구슬 찾기

모양은 같으나, 무게가 모두 다른 N개의 구슬이 있다. N은 홀수이며, 구슬에는 번호가 $1, 2, \ldots, N$으로 붙어 있다. 이 구슬 중에서 무게가 전체의 중간인 (무게 순서로 $(N+1)/2$번째) 구슬을 찾기 위해서 아래와 같은 일을 하려 한다. 우리에게 주어진 것은 양팔 저울이다. 한 쌍의 구슬을 골라서 양팔 저울의 양쪽에 하나씩 올려 보면 어느 쪽이 무거운 가를 알 수 있다. 이렇게 M개의 쌍을 골라서 각각 양팔 저울에 올려서 어느 것이 무거운 가를 모두 알아냈다. 이 결과를 이용하여 무게가 중간이 될 가능성이 전혀 없는 구슬들은 먼저 제외한다.

예를 들어, $N=5$이고, $M=4$ 쌍의 구슬에 대해서 어느 쪽이 무거운가를 알아낸 결과가 아래에 있다.

① 구슬 2번이 구슬 1번 보다 무겁다.
② 구슬 4번이 구슬 3번 보다 무겁다.
③ 구슬 5번이 구슬 1번 보다 무겁다.
④ 구슬 4번이 구슬 2번 보다 무겁다.

위와 같이 네 개의 결과만을 알고 있으면, 무게가 중간인 구슬을 정확하게 찾을 수는 없지만, 1번 구슬과 4번 구슬은 무게가 중간인 구슬이 절대 될 수 없다는 것은 확실히 알 수 있다. 1번 구슬보다 무거운 것이 2, 4, 5번 구슬이고, 4번 보다 가벼운 것이 1, 2, 3번이다. 따라서 답은 2개이다.

M개의 쌍에 대한 결과를 보고 무게가 중간인 구슬이 될 수 없는 구슬의 개수를 구하는 프로그램을 작성하시오.

실행 파일의 이름은 BALL.EXE로 하고 실행 시간은 1초를 초과할 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 입력 자료의 첫 줄은 구슬의 개수를 나타내는 정수 N ($1 \le N \le 99$)과 저울에 올려 본 쌍의 개수 M이 주어진다. 그 다음 M개의 줄은 각 줄마다 두 개의 구슬 번호가 주어지는데, 앞 번호의 구슬이 뒤 번호의 구슬보다 무겁다는 것을 뜻한다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫 줄에 무게가 중간이 절대로 될 수 없는 구슬의 수를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
5 4
2 1
4 3
5 1
4 2
```

출력(OUTPUT.TXT)

```
2
```

풀이

이 문제는 두 구슬들 간의 무게 관계가 주어졌을 때, 이를 토대로 무게의 순서를 추론하여 무게가 중간이 될 수 없는 구슬을 찾아내는 문제이다. 여러 가지 방법으로 무게 정보를 추론할 수 있는데, 가장 간명한 것은 구슬간의 관계를 그래프 형태로 모델링하여 처리하는 것이다.

이를 위해서 각 구슬을 정점으로 놓고 무거운 구슬에서 가벼운 구슬로 간선을 주면 방향 그래프가 만들어진다. 이제 각각의 구슬에 대해서 자기 자신보다 무거운 구슬의 개수와 가벼운 구슬의 개수를 세어 준 뒤, 이 둘 중 하나라도 과반수가 되면 그 구슬은 절대로 중간값이 될 수 없음을 알 수 있다. 이제 각각의 구슬에 대해 너비우선탐색(BFS) 또는 깊이우선탐색(DFS)을 두 번씩 진행하는 방법을 이용할 수도 있고, 플로이드(Floyd) 알고리즘을 이용할 수도 있다. 모두 시간복잡도는 $O(N^3)$이 된다. 이처럼 구슬에 순서를 주는 문제이므로 방향 그래프의 위상 정렬과 비슷한 문제가 되지만, 이와는 다른 경우가 생길 수 있음에 주의해야 한다.

```
#include <fstream.h>

#define maxn 101
#define infile "input.txt"
#define outfile "output.txt"

int n,m;
int arr1[maxn][maxn];
int arr2[maxn][maxn];
int arr1n[maxn];
int arr2n[maxn];
int visited[maxn];
int visitedn;
int res;


void bfs1(int);
void bfs2(int);
void input_data();
void solving_problem();
void output_data();

void main()
{
	input_data();

	solving_problem();

	output_data();
}

void input_data()
{

	int tmp1,tmp2;
	int i,j;

	for(i=0;i<maxn;i++)
	{
		arr1n[i]=0;
		arr2n[i]=0;
		for(j=0;j<maxn;j++)
		{
			arr1[i][j]=0;
			arr2[i][j]=0;
		}
	}


	ifstream indat(infile);
```

```
	indat >> n >> m;

	for(i=0;i<m;i++)
	{
		indat >> tmp1 >> tmp2;

		tmp1--;
		tmp2--;


		arr1[tmp1][arr1n[tmp1]]=tmp2;
		arr1n[tmp1]++;

		arr2[tmp2][arr2n[tmp2]]=tmp1;
		arr2n[tmp2]++;
	}


	indat.close();
}


void solving_problem()
{
	int i,j;

	for(i=0;i<n;i++)
	{

		visitedn=0;
		for(j=0;j<n;j++)
			visited[j]=0;
		bfs1(i);
//		cout << i << " LEFT : " << visitedn << '\n';
		if(visitedn>=(n+1)/2)
		{
			res++;
		}
		else
		{
			visitedn=0;
			for(j=0;j<n;j++)
				visited[j]=0;
			bfs2(i);
//		cout << i << " RIGHT : " << visitedn << '\n';
			if(visitedn>=(n+1)/2)
				res++;
		}
	}
}
```

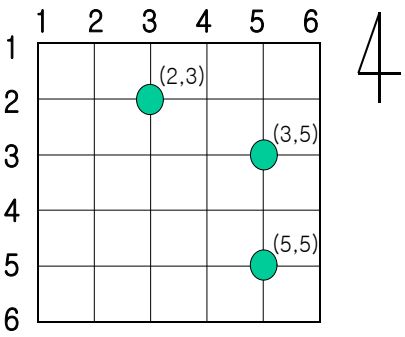
```
void output_data()
{
	ofstream outdat(outfile);
	outdat << res << '\n';
	outdat.close();
}

void bfs1(int i)
{
	int j;
	for(j=0;j<arr1n[i];j++)
	{
		if(visited[arr1[i][j]]==0)
		{
			visited[arr1[i][j]]=1;
			visitedn++;
			bfs1(arr1[i][j]);
		}
	}
}


void bfs2(int i)
{
	int j;
	for(j=0;j<arr2n[i];j++)
	{
		if(visited[arr2[i][j]]==0)
		{
			visited[arr2[i][j]]=1;
			visitedn++;
			bfs2(arr2[i][j]);
		}
	}
}
```

경찰차

어떤 도시의 중심가는 N개의 동서방향 도로와 N개의 남북방향 도로로 구성되어 있다. 모든 도로에는 도로 번호가 있으며 남북방향 도로는 왼쪽부터 1에서 시작하여 N까지 번호가 할당되어 있고 동서방향 도로는 위부터 1에서 시작하여 N까지 번호가 할당되어 있다. 또한 동서방향 도로 사이의 거리와 남북방향 도로 사이의 거리는 모두 1이다. 동서방향 도로와 남북방향 도로가 교차하는 교차로의 위치는 두 도로의 번호의 쌍인 (동서방향 도로 번호, 남북방향 도로 번호)로 나타낸다. N이 6인 경우의 예를 들면 다음과 같다.



이 도시에는 두 대의 경찰차가 있으며 두 차를 경찰차1과 경찰차2로 부른다. 처음에는 항상 경찰차1은 (1, 1)의 위치에 있고 경찰차2는 (N, N)의 위치에 있다. 경찰 본부에서는 처리할 사건이 있으면 그 사건이 발생된 위치를 두 대의 경찰차 중 하나에 알려 주고, 연락 받은 경찰차는 그 위치로 가장 빠른 길을 통해 이동하여 사건을 처리한다. (하나의 사건은 한 대의 경찰차가 처리한다.) 그리고 사건을 처리한 경찰차는 경찰 본부로부터 다음 연락이 올 때까지 처리한 사건이 발생한 위치에서 기다린다.

경찰 본부에서는 사건이 발생한 순서대로 두 대의 경찰차에 맡기려고 한다. 처리해야 될 사건들은 항상 교차로에서 발생하며 경찰 본부에서는 이러한 사건들을 나누어 두 대의 경찰차에 맡기되, 두 대의 경찰차들이 이동하는 거리의 합을 최소화 하도록 사건을 맡기려고 한다.

예를 들어 앞의 그림처럼 $N=6$인 경우, 처리해야 하는 사건들이 3개 있고 그 사건들이 발생된 위치를 순서대로 (3, 5), (5, 5), (2, 3)이라고 하자. (3, 5)의 사건을 경찰차2에 맡기고 (5, 5)의 사건도 경찰차2에 맡기며, (2, 3)의 사건을 경찰차1에 맡기면 두 차가 이동한 거리의 합은 4 + 2 + 3 = 9가 되고, 더 이상 줄일 수는 없다.

처리해야 할 사건들이 순서대로 주어질 때, 두 대의 경찰차가 이동하는 거리의 합을 최소화 하도록 사건들을 맡기는 프로그램을 작성하시오.

실행파일의 이름은 POLICE.EXE로 하고, 프로그램의 실행시간은 1초를 넘을 수 없다. 부분 점수는 없다.

입력 형식

입력 파일의 이름은 INPUT.TXT로 한다. 입력 파일의 첫째 줄에는 동서방향 도로의 개수를 나타내는 정수 $N(5 \le N \le 1,000)$이 주어진다. 둘째 줄에는 처리해야 하는 사건의 개수를 나타내는 정수 $W(1 \le W \le 1,000)$가 주어진다. 셋째 줄부터 ($W+2$)번째 줄까지 사건이 발생된 위치가 한 줄에 하나씩 주어진다. 경찰차들은 이 사건들을 주어진 순서대로 처리해야 한다. 각 위치는 동서

방향 도로 번호를 나타내는 정수와 남북 방향 도로 번호를 나타내는 정수로 주어지며 두 정수 사이에는 빈칸이 하나 있다. 두 사건이 발생한 위치가 같을 수 있다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT로 한다. 첫째 줄에 두 경찰차가 이동한 총 거리를 출력한다. 둘째 줄부터 시작하여 $(i+1)$번째 줄에 i $(1 \le i \le W)$번째 사건이 맡겨진 경찰차 번호 1 또는 2를 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
6
3
3 5
5 5
2 3
```

출력(OUTPUT.TXT)

```
9
2
2
1
```

풀이

이 문제는 처리해야 하는 사건들에 대한 정보가 주어졌을 때, 두 대의 경찰차를 이용하여 최소 거리를 이동하며 사건을 맡는 방법을 찾아내는 문제이다. 각 경찰이 사건을 처리하면 해당 경찰은 해당 위치에 있게 되므로, 가능한 경우가 얼마 되지 않음을 알 수 있다. 따라서 이러한 경우들을 잘 추적하면서, 각 경우의 최소 이동 거리를 구해 나가는 동적 계획법으로 해결할 수 있다. 즉, 두 경찰차의 현재 위치와 다음 처리할 사건을 변수로 두어, 현재 사건에 두 경찰차 중 어느 것을 배치하는 것이 유리한가를 계산하여 최적화하는 방향으로 경찰차를 선택해 나가면 된다.

D[i][j]를 현재 i번째 사건까지 처리했고, 나머지 하나의 경찰차가 j번 사건 현장에 출동하여 있을 때 거리의 최소값으로 정의한다. 이 때 편의를 위해서 i > j가 항상 만족된다고 생각하자. 이제 가능한 경우는 i-1번 사건 현장에 있던 경찰차가 i번으로 이동한 경우와, 다른 k번 사건 현장에 있던 경찰차가 i번으로 이동한 경우가 있음을 알 수 있다. 따라서 D[i-1][j] + (i번과 i+1번 사이의 거리)와 D[j][k] + (i번과 k번 사이의 거리) 중 최소값을 선택해 나가면 된다. 이 때 각각의 수식은 각각의 경우에 차례로 대응된다. 이와 같은 방식으로 D 배열을 구한 후에, 사건이 N개일 때, D[N][i]들 중 최소값이 구하고자 하는 답이 된다.

```
#include <fstream.h>
#include <math.h>

#define maxn 1001
#define infile "input.txt"
#define outfile "output.txt"

int arr1[maxn];
int arr2[maxn];
int his1[maxn];
int his2[maxn];
int n,w;
int work[maxn][2];


void input_data();
void solving_problem();
void output_data();
int dist(int,int);

void main()
{
	input_data();

	solving_problem();

	output_data();
}

void input_data()
{
	ifstream indat(infile);
	indat >> w >> n;
	int i;
	for(i=1;i<=n;i++)
		indat >> work[i][0] >> work[i][1];
	indat.close();
}

void solving_problem()
{
	int i,j;
	for(i=0;i<maxn;i++)
	{
		arr1[i]=0;
		arr2[i]=0;
		his1[i]=0;
		his2[i]=0;
	}

	arr1[0]=abs(work[1][0]-1)+abs(work[1][1]-1);
```

```
arr2[0]=abs(work[1][0]-w)+abs(work[1][1]-w);

int tmp1[maxn];
int tmp2[maxn];
int tmp3,tmp4;
for(i=2;i<=n;i++)
{
        for(j=0;j<=i-2;j++)
        {
                tmp1[j]=arr1[j]+dist(i,i-1);

        }
        work[0][0]=1;
        work[0][1]=1;

        tmp3=-1;
        for(j=0;j<=i-2;j++)
        {
                if(tmp3>arr2[j]+dist(j,i) || tmp3==-1)
                {
                        tmp3=arr2[j]+dist(j,i);
                        his1[i]=j;
                }
        }
        tmp1[i-1]=tmp3;


        for(j=0;j<=i-2;j++)
        {
                tmp2[j]=arr2[j]+dist(i,i-1);

        }
        work[0][0]=w;
        work[0][1]=w;

        tmp3=-1;
        for(j=0;j<=i-2;j++)
        {
                if(tmp3>arr1[j]+dist(j,i) || tmp3==-1)
                {
                        tmp3=arr1[j]+dist(j,i);
                        his2[i]=j;
                }
        }
        tmp2[i-1]=tmp3;

        for(j=0;j<=i-1;j++)
        {
                arr1[j]=tmp1[j];
                arr2[j]=tmp2[j];
        }
```

```
		}

}

void output_data()
{
		int i,j;
		int tmp1,tmp2,tmp3;
		int tmp4[maxn];
		tmp1=-1;

		for(i=0;i<=n-1;i++)
		{
			if(tmp1==-1 || arr1[i]<tmp1)
			{
				tmp1=arr1[i];
				tmp2=1;
				tmp3=i;
			}
		}

		for(i=0;i<=n-1;i++)
		{
			if(tmp1==-1 || arr2[i]<tmp1)
			{
				tmp1=arr2[i];
				tmp2=2;
				tmp3=i;
			}
		}

		ofstream outdat(outfile);
		outdat << tmp1 << '\n';
		for(i=n;i>=1;i--)
		{
			tmp4[i]=tmp2;
			if(tmp2==1 && tmp3==i-1)
			{
				tmp2=2;
				tmp3=his1[i];
			}
			else if(tmp2==2 && tmp3==i-1)
			{
				tmp2=1;
				tmp3=his2[i];
			}
		}

		for(i=1;i<=n;i++)
		{
```

```
                outdat << tmp4[i] << '\n';
        }
        outdat.close();
}

int dist(int i,int j)
{
        return (abs(work[i][0]-work[j][0])+abs(work[i][1]-work[j][1]));
}
```

단순사각형

철수는 모눈종이 위의 한 점에서 시작하여 수평, 수직 또는 수직, 수평으로 방향을 바꾸면서 모눈종이의 선들을 따라서 시작점에 다시 돌아 올 때까지 선을 그렸다. 철수가 그린 선은 여러 개의 수직선분과 수평선분으로 구성되고 다음의 조건들을 만족 한다 (그림1 참조).

① 각 선분은 서로 교차할 수 있으나 수직선분과 수직선분 또는 수평선분과 수평선분은 서로 연결되거나 교차 할 수 없다.
② 각 선분들의 끝점은 항상 다른 하나의 선분과 연결되어 있다.

철수가 그린 선은 모서리가 직각인 여러 개의 다각형을 구성하게 되는데 철수는 이들 중에서 내부에 어떤 선분도 포함하지 않는 단순사각형의 개수를 구하고자 한다. 아래 예를 살펴보자. 이 선은 16개의 선분으로 구성되어 있고 이 선이 만드는 단순사각형은 B, C, E 3개이다. C, D, E를 합하면 사각형이 되지만 내부에 다른 선분을 포함하고 있으므로 단순사각형이 아니다.

철수가 모눈 종이위에 그린선이 만드는 단순사각형의 개수를 구하는 프로그램을 작성하시오. 실행시간은 1초를 초과 할 수 없으며 부분점수는 없다. 실행 파일의 이름은 RECT.EXE로 한다.

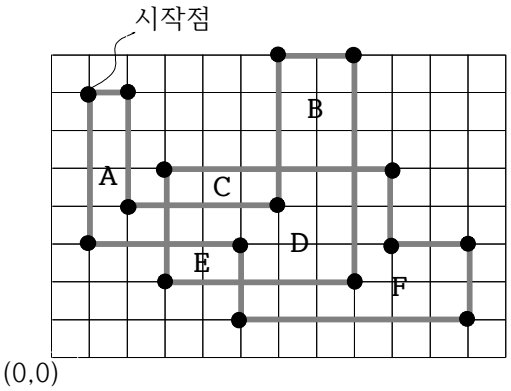


그림 1

입력 형식

입력 파일의 이름은 INPUT.TXT이다. 첫째 줄에는 꼭지점의 개수를 나타내는 정수 $N(4 \le N \le 1000)$이 나온다. 그 다음 N개의 줄에는 각각 하나의 꼭지점에 대한 좌표를 나타내는 두 개의 정수 x와 y $(0 \le x, y \le 10,000)$가 입력되는데, 첫 번째 정수 x는 그 꼭지점의 X-좌표를 나타내며, 두 번째 정수 y는 그 꼭지점의 Y-좌표를 나타낸다.

출력 형식

출력 파일의 이름은 OUTPUT.TXT이다. 첫째 줄에 단위 사각형의 개수를 출력한다. 만일 그러한 단순사각형이 없는 경우에는 0을 출력한다.

입력과 출력의 예

입력(INPUT.TXT)

```
16
1 7
2 7
2 4
6 4
6 8
8 8
8 2
3 2
3 5
9 5
9 3
11 3
11 1
5 1
5 3
1 3
```

출력(OUTPUT.TXT)

```
3
```

풀이

이 문제는 모눈종이에 그려진 선들에 대한 정보가 주어졌을 때, 만들어지는 단순사각형의 개수를 구하는 문제이다. 먼저 사전 처리 작업을 이용하여 선분에 대한 정보를 잘 처리한 후, 실제로 다각형을 그려 보면서 단순사각형이 언제 어떻게 만들어나가는지를 하나하나 따져보면 된다.

이러한 방법의 한 가지로 평면 쓸기(Plane sweeping) 기법을 이용하여 해결할 수 있다. 좌표의 범위는 0 이상 10,000이하로 주어지지만 실제로는 다각형의 꼭지점이 위치하고 있는 좌표들만이 단순 사각형의 개수에 영향을 줄 수 있다는 사실을 이용한다. 이러한 조건을 만족하는 좌표는 x좌표와 y좌표에 대해 각각 많아야 N/2개이므로 최대 500개가 됨을 알 수 있다. 따라서 2차원 배열을 잡고 실제로 선분을 그려 가면서 모든 경우를 세어 주면 시간과 공간 모두에 대해 $O(N^2)$에 해결할 수 있다.

```
#include <stdio.h>
#include <set>
#include <vector>
#include <map>
#include <deque>
#include <conio.h>

using namespace std ;

#define         inputfilename       "input.txt"
#define         outputfilename      "output.txt"
#define         MAX                                 1020
#define         UNDEFINE                    -2

int inline min2(int a, int b)     {         if ( a<b ) return a ; else return b ; }
int inline max2(int a, int b )    {         if ( a<b ) return b ; else return a ; }

class       coord {
public :
            bool        row,column ;
            int                 rSequence, cSequence ;
            coord() {   row=false ; column = false ;
                                    rSequence=UNDEFINE ; cSequence = UNDEFINE ;
            }
} table[MAX][MAX] ;

class       point       {
public :
            int                 x, y ;
            point(int a, int b ) {
                        x= a ; y= b ;
            }
} ;

int visit[MAX][MAX] , xValue[MAX], yValue[MAX];

void swapint(int &a , int &b ) {            // next time , try!!
            int temp = a ; a =b ; b = temp ;
}

void test() {
            int i,j ;
            printf("-------------------------------------------\n");
            for(i=MAX-1;i>=0;i--) {
                        for(j=0;j<MAX;j++)  {
                                    printf("%c",(table[j][i].row)?'|':' ');
                                    if (table[j][i].rSequence!=-1) {
                                                printf("(%d,",table[j][i].rSequence);
                                    } else { printf("( ,") ; }

                                    if (table[j][i].cSequence!=-1) {
```

```
                                printf("%d)",table[j][i].cSequence);
                        } else { printf(" )") ; }

                }
                printf("++\n");
                for(j=0;j<MAX;j++)   printf("    %c   ",(table[j][i].column)?'_':' ');
                printf("++\n");
        }
}

void readData() {
        set<int>            xIndex,    yIndex ;   xIndex.clear() ;       yIndex.clear() ;
        map<int,int>        xFunc , yFunc ;              xFunc.clear() ;                yFunc.clear() ;
        vector<point>       polyline ;                   polyline.clear() ;

        FILE *in=fopen(inputfilename,"r");
        int i ,n  , a, b;

        fscanf(in,"%d\n",&n) ;

        for(i=0;i<n;i++) {
                fscanf(in,"%d%d\n",&a,&b) ;
                //printf("%d %d\n",a,b);
                polyline.push_back(point(a,b)) ;
                xIndex.insert(a) ;
                yIndex.insert(b) ;
        }

        int count = 0 ; xFunc[0] = 0 ; xValue[count++] = 0 ;
        for(set<int>::iterator it = xIndex.begin() ; it!= xIndex.end() ; it++) {
                xFunc[*it] = count ; xValue[count++] = *it ;
        }
        count=0 ;           yFunc[0] = 0 ; yValue[count++] = 0 ;
        for(it = yIndex.begin() ; it!= yIndex.end() ; it++) {
                yFunc[*it] = count ; yValue[count++] = *it ;
        }

        for(i=0 ; i<n ; i++) {
                //printf("(%d,%d)                                    ->                                (%d,%d)
\n",polyline[i].x,polyline[i].y,xFunc[polyline[i].x],yFunc[polyline[i].y]) ;
        }

        fclose(in);

        int ax, ay , bx, by , j ;
        ax = xFunc[polyline[n-1].x] ; ay = yFunc[polyline[n-1].y] ;
        for(i=0; i<n ; i++) {
                bx = xFunc[polyline[i].x] ; by = yFunc[polyline[i].y] ;
                if ( ax==bx) {
                        for(j=min2(ay,by) ; j!=max2(ay,by) ; j++) {
                                table[ax][j].row = true ;
```

```
                        }
                } else {
                        for(j=min2(ax,bx) ; j!=max2(ax,bx) ; j++) {
                                table[j][ay].column = true ;
                        }
                }
                ax = bx ; ay = by ;
//              test();
//              getch();
        }
}

int dfssearch(int startx, int starty) {
        int area = 0 ;
        deque<point> que ;
        que.clear(); que.push_back(point(startx,starty));
        visit[startx][starty]=true ;
        int x,y ;
        while(!que.empty()) {
                x = que.front().x ; y = que.front().y ;
                area += (xValue[x+1]-xValue[x])*(yValue[y+1]-yValue[y]);
                que.pop_front() ;

                // up
                if (!table[x][y+1].column && y<MAX-1 && !visit[x][y+1] ) {
                        visit[x][y+1] =true ;
                        que.push_back(point(x,y+1)) ;
                }

                // down
                if (!table[x][y].column && y>0 && !visit[x][y-1] ) {
                        visit[x][y-1] =true ;
                        que.push_back(point(x,y-1)) ;
                }

                // left
                if (!table[x][y].row && x>0 && !visit[x-1][y] ) {
                        visit[x-1][y] =true ;
                        que.push_back(point(x-1,y)) ;
                }

                // right
                if (!table[x+1][y].row && x<MAX-1 && !visit[x+1][y] ) {
                        visit[x+1][y] =true ;
                        que.push_back(point(x+1,y)) ;
                }

        }
        return area ;
}
```

```
void calcsequence() {
	int i ,j , flag=0 ,count=0;
	for(j=0;j<MAX;j++) {
		for(i=MAX-2;i>=0;i--) {
			if ( table[i+1][j].row )		{  count= 0 ; }
			count++ ;
			if ( !visit[i][j] && table[i][j].row )
			{
				table[i][j].rSequence = count ;
			} else {
				table[i][j].rSequence = -1 ;
			}
		}
	}

	flag = 0 ; count = 0 ;

	for(i=0;i<MAX;i++) {
		for(j=MAX-2;j>=0;j--) {
			if ( table[i][j+1].column )		{ count= 0 ; }
			count++ ;
			if ( !visit[i][j] && table[i][j].column )
			{
				table[i][j].cSequence = count ;
			} else {
				table[i][j].cSequence = -1 ;
			}
		}
	}

}

void printsolution(int area) {
	FILE *out=fopen(outputfilename,"w");
	fprintf(out,"%d\n",area);
	fclose(out);
}

void process() {
	memset(visit,false,sizeof(visit));
	dfssearch(0,0);
	calcsequence();
//	test();
	int count=0 ;
	int i ,j ,k ;
	for(i=0;i<MAX-1;i++) {
		for(j=0;j<MAX-1;j++) {
			bool okay = true ;
			if (table[i][j].rSequence>0 && table[i][j].cSequence>0 ) {
				for(k=j;k<=j+table[i][j].cSequence-1;k++) {
					if (table[i][j].rSequence != table[i][k].rSequence ) {
```

```
                                                okay = false ; break;
                                        }
                                }
                                for(k=i;k<=i+table[i][j].rSequence-1;k++) {
                                        if (table[i][j].cSequence != table[k][j].cSequence ) {
                                                okay = false ; break;
                                        }
                                }
                                if ( okay ) {
                                        count++; //printf("found !! %d %d \n",i,j);
                                }
                        }
                }
        }
        printsolution(count);
}

int main() {
        readData();
        //test();
        process();
        return 0 ;
}
```