# Simple DP

In this chapter, we cover some simple DP algorithm. The core idea of DP is that you calculate the result based on the previous calculated result, this normally happened in the array and you need to scane the array properly.

## 119. Pascal's Triangle II

Easy

Given a non-negative index *k* where *k* ≤ 33, return the *k*th index row of the Pascal's triangle.

Note that the row index starts from 0.

  
In Pascal's triangle, each number is the sum of the two numbers directly above it.

**Example:**

**Input:** 3

**Output:** [1,3,3,1]

**Follow up:**

Could you optimize your algorithm to use only *O*(*k*) extra space?

### Analysis:

Each row is calculated from previous row, except for the first and last element. You do not need to remember all the rows just last row.

/// <summary>

/// Leet code #119. Pascal's Triangle II

/// Given an index k, return the kth row of the Pascal's triangle.

/// For example, given k = 3,

/// Return [1,3,3,1].

/// Note:

/// Could you optimize your algorithm to use only O(k) extra space?

/// </summary>

vector<int> LeetCode::getPascalTriangleRow(int rowIndex)

{

vector<int> level;

if (rowIndex < 0) return level;

for (int i = 0; i <= rowIndex; i++)

{

vector<int> prev\_level = level;

level.clear();

for (int j = 0; j <= i; j++)

{

if ((j == 0) || (j == i))

{

level.push\_back(1);

}

else

{

level.push\_back(prev\_level[j - 1] + prev\_level[j]);

}

}

}

return level;

}

## 562. Longest Line of Consecutive One in Matrix

Medium

Given a 01 matrix **M**, find the longest line of consecutive one in the matrix. The line could be horizontal, vertical, diagonal or anti-diagonal.

**Example:**

**Input:**

[[0,1,1,0],

[0,1,1,0],

[0,0,0,1]]

**Output:** 3

**Hint:** The number of elements in the given matrix will not exceed 10,000.

### Analysis:

There are four directions, left, up, diag and anti-diag, the previous result is either on previous rows or previous columns, so you only need one scan and just keep the result in 4 directions.

/// <summary>

/// Leet code #562. Longest Line of Consecutive One in Matrix

/// Given a 01 matrix M, find the longest line of consecutive one in the

/// matrix. The line could be horizontal, vertical, diagonal or

/// anti-diagonal.

///

/// Example:

///

/// Input:

/// [

/// [0,1,1,0],

/// [0,1,1,0],

/// [0,0,0,1]

/// ]

/// Output: 3

/// Hint: The number of elements in the given matrix will not exceed 10,000.

/// </summary>

int LeetCodeDP::longestLine(vector<vector<int>>& M)

{

vector<vector<vector<int>>> dp;

int max\_length = 0;

for (size\_t i = 0; i < M.size(); i++)

{

dp.push\_back(vector<vector<int>>(M[i].size(), vector<int>(4, 0)));

for (size\_t j = 0; j < M[i].size(); j++)

{

// if zero we ignore

if (M[i][j] == 1)

{

dp[i][j][0] = (i > 0 && M[i - 1][j] == 1) ?

dp[i - 1][j][0] + 1 : 1;

dp[i][j][1] = (j > 0 && M[i][j - 1] == 1) ?

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

dp[i][j][2] = (i > 0 && j > 0 && M[i - 1][j - 1] == 1) ?

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

dp[i][j][3] = (i > 0 && j < M[i].size() - 1 &&

M[i - 1][j + 1] == 1) ?

dp[i - 1][j + 1][3] + 1 : 1;

}

for (size\_t k = 0; k < 4; k++)

{

max\_length = max(max\_length, dp[i][j][k]);

}

}

}

return max\_length;

}

## 361. Bomb Enemy

Medium

Given a 2D grid, each cell is either a wall 'W', an enemy 'E' or empty '0' (the number zero), return the maximum enemies you can kill using one bomb.  
The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since the wall is too strong to be destroyed.  
**Note:** You can only put the bomb at an empty cell.

**Example:**

**Input:** [["0","E","0","0"],["E","0","W","E"],["0","E","0","0"]]

**Output:** 3

**Explanation:** For the given grid,

0 E 0 0

E 0 W E

0 E 0 0

Placing a bomb at (1,1) kills 3 enemies.

### Analysis:

If you select a position then calculate the number enemies get killed, you will end up with O(N^3), so you need some optimization. The idea is to scan from top-left to bottom-right. Then you can calculate the enemies on your left and up. Then you scan from bottom-right to top-left, calculate the enemies on your bottom and right.

/// <summary>

/// Leet code #361. Bomb Enemy

///

/// Given a 2D grid, each cell is either a wall 'W', an enemy 'E' or empty '0'

/// (the number zero), return the maximum enemies you can kill using one bomb.

/// The bomb kills all the enemies in the same row and column from the planted

/// point until it hits the wall since the wall is too strong to be destroyed.

/// Note that you can only put the bomb at an empty cell.

///

/// Example:

///

/// For the given grid

/// 0 E 0 0

/// E 0 W E

/// 0 E 0 0

///

/// return 3. (Placing a bomb at (1,1) kills 3 enemies)

/// </summary>

int LeetCode::maxKilledEnemies(vector<vector<char>>& grid)

{

if (grid.size() == 0 || grid[0].size() == 0) return 0;

int max\_enemies = 0;

pair<int, int> max\_pos;

vector<vector<pair<int, int>>> sum(grid.size(), vector<pair<int, int>>(grid[0].size()));

for (size\_t i = 0; i < grid.size(); i++)

{

for (size\_t j = 0; j < grid[i].size(); j++)

{

pair<int, int> count;

if (grid[i][j] == 'W')

{

count = make\_pair(0, 0);

}

else if (grid[i][j] == '0')

{

if (i == 0) count.first = 0;

else count.first = sum[i - 1][j].first;

if (j == 0) count.second = 0;

else count.second = sum[i][j - 1].second;

}

else if (grid[i][j] == 'E')

{

if (i == 0) count.first = 1;

else count.first = sum[i - 1][j].first + 1;

if (j == 0) count.second = 1;

else count.second = sum[i][j - 1].second + 1;

}

sum[i][j] = count;

}

}

for (int i = grid.size() - 1; i >= 0; i--)

{

for (int j = grid[i].size() - 1; j >= 0; j--)

{

if (grid[i][j] == 'W') continue;

if (i < (int)grid.size() - 1)

{

sum[i][j].first = max(sum[i + 1][j].first, sum[i][j].first);

}

if (j < (int)grid[i].size() - 1)

{

sum[i][j].second = max(sum[i][j + 1].second, sum[i][j].second);

}

if (grid[i][j] == '0')

{

int enemies = sum[i][j].first + sum[i][j].second;

if (enemies > max\_enemies)

{

max\_enemies = enemies;

max\_pos = make\_pair(i, j);

}

}

}

}

return max\_enemies;

}

## 213. House Robber II

Medium

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

**Example 1:**

**Input:** [2,3,2]

**Output:** 3

**Explanation:** You cannot rob house 1 (money = 2) and then rob house 3 (money = 2), because they are adjacent houses.

**Example 2:**

**Input:** [1,2,3,1]

**Output:** 4

**Explanation:** Rob house 1 (money = 1) and then rob house 3 (money = 3). Total amount you can rob = 1 + 3 = 4.

### Analysis:

It is very easy to think rob odd houses or even houses, but this is wrong. You should think this way, if you stand on any house, you have two choices, one is if you did not rob the previous house, which is to say the previous sum before last house plus the value of current house, another is sum on the last house (rob it or not actually not matter), and you should give up current house. The last sum become the previous sum of this house and the sum of this house become the previous sum of last house.

Because the house is in circle, you need to choose rob the first one or last one, in two iteration.

/// <summary>

/// Leet code #213. House Robber II

///

/// You are a professional robber planning to rob houses along a street.

/// Each house has a certain amount of money stashed. All houses at this

/// place are arranged in a circle. That means the first house is the

/// neighbor of the last one. Meanwhile, adjacent houses have security

/// system connected and it will automatically contact the police if two

/// adjacent houses were broken into on the same night.

///

/// Given a list of non-negative integers representing the amount of money

/// of each house, determine the maximum amount of money you can rob

/// tonight without alerting the police.

///

/// Example 1:

///

/// Input: [2,3,2]

/// Output: 3

/// Explanation: You cannot rob house 1 (money = 2) and then rob house 3

/// (money = 2), because they are adjacent houses.

///

/// Example 2:

///

/// Input: [1,2,3,1]

/// Output: 4

/// Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

/// Total amount you can rob = 1 + 3 = 4.

/// </summary>

int LeetCodeDP::robII(vector<int>& nums)

{

// special case

if (nums.empty()) return 0;

if (nums.size() == 1) return nums[0];

int last\_prev = 0;

int sum = 0;

int result = 0;

for (size\_t i = 0; i < nums.size() - 1; i++)

{

int prev = sum;

sum = max(last\_prev + nums[i], sum);

last\_prev = prev;

}

result = sum;

last\_prev = 0;

sum = 0;

for (size\_t i = 1; i < nums.size(); i++)

{

int prev = sum;

sum = max(last\_prev + nums[i], sum);

last\_prev = prev;

}

result = max(result, sum);

return result;

}

## 322. Coin Change

Medium

You are given coins of different denominations and a total amount of money *amount*. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

**Example 1:**

**Input:** coins = [1, 2, 5], amount = 11

**Output:** 3

**Explanation:** 11 = 5 + 5 + 1

**Example 2:**

**Input:** coins = [2], amount = 3

**Output:** -1

**Note**:  
You may assume that you have an infinite number of each kind of coin.

### Analysis:

To get any money value, you can add one more coin from past values. But if past value is impossible ignore it.

/// <summary>

/// Leet code #322. Coin Change

/// You are given coins of different denominations and a total amount of money

/// amount. Write a function to

/// compute the fewest number of coins that you need to make up that amount.

/// If that amount of money cannot

/// be made up by any combination of the coins, return -1.

/// Example 1:

/// coins = [1, 2, 5], amount = 11

/// return 3 (11 = 5 + 5 + 1)

/// Example 2:

/// coins = [2], amount = 3

/// return -1.

/// Note:

/// You may assume that you have an infinite number of each kind of coin.

/// </summary>

int LeetCodeDP::coinChange(vector<int>& coins, int amount)

{

vector<int> dp(amount + 1, -1);

for (int i = 0; i <= amount; i++)

{

if (i == 0) dp[i] = 0;

else if (dp[i] == -1) continue;

for (size\_t j = 0; j < coins.size(); j++)

{

// take care overflow

if ((long long)i + coins[j] > (long long) amount) continue;

if (dp[i + coins[j]] == -1)

{

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

}

else

{

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

}

}

}

return dp[amount];

}

## 91. Decode Ways

Medium

A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given a **non-empty** string containing only digits, determine the total number of ways to decode it.

**Example 1:**

**Input:** "12"

**Output:** 2

**Explanation:** It could be decoded as "AB" (1 2) or "L" (12).

**Example 2:**

**Input:** "226"

**Output:** 3

**Explanation:** It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

### Analysis:

Move forward is bettwe than look back, and add the current sum to the next position, if the current position is ‘1’ or ‘2’, we may add the current sym to the one after next position.

/// <summary>

/// Leet code 91. Decode Ways

///

/// A message containing letters from A-Z is being encoded to numbers using

/// the following mapping:

///

/// 'A' -> 1

/// 'B' -> 2

/// ...

/// 'Z' -> 26

/// Given a non-empty string containing only digits, determine the total

/// number of ways to decode it.

///

/// Example 1:

///

/// Input: "12"

/// Output: 2

/// Explanation: It could be decoded as "AB" (1 2) or "L" (12).

///

/// Example 2:

///

/// Input: "226"

/// Output: 3

/// Explanation: It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF"

/// (2 2 6).

/// </summary>

int LeetCodeDP::numDecodings(string s)

{

if (s.empty()) return 0;

vector<int> dp(s.size()+1);

dp[0] = 1;

for (size\_t i = 0; i < s.size(); i++)

{

if (s[i] != '0') dp[i+1] += dp[i];

if (i < s.size() - 1)

{

if ((s[i] == '1') ||

(s[i] == '2' && s[i + 1] >= '0' && s[i + 1] <= '6'))

{

dp[i + 2] += dp[i];

}

}

}

return dp[s.size()];

}

## 256. Paint House

Easy

There are a row of *n* houses, each house can be painted with one of the three colors: red, blue or green. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a *n* x*3* cost matrix. For example, costs[0][0] is the cost of painting house 0 with color red; costs[1][2] is the cost of painting house 1 with color green, and so on... Find the minimum cost to paint all houses.

**Note:**  
All costs are positive integers.

**Example:**

**Input:** [[17,2,17],[16,16,5],[14,3,19]]

**Output:** 10

**Explanation:** Paint house 0 into blue, paint house 1 into green, paint house 2 into blue. Minimum cost: 2 + 5 + 3 = 10.

### Analysis:

Record the minimum cost on every house with every color, choose the minimum cost based on different previous color

/// <summary>

/// Leet code 256. Paint House

///

/// There are a row of n houses, each house can be painted with one of

/// the three colors: red, blue or green. The cost of painting each

/// house with a certain color is different. You have to paint all

/// the houses such that no two adjacent houses have the same color.

///

/// The cost of painting each house with a certain color is

/// represented by a n x 3 cost matrix. For example, costs[0][0] is

/// the cost of painting house 0 with color red; costs[1][2] is the

/// cost of painting house 1 with color green, and so on... Find the

/// minimum cost to paint all houses.

///

/// Note:

/// All costs are positive integers.

///

/// Example:

/// Input: [[17,2,17],[16,16,5],[14,3,19]]

/// Output: 10

/// Explanation: Paint house 0 into blue, paint house 1 into green, paint

/// house 2 into blue.

/// Minimum cost: 2 + 5 + 3 = 10.

/// </summary>

int LeetCode::minCost(vector<vector<int>>& costs)

{

vector<vector<int>> sum;

if (costs.size() == 0) return 0;

for (size\_t i = 0; i < costs.size(); i++)

{

sum.push\_back(vector<int>(3));

if (i == 0)

{

sum[i][0] = costs[i][0];

sum[i][1] = costs[i][1];

sum[i][2] = costs[i][2];

}

else

{

sum[i][0] = costs[i][0] + min(sum[i - 1][1], sum[i - 1][2]);

sum[i][1] = costs[i][1] + min(sum[i - 1][0], sum[i - 1][2]);

sum[i][2] = costs[i][2] + min(sum[i - 1][0], sum[i - 1][1]);

}

}

return min(min(sum[costs.size() - 1][0], sum[costs.size() - 1][1]),

sum[costs.size() - 1][2]);

}

## 576. Out of Boundary Paths

Medium

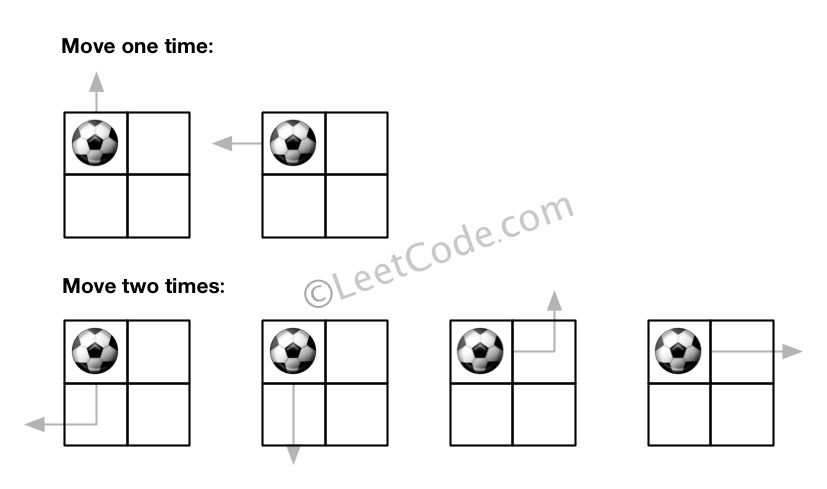
There is an **m** by **n** grid with a ball. Given the start coordinate **(i,j)** of the ball, you can move the ball to **adjacent** cell or cross the grid boundary in four directions (up, down, left, right). However, you can **at most** move **N** times. Find out the number of paths to move the ball out of grid boundary. The answer may be very large, return it after mod 109 + 7.

**Example 1:**

**Input:** m = 2, n = 2, N = 2, i = 0, j = 0

**Output:** 6

**Explanation:**

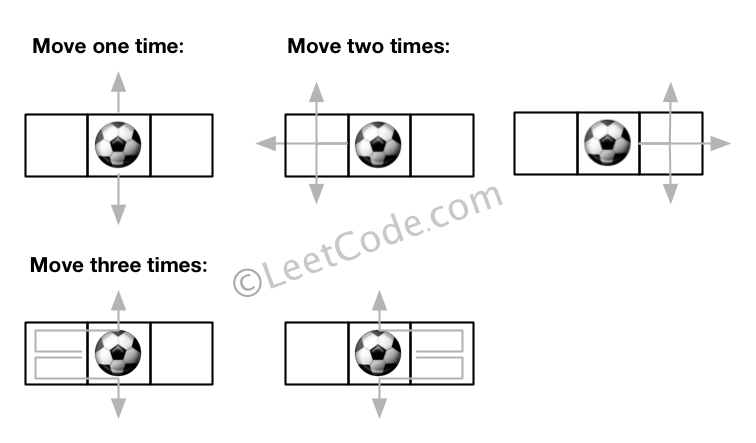


**Example 2:**

**Input:** m = 1, n = 3, N = 3, i = 0, j = 1

**Output:** 12

**Explanation:**



**Note:**

1. Once you move the ball out of boundary, you cannot move it back.
2. The length and height of the grid is in range [1,50].
3. N is in range [0,50].

### Analysis:

For such problem you need to iterate, iterate over from current position to next position, iterate from current grid to next grid.

/// <summary>

/// Leet code #576. Out of Boundary Paths

///

/// There is an m by n grid with a ball. Given the start coordinate (i,j) of

/// the ball, you can move the ball to adjacent cell or cross the grid boundary

/// in four directions (up, down, left, right). However, you can at most move N

/// times. Find out the number of paths to move the ball out of grid boundary.

/// The answer may be very large, return it after mod 10^9 + 7.

///

/// Example 1:

/// Input:m = 2, n = 2, N = 2, i = 0, j = 0

/// Output: 6

/// Explanation:

///

///

/// Example 2:

/// Input:m = 1, n = 3, N = 3, i = 0, j = 1

/// Output: 12

/// Explanation:

///

/// Note:

/// Once you move the ball out of boundary, you cannot move it back.

/// The length and height of the grid is in range [1,50].

/// N is in range [0,50].

/// </summary>

int LeetCodeDP::findPaths(int m, int n, int N, int i, int j)

{

vector<vector<int>> dp(m, vector<int>(n));

dp[i][j] = 1;

int result = 0;

int M = 1000000007;

vector<vector<int>> directions = { {-1, 0}, {1, 0}, {0, -1}, {0, 1} };

for (int k = 0; k < N; k++)

{

vector<vector<int>> next\_dp(m, vector<int>(n));

for (int r = 0; r < m; r++)

{

for (int c = 0; c < n; c++)

{

if (dp[r][c] == 0) continue;

for (size\_t d = 0; d < directions.size(); d++)

{

int next\_r = r + directions[d][0];

int next\_c = c + directions[d][1];

if (next\_r < 0 || next\_r >= m || next\_c < 0 || next\_c >= n)

{

result = (result + dp[r][c]) % M;

}

else

{

next\_dp[next\_r][next\_c] =

(next\_dp[next\_r][next\_c] + dp[r][c]) % M;

}

}

}

}

dp = next\_dp;

}

return result;

}