交互题

在输出后面清空一下缓存即可

`fflush(stdout);`

```
// 交互题板。
// 判断一个在[2,100]的未知数是否为质数，提问会回答你是否是因子，最多20次机会。
void solve() {
    vector<bool> vis(102);
    int cnt = 0;
    string s;
    fr(i, 2, 50){
        if(vis[i]) continue;
        else{
            if(i * i <= 100){
                cout << i*i <<"\n";
                fflush(stdout);
                cin >> s;
                if(s == "yes")  cnt++;
            }
            cout << i << "\n";
            fflush(stdout);
            cin >> s;
            if(s == "yes")  cnt++;
            int x = i;
            while( x <= 100){
                vis[x] = true;
                x += i;
            }
        }
    }
    if(cnt >= 2){
        cout << "composite\n";
    }else{
        cout << "prime\n";
    }
    fflush(stdout);
}
```

4.维护区间的最小值，及最小值的数量

```
struct info{
    int cnt, minn;
};
struct node{
    int lazy, len;
    info val;
} seg[N << 2];
info operator+(const info &a, const info &b){
    info c;
    c.minn = min(a.minn, b.minn);
    c.cnt = 0;
    a.minn == c.minn ? c.cnt += a.cnt : c.cnt;
    b.minn == c.minn ? c.cnt += b.cnt : c.cnt;
    return c;
}
void settag(int id, int tag){
    seg[id].val.minn += tag;
    seg[id].lazy += tag;
}
void up(int id){
    seg[id].val = seg[id << 1].val + seg[id << 1 | 1].val;
}
void down(int id)
{
    if (seg[id].lazy == 0)
        return;
    settag(id << 1, seg[id].lazy);
    settag(id << 1 | 1, seg[id].lazy);
    seg[id].lazy = 0;
}
void build(int id, int l, int r){
    seg[id].len = r - l + 1;
    if(l == r){
        seg[id].val.minn = 0;
        seg[id].val.cnt = 1;
        seg[id].lazy = 0;
        return;
    }
    int mid = (l + r) >> 1;
    build(id << 1, l, mid);
```

```
    build(id << 1 | 1, mid + 1, r);
    up(id);
}
void modify(int id, int l, int r, int ql, int qr, int val){
    if(ql <= l && r <= qr){
        settag(id, val);
        return;
    }
    down(id);
    int mid = (l + r) >> 1;
    if(qr <= mid)
        modify(id << 1, l, mid, ql, qr, val);
    else if(ql > mid)
        modify(id << 1 | 1, mid + 1, r, ql, qr, val);
    else{
        modify(id << 1, l, mid, ql, qr, val);
        modify(id << 1 | 1, mid + 1, r, ql, qr, val);
    }
    up(id);
}
info query(int id, int l, int r, int ql, int qr)
{
    if (ql <= l && r <= qr)
    {
        return seg[id].val;
    }
    down(id);
    int mid = (l + r) >> 1;
    if (qr <= mid)
        return query(id << 1, l, mid, ql, qr);
    else if (ql > mid)
        return query(id << 1 | 1, mid + 1, r, ql, qr);
    else
        return query(id << 1, l, mid, ql, qr) + query(id << 1 | 1,
mid + 1, r, ql, qr);
}
void solve() {
    int n;  cin >> n;
    build(1, 1, n);
    int t;  cin >> t;
    while(t--){
        char q; cin >> q;
        if(q == 'q'){
```

```
            int l, r;   cin >> l >> r;
            info ans = query(1, 1, n, l, r);
            cout << ans.minn <<" "<< ans.cnt;
        }else{
            int l, r, add;  cin >> l >> r >> add;
            modify(1, 1, n, l, r, add);
        }
    }
}
```

线性基

重要性质

1、原序列任何数都可以通过线性基异或得到。

2、线性基内部任何数异或都不为0。

3、线性基个数唯一且最少。

插入

```
inline void insert(ll x){
    for(int i = 52; i >= 0; i--){
        if(x >> i & 1){
            if(!p[i]){
                p[i] = x;
                break;
            }
            x ^= p[i];
        }
    }
}
```

求Max

```
ll getmx(){
    ll ans = 0;
    for(int i = 52; i >= 0; i--)
        if((ans ^ p[i]) > ans)  ans ^= p[i];
    return ans;
}
```

求k小值

```
inline void prework() { //预处理线性基p[i]，如果p[i]二进制第j位为1则异或p[j - 1]
    for(int i=1;i<=60;++i)
        for(int j=1;j<=i;++j)
            if(p[i]&(1LL<<j-1)) p[i]^=p[j-1];
}
inline ll getkth(int k) {//第k小则是ans异或线性基中每一位为1的元素
    if(k==1&&size<n) return 0;
    if(size<n) --k;
    //n表示序列长度，size表示线性基内部元素个数
    prework();
    ll ans=0;
    for(int i=0;i<=60;++i)
        if(p[i]) {
            if(k&1) ans^=p[i];
            k>>=1;
        }
    return ans;
}
```

最长上升子序列问题(LIS)

策略：二分查找最大长度的最大值

```
    int n;  cin >> n;
    vc<int>a(n + 2 ), g(n + 2, INF),dp(n + 2);
    // dp[i]代表以i结尾的最长上升子序列长度
    fr(i, 1, n){
        cin >> a[i];
        int k = lower_bound(g.begin() + 1, g.begin() + n + 1, a[i])
- g.begin();
        dp[i] = k;
        g[k] = a[i];
    }
```

最长公共子序列(LCS)

dp[i] [j] 代表a[1] - a[i] 和b[1] - b[j] 的LCS长度

dp[i] [j] = max(dp[i - 1] [j], dp[i] [j - 1]);

当a[i] = b[j] 时

dp[i] [j] = dp[i - 1] [j - 1] + 1

```
fr(i, 1, n)
    fr(j, 1, m){
        dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        if(a[i]==b[j])
        dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
    }
```

变种

两个数组都是排列

A数组映射到B数组，然后转换为LIS 问题。

矩阵连乘问题(MCM)