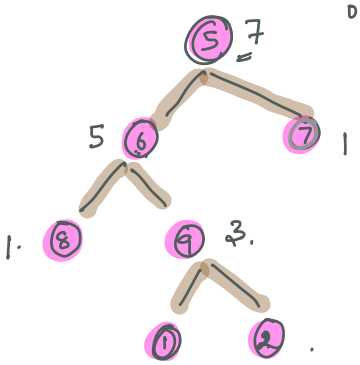
Today's agenda.

9:05.

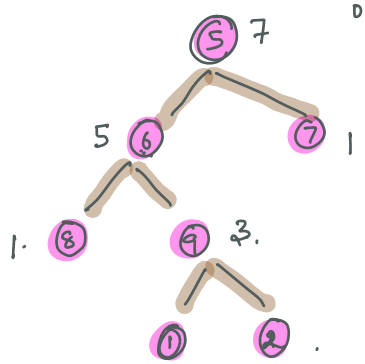
i) Count nodes in a binary tree → Tree.
ii) flatten a nested list
iii) Sudoku validator
iv) Matrix operations

(i) Count nodes in a binary tree.

```
int countNodes(Node node)
{
    if (node == null)
        return 0;
    return 1 + countNodes(node.left)
             + countNodes(node.right);
}
```



o/p = 7.



o/p = 7.

```
class Node {
    int data;
    Node left;
    Node right;
}
```

f(5) = 7
1 + f(6) + f(7).

f(6)
1 + f(8) + f(9)
1 3

f(8) = 1.
1 + f(null) + f(null)
1 + 0 + 0 = 1

f(9) = 3.
1 + f(1) + f(2).
1 + 1 + 1

[4,5,6]. → [7,8]

2) flatten a nested list.

List<NestedInteger>. → i/p.
[1, 2, 3, 4]. → o/p.

[1], [1,2,3], [[1,2],[3,4]].

```
class NestedInteger
{
    boolean isInteger();
    int getInteger();
    ArrayList<NestedInteger> getList();
}
```

i/p. [1, [3,4], 5, [6,7]]

```
class NestedIterator
{
    ArrayList<Integer> flattenedList = new ArrayList<>();
    int currentReadIndex = 0;

    NestedIterator(ArrayList<NestedInteger> nestedList)
    {
        populateNestedList(nestedList);
    }

    int next() {
        return flattenedList.get(currentReadIndex++);
    }

    boolean hasNext() {
        return flattenedList.size() > currentReadIndex;
    }

    void populateNestedList(ArrayList<NestedInteger> nestedList)
    {
        for(NestedInteger nestedValue : nestedList)
        {
            if(nestedValue.isInteger())
            {
                flattendList.add(nestedValue.getInteger());
            }
            else
            {
                populateNestedList(nestedValue.getList());
            }
        }
    }
}
```

a = i++;

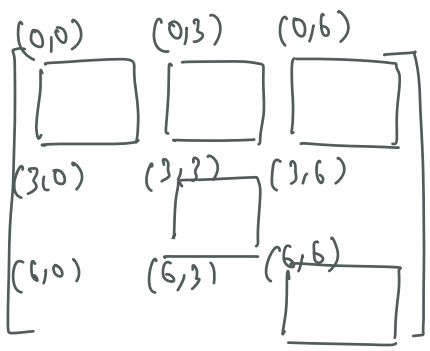
a = i;
i = i+1;

iii) Sudoku validator.

(i) for row no's [1-9].
(ii) for columns no's [1-9]
(iii) for boxes no's [1-9].

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
bookan isValidSudoku(char[][] matrix)
{
    // valid rows
    for(int i=0; i<9; i++)
    {
        if(!isValidRow(i, matrix))
            return false;
    }
    // valid columns
    for(int i=0; i<9; i++)
    {
        if(!isValidColumn(i, matrix))
            return false;
    }
    // valid boxes
    for(int i=0; i<3; i++) {
        for(int j=0; j<3; j++)
            if(!validBox(3i, 3j, matrix))
                return false;
```



0, 1, 2.

0, 1, 2.

(0, 0), (0, 3), (0, 6)

(3,0) (3,3) (3,6)
(6,0) (6,3) (6,6)

```
    }
}
```

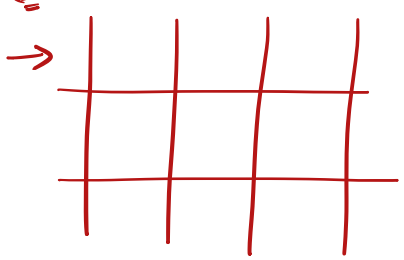
```
boolean isValidRow(int row, char[][] matrix)
{
    int[] hCount = new int[9];
    for(int i=0; i<9; i++)
    {
        if(matrix[row][i]!='.'){
            if(hCount[matrix[row][i]-'1']!=0){
                return false;
            }
            hCount[matrix[row][i]-'1']=1;
        }
    }
    return true;
}
```

[6 1 9 5]. 6].

[0 0 0 0 0 0 0 0 0]
0 1 2 3 4 5 6 7 8

```
boolean isValidBox(int sR, int sC, char[][] matrix)
{
    int[] hCount = new int[9];
    for(int i=sR; i<sR+3; i++)
    {
        for(int j=sC; j<sC+3; j++)
        {
            if(matrix[i][j]!='.'){
                if(hCount[matrix[i][j]-'1']!=0){
                    return false;
                }
                hCount[matrix[i][j]-'1']=1;
            }
```

(3,0)



(3,0) to (3,2)
(4,0) to (4,2)
(5,0) to (5,2)

```
    }
}
return true;
}
```

4) Matrix Operations.

r0 → 0
r1 → 1

11:05 , breaks.

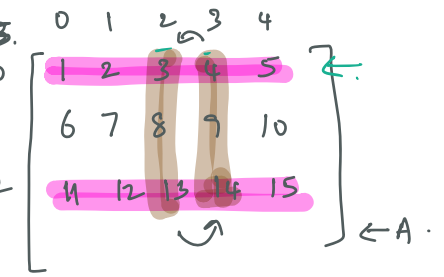
Q queries.

1. C1 C2. -x
2. R1 R2 ✓ x
3. x1, y1, x2, y2 ✓
4. x1, y1, x2, y2. ✓

C1 → 1
C2 → 2.

N, M.
N = 3, M = 5.

[2 3].
[0, 2].
(0,0) (1,2).



A[x1][y1] 11. A[x2][y2] ← ③.
A[x1][y1] & A[x2][y2] ← ④

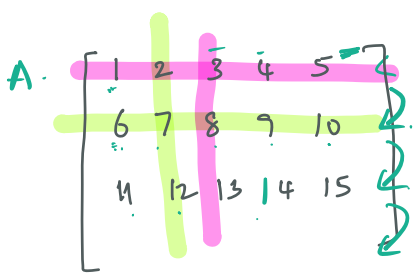
Soln:

SC: O(N×M).
TC: O(N×M) + Q(max(N,M))

} Brute force.

NxM:

A(0)(2), A(1)(1).
3 7.



A[row][col] = 1 + col + row * m.

A[0][4] = 1 + 4 + 0 = 5.

A[0][2] = 1 + 2 + 0 * 5 = 3.

A[1][1] = 1 + 1 + 1 * 5 = 7.

A[2][3] = 1 + col + row * m.
A[1][3] =

row → 1
row → 2.
A[0][2] = _____

before swapping.

A[2][2] = _____

A[2][1] = 1+col+row*m.
= 1.

swap R1 & R0.

0. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.

R0, R1 $\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}$.

```
public static void main(String[] args)
{
        Scanner sc = new Scanner(System.in);
        long n = sc.nextInt(), m = sc.nextInt(), q = sc.nextInt();
        long[] r = new long[100005];
        long[] c = new long[100005];
        // initialize rows.
        for(int i=0; i<n; i++)
                r[i] = i;
        // initialize cols.
        for(int i=0; i<m; i++)
                c[i] = i;

        // Q operations.
        for(int i=0; i<q; i++)
        {
                int t = sc.nextInt();
                if(t==1)
                {
                        int c1 = sc.nextInt();
                        int c2 = sc.nextInt();
                        long temp = c[c1-1];
                        c[c1-1] = c[c2-1];
                        c[c2-1] = temp;
                }
                if(t==2)
                {
                        int R1 = sc.nextInt();
```

c[2] → 2
c[10] → 10.

c[10] => 2
c[2] -> 10.

swap. (a,b)

int temp = a
a = b.
b = temp.

```
            int R2= sc.nextInt();
            long temp= R[R1-1];
            R[R1-1] = R[R2-1];
            R[R2-1] = temp;
        }
        if(t==3)
        {
            // read x1, y1, x2, y2.
            long a= 1+ C[y1-1]+ r[x1-1]*m;
            long b= 1+ C[y2-1]+ r[x2-1]*m;
            print(a||b);
        }
        if(t==4)
        {
            // read x1, y1, x2, y2.
            long a= 1+ C[y1-1]+ r[x1-1]*m;
            long b= 1+ C[y2-1]+ r[x2-1]*m;
            print(a&b);
        }
    }
}
```

1+C[1]+row*m.