单调栈模板

使用逻辑技巧，保证每次新元素入栈，栈内元素保持有序。

> 给你一个数组，返回一个等长的数组，对应索引存储着下一个更大元素，如果没有更大的元素，就存-1。

分析： 题目可以抽象为一群人，站成一排，从前向后，找到第一个比当前元素高的人。

```
std::vector<int> nextGreaterElement(std::vector<int>& nums){
  std::vector<int> res =  std::vector<int>(nums.size());
  std::stack<int> s;

  // 倒序向栈中放入元素
  for(int i = nums.size() - 1; i >= 0;--i) {
    // 判断高低
    while(!s.empty() && s.top() <= nums[i]){
      // 比当前元素低，弹出
      s.pop();
    }

    res[i] = s.empty()?-1:s.top();
    s.push(nums[i]);
  }

  return res;
}
```

leetcode 496:

```
class Solution {
public:
  std::vector<int> nextGreaterElement(std::vector<int>& nums1,
                                      std::vector<int>& nums2) {
    std::vector<int>   res(nums1.size());
    std::map<int, int> map;
    std::stack<int>    s;

    for (int i = nums2.size() - 1; i >= 0; i--) {
      while (!s.empty() && s.top() <= nums2[i]) {
        s.pop();
      }

      map[nums2[i]] = s.empty() ? -1 : s.top();
      s.push(nums2[i]);
    }
```

```
    for (int i = 0; i < nums1.size(); ++i) {
      res[i] = map[nums1[i]];
    }

    return res;
  }
};
```

一月有多少天

> 给你一个数组T，这个数组存放的是近几天的天气气温，你返回一个等长的数组，计算：对于每一天，你还要至少等多少天才能等到一个更暖和的气温；如果等不到那一天，填 0。

调用单调栈模板：

```
std::vector<int> dailyTemperatures(std::vector<int>& T){
  std::vector<int> res(T.size());
  std::stack<int> s;

  for(int i = 0;i<T.size();i++) {
    // 存储索引，保证当前栈顶一定比当前元素高
    while(!s.empty() && T[s.top()] <= T[i]) {
      s.pop();
    }
    res[i] = s.empty() ? 0:s.top()-i;
    s.push(i);
  }

  return res;
}
```

如何处理环形数组

leetcode 503

对于环形数组，一般用`i % n`来就计算当前索引； 所以可以将数组抽象成两个数组相连。

```
class Solution {
public:
  std::vector<int> nextGreaterElements(std::vector<int>& nums) {
    int              n = nums.size();
    std::vector<int> res(n);
    std::stack<int>  s;

    for (int i = 2 * n - 1; i >= 0; i--) {
      while (!s.empty() && s.top() <= nums[i % n]) {
        s.pop();
```

```
      }

      res[i % n] = s.empty() ? -1 : s.top();

      s.push(nums[i % n]);
    }

    return res;
  }
};
```

最大子矩阵

```
#include <iostream>
#include <stack>
#include <vector>

class Solution {
public:
  int maxRecSize(std::vector<std::vector<int>> &board) {
    std::vector<int> height = std::vector<int>(
        board[0].size(),
        0);  // 构建一个从低向上看的高度数组，如果底为0，在高度为0
    for (int i = 0; i < board.size(); i++) {
      for (int j = 0; j < board[i].size(); j++) {
        height[j] == (board[i][j] == 0 ? 0 : height[j] + 1);
      }
    }

    return maxArea(height);
  }

private:
  int maxArea(std::vector<int> &height) {
    std::stack<int> maxIndex;  // 存放从高到底的高度索引
    int             maxArea = 0;
    int             len     = height.size();
    for (int i = 0; i < len; i++) {
      while (maxIndex.size() && height[i] <= height[maxIndex.top()]) {
        // 如果当前元素小于等于栈顶元素，弹出栈顶元素，计算栈顶元素扩展的面积
        int j = maxIndex.top();
        maxIndex.pop();
        int k   = maxIndex.empty() ? -1 : maxIndex.top();
        maxArea = std::max(maxArea,
                           (i - k - 1) * height[j]);  // 面积为(i-k-
1)*height[j]
      }
      maxIndex.push(i);
    }

    // 如果结束后栈非空
```

```
    while (maxIndex.size()) {
      int top = maxIndex.top();
      maxIndex.pop();
      int k = maxIndex.empty() ? -1 : maxIndex.top();
      // 对栈中存留的元素，其扩展方向一定到数组尾
      maxArea = std::max(maxArea, (len - k - 1) * height[top]);
    }

    return maxArea;
  }
};
```