

Recently I have done Dynamic Programming cause it is one of the hardest topics in an interview prep. After solving approximately 90-100 problems in DP I have noticed that there are few patterns that can be found in different problems.

**Patterns**

Minimum(maximum) Path to reach a target

Distinct ways

Merge intervals

Dp on strings

Decision dp

Subarray problems(1-d) dp

Minimum (Maximum) Path to Reach a Target

Problem list: https://lee tcode.com/list/55ac4kuc.

### **Statement**

Given a target find minimum (maximum) cost / path / sum to reach the target.

### **Approach**

Choose minimum (maximum) path among all possible paths before the current state, then add value for the current state.

routes[i] = min(routes[i-1], routes[i-2], ... , routes[i-k]) + cost[i]

Generate optimal solutions for all values in the target and return the value for the target.

### **Top-Down**

for (int j = 0; j < ways.size(); ++j) {

result = min(result, topDown(target - ways[j]) + cost/ path / sum);

}

return memo[/\*state parameters\*/] = result;

### **Bottom-Up**

for (int i = 1; i <= target; ++i) {

for (int j = 0; j < ways.size(); ++j) {

if (ways[j] <= i) {

dp[i] = min(dp[i], dp[i - ways[j]] + cost / path / sum) ;

}

}

}

return dp[target]

### **Similar Problems**

[746. Min Cost Climbing Stairs](https://leetcode.com/problems/min-cost-climbing-stairs/) Easy

### **Top-Down**

int result = min(minCost(n-1, cost, memo), minCost(n-2, cost, memo)) + (n == cost.size() ? 0 : cost[n]);

return memo[n] = result;

### **Bottom-Up**

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

dp[i] = min(dp[i-1], dp[i-2]) + (i == n ? 0 : cost[i]);

}

return dp[n]

[64. Minimum Path Sum](https://leetcode.com/problems/minimum-path-sum/) Medium

### **Top-Down**

int result = min(pathSum(i+1, j, grid, memo), pathSum(i, j+1, grid, memo)) + grid[i][j];

return memo[i][j] = result;

### **Bottom-Up**

for (int i = 1; i < n; ++i) {

for (int j = 1; j < m; ++j) {

grid[i][j] = min(grid[i-1][j], grid[i][j-1]) + grid[i][j];

}

}

return grid[n-1][m-1]

[322. Coin Change](https://leetcode.com/problems/coin-change/) Medium

### **Top-Down**

for (int i = 0; i < coins.size(); ++i) {

if (coins[i] <= target) { // check validity of a sub-problem

result = min(ans, CoinChange(target - coins[i], coins) + 1);

}

}

return memo[target] = result;

### **Bottom-Up**

for (int j = 1; j <= amount; ++j) {

for (int i = 0; i < coins.size(); ++i) {

if (coins[i] <= j) {

dp[j] = min(dp[j], dp[j - coins[i]] + 1);

}

}

}

[931. Minimum Falling Path Sum](https://leetcode.com/problems/minimum-falling-path-sum/) Medium

[983. Minimum Cost For Tickets](https://leetcode.com/problems/minimum-cost-for-tickets/) Medium

[650. 2 Keys Keyboard](https://leetcode.com/problems/2-keys-keyboard/) Medium

[279. Perfect Squares](https://leetcode.com/problems/perfect-squares/) Medium

[1049. Last Stone Weight II](https://leetcode.com/problems/last-stone-weight-ii/) Medium

[120. Triangle](https://leetcode.com/problems/triangle/) Medium

[474. Ones and Zeroes](https://leetcode.com/problems/ones-and-zeroes/) Medium

[221. Maximal Square](https://leetcode.com/problems/maximal-square/) Medium

[322. Coin Change](https://leetcode.com/problems/coin-change/) Medium

[1240. Tiling a Rectangle with the Fewest Squares](https://leetcode.com/problems/tiling-a-rectangle-with-the-fewest-squares/) Hard

[174. Dungeon Game](https://leetcode.com/problems/dungeon-game/) Hard

[871. Minimum Number of Refueling Stops](https://leetcode.com/problems/minimum-number-of-refueling-stops/) Hard

# Distinct Ways

Problem List: <https://leetcode.com/list/55ajm50i>

Generate problem statement for this pattern

### **Statement**

Given a target find a number of distinct ways to reach the target.

### **Approach**

Sum all possible ways to reach the current state.

routes[i] = routes[i-1] + routes[i-2], ... , + routes[i-k]

Generate sum for all values in the target and return the value for the target.

### **Top-Down**

for (int j = 0; j < ways.size(); ++j) {

result += topDown(target - ways[j]);

}

return memo[/\*state parameters\*/] = result;

### **Bottom-Up**

for (int i = 1; i <= target; ++i) {

for (int j = 0; j < ways.size(); ++j) {

if (ways[j] <= i) {

dp[i] += dp[i - ways[j]];

}

}

}

return dp[target]

### **Similar Problems**

[70. Climbing Stairs](https://leetcode.com/problems/climbing-stairs/) Easy

### **Top-Down**

int result = climbStairs(n-1, memo) + climbStairs(n-2, memo);

return memo[n] = result;

### **Bottom-Up**

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

for (int step = 1; step <= 2; ++step) {

dp[stair] += dp[stair-step];

}

}

[62. Unique Paths](https://leetcode.com/problems/unique-paths/) Medium

### **Top-Down**

int result = UniquePaths(x-1, y) + UniquePaths(x, y-1);

return memo[x][y] = result;

### **Bottom-Up**

for (int i = 1; i < m; ++i) {

for (int j = 1; j < n; ++j) {

dp[i][j] = dp[i][j-1] + dp[i-1][j];

}

}

[1155. Number of Dice Rolls With Target Sum](https://leetcode.com/problems/number-of-dice-rolls-with-target-sum/) Medium

for (int rep = 1; rep <= d; ++rep) {

vector<int> new\_ways(target+1);

for (int already = 0; already <= target; ++already) {

for (int pipe = 1; pipe <= f; ++pipe) {

if (already - pipe >= 0) {

new\_ways[already] += ways[already - pipe];

new\_ways[already] %= mod;

}

}

}

ways = new\_ways;

}

**Note**

Some questions point out the number of repetitions, in that case, add one more loop to simulate every repetition.

[688. Knight Probability in Chessboard](https://leetcode.com/problems/knight-probability-in-chessboard/) Medium

[494. Target Sum](https://leetcode.com/problems/target-sum/) Medium

[377. Combination Sum IV](https://leetcode.com/problems/combination-sum-iv/) Medium

[935. Knight Dialer](https://leetcode.com/problems/knight-dialer/) Medium

[1223. Dice Roll Simulation](https://leetcode.com/problems/dice-roll-simulation/) Medium

[416. Partition Equal Subset Sum](https://leetcode.com/problems/partition-equal-subset-sum/) Medium

[808. Soup Servings](https://leetcode.com/problems/soup-servings/) Medium

[790. Domino and Tromino Tiling](https://leetcode.com/problems/domino-and-tromino-tiling/) Medium

[801. Minimum Swaps To Make Sequences Increasing](https://leetcode.com/problems/minimum-swaps-to-make-sequences-increasing/)

[673. Number of Longest Increasing Subsequence](https://leetcode.com/problems/number-of-longest-increasing-subsequence/) Medium

[63. Unique Paths II](https://leetcode.com/problems/unique-paths-ii/) Medium

[576. Out of Boundary Paths](https://leetcode.com/problems/out-of-boundary-paths/) Medium

[1269. Number of Ways to Stay in the Same Place After Some Steps](https://leetcode.com/problems/number-of-ways-to-stay-in-the-same-place-after-some-steps/) Hard

[1220. Count Vowels Permutation](https://leetcode.com/problems/count-vowels-permutation/) Hard

# Merging Intervals

Problem List: <https://leetcode.com/list/55aj8s16>

Generate problem statement for this pattern

### **Statement**

Given a set of numbers find an optimal solution for a problem considering the current number and the best you can get from the left and right sides.

### **Approach**

Find all optimal solutions for every interval and return the best possible answer.

// from i to j

dp[i][j] = dp[i][k] + result[k] + dp[k+1][j]

Get the best from the left and right sides and add a solution for the current position.

### **Top-Down**

for (int k = i; k <= j; ++k) {

result = max(result, topDown(nums, i, k-1) + result[k] + topDown(nums, k+1, j));

}

return memo[/\*state parameters\*/] = result;

### **Bottom-Up**

for(int l = 1; l<n; l++) {

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

int j = i+l;

for(int k = i; k<j; k++) {

dp[i][j] = max(dp[i][j], dp[i][k] + result[k] + dp[k+1][j]);

}

}

}

return dp[0][n-1];

for(int l = 1; l<n; l++) {

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

int j = i+l;

for(int k = i; k<j; k++) {

dp[i][j] = max(dp[i][j], dp[i][k] + result[k] + dp[k+1][j]);

}

}

}

return dp[0][n-1]

### **Similar Problems**

[1130. Minimum Cost Tree From Leaf Values](https://leetcode.com/problems/minimum-cost-tree-from-leaf-values/) Medium

for (int l = 1; l < n; ++l) {

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

int j = i + l;

dp[i][j] = INT\_MAX;

for (int k = i; k < j; ++k) {

dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] + maxs[i][k] \* maxs[k+1][j]);

}

}

}

[96. Unique Binary Search Trees](https://leetcode.com/problems/unique-binary-search-trees/) Medium

[1039. Minimum Score Triangulation of Polygon](https://leetcode.com/problems/minimum-score-triangulation-of-polygon/) Medium

[546. Remove Boxes](https://leetcode.com/problems/remove-boxes/) Medium

[1000. Minimum Cost to Merge Stones](https://leetcode.com/problems/minimum-cost-to-merge-stones/) Medium

[312. Burst Balloons](https://leetcode.com/problems/burst-balloons/) Hard

### **Top-Down**

for (int k = i; k <= j; ++k) {

result = max(result, topDown(nums, i, k-1, memo) + (i-1 >= 0 ? nums[i-1] : 1) \* nums[k] \* (j+1 < nums.size() ? nums[j+1] : 1) + topDown(nums, k+1, j, memo));

}

return memo[i][j] = result;

### **Bottom-Up**

for(int l = 1; l < n; l++) {

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

int j = i+l;

for(int k = i; k <= j; k++) {

dp[i][j] = max(dp[i][j], (((k>i && k>0) ? dp[i][k-1] : 0) + (i>0 ? nums[i-1] : 1) \* nums[k] \* (j<n-1 ? nums[j+1] : 1) + ((k<j && k<n-1) ? dp[k+1][j] : 0)));

}

}

}

return dp[0][n-1];

[375. Guess Number Higher or Lower II](https://leetcode.com/problems/guess-number-higher-or-lower-ii/) Medium

# DP on Strings

Problem List: <https://leetcode.com/list/55afh7m7>

General problem statement for this pattern can vary but most of the time you are given two strings where lengths of those strings are not big

### **Statement**

Given two strings s1 and s2, return some result.

### **Approach**

Most of the problems on this pattern requires a solution that can be accepted in O(n^2) complexity.

// i - indexing string s1

// j - indexing string s2

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= m; ++j) {

if (s1[i-1] == s2[j-1]) {

dp[i][j] = /\*code\*/;

} else {

dp[i][j] = /\*code\*/;

}

}

}

If you are given one string s the approach may little vary

for (int l = 1; l < n; ++l) {

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

int j = i + l;

if (s[i] == s[j]) {

dp[i][j] = /\*code\*/;

} else {

dp[i][j] = /\*code\*/;

}

}

}

[1143. Longest Common Subsequence](https://leetcode.com/problems/longest-common-subsequence/) Medium

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= m; ++j) {

if (text1[i-1] == text2[j-1]) {

dp[i][j] = dp[i-1][j-1] + 1;

} else {

dp[i][j] = max(dp[i-1][j], dp[i][j-1]);

}

}

}

[647. Palindromic Substrings](https://leetcode.com/problems/palindromic-substrings/) Medium

for (int l = 1; l < n; ++l) {

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

int j = i + l;

if (s[i] == s[j] && dp[i+1][j-1] == j-i-1) {

dp[i][j] = dp[i+1][j-1] + 2;

} else {

dp[i][j] = 0;

}

}

}

[516. Longest Palindromic Subsequence](https://leetcode.com/problems/longest-palindromic-subsequence/) Medium

[1092. Shortest Common Supersequence](https://leetcode.com/problems/shortest-common-supersequence/) Medium

[72. Edit Distance](https://leetcode.com/problems/edit-distance/) Hard

[115. Distinct Subsequences](https://leetcode.com/problems/distinct-subsequences/) Hard

[712. Minimum ASCII Delete Sum for Two Strings](https://leetcode.com/problems/minimum-ascii-delete-sum-for-two-strings/) Medium

[5. Longest Palindromic Substring](https://leetcode.com/problems/longest-palindromic-substring/) Medium

# Decision Making

Problem List: <https://leetcode.com/list/55af7bu7>

The general problem statement for this pattern is forgiven situation decide whether to use or not to use the current state. So, the problem requires you to make a decision at a current state.

### **Statement**

Given a set of values find an answer with an option to choose or ignore the current value.

### **Approach**

If you decide to choose the current value use the previous result where the value was ignored; vice-versa, if you decide to ignore the current value use previous result where value was used.

// i - indexing a set of values

// j - options to ignore j values

for (int i = 1; i < n; ++i) {

for (int j = 1; j <= k; ++j) {

dp[i][j] = max({dp[i][j], dp[i-1][j] + arr[i], dp[i-1][j-1]});

dp[i][j-1] = max({dp[i][j-1], dp[i-1][j-1] + arr[i], arr[i]});

}

}

[198. House Robber](https://leetcode.com/problems/house-robber/) Easy

for (int i = 1; i < n; ++i) {

dp[i][1] = max(dp[i-1][0] + nums[i], dp[i-1][1]);

dp[i][0] = dp[i-1][1];

}

[121. Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/) Easy

[714. Best Time to Buy and Sell Stock with Transaction Fee](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/) Medium

[309. Best Time to Buy and Sell Stock with Cooldown](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/) Medium

[123. Best Time to Buy and Sell Stock III](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/) Hard

[188. Best Time to Buy and Sell Stock IV](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/) Hard

**Subarray Problems (1-D)**

kadanes algorithm: <https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/>

k concatenation maximum sum: <https://leetcode.com/problems/k-concatenation-maximum-sum/>

sum of subarray minimums: <https://leetcode.com/problems/sum-of-subarray-minimums/>

maximum length of repeated subarray: <https://leetcode.com/problems/maximum-length-of-repeated-subarray/>

total apeal of a string: <https://leetcode.com/problems/total-appeal-of-a-string/>

maximum product subarray: <https://www.geeksforgeeks.org/maximum-product-subarray/>

partiton array for maximum sum: <https://leetcode.com/problems/partition-array-for-maximum-sum/>