练习6 – 0-1背包

- 有 n 件物品和一个最大承重为 W 的背包，每件物品的重量是 w_i、价值是 v_i
  - 在保证总重量不超过 W 的前提下，选择某些物品装入背包，背包的最大总价值是多少？
  - 注意：每个物品只有 1 件，也就是每个物品只能选择 0 件或者 1 件

- 假设 values 是价值数组，weights 是重量数组
  - 编号为 k 的物品，价值是 values[k]，重量是 weights[k]，k ∈ [0, n)

- 假设 dp(i, j) 是 **最大承重为 j、有前 i 件物品可选** 时的最大总价值，i ∈ [1, n]，j ∈ [1, W]
  - dp(i, 0)、dp(0, j) 初始值均为 0
  - 如果 j < weights[i – 1]，那么 dp(i, j) = dp(i – 1, j)
  - 如果 j ≥ weights[i – 1]，那么 dp(i, j) = max { dp(i – 1, j), dp(i – 1, j – weights[i – 1]) + values[i – 1] }

小码哥教育 SEEMYGO

0-1背包 – 实现

```
int select(int[] values, int[] weights, int capacity) {
    if (values == null || values.length == 0) return 0;
    if (weights == null || weights.length == 0) return 0;
    if (weights.length != values.length) return 0;
    if (capacity <= 0) return 0;
    int[][] dp = new int[values.length + 1][capacity + 1];
    for (int i = 1; i <= values.length; i++) {
        for (int j = 1; j <= capacity; j++) {
            if (j < weights[i - 1]) {
                dp[i][j] = dp[i - 1][j];
            } else {
                dp[i][j] = Math.max(dp[i - 1][j],
                        dp[i - 1][j - weights[i - 1]] + values[i - 1]);
            }
        }
    }
    return dp[values.length][capacity];
}
```

0-1背包 – 非递归实现

- dp 数组的计算结果如下所示

	i \ j	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
v=6, w=2	1	0	0	6	6	6	6	6	6	6	6	6
v=3, w=2	2	0	0	6	6	9	9	9	9	9	9	9
v=5, w=6	3	0	0	6	6	9	9	9	9	11	11	14
v=4, w=5	4	0	0	6	6	9	9	9	10	11	13	14
v=6, w=4	5	0	0	6	6	9	9	12	12	15	15	15

0-1背包 – 非递归实现 – 一维数组

- dp(i, j) 都是由 dp(i – 1, k) 推导出来的，也就是说，第 i 行的数据是由它的上一行第 i – 1 行推导出来的
- 因此，可以使用一维数组来优化
- 另外，由于 k ≤ j ，所以 j 的遍历应该由大到小，否则导致数据错乱

```
int select(int[] values, int[] weights, int capacity) {
    if (values == null || values.length == 0) return 0;
    if (weights == null || weights.length == 0) return 0;
    if (weights.length != values.length) return 0;
    if (capacity <= 0) return 0;
    int[] dp = new int[capacity + 1];
    for (int i = 1; i <= values.length; i++) {
        for (int j = capacity; j >= 1; j--) {
            if (j < weights[i - 1]) continue;
            dp[j] = Math.max(dp[j],
                    dp[j - weights[i - 1]] + values[i - 1]);
        }
    }
    return dp[capacity];
}
```

0-1背包 – 非递归实现 – 一维数组优化

- 观察二维数组表，得出结论：j 的下界可以从 1 改为 weights[i – 1]

```
int select(int[] values, int[] weights, int capacity) {
    if (values == null || values.length == 0) return 0;
    if (weights == null || weights.length == 0) return 0;
    if (weights.length != values.length) return 0;
    if (capacity <= 0) return 0;
    int[] dp = new int[capacity + 1];
    for (int i = 1; i <= values.length; i++) {
        for (int j = capacity; j >= weights[i - 1]; j--) {
            dp[j] = Math.max(dp[j],
                    dp[j - weights[i - 1]] + values[i - 1]);
        }
    }
    return dp[capacity];
}
```

0-1背包 – 恰好装满

- 有 n 件物品和一个最大承重为 W 的背包，每件物品的重量是 w_i、价值是 v_i
- 在保证总重量恰好等于 W 的前提下，选择某些物品装入背包，背包的最大总价值是多少？
- 注意：每个物品只有 1 件，也就是每个物品只能选择 0 件或者 1 件

- dp(i, j) 初始状态调整
- dp(i, 0) = 0，总重量恰好为 0，最大总价值必然也为 0
- dp(0, j) = –∞（负无穷），j ≥ 1，负数在这里代表无法恰好装满

i \ j	0	1	2	3	4	5	6	7	8	9	10
0	0	-	-	-	-	-	-	-	-	-	-
v=6, w=2 1	0	-	6	-	-	-	-	-	-	-	-
v=3, w=2 2	0	-	6	-	9	-	-	-	-	-	-
v=5, w=6 3	0	-	6	-	9	-	5	-	11	-	14
v=4, w=5 4	0	-	6	-	9	4	5	10	11	13	14
v=6, w=4 5	0	-	6	-	9	4	12	10	15	13	14

小码哥教育 SEEMYGO

0-1背包 – 恰好装满 – 实现

```
int selectExactly(int[] values, int[] weights, int capacity) {
    if (values == null || values.length == 0) return -1;
    if (weights == null || weights.length == 0) return -1;
    if (weights.length != values.length) return 0;
    if (capacity <= 0) return 0;
    int[] dp = new int[capacity + 1];
    for (int j = 1; j <= capacity; j++) {
        dp[j] = Integer.MIN_VALUE;
    }
    for (int i = 1; i <= values.length; i++) {
        for (int j = capacity; j >= weights[i - 1]; j--) {
            dp[j] = Math.max(dp[j],
                    dp[j - weights[i - 1]] + values[i - 1]);
        }
    }
    return dp[capacity] < 0 ? -1 : dp[capacity];
}
```