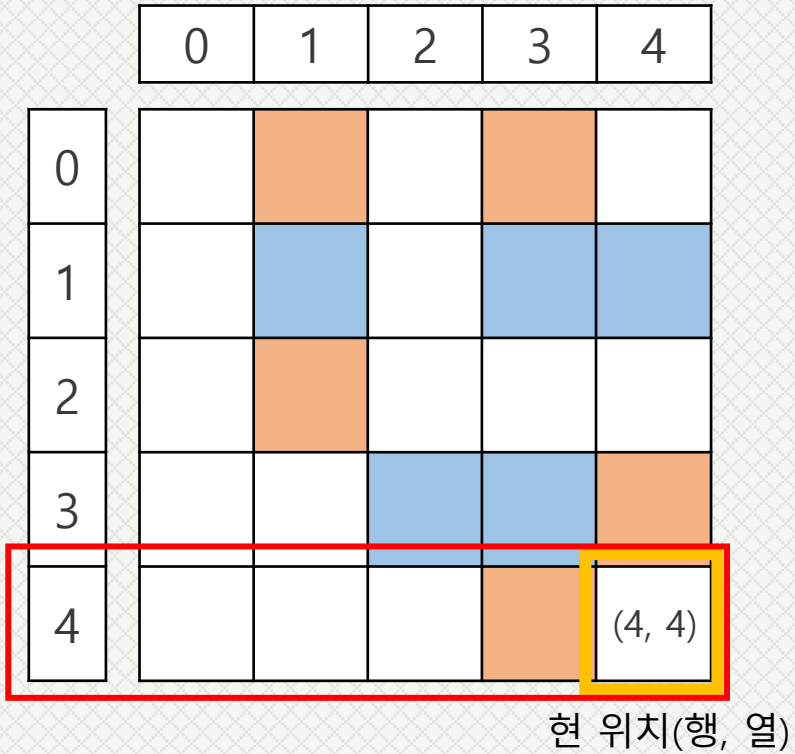
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

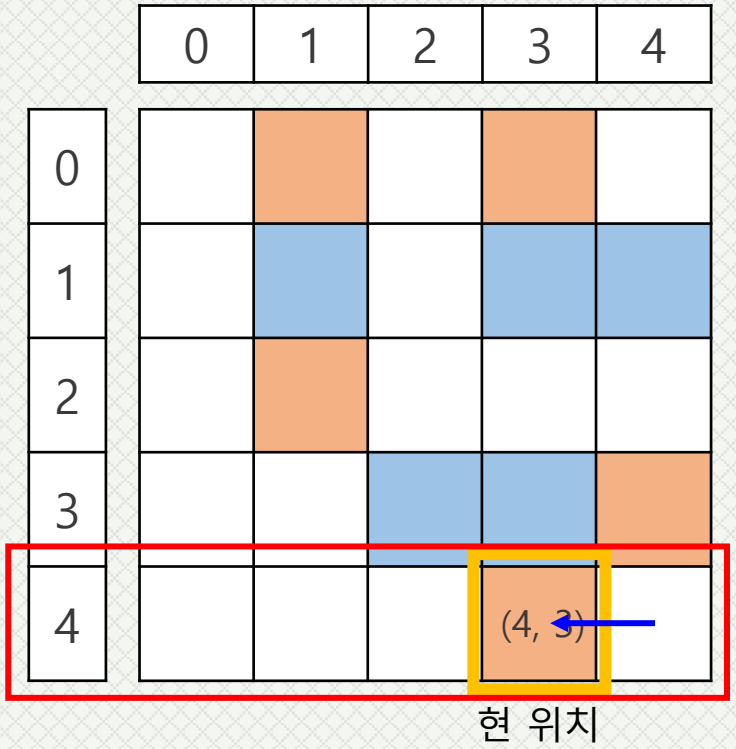
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 **빨간 패치**부터 **방문**
   2-1. 없다면 바로 위에 행 탐색

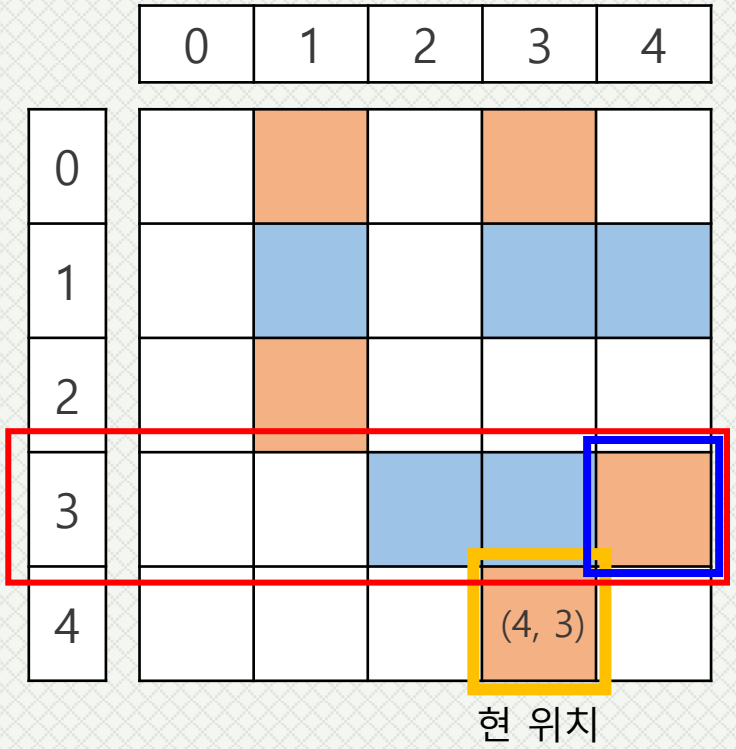
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

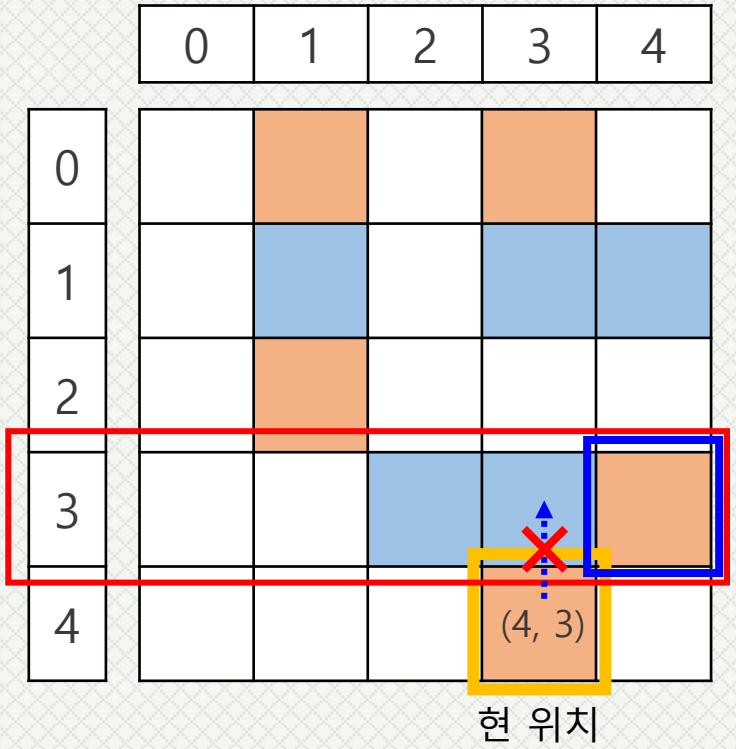
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

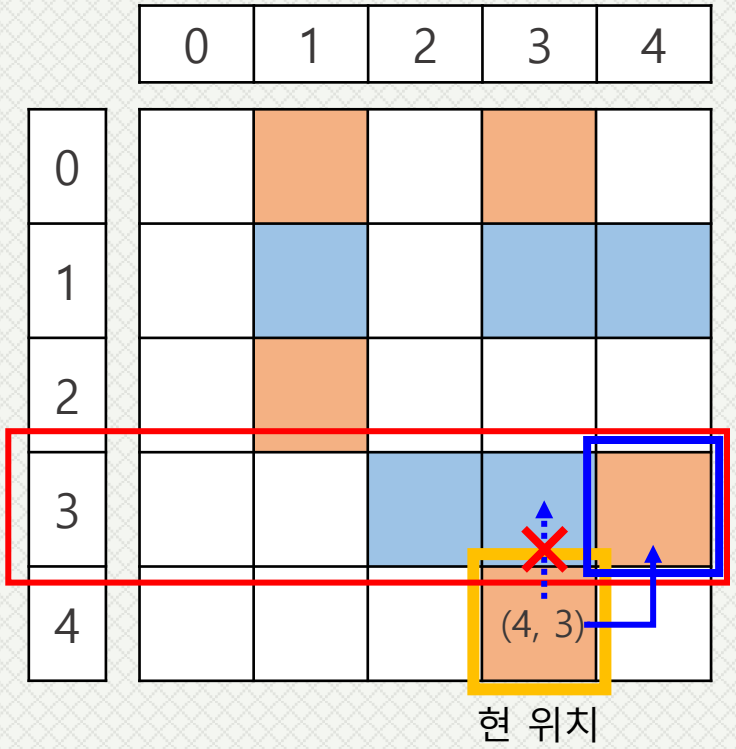
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

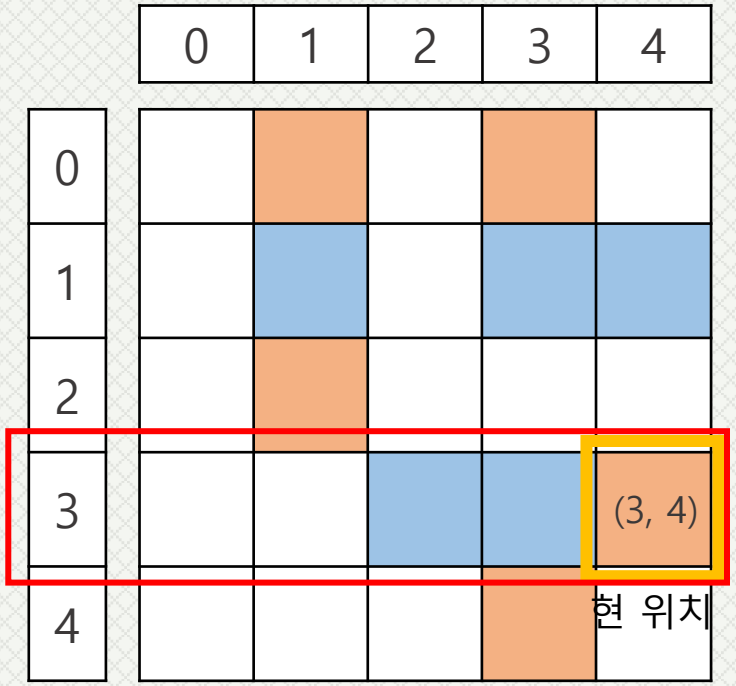
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 **파란 패치가 있다면**, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, **현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)**

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

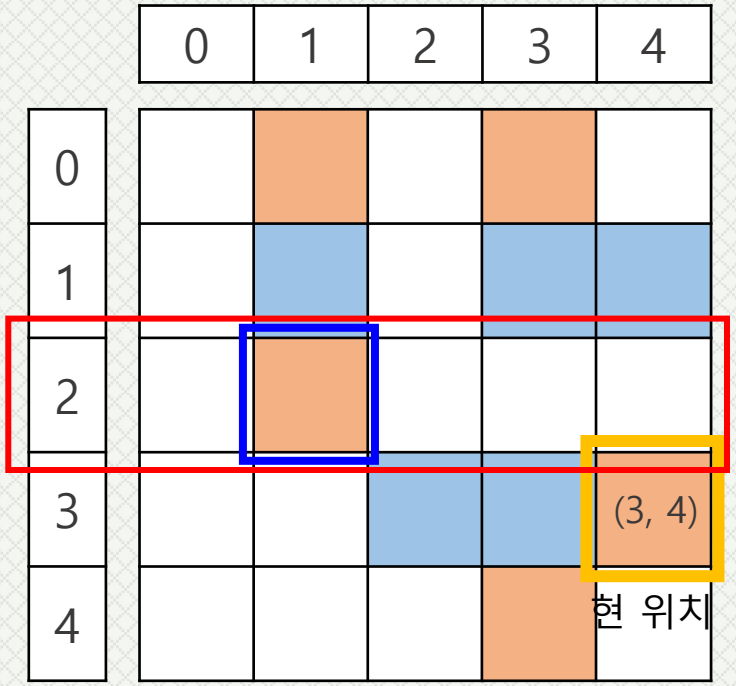
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

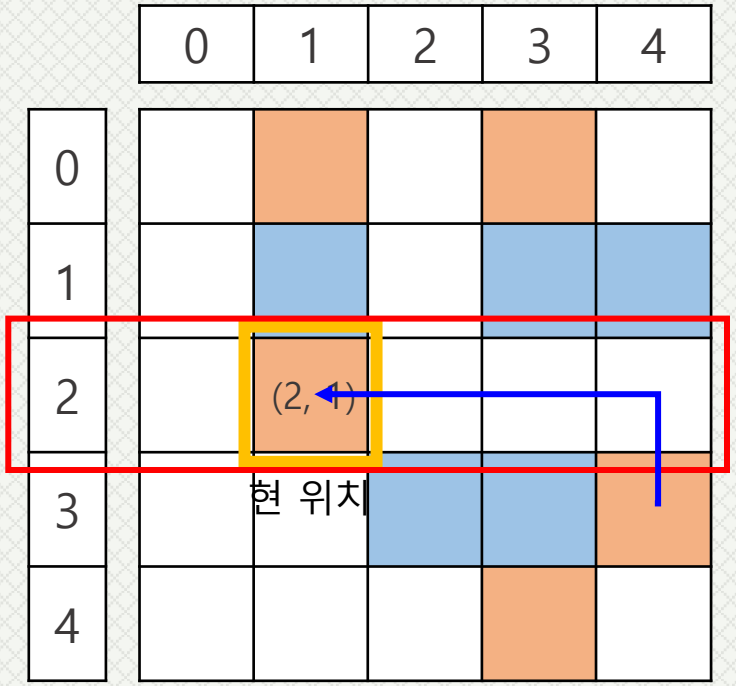
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남 (이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

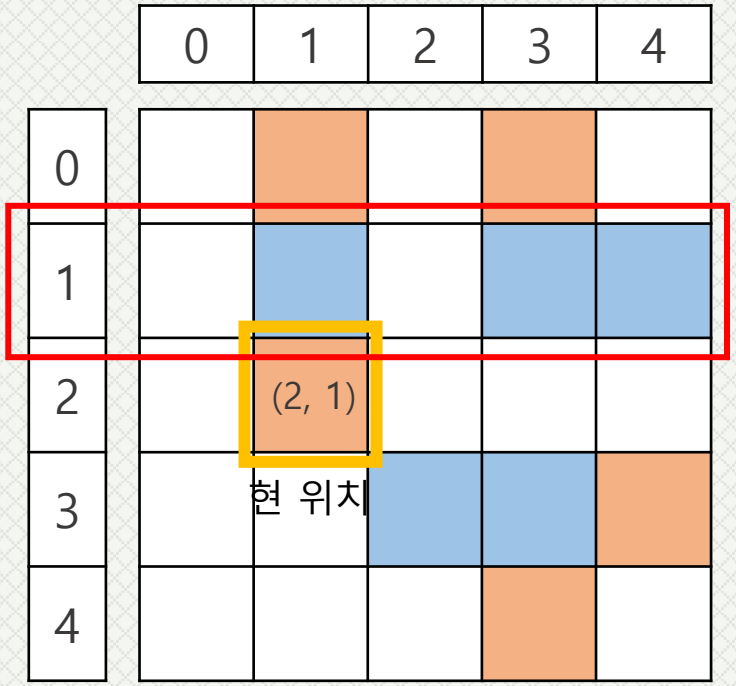
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

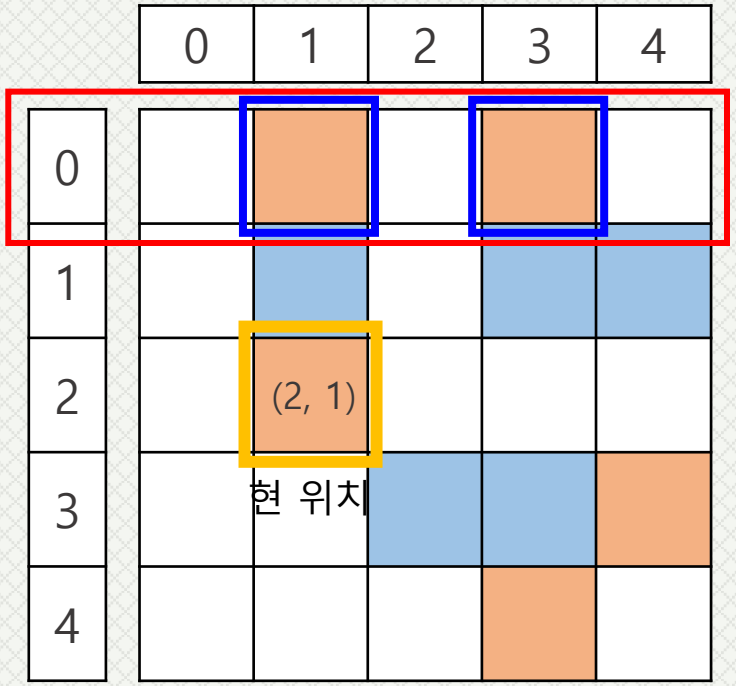
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

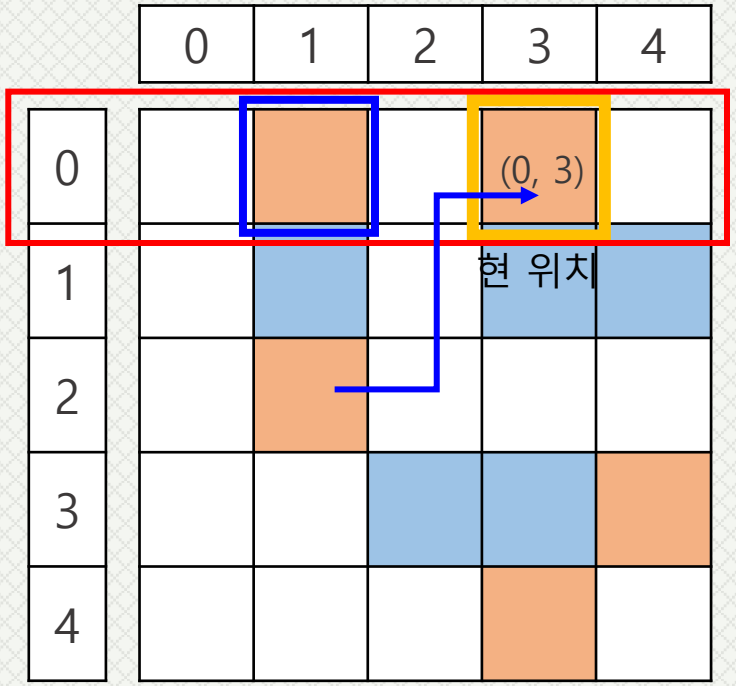
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

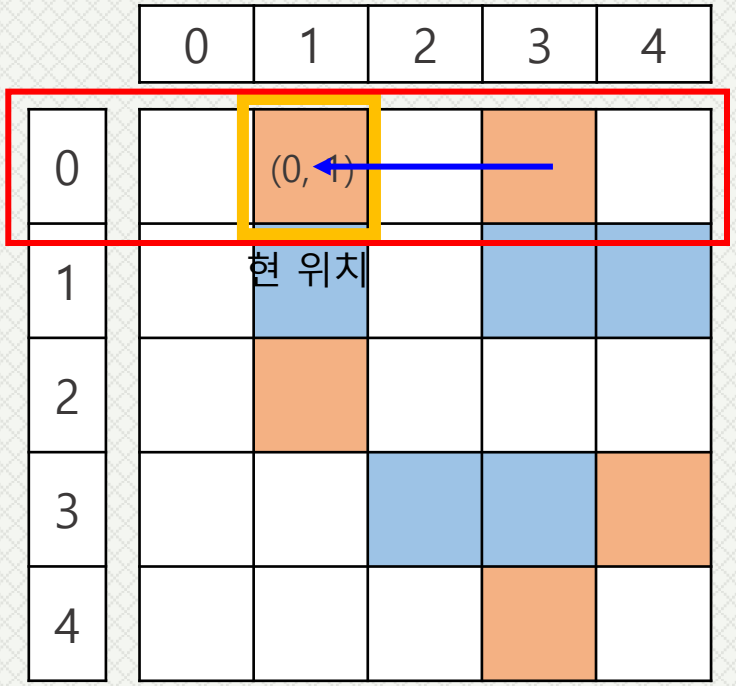
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남 (이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

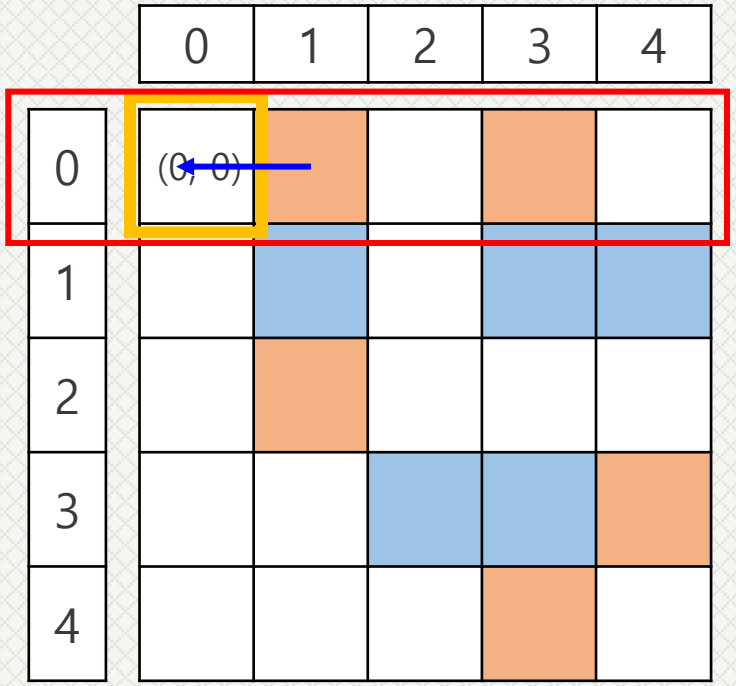
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남 (이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

알고리즘 설명



1. 현재 위치의 행 탐색

2. 빨간 패치가 있다면 가까운 빨간 패치부터 방문
   2-1. 없다면 바로 위에 행 탐색

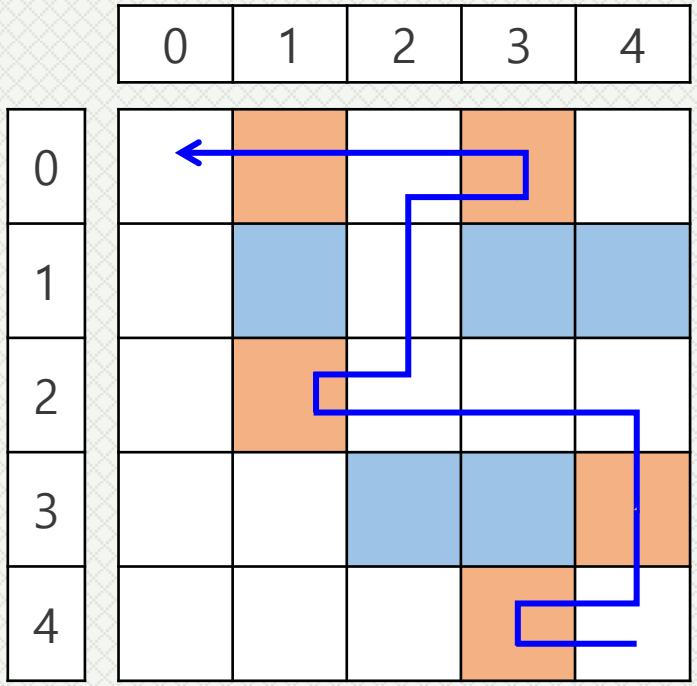
3. 목표 위치(빨간 패치)로 이동 (대각 선 위치 시 이동 방향은↑ ↓ ← → 순서)
   3-1. 만약 이동경로(위)에 파란 패치가 있다면, 왼쪽 혹은 오른쪽 방향으로 이동하여 벗어남
(이때, 현재 위치에서 가장 가까운 쪽의 방향으로 한 칸 이동(왼쪽 or 오른쪽)

4. 1~3의 과정을 계속 반복

5. 행이 0인경우는 마지막 행이므로, 도착점에서 가장 먼 곳부터 방문 하도록 한다.
(최소의 경로로 최대한 많이 얻는 것이 이득이기 때문이다.)

6. 도착점(0,0)을 향해 바로 나아간다.

최종 점수



총 방문횟수 14

빨간 패치 5개 (재방문 0)

파란 패치 0개

(-14) + (+5) * 5 + (-3) * 0

총점 = 11

과제 2

코드 설명

코드 설명

```
void printgraph()
{
    char c;
    for (int y = 0; y<5; y++) {
        for(int x = 0; x<5; x++){
            if (S[y][x] == -3) c = 'X';
            else if(S[y][x] == +5 ) c = 'O';
            else c = '+';

            displayStringAt( (x+1) * 10 , 100 - (y+1)*10, "%c", c);
        }
    }
    sleep(10000);
}
```

과제 1 과 동일한 printgraph()

빨간 패치와 파란 패치의 값이 변경됨에 따라 이에 맞춰 X와 O를 출력한다

코드 설명

```
int make_9()
{
   for(int i=0;i<5;i++)
   {
      red[i]=9;
   }
   return 0;
}
```

red배열에 모든 요소를 9로 초기화해주는 함수

Red배열은 현재 행에서 빨간 패치가 있다면 그 위치와 현재 위치와의 거리를 넣어줄
처음에 모든 배열을 충분히 큰 값인 9로 초기화 해준다.

```
int not_nine_index() //오른쪽 부터 검사해 9가 아닌 인덱스 출력
{
   int ind=0;
   for (int i=4;i>=0;i--)
   {
      if(red[i]!=9)
      {
         ind=i;
         return ind;
      }
   }
   return ind;
}
```

1행일 때 도착점에서 더 멀리 있는 인덱스를 리턴 해주는 함수

맨 위 0번째 행의 빨간 패치와 현재 위치사이의 거리가 들어있는 red배
오른쪽 열에서부터 하나씩 검사하여 가장 가까운 빨간 패치가 있는
위치 인덱스를 Ind 변수에 할당하여 리턴 한다.

코드 설명

```
int min_list_index() //red
{
  int min=red[0];
  int index=0;
  for(int i=1;i<5;i++)
  {
    if(red[i]<min)
    {
      min=red[i];
      index=i;
    }
  }
  return index;
}
```

가장 가까운 빨간 패치의 index값을 찾기 위한 함수

첫 번째 값을 최솟값 변수 min에 할당함 (red배열은 최대값이 9)

Red배열안의 인덱스 값들을 최솟값 변수 min과 비교하며
현재 위치 기준 제일 짧은 거리의 빨간 패치 위치를 갱신한다.
최종적으로 제일 작은 인덱스를 index변수에 할당한다.

Int Index를 return하여 현재 행과 최소 거리 빨간 패치의 위치를 알린

```
void gogo()
{
  while (x!=X || y!= Y)
  {
    if(S[X-1][Y]==-3)//만약 위에 파란패치있다면
      {//다음패치에 가까운 열로 한칸 이동해서 똑같은 곳을 다른 경로로 감
        if(Y<=y)
        {
           goRight();
           Y++;
           result -=1;
        }
        else
        {
           goLeft();
           Y--;
           result -=1;
        }
     }
    else if (x<X){ //when go up
         goUp();
         X--;
         result -=1;
    }
    else if(x>X){
         goDown();
         X++;
         result -=1;
    }
    else if(y<Y){//when go left
         goLeft();
         Y--;
         result -=1;
    }
    else if(y>Y){//when go right
         goRight();
         Y++;
         result -=1;
    }
  }
```

가야하는 위치(x,y)까지 이동하는 함수

가야할 위치의 index (x,y) 현재 위치 index(X,Y)로 할당
가야하는 위치에 도달 하기 전까지 계속 조건문을 실행하여 이동함

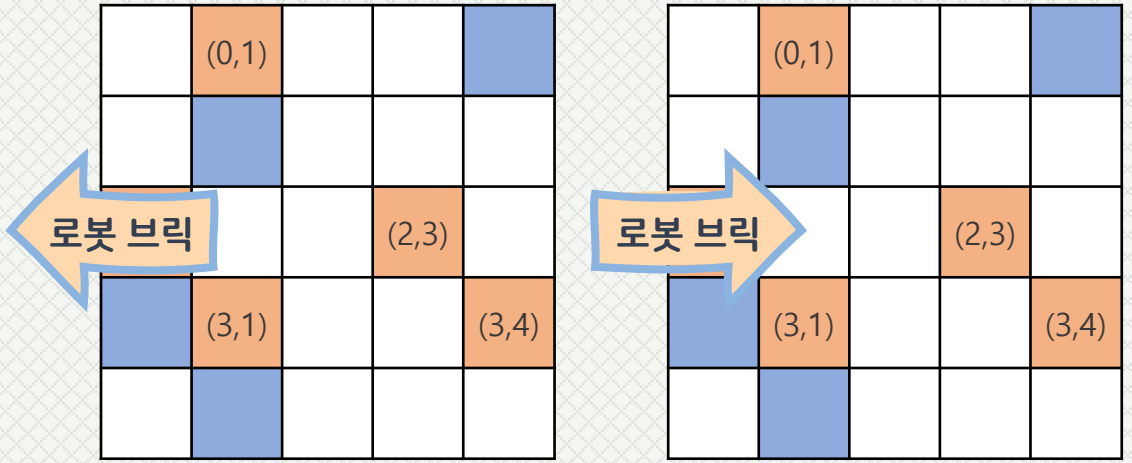
만약 이동 경로(위)에 파란 패치가 있다면 기존의 이동 순서↑ ↓ ← →
에서
가야하는 위치(x,y)와 가장 가까운 열로 한 칸 이동하여 똑같은 목표를 향해 이동한다.

행이 차이가 날 경우, goUp() goDown()함수를 호출하여 이동하고
현재 위치 index 변수를 조정하여 위치를 갱신 시킨다.
칸이 이동할 때 마다 최종 득점 변수 result에서 -1 함

열이 차이가 날 경우도 동일하게 작용한다.

코드 설명

```
void goRight() //I modify opposite of goLeft()
{
    c++;
    count = 0;
    turnRight();
    setMotorSpeed(lm,0);
    setMotorSpeed(rm,0);
    sleep(100);
    //to call turnRight 1 time we turn 90 degree(My thought)
    turnRight();
    //so we call one more time to turn 90 degree more
    while(true)
    {
        go();
        if(count == 1)
        {
            setMotorSpeed(lm, 30);
            setMotorSpeed(rm, 30);
            sleep(400);
            break;
        }
    }
}
```



goRight함수

로봇 브릭이 오른쪽으로 가기 위해서 필요한 함수

오른쪽 방향으로 총 180도 회전하여 직진해야 하므로

turnRight함수를 호출함

```
task main()
{

  while(getButtonPress(1) == 0) sleep(10);
   completeSearch();
   setMotorSpeed(lm, 0);
   setMotorSpeed(rm, 0);
   sleep(1000);
   printgraph();

   setMotorSpeed(lm, 20);
   setMotorSpeed(rm, 20);
   sleep(1000);

   setMotorSpeed(lm, 20);
   setMotorSpeed(rm, -20);
   sleep(400);
   turnRight();

   count = row = 0;

   make_9();
   int cnt=0;
   for (int rr =4; rr >=0 ; rr--)
   {
     for(int cc=4;cc>=0;cc--)
     {
       if(S[rr][cc]==5)
       {
         red[cc]=abs(Y-cc);
         cnt++;
       }
     }
```

버튼을 누르면 시작

행 우선 탐색으로 패치를 S[][]배열에 값을 더하고,

그에 맞는 graph를 디스플레이에 출력

다시 출발했던 최종 도착지점으로 가기 위해,

잠깐 멈추고 U턴함

```
int abs(int a)
{
  if(a<0)
  {
    a=a*-1;
  }
  return a;
}
```

절대값 함수

행을 스캔하고 만약 빨간 패치가 존재한다면 그 index를 red배열에 현재 위치와 차이의 절대 값을 red배열에 저장한다.

한 행에 몇 개의 빨간 패치가 있는지 cnt로 세아린다. 이는 해당 행에서 몇번의 gogo()를 호출해야하는 지 알려준다.

```
    while(cnt>0)
    {
      int ind=min_list_index();
      x=rr;
      if(x==0) //마지막 행에 패치가 2개 이상있다면 도착지점보다 멀리있는 패치를 먼저 방문해야 이득이다.
      {
        ind=not_nine_index();
        y=ind;
      }
      else
      {
        y=ind;
      }
      gogo();

      result +=S[X][Y];
      red[ind]=9;
      S[x][y]=-3;

      cnt--;
      totalred--;
      if(totalred==0)
      {
        x=0;
        y=0;
        gogo();
      }

    }
  }

  displayBigTextLine(5,"final score is %d",result);
  stopMotor();
  while(getButtonPress(1)==0) sleep(10);
}
```

마지막 행에 빨간 패치가 2개 이상 있다면 도착점으로부터 멀리 떨어져 있는 패치를 먼저 거치고 도착점을 가는 것이 이득이다.

따라서, not_nine_index함수를 호출해 더 멀리 있는 빨간 패치의 위치를 먼저 할당한다.

한 행에 빨간 패치 개수 변수 인 cnt가 0이 될 때 까지 가장 가까운 red 패치 index를 변수 ind에 할당하고, 가야하는 패치의 위치를 (x,y)로 설정한다.

Gogo 함수를 호출하여 해당 위치까지 이동한다.

이동한 위치의 패치 점수를 최종 점수 result 변수에 계속 갱신한다.

이동한 후 패치 index라는 사실을 알려주기 위해 red배열에 9로 초기화 한다.

한 번 지나간 빨간 패치 자리는 -3으로 바꾼다.

Cnt변수와 총 빨간 패치 개수를 알리는 totalred변수에 -1씩 차감 시켜 현재 한 행에 몇 개의 빨간 패치가 남았고 총 가야하는 빨간 패치 수를 갱신한다.

만약 빨간 패치를 다 다녀왔다면, 가야하는 위치 index를 (0,0)으로 설정하여 마지막 도착지까지 이동시킨다.

최종 득점 점수 변수 result 값을 displayBigTextLine으로 디스플레이에 출력함 모터를 멈추고 프로그램이 마무리 된다.

끝!