股票买卖问题

穷举框架：

```
for 状态1 in 状态1的所有取值 {
  for 状态2 in 状态2的所有取值 {
    for ... {
      dp[状态1][状态2][...] = 择优(选择1, 选择2, ...)
    }
  }
}
```

针对股票问题有三种选择：买入，卖出，无操作 ==> `buy,sell,reset`(代表其状态) 对其有次序要求：

1. `sell`必须在`buy`之后；
2. `buy`必须在`sell`之前。

因此`reset`应该有两种状态：

1. `buy`之后的`reset`;
2. `sell`之后`reset`。

因此其共有三种状态：

> 天数，交易次数，当前持有状态(0->没有持有，1->持有)

利用一个三维数组可以表示全部状态组合：`dp[i][k][0 or 1]`,$0<=i<n, 1 <= k < K $。 其中`n为天数，K为最大交易次数`. 全部状态的枚举为：

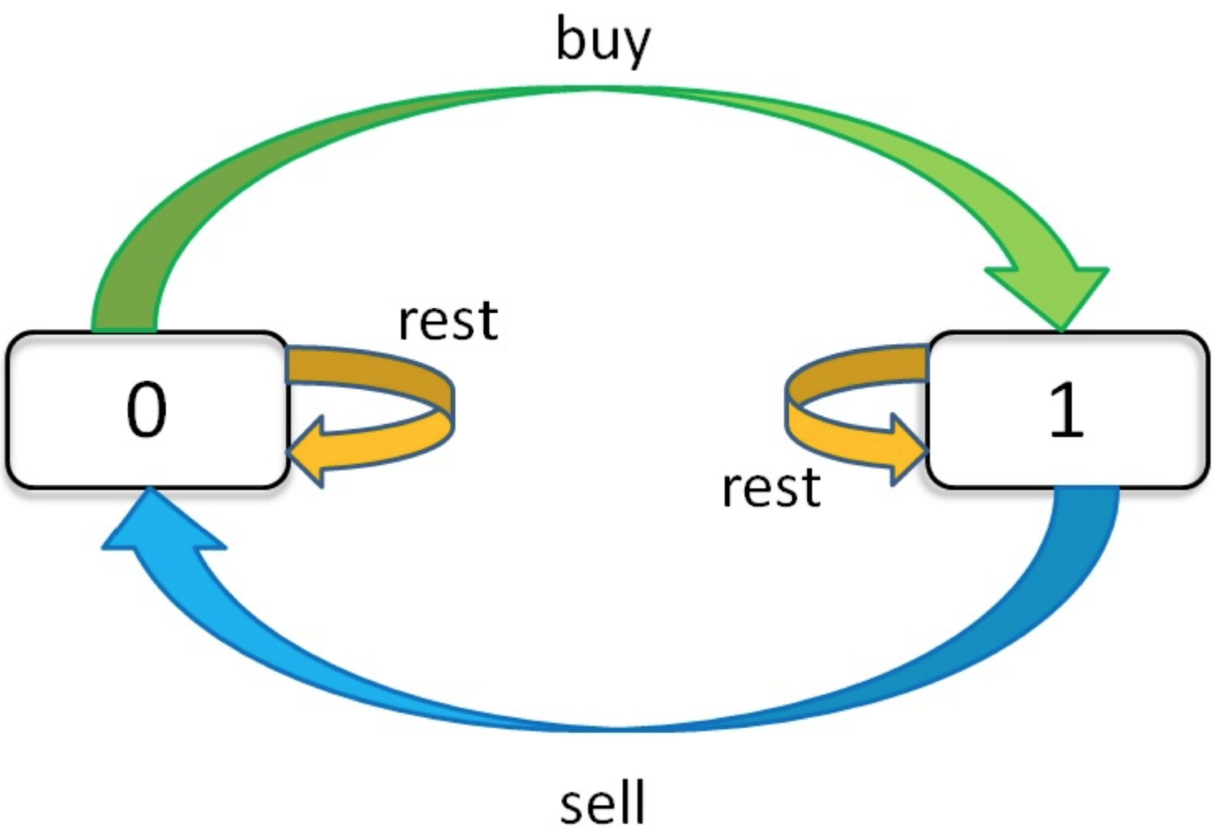
```
for 0 <= i < n {
  for 0 <= k < K {
    for s in {0,1} {
      dp[i][k][s] = max(buy, sell, reset)
    }
  }
}
```

对`dp table`可以描述为：`dp[3][2][1]`:今天是第三天，我持有股票，至今最多进行过两次交易；`dp[2][3][0]`:今天是第2天，我没有持有股票，至今最多交易三次。

题目转变为求`dp[n-1][K][0]`的最大值，即最后一天，我没有持有股票，最多交易`K`次的最大收益。

状态转移方程

其状态转移图为：



状态转移方程为：

```
dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i] )
            = max(  选择reset,   选择sell   )
解释：今天没有股票有两种可能
1. 前一天没有股票，今天也没购买
2. 前一天有股票，今天卖出

dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k -1][0] - prices[i])
            = max(  选择继续持有,     选择买入 )
解释：今天持有股票有两种可能
1. 前一天有股票，今天继续持有
2. 前一天没有股票，今天买入
```

定义`base case`:

```
dp[-1][k][0] = 0
解释：i从0开始，i=-1表示还没开始。
dp[-1][k][1] = -infinity
解释：还没开始时，不能持有股票所以为负无穷
dp[i][0][0] = 0
解释：k从1开始，k=0时不允许交易，利润为0
dp[i][0][1] = -infinity
解释：k从1开始，k=0时不允许交易,不能持有股票,所以为负无穷
```

状态转移方程：

```
base case:
dp[-1][k][0] = dp[i][0][0] = 0
dp[-1][k][1] = dp[i][0][1] = -infinity

状态转移方程:
dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i])
dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])

最终结果:
dp[n-1][k][0]
```

当允许交易一次时

因为只允许交易一次，所以动态规划数组`dp[n][k][rest]`中的`k`取值为$k \in [0,1], k为整数$， 当$k=0$是，动态规划数组`dp[n][0][rest] = 0`无意义，可以忽略; 只剩下$k=1$,此时`k`的值无变化，可以忽略，则动态规划数组变为:`dp[i][rest]`. 由此可以得出动态规划方程为： $$ dp[i][1][0] = std::max(dp[i-1][1][0], dp[i-1][1][1]+prices[i]) \ dp[i][1][1] = std::max(dp[i-1][1][1], dp[i-1][0][0] - prices[i])$$ 其中$dp[i-1][0][0] = 0$, 方程简化为： $$ dp[i][0] = std::max(dp[i-1][0], dp[i-1][1]+prices[i]) \ dp[i-1][1] = std::max(dp[i-1][1], -prices[i]) $$ 最后返回的结果为: `dp[n-1][0]`; 转化为代码如下：

```
class Solution {
public:
  int maxProfit(std::vector<int>& prices) {
    int                           size = prices.size();
    std::vector<std::vector<int>> dp(size + 1, std::vector<int>(2, 0));

    for (int i = 0; i < size; ++i) {
      if (-1 == i - 1) {
        dp[i][0] = 0;
        dp[i][1] = -prices[i];

        continue;
      }
      dp[i][0] = std::max(dp[i - 1][0], dp[i - 1][1] + prices[i]);
      dp[i][1] = std::max(dp[i - 1][1], -prices[i]);
    }

    return dp[size - 1][0];
  }
};
```

交易次数不限时

因为可以交易无穷次，所以动态规划数组`dp[n][k][rest]`中的`k`取值为$k = +\infty , k为整数$， $k与k-1$时相同的，对`k`区分无意义，动态规划数组`dp[n][k][rest]`中`k`的变化无意义，可以忽略; 则动态规划数组变为:`dp[i][rest]`. 由此可以得出动态规划方程为： $$ dp[i][k][0] = std::max(dp[i-1][k][0], dp[i-1][k]

[1]+prices[i]) \ dp[i][k][1] = std::max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])$$ 其中`k`变化无意义, 方程简化为： $$ dp[i][0] = std::max(dp[i-1][0], dp[i-1][1]+prices[i]) \ dp[i-1][1] = std::max(dp[i-1][1], dp[i-1][0]-prices[i]) $$ 最后返回的结果为: `dp[n-1][0]`;

```
class Solution {
public:
  int maxProfit(std::vector<int>& prices) {
    int                              size = prices.size();
    std::vector<std::vector<int>> dp(size + 1, std::vector<int>(2, 0));

    for (int i = 0; i < size; i++) {
      if (-1 == i - 1) {
        dp[i][0] = 0;
        dp[i][1] = -prices[i];
        continue;
      }

      dp[i][0] = std::max(dp[i - 1][0], dp[i - 1][1] + prices[i]);
      dp[i][1] = std::max(dp[i - 1][1], dp[i - 1][0] - prices[i]);
    }

    return dp[size - 1][0];
  }
};
```

最多交易2次

因为最多交易两次，所以动态规划数组`dp[n][k][rest]`中的`k`取值为$k = [0,1,2] , k为整数$， $k = 0$时相同的，`dp[n][0][rest] = 0`; 由此可以得出动态规划方程为： $$ dp[i][k][0] = std::max(dp[i-1][k][0], dp[i-1][k][1]+prices[i]) \ dp[i][k][1] = std::max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])$$ 最后返回的结果为: `dp[n-1][k-1][0]`;

```
class Solution {
public:
  int maxProfit(std::vector<int>& prices) {
    int                                             size = prices.size();
    int                                             k    = 2;
    std::vector<std::vector<std::vector<int>>> dp(
        size + 1,
        std::vector<std::vector<int>>(k + 1, std::vector<int>(2, 0)));

    for (int i = 0; i < size; i++) {
      for (int j = 1; j <= k; j++) {
        if (-1 == i - 1) {
          dp[i][1][1] = -prices[i];
          dp[i][1][0] = 0;
          dp[i][2][0] = 0;
          dp[i][2][1] = -prices[i];
```

```
            continue;
          }
          dp[i][j][0] = std::max(dp[i - 1][j][0], dp[i - 1][j][1] +
prices[i]);
          dp[i][j][1] =
              std::max(dp[i - 1][j][1], dp[i - 1][j - 1][0] - prices[i]);
        }
      }

      return dp[size - 1][k][0];
    }
};
```

最多交易`k`次

因为最多交易`k`次，所以动态规划数组`dp[n][k][rest]`中的`k`取值为$k = [0,1,2] , k为整数$， $k = 0$时相同的，`dp[n][0][rest] = 0`; 由此可以得出动态规划方程为： $$ dp[i][k][0] = std::max(dp[i-1][k][0], dp[i-1][k][1]+prices[i]) \ dp[i][k][1] = std::max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])$$ 最后返回的结果为: `dp[n-1][k-1][0]`;

```
class Solution {
public:
  int maxProfit(int k, std::vector<int>& prices) {
    int size = prices.size();
    if (size < 1 || k < 1) {
      return 0;
    }
    std::vector<std::vector<std::vector<int>>> dp(
        size + 1,
        std::vector<std::vector<int>>(k + 1, std::vector<int>(2, 0)));

    for (int j = 0; j <= k; j++) {
      dp[0][j][0] = 0;
      dp[0][j][1] = -prices[0];
    }

    for (int i = 1; i < size; i++) {
      for (int j = 1; j <= k; j++) {
        dp[i][j][0] = std::max(dp[i - 1][j][0], dp[i - 1][j][1] +
prices[i]);
        dp[i][j][1] =
            std::max(dp[i - 1][j][1], dp[i - 1][j - 1][0] - prices[i]);
      }
    }

    return dp[size - 1][k][0];
  }
};
```

带交易费用的无穷次交易

因为可以交易无穷次，所以动态规划数组`dp[n][k][rest]`中的`k`取值为$k = +\infty , k为整数$， $k与k-1$时相同的，对`k`区分无意义，动态规划数组`dp[n][k][rest]`中`k`的变化无意义，可以忽略; 则动态规划数组变为:`dp[i][rest]`. 由此可以得出动态规划方程为： $$ dp[i][k][0] = std::max(dp[i-1][k][0], dp[i-1][k][1]+prices[i]) \ dp[i][k][1] = std::max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i] - fee)$$ 其中`k`变化无意义, 方程简化为： $$ dp[i][0] = std::max(dp[i-1][0], dp[i-1][1]+prices[i]) \ dp[i-1][1] = std::max(dp[i-1][1], dp[i-1][0]-prices[i] - fee) $$

```
class Solution {
public:
  int maxProfit(std::vector<int>& prices, int fee) {
    int                              size = prices.size();
    std::vector<std::vector<int>> dp(size + 1, std::vector<int>(2, 0));

    for (int i = 0; i < size; i++) {
      if (-1 == i - 1) {
        dp[i][0] = 0;
        dp[i][1] = -prices[i] - fee;
        continue;
      }

      dp[i][0] = std::max(dp[i - 1][0], dp[i - 1][1] + prices[i]);
      dp[i][1] = std::max(dp[i - 1][1], dp[i - 1][0] - prices[i] - fee);
    }

    return dp[size - 1][0];
  }
};
```

股票买卖带冷却期

因为可以交易无穷次，所以动态规划数组`dp[n][k][rest]`中的`k`取值为$k = +\infty , k为整数$， $k与k-1$时相同的，对`k`区分无意义，动态规划数组`dp[n][k][rest]`中`k`的变化无意义，可以忽略; 则动态规划数组变为:`dp[i][rest]`. 由此可以得出动态规划方程为： $$ dp[i][k][0] = std::max(dp[i-1][k][0], dp[i-1][k][1]+prices[i]) \ dp[i][k][1] = std::max(dp[i-1][k][1], dp[i-2][k-1][0] - prices[i])$$ 其中`k`变化无意义, 方程简化为： $$ dp[i][0] = std::max(dp[i-1][0], dp[i-1][1]+prices[i]) \ dp[i-1][1] = std::max(dp[i-1][1], dp[i-2][0]-prices[i]) $$

```
class Solution {
public:
  int maxProfit(std::vector<int>& prices) {
    int                              size = prices.size();
    std::vector<std::vector<int>> dp(size + 1, std::vector<int>(2, 0));

    for (int i = 0; i < size; i++) {
      if (-1 == i - 1) {
        dp[i][0] = 0;
```

```
        dp[i][1] = -prices[i];

        continue;
      }

      if (0 == i - 1) {
        dp[i][0] = std::max(dp[i - 1][0], dp[i - 1][1] + prices[i]);
        dp[i][1] = std::max(-prices[0], -prices[i]);
        continue;
      }

      dp[i][0] = std::max(dp[i - 1][0], dp[i - 1][1] + prices[i]);
      dp[i][1] = std::max(dp[i - 1][1], dp[i - 2][0] - prices[i]);
    }

    return dp[size - 1][0];
  }
};
```