> 你和你的朋友面前有一排石头堆，用一个数组 piles 表示，piles[i] 表示第 i 堆石子有多少个。你们轮流拿石头，一次拿一堆，但是只能拿走最左边或者最右边的石头堆。所有石头被拿完后，谁拥有的石头多，谁获胜。
> 石头的堆数可以是任意正整数，石头的总数也可以是任意正整数，这样就能打破先手必胜的局面了。比如有三堆石头 piles = [1,100,3]，先手不管拿 1 还是 3，能够决定胜负的 100 都会被后手拿走，后手会获胜。
> 假设两人都很聪明，请你设计一个算法，返回先手和后手的最后得分（石头总数）之差。比如上面那个例子，先手能获得 4 分，后手会获得 100 分，你的算法应该返回 -96。

分析：

1. dp[i][j].fir 表示，对于 piles[i..j] 这部分石头堆，先手能获得的最高分数。
2. dp[i][j].sec 表示，对于 pilest[i..j]这部分石头堆，后手能获得的最高分数。

状态：开始的索引i,结束的索引j和当前轮到的人

```
dp[i][j][fir or sec]
其中:
0 <= i < piles.length
i <= j < piles.length
```

对每个状态有两种选择：选择最左边或最右边的一堆石头。

```
n = piles.size();
for 0 <= i < n {
    for j <= i < n {
        for who in {frc, sec} {
            dp[i][j][who] = std::max(left, right);
        }
    }
}
```

状态转移方程

```
// 对先手的最大值，为(选中左节点+余下nums[i+1...j]中后手的最大值，选中右节点
+nums[i...j-1]后手的最大值)
// 因为先手选过最左或最右节点，此后先手等待后手选择变为了后手
dp[i][j].first = std::max(nums[i]+dp[i+1][j].second , nums[j]+dp[i][j-
1].second);

if 选择了左边:
// 此时余下为nums[i+1...j]中最大值，此时后手变为了此区间内的先手
dp[i][j].second = dp[i+1][j].first
if 选择了右边
```

```
// 此时余下为nums[i...j-1]中最大值，此时后手变为此区间内的先手
dp[i][j].second = dp[i][j-1].first
```

base case:

```
// 当i == j 时，表示在先手面前只有一堆石子，那么先手最大值为nums[i],后手无法拿到石子，所以为0
dp[i][i].first = nums[i]
dp[i][i].second = 0
```

通过分析可以看出，对dp矩阵，计算的部分为上三角区域。 每次遍历均为矩形上半部分的对角线。 遍历框架为：

```
for (int l = 2 ;l <= n;l++){
    for(int i = 0;i<= n-l;i++){
        int j = i+l-1;
        // dp[i][j]即为上三角对角线
    }
}
```

算法实现为：

```
#include <vector>

// 返回先后手分值差
int stoneGame(std::vector<int> &piles) {
  int n = piles.size();
  if (n == 0) {
    return 0;
  }

  std::vector<std::vector<std::pair<int, int>>> dp =
      std::vector<std::vector<std::pair<int, int>>>(
          n + 1,
          std::vector<std::pair<int, int>>(n + 1, std::make_pair(0, 0)));

  // base case
  for (int i = 0; i < n; i++) {
    dp[i][0] = std::make_pair(piles[i], 0);
  }

  // 计算dp数组
  for (int i = 2; i <= n; i++) {
    for (int j = 0; j <= n - i; j++) {
      // 斜着遍历数组
      int l = i + j - 1;
      // 先手选择左边或右边的石子
```

```
        int left  = piles[i] + dp[i + 1][l].second;
        int right = piles[l] + dp[i][l - 1].second;

        // 套用状态转移方程
        if (left > right) {
          dp[i][l].first  = left;
          dp[i][l].second = dp[i + 1][l].first;
        } else {
          dp[i][l].first  = right;
          dp[i][l].second = dp[i][l - 1].first;
        }
      }

      auto res = dp[0][n - 1];

      return res.second - res.first;
    }
  }
```