# 【第四十五课】状态机模型与语言解释器(一)

## 1、1575. 统计所有可行路径

1. 题目要求到达finish的方案数，所以dp数组的一个维度可以定为到达的城市；并且到达某个城市时的油量是不固定的，所以dp数组的另一个维度可以定为剩余的油量。dp[i][j]表示到达城市i时，剩下j格油，的方案数
2. 来看状态转移方程，假设从城市a出发，到达城市b， consume = abs(loactions[b] - locations[a])
3. 因此dp[b][j] = dp[b][j] + dp[a][j+consume]
4. 根据题意，初始条件dp[start][fuel] = 1;
5. 由于油量是单调递减的，因此可以考虑最外层枚举当前剩余的油量，第二层枚举当前到达的城市，第三层枚举上一步出发的城市。

```
/**
 * @param {number[]} locations
 * @param {number} start
 * @param {number} finish
 * @param {number} fuel
 * @return {number}
 */
var countRoutes = function(locations, start, finish, fuel) {
    const mod = 1e9+7;
    const n = locations.length;
    const dp = Array(fuel + 1).fill(0).map(() =>Array(n).fill(0));
    dp[0][start] = 1;
    let res = dp[0][finish];
    for(let cost = 1;cost <= fuel;cost++){
        for(let i = 0;i < n;i++){
            for(let j = 0;j < n;j++){
                if(i == j){
                    continue;
                }
                const abs = Math.abs(locations[i] - locations[j]);
                if(cost >= abs){
                    dp[cost][i] += dp[cost - abs][j];
                    dp[cost][i] %= mod;
                }
            }
        }
        res += dp[cost][finish];
        res %= mod;
    }
    return res;
```

```
};
```

## 2、2218. 从栈中取出 K 个硬币的最大面值和

1. 对每个栈求其前缀和 sum，sum 的第 j 个元素视作一个体积为 j，价值为 sum[j] 的物品。
2. 问题转化成求从 n 个物品组里面取物品体积和为 k 的物品，且每组至多取一个物品时的物品价值最大和，这种的就是分组背包模型。
3. 定义 f[i][j] 表示从前 i 个组取体积之和为 j 的物品时，物品价值之和的最大值。
4. 枚举第 i 个组的所有物品，设当前物品体积为 w，价值为 v，则有公式：最后答案为 f[n][k]。
5. $$f[i][j] = \max(f[i][j], f[i-1][j-w] + v)$$

```
/**
 * @param {number[][]} piles
 * @param {number} k
 * @return {number}
 */
var maxValueOfCoins = function(piles, k) {
    const f = new Array(k + 1).fill(0);
    let sumN = 0;
    for(const pile of piles){
        let n = pile.length;
        // 单独去计算前缀和
        for(let i = 1;i < n;i++){
            pile[i] += pile[i - 1];
        }
        // 优化：j 从前 i的栈的大小之和开始枚举（不超过k）
        sumN = Math.min(sumN + n,k);
        for(let j = sumN;j;j--){
            // w 从0开始，物品的体积 w + 1
            for(let w = 0;w < Math.min(n,j);w++){
                f[j] = Math.max(f[j],f[j - w - 1] + pile[w]);
            }
        }
    }
    return f[k];
};
```

## 3、1583. 统计不开心的朋友

1. 创建 n 行 n 列的二维数组 order，其中 order[i][j] 表示朋友 j 在 i 的朋友列表中的亲近程度下标。

遍历 preferences 即可填入 order 中的全部元素的值。

2. 所有的朋友被分成二分之n 对，为了快速知道每位朋友的配对的朋友，对于配对情况也需要进行预处理。创建长度为 n 的数组 match，如果 x 和 y 配对，则有 match[y]=x。
3. 遍历从 0 到 n−1 的每位朋友 x，进行如下操作。找到与朋友 x 配对的朋友 y。
4. 找到朋友 y 在朋友 x 的朋友列表中的亲近程度下标，记为 index。
5. 朋友 x 的朋友列表中的下标从 0 到 index−1 的朋友都是可能的 u。遍历每个可能的 u，找到与朋友 u 配对的朋友 v。
6. 如果 order[u][x]<order[u][v]，则 x 是不开心的朋友。
7. 需要注意的是，对于每个朋友 x，只要能找到一个满足条件的四元组 (x,y,u,v)，则 xx 就是不开心的朋友。

```
/**
 * @param {number} n
 * @param {number[][]} preferences
 * @param {number[][]} pairs
 * @return {number}
 */
var unhappyFriends = function(n, preferences, pairs) {
    const order = new Array(n).fill(0).map(() => new Array(n).fill(0));
    for(let i = 0;i < n;i++){
        for(let j = 0;j < n - 1;j++){
            order[i][preferences[i][j]] = j;
        }
    }
    const match = new Array(n).fill(0);
    for(const pair of pairs){
        let preSon0 = pair[0],
        preSon1 = pair[1];
        match[preSon0] = preSon1;
        match[preSon1] = preSon0;
    }
    let unhappyCount = 0;
    for(let x = 0;x < n;x++){
        const y = match[x];
        const index = order[x][y];
        for(let i = 0;i < index;i++){
            const u = preferences[x][i];
            const v = match[u];
            if(order[u][x] < order[u][v]){
                unhappyCount++;
                break;
            }
        }
    }
    return unhappyCount;
};
```

# 4、1585. 检查字符串是否可以通过排序子字符串得到另一个字符串

1. 由于字符只包括0-9,因此可以用一个长度为10的包含栈的数组来记录它们的位置。
2. 从后往前遍历源字符串，这样就能保证先出现的数字在栈顶，方便取用。
3. 遍历目标字符串，然后去栈中寻找需要的数字。
4. 如果对应数字的栈为空，说明没有找到数字，直接返回false。
5. 接下来，考虑题目的要求，通过升序排序才可以把后面的数字提到前面来，这就意味着如果前面有更小的数字，则无法从后面取数字来满足要求。
6. 因此接下来要遍历比当前数字小的所有栈，如果它们的栈顶元素的位置比当前数字的位置更靠前，则无法满足要求，返回false。

```
/**
 * @param {string} s
 * @param {string} t
 * @return {boolean}
 */
var isTransformable = function(s, t) {
    const n = s.length;
    const indices = Array.from({length:10},() => []);
    for(let i = n - 1;i >= 0;i--){
        indices[parseInt(s[i])].push(i);
    }
    for(let i = 0;i < n;i++){
        const num = parseInt(t[i]);
        if(indices[num].length){
            const index = indices[num].pop();
            for(let i = 0;i < num;i++){
                if(indices[i].length && indices[i][indices[i].length - 1]
                < index){
                    return false;
                }
            }
        }else{
            return false;
        }
    }
    return true;
};
```