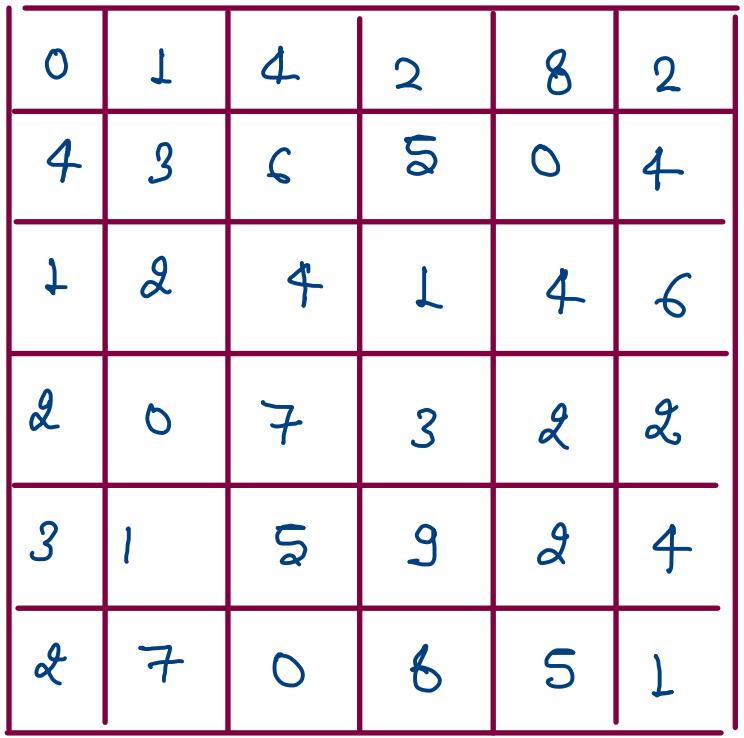
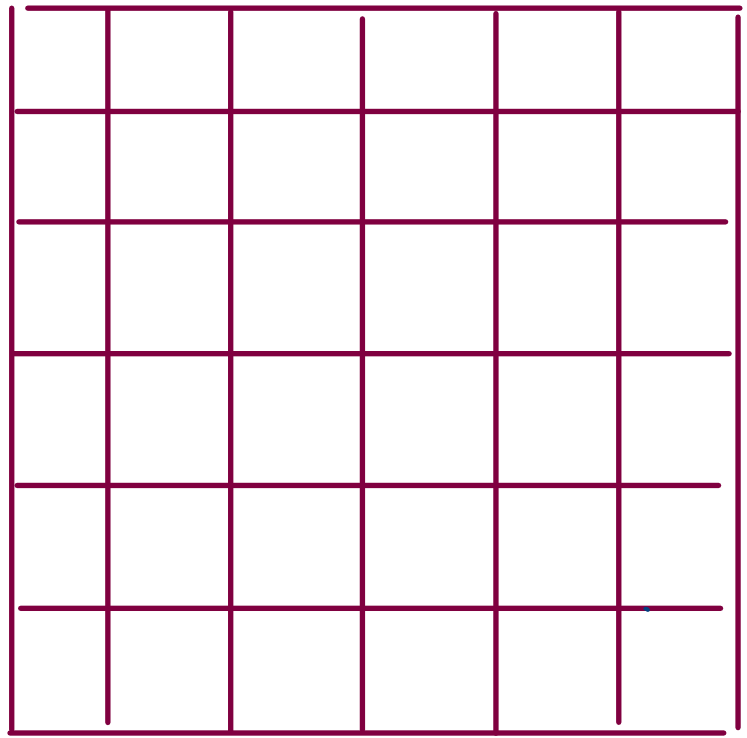
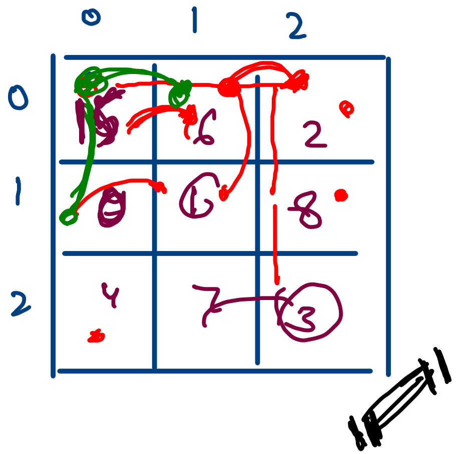
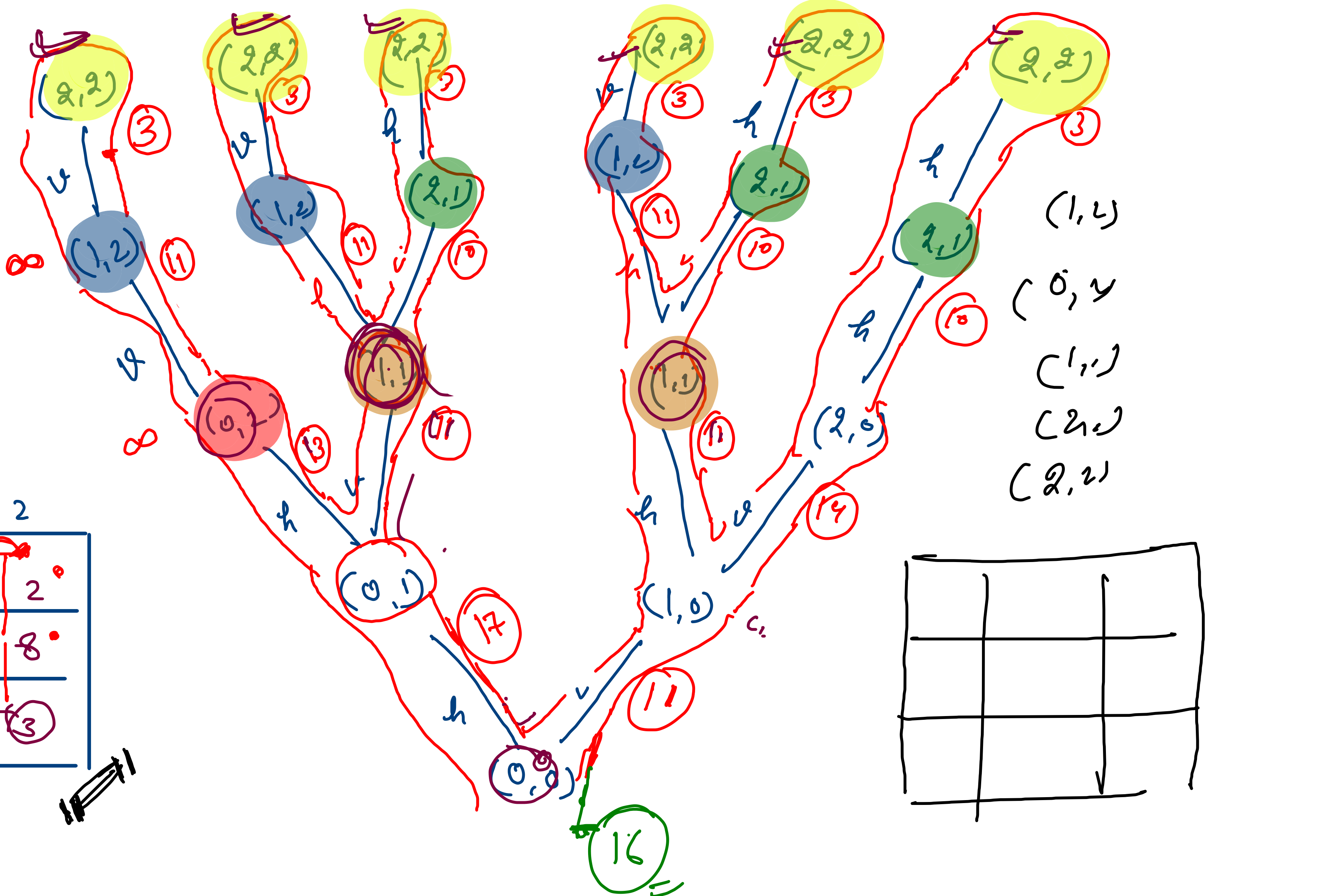

0 1 4 2 8 2
4 3 6 5 0 4
1 2 4 1 4 6
2 0 7 3 2 2
3 1 5 9 2 4
2 7 0 6 5 1



dr = 2
dc = 2





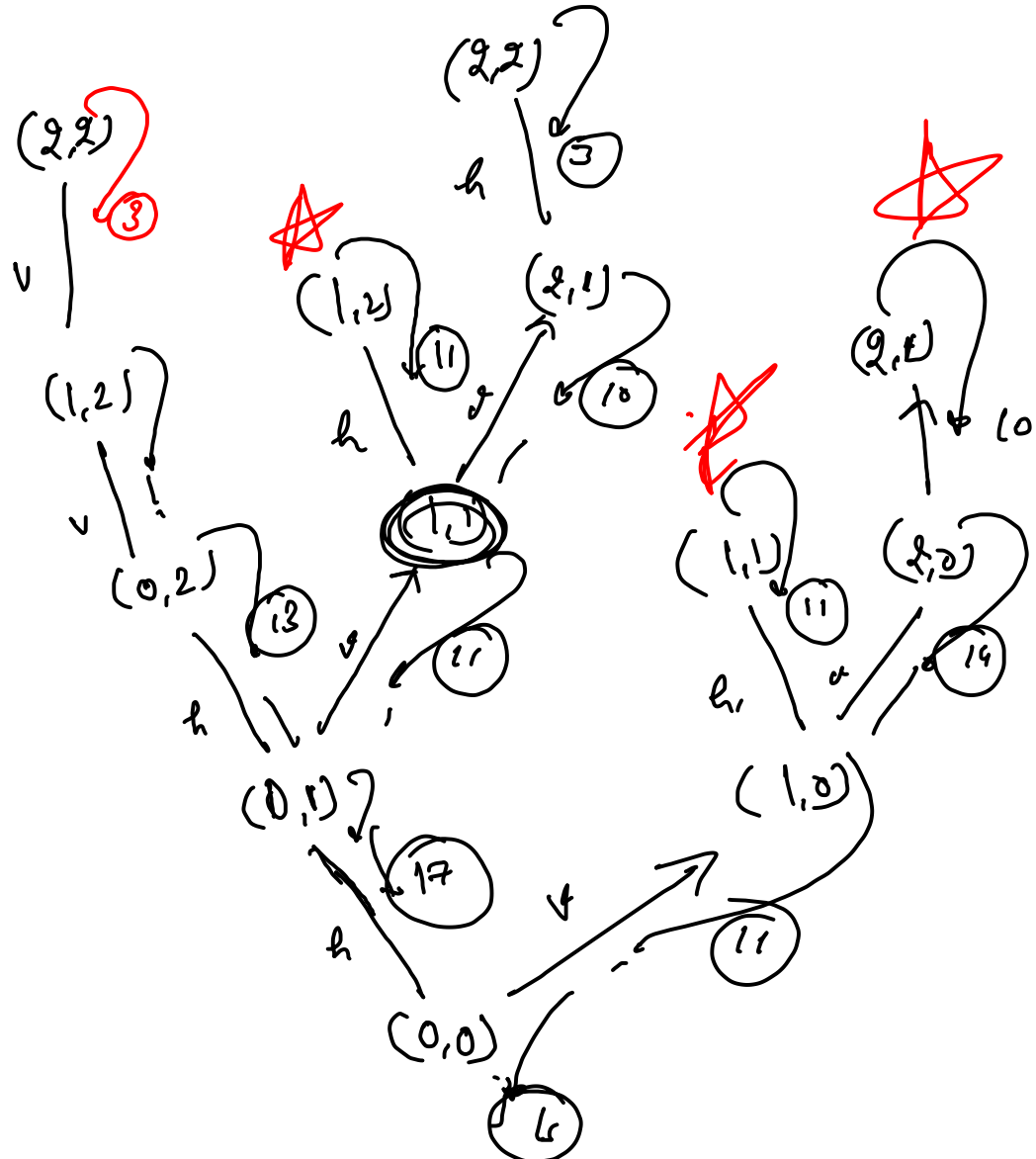
Maze

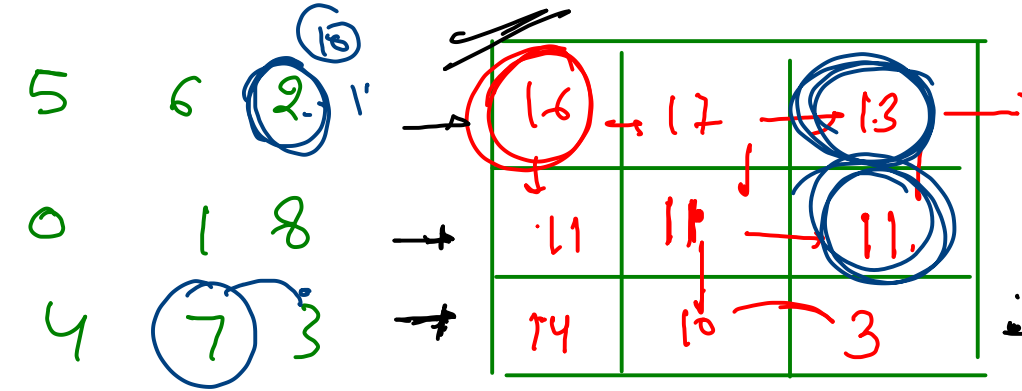
	0	1	2
0	5	6	2
1	0	1	8
2	4	7	3

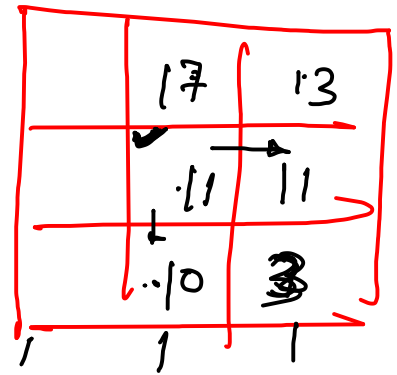
dp

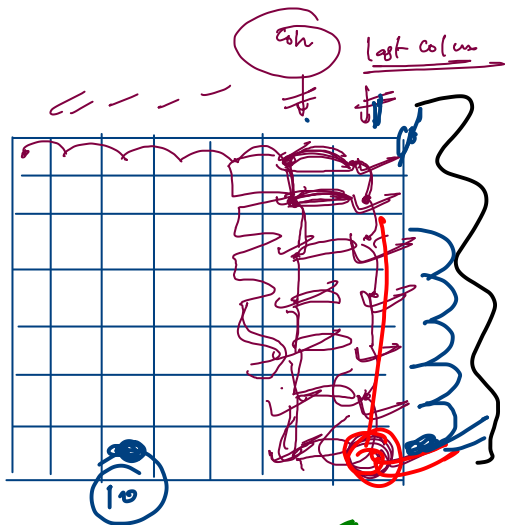
	0	1	2
0	16	17	13
1	11	11	11
2	14	10	3

```java
public static int minCostPath_memo(int[][] maze, int sr, int sc, int[][] dp) {
    if(sr == maze.length - 1 && sc == maze[0].length - 1) {
        return maze[sr][sc];
    }

    if(dp[sr][sc] != 0) {
        return dp[sr][sc];
    }

    int minCost = Integer.MAX_VALUE;
    // horizontal
    if(sc + 1 < maze[0].length) {
        minCost = Math.min(minCost, minCostPath_memo(maze, sr, sc + 1, dp));
    }

    // vertical
    if(sr + 1 < maze.length) {
        minCost = Math.min(minCost, minCostPath_memo(maze, sr + 1, sc, dp));
    }
    dp[sr][sc] = maze[sr][sc] + minCost;
    return dp[sr][sc];
}
```

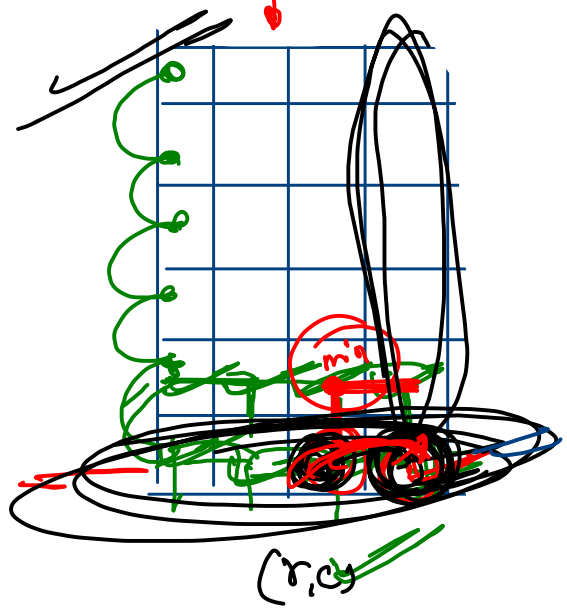




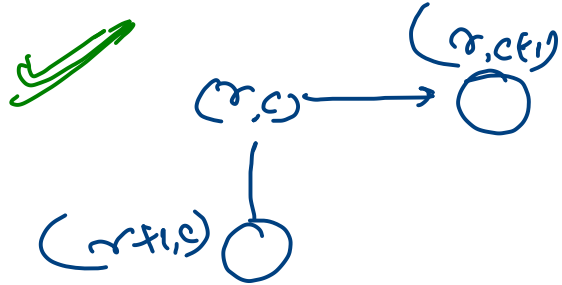





```
fun() {
call horizontal
call vertical
}
```



```
fun() {
call vertical()
call horizento()
}
```

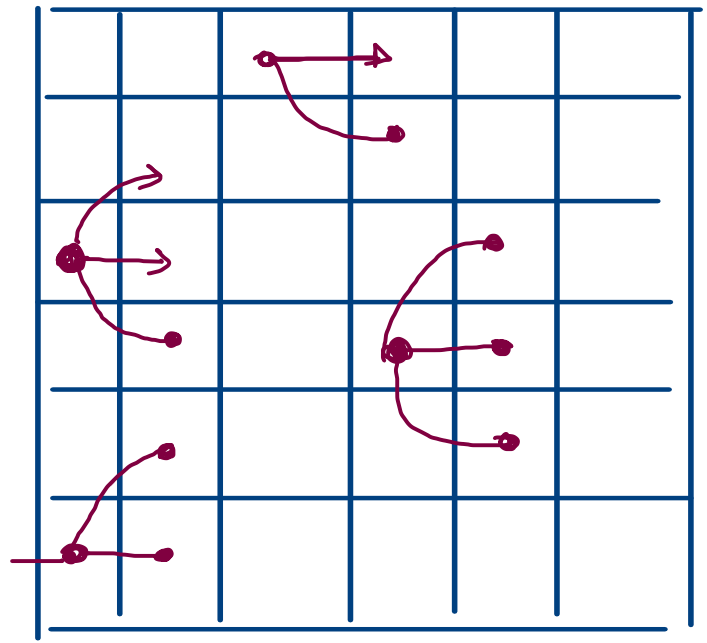


$$dp[r][c] = \max\{r][c] + mat[r][c+1]$$

Goldmine.

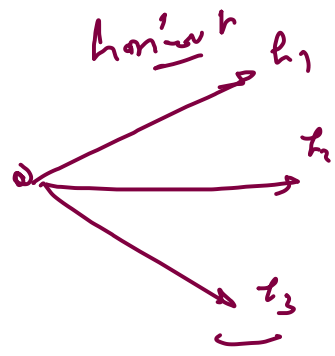
	0	1	2	3	4	5
0	0	1	4	2	8	2
1	4	3	6	5	0	4
2	1	2	4	1	4	6
3	2	0	7	3	2	2
4	3	1	5	9	2	4
5	2	7	0	8	5	1

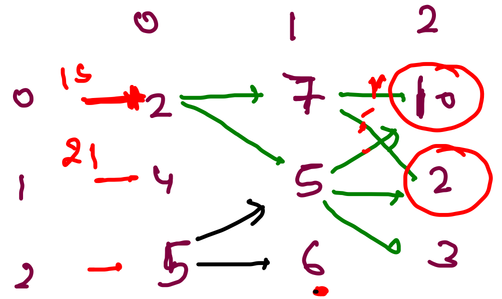
```
014282
436504
124146
207322
315924
270851
```

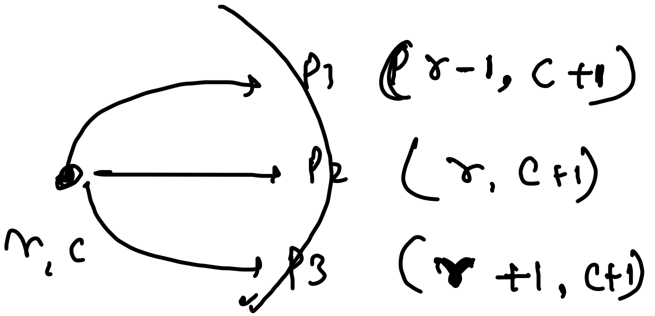


Direction of Movement

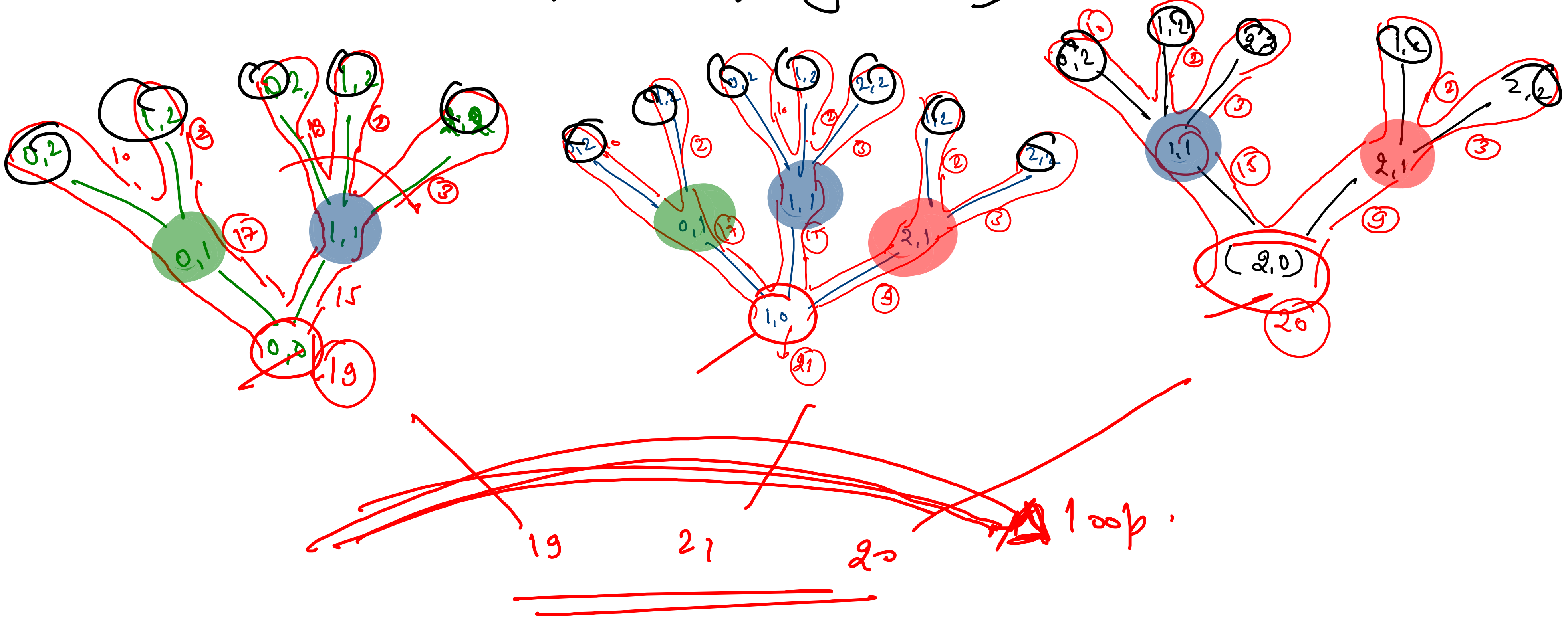
→ Horizontal toward Right

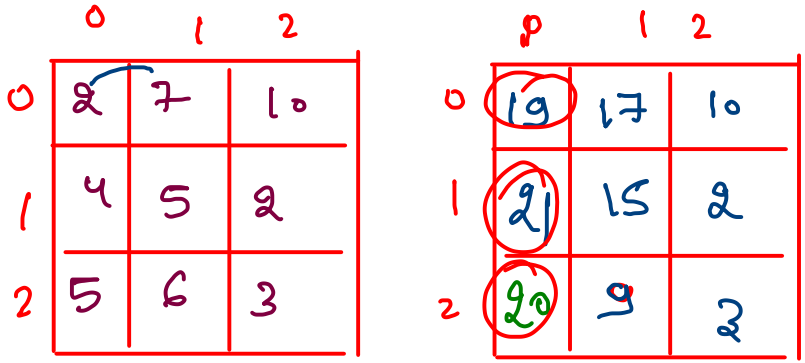


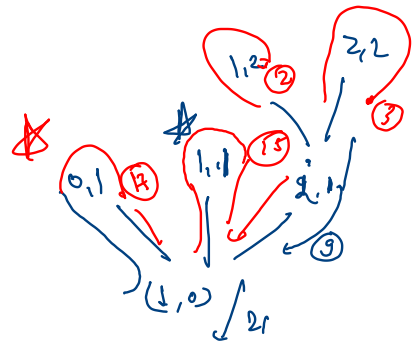


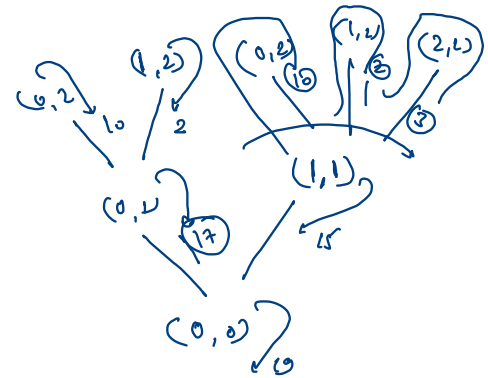


c == matrix[0].length-1] base case

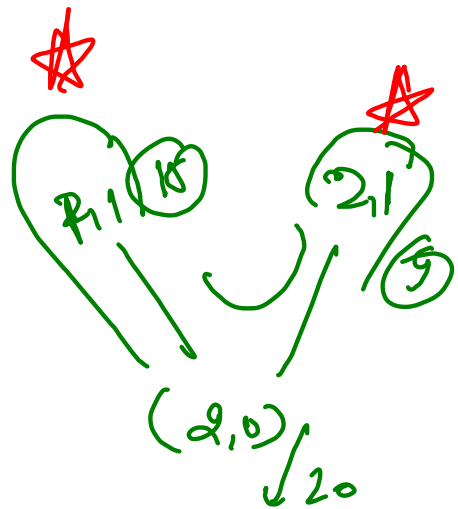


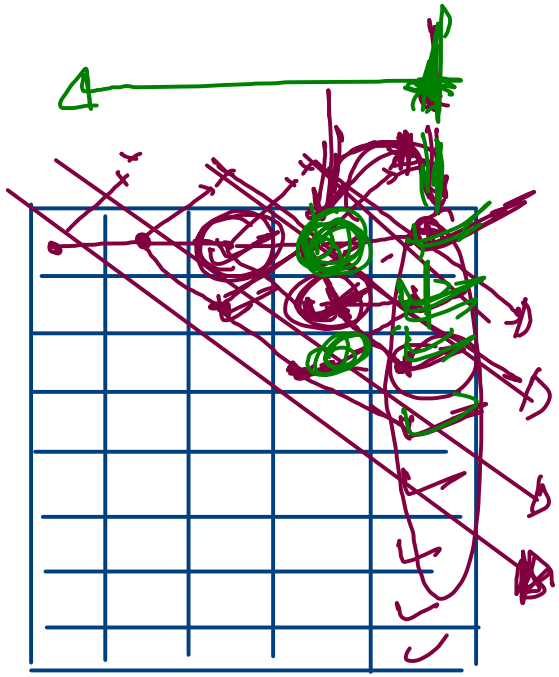


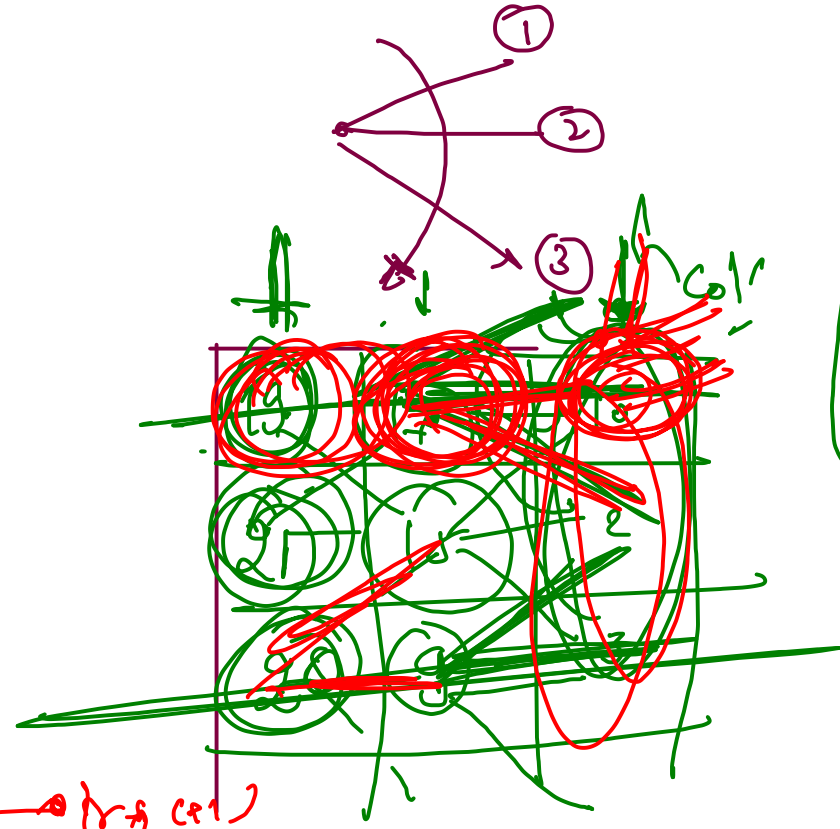


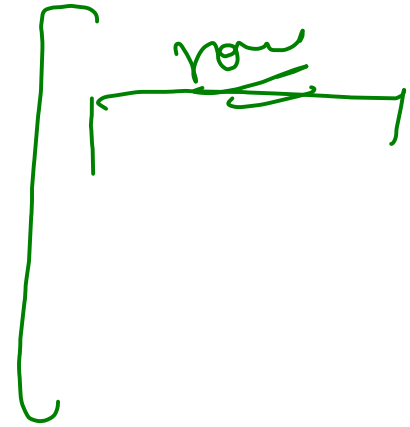



```java
public static int goldmine_memo(int[][] mine, int r, int c, int[][] dp) {
    if(c == mine[0].length - 1) {
        return dp[r][c] = mine[r][c];
    }

    if(dp[r][c] != 0) return dp[r][c];

    int maxGold = 0;
    for(int d = 0; d <= 2; d++) {
        int rr = r + rdir[d];
        int cc = c + cdir[d];
        if(rr >= 0 && rr < mine.length) {
            maxGold = Math.max(maxGold, goldmine_memo(mine, rr, cc, dp));
        }
    }
    return dp[r][c] = maxGold + mine[r][c];
}

public static int goldmine_memoHelp(int[][] mine) {

    int[][] dp = new int[mine.length][mine[0].length];

    int maxGold = 0;
    for(int r = 0; r < mine.length; r++) {
        maxGold = Math.max(maxGold, goldmine_memo(mine, r, 0, dp));
    }
    return maxGold;
}
```









2 7 10

4 5 2

5 6 3