题目描述

在学校中，N个小朋友站成一队， 第i个小朋友的身高为height[i]，

第i个小朋友可以看到的第一个比自己身高更高的小朋友j，那么j是i的好朋友(要求j > i)。

请重新生成一个列表，对应位置的输出是每个小朋友的好朋友位置，如果没有看到好朋友，请在该位置用0代替。

小朋友人数范围是 [0, 40000]。

输入描述

第一行输入N，N表示有N个小朋友

第二行输入N个小朋友的身高height[i]，都是整数

输出描述

输出N个小朋友的好朋友的位置

用例

输入	2 100 95
输出	0 0
说明	第一个小朋友身高100，站在队尾位置，向队首看，没有比他身高高的小朋友，所以输出第一个值为0。 第二个小朋友站在队首，前面也没有比他身高高的小朋友，所以输出第二个值为0。

输入	8 123 124 125 121 119 122 126 123
输出	1 2 6 5 5 6 0 0
说明	123的好朋友是1位置上的124 124的好朋友是2位置上的125 125的好朋友是6位置上的126 以此类推

题目解析

感觉很简单，直接双重for，逻辑如下

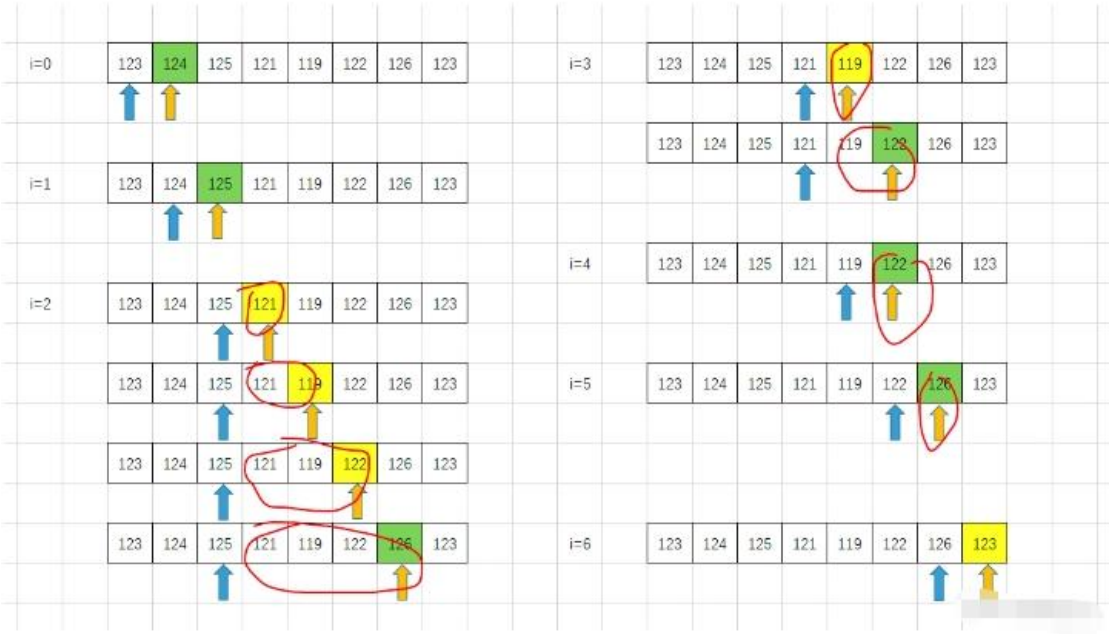
```
/* 记录前面出现的第一个比自己高的人的序号 */
function getHigherIndex(arr) {
  let higherIndex = arr.slice().fill(0);
  for (let i = 0; i < arr.length - 1; i++) {
    for (let j = i + 1; j < arr.length; j++) {
      if (arr[j] > arr[i]) {
        higherIndex[i] = j;
        break;
      }
    }
  }
  return higherIndex;
}
```

但是时间复杂度是O(n^2)。那么是不是有优化的可能呢?

下面图示中，我们蓝色的是i指针，橙色的是j指针，我么们可以发现j指针的扫描过程存在重复，比如：

- i=2时，j从121扫描到了126，将扫描过程中j指向的每一个数都和i指向的125进行了比较。
- i=3时，j又扫描了一遍119，122；
- i=4时，j又扫描了一遍122
- i=5时，j又扫描了一遍126

那么如何才能避免重复扫描呢?

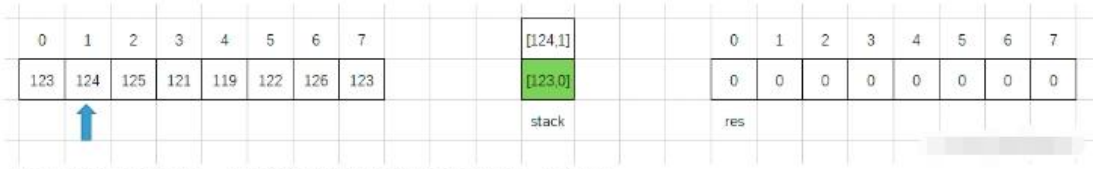


此时我们可以利用栈结构

for循环遍历输入数组的每一个元素，并将遍历出来的 [元素值,索引位置] 尝试压入stack栈中。

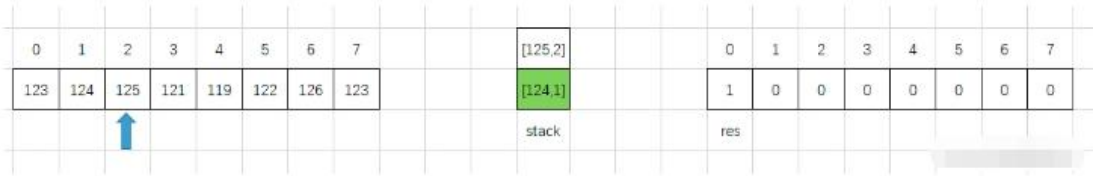
如果栈顶没有有元素，则直接压入。

如果栈顶有元素，比如下面图示情况，则比较遍历出来的元素 124 和 stack栈顶元素 123的大小，如果遍历元素值较大，

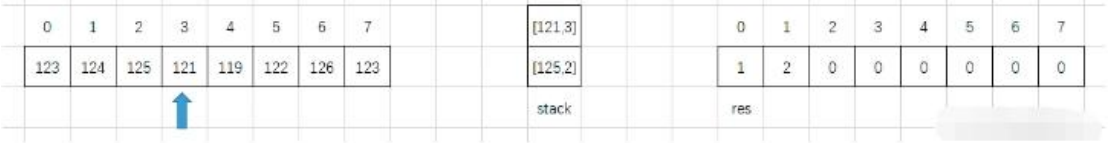


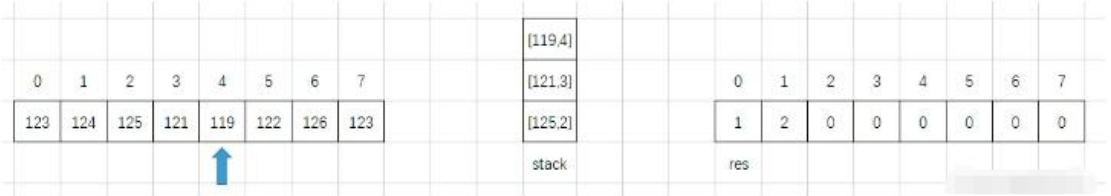
则将124的索引位置录入 res[123的索引位置],并将栈顶123弹栈后，压入124

下面操作同理



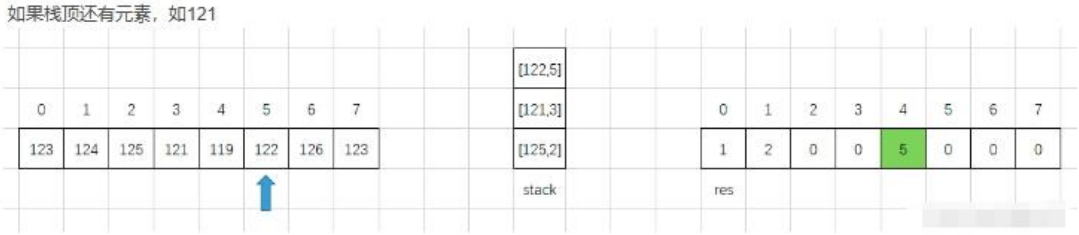
如果遍历的元素值小于栈顶的元素值，则继续下次遍历，如下面例子121小于125，则正常压入后，继续下次遍历



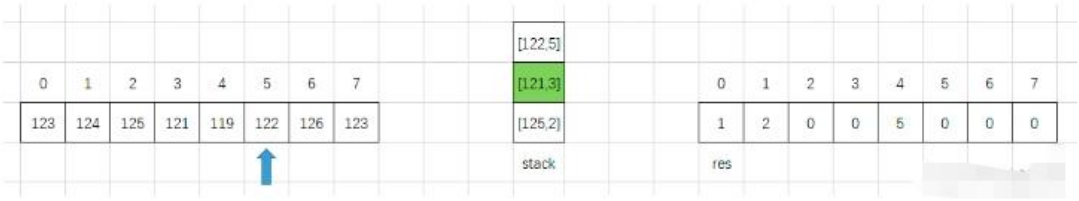


如果遍历的元素122比栈顶元素119大，则栈顶元素119弹栈

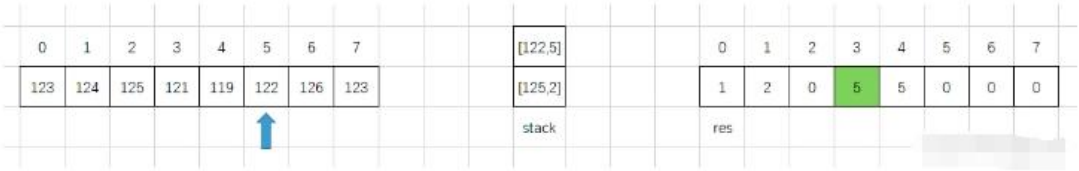
如果栈顶还有元素，如121

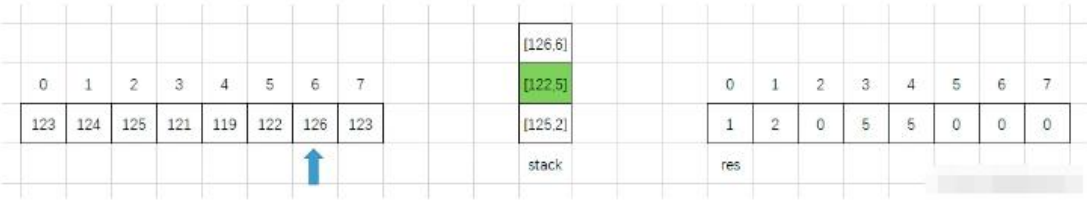


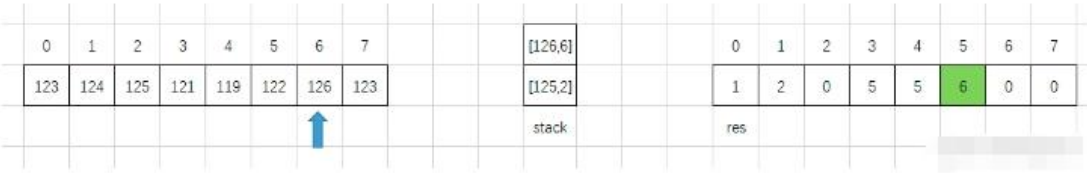
则此时遍历的元素122不执行压入，而是继续和栈顶元素121比较

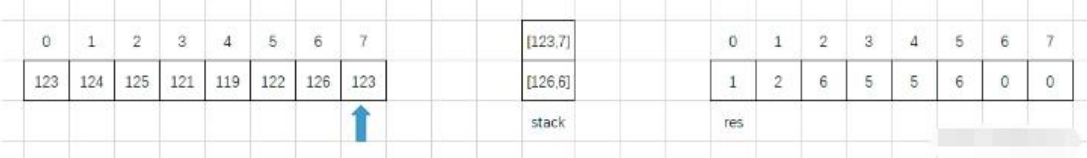


直到遍历的元素122遇到栈顶元素比自己大时，才执行压入









这种栈操作的好处，可以避免重复的j指针扫描。

Java算法源码

```java
import java.util.LinkedList;
import java.util.Scanner;
import java.util.StringJoiner;

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

    int n = sc.nextInt();

    int[] arr = new int[n];
    for (int i = 0; i < n; i++) arr[i] = sc.nextInt();

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

  public static String getResult(int[] arr) {
    LinkedList<int[]> stack = new LinkedList<>();

    int len = arr.length;
    int[] res = new int[len];

    for (int i = 0; i < len; i++) {
      int ele = arr[i];

      while (true) {
        if (stack.size() == 0) {
          stack.add(new int[] {ele, i});
          break;
        }

```

```
        int[] peek = stack.getLast();
        int peekEle = peek[0];
        int peekIndex = peek[1];

        if (ele > peekEle) {
          res[peekIndex] = i;
          stack.removeLast();
        } else {
          stack.add(new int[] {ele, i});
          break;
        }
      }
    }

    StringJoiner sj = new StringJoiner(" ");
    for (int v : res) {
      sj.add(v + "");
    }

    return sj.toString();
  }
}
```

JS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,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    let n = lines[0];
    let arr = lines[1]
      .split(" ")
      .slice(0, n)
      .map((ele) => parseInt(ele));

    console.log(getHigherIndex(arr).join(" "));

    lines.length = 0;
  }
});

```

```
/* 记录第一眼看到比自己高的的人的序号 */
function getHigherIndex(arr) {
  let stack = [];

  let len = arr.length;
  let res = new Array(len).fill(0);

  for (let i = 0; i < len; i++) {
    let ele = arr[i];
    let index = i;

    while (true) {
      if (stack.length === 0) {
        stack.push([ele, index]);
        break;
      }

      let [peekEle, peekIndex] = stack[stack.length - 1];

      if (ele > peekEle) {
        res[peekIndex] = index;
        stack.pop();
      } else {
        stack.push([ele, index]);
        break;
      }
    }
  }

  return res;
}
```

Python算法源码

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


# 算法入口
def getResult():
    stack = []

    res = [0]*(len(arr))

    for i in range(len(arr)):
        ele = arr[i]

        while True:
            if len(stack) == 0:
                stack.append([ele, i])
                break

            peekEle, peekIndex = stack[-1]

            if ele > peekEle:
                res[peekIndex] = i
                stack.pop()
            else:
                stack.append([ele, i])
                break

    return " ".join(map(str, res))


# 算法调用
print(getResult())
```