Coin Change

每个面值的硬币无限使用

1. 问能否凑出 target

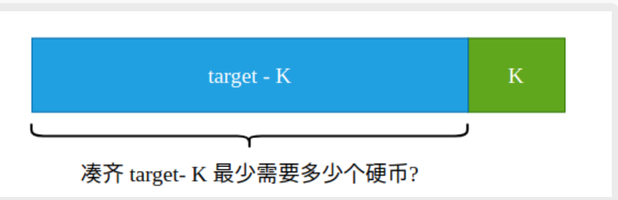
- 我们将目光放在最后一个硬币上，假设最后一个硬币面值为 K，那么我们只需要知道 target - K 这么多金额能否被凑出即可
- 如果 target - K 的面值可以被凑出，加上最后一枚硬币 K，就可以凑出 target
- 原问题是能否凑出 target，现在我们想知道能否凑齐 target - K，子问题出现
- f(t) 表示金额为 t 能够被凑出，那么 f(t) = OR f(t - coins[i])

```
bool canMakeUp(vector<int>& coins, int target) {
    vector<bool> dp(target + 1, false);

    dp[0] = true;

    for (int sub = 1; sub <= target; sub++) {
        for (int coin : coins) {
            if (sub >= coin)
                dp[sub] = dp[sub] || dp[sub - coin];
        }
    }
    return dp[target];
}
```

2. 问凑出 target 最少需要多少硬币



- 同样看最后一个硬币，假设最后一个硬币的面值为 K，如果我们知道了凑齐 target -K 需要的最少硬币数，那么再加上最后一枚 K，就是答案
- 原问题问凑出 target 最少硬币数，现在我们想知道凑齐 target - K 的最少硬币数，子问题出现
- f(t) 表示凑齐 t 的最少硬币数，那么 f(t) = min(f(t), f(t - coins[i]) + 1)

```
int minCoinsNumber(vector<int>& coins, int target) {
    vector<int> dp(target + 1, INT_MAX);

    dp[0] = 0;

    for (int sub = 1; sub <= target; sub++) {
        for (int coin : coins) {
            if (sub >= coin && dp[sub - coin] != INT_MAX)
                dp[sub] = min(dp[sub], dp[sub - coin] + 1);
        }
    }
    return dp[target] == INT_MAX ? -1 : dp[target];
}
```

3. 问凑出 target 有多少种凑法

- 2 + 1 + 1 与 1 + 2 + 1 属于不同的凑法
  - 继续看最后一枚硬币，假设最后一个硬币的面值为 K，如果我们知道了凑齐 target - K 有多少种凑法，也就知道了当最后一枚硬币为 K 时有多少种凑法，就是凑齐 target - K 的凑法
  - 原问题问凑出 target 有多少种凑法，现在我们想知道凑齐 target - K 有多少种凑法，子问题出现
  - f(t) 表示凑齐 t 的方案数，那么 f(t) = Sum(f(t - coins[i]))
- 2 + 1 + 1 与 1 + 2 + 1 属于相同的凑法
  - 只需要换一个 for 循环的顺序即可，因为我们需要一枚一枚的往上加，才不会出现重复

```
int totalSolutions(vector<int>& coins, int target) {
    vector<int> dp(target + 1, 0);

    dp[0] = 1;

    for (int sub = 1; sub <= target; sub++) {
        for (int coin : coins) {
            if (sub >= coin)
                dp[sub] += dp[sub - coin];
        }
    }
    return dp[target];
}
```

```
int change(vector<int>& coins, int target)  {
    vector<int> dp(target + 1, 0);

    dp[0] = 1;

    for (int coin : coins)  {
        for (int sub = coin; sub <= target; sub++) {
            dp[sub] += dp[sub - coin];
        }
    }
    return dp[target];
}
```

每个面值的硬币单次使用

1. 问能否凑出 target

```
for (int sub = 1; sub <= target; sub++) {
    for (int coin : coins)
        if (sub >= coin)
            dp[sub] = dp[sub] || dp[sub - coin];
}
```

首先我们来看一下无限硬币的情况。上面的第二层 for 循环表示我们可以使用无限硬币。因为当子问题的金额为 1 时，我们可以使用 coins 这么多硬币，当金额为 2 时，我们还是能够使用 coins 这么多硬币

- 由于每一个面值的硬币均只有一枚，所以硬币的数量也将作为状态
- 原问题在问 n 枚硬币能否凑出 target，现在我们想知道前 n-1 枚硬币能够凑出 target - K，子问题出现
- f(i, t) 表示前 i 枚硬币能否凑出 t
  那么 f(i, t) = f(i-1, t) OR f(i-1, t - coins[i-1])
- 可以看到 dp[i] 只和 dp[i-1] 有关，所以我们可以使用一维的数组对其进行覆盖更新，因篇幅有限，此处不再详述

```
bool canMakeUp2(vector<int>& coins, int target) {
    int n = coins.size();
    vector<vector<bool>> dp(n + 1, vector<bool>(target + 1, false));

    for (int i = 0; i <= n; i++) dp[i][0] = true;

    for (int i = 1; i <= n; i++) {
        for (int sub = 1; sub <= target; sub++) {
            if (coins[i-1] > sub) dp[i][sub] = dp[i-1][sub];
            else
                dp[i][sub] = dp[i-1][sub] || dp[i-1][sub - coins[i-1]];
        }
    }
    return dp[n][target];
}
```

2. 问凑出 target 最少需要多少硬币

只需要将 dp[i][j] 重新定义一下，变为前 i 枚硬币凑出 j 所需要的最小硬币数，将 OR 改为 min 即可

3. 问凑出 target 有多少种凑法

- 2 + 1 + 1 与 1 + 2 + 1 属于相同的凑法
  - dp[i][j] 表示前 i 枚硬币凑成 j 的总方案数，那么 dp[i][j] = dp[i-1][j] + dp[i-1][j - coins[i-1]]。
    当然了，若 i < coins[i-1]，那么 dp[i][j] = dp[i-1][j]