Patterns

Minimum (Maximum) Path to Reach a Target

Problem list: https://leetcode.com/list/55ac4kuc

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.

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 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 `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 `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 `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

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 `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 `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 `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

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 `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`

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 `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 `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

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 `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`