Another searching method

Another searching method is breadth-first search. Each of different arrangement of numbers is considered as one state. Each state has 12 attributes: rows n , columns m , an two dimensional array representing location of each number $board$, total number of zeros $neocount$, a link pointing to father, a one dimensional array of horizontal coordinates about zeros x_0 , a one dimensional array of vertical coordinates about zeros y_0 , total number of blocks $numberofblock$, an two dimensional array recording all the values in blocks of $1 * 2$ $block1$, an two dimensional array recording all the values in blocks of $2 * 1$ $block2$, an two dimensional array recording all the values in blocks of $2 * 2$ $block3$ and one direction moving to get the current state.

```
int n;
    int m;
    int[][] board = new int[n][m];
    final int neoCount;
    Node father;
    int[] x0;
    int[] y0;


    //block
    int numberOfblock;
    //行数代表该种block的种类
    int[][] block1;//1*2的block,以二维数组的形式记录，行数代表该种block的个数，每行的元素代表该block中的元素
    int[][] block2;//2*1的block，以二维数组的形式记录，行数代表该种block的个数，每行的元素代表该block中的元素
    int[][] block3;//2*2的block，以二维数组的形式记录，行数代表该种block的个数，每行的元素代表该block中的元素


    String movement;
```

An ArrayList named $close$ is created to record the states that have been searched and a queue named $queue$ is created to record the states that haven't been searched and indicates the next state needed to be searched. When moving one step, the gotten state is added into $queue$ if the state hasn't been searched while the $queue$ dequeues one new state to search if the state has been searched. The ideas of the two methods are similar.

The following is comparison of the time complexity of them. The sample input is following:

Sample Input 1

```
2 2
1 2
0 3
0
```

Sample Input 2

```
2 2
2 1
0 3
0
```

Sample Input 3

```
3 3
1 2 3
4 0 0
7 5 6
1
5 1*2
```

Sample Input 4

```
5 4
3 7 4 8
1 2 0 11
5 6 15 10
9 14 0 12
13 17 0 16
2
12 2*1
1 2*2
```

The following are the time used for the above 4 inputs using breadth-first search:

```
D:\Study in SUSTech\First semester of junior year\dsaaB\lab\project\Numeric_Klotski> d: && cd "d:\Study in SUSTech\First semester of junior year\dsaaB\lab\project\Numeric_Klotski" && cmd /C ""C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_9jl9qgyvfb3upo2hqbc42xism.jar Breadth_First.Run "
2 2
1 2
0 3
0
Yes
1
3 L
6080 ms
```

```
D:\Study in SUSTech\First semester of junior year\dsaaB\lab\project\Numeric_Klotski> d: && cd "d:\Study in SUSTech\First semester of junior year\dsaaB\lab\project\Numeric_Klotski" && cmd /C ""C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_9jl9qgyvfb3upo2hqbc42xism.jar Breadth_First.Run "
2 2
2 1
0 3
0
No
15479 ms
```

```
D:\Study in SUSTech\First semester of junior
year\dsaaB\lab\project\Numeric_Klotski> d: && cd "d:\Study in
SUSTech\First semester of junior
year\dsaaB\lab\project\Numeric_Klotski" && cmd /C ""C:\Program
Files\Java\jdk1.8.0_271\bin\java.exe" -cp
C:\WINDOWS\TEMP\cp_9jl9qgyvfb3upo2hqbc42xism.jar Breadth_First.Run
"
3 3
1 2 3
4 0 0
7 5 6
1
5 1*2
Yes
1
5 U
31883 ms
```

For the 4th input, the breadth-first search method can't solve the problem in a reasonable time.

The following are the time used for the above 4 inputs using optimized search(A STAR Algorithm):

```
D:\Study in SUSTech\First semester of junior
year\dsaaB\lab\project\Numeric_Klotski> d: && cd "d:\Study in
SUSTech\First semester of junior
year\dsaaB\lab\project\Numeric_Klotski" && cmd /C ""C:\Program
Files\Java\jdk1.8.0_271\bin\java.exe" -cp
C:\WINDOWS\TEMP\cp_9jl9qgyvfb3upo2hqbc42xism1.jar Node_and_Main.Run
"
2 2
1 2
0 3
0
Yes
1
3 L
4709 ms
```

```
D:\Study in SUSTech\First semester of junior
year\dsaaB\lab\project\Numeric_Klotski> cmd /C ""C:\Program
Files\Java\jdk1.8.0_271\bin\java.exe" -cp
C:\WINDOWS\TEMP\cp_9jl9qgyvfb3upo2hqbc42xism.jar Node_and_Main.Run
"
2 2
2 1
0 3
0
No
10298 ms
```

```
D:\Study in SUSTech\First semester of junior
year\dsaaB\lab\project\Numeric_Klotski> cmd /C ""C:\Program
Files\Java\jdk1.8.0_271\bin\java.exe" -cp
C:\WINDOWS\TEMP\cp_9jl9qgyvfb3upo2hqbc42xism.jar Node_and_Main.Run
"
3 3
1 2 3
4 0 0
7 5 6
1
5 1*2
Yes
1
5 U
3972 ms
```

```
D:\Study in SUSTech\First semester of junior
year\dsaaB\lab\project\Numeric_Klotski> cmd /C ""C:\Program
Files\Java\jdk1.8.0_271\bin\java.exe" -cp
C:\WINDOWS\TEMP\cp_9jl9qgyvfb3upo2hqbc42xism.jar Node_and_Main.Run
"
5 4
3 7 4 8
1 2 0 11
5 6 15 10
9 14 0 12
13 17 0 16
2
```

```
12 2*1
1 2*2
Yes
16
15 D
10 L
11 D
8 D
4 R
7 R
7 D
3 R
3 R
1 U
10 L
11 L
12 U
9 U
13 U
17 L
4212 ms
```

According to the comparison above, it is obvious that the performance of A STAR Algorithm is better than that of breadth-first search method. The reason is that for breadth-first search method, it needs to search every new state while it only need to search the state whose cost is minimized in son state for A STAR Algorithm.

The code for breadth-first search method is following:

```
package Breadth_First;

import java.util.ArrayList;
import java.util.PriorityQueue;

import edu.princeton.cs.algs4.Queue;

public class Node {
    int n;
    int m;
    int[][] board = new int[n][m];
    final int neoCount;
```

```
    Node father;
    int[] x0;
    int[] y0;


    //block
    int numberOfblock;
    //行数代表该种block的种类
    int[][] block1;//1*2的block,以二维数组的形式记录，行数代表该种block的个数，每行的元素代表该block中的元素
    int[][] block2;//2*1的block，以二维数组的形式记录，行数代表该种block的个数，每行的元素代表该block中的元素
    int[][] block3;//2*2的block，以二维数组的形式记录，行数代表该种block的个数，每行的元素代表该block中的元素


    String movement;


    //初始化Node
    public Node(int n, int m, int neoCount, Node father){
        this.n = n;
        this.m = m;
        this.board = new int[n][m];
        this.neoCount = neoCount;
        this.father = father;
        this.x0 = new int[neoCount];
        this.y0 = new int[neoCount];

        this.numberOfblock=father.numberOfblock;
        this.block1=new int[n*m/2][2];
        this.block2=new int[n*m/2][2];
        this.block3=new int[n*m/4][4];



    }

    public Node(int[][] board, Node father,int numberOfblock,int[][] allOfBlock){
        int count = 0;
        for (int i = 0; i < board.length; i++) {
```

```
            for (int j = 0; j < board[0].length; j++) {
                if(board[i][j] == 0){
                    count++;
                }
            }
        }
        this.neoCount = count;
        int neo = 0;
        this.n = board.length;
        this.m = board[0].length;
        int[][] array = new int[board.length][board[0].length];
        for(int i = 0; i < board.length; i++){
            System.arraycopy(board[i], 0, array[i], 0,
board[0].length);
        }
        this.board = array.clone();
        this.father = father;
        this.x0 = new int[count];
        this.y0 = new int[count];
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                if(board[i][j] == 0){
                    x0[neo] = i;
                    y0[neo] = j;
                    neo++;
                }
            }
        }

        //block的一些初始化
        this.numberOfblock=numberOfblock;
        this.block1=new int[n*m/2][2];
        this.block2=new int[n*m/2][2];
        this.block3=new int[n*m/4][4];

        //根据allOfBlock中每行第二列元素来判断属于哪种block，并将其存入相应的
block中
        int numberOfblock1=0;
        int numberOfblock2=0;
        int numberOfblock3=0;
```

```
        //allofblock为numofblock*2的二维数组，第二列代表block的种类，第一
列代表block中左上角的数字，
        if(numberOfblock>0){
            for(int i=0;i<numberOfblock;i++){
                //判断是否属于第一种block 1*2
                if(allOfBlock[i][1]==1){
                    //将属于该block的值存入其中
                    this.block1[numberOfblock1][0]=allOfBlock[i]
[0];
                    this.block1[numberOfblock1]
[1]=board[coordinateOfTopInBlock(board, allOfBlock[i][0])[0]]
[coordinateOfTopInBlock(board, allOfBlock[i][0])[1]+1];
                    numberOfblock1=numberOfblock1+1;

                }
                //判断是否属于第二种block 2*1
                if(allOfBlock[i][1]==2){
                    //将属于该block的值存入其中
                    this.block2[numberOfblock2][0]=allOfBlock[i]
[0];
                    this.block2[numberOfblock2]
[1]=board[coordinateOfTopInBlock(board, allOfBlock[i][0])[0]+1]
[coordinateOfTopInBlock(board, allOfBlock[i][0])[1]];
                    numberOfblock2=numberOfblock2+1;

                }
                //判断是否属于第二种block 2*2
                if(allOfBlock[i][1]==3){
                    //将属于该block的值存入其中
                    this.block3[numberOfblock3][0]=allOfBlock[i]
[0];
                    this.block3[numberOfblock3]
[1]=board[coordinateOfTopInBlock(board, allOfBlock[i][0])[0]]
[coordinateOfTopInBlock(board, allOfBlock[i][0])[1]+1];
                    this.block3[numberOfblock3]
[2]=board[coordinateOfTopInBlock(board, allOfBlock[i][0])[0]+1]
[coordinateOfTopInBlock(board, allOfBlock[i][0])[1]];
                    this.block3[numberOfblock3]
[3]=board[coordinateOfTopInBlock(board, allOfBlock[i][0])[0]+1]
[coordinateOfTopInBlock(board, allOfBlock[i][0])[1]+1];
                    numberOfblock3=numberOfblock3+1;
                }
```

```
            }
        }
    }

    //复制Node
    public Node(Node node){
        this.n = node.n;
        this.m = node.m;
        int[][] array=new int[node.board.length]
[node.board[0].length];
        this.x0 = new int[node.x0.length];
        this.y0 = new int[node.y0.length];
        for(int i = 0; i < node.board.length; i++){
            System.arraycopy(node.board[i], 0, array[i], 0,
node.board[0].length);
        }
        this.board = array.clone();
        this.neoCount = node.neoCount;
        this.father = node;
        System.arraycopy(node.x0, 0, this.x0, 0, node.x0.length);
        System.arraycopy(node.y0, 0, this.y0, 0, node.y0.length);



        this.numberOfblock=node.numberOfblock;

        //先判断各种block数组是否为空，不为空就拷贝
        //判断第一种block 1*2是否为空

 if(node.block1!=null&&node.block1.length!=0&&node.block1[0].length
!=0){
            int[][] array1=new int[node.block1.length]
[node.block1[0].length];
            for(int i = 0; i < node.block1.length; i++){
                System.arraycopy(node.block1[i], 0, array1[i], 0,
node.block1[0].length);
            }
            this.block1 = array1.clone();
        }
        //判断第二种block 2*1是否为空

 if(node.block2!=null&&node.block2.length!=0&&node.block2[0].length
!=0){
```

```
            int[][] array2=new int[node.block2.length]
[node.block2[0].length];
            for(int i = 0; i < node.block2.length; i++){
                System.arraycopy(node.block2[i], 0, array2[i], 0,
node.block2[0].length);
            }
            this.block2 = array2.clone();

        }
        //判断第三种block 2*2是否为空

 if(node.block3!=null&&node.block3.length!=0&&node.block3[0].length
!=0){
            int[][] array3=new int[node.block3.length]
[node.block3[0].length];
            for(int i = 0; i < node.block3.length; i++){
                System.arraycopy(node.block3[i], 0, array3[i], 0,
node.block3[0].length);
            }
            this.block3 = array3.clone();

        }




    }

    public static String printState(Node state){
        StringBuilder output = new StringBuilder();
        for(int i=0;i<state.n;i++){
            for(int j=0;j<state.m;j++){
                output.append(state.board[i][j]).append(" ");
            }
            output.append("\n");
        }
        output.append("\n");
        return output.toString();
    }

    public static String arrayToString(Node state){
        String s="";
```

```
        for(int i =0;i<state.n;i++){
            for(int j =0;j<state.m;j++){
                s=s.concat(String.valueOf(state.board[i][j]));//此处
可能有问题
            }
        }
        return s;
    }

    public boolean isComplete(){
        int mistake = 0;
        int[] total = new int[n * m];
        for (int i = 0; i < n * m; i++) {
            total[i] = this.board[i / board[0].length][i %
board[0].length];
        }
        for (int i = 0; i < total.length - neoCount; i++) {
            if(total[i] != i + 1){
                mistake++;
            }
        }
        for (int i = total.length - neoCount; i < total.length;
i++) {
            if (total[i] != 0){
                mistake++;
            }
        }
        return mistake == 0;
    }

    public void findNext(Queue<Node> queue, ArrayList<String>
close){//queue代表为搜索的状态，close代表为已经搜索过的状态
        Node currentState = this;
        close.add(arrayToString(currentState));

        for (int i = 0; i < currentState.x0.length; i++) {
            int x = currentState.x0[i];
            int y = currentState.y0[i];


            //上面能不能换
            Node next1= up(currentState, x, y, i);
```

```
            //以下是判断移动的那个数字属于哪种block，属于block则需要打印该
block中最小值及动作
            if (!close.contains(arrayToString(next1))){
                //判断属于哪种类型的block，如果是block，则需要打印左上角的数
值和动作
                boolean flagofblock1 = find(currentState.block1,
currentState.board[x - 1][y]);
                boolean flagofblock2 = find(currentState.block2,
currentState.board[x - 1][y]);
                boolean flagofblock3 = find(currentState.block3,
currentState.board[x - 1][y]);


 if(flagofblock1==false&&flagofblock2==false&&flagofblock3==false){
                    //属于1*1，直接打印就行
                    next1.movement = currentState.board[x - 1][y] +
" D";
                    queue.enqueue(next1);

                }


 if(flagofblock1==true&&flagofblock2==false&&flagofblock3==false){
                    //属于1*2，找左上角的值，并打印以及相应动作
                    int j=rangeofblock1(currentState, x-1, y);
                    if(j==1){//代表就是左上角的值
                        next1.movement = currentState.board[x - 1]
[y] + " D";
                        queue.enqueue(next1);
                    }
                    else{//在它左边的数字就是左上角的值
                        next1.movement = (currentState.board[x - 1]
[y-1]) + " D";
                        queue.enqueue(next1);
                    }

                }


 if(flagofblock1==false&&flagofblock2==true&&flagofblock3==false){
                    //属于2*1，找左上角的值，并打印以及相应动作
```

```
                    next1.movement = currentState.board[x - 2][y] +
" D";
                    queue.enqueue(next1);
                }


 if(flagofblock1==false&&flagofblock2==false&&flagofblock3==true){
                    //属于2*2，找左上角的值，并打印以及相应动作
                    int j=rangeofblock3(currentState, x-1, y);
                    if(j==3){//代表就是左下角的值
                        next1.movement = currentState.board[x - 2]
[y] + " D";
                        queue.enqueue(next1);
                    }
                    else{//j==4,代表是右下角的值
                        next1.movement = (currentState.board[x - 2]
[y-1]) + " D";
                        queue.enqueue(next1);
                    }

                }









            }

            //下边能不能换
            Node next2=down(currentState, x, y, i);

            //以下是判断移动的那个数字属于哪种block，属于block则需要打印该
block中最小值及动作
            if (!close.contains(arrayToString(next2))){
                //判断属于哪种类型的block，如果是block，则需要打印左上角的数
值和动作
                boolean flagofblock1 = find(currentState.block1,
currentState.board[x + 1][y]);
                boolean flagofblock2 = find(currentState.block2,
currentState.board[x + 1][y]);
```

```
                boolean flagofblock3 = find(currentState.block3,
currentState.board[x + 1][y]);


 if(flagofblock1==false&&flagofblock2==false&&flagofblock3==false){
                    //属于1*1，直接打印就行
                    next2.movement = currentState.board[x + 1][y] +
" U";
                    queue.enqueue(next2);
                }


 if(flagofblock1==true&&flagofblock2==false&&flagofblock3==false){
                    //属于1*2，找左上角的值，并打印以及相应动作
                    int j=rangeofblock1(currentState, x+1, y);
                    if(j==1){//代表就是左上角的值
                        next2.movement = currentState.board[x + 1]
[y] + " U";
                        queue.enqueue(next2);
                    }
                    else{//在它左边的数字就是左上角的值
                        next2.movement = currentState.board[x + 1]
[y-1] + " U";
                        queue.enqueue(next2);
                    }

                }


 if(flagofblock1==false&&flagofblock2==true&&flagofblock3==false){
                    //属于2*1，找左上角的值，并打印以及相应动作
                    next2.movement = currentState.board[x + 1][y] +
" U";
                    queue.enqueue(next2);
                }


 if(flagofblock1==false&&flagofblock2==false&&flagofblock3==true){
                    //属于2*2，找左上角的值，并打印以及相应动作
                    int j=rangeofblock3(currentState, x+1, y);
                    if(j==1){//代表就是左上角的值
```

```
                        next2.movement = currentState.board[x + 1]
[y] + " U";
                        queue.enqueue(next2);
                    }
                    else{//j==2,代表是右上角的值
                        next2.movement = currentState.board[x + 1]
[y-1] + " U";
                        queue.enqueue(next2);
                    }

                }









            }



            //左边能不能换
            Node next3=left(currentState, x, y, i);

            //以下是判断移动的那个数字属于哪种block，属于block则需要打印该
block中最小值及动作
            if (!close.contains(arrayToString(next3))){
                //判断属于哪种类型的block，如果是block，则需要打印左上角的数
值和动作
                boolean flagofblock1 = find(currentState.block1,
currentState.board[x][y-1]);
                boolean flagofblock2 = find(currentState.block2,
currentState.board[x][y-1]);
                boolean flagofblock3 = find(currentState.block3,
currentState.board[x][y-1]);


 if(flagofblock1==false&&flagofblock2==false&&flagofblock3==false){
                    //属于1*1，直接打印就行
                    next3.movement = currentState.board[x][y - 1] +
" R";
                    queue.enqueue(next3);
```

```
                }


 if(flagofblock1==true&&flagofblock2==false&&flagofblock3==false){
                    //属于1*2，有解的情况下是j==2
                    next3.movement = currentState.board[x][y - 2] +
" R";
                    queue.enqueue(next3);
                }


 if(flagofblock1==false&&flagofblock2==true&&flagofblock3==false){
                    //属于2*1
                    int j=rangeofblock2(currentState, x, y-1);
                    if(j==1){//代表就是上面的值
                        next3.movement = currentState.board[x][y -
1] + " R";
                        queue.enqueue(next3);
                    }
                    else{//代表就是下面的值
                        next3.movement = currentState.board[x-1][y
- 1] + " R";
                        queue.enqueue(next3);
                    }


                }


 if(flagofblock1==false&&flagofblock2==false&&flagofblock3==true){
                    //属于2*2，直接打印就行
                    int j=rangeofblock3(currentState, x, y-1);
                    //j在有解的情况下只有两个值，2和4
                    if(j==2){//代表就是右上角的值
                        next3.movement = currentState.board[x][y -
2] + " R";
                        queue.enqueue(next3);
                    }
                    else{//代表就是右下角的值

                        next3.movement = currentState.board[x-1][y
- 2] + " R";
```

```
                        queue.enqueue(next3);

                    }
                }

            }



            //右边能不能换
            Node next4=right(currentState, x, y, i);


            //以下是判断移动的那个数字属于哪种block，属于block则需要打印该block中最小值及动作
            if (!close.contains(arrayToString(next4))){
                //判断属于哪种类型的block，如果是block，则需要打印左上角的数值和动作
                boolean flagofblock1 = find(currentState.block1, currentState.board[x][y+1]);
                boolean flagofblock2 = find(currentState.block2, currentState.board[x][y+1]);
                boolean flagofblock3 = find(currentState.block3, currentState.board[x][y+1]);


 if(flagofblock1==false&&flagofblock2==false&&flagofblock3==false){
                    //属于1*1，直接打印就行
                    next4.movement = currentState.board[x][y + 1] + " L";
                    queue.enqueue(next4);

                }

 if(flagofblock1==true&&flagofblock2==false&&flagofblock3==false){
                    //属于1*2，有解的情况下就是左上角的值，直接打印就行
                    next4.movement = currentState.board[x][y + 1] + " L";
                    queue.enqueue(next4);
```

```
                }


 if(flagofblock1==false&&flagofblock2==true&&flagofblock3==false){
                    //属于2*1
                    int j=rangeofblock2(currentState, x, y+1);
                    if(j==1){//代表就是左上角的值
                        next4.movement = currentState.board[x][y +
1] + " L";
                        queue.enqueue(next4);

                    }
                    else{//代表就是下面的值
                        next4.movement = currentState.board[x-1][y
+ 1] + " L";
                        queue.enqueue(next4);
                    }

                }


 if(flagofblock1==false&&flagofblock2==false&&flagofblock3==true){
                    //属于2*2
                    int j=rangeofblock3(currentState, x, y+1);
                    //有解的情况下j==1or3
                    if(j==1){//代表就是左上角的值
                        next4.movement = currentState.board[x][y +
1] + " L";
                        queue.enqueue(next4);
                    }
                    else{//代表就是左下角的值
                        next4.movement = currentState.board[x-1][y
+ 1] + " L";
                        queue.enqueue(next4);
                    }


                }


            }
        }
```

```
    }



    //将三种类型的字符串转为对应的标识
    public static int transform(String s){
        if(s.equals("1*2")){
            return 1;
        }
        else if(s.equals("2*1")){
            return 2;
        }
        else if(s.equals("2*2")){
            return 3;
        }
        else{
            return 0;//0代表不支持以外的block
        }

    }

    //获取各种block中左上角元素在数组中位置的方法
    public static int[] coordinateOfTopInBlock(int[][] array,int
number){
        //coordinate第一个元素代表所在行数，第二元素代表所在列数
        int[] coordinate= new int[2];
        for(int i=0;i<array.length;i++){
            for(int j=0;j<array[0].length;j++){
                if(array[i][j]==number){
                    coordinate[0]=i;
                    coordinate[1]=j;
                    break;
                }
            }
        }
        return coordinate;
    }

    //判断上面能不能换及结果
    public static Node up(Node state,int x,int y,int i){
```

```
        //state表示当前状态，[x,y]表示当前选定0的坐标,i代表选定的0在x0,y0中
的index
        if(x-1>=0&&state.board[x-1][y]!=0){
            Node next = new Node(state);

            //判断属于哪种类型的block
            //需要判断的点的坐标在array中的坐标是[x-1,y]
            boolean flag1=find(state.block1, state.board[x-1][y]);
            boolean flag2=find(state.block2, state.board[x-1][y]);
            boolean flag3=find(state.block3, state.board[x-1][y]);
            if(flag1==false&&flag2==false&&flag3==false){//属于1*1

                    //更新数值
                    next.board[x][y]=next.board[x-1][y];
                    next.board[x-1][y]=0;

                    next.x0[i]=x-1;
                    next.y0[i]=y;


                    return next;



            }else if(flag1==true&&flag2==false&&flag3==false){//属于
1*2

                    int j=rangeofblock1(state, x-1, y);
                    if(j==1){
                        //下处需不需要判定y+1是否超出边界
                        if(state.board[x][y+1]==0){
                            //更新数值
                            next.board[x][y]=next.board[x-1][y];
                            next.board[x][y+1]=next.board[x-1]
[y+1];

                            next.board[x-1][y]=0;
                            next.board[x-1][y+1]=0;

                            next.x0[i]=x-1;
                            next.y0[i]=y;
                            next.x0[indexOfZero(state, x, y+1)]=x-
1;
```

```
                                next.y0[indexOfZero(state, x,
y+1)]=y+1;



                                return next;
                            }
                            else{
                                return state;
                            }
                        }
                        else{
                            //下处需不需要判定y-1是否超出边界
                            if(state.board[x][y-1]==0){
                                //更新数值
                                next.board[x][y]=next.board[x-1][y];
                                next.board[x][y-1]=next.board[x-1][y-
1];
                                next.board[x-1][y]=0;
                                next.board[x-1][y-1]=0;

                                next.x0[i]=x-1;
                                next.y0[i]=y;
                                next.x0[indexOfZero(state, x, y-1)]=x-
1;
                                next.y0[indexOfZero(state, x, y-1)]=y-
1;



                                return next;
                            }
                            else{
                                return state;
                            }
                        }


                }
                else if(flag1==false&&flag2==true&&flag3==false){//属于
2*1

                        //更新数值
```

```
                    next.board[x][y]=next.board[x-1][y];
                    next.board[x-1][y]=next.board[x-2][y];
                    next.board[x-2][y]=0;

                    next.x0[i]=x-2;
                    next.y0[i]=y;



                    return next;

            }else{//属于2*2
                    int j=rangeofblock3(state, x-1, y);
                    if(j==1||j==2){
                        return state;
                    }
                    else if(j==3){
                        //下处需不需要判定y+1是否超出边界
                        if(state.board[x][y+1]==0){
                            //更新数值
                            next.board[x][y]=next.board[x-1][y];
                            next.board[x][y+1]=next.board[x-1]
[y+1];
                            //是否需要判断x-2超出边界
                            next.board[x-1][y]=next.board[x-2][y];
                            next.board[x-1][y+1]=next.board[x-2]
[y+1];
                            next.board[x-2][y]=0;
                            next.board[x-2][y+1]=0;

                            next.x0[i]=x-2;
                            next.y0[i]=y;
                            next.x0[indexOfZero(state, x, y+1)]=x-
2;
                            next.x0[indexOfZero(state, x, y+1)]=y;



                            return next;
                        }
                        else{
```

```
                            return state;
                        }
                    }
                    else{
                        //下处需不需要判定y-1是否超出边界
                        if(state.board[x][y-1]==0){
                            //更新数值
                            next.board[x][y]=next.board[x-1][y];
                            next.board[x][y-1]=next.board[x-1][y-
1];
                            //是否需要判断x-2超出边界
                            next.board[x-1][y]=next.board[x-2][y];
                            next.board[x-1][y-1]=next.board[x-2][y-
1];
                            next.board[x-2][y]=0;
                            next.board[x-2][y-1]=0;

                            next.x0[i]=x-2;
                            next.y0[i]=y;
                            next.x0[indexOfZero(state, x, y-1)]=x-
2;
                            next.y0[indexOfZero(state, x, y-1)]=y;


                            return next;
                        }
                        else{
                            return state;
                        }

                    }

                }

        }else{
            return state;
        }

    }

    //判断下面能不能换及结果
    public static Node down(Node state,int x,int y,int i){
```

```
        //state表示当前状态，[x,y]表示当前选定0的坐标,i代表选定的0在x0,y0中
的index
        if(x+1<=state.n-1&&state.board[x+1][y]!=0){
            Node next = new Node(state);

            //判断属于哪种类型的block
            //需要判断的点的坐标在array中的坐标是[x+1,y]
            boolean flag1=find(state.block1, state.board[x+1][y]);
            boolean flag2=find(state.block2, state.board[x+1][y]);
            boolean flag3=find(state.block3, state.board[x+1][y]);

            if(flag1==false&&flag2==false&&flag3==false){//属于1*1

                    //更新数值
                    next.board[x][y]=next.board[x+1][y];
                    next.board[x+1][y]=0;

                    next.x0[i]=x+1;
                    next.y0[i]=y;


                    return next;


            }else if(flag1==true&&flag2==false&&flag3==false){//属于
1*2



                    int j=rangeofblock1(state, x+1, y);
                    if(j==1){
                        //下处需不需要判定y+1是否超出边界
                        if(state.board[x][y+1]==0){
                            //更新数值
                            next.board[x][y]=next.board[x+1][y];
                            next.board[x][y+1]=next.board[x+1]
[y+1];

                            next.board[x+1][y]=0;
                            next.board[x+1][y+1]=0;

                            next.x0[i]=x+1;
                            next.y0[i]=y;
```

```
                            next.x0[indexOfZero(state, x,
y+1)]=x+1;
                            next.y0[indexOfZero(state, x,
y+1)]=y+1;



                            return next;
                        }
                        else{
                            return state;
                        }
                    }
                    else{
                        //下处需不需要判定y-1是否超出边界
                        if(state.board[x][y-1]==0){
                         //更新数值
                            next.board[x][y]=next.board[x+1][y];
                            next.board[x][y-1]=next.board[x+1][y-
1];
                            next.board[x+1][y]=0;
                            next.board[x+1][y-1]=0;

                            next.x0[i]=x+1;
                            next.y0[i]=y;
                            next.x0[indexOfZero(state, x, y-
1)]=x+1;
                            next.y0[indexOfZero(state, x, y-1)]=y-
1;



                            return next;
                        }
                        else{
                            return state;
                        }
                    }
```

```
            }else if(flag1==false&&flag2==true&&flag3==false){//属于
2*1



                    //更新数值
                    //是否需要判断x+2超出边界
                    next.board[x][y]=next.board[x+1][y];
                    next.board[x+1][y]=next.board[x+2][y];
                    next.board[x+2][y]=0;

                    next.x0[i]=x+1;
                    next.y0[i]=y;




                    return next;


            }else{//属于2*2


                    int j=rangeofblock3(state, x+1, y);
                    if(j==3||j==4){
                        return state;
                    }else if(j==1){
                        //下处需不需要判定y+1是否超出边界
                        if(state.board[x][y+1]==0){
                        //更新数值
                        next.board[x][y]=next.board[x+1][y];
                        next.board[x][y+1]=next.board[x+1][y+1];
                        //是否需要判断x+2超出边界
                        next.board[x+1][y]=next.board[x+2][y];
                        next.board[x+1][y+1]=next.board[x+2][y+1];
                        next.board[x+2][y]=0;
                        next.board[x+2][y+1]=0;



                        next.x0[i]=x+2;
                        next.y0[i]=y;
                        next.x0[indexOfZero(state, x, y+1)]=x+2;
                        next.y0[indexOfZero(state, x, y+1)]=y+1;
```

```
                    return next;
                }
                    else{
                    return state;
                }

                }else{
                //下处需不需要判定y-1是否超出边界
                if(state.board[x][y-1]==0){
                    //更新数值
                    next.board[x][y]=next.board[x+1][y];
                    next.board[x][y-1]=next.board[x+1][y-1];
                    //是否需要判断x+2超出边界
                    next.board[x+1][y]=next.board[x+2][y];
                    next.board[x+1][y-1]=next.board[x+2][y-1];
                    next.board[x+2][y]=0;
                    next.board[x+2][y-1]=0;

                    next.x0[i]=x+2;
                    next.y0[i]=y;
                    next.x0[indexOfZero(state,x,y-1)]=x+2;
                    next.y0[indexOfZero(state,x,y-1)]=y-1;


                    return next;
                }
                else{
                    return state;
                }
            }


        }

    }
    else{
        return state;
    }
```

```
    }

    //判断左面能不能换及结果
    public static Node left(Node state,int x,int y,int i){
        //state表示当前状态，[x,y]表示当前选定0的坐标
        if(y-1>=0&&state.board[x][y-1]!=0){
            Node next = new Node(state);

            //判断属于哪种类型的block
            //需要判断的点的坐标在array中的坐标是[x,y-1]
            boolean flag1=find(state.block1, state.board[x][y-1]);
            boolean flag2=find(state.block2, state.board[x][y-1]);
            boolean flag3=find(state.block3, state.board[x][y-1]);
            if(flag1==false&&flag2==false&&flag3==false){//属于1*1

                    //更新数值
                    next.board[x][y]=next.board[x][y-1];
                    next.board[x][y-1]=0;

                    next.x0[i]=x;
                    next.y0[i]=y-1;


                    return next;


            }else if(flag1==true&&flag2==false&&flag3==false){//属于
1*2

                    //更新数值
                    //是否需要判断y-2超出边界
                    next.board[x][y]=next.board[x][y-1];
                    next.board[x][y-1]=next.board[x][y-2];
                    next.board[x][y-2]=0;

                    next.x0[i]=x;
                    next.y0[i]=y-2;


                    return next;
```

```
            }else if(flag1==false&&flag2==true&&flag3==false){//属于
2*1

                    int j=rangeofblock2(state, x, y-1);
                    if(j==1){
                        //下处需不需要判定x+1是否超出边界
                        if(state.board[x+1][y]==0){
                            //更新数值
                            next.board[x][y]=next.board[x][y-1];
                            next.board[x+1][y]=next.board[x+1][y-
1];
                            next.board[x][y-1]=0;
                            next.board[x+1][y-1]=0;

                            next.x0[i]=x;
                            next.y0[i]=y-1;
                            next.x0[indexOfZero(state, x+1,
y)]=x+1;
                            next.y0[indexOfZero(state, x+1, y)]=y-
1;



                            return next;
                        }else{
                            return state;
                        }

                    }else{
                        //下处需不需要判定x-1是否超出边界
                        if(state.board[x-1][y]==0){
                            //更新数值
                            next.board[x][y]=next.board[x][y-1];
                            next.board[x-1][y]=next.board[x-1][y-
1];
                            next.board[x][y-1]=0;
                            next.board[x-1][y-1]=0;

                            next.x0[i]=x;
                            next.y0[i]=y-1;
                            next.x0[indexOfZero(state, x-1, y)]=x-
1;
```

```
                            next.y0[indexOfZero(state, x-1, y)]=y-
1;




                            return next;
                        }
                        else{
                            return state;
                        }
                    }


            }
            else{//属于2*2

                    int j=rangeofblock3(state, x, y-1);
                    if(j==1||j==3){
                        return state;
                    }else if(j==2){
                        //下处需不需要判定x+1是否超出边界
                        if(state.board[x+1][y]==0){
                            //更新数值
                            next.board[x][y]=next.board[x][y-1];
                            next.board[x+1][y]=next.board[x+1][y-
1];
                            //是否需要判断y-2超出边界
                            next.board[x][y-1]=next.board[x][y-2];
                            next.board[x+1][y-1]=next.board[x+1][y-
2];
                            next.board[x][y-2]=0;
                            next.board[x+1][y-2]=0;

                            next.x0[i]=x;
                            next.y0[i]=y-2;
                            next.x0[indexOfZero(state, x+1,
y)]=x+1;
                            next.y0[indexOfZero(state, x+1, y)]=y-
2;
```

```
                            return next;
                        }
                        else{
                            return state;
                        }
                    }else{
                        //下处需不需要判定x-1是否超出边界
                        if(state.board[x-1][y]==0){
                            //更新数值
                            next.board[x][y]=next.board[x][y-1];
                            next.board[x-1][y]=next.board[x-1][y-
1];
                            //是否需要判断x-2超出边界
                            next.board[x][y-1]=next.board[x][y-2];
                            next.board[x-1][y-1]=next.board[x-1][y-
2];
                            next.board[x-1][y-2]=0;
                            next.board[x][y-2]=0;

                            next.x0[i]=x;
                            next.y0[i]=y-2;
                            next.x0[indexOfZero(state, x-1, y)]=x-
1;
                            next.y0[indexOfZero(state, x-1, y)]=y-
2;

                            return next;
                        }
                        else{
                            return state;
                        }
                    }



            }


        }else{
            return state;
        }
```

```
    }

    //判断右面能不能换及结果
    public static Node right(Node state,int x,int y,int i){
        //state表示当前状态，[x,y]表示当前选定0的坐标
        if(y+1<=state.m-1){
            Node next = new Node(state);

        //判断属于哪种类型的block
        //需要判断的点的坐标在array中的坐标是[x,y+1]
        boolean flag1=find(state.block1, state.board[x][y+1]);
        boolean flag2=find(state.block2, state.board[x][y+1]);
        boolean flag3=find(state.block3, state.board[x][y+1]);

        if(flag1==false&&flag2==false&&flag3==false){//属于1*1

                //更新数值
             next.board[x][y]=next.board[x][y+1];
             next.board[x][y+1]=0;

             next.x0[i]=x;
             next.y0[i]=y+1;


             return next;


        }else if(flag1==true&&flag2==false&&flag3==false){//属于1*2

                //更新数值
                //是否需要判断y+2超出边界
                next.board[x][y]=next.board[x][y+1];
                next.board[x][y+1]=next.board[x][y+2];
                next.board[x][y+2]=0;

                next.x0[i]=x;
                next.y0[i]=y+2;



                return next;
```

```
        }else if(flag1==false&&flag2==true&&flag3==false){//属于2*1

                int j=rangeofblock2(state, x, y+1);
                if(j==1){
                    //下处需不需要判定x+1是否超出边界
                    if(state.board[x+1][y]==0){
                        //更新数值
                        next.board[x][y]=next.board[x][y+1];
                        next.board[x+1][y]=next.board[x+1][y+1];
                        next.board[x][y+1]=0;
                        next.board[x+1][y+1]=0;

                        next.x0[i]=x;
                        next.y0[i]=y+1;
                        next.x0[indexOfZero(state, x+1, y)]=x+1;
                        next.y0[indexOfZero(state, x+1, y)]=y+1;



                        return next;
                    }else{
                        return state;
                    }

                }else{
                    //下处需不需要判定x-1是否超出边界
                    if(state.board[x-1][y]==0){
                        //更新数值
                        next.board[x][y]=next.board[x][y+1];
                        next.board[x-1][y]=next.board[x-1][y+1];
                        next.board[x][y+1]=0;
                        next.board[x-1][y+1]=0;

                        next.x0[i]=x;
                        next.y0[i]=y+1;
                        next.x0[indexOfZero(state, x-1, y)]=x-1;
                        next.y0[indexOfZero(state, x-1, y)]=y+1;



                        return next;
```

```
                    }
                    else{
                        return state;
                    }



        }
    }else{//属于2*2

            int j=rangeofblock3(state, x, y+1);
            if(j==2||j==4){
                return state;
            }else if(j==1){
                //下处需不需要判定x+1是否超出边界
                if(state.board[x+1][y]==0){
                    //更新数值
                    next.board[x][y]=next.board[x][y+1];
                    next.board[x+1][y]=next.board[x+1][y+1];
                    //是否需要判断y-2超出边界
                    next.board[x][y+1]=next.board[x][y+2];
                    next.board[x+1][y+1]=next.board[x+1][y+2];
                    next.board[x][y+2]=0;
                    next.board[x+1][y+2]=0;

                    next.x0[i]=x;
                    next.y0[i]=y+2;
                    next.x0[indexOfZero(state, x+1, y)]=x+1;
                    next.y0[indexOfZero(state, x+1, y)]=y+2;




                    return next;
                }
                else{
                    return state;
                }

            }else{
                //下处需不需要判定x-1是否超出边界
                if(state.board[x-1][y]==0){
```

```
                    //更新数值
                    next.board[x][y]=next.board[x][y+1];
                    next.board[x-1][y]=next.board[x-1][y+1];
                    //是否需要判断y+2超出边界
                    next.board[x][y+1]=next.board[x][y+2];
                    next.board[x-1][y+1]=next.board[x-1][y+2];
                    next.board[x][y+2]=0;
                    next.board[x-1][y+2]=0;

                    next.x0[i]=x;
                    next.y0[i]=y+2;
                    next.x0[indexOfZero(state, x-1, y)]=x-1;
                    next.y0[indexOfZero(state, x-1, y)]=y+2;



                    return next;
                }
                else{
                    return state;
                }


            }

        }

        }
        else{
            return state;
        }

    }

    //判断一个数是否属于二维数组中
    public static boolean find(int[][]array,int target){
        boolean flag = false;
        if(array!=null&&array.length!=0&&array[0].length!=0){
            for (int i = 0; i < array.length; i++) {
                for (int j = 0; j <array[i].length ; j++) {
                    if (target == array[i][j]) { //判断是否存在该整数
                        flag = true;  //存在
```

```
                        return flag;
                    }
                }
            }

        }
        return flag;
    }

    //判断属于1*2block中的第一个元素还是第二个元素
    //如果是第一个，就返回1，如果是第二个，就返回2
    public static int rangeofblock1(Node state,int x,int y){
        //state为目前需要判断的状态，[x,y]为需要判断的点在state.array中的坐
标
        int column=0;
        for(int i=0;i<state.block1.length;i++){
            for(int j=0;j<state.block1[0].length;j++){
                if(state.block1[i][j]==state.board[x][y]){
                    column=j;
                }
            }
        }
        if(column==0){
            return 1;
        }else{
            return 2;
        }
    }

    //判断属于2*1 block中的第一个元素还是第二个元素
    //如果是第一个，就返回1，如果是第二个，就返回2
    public static int rangeofblock2(Node state,int x,int y){
        //state为目前需要判断的状态，[x,y]为需要判断的点在state.array中的坐
标
        int column=0;
        for(int i=0;i<state.block2.length;i++){
            for(int j=0;j<state.block2[0].length;j++){
                if(state.block2[i][j]==state.board[x][y]){
                    column=j;
                }
            }
        }
```

```
        if(column==0){
            return 1;
        }else{
            return 2;
        }
    }

    //判断属于2*2 block中的第几个元素
    //如果是第一个，就返回1，如果是第二个，就返回2，以此类推
    public static int rangeofblock3(Node state,int x,int y){
        int column=0;
        for(int i=0;i<state.block3.length;i++){
            for(int j=0;j<state.block3[0].length;j++){
                if(state.block3[i][j]==state.board[x][y]){
                    column=j;
                }
            }
        }
        if(column==0){
            return 1;
        }else if(column==1){
            return 2;
        }else if(column==2){
            return 3;
        }else{
            return 4;
        }
    }

    //根据选定的0的坐标判断这个0属于neoCount中第几个0
    public static int indexOfZero(Node state,int x,int y){
        //[x,y]为该0在state.board中的坐标
        int index=0;

        for(int i=0;i<state.x0.length;i++){
            if(state.x0[i]==x&&state.y0[i]==y){
                index=i;
            }
        }

        return index;
```

```
    }


}
```

```
package Breadth_First;

import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.Stack;

import edu.princeton.cs.algs4.Queue;

import static Breadth_First.Node.arrayToString;
import static Breadth_First.Node.printState;
import static Breadth_First.Node.transform;



public class Run {
    public static void main(String[] args) throws
FileNotFoundException {
        long start = System.currentTimeMillis();
        //standard input
        try (Scanner input = new Scanner(System.in)) {
            int N = input.nextInt();
            int M = input.nextInt();
            int[][] initArray = new int[N][M];
            for(int i=0;i<N;i++){
                for(int j=0;j<M;j++){
                    initArray[i][j]=input.nextInt();
                }
            }
            int numberOfblock = input.nextInt();
            int[][] allOfBlock=new int[numberOfblock][2];
            if(numberOfblock!=0){
                for(int i=0;i<numberOfblock;i++){
                    allOfBlock[i][0]=input.nextInt();
```

```
                    //因为表示block种类的输入是字符类型，将各种种类转换为整数来分类，
                        allOfBlock[i][1]=transform(input.next());
                    }
                }else{
                    allOfBlock=null;
                }

                ArrayList<String> close = new ArrayList<String>();//判断是否在close中，表示已经搜索过的状态
                Queue<Node> queue = new Queue<Node>();//要搜索的状态



                //初始状态
                Node initState = new Node(initArray, null,numberOfblock,allOfBlock);
                //将初始状态放入
                queue.enqueue(initState);

                boolean flag = false;
                Node ans = null;

                while (!queue.isEmpty()){
                    Node currentState = queue.dequeue();
                    if(currentState.isComplete()){
                        flag = true;
                        ans = currentState;
                        break;
                    }
                    currentState.findNext(queue,close);
                }

                Stack<String> move = new Stack<String>();
                Stack<String> boards = new Stack<String>();
                int moveCount = 0;

                if(flag){
                    Node t = ans;
```

```
                boards.push(printState(t));
                move.push(t.movement);
                t = t.father;

                while(t != null){
                    boards.push(printState(t));
                    if(t.movement != null){
                        move.push(t.movement);
                    }
                    moveCount++;
                    t = t.father;
                }

                System.out.println("Yes");
                System.out.println(moveCount);

                while (!move.isEmpty()){
                    System.out.println(move.pop());

                }
            }
            else{
                System.out.print("No\n");
            }
            System.out.print((System.currentTimeMillis() - start) +
" ms");
        }


    }
}
```