特殊的数据结构

1、单调栈：

本质:就是一个栈只是给栈增加了一个逻辑使得**每次新元素入栈后，栈内的元素都保持单调(单调递增或单调递减)**

用途：并不广泛,用来解决“Next Greater Element”

> **题目:输入一个数组，返回一个等长的数组，对应索引存储着下一个更大的元素，如果没有更大的元素，就存-1。**
> **比如:输入一个数组nums=[2,1,2,4,3] 返回[4,2,4,-1,-1]**
> **解释：第一个2后面比2大的数是4;1后面比1大的数是2;.....**

```
vector<int> nextGreaterElement(Vector<int>&nums){
 vector<int> ans(nums.size());
 stack<int> s;
 for(int i=nums.size()-1;i>=0;i--){//倒着入栈则正着出栈
 while(!s.isEmpty()&&s.top()<=nums[i]){//判定个子高矮
 s.pop();//正着出栈
 }//此时栈顶元素为最大且最靠近nums[i]的元素
 ans[i]=s.isEmpty?-1:s.top();
 s.push(nums[i]);
  }
 }
```

分析时间复杂度：不要因为两层循环就认为其时间复杂度为n^2实际上只有n，因为每个元素最多只能push一次和pop一次

思考？**思考一下栈中从栈底到栈顶单调递增怎么处理？单调递减怎么处理？**

```
前者：
while(!s.isEmpty()&&s.top>=T[i]){
 s.pop();
}
后者：
while(!s.isEmpty()&&s.top<=T[i]){
 s.pop();
}
```

例题：

> 现在给你一个数组,这个数组里存放的近几天的气温，算法需返回一个数组，对于每一天，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

```
vector<int> dailyTemperature(vector<int>& T){
 vector<int> ans;
 stack<int> stack;//此时放的是元素索引而不是元素,是一个单调栈后入的值要比前面的值小即栈底往栈顶为单调递增
```

```
for(int i=T.size()-1;i>=0;i--){
while(!stack.isEmpty()&&T[stack.top]<=T[i]){//倒着入栈就会正着出栈
stack.pop();//栈顶值比现在的元素要小就会弹出
}
//记录值
ans[i]=stack.isEmpty()?0:(stack.top()-i);
 stack.push(i);
 }
}
```

2:如何处理循环数组

由于计算机的内存都是线性的没有真正意义上的循环数组，因此**采用的%运算符求模**

```
int n=arr.length();
int index=0;
index=(index+1)&n;
```

假设以上的单调栈问题中的数组是一个循环数组那么该如何解决?

假设我们的数组是原来的数组两倍，真正定位的时候通过%运算符求模

```
vector<int> nextGreaterElementes(vector<int>& T){
 vector<int> ans;
 stack<int> stack;
 int n=T.size();
 //假设我们的数组是原来的数组两倍即用一个同样大小的数组接在后面
 for(int i=T.size()*2-1;i>=0;i--){
 while(!stack.isEmpty()&&stack.top()<=T[i%n]){
 stack.pop();
 }
  ans.push_back(stack.isEmpty()?0:stack.top());
  stack.push(T[i%n])
  }
  return ans;
}
```

2、单调队列:

单调队列就是在队列的基础上使用一些技巧使得队列中的元素满足**单调递增或单调递减**的性质。

例题

> 输入一个**数组nums**和一个**正整数k**，有一个大小为**k的窗口在nums**上从左至右滑动，请输出每次滑动时窗口中的最大值

已知一个结论：

在一个数堆A中，且已知其中的最大值如果增加一个数可以很快的算出最大值，如果减少一个数则可能需要重新遍历该数堆获得最大值。

```
单调队列的数据结构：
class MonotonicQueue{
 //在队尾添加元素n,但是要把前面比自己小的元素都删掉
 void push(int n);
 //删除队头元素
 void pop();
 //返回当前队列的最大值
 int max();
 }

 实现方法
 void push(int n){
 while(!q.isEmpty()&&q.top()<=n){
 q.pop();
 }
 q.push(n);
 }
 int max(){
 return q.top();
 }
 void pop(int n){
 //移除元素n
 if(q.top()==n){
 q.pop();
 }
 }
```