### 粗略总结

- 这样下去区域赛等死吧。简单题都不快，思路不简洁。

## $B$. 走廊的灯

ad

> 对于一个字符串。（子串中只包含0，1，2）
> 寻找不含1或者不含0的最大连续子串的长度。

- 妈的当时搞了前缀后缀数组。然后双指针，出了一堆的问题。
- 算法不清晰就开始匆忙实现。

**最后的解决方法：**

> 两次处理，两次双指针。

- 当自己觉得算法很不清晰，分类讨论很多，实现难度比较大的时候，三思而后行啊。

```
const int maxn = 2e5 + 10;
int a[maxn];
void solve()
{
    int n;
    cin >> n;
    string s;
    cin >> s;
    for (int i = 0; i < n; i++)
        if (s[i] == '0')
            a[i] = 0;
        else
            a[i] = 1;
    int l = 0, r = 0;
    int ans = 1;
    a[n] = 3;
    while (r <= n)
    {
        if (a[r] == a[l])
            r++;
        else
            ans = max(ans, r - l), l = r;
    }
    for (int i = 0; i < n; i++)
        if (s[i] == '1')
            a[i] = 1;
        else
            a[i] = 0;
    l = 0, r = 0;
    a[n] = 3;
```

```
    while (r <= n)
    {
        if (a[r] == a[l])
            r++;
        else
            ans = max(ans, r - l), l = r;
    }
    cout << ans << '\n';
}
```

---

## $C$. 输出练习

ad

> 就是输出，关于某一段区间中，可以表示为k的非负数整数倍的数。

**情况**

- 赛时，一堆特判没有发现。
- 溢出这种，原来已经想过的问题，琢磨了很久。
- 具体情况的模拟不够充分。

$solve$

- $0, 1$要进行特判。
- 对于其它普通的数字。
  - 首先关注——溢出边界：$flag = \frac{(2^{63}-1)}{k}$
  - 关注$L, R$
  - 几个的相对位置有如下情况：
    - $L, R, flag$
    - $L, flag, R$
    - $flag, L, R$
  - 慢慢变大的过程中
    - 第一种，注意如果完全跳出范围就没有。
    - 如果超出了flag，要进行特判。是否在区间范围之内。
    - 本质上，就是一个不断枚举数字的过程，要把握住当前的情况，小心溢出。
- 生长思考：
  - 技巧总结，总会遇到的子问题：第一关于一些比较大的边界数字怎么表示？
    - 整数$2^{63}$
      - 用一个十六进制表示，由于一个符号就表示了$4$个二进制比特位。
      - 移位运算符的方法。（注意必须选取合适）
        - $fflag = 1ULL << 63ULL;$

---

*code*

```
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long ll;
const int maxn = 2e5 + 10;

ll fflag = 0;

void solve()
{
    ll l, r, k;
    cin >> l >> r >> k;

    ll temp = 1;
    if (k == 1)
    {
        if (l <= 1 && r >= 1)
            cout << 1 << '\n';
        else
            cout << "None." << '\n';
        return;
    }
    if (k == 0)
    {
        if (l == 0 && r == 0)
        {
            cout << 0 << '\n';
            return;
        }
        else if (l == 0 && r >= 1)
        {
            cout << 0 << ' ' << 1 << '\n';
        }
        else if (l == 1)
            cout << 1 << '\n';
        else
            cout << "None." << '\n';
        return;
    }
    ll flag = (fflag - 1) / k;
    while (temp < l && temp <= flag) //当前的情况。
        temp *= k;
    if (temp > r)
    {
        cout << "None." << '\n';
        return;
    }
    if (temp > flag && temp < l)
    {
        cout << "None." << '\n';
        return;
    }
    while (temp <= r && temp <= flag)
    {
```

```
        cout << temp << ' ';
        temp *= k;
    }
    if (temp > flag && temp <= r) // temp这里再次乘法再次乘法的时候，就会出错。
        cout << temp;
    cout << '\n';
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int t;
    cin >> t;
    fflag = 0x8000000000000000;
    while (t--)
        solve();
}
```