**题目描述**

公元2919年，人类终于发现了一颗宜居星球——X星。
现想在X星一片连绵起伏的山脉间建一个天热蓄水库，如何选取水库边界，使蓄水量最大?

要求：

- 山脉用正整数数组s表示，每个元素代表山脉的高度。
- 选取山脉上两个点作为蓄水库的边界，则边界内的区域可以蓄水，蓄水量需排除山脉占用的空间
- 蓄水量的高度为两边界的最小值。
- 如果出现多个满足条件的边界，应选取距离最近的一组边界。

输出边界下标（从0开始）和最大蓄水量；如果无法蓄水，则返回0，此时不返回边界。
例如，当山脉为s=[3,1,2]时，则选取s[0]和s[2]作为水库边界，则蓄水量为1，此时输出：0 2:1
当山脉s=[3,2,1]时，不存在合理的边界，此时输出：0。

**输入描述**

一行正整数，用空格隔开，例如输入

```
1 2 3
```

表示s=[1,2,3]

**输出描述**

当存在合理的水库边界时，输出左边界、空格、右边界、英文冒号、蓄水量；例如

```
0 2:1
```

当不存在合理的书库边界时，输出0；例如

```
0
```

**备注**

- 1 ≤ length(s) ≤ 10000
- 0 ≤ s[i] ≤ 10000

**用例**

| 输入 | 1 9 6 2 5 4 9 3 7 |
| --- | --- |
| 输出 | 1 6:19 |
| 说明 | 经过分析，选取s[1]和s[6]，水库蓄水量为19（3+7+4+5） |

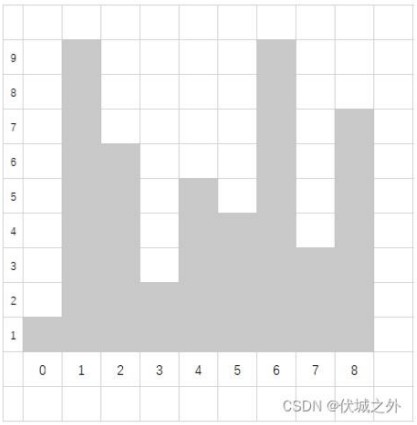
| 输入 | 1 8 6 2 5 4 8 3 7 |
| --- | --- |
| 输出 | 1 6:15 |
| 说明 | 经过分析，选取s[1]和s[8]时，水库蓄水量为15；同样选取s[1]和s[6]时，水库蓄水量也为15。由于后者下标距离小（为5），故应选取后者。 |

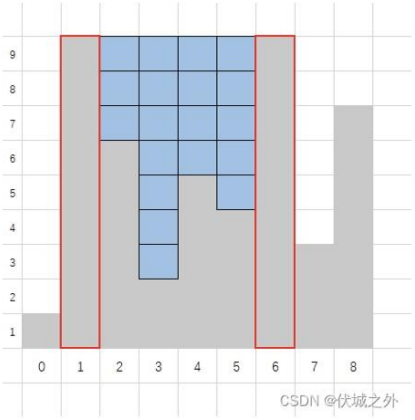
| 输入 | 1 2 3 |
| --- | --- |
| 输出 | 0 |
| 说明 | 不存在合理的水库边界 |

**题目解析**

**题目解析**

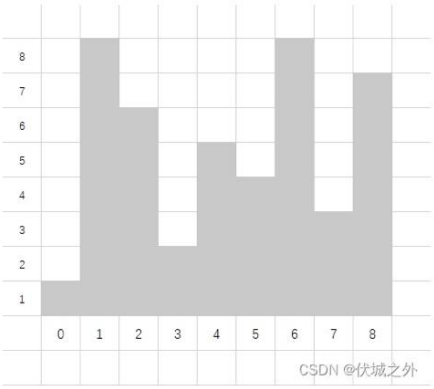
用例1图示




我们可以很明显看出选择山脉1和山脉6作为蓄水库边界时，产生的水量最大。

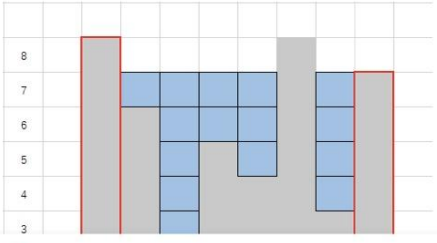
用例2图示



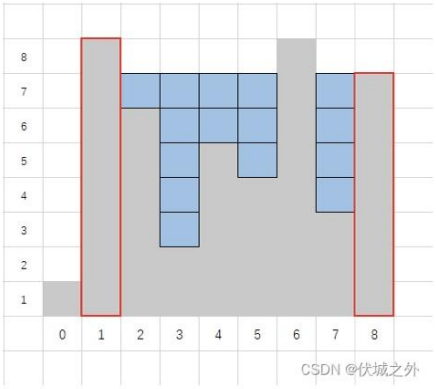

可以发现，选取山脉1和山脉8作为边界的水库蓄水量有15，而选取山脉1和山脉6作为边界的水库蓄水量也有15，题目要求：

> 如果出现多个满足条件的边界，应选取距离最近的一组边界

因此最终选择山脉1~山脉6作为边界

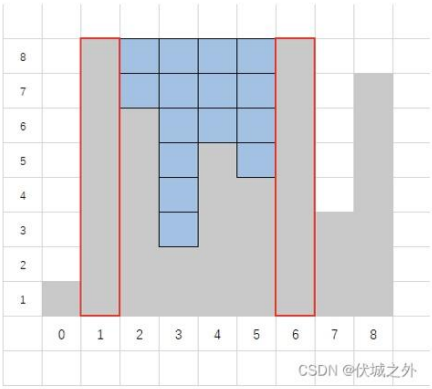


因此最终选择山脉1~山脉6作为边界

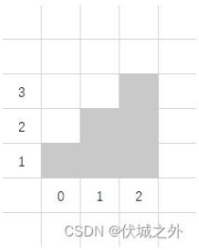



另外，选择山脉1~山脉8作为边界的水库高度选取的时较短的山脉8的高度，因为题目说：

> 蓄水量的高度为两边界的最小值。




用例3图示如下




可以发现，无法形成蓄水库。

本题，最简单的解题思路就是双重for，将任意两个山脉之间的蓄水量都计算出来，这样的话，时间复杂度是O(n^2)，1 ≤ length(s) = n ≤ 10000，也就是会有10^8次循环，另外计算蓄水量时，我们还需要遍历两个边界山脉之间的比较矮边界山脉更矮的山脉，这样的话，其实将整体时间复杂度推到了O(n^3)，这样毫无疑问的会超时。

本题非常类似于华为OD机试 - 太阳能板最大面积_伏城之外的博客-CSDN博客

上面这题的最优解题思路是使用 双指针，具体解题思路如下：

定义两个指针L、R，其中L指向数组首元素，R指向数组尾元素，然后比较h[L]和h[R]的大小：

- 如果h[L] > h[R]，则h[R]是较矮的，那么对高度h[R]来说，此时选取宽度R-L，可得最大面积，并且由于h[R]柱子已经找到的最大面积，而h[L]柱子还没有找到，因此我们应该R--
- 如果h[L] < h[R]，则h[L]是较矮的，那么对高度h[L]来说，此时选取宽度R-L，可得最大面积，并且由于h[L]柱子已经找到的最大面积，而h[R]柱子还没有找到，因此我们应该L++
- 如果h[L] == h[R]，两个柱子高度相同，则此时无论内移哪一根柱子都没关系，因此无论是L++，还是R--，取得的新面积都比不上当前的面积，因为无论内移哪一根柱子，新面积的矮柱都只会<= h[L] 和 h[R]，因此新面积的高度只会不变或更矮，宽度已经确定更短了，因此新面积只会比当前面积小。

本题可以借鉴上面的双指针思路，也可以定义两个指针L、R，分别指向数组h的首尾，然后比较h[L]和h[R]的大小，得出矮峰高度，计算矮峰对应的最大蓄水量。

但是本题多了一个条件：要得到矮峰对应的最大蓄水量的同时，保证两峰距离最小。即存在下面情况：

可以发现，无法形成蓄水库。

本题，最简单的解题思路就是双重for，将任意两个山脉之间的蓄水量都计算出来，这样的话，时间复杂度是O(n^2)，1 ≤ length(s) = n ≤ 10000，也就是会有10^8次循环，另外计算蓄水量时，我们还需要遍历两个边界山脉之间的比较矮边界山脉更矮的山脉，这样的话，其实将整体时间复杂度推到了O(n^3)，这样毫无疑问的会超时。

本题非常类似于华为OD机试 - 太阳能板最大面积_伏城之外的博客-CSDN博客

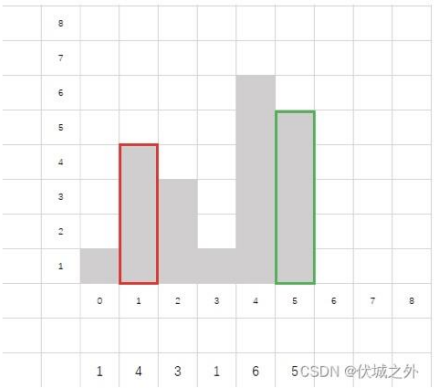
上面这题的最优解题思路是使用 双指针，具体解题思路如下：

定义两个指针L、R，其中L指向数组首元素，R指向数组尾元素，然后比较h[L]和h[R]的大小：

- 如果h[L] > h[R]，则h[R]是较矮的，那么对高度h[R]来说，此时选取宽度R-L，可得最大面积，并且由于h[R]柱子已经找到的最大面积，而h[L]柱子还没有找到，因此我们应该R--
- 如果h[L] < h[R]，则h[L]是较矮的，那么对高度h[L]来说，此时选取宽度R-L，可得最大面积，并且由于h[L]柱子已经找到的最大面积，而h[R]柱子还没有找到，因此我们应该L++
- 如果h[L] == h[R]，两个柱子高度相同，则此时无论内移哪一根柱子都没关系，因此无论是L++，还是R--，取得的新面积都比不上当前的面积，因为无论内移哪一根柱子，新面积的矮柱都只会<= h[L] 和 h[R]，因此新面积的高度只会不变或更矮，宽度已经确定更短了，因此新面积只会比当前面积小。

本题可以借鉴上面的双指针思路，也可以定义两个指针L、R，分别指向数组h的首尾，然后比较h[L]和h[R]的大小，得出矮峰高度，计算矮峰对应的最大蓄水量。

但是本题多了一个条件：要得到矮峰对应的最大蓄水量的同时，保证两峰距离最小。即存在下面情况：

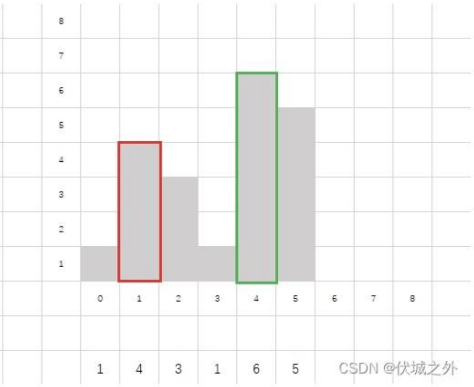



如果按照太阳能板那题的逻辑，

L = 1 , R = 5,

h[L] < h[R]，此时应该L++，但是在L++之前，我们应该思考一下此时对于矮峰h[1]而言，已经获得了最大蓄水量，但是此时两峰之间的距离是最短的吗？或者在缩小两峰距离的情况下，是否可以保持最大蓄水量呢？

答案是：可以的，如下图所示。

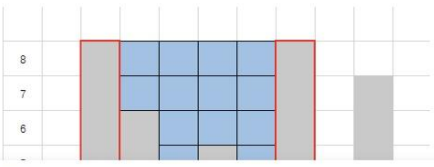



此时对于矮峰h[1]而言，最大蓄水量并没有发生变化，但是两峰距离更短了。

因此，虽然按照太阳能板最大面积那题的逻辑来看，求得最大蓄水量后，应该将矮柱指针内移一格，但是对于本题来说却并不适用。

此时，我们应该看看高峰内移一格会不会导致最大蓄水量减少，如果不会，则此时应该优先将高峰指针内移。

那么如果两峰高度相同呢？

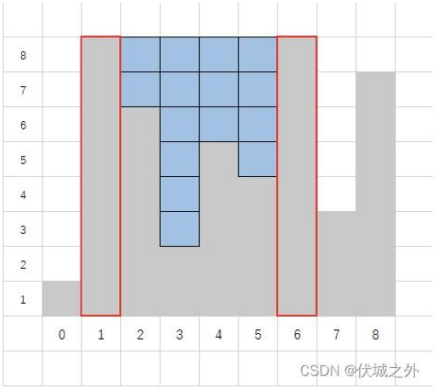


此时对于矮峰h[1]而言，最大蓄水量并没有发生变化，但是两峰距离更短了。

因此，虽然按照太阳能板最大面积那题的逻辑来看，求得最大蓄水量后，应该将矮柱指针内移一格，但是对于本题来说却并不适用。

此时，我们应该看看高峰内移一格会不会导致最大蓄水量减少，如果不会，则此时应该优先将高峰指针内移。

那么如果两峰高度相同呢？




此时，我们应该内移哪一个呢？

通过上面图示可以看出，此时我们应该L++，而不是R--，原因是L++后得到的最大蓄水量高于R--后的，更本质的原因是 h[L+1] > h[R-1]。

因此，可得逻辑如下：

```
if (h[l] < h[r]) { // h[L]是矮峰
  if (h[r - 1] >= h[r]) r--; // 此时我们不应该直接L++，而是需要检查h[R-1] 是否不低于 h[R]，如果不低于，则应该R--，因为
  else l++; // 否则L++
} else if (h[l] > h[r]) { // 原理同上
  if (h[l + 1] >= h[l]) l++;
  else r--;
} else { // 如果两峰高度相同
  if (h[l + 1] > h[r - 1]) l++;
  else r--;
}
```

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const h = line.split(" ").map(Number);

  console.log(getResult(h));
});

function getResult(h) {
  let ans = [0, 0, 0];

  let l = 0;
  let r = h.length - 1;
  let maxSum = 0;

  while (l < r) {
    let sum = 0;
    let lower = Math.min(h[l], h[r]);

    for (let i = l; i <= r; i++) {
      sum += Math.max(0, lower - h[i]);
    }

    if (sum >= maxSum) {
      maxSum = sum;
      ans = [l, r, sum];
    }

    if (h[l] < h[r]) {
      if (h[r - 1] >= h[r]) r--;
      else l++;
    } else if (h[l] > h[r]) {
      if (h[l + 1] >= h[l]) l++;
      else r--;
    } else {
      if (h[l + 1] > h[r - 1]) l++;
      else r--;
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const h = line.split(" ").map(Number);

  console.log(getResult(h));
});

function getResult(h) {
  let ans = [0, 0, 0];

  let l = 0;
  let r = h.length - 1;
  let maxSum = 0;

  while (l < r) {
    let sum = 0;
    let lower = Math.min(h[l], h[r]);

    for (let i = l; i <= r; i++) {
      sum += Math.max(0, lower - h[i]);
    }

    if (sum >= maxSum) {
      maxSum = sum;
      ans = [l, r, sum];
    }

    if (h[l] < h[r]) {
      if (h[r - 1] >= h[r]) r--;
      else l++;
    } else if (h[l] > h[r]) {
      if (h[l + 1] >= h[l]) l++;
      else r--;
    } else {
      if (h[l + 1] > h[r - 1]) l++;
      else r--;
    }
  }

  [l, r, sum] = ans;
  if (sum === 0) return 0;
  return `${l} ${r}:${sum}`;
}
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    Integer[] h =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(h));
  }

  public static String getResult(Integer[] h) {
    Integer[] ans = new Integer[] {0, 0, 0};

    int l = 0;
    int r = h.length - 1;
    int maxSum = 0;

    while (l < r) {
      int sum = 0;
      int lower = Math.min(h[l], h[r]);

      for (int i = l; i <= r; i++) {
        sum += Math.max(0, lower - h[i]);
      }

      if (sum >= maxSum) {
        ans = new Integer[] {l, r, sum};
        maxSum = sum;
      }

      if (h[l] < h[r]) {
        if (h[r - 1] >= h[r]) r--;
        else l++;
      } else if (h[l] > h[r]) {
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    Integer[] h =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(h));
  }

  public static String getResult(Integer[] h) {
    Integer[] ans = new Integer[] {0, 0, 0};

    int l = 0;
    int r = h.length - 1;
    int maxSum = 0;

    while (l < r) {
      int sum = 0;
      int lower = Math.min(h[l], h[r]);

      for (int i = l; i <= r; i++) {
        sum += Math.max(0, lower - h[i]);
      }

      if (sum >= maxSum) {
        ans = new Integer[] {l, r, sum};
        maxSum = sum;
      }

      if (h[l] < h[r]) {
        if (h[r - 1] >= h[r]) r--;
        else l++;
      } else if (h[l] > h[r]) {
        if (h[l + 1] >= h[l]) l++;
        else r--;
      } else {
        if (h[l + 1] > h[r - 1]) l++;
        else r--;
      }
    }

    if (ans[2] == 0) return "0";
    return ans[0] + " " + ans[1] + ":" + ans[2];
  }
}
```

Python算法源码

```python
# 输入获取
h = list(map(int, input().split()))


# 算法入口
def getResult(h):
    ans = [0, 0, 0]

    l = 0
    r = len(h) - 1
    maxSum = 0

    while l < r:
        sum = 0
        lower = min(h[l], h[r])

        for i in range(l, r + 1):
            sum += max(0, lower - h[i])

        if sum >= maxSum:
            maxSum = sum
            ans = [l, r, sum]

        if h[l] < h[r]:
            if h[r-1] >= h[r]:
                r -= 1
            else:
                l += 1
        elif h[l] > h[r]:
            if h[l+1] >= h[l]:
                l += 1
            else:
                r -= 1
        else:
            if h[l+1] > h[r-1]:
                l += 1
            else:
                r -= 1
```

Python算法源码

复制

```python
# 输入获取
h = list(map(int, input().split()))


# 算法入口
def getResult(h):
    ans = [0, 0, 0]

    l = 0
    r = len(h) - 1
    maxSum = 0

    while l < r:
        sum = 0
        lower = min(h[l], h[r])

        for i in range(l, r + 1):
            sum += max(0, lower - h[i])

        if sum >= maxSum:
            maxSum = sum
            ans = [l, r, sum]

        if h[l] < h[r]:
            if h[r-1] >= h[r]:
                r -= 1
            else:
                l += 1
        elif h[l] > h[r]:
            if h[l+1] >= h[l]:
                l += 1
            else:
                r -= 1
        else:
            if h[l+1] > h[r-1]:
                l += 1
            else:
                r -= 1

    ans_l, ans_r, ans_sum = ans

    if ans_sum == 0:
        print(0)
    else:
        print(f"{ans_l} {ans_r}:{ans_sum}")


# 算法调用
getResult(h)
```