Problem with Random Tests

chenjiuri_problemwithrandomtests

> 给定一个零1字符串，
> 寻找两个子串进行或运算：
> 是试求出最大的最大的运算结果
> 用二进制表示：

20mins

- 发现，其中有一串，必然是整一串s。
  - 这里减少了其中一串的枚举。
  - 第二个点就是，我们的选择必然是一个前缀。我们发现将任何一个选择拓展到前缀之后，对我们的结果影响都不会变坏。
- 然后滚去看题解。

题解

- 第一点：
  - 就是其中有一个子串必然是本身。
- 第二点，必须可以保证第一个1之后的第一个0被或运算成1.（前提是不全为0或者不全为1）.
  - 这样就可以减少枚举域了。
  - 记p_1, p_2分别表示第一个1和第一个0的位置。
    - 那么第一个0可以被p1----(p2-1)中的1亦或成1.
    - 所以只需要枚举这一部分，然后逐渐匹配即可。看上去大小应该是$O(n^2)$
    - 但是由于题目中强调了，随机的产生数据，所以非常变态的连续20个1的这种数据概率非常小，也就是没有这一种数据。
  - 综上直接暴力枚举即可解决。

一份非常简洁地代码

```
#include <bits/stdc++.h>
using namespace std;

void MAIN();
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    MAIN();
}
typedef long long ll;
const int maxn = 2e5 + 10;
//------code-----٩(´ω`*)و ------靓仔代码-----٩(´ω`*)و ----talk is cheap , show me
the code--------

void MAIN()
{
    int n;
```

```
    cin >> n;
    string s;
    cin >> s;
    int p1 = s.find('1');
    if (p1 == -1)
    {
        cout << 0 << '\n';
        return;
    }
    //寻找第一个1，寻找1之后的第一个0；
    //确定位置之后就进行匹配。
    string ans = s; //这里用到几个有意思的函数。
    int p2 = s.find('0', p1);
    for (int i = p1; i < p2; i++)
    {
        string t = s;
        for (int j = 0; p2 + j < s.size(); j++)
            t[p2 + j] |= s[i + j]; //这里很巧妙地应用了一个位运算。要对两者有一定地了解。
        ans = max(ans, t);
    }
    cout << ans.substr(ans.find('1')) << '\n';
}
```

生长思考

- 第一点，string的使用
  - 如果一些函数的功能靠自己来实现，代码量会变大不少。
- 位运算：
  - 这里的关于字符的位运算，结合了自己的思考。从上面来看，
  - $char(1)|char(0)=char(1), char(1)|char(1)=char(1), char(0)|char(0)=char(0)$
