Announcements

Video competition registration form: https://forms.gle/WTJfmzTgJxyJ2nw29
Send video to or ask questions: stem2innovation@gmail.com

We still have class on Sunday 07/03

XJOI: No offending/inappropriate words are allowed in your code!

Homework last week (P3564)

Print scores

Print out all scores below average among n scores, 1<=n<=100.

Sample Input

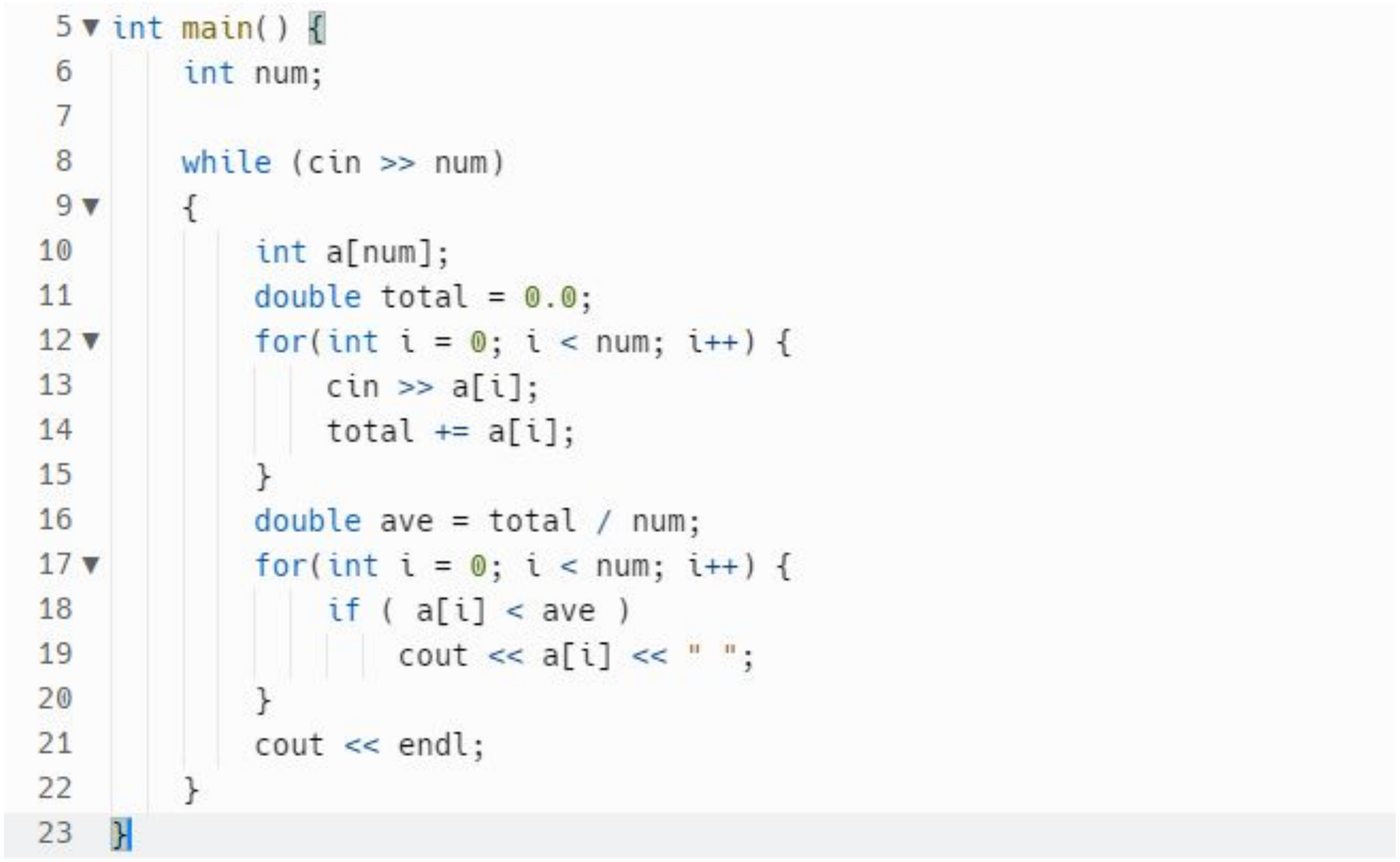
```
3 40 50 60
2 90 80
5 10 10 90 80 80
```

Sample Output

```
40
80
10 10
```

1. How to read input continually?
2. How to calculate average score?

Homework last week (P3564)



```
int main() {
    int num;

    while (cin >> num)
    {
        int a[num];
        double total = 0.0;
        for(int i = 0; i < num; i++) {
            cin >> a[i];
            total += a[i];
        }
        double ave = total / num;
        for(int i = 0; i < num; i++) {
            if ( a[i] < ave )
                cout << a[i] << " ";
        }
        cout << endl;
    }
}
```

Homework last week (P8120)

Red and black

A rectangular room is covered by square tiles which is either red or black. A man stands on a black tile, and he can move to one of the adjacent four tiles. The rule is he can only move to the black tiles, but not the red ones. Write a program to calculate the number of black tiles he can reach.

Use BFS

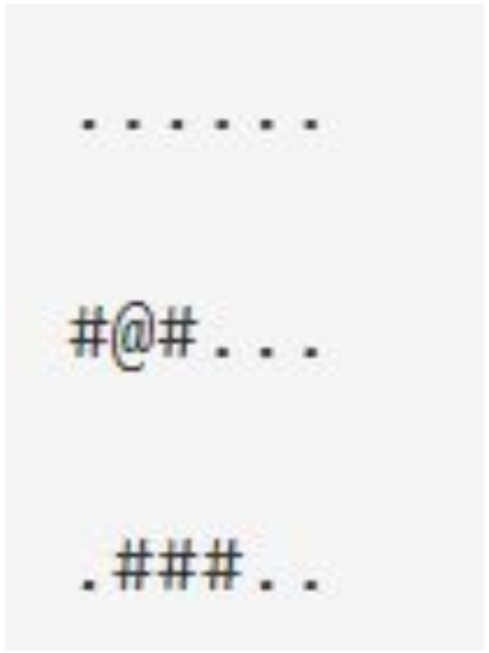
How to check if a neighbor is available?

Print out maze after reading to ensure it is correct.

Homework this week (P8120)

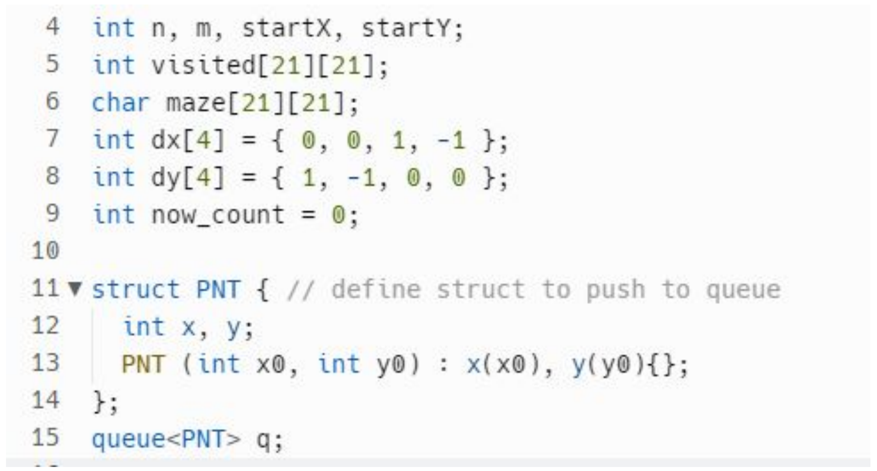
Use pen and paper to sort out your idea and data structure

- Understand problem, test sample cases by hand
- How to read input? (output to console to make sure it is correct)
- What is starting point?
- What data structure do we need for BFS?
  - maze, visited, node, queue, dir, count
- How to check transition condition (if neighbor is available)?
  - in bound
  - only black tiles

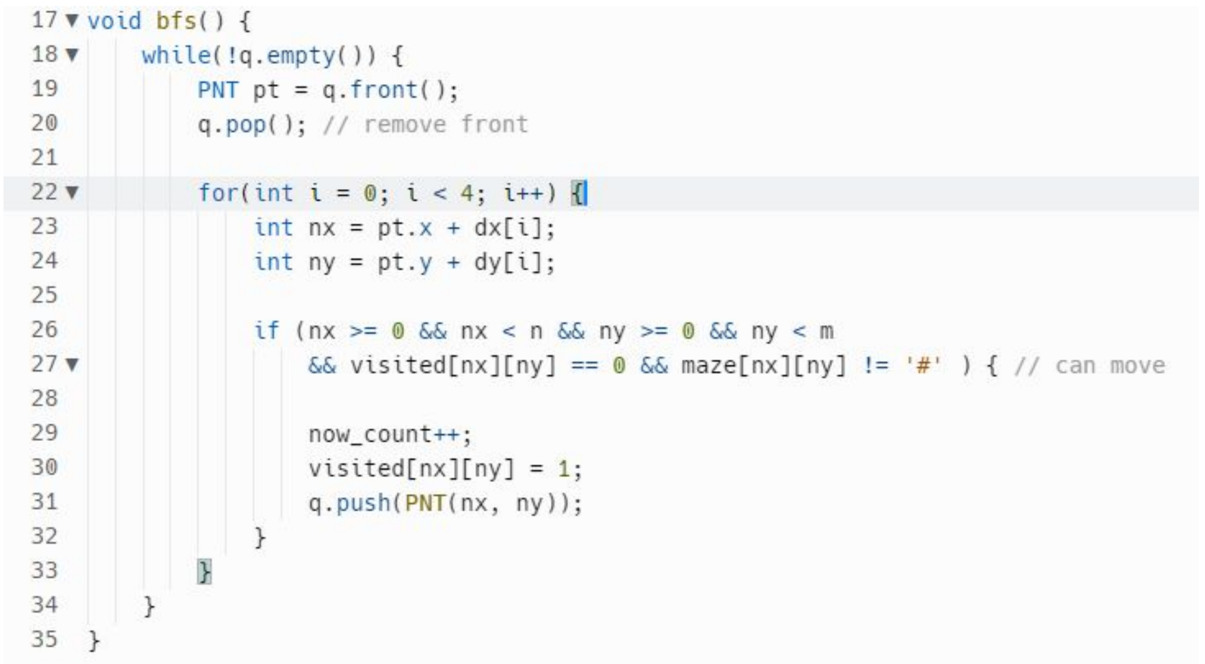


```
......
#@#...
.###..
```

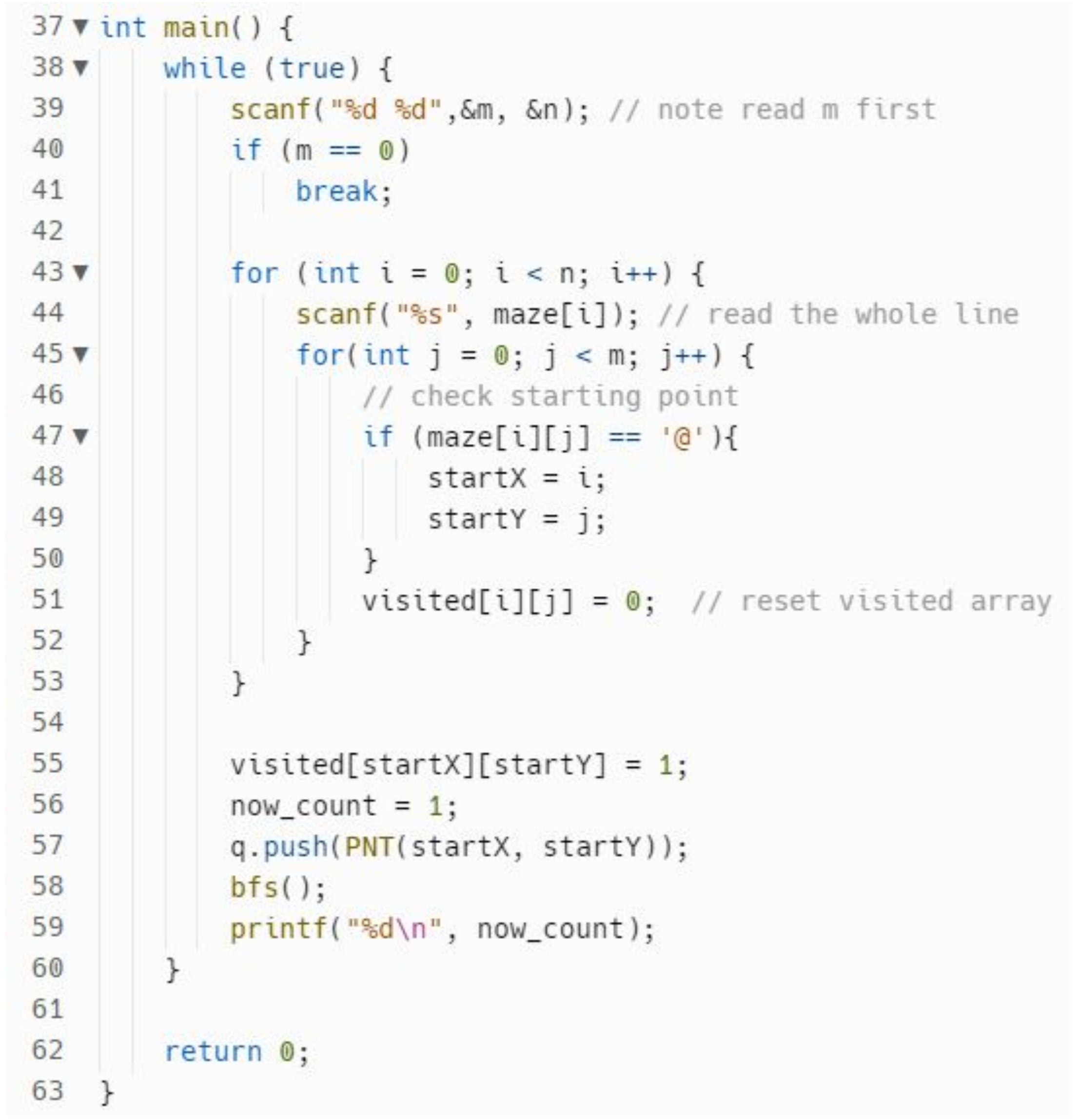
Homework this week (P8120)



```
int n, m, startX, startY;
int visited[21][21];
char maze[21][21];
int dx[4] = { 0, 0, 1, -1 };
int dy[4] = { 1, -1, 0, 0 };
int now_count = 0;

struct PNT { // define struct to push to queue
  int x, y;
  PNT (int x0, int y0) : x(x0), y(y0){};
};
queue<PNT> q;
```



```
void bfs() {
    while(!q.empty()) {
        PNT pt = q.front();
        q.pop(); // remove front

        for(int i = 0; i < 4; i++) {
            int nx = pt.x + dx[i];
            int ny = pt.y + dy[i];

            if (nx >= 0 && nx < n && ny >= 0 && ny < m
                && visited[nx][ny] == 0 && maze[nx][ny] != '#' ) { // can move

                now_count++;
                visited[nx][ny] = 1;
                q.push(PNT(nx, ny));
            }
        }
    }
}
```



```
int main() {
    while (true) {
        scanf("%d %d",&m, &n); // note read m first
        if (m == 0)
            break;

        for (int i = 0; i < n; i++) {
            scanf("%s", maze[i]); // read the whole line
            for(int j = 0; j < m; j++) {
                // check starting point
                if (maze[i][j] == '@'){
                    startX = i;
                    startY = j;
                }
                visited[i][j] = 0;  // reset visited array
            }
        }

        visited[startX][startY] = 1;
        now_count = 1;
        q.push(PNT(startX, startY));
        bfs();
        printf("%d\n", now_count);
    }

    return 0;
}
```

Homework last week (P8110)

There is a n×n maze which only has number 0 and 1. If you are in a spot 0, then you can move to one of the four adjacent cells with 1 in it. If you are in 1, then you can move to one of the four adjacent cells with 0 in it. Given the maze, find out how many cells you can move into (including the original cell).

Input:

The first line contains two positive integers n, m.

The next n lines, each line contains n characters (0 or 1, with no space in between)

The next m lines, each line contains two positive integers i and j (separated by space), which means the cell in the i-th row and the j-th column (the starting point).

Output:

m lines, output the answer for each inquiry.

sample input 1:

```
2 2
01
10
1 1
2 2
```

sample output 1:

```
4
4
```

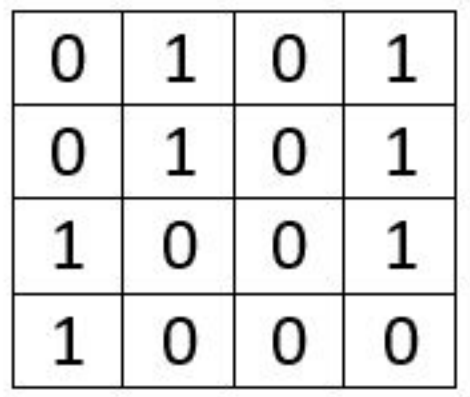
Note:

For 100% data， n≤1000,m≤100000。

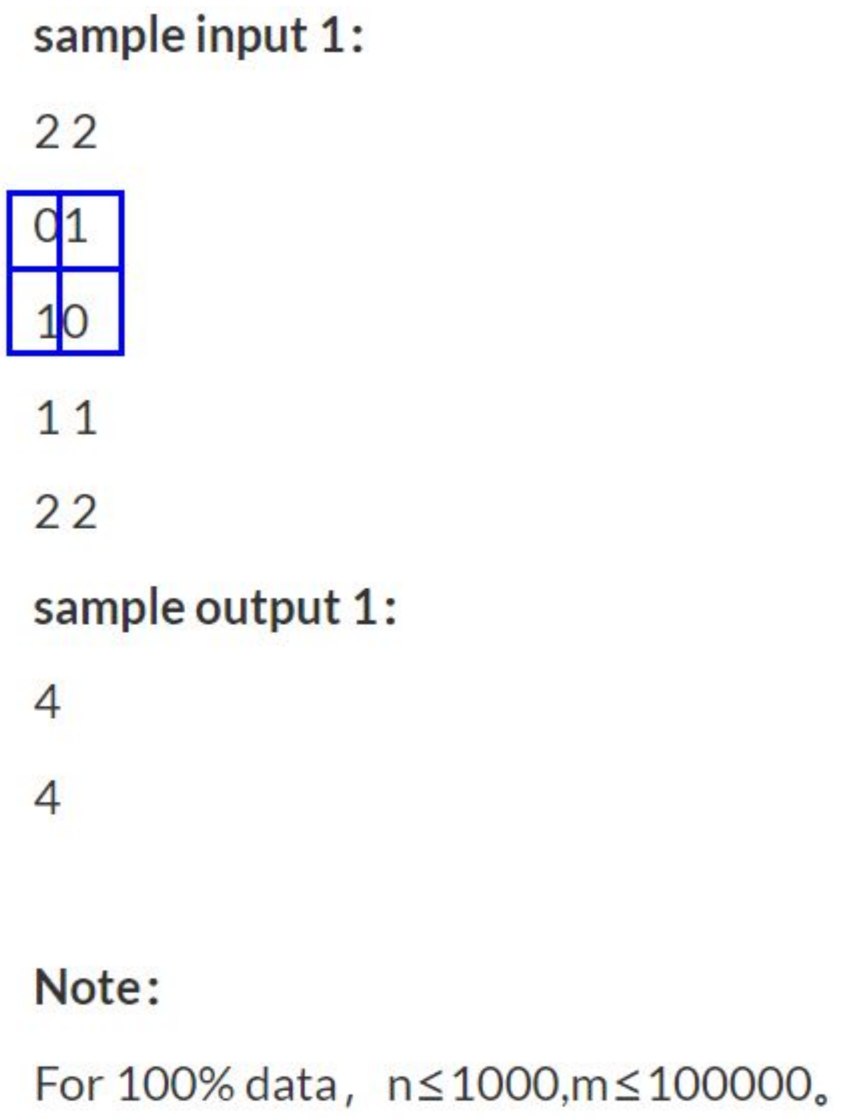
Homework this week (P8110)

Use pen and paper. Then possible solution:

- Read input correctly (try your own way but if you have trouble, there is hint on homework page)
- DFS or BFS are fine. I choose DFS flood fill
- Use another 2D array to store area ID
- Use result[area_id] to store filled cells for this area
- For a new query, if cell has been filled, get existing answer
- Sample case is too simple. Make up your own test cases



0	1	0	1
0	1	0	1
1	0	0	1
1	0	0	0



sample input 1:

2 2

01

10

1 1

2 2

sample output 1:

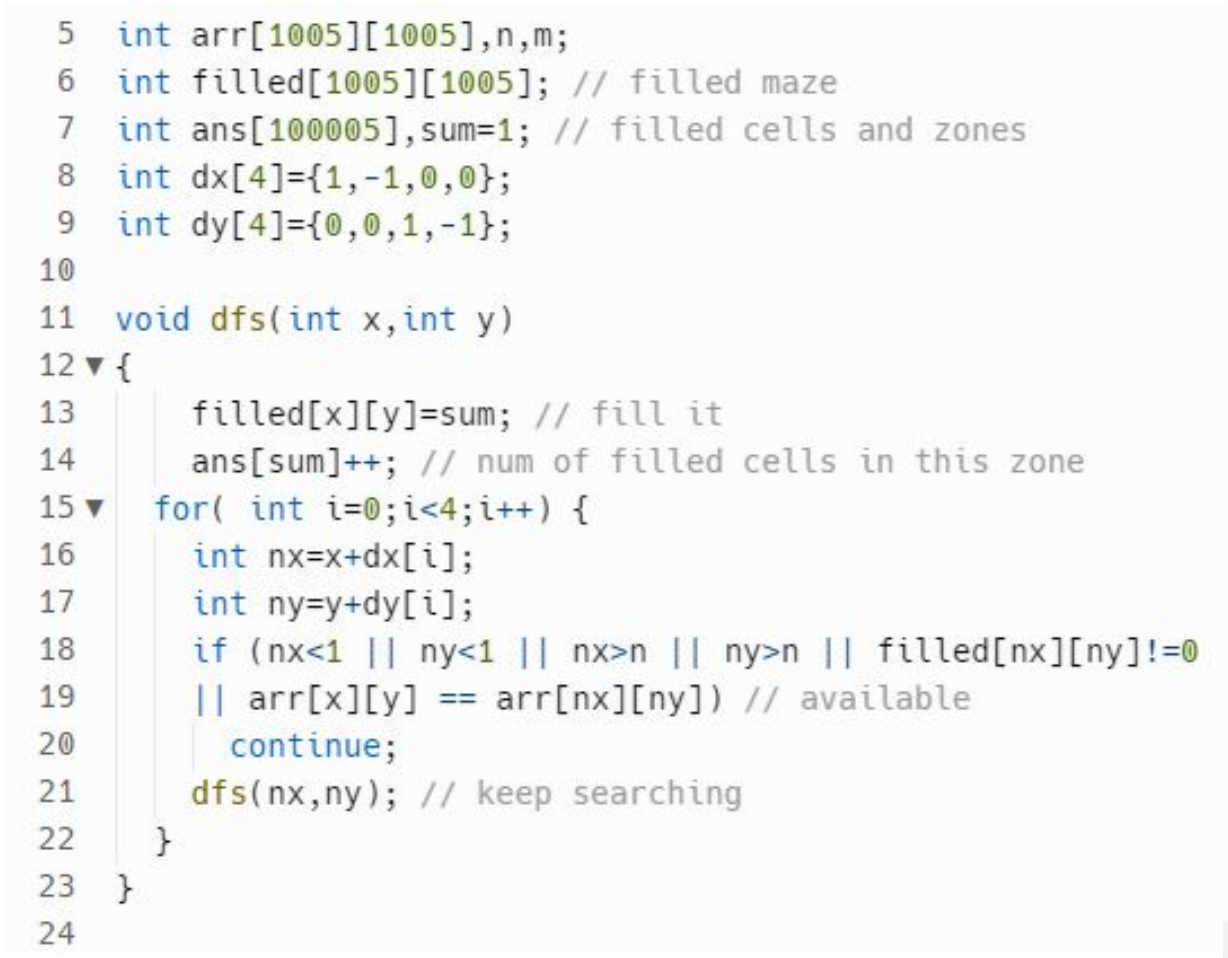
4

4

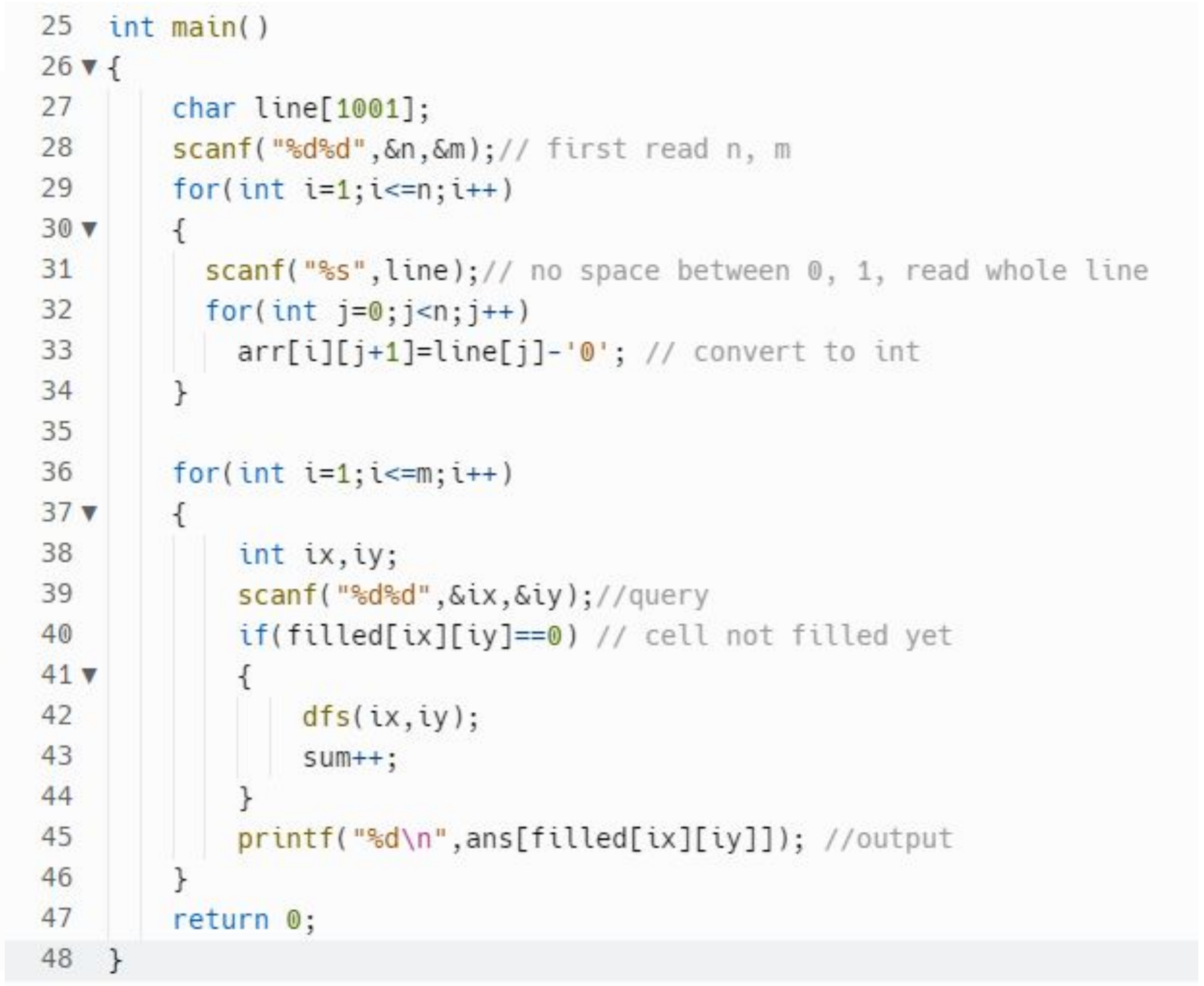
Note:

For 100% data， n≤1000,m≤100000。

Homework this week (P8110)



```
int arr[1005][1005],n,m;
int filled[1005][1005]; // filled maze
int ans[100005],sum=1; // filled cells and zones
int dx[4]={1,-1,0,0};
int dy[4]={0,0,1,-1};

void dfs(int x,int y)
{
    filled[x][y]=sum; // fill it
    ans[sum]++; // num of filled cells in this zone
  for( int i=0;i<4;i++) {
    int nx=x+dx[i];
    int ny=y+dy[i];
    if (nx<1 || ny<1 || nx>n || ny>n || filled[nx][ny]!=0
    || arr[x][y] == arr[nx][ny]) // available
      continue;
    dfs(nx,ny); // keep searching
  }
}

```



```
int main()
{
    char line[1001];
    scanf("%d%d",&n,&m);// first read n, m
    for(int i=1;i<=n;i++)
    {
      scanf("%s",line);// no space between 0, 1, read whole line
      for(int j=0;j<n;j++)
        arr[i][j+1]=line[j]-'0'; // convert to int
    }

    for(int i=1;i<=m;i++)
    {
        int ix,iy;
        scanf("%d%d",&ix,&iy);//query
        if(filled[ix][iy]==0) // cell not filled yet
        {
            dfs(ix,iy);
            sum++;
        }
        printf("%d\n",ans[filled[ix][iy]]); //output
    }
    return 0;
}
```

X-Camp

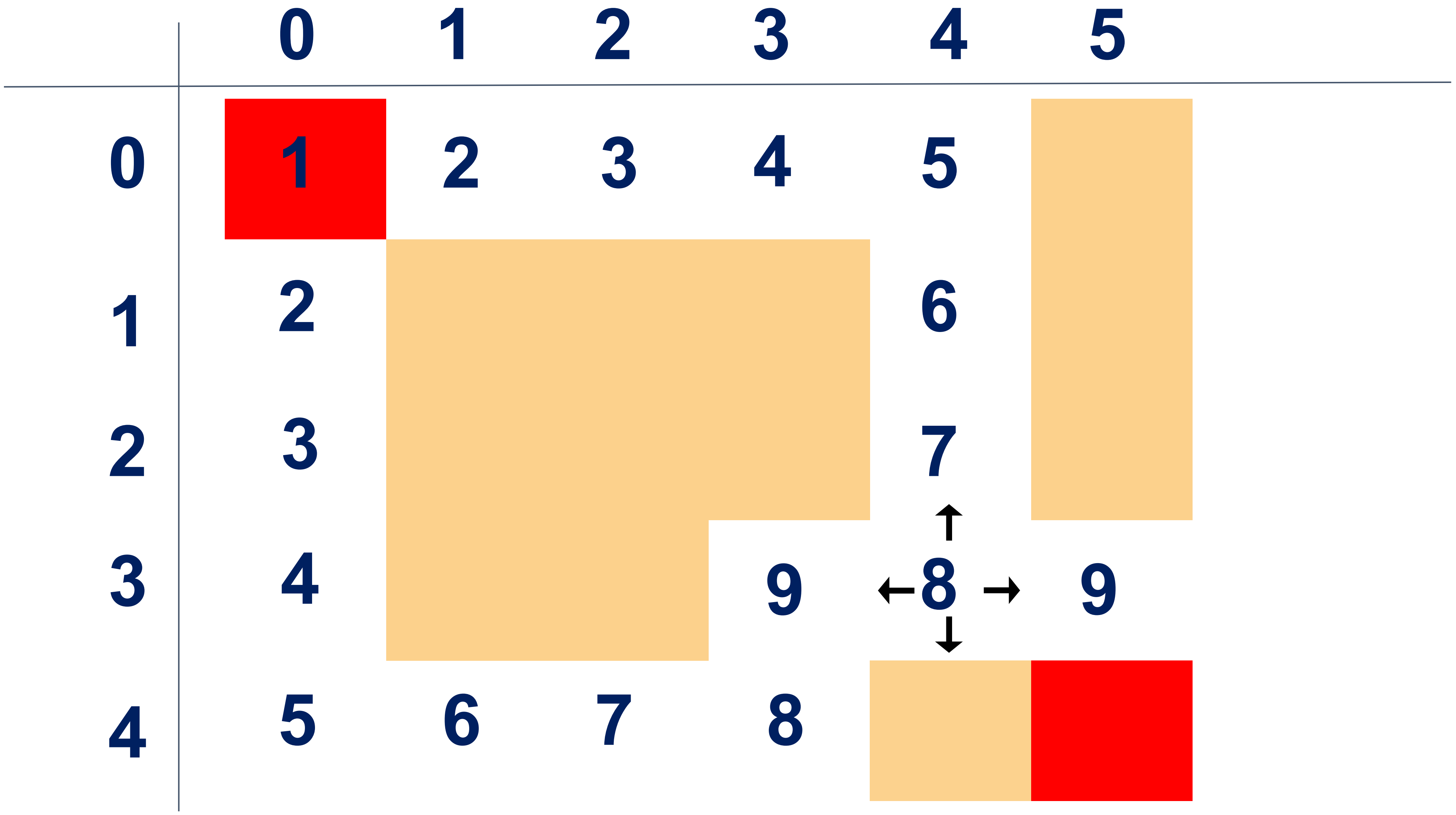
Breadth First Search

Search algorithm (BFS), implemented using queue

In this course, we'll go through some exercises on breadth-first search

Class : 202

Use BFS to walk the maze



	0	1	2	3	4	5
0	🟥 1	2	3	4	5	🟧
1	2	🟧	🟧	🟧	6	🟧
2	3	🟧	🟧	🟧	7	🟧
3	4	🟧	🟧	9	↑ ←8→ ↓	9
4	5	6	7	8	🟧	🟥

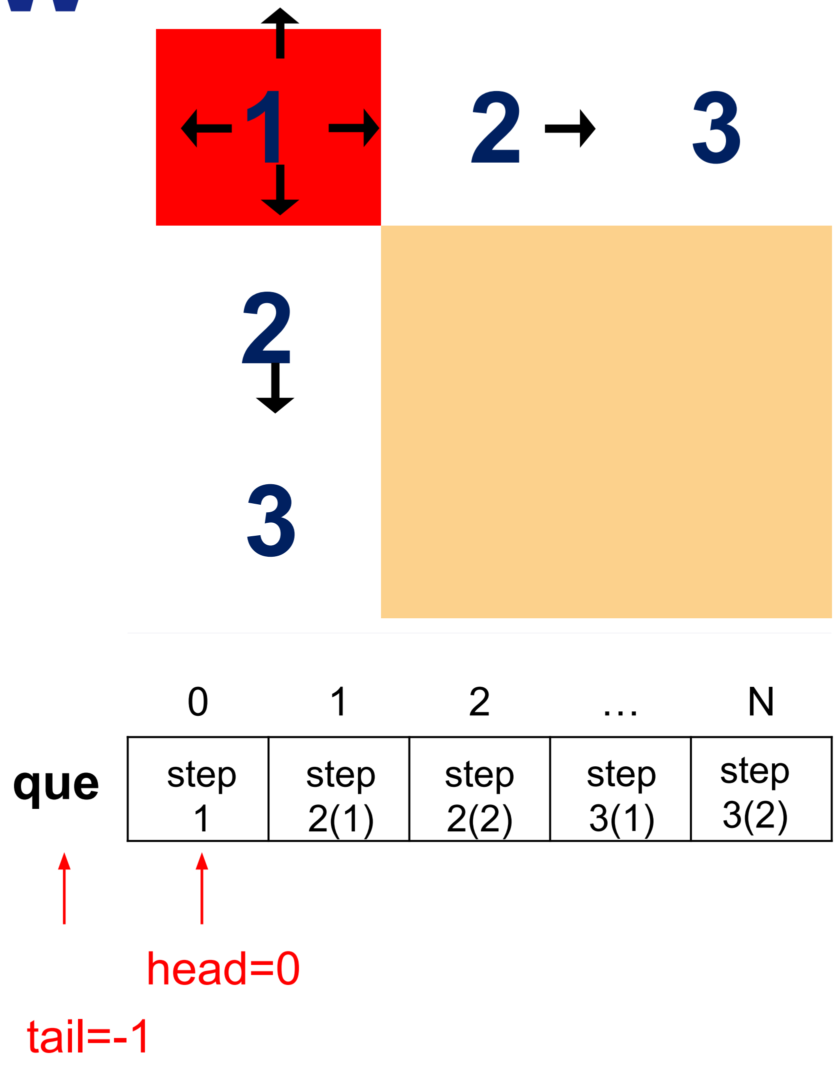
BFS review

BFS walks the maze:

Like a team starting from a starting point, there are the following rules:

- 1. Encounter a branch to split the team
- 2. Synchronized forward

Synchronous advance can use queue as the picture shows：



	0	1	2	…	N
que	step 1	step 2(1)	step 2(2)	step 3(1)	step 3(2)

BFS - find distance of the shortest path

Data structure - what do you need?

● maze - a 2D array of char or int	**int maze[N][N]**
● visited - a 2D array of bool (may not need)	**bool visited[N][N]**
● distance of each point - a 2D array of unsigned int	**unsigned dist[N][N]**
● struct node - (int x, y)	**struct node {int x, y;}**
● node queue - the queue of the current bases to expand from	**queue<node> q;**
● m expanding directions	**int dx[m], dy[m];**

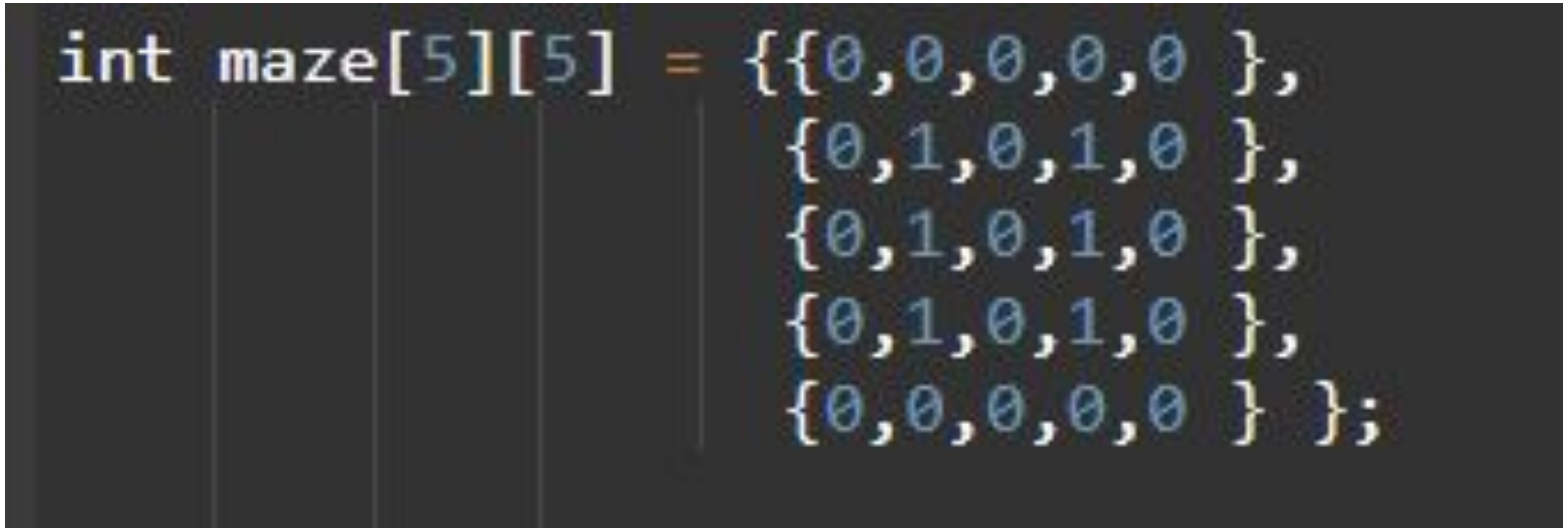
BFS - how to do it bfs (sx, sy)

Procedure - how to do it bfs (sx, sy)

1. Add the start node to the queue
2. While queue is not empty
   - a. Take the front node out of the queue
   - b. Visit all next nodes from this node
     - i. Set the distance (current dist + 1)
     - ii. If it's the end node, done.
     - iii. Mark visited true
     - iv. Push the new node to the queue

```
void bfs(sx,sy){
    q.push(sx,sy);
    while(!q.empty()){
        (x,y) = q.front();
        q.pop();
        for(all the directions){
            nx = x + dx[i];
            ny = t + dy[i];
            if(new location not valid)
        continue;
            dist[nx][ny] = dist[x][y] + 1;
            if( (nx,ny) is end point ){ done. };
            visited[nx][ny] = 1;
            q.push( point(nx,ny) );
        }
    }
}
```

BFS - find number of the shortest paths



```
int maze[5][5] = {{0,0,0,0,0 },
                  {0,1,0,1,0 },
                  {0,1,0,1,0 },
                  {0,1,0,1,0 },
                  {0,0,0,0,0 } };
```

What is the shortest path?

How many of them do we have?

What is extra work?

BFS - find number of the shortest paths

Data structure - what do you need?

maze - a 2D array of char of int

visited - a 2D array of bool

distance of each point - a 2D array of unsigned int
ways to each point - a 2D array of unsigned int

Struct node - (int x, y)

node queue - the queue of the current bases to expand from

m expanding directions

```
Int maze[N][N]

bool visited[N][N]

unsigned dist[N][N]
unsigned ways[N][N]

struct node {int x,
y;}

queue<node> q;

int dx[m], dy[m];
```

BFS - find number of shortest path

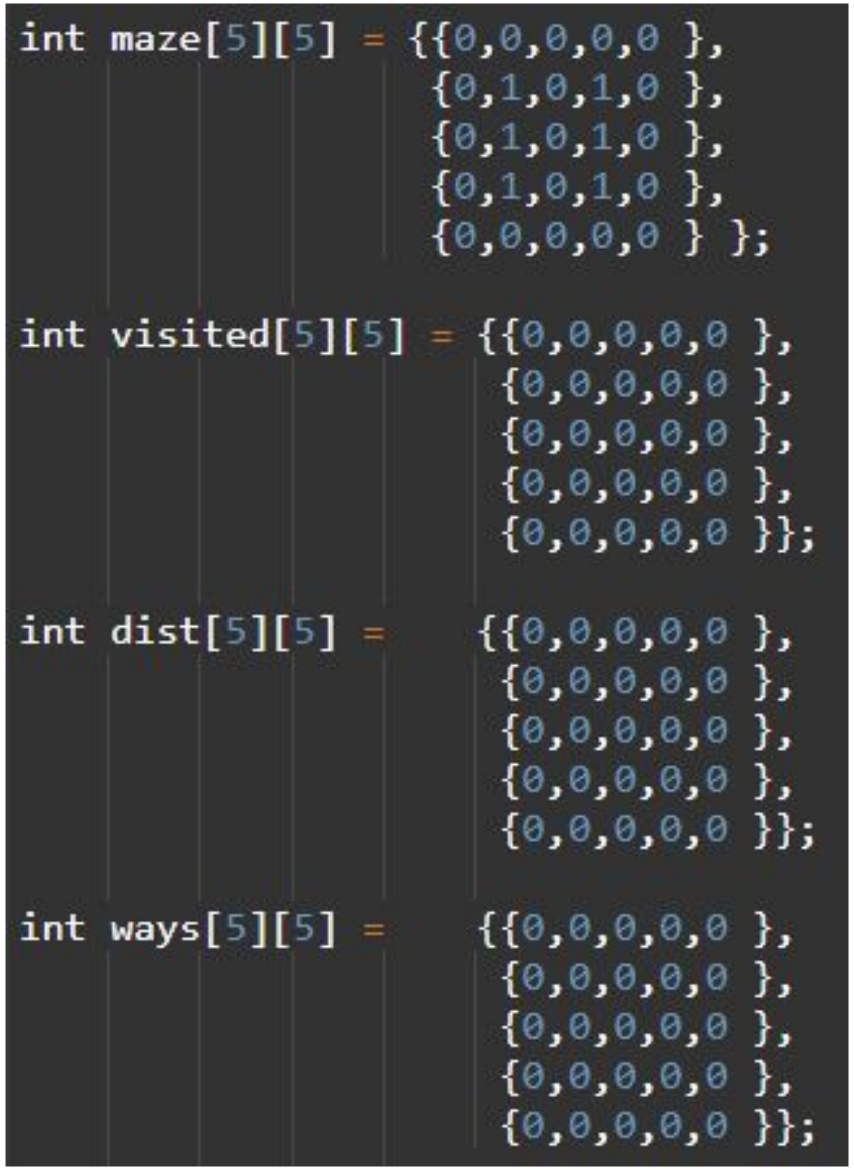
Procedure - how to do it dfs (sx, sy)

1. Add the start node to the queue
2. While queue is not empty
   a. Take the front node out of the queue
   b. Visit all next nodes from this node
      i. Set the distance (current dist + 1)?
      ii. **Calculate the ways to this node**
      iii. **If it’s the end node, done?**
      iv. Mark visited true
      v. Push the new node to the queue

```
q.push(sx,sy);
While (!q.empty()) {
        (x, y) = q.front();
        For (all the directions) {
                nx = x + dx[i]; ny = y+dy[i];
                Dist[nx][ny] = dist[x][y]+1;
                    // if (nx,ny) not visited
                ways[nx][ny] = ways[x][y];
                //what if the (nx,ny) already visited
                If ( (nx, ny) is end point) // done or keep going?
                Visited[nx][ny] = 1;
                q.push(point(nx, ny));
                }
    }
    Cout << dist[ex][ey]
```

Refer to https://stackoverflow.com/questions/15211611/number-of-shortest-paths-in-a-graph/15212503

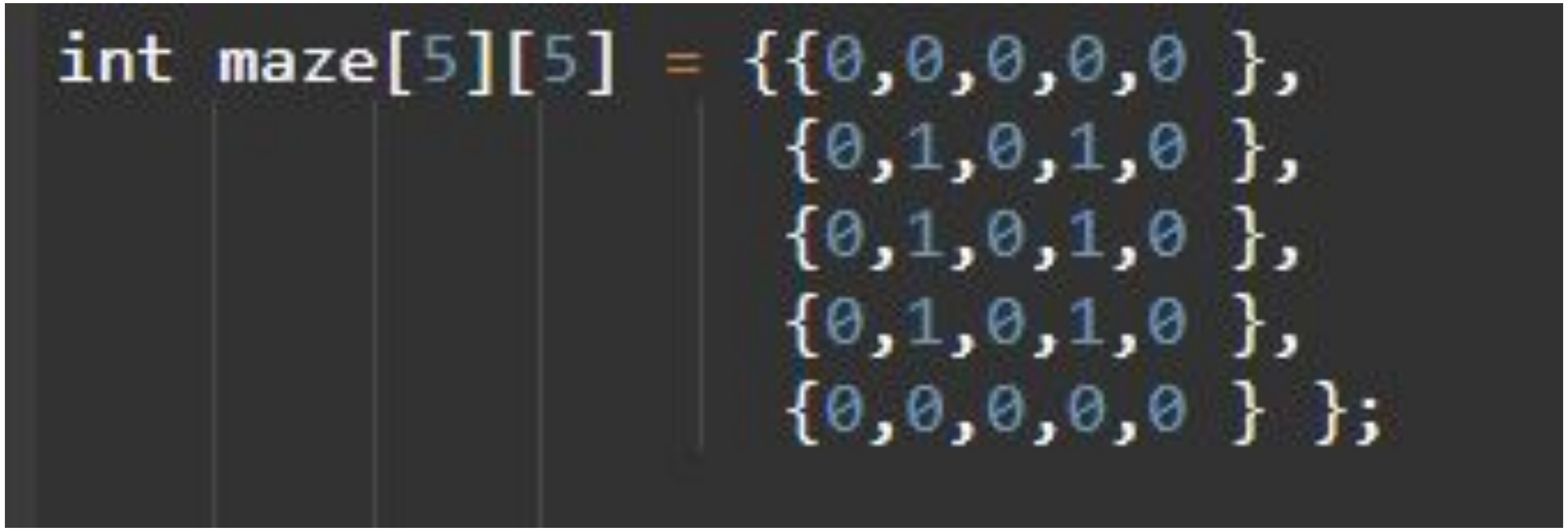
BFS - find number of shortest path



```
int maze[5][5] = {{0,0,0,0,0 },
                  {0,1,0,1,0 },
                  {0,1,0,1,0 },
                  {0,1,0,1,0 },
                  {0,0,0,0,0 } };

int visited[5][5] = {{0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 }};

int dist[5][5] =    {{0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 }};

int ways[5][5] =    {{0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 },
                     {0,0,0,0,0 }};
```

```
q.push(sx,sy);
While (!q.empty()) {
        (x, y) = q.front();
        For (all the directions) {
                nx = x + dx[i]; ny = y+dy[i];
                Dist[nx][ny] = dist[x][y]+1;
                    // if (nx,ny) not visited
                ways[nx][ny] = ways[x][y];
                //what if the (nx,ny) already visited
                If ( (nx, ny) is end point) // done or keep going?
                Visited[nx][ny] = 1;
                q.push(point(nx, ny));
                }
          }
          Cout << dist[ex][ey]
```

BFS Practice - num of shortest path



```
int maze[5][5] = {{0,0,0,0,0 },
                  {0,1,0,1,0 },
                  {0,1,0,1,0 },
                  {0,1,0,1,0 },
                  {0,0,0,0,0 } };
```

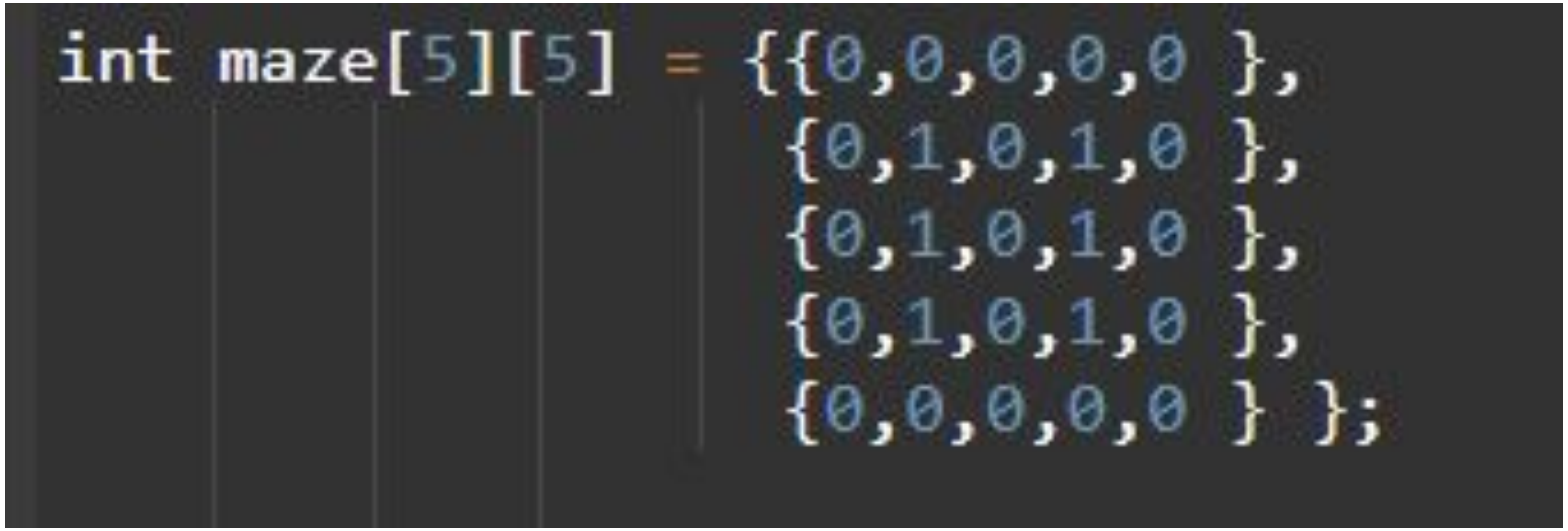
https://onlinegdb.com/OBBdGdG5n

Find number of shortest path from (0, 0) to (4, 4). Should be 3

Sample code - num of shortest path

https://onlinegdb.com/Pv0AGz57a

Find number of shortest path from (0, 0) to (4, 4)

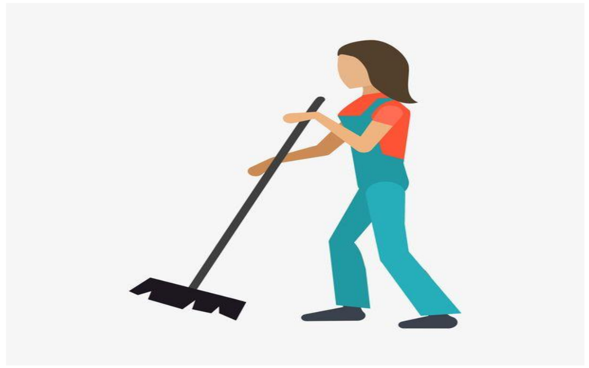


```
int maze[5][5] = {{0,0,0,0,0 },
                  {0,1,0,1,0 },
                  {0,1,0,1,0 },
                  {0,1,0,1,0 },
                  {0,0,0,0,0 } };
```

BFS - Breadth First Search

BFS - like water pouring on a surface, starting from one point and expanding to all directions layer by layer.

DFS - like the Roomba cleaning the house, going to one direction till it cannot go, and then turn to another direction.

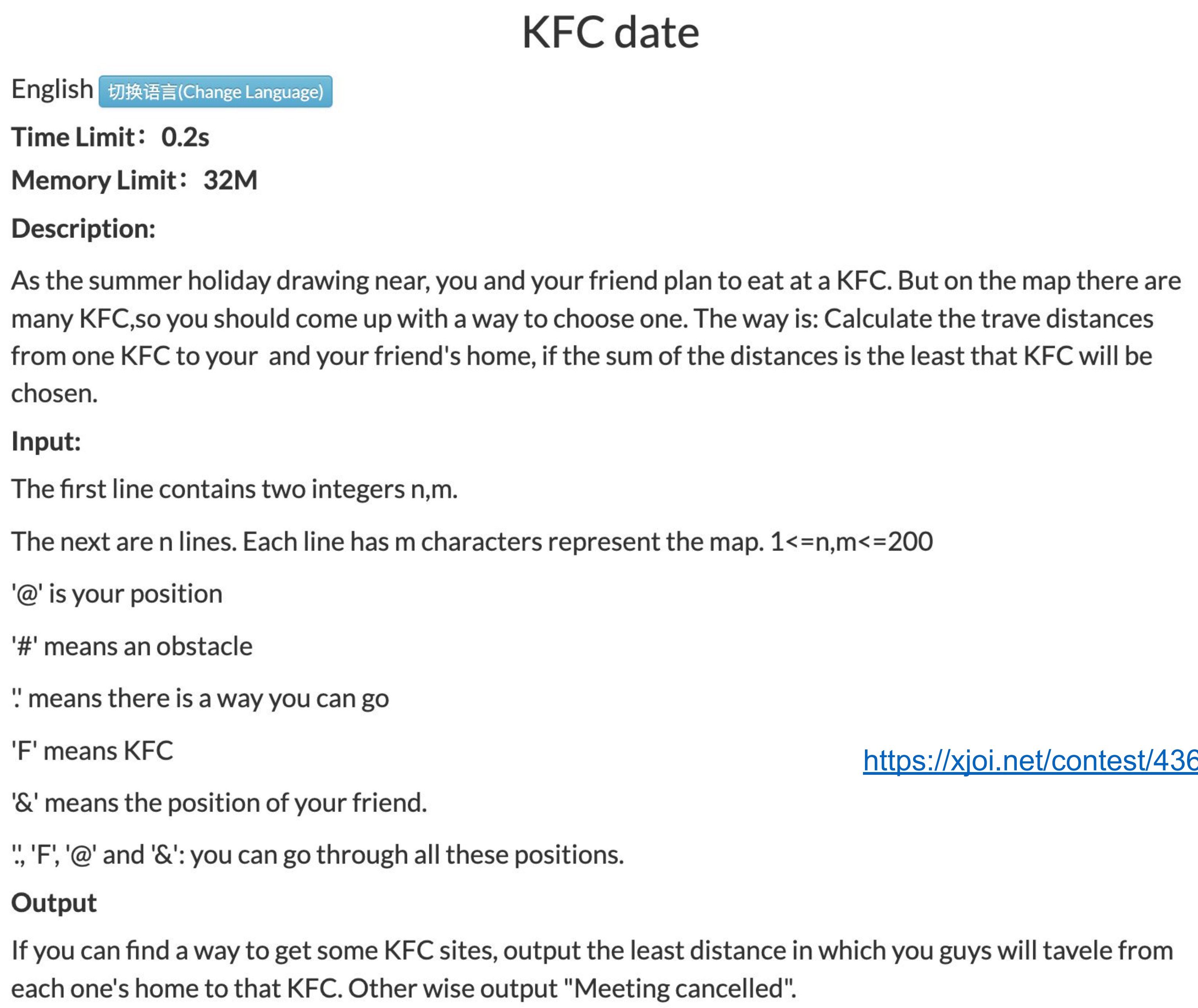


By this analogy, which method is more predictable to reach a given point of certain distance?

What if I ask you to find the shortest distance of multiple points from the start point? How will these 2 methods work differently?

BFS vs. DFS - typical problems

Problem	BFS	DFS
Floodfill - just need to visit all nodes.	Yes	Yes
Find shortest distance Additionally track the distance to each node	Better	backtracking
Find shortest distance and the number of shortest paths Additionally track the number of ways to each node	Better	backtracking
Find shortest distances to multiple points	Yes	No



KFC date

English 切换语言(Change Language)

Time Limit: 0.2s

Memory Limit: 32M

Description:

As the summer holiday drawing near, you and your friend plan to eat at a KFC. But on the map there are many KFC,so you should come up with a way to choose one. The way is: Calculate the trave distances from one KFC to your and your friend's home, if the sum of the distances is the least that KFC will be chosen.

Input:

The first line contains two integers n,m.

The next are n lines. Each line has m characters represent the map. 1<=n,m<=200

'@' is your position

'#' means an obstacle

'.' means there is a way you can go

'F' means KFC

'&' means the position of your friend.

'.', 'F', '@' and '&': you can go through all these positions.

Output

If you can find a way to get some KFC sites, output the least distance in which you guys will tavele from each one's home to that KFC. Other wise output "Meeting cancelled".

https://xjoi.net/contest/4362 problem 2

KFC（P3586）

@	.	#	F
.	.	.	.
.	#	.	.
F	.	.	&

0	1	inf	5
1	2	3	4
2	inf	4	5
3	4	5	6

6	5	inf	3
5	4	3	2
4	inf	2	1
3	2	1	0

KFC 3586

@		
		F
&		

	#	F	F						
@					F		#		F
#	#					#	F		
F			#						
		#	&		F	#			
		F	#						

In class exercise: https://xjoi.net/contest/4362 problem 2

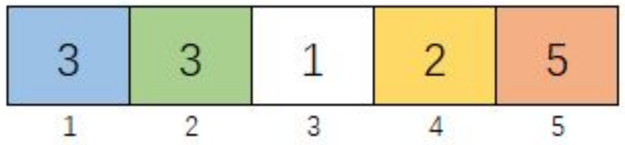
KFC 3586

In class exercise: https://xjoi.net/contest/4362 problem 2

Sample code: https://onlinegdb.com/HNqwNCDEy

Strange elevator (P1642)

https://xjoi.net/contest/4435



step 1

step 2

step 3

Sample input:

5 1 5

3 3 1 2 5

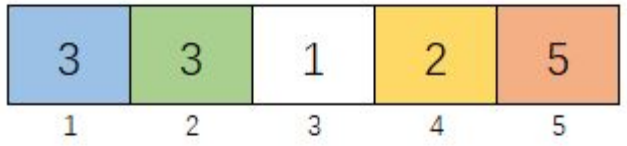
Sample output:

3

Strange elevator

Use BFS

- Input data
- Push the start to the queue
- Add all reachable floors to queue
- Until it reaches the destination or fails
- ps： Watch out for out of bounds



XJOI problem:1642

Sea Battle (P9431)

In the game, some ships are placed on a square table, and each ship cannot touch other ships. In this problem, we only consider boats to be rectangles, all boats are rectangles composed of "*". Write a program to find the total number of ships placed in the game.

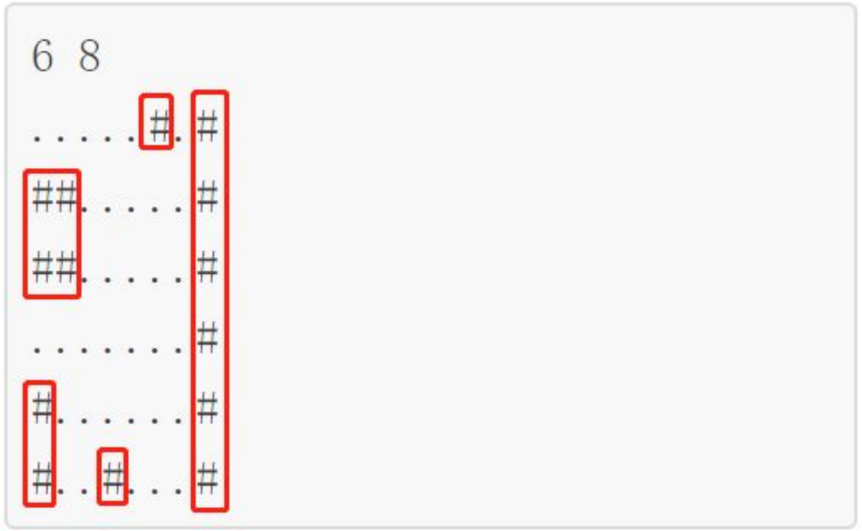
input sample

```
6 8
.....#.#
##.....#
##.....#
.......#
#......#
 #..#...#
```

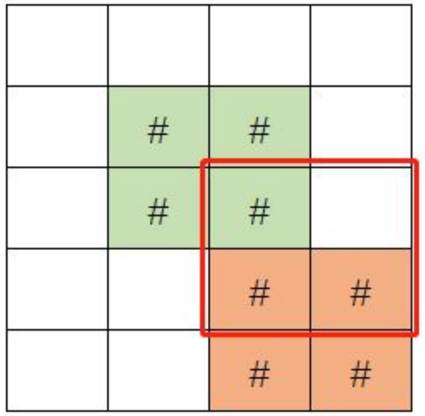
output sample

```
There are 5 ships.
```

Sea Battle (P9431)



```
6 8
.....#.#
##.....#
##.....#
.......#
#......#
#..#...#
```



- As can be seen from the figure, the five ships are five connected areas

- This is the case of mutual contact, please observe its characteristics

- How do we check the shape is rectangle?

Minimum turns (P8121)

Given a map, please find the least times of making turns from the starting point to the destination,if no accessible road,output -1

input:
The first line contains n and m, which are rows and columns.

Next describe the n*m matrix. 0 means ok to go, 1 means not.

Next input the position of the starting point and destination.

output:
output the answer

sample input 1:

```
5 7
1 0 0 0 0 1 0
0 0 1 0 1 0 0
0 0 0 0 1 0 1
0 1 1 0 0 0 0
0 0 0 0 1 1 0
1 3 1 7
```

sample output 1:

```
5
```

Minimum turns (P8121)

What's given?

- N, M: size of the maze matrix
- NxM matrix: 0 path; 1 wall
- start and end points: sx, sy, ex, ey

What's asked?

- the minimum number of turns taken from the start port to the end point.

What's your strategy?

- Try all possible paths.
- Count how many turns happened.
- How to count number of turns? How do we know there is a turn?

If the new direction is different, it's a turn.

```
1 0 0 0 0 1 0
0 0 1 0 1 0 0
0 0 0 0 1 0 1
0 1 1 0 0 0 0
0 0 0 0 1 1 0
```

Minimum turns (P8121)

What data structure we need?

- N, M: size of the maze matrix
- NxM matrix: 0 path; 1 wall
- start and end points: sx, sy, ex, ey

➔ int maze[101][101], turn[101][101], dir[101][101];

➔ Do we need visited[][]?

➔ what should be initial value?

➔ Does starting point have a direction?

```
1000010
0010100
0000101
0110000
0000110
```