BITTIGER

CLASS_4
BACKTRACKING & DYNAMIC PROGRAMMING

Keen Wang

Content of Class_4

Memorized Search & Dynamic Programming	***DFS***
322. Coin Change	542. 01 Matrix
79. Word Search	
198. House Robber	

542. 01 Matrix

Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

Example 1:

Input:

```
0 0 0
0 1 0
0 0 0
```

Output:

```
0 0 0
0 1 0
0 0 0
```

Example 2:

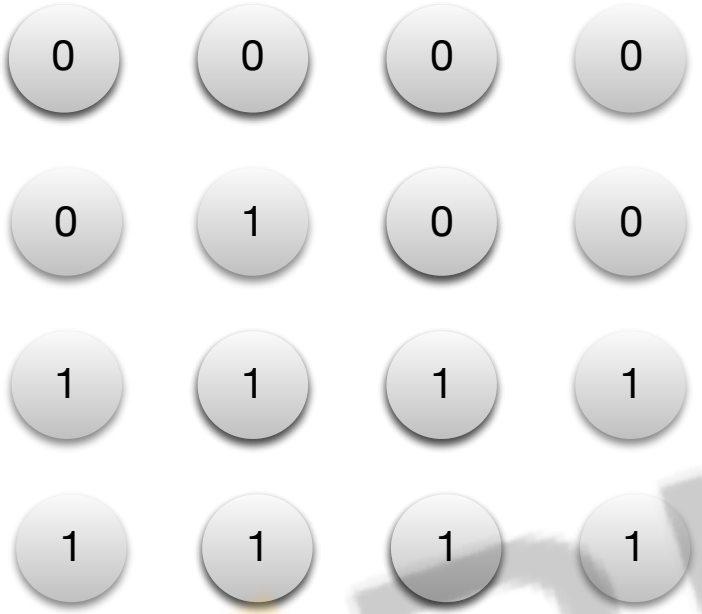
Input:

```
0 0 0
0 1 0
1 1 1
```

Output:

```
0 0 0
0 1 0
1 2 1
```

DFS 优化策略

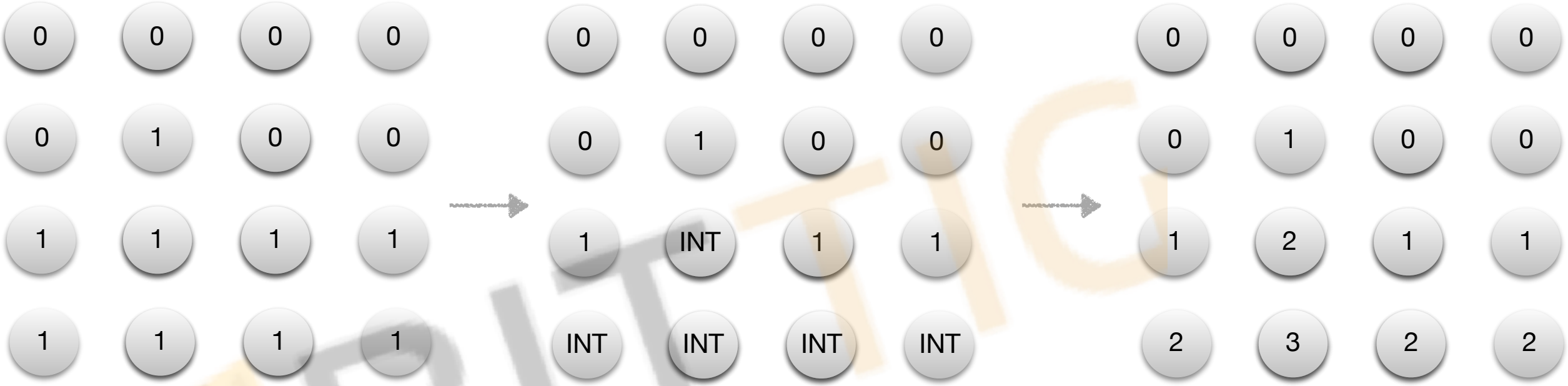


各种memorized办法（染色， cache）

Pruning(变tree为graph 避免重复计算)

预处理graph / 优化起始位置

优化，优化，还是优化！



原始矩阵

预处理（保留临界1点）

从1点开始DFS

```java
public class Solution {
    int row;
    int col;

    int[] dx = {1, -1, 0, 0};
    int[] dy = {0, 0, 1, -1};

    public List<List<Integer>> updateMatrix(List<List<Integer>> matrix) {

        if(matrix == null || matrix.size() == 0 || matrix.get(0).size() == 0){
            return matrix;
        }
        row = matrix.size();
        col = matrix.get(0).size();
        preSet(matrix);

        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(matrix.get(i).get(j) == 1){
                    helper(matrix, 1, i, j);
                }
            }
        }

        return matrix;
    }
```

四向dfs的小技巧 eg: game of life

预处理

DFS

```
29      public void preSet(List<List<Integer>> matrix){
30          for(int i = 0; i < row; i++){
31              for(int j = 0; j < col; j++){
32
33                  if(matrix.get(i).get(j) == 0){
34                      continue;
35                  }
36
37                  boolean flag = false;
38                  for(int k = 0; k < 4; k++){
39                      int x = i + dx[k];
40                      int y = j + dy[k];
41                      if(x < 0 || x >= row || y < 0 || y >= col){
42                          continue;
43                      }
44                      flag |= matrix.get(x).get(y) == 0;
45                  }
46                  if(!flag){
47                      matrix.get(i).set(j, Integer.MAX_VALUE);
48                  }
49
50              }
51          }
52      }
```

标记非边界1点为INT

```
54      public void helper(List<List<Integer>> matrix, int dist, int x, int y){
55          if(matrix.get(x).get(y) < dist){
56              return;
57          }
58           if(matrix.get(x).get(y) == dist && matrix.get(x).get(y) != 1){
59               return;
60           }
61
62          matrix.get(x).set(y, dist);
63
64          for(int i = 0; i < 4; i++){
65              int xNext = x + dx[i];
66              int yNext = y + dy[i];
67              if(xNext < 0 || xNext >= row || yNext < 0 || yNext >= col){
68                  continue;
69              }
70              helper(matrix, dist + 1, xNext, yNext);
71          }
72      }
73  }
```

corner/base case (遇0点和已优化路径返回)

current layer set dist

next layer : dist + 1

79. Word Search

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring.

The same letter cell may not be used more than once.

For example,

Given **board** =

```
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
```

word = `"ABCCED"`, -> returns `true`,

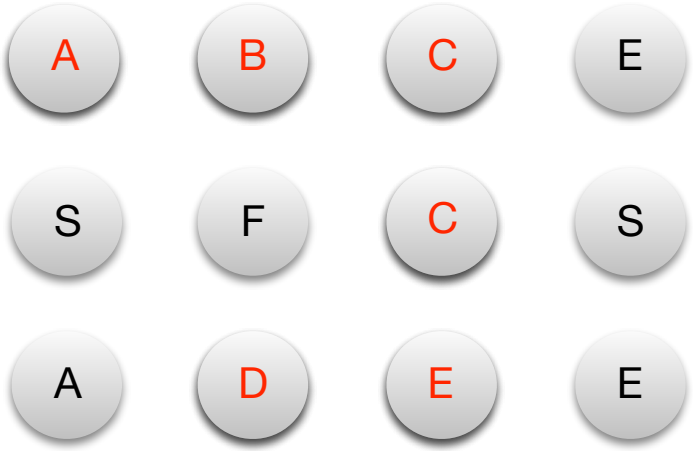
word = `"SEE"`, -> returns `true`,

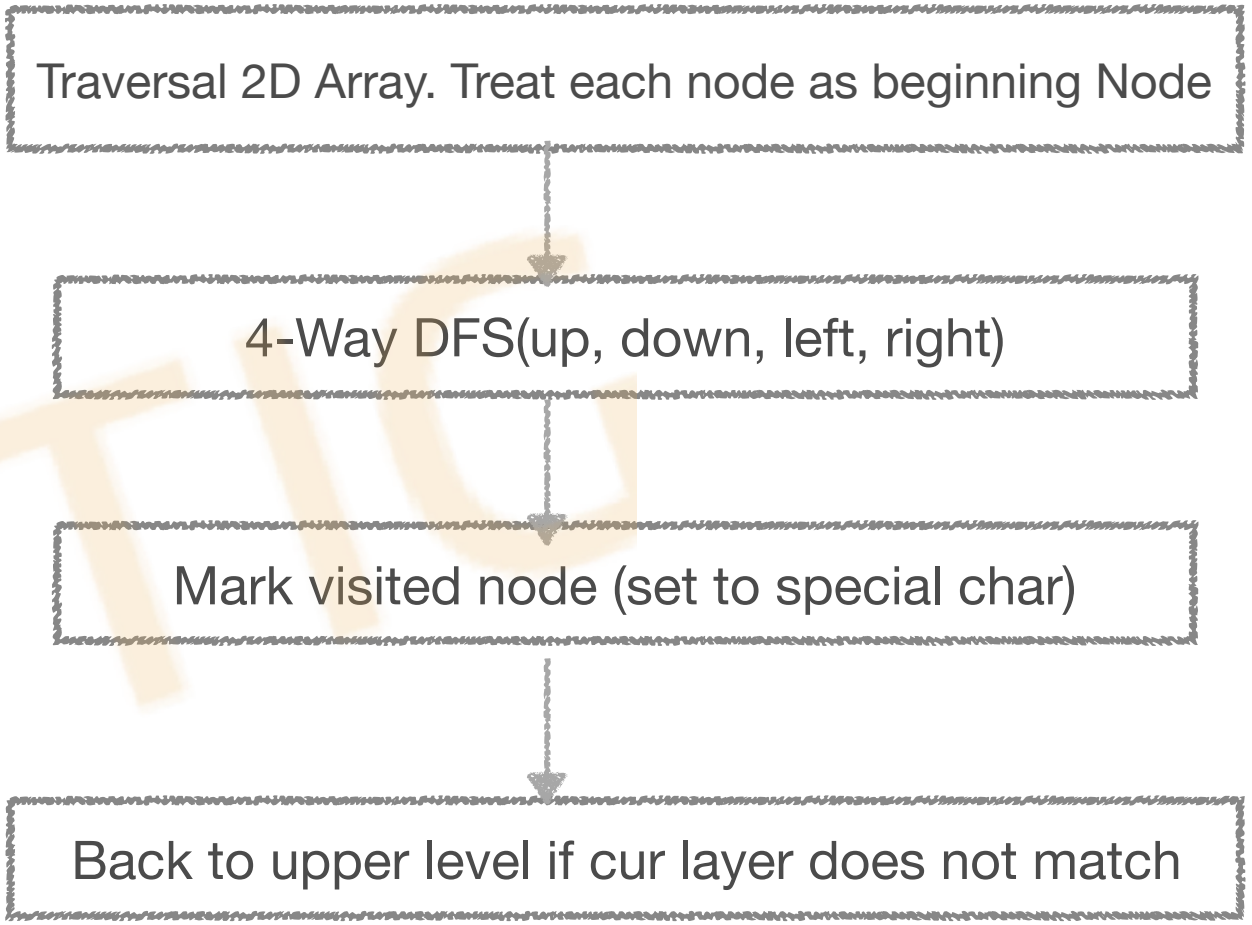
word = `"ABCB"`, -> returns `false`.

10 min

```
public boolean exist(char[][] board, String word) {

}
```

DFS 染色


A B C E
S F C S
A D E E


Traversal 2D Array. Treat each node as beginning Node
4-Way DFS(up, down, left, right)
Mark visited node (set to special char)
Back to upper level if cur layer does not match

```java
public boolean exist(char[][] board, String word) {
    if(word == null || word.length() == 0){
        return true;
    }
    char[] str = word.toCharArray();

    for (int i = 0; i < board.length; i++) {
      for (int j = 0; j < board[0].length; j++) {
        if (helper(board, str, i, j, 0)){
          return true;
        }
      }
    }
    return false;
}

public boolean helper(char[][] board, char[] str, int i, int j, int pos) {

    if(pos == str.length){
        return true;
    }
    if(i < 0 || i >= board.length || j < 0 || j >= board[0].length || board[i][j] != str[pos]){
        return false;
    }

    board[i][j] ^= 256;
    // char cur = board[i][j];
    // board[i][j] = '$';
    boolean res = helper(board, str, i - 1, j, pos + 1)
                || helper(board, str, i + 1, j, pos + 1)
                || helper(board, str, i, j - 1, pos + 1)
                || helper(board, str, i, j + 1, pos + 1);

    board[i][j] ^= 256;
    // board[i][j] = cur;
    return res;
}
```

if (true) return true directly

Mark visited (set to special char)

Time: O(mn*4^len)

Space: O(1)

Corner/Base Case

why not DP ?

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:

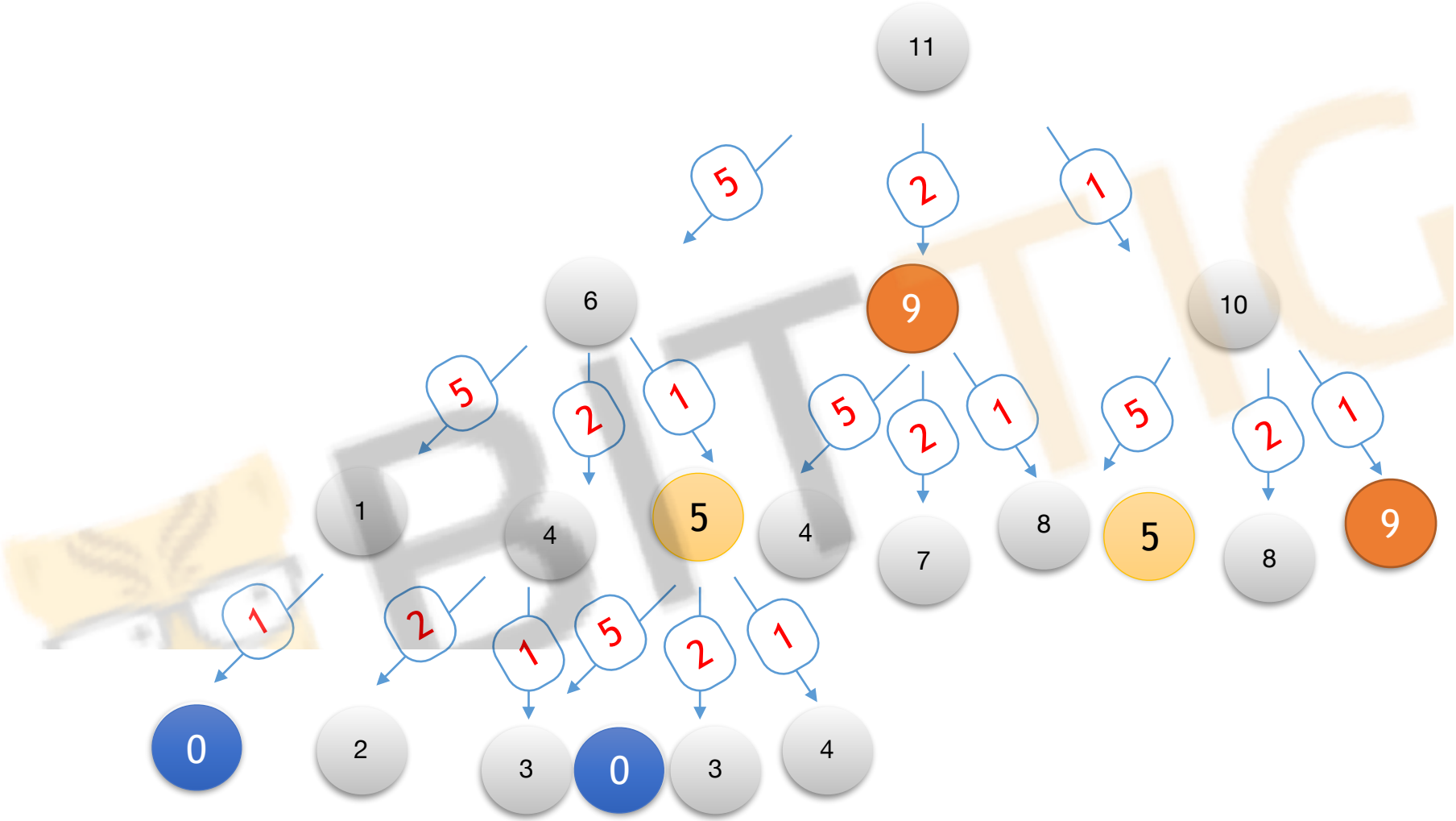
You may assume that you have an infinite number of each kind of coin.

20 min

```
public int coinChange(int[] coins, int amount) {

}
```

Memorized Search V.S. Dynamic Programming

combination sum （最多） 的变种题。
coin change （最少）



Memorized Search ——> DP

Divid & Conquer ——> Problem N depends on Problem N-1

Dynamic Programming 3 steps

1. DP 数组定义（最值为题max, min and true/false）
2. 初始值set up
3. induction rule（数学归纳递推公式）

dp[i]只依赖有限个dp[i - k]的时候Rolling array优化space （eg. Fibonacci）

```java
public int coinChange(int[] nums, int amount) {
    int[] count = new int[amount + 1];
    Arrays.fill(count, Integer.MAX_VALUE);
    Arrays.sort(nums);
    count[0] = 0;
    return helper(nums, amount, count);
}

public int helper(int[] nums, int remain, int[] count){

    if(count[remain] != Integer.MAX_VALUE){
        return count[remain];
    }
    int res = Integer.MAX_VALUE;

    for(int i = 0; i < nums.length; i++){

        if(remain >= nums[i]){
            int temp = helper(nums, remain - nums[i], count);
            if(temp != -1){
                res = Math.min(res, temp + 1);
            }
        }
    }

    count[remain] = (res == Integer.MAX_VALUE ? -1 : res);
    return count[remain];
}
```

注意返回值

P(N) depends on P(N - nums[i])

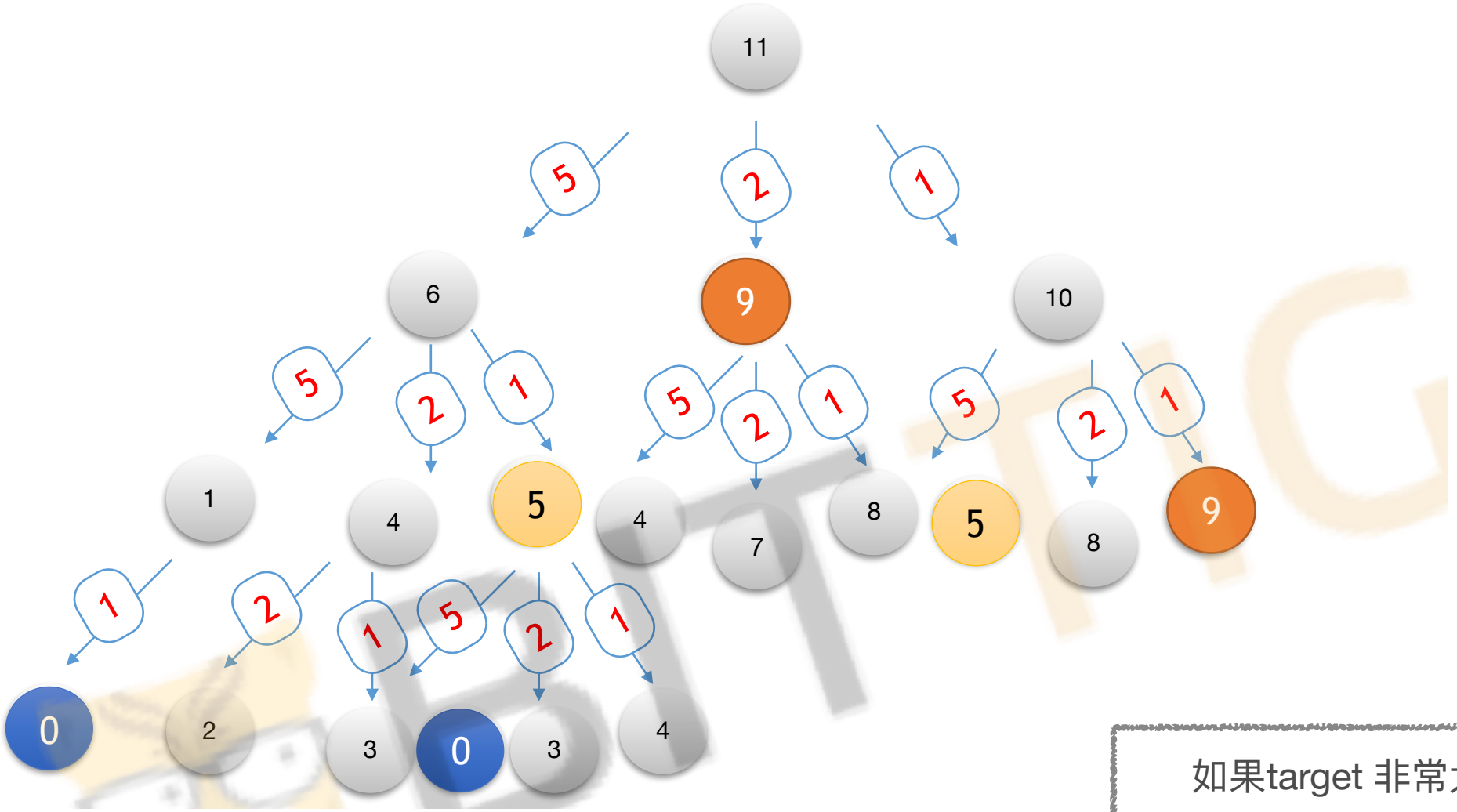
Divid and Conquer

```java
public int coinChangeDP(int[] coins, int amount) {
    if(amount <= 0){
        return 0;
    }
    int[] dp = new int[amount + 1];
    Arrays.fill(dp, Integer.MAX_VALUE);
    Arrays.sort(coins);
    dp[0] = 0;

    for(int i = 1; i <= amount; i++){
        for(int coin : coins){
            if(coin > i){
                break;
            }
            if(dp[i - coin] != Integer.MAX_VALUE){
                dp[i] = Math.min(dp[i], dp[i - coin] + 1);
            }
        }
    }

    return dp[amount] == Integer.MAX_VALUE ? -1 : dp[amount];
}
```

dp 数组定义

初始值设置

induction rule



如果target 非常大怎么办?

198. House Robber

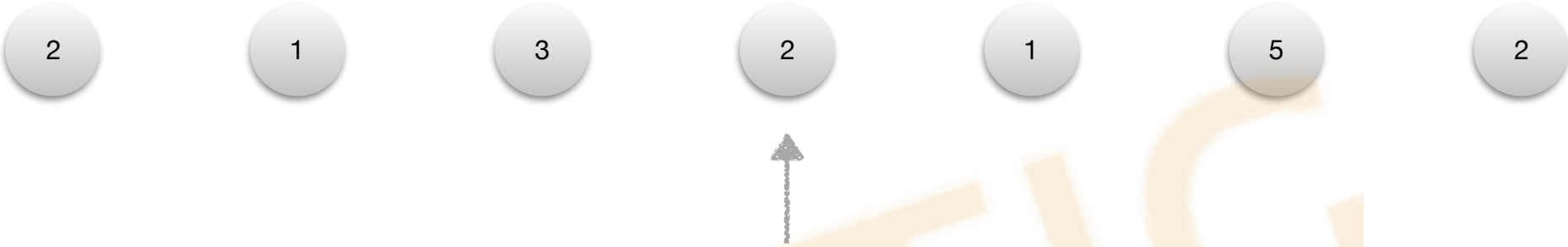
You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses 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**.

20 min

```
public int rob(int[] nums) {

}
```

Memorized Search V.S. Dynamic Programming



N house depends on (N-1) and (N-2) house

rob N-1 skip N OR rob N-2 and N

```java
public int robDFS(int[] nums){
    if(nums == null || nums.length == 0){
        return 0;
    }
    if(nums.length == 1){
        return nums[0];
    }
    return helper(nums, nums.length - 1);
}

public int helper(int[] nums, int pos){
    if(pos == 0){
        return nums[0];
    }
    if(pos == 1){
        return Math.max(nums[0], nums[1]);
    }

    return Math.max(helper(nums, pos - 1), helper(nums, pos - 2) + nums[pos]);
}
```

corner/base case

induction rule

```java
public int rob(int[] nums) {
    if(nums == null || nums.length == 0){
        return 0;
    }
    if(nums.length == 1){
        return nums[0];
    }
    int[] dp = new int[nums.length];
    dp[0] = nums[0];
    dp[1] = Math.max(nums[0], nums[1]);

    for(int i = 2; i < nums.length; i++){
        dp[i] = Math.max(dp[i - 1], dp[i - 2] + nums[i]);
    }
    return dp[nums.length - 1];
}
```

dp 数组定义

初始值设置

induction rule

如果有负数怎么办?

规定必须连续抢怎么办?

Homework

Memorized Search & Dynamic Programming	***DFS & BackTracking***
64. Minimum Path Sum	291 Word Pattern II
265 Paint House II	51 N-Queens
213. House Robber II	
70. Climbing Stairs	
121. Best Time to Buy and Sell Stock	
53. Maximum Subarray	
139. Word Break	
279. Perfect Squares	

Q & A

Thank you