1.

Each row has 12 possible states: RGB, RBG, BGR, BRG, GBR, GRB, RGR, RBR, GRG, GBG, BRB, BGB.

Let OPT(n,i) be the maximum number of colorings for an n x 3 matrix whose top row is in state i, for i in 1...12.

The base case is OPT(1,i)=1 for all i.

The update rule: for each state of the top row, it is compatible with only certain states of the row below, which we can precompute. e.g. RGR is compatible with BRB but not BGB. Then let OPT(n, i) = \sum_{j=1}^12 OPT(n-1, j) * [indicator that is 1 if state is compatible with state j]

Final output: sum of OPT(N,j) for all j from 1 to 12. (summing up over all possible states of the top row)

Runtime: O(N)

```
Input: height of the matrix N
Output: number of ways to color the matrix

states=[\doBlank{rgb, rbg, bgr, brg, gbr, grb, rgr, rbr, grg, gbg,
brb, bgb}] (5 pts)
# initialize a 2D OPT matrix of size N*|states| with all entries to
be zero
OPT[N, |states|] = 0
# base case
For s = 1 to |states|
    OPT[\doBlank{1, s}] = \doBlank{1} (2 pts)
EndFor
# dp update
For i = 2 to N
    For s1 = 1 to |states|
        For s2 = 1 to |states|
            state1 = states[s1]
            state2 = states[s2]
            If \doBlank{state1[1]!=state2[1] & state1[2]!=states2[2]
& state1[3]!=state2[3]} (5 pts)
                OPT[\doBlank{i, s1}]=\doBlank{OPT[i, s1] + OPT[i-1,
s2]} (5 pts)
            EndIF
```

```
        EndFor
    EndFor
EndFor
Return \doBlank{sum(OPT[n, :])} (3 pts)
```

Note that |states| represents the length of the array states

==========

2.

Given string $a = a_1 a_2 \cdots a_n$ without spaces, we want to add spaces to break it into words $w_1 \cdots w_k$ maximizing $\sum_i \text{score}(w_i)$.

Sol $O(n^3)$ solution that should get full credit, though not optimal!

Let $OPT(i,j)$ = maximum score for substring $a_i \cdots a_j$.

Notice $$OPT(i,j) = \max \begin{cases} \text{score}(a_i \cdots a_j) \\ \max_{t=i, i+1, \ldots, j} \left[OPT(i,t) + OPT(t+1, j) \right] \end{cases}$$

with a base case of $OPT(i,i) = \text{score}(a_i)$ $\forall i = 1, \ldots, n$. (single letters)

The DP matrix is $n \times n$, though we only use the half where $j \geq i$.

Each entry takes $O(n)$ time to compute for all values for t.

We fill in the DP matrix by increasing order of $j - i$.

Sol 2: $O(n^2)$ solution

Let $OPT(i)$ = maximum score for substring $a_1 \cdots a_i$.

Notice $$OPT(i) = \max \begin{cases} \text{score}(a_1 \cdots a_i) \\ \max_{t=1 \ldots i} \left[OPT(t) + \text{score}(a_{t+1} \cdots a_i) \right] \end{cases}$$

with a base case of $OPT(1) = \text{score}(a_1)$.

The DP matrix is 1D and size n, and each entry takes $O(n)$ time to compute all values for t.

==========
3.

Initialize $Q(i, j, k) = 0$ if $i = 0$ or $j = 0$ or $k = 0$. This is because if at least one string is empty, there cannot be a common subsequence. The recurrence relation for dynamic programming is: \\
$Q(i, j, k) = Q(i − 1, j − 1, k − 1) + 1 (if a_i = b_j = c_k)$
$Q(i, j, k) = max{Q(i − 1, j, k), Q(i, j − 1, k), Q(i, j, k − 1)}
(if a_i \neq b_j or b_j \neq c_k or c_k \neq a_i)$
Runtime for this algorithm is O(N^3).