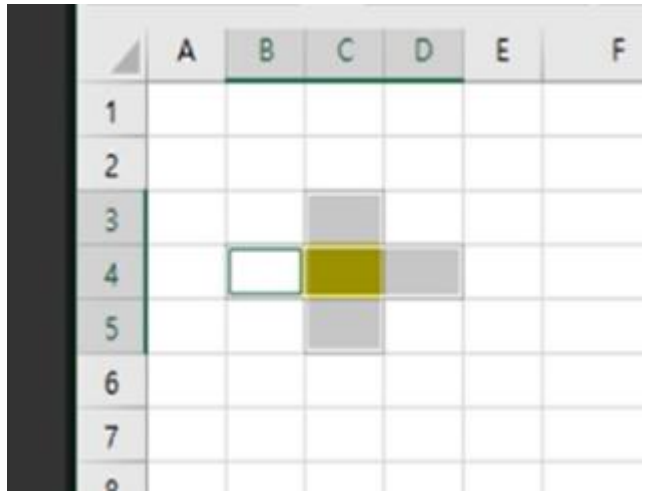
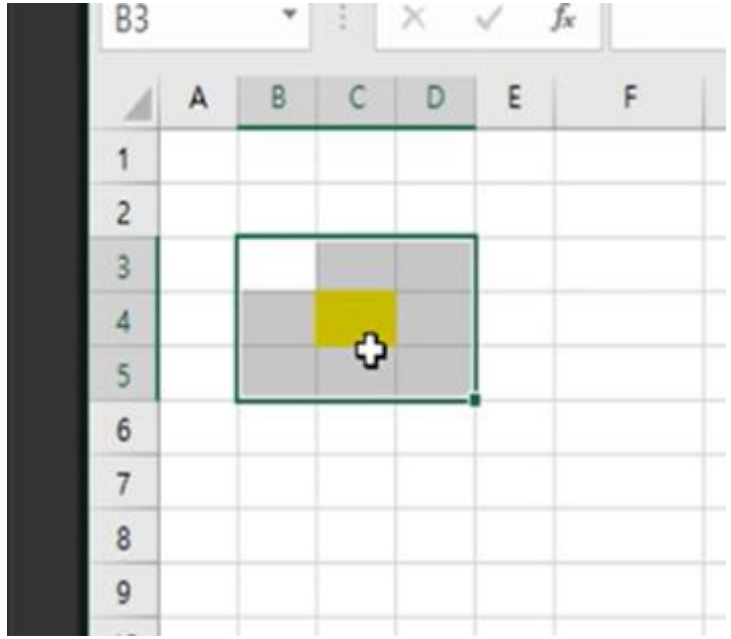



4방향 구조요소 모양



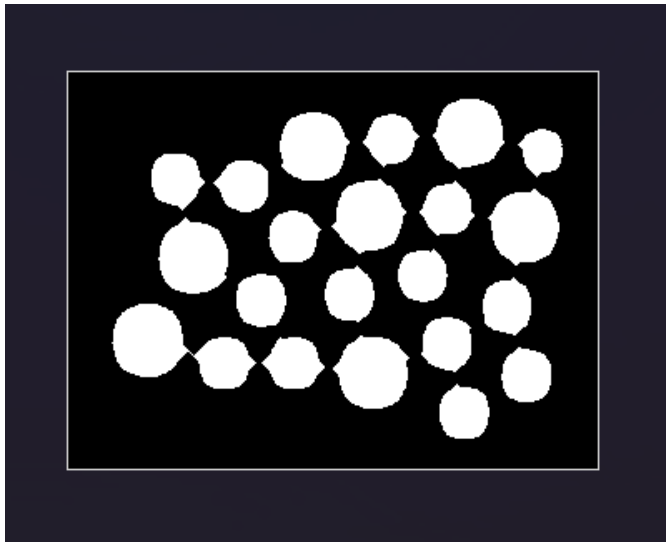

8방향 구조요소 모양

대각선 등등 다양한 구조요소가
나올 수 있지만 이 2개가 주로 쓰인다

# 침식

```
538 void Erosion(BYTE* Image, BYTE* Output, int W, int H)
539 {
540     //모폴로지 연산은 구조 요소이다
541     //구조 요소란? => ppt
542     for (int i = 1; i < H - 1; i++)
543     {
544         for (int j = 1; j < W - 1; j++)
545         {
546             //흰색이 전경이고 블랙이 배경
547             if (Image[i * W + j] == 255)
548             {
549                 //전경화소처리 => 전경화소라면
550                 //주변을 확인해서(구조요소 체크) 하나라도 배경이면 배경으로 지정
551                 //모두가 전경화소인지 확인
552                 //하나라도 전경 화소가 아니면 Output의 i*W+j를 0 즉 배경으로 바꿔줘야해, 아니면 255
553                 if (!(Image[(i - 1) * W + j] == 255 &&
554                     Image[(i + 1) * W + j] == 255 &&
555                     Image[i * W + (j - 1)] == 255 &&
556                     Image[i * W + (j + 1)] == 255)
557                     )// 위 아래 좌 우 순서로확인
558
559                     Output[i * W + j] = 0; //
560
561                 else Output[i * W + j] = 255;
562             }
563             //i*W+j가 배경 화소면 신경쓸필요없이 그대로 0처리 신경쓸 필요도 없음
564             else Output[i * W + j] = 0;
565         }
566
567     }
568 }
```

"="개수에 유의하기

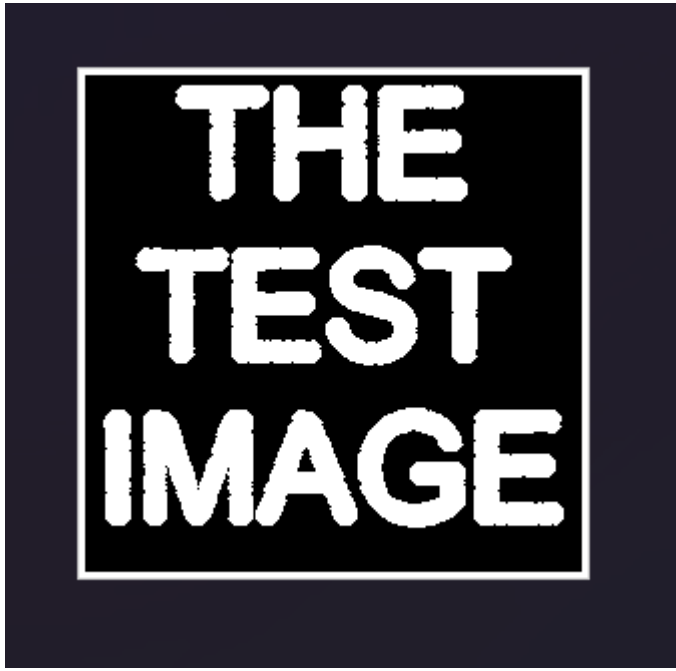


```
731
732     Erosion(Image, Output, W, H);//침식은 한 번으로 끝나지 않는다
733     Erosion(Output, Image, W, H);
734     Erosion(Image, Output, W, H);
```

But 원래의 크기도 줄어든다는 단점이 있어
So 침식 후 팽창인 opening을 써줘야해

팽창

```
570 void Dilation(BYTE* Image, BYTE* Output, int W, int H)
571 {
572     //모폴로지 연산은 구조 요소이다
573     //구조 요소란? => ppt
574     for (int i = 1; i < H - 1; i++)
575     {
576         for (int j = 1; j < W - 1; j++)
577         {
578             //흰색이 전경이고 블랙이 배경
579             if (Image[i * W + j] == 0) //배경화소라면
580             {
581                 //배경화소처리 => 배경화소라면
582                 //주변을 확인해서(구조요소 체크) 하나라도 전경이면 전경으로 지정
583                 //모두가 배경화소인지 확인
584                 //하나라도 전경 화소면 Output의 i*W+j를 255 즉 전경으로 바꿔줘야해, 아니면 0
585                 if (!(Image[(i - 1) * W + j] == 0 &&
586                     Image[(i + 1) * W + j] == 0 &&
587                     Image[i * W + (j - 1)] == 0 &&
588                     Image[i * W + (j + 1)] == 0)
589                     )// 위 아래 좌 우 순서로확인
590
591                     Output[i * W + j] = 255; //
592
593                 else Output[i * W + j] = 0;
594             }
595             //i*W+j가 전경 화소면 신경쓸필요없이 그대로 0처리 신경쓸 필요도 없음
596             else Output[i * W + j] = 255;
597         }
598
599     }
600 }
601
```




```
763
764 Dilation(Image, Output, W, H);
765 Dilation(Output, Image, W, H);
766 Dilation(Image, Output, W, H); //팽창은 너무 뚱뚱하게 보일 수 있다는 단점이 있지
767 //Dilation(Output, Image, W, H);
768 //Dilation(Image, Output, W, H);
```

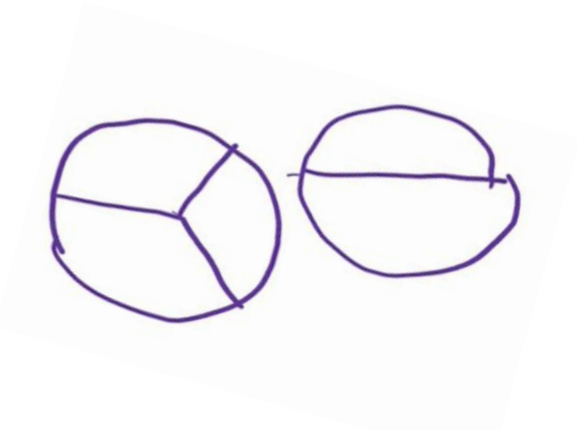
But 원래의 크기가 뚱뚱해진다는 단점이 있어
So 팽창 후 침식 closing을 써줘야해

```
763
764     Dilation(Image, Output, W, H);
765     Dilation(Output, Image, W, H);
766     Dilation(Image, Output, W, H); //팽창은 너무 뚱뚱하게 보일 수 있다는 단점이 있지
767     //Dilation(Output, Image, W, H);
768     //Dilation(Image, Output, W, H);
769     Erosion(Image, Output, W, H);
770     Erosion(Output, Image, W, H);
771     Erosion(Image, Output, W, H); //팽창은 너무 뚱뚱하게 보일 수 있다는 단점이 있지
772
```

평창 침식을 같은 횟수로 적용시켜주며

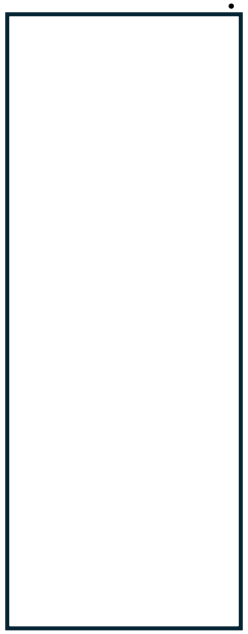
원래 (전경)영상의 사이즈를
유지시켜줄 수 있어

```
42  //zhangSuen => thining을 할 수 있는 함수
43  int zhangSuenTest1(int row, int col) {
44      int neighbours = getBlackNeighbours(row, col);
45
46      return ((neighbours >= 2 && neighbours <= 6)
47          && (getBWTransitions(row, col) == 1)
48          && (imageMatrix[row - 1][col] == blankPixel || imageMatrix[row][col + 1] == blankPixel || imageMatrix[row + 1][
49          && (imageMatrix[row][col + 1] == blankPixel || imageMatrix[row + 1][col] == blankPixel || imageMatrix[row][col
50  }
51
52  int zhangSuenTest2(int row, int col) {
53      int neighbours = getBlackNeighbours(row, col);
54
55      return ((neighbours >= 2 && neighbours <= 6)
56          && (getBWTransitions(row, col) == 1)
57          && (imageMatrix[row - 1][col] == blankPixel || imageMatrix[row][col + 1] == blankPixel || imageMatrix[row][col
58          && (imageMatrix[row - 1][col] == blankPixel || imageMatrix[row + 1][col] == blankPixel || imageMatrix[row][col
59  }
```



골격화 진행시
: 뼈대가 나옴

세선화 진행시
: 경로가 나옴



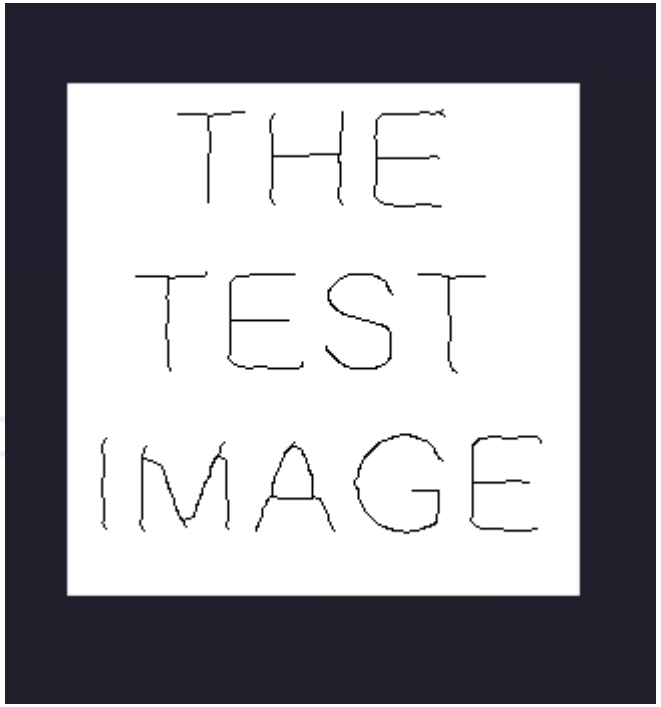
강의에서는 골격화에 대해서만 설명하긴 함..

Thining을 설명하자면,,

전경이 사각형 모양으로 있다면
1. 골격화 진행 (skeletionization)
2. 세선화 (Thining)
=> 둘 다 비슷, 두께가 1픽셀인 라인 형태로 변형이 되는 것

# Thining (골격화보다 세선화를 더 많이 씀)

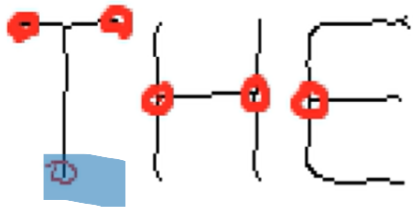
```
886
887     Dilation(Image, Output, W, H);
888     Dilation(Output, Image, W, H);
889     Dilation(Image, Output, W, H); //팽창은 너무 뚱뚱하게 보일 수 있다는 단점이 있지
890     //Dilation(Output, Image, W, H);
891     ////Dilation(Image, Output, W, H);
892     Erosion(Image, Output, W, H);
893     Erosion(Output, Image, W, H);
894     Erosion(Image, Output, W, H); //팽창은 너무 뚱뚱하게 보일 수 있다는 단점이 있지
895
896     //zhangSuen 코드때문에 한 번 뒤집어 줘야해 전경화소랑 배경화소가 반대로 되어있음
897
898     InverseImage(Image, Image, W, H);
899
900     zhangSuen(Image, Output, H, W);
901
902     SaveBMPFile(hf, hInfo, hRGB, Output, hInfo.biWidth, hInfo.biHeight, "output.bmp");
903
```

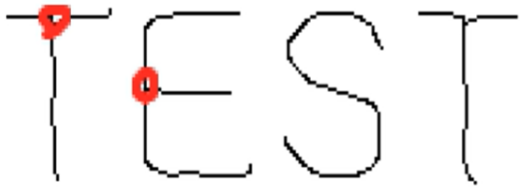


# Thining

분기점

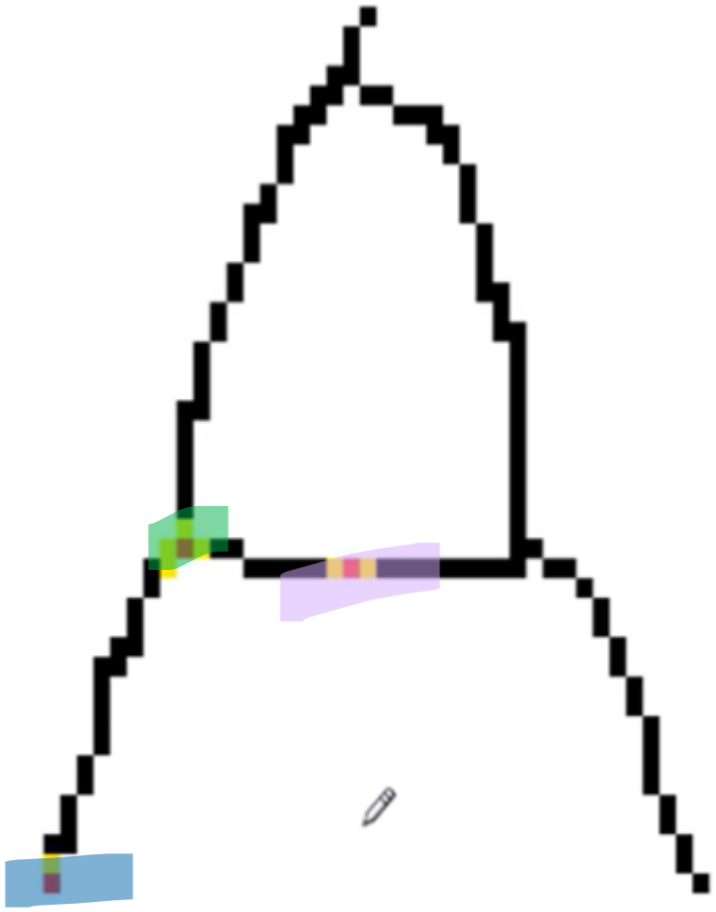
끝점





- 지문인식 / 글자 필체 인식
- → 지문의 경로를 가지고 분기점/끝점을 뽑을 때 많이 씀
- → 분기점/끝점이란?
- ➔ 과제로 낼 거임 집중
- 분기점?
- -> 라인이 진행되다가 양갈래로 나뉘어지는, 갈라지는 지점
- 끝점?
- -> 라인이 지나다가 끝나는 지점
- 이 두개를 특징으로 하여 같은 지문인지 아닌지를 알아볼 수 있음!

- 과제: 분기점과 끝점의 위치를 찾아서 원래 영상에 표시를 하면 돼
    - 찾았으면 그 위치에 plus를 하는 등등의 과제를 하면 돼



블랙화소인 전경화소 주변 8방향을 검사해서
블랙이 2개 이상이면 그냥 진행되는 라인

But 끝점을 기준으로 8방향을 검사했을 때
1개만 전경화소면 끝점

And 분기점은 8방향 주변화소 중에
3개 이상이 전경화소면 분기점인 거야

So!!
Thining된 영상에다가 끝점을 중심으로
8주변 화소를 회색으로 감싸주기

내가 짠 처음 코드..

```
231 void FeatureExtractThinImage(BYTE* Image, BYTE* Output, int W, int H)
232 {
233     //분기점 및 끝점 표시
234
235     for (int i = 0; i < H; i++)
236     {
237         for (int j = 0; j < W; j++)
238         {
239             if (Image[i * W + j] == 0)
240             {
241                 int count = 0;
242
243                 for (int ii = -1; ii <= 1; ii++)
244                 {
245                     for (int jj = - 1; jj <= 1; jj++)
246                     {
247                         if (Image [(i+ii) * W + (j+jj)] == 0)
248                         {
249                             count++;
250                         }
251
252                         else
253                         {
254                             k = k;
255                         }
256
257                     }
258                 }
259
260                 if (k == 1) //끝점
261                 {
262                     int m, n;
263                     for (m = i - 1; i + 1; i++)
264                     {
265                         for (n = j - 1; j + 1; j++)
266                         {
267                             Output[m * W + n] = 128;
268                         }
269                     }
270                 }
```

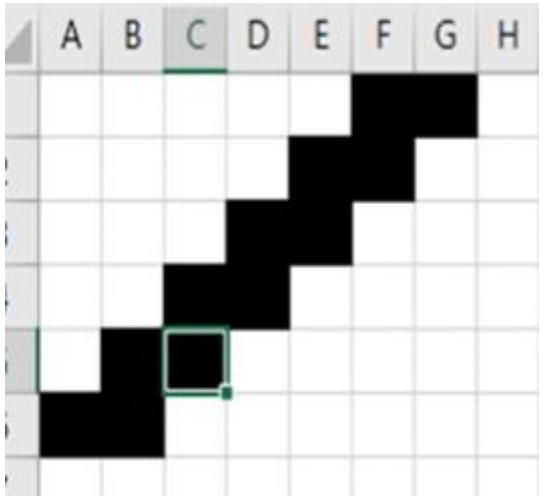
```
242
243     for (int ii = -1; ii <= 1; ii++)
244     {
245         for (int jj = - 1; jj <= 1; jj++)
246         {
247             if (Image [(i+ii) * W + (j+jj)] == 0)
248             {
249                 count++;
250             }
251
252             if (ii == 0 && jj == 0) continue; //주변 픽셀이 아니라 픽셀 자기 자신
253
254         }
255     }
256
```

```
272                 else if (k == 3) //분기점
273                 {
274                     int m, n;
275                     for (m = i - 1; i + 1; i++)
276                     {
277                         for (n = j - 1; j + 1; j++)
278                         {
279                             Output[m * W + n] = 128;
280                         }
281                     }
282                 }
283 
284                 else //2개나 4개이상
285                 {
286                     int m, n;
287                     for (m = i - 1; i + 1; i++)
288                     {
289                         for (n = j - 1; j + 1; j++)
290                         {
291                             Output[m * W + n] = Image[m * W + n];
292                         }
293                     }
294                 }
295 
296             else
297             {
298                 Output[i * W + j] = 255;
299             }
300 
301             }
302         }
303     }
304 
305 }
```

GPT랑 같이 만든 최종본..!

```
void FeatureExtractInInImage(BYTE* Image, BYTE* Output, int W, int H){
    //분기점 및 끝점 표시

    for (int i = 1; i < H - 1; i++)
    {
        for (int j = 1; j < W - 1; j++)
        {
            if (Image[i * W + j] == 0)
            {
                int count = 0;

                for (int ii = -1; ii <= 1; ii++)
                {
                    for (int jj = -1; jj <= 1; jj++)
                    {

                        if (ii == 0 && jj == 0) continue; //주변 픽셀이 아니라 픽셀 자기 자신

                        if (Image[(i + ii) * W + (j + jj)] == 0)
                        {
                            count++;
                        }


                    }
                }

                // 회색으로 둘러싸기
                if (count == 1 || count >= 3)
                {

                    for (int ii = -1; ii <= 1; ii++)
                    {
                        for (int jj = -1; jj <= 1; jj++)
                        {
                            int y = i + ii, x = j + jj;
                            if (y >= 0 && y < H && x >= 0 && x < W)
                                Output[y * W + x] = 128; // 회색으로 표시
                        }
                    }
                }
                else
                {
                    Output[i * W + j] = 0;
                }
            }
            else
            {
                Output[i * W + j] = 255; // 배경은 흰색
            }
        }
    }
}
```

# 과제 풀이




사실 지난번처럼 3개의 전경화소랑 만날때 분기점이라고 하면
왼쪽과 같이 분기점이 아닌데에도 분기점이라고 표시한다
So, 다른 해결 방안 필요

➔ 제일 좋은 건 thining함수가 예쁜 결과를 뽑는 것
➔ 하지만 안 되니까 주변 검사할때 2픽셀 거리에 있는 값을 검사
➔ 4픽셀말고 8픽셀로 검사를 진행하는 것은 도움 안 돼 결과가 같을것

# 어쩔 수 없어!

```
289 void FeatureExtractThin(BYTE* Image, BYTE* Output, int W, int H)
290 {
291     for (int i = 0; i < W * H; i++)
292     {
293         Output[i] = Image[i];
294     }
295
296     int cnt = 0;
297
298     for (int i = 2; i < H - 2; i++)
299     {
300         for (int j = 2; j < W - 2; j++)
301         {
302             //전경화소라면
303             if (Image[i * W + j] == 0)
304             {
305                 //8방향 검사하기
306
307                 if (Image[(i+1) * W + j+1] == 0) cnt++;
308                 if (Image[(i+1) * W + j] == 0) cnt++;
309                 if (Image[(i+1) * W + j-1] == 0) cnt++;
310                 if (Image[i * W + j+1] == 0) cnt++;
311
312                 if (Image[i * W + j-1] == 0) cnt++;
313                 if (Image[(i-1) * W + j+1] == 0) cnt++;
314                 if (Image[(i-1) * W + j] == 0) cnt++;
315                 if (Image[(i-1) * W + j-1] == 0) cnt++;
316             }
317
318             if (cnt == 1)
319             {
320                 Output[i * W + j] = 128;
321             }
322
323             else if (cnt >= 3)
324             {
325                 Output[i * W + j] = 128;
326             }
327
328             cnt = 0;
329         }
330     }
331 }
```

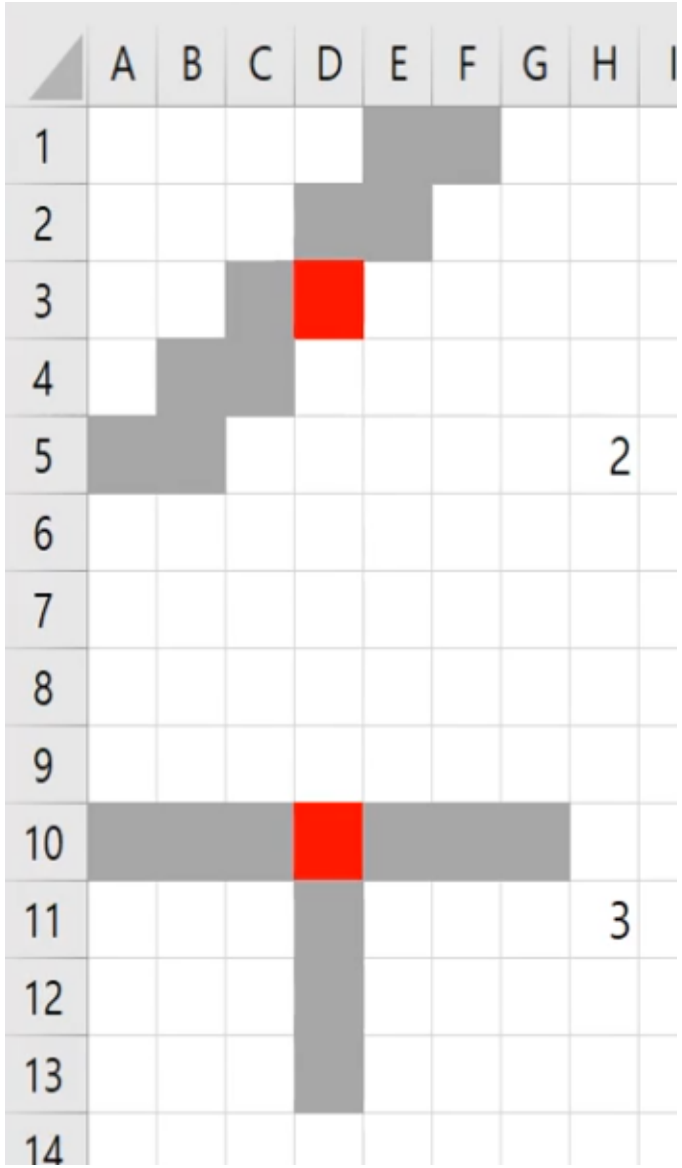
문자의 분기점, 끝점을 가지고 동일값인지
판단하는 방법!

우선 8픽셀을 검사했을때 결과 ->


문제해결X
Output이 문제인 것~
근본적인 건! 해결 방법은!
대각선도 예쁘게 씨닝되어있어야해(1픽셀로 되도록)

# Idea! ➔ NEW 개선 version

- 분기점에 대해 개선할 수 있을 거 같아!
- 카운팅으로 고려하면 연결성 문제로 두께가 한 픽셀이 아니라고 판단되어서 분기점이 엄청 많이 검출되게 돼
- So, 시계방향으로 움직이면서 2개 픽셀씩 비교를 해
- → 실제 분기점에선 검사를 해보면 3번이 흑->백으로 바뀌게 돼
- ➔ 바뀌는 횟수를 세면 되겠어!



# 시계방향으로 움직이면서 2개 픽셀씩 비교를 해
# → 실제 분기점에선 검사를 해보면 3번이 흑->백으로 바뀌게 돼

```
879     for (int i = 0; i < W * H; i++) Output[i] = Image[i];
880     int cnt = 0;
881     for (int i = 1; i < H - 1; i++) {
882         for (int j = 1; j < W - 1; j++) {
883             if (Image[i * W + j] == 0) {
884                 if (Image[(i - 1) * W + j - 1] == 0 && Image[(i - 1) * W + j] == 255) cnt++;
885                 if (Image[(i - 1) * W + j] == 0 && Image[(i - 1) * W + j + 1] == 255) cnt++;
886                 if (Image[(i - 1) * W + j + 1] == 0 && Image[i * W + j + 1] == 255) cnt++;
887                 if (Image[i * W + j + 1] == 0 && Image[(i + 1) * W + j + 1] == 255) cnt++;
888                 if (Image[(i + 1) * W + j + 1] == 0 && Image[(i + 1) * W + j] == 255) cnt++;
889                 if (Image[(i + 1) * W + j] == 0 && Image[(i + 1) * W + j - 1] == 255) cnt++;
890                 if (Image[(i + 1) * W + j - 1] == 0 && Image[i * W + j - 1] == 255) cnt++;
891                 if (Image[i * W + j - 1] == 0 && Image[(i - 1) * W + j - 1] == 255) cnt++;
892             }
893             if (cnt == 1) // 끝점
894             {
895                 printf("e");
896                 Output[i * W + j] = 128;
897             }
898             else if (cnt >= 3) // 분기점
899             {
900                 printf("b");
901                 Output[i * W + j] = 128;
902             }
903
904             cnt = 0;
```