Paint House

There is a row of n houses, where each house can be painted one of three colors: red, blue, or green. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.
The cost of painting each house with a certain color is represented by an n x 3 cost matrix costs.

- For example, costs[0][0] is the cost of painting house 0 with the color red; costs[1][2] is the cost of painting house 1 with color green, and so on...

Return *the minimum cost to paint all houses*.

Example 1:
Input: costs = [[17,2,17],[16,16,5],[14,3,19]]
Output: 10
Explanation: Paint house 0 into blue, paint house 1 into green, paint house 2 into blue.
Minimum cost: 2 + 5 + 3 = 10.
Example 2:
Input: costs = [[7,6,2]]
Output: 2

Constraints:

- costs.length == n
- costs[i].length == 3
- 1 <= n <= 100
- 1 <= costs[i][j] <= 20

Code
// I used greedy first. Then realized that it does not work with the constraint. It took me a while to realize the simple approach. This problem is same as getting the shortest path from start to goal in a matrix.

// 0ms, eliminate 1 declaration with for loop is faster than using while loop

```
class Solution {
    public int minCost(int[][] costs) {
        //int i = 0;
```

```
        //while (++i < costs.length){

        for (int i = 1; i < costs.length; i++){
            costs[i][0] += Math.min(costs[i-1][1], costs[i-1][2]);
            costs[i][1] += Math.min(costs[i-1][0], costs[i-1][2]);
            costs[i][2] += Math.min(costs[i-1][0], costs[i-1][1]);
        }
       // return Math.min(Math.min(costs[i-1][0], costs[i-1][1]), costs[i-1][2]);
       return Math.min(Math.min(costs[costs.length-1][0],
costs[costs.length-1][1]), costs[costs.length-1][2]);
    }
}
```