单调栈

栈内元素呈现单调性的栈。

问题背景之下。

对于入栈的元素，关注和前第一个小于等于它的元素（或大于等于它的元素）我们找出这样每一个元素这样的关系，利用这个资源来解决问题。

理解

理解过程即可。

这个过程的详细证明，也是最开始的情况进行数学归纳法得到的结果。

生长思考

[COCI2010-2011#3] DIFERENCIJA.md

https://www.luogu.com.cn/problem/P5788

【模板】单调栈

题目背景

模板题，无背景。

2019.12.12 更新数据，放宽时限，现在不再卡常了。

题目描述

给出项数为 n 的整数数列 $a_{1\dots n}$。

定义函数 $f(i)$ 代表数列中第 i 个元素之后第一个大于 a_i 的元素的**下标**，即 $f(i)=\min_{i<j\leq n,a_j>a_i}\{j\}$。若不存在，则 $f(i)=0$。

试求出 $f(1\dots n)$。

输入格式

第一行一个正整数 n。

第二行 n 个正整数 $a_{1\dots n}$。

输出格式

一行 n 个整数 $f(1\dots n)$ 的值。

样例 #1

样例输入 #1

```
5
1 4 2 3 5
```

样例输出 #1

```
2 5 4 5 0
```

提示

【数据规模与约定】

对于 30% 的数据，$n \leq 100$;

对于 60% 的数据，$n \leq 5 \times 10^3$;

对于 100% 的数据，$1 \leq n \leq 3 \times 10^6$，$1 \leq a_i \leq 10^9$。

---

solve

和一般的暴力做法相比

利用单调栈:

一直维护，如果出现了一个比栈顶高的元素，就进行更新。如果某一个更新失败了。更底部的元素就不需要再进行更新了。这样就大大的降低了复杂度。

单调栈本质上就维护了这样一个关系。很多问题的解决，需要利用这种关系信息。比方说最大正方形等等。

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 3E6 + 10;

int a[N];
int ans[N];
int stk[N], top = 0;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n;
    cin >> n;

    for ( int i = 1; i <= n ; i++ ) {
        cin >> a[i];
        while (top != 0 && a[stk[top]] < a[i]) {
            ans[stk[top]] = i;
            top--;
```

```
        }
        stk[++top] = i;
    }
    for (int i = 1 ; i <= n  ; i++) {
        cout << ans[i] << ' ';
    }
    cout << '\n';

}

/* stuff you should look for
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/
```

https://www.luogu.com.cn/problem/P6503

introduce

一道dp思想，单调栈优化的问题。

单调栈在这里的优化是，快速的计算出dp转移中要使用的关键的量。可能所有的dp优化都是从这一个角度做起的。

思路设计，思路历程

其一：首先设计状态。

我们总是尝试把区间扫一遍，在此之间，我们总是可以关注什么子问题呢?

我们关注以 i为左边界的区间。然后考虑这一些区间的总贡献。

对答案的贡献为$\sum f(i)-g(i)$。设计f_i为所有区间的最大值之和。g_i为所有的最小值之和。

于是我们发现 $f_i=f_{p_i}+(i-p_i)*a_i$其中p_i为大于a_i的第一个位置。

这个我们可以利用单调栈来快速找出。

同理我们可以设计g_i的计算方法。

面对解是一堆区间的。所有区间的计算。我们考虑区间与区间的关系。

总是由于它们的交集，包含关系导致了状态的迁移。

其中区间结构的关注角度为最值。可以分类出一些区间共享最大值。

总而言之，这是一道去感受区间结构以及单调栈优化的非常好的问题。

生长思考：

- 数组，区间资源上dp的一种非常经典，天然的设计角度。
- 单调栈总结
  - 从哪个方向构造意味着维护的第一个最值在哪个方向
  - 递减栈维护最大值。递增栈维护最小值。
    - 某一个状态下的相邻的两个栈元素。如果是递减栈，那么就在原数组中夹在这两个元素之间的是，小于或等于个元素同时满足的项。
    - 如果是递增栈，省略的就是大于等于两个元素的项。
  - 很多时候，当我们维护一个单调栈时候，不妨去问，我为什么要关注第一个大于或者小它的元素？

    [COCI2010-2011#3] DIFERENCIJA.md 这里我们不断维护的过程中，我们关注前面是否有等于当前b_i长度的剪刀，其实就是一种维护第一个大于等于它的结果。看是否为当前b_i的大小如果是就可以节省一把剪刀。

    寻找连续矩阵中的最大值问题中：我们建立一个一个以当前a_i为高度的矩形。然后从左从右找第一个比它矮的。显然这样成立了。正方形为什么不更低一点？更低就可以转化成其它点上高度的问题。我们枚举的最优正方形都应该贴着一个矩形块上走。

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 1E6 + 10;
typedef pair<int, int> iip;

int Max[N], top1, Min[N], top2;
ll a[N];

ll f[N], g[N];


ll ans = 0;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n;
    cin >> n;
    for ( int  i = 1; i <= n; i++ ) {
        cin >> a[i];
        //怎么维护？
        while (top1 && a[i] >= a[Max[top1]])top1--;
        while (top2 && a[i] <= a[Min[top2]])top2--;
        f[i] = f[Max[top1]] + (i - Max[top1]) * a[i];
        g[i] = g[Min[top2]] + (i - Min[top2]) * a[i];
        ans += (f[i] - g[i]);
        Max[++top1] = i;
        Min[++top2] = i;
    }
    cout << ans << '\n';
```

```
}

/* stuff you should look for
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/
```

https://codeforces.com/problemset/problem/1779/D

算法描述：

从高往低减少，争取让一把剪刀减取足够多的头发。

对于每一根头发我们都必然会匹配到一把剪刀。

所以我们用一个单调栈往下搜，在这个过程中保证单调性。一旦有新的元素（不和栈中的一个某一个元素相同，就使用一把剪刀）这样一直进行。

维护剪刀的使用情况，最终就可以判断是否能够实现理想长度的头发。

注意：

- 头发比目标长度短。不可能实现。
- 头发和目标一样短，省一把剪刀。

算法理解

首先将第一个元素入栈；如果当前$a_i > b_i$使用一把剪刀。使用了一把剪刀之后就入栈。否则不需要。

如果遇到比top大的，就可以全部都放出来了，因为我们保留栈是希望用当前长度的剪刀，去使得后面遇到的同要使用同一种梳子的头发也使用保留的剪刀从而实现节省一把剪刀。

如果遇到了很大的b_i就失效了。所以此时就重新调整单调栈。要保证中间没有比剪刀长度更高的头发才能重复利用。

生长思考：

怎么有一个单调栈的意识？ 扫描一遍数组。

单调栈是一种很天然的维护方法。还得慢慢体会。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 2e5 + 10;
int a[maxn], b[maxn], x[maxn];

void solve()
```

```
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 1; i <= n; i++)
        cin >> b[i];
    int m;
    cin >> m;
    map<int, int> rec;
    for (int i = 1; i <= m; i++)
    {
        cin >> x[i];
        rec[x[i]]++;
    }
    stack<int> stk;
    for (int i = 1; i <= n; i++)
    {
        if (b[i] > a[i])
        {
            cout << "NO\n";
            return;
        }
        // 如果已经向等了那么就不用减。所以这里有一个特判断点。
        while (!stk.empty() && stk.top() < b[i])
            stk.pop();
        if (a[i] != b[i])
        {
            if (stk.empty() || stk.top() != b[i])
                stk.push(b[i]), rec[b[i]]--;
        }
        if (rec[b[i]] < 0)
        {

            cout << "NO\n";
            return;
        }
    }
    cout << "YES\n";
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int t;
    cin >> t;
    while (t--)
        solve();
}
```

一个明智地追求快乐的人，除了培养生活赖以支撑的主要兴趣之外，总得设法培养其他许多闲情逸致。