1、递归实现

参考：https://blog.csdn.net/hit\_lk/article/details/53967627

|  |
| --- |
| **public** **class** Test {    @org.junit.Test  **public** **void** test() {  System.***out***.println("方案数:" + *getAllSchemeNum*(**new** **int**[]{ 5, 5, 5, 2, 3 }, 15));  } // out : 方案数:4  /\*\*  \* 从数组中选择和为sum的任意个数的组合数  \*/  **public** **static** **int** getAllSchemeNum(**int**[] arr, **int** sum) {  **int** count = 0;  // 将 选择一个数的组合数、选择两个数的组合数、...选择n个数的组合数 相加  **for** (**int** numToSelect = 1; numToSelect <= arr.length; numToSelect++) {  count += *getSchemeNumByNumToSelect*(arr, numToSelect, sum, 0);  }  **return** count;  }    /\*\*  \* 求【从数组的[arr[index], arr[length-1]]片段中获取和为sumToSelect的numToSelect个数】的方案数  \* **@param** arr 数组  \* **@param** numToSelect 还需要选择的数的个数  \* **@param** sumToSelect 还需要选择数之和  \* **@param** index 可选的范围的左边界  \* **@return**  \*/  **public** **static** **int** getSchemeNumByNumToSelect(**int**[] arr, **int** numToSelect, **int** sumToSelect, **int** index) {  **int** count = 0;  // 递归出口，如果数全部选择完成，则只需判定sumToSelect是否为零，如果为零，符合条件，返回1，否则返回0  **if** (numToSelect == 0) {  **return** sumToSelect == 0 ? 1 : 0;  }  /\*  \* 将问题按选择的第一个数的不同做分解，第一个数可选的范围为[index, arr.length - numToSelect]，  \* 所以就分解成了(arr.length - numToSelect - index + 1)个子问题。可为什么可选下标的右边界是  \* (arr.length - numToSelect)呢？是因为如果第一个数的下标是(arr.length - numToSelect + 1)，  \* 那么后面只剩(numToSelect - 2)个位置，是不够放下剩余的(numToSelect - 1)个值的。  \*/  **for** (**int** i = index; i <= arr.length - numToSelect; i++) {  **if** (arr[i] <= sumToSelect) {  /\*  \* 选择了第一个数arr[i]，还需要在剩余数组片段中选择和为(sumToSelect-arr[i])  \* 的(numToSelect-1)个数。  \* >> 需要递归  \*/  count += *getSchemeNumByNumToSelect*(arr, numToSelect - 1, sumToSelect - arr[i], i + 1);  }  }  **return** count;  }  } |

2、动态规划dp[][]

|  |
| --- |
| @Test  **public** **void** test1() {  // 指定输入 >>  **int**[] arr = { 5, 5, 10, 2, 3 };  **int** sum = 15;  // ================================================    // 初始化dp二维数组 【dp[i][j]表示用前i个数组成和为j的方案个数】  **int** rows = arr.length + 1;  **int** cols = sum + 1;  **int**[][] dp = **new** **int**[rows][cols];  // 初始化dp的第一列，用前i个数组成和为0的方案都只有1种，就是什么都不取；  **for** (**int** i = 0; i < rows; i++) {  dp[i][0] = 1;  }  // 初始化dp的第一行，用0个元素不能组成1~sum  **for** (**int** j = 1; j <= sum; j++) {  dp[0][j] = 0;  }    System.***out***.println("-- 处理前dp:");  **for** (**int** i = 0; i < rows; i++) {  System.***out***.println((i > 0 ? arr[i - 1] : "附加0") + "\t" + Arrays.*toString*(dp[i]));  }  System.***out***.println();    // 一行行的计算dp中每个元素的值  //System.out.println("附加0 \t"+Arrays.toString(dp[0]));  **for** (**int** i = 1; i < rows; i++) {  **for** (**int** j = 1; j <= sum; j++) {  /\*  \* 用前i个数来组成和为j的组合，所有成功的组合可分下面两种情况：  \* 1、 组合中不包含第i个数 ，即只用前i-1个数来组成和为j的组合。  \* 2、组合中包含第i个数，这要求第i个数不能比和大（前i-1个数要组成和为：j-第i个数）。  \*/  dp[i][j] = dp[i - 1][j];  **if** (arr[i-1] <= j) { // 第i个数为arr[i-1]  dp[i][j] += dp[i - 1][j - arr[i-1]];  }  }  //System.out.println(arr[i-1]+"\t"+Arrays.toString(dp[i]));  }    System.***out***.println("-- 处理后dp:");  **for** (**int** i = 0; i < rows; i++) {  System.***out***.println((i > 0 ? arr[i - 1] : "附加0") + "\t" + Arrays.*toString*(dp[i]));  }  System.***out***.println("答案：" + dp[rows-1][sum]);  }  /\* out:  -- 处理前dp:  附加0 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  5 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  5 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  10 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  2 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  3 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  -- 处理后dp:  附加0 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  5 [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  5 [1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  10 [1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2]  2 [1, 0, 1, 0, 0, 2, 0, 2, 0, 0, 2, 0, 2, 0, 0, 2]  3 [1, 0, 1, 1, 0, 3, 0, 2, 2, 0, 4, 0, 2, 2, 0, 4]  答案：4  \*/ |

3、动态规划dp[]

|  |
| --- |
| @Test  **public** **void** test2() {  // 指定输入 >>  **int**[] arr = { 5, 5, 10, 2, 3 };  **int** sum = 15;  // ================================================    **int**[] dp = **new** **int**[sum + 1];  dp[0] = 1;  System.***out***.println("init\t\t" + Arrays.*toString*(dp));    **for** (**int** i = 0; i < arr.length; i++) {  **for** (**int** j = sum; j >= arr[i]; j--) {  dp[j] = dp[j] + dp[j - arr[i]];  }  System.***out***.println(arr[i] + "\t\t" + Arrays.*toString*(dp));  }  System.***out***.println("答案：" + dp[sum]);  }  /\* out:  init [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  5 [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  5 [1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]  10 [1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 2]  2 [1, 0, 1, 0, 0, 2, 0, 2, 0, 0, 2, 0, 2, 0, 0, 2]  3 [1, 0, 1, 1, 0, 3, 0, 2, 2, 0, 4, 0, 2, 2, 0, 4]  答案：4  \*/ |