4 ways | Step by step from Recursion -> top down DP -> bottom up DP -> fine tuning

[avval](https://leetcode.com/avval/)

2563

90566

Jan 11, 2020

Java

Dynamic Programming

We start at either step 0 or step 1. The target is to reach either last or second last step, whichever is minimum.

**Step 1 - Identify a recurrence relation between subproblems.** In this problem,  
Recurrence Relation:  
mincost(i) = cost[i]+min(mincost(i-1), mincost(i-2))  
Base cases:  
mincost(0) = cost[0]  
mincost(1) = cost[1]

**Step 2 - Convert the recurrence relation to recursion**

// Recursive Top Down - O(2^n) Time Limit Exceeded

public int minCostClimbingStairs(int[] cost) {

int n = cost.length;

return Math.min(minCost(cost, n-1), minCost(cost, n-2));

}

private int minCost(int[] cost, int n) {

if (n < 0) return 0;

if (n==0 || n==1) return cost[n];

return cost[n] + Math.min(minCost(cost, n-1), minCost(cost, n-2));

}

**Step 3 - Optimization 1 - Top Down DP - Add memoization to recursion** - From exponential to linear.

// Top Down Memoization - O(n) 1ms

int[] dp;

public int minCostClimbingStairs(int[] cost) {

int n = cost.length;

dp = new int[n];

return Math.min(minCost(cost, n-1), minCost(cost, n-2));

}

private int minCost(int[] cost, int n) {

if (n < 0) return 0;

if (n==0 || n==1) return cost[n];

if (dp[n] != 0) return dp[n];

dp[n] = cost[n] + Math.min(minCost(cost, n-1), minCost(cost, n-2));

return dp[n];

}

**Step 4 - Optimization 2 -Bottom Up DP - Convert recursion to iteration** - Getting rid of recursive stack

// Bottom up tabulation - O(n) 1ms

public int minCostClimbingStairs(int[] cost) {

int n = cost.length;

int[] dp = new int[n];

for (int i=0; i<n; i++) {

if (i<2) dp[i] = cost[i];

else dp[i] = cost[i] + Math.min(dp[i-1], dp[i-2]);

}

return Math.min(dp[n-1], dp[n-2]);

}

**Step 5 - Optimization 3 - Fine Tuning - Reduce O(n) space to O(1)**.

// Bottom up computation - O(n) time, O(1) space

public int minCostClimbingStairs(int[] cost) {

int n = cost.length;

int first = cost[0];

int second = cost[1];

if (n<=2) return Math.min(first, second);

for (int i=2; i<n; i++) {

int curr = cost[i] + Math.min(first, second);

first = second;

second = curr;

}

return Math.min(first, second);

}