Before starting the topic let me introduce myself. I am a Mobile Developer currently working in Warsaw and spending my free time for interview preparations. I started to prepare for interviews two years ago. At that time I should say I could not solve the two sum problem. Easy problems seemed to me like hard ones so most of the time I had to look at editorials and discuss section. Currently, I have solved ~800 problems and time to time participate in contests. I usually solve 3 problems in a contest and sometimes 4 problems. Ok, lets come back to the topic.

Recently I have concentrated my attention on Dynamic Programming cause its one of the hardest topics in an interview prep. After solving ~140 problems in DP I have noticed that there are few patterns that can be found in different problems. So I did a research on that and find the following topics. I will not give complete ways how to solve problems but these patterns may be helpful in solving DP.

Patterns

Minimum (Maximum) Path to Reach a Target

Generate problem statement for this pattern

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.

```
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 `Easy`

```
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 `Medium`

```
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 `Medium`

```
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 `Medium`

983. Minimum Cost For Tickets `Medium`

650. 2 Keys Keyboard `Medium`

279. Perfect Squares `Medium`

1049. Last Stone Weight II `Medium`

120. Triangle `Medium`

474. Ones and Zeroes `Medium`

221. Maximal Square `Medium`

322. Coin Change `Medium`

1240. Tiling a Rectangle with the Fewest Squares `Hard`

174. Dungeon Game `Hard`

871. Minimum Number of Refueling Stops `Hard`

Distinct Ways

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.

```
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 `easy`

```
for (int stair = 2; stair <= n; ++stair) {
   for (int step = 1; step <= 2; ++step) {
       dp[stair] += dp[stair-step];
   }
}
```

62. Unique Paths `Medium`

```
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 `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 `Medium`

494. Target Sum `Medium`

377. Combination Sum IV `Medium`

935. Knight Dialer `Medium`

1223. Dice Roll Simulation `Medium`

416. Partition Equal Subset Sum `Medium`

808. Soup Servings `Medium`

790. Domino and Tromino Tiling `Medium`

801. Minimum Swaps To Make Sequences Increasing

673. Number of Longest Increasing Subsequence `Medium`

63. Unique Paths II `Medium`

576. Out of Boundary Paths `Medium`

1269. Number of Ways to Stay in the Same Place After Some Steps `Hard`

1220. Count Vowels Permutation `Hard`

Merging Intervals

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.

```
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 `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 `Medium`

1039. Minimum Score Triangulation of Polygon `Medium`

546. Remove Boxes `Medium`

1000. Minimum Cost to Merge Stones `Medium`

312. Burst Balloons `Hard`

375. Guess Number Higher or Lower II `Medium`

DP on Strings

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 `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 `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 `Medium`

1092. Shortest Common Supersequence `Medium`

72. Edit Distance `Hard`

115. Distinct Subsequences `Hard`

712. Minimum ASCII Delete Sum for Two Strings `Medium`

5. Longest Palindromic Substring `Medium`

Decision Making

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 `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 `Easy`

714. Best Time to Buy and Sell Stock with Transaction Fee `Medium`

309. Best Time to Buy and Sell Stock with Cooldown `Medium`

123. Best Time to Buy and Sell Stock III `Hard`

188. Best Time to Buy and Sell Stock IV `Hard`

I hope these tips will be helpful 😊