E - Complicated GCD

যেকোনো দুইটি ক্রমিক স্বাভাবিক সংখ্যার gcd অবশ্যই 1. অর্থাৎ gcd(x, x+1) = 1. তাই L এবং R যদি সমান না হয় তাহলে অবশ্যই answer হবে 1. আর L, R সমান হলে তো সেটাই answer. আর বড় সংখ্যাকে আমরা string এর মাধ্যমে খুব সহজেই ইনপুট নিতে পারি।

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

int main()
{
    ios_base::sync_with_stdio(0); cin.tie(0);
    string a, b;
    cin >> a >> b;
    if(a == b)
        cout << a << '\n';
    else
        cout << 1 << '\n';


    return 0;
}
```

F - Zigzag Parity

1, n, 2, n - 1, 3, n - 2 এই সিকুয়েন্স এ যদি দেখি তাহলে যোগফল হবে যথাক্রমে (n + 1), (n + 2), (n + 1), (n + 2), ... অর্থাৎ একটা জোড় হলে পরেরটা বিজোড়, তারপর আবার জোড়।

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

void solve()
{
    int n;
    cin >> n;
    for(int i = 1, j = n; i <= j; i++, j--)
    {
        if(i == j)
            cout << i << " ";
        else
            cout << i << " " << j << " ";
    }
    cout << '\n';
}

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

    int t;
    cin >> t;
    while(t--)
    {
        solve();
    }
    return 0;
}
```

G - String Game

প্রতি মুভে কিন্তু একটা 0 এবং একটা 1 কমছে। অর্থাৎ 0 এবং 1 এর মাঝে যেটা কম আছে সেটা আগে শেষ হবে। তাই কমটা যদি বিজোড় সংখ্যা হয় তো Zlatan শেষ মুভ দিয়ে জিতবে এবং জোড় হলে Ramos জিতবে।

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

void solve()
{
    int n; cin >> n;
    string s;
    cin >> s;
    map<char, int> cnt;
    for(int i = 0; i < n; i++)
        cnt[s[i]]++;

    if(min(cnt['0'], cnt['1']) % 2 == 1)
        cout << "Zlatan\n";
    else
        cout << "Ramos\n";
}

int main()
{
    ios_base::sync_with_stdio(0); cin.tie(0);
    int t;
    cin >> t;
    while(t--)
    {
        solve();
    }
}
```

H - Useful Decomposition

Simple path হলো কয়েকটা কানেক্টেড নোড যেখানে কোনো cycle নেই এবং কোনো node এর degree (node এর সাথে directly connected node সংখ্যা) ২ এর বেশি না। অন্যভাবে বলতে পারি, কয়েকটা কানেক্টেড নোড যেখানে leaf সংখ্যা সর্বোচ্চ ২।
এখন আমাদের একটা ট্রি কে এমনভাবেকয়েকটা simple path এ ভাগ করতে হবে যেন তাদের সবার মাঝে একটা কমন node থাকে।

আমরা node গুলোকে ৩ প্রকারে আলাদা করে ফেলতে পারি।

leaf - যাদের সাথে একটা node এর সরাসরি কানেকশন আছে।
connector - যাদের সাথে দুইটা node এর সরাসরি কানেকশন আছে।
head - যাদের সাথে দুই এর অধিক node এর সরাসরি কানেকশন আছে।

কোনো Tree তে যদি কোনো head না ই থাকে, তাহলে Tree টা নিজেই সিম্পল পাথ। দুইপাশের দুইটা leaf এর মাধমে পুরো path টা কে রিপ্রেজেন্ট করলেই হয়ে গেল। একটা পাথ যেহেতু, কমন এর কিছু নাই।

কোনো Tree তে যদি একটামাত্র head থাকে, তার সাথে বাকি leaf গুলো সবই simple path. তাই head এর সাথে বাকি leaf গুলো প্রিন্ট করে দিলে, সবগুলো simple path এর মাঝে head টা কমন।

কোনো Tree তে যদি একাধিক head নোড থাকে তাহলে একটা node কমন রেখে একাধিক simple ভাগ করতে গেলে head গুলোর মধ্যবর্তী edge গুলো বারবার নেওয়ার প্রয়োজন হবে। তারমানে এটা সম্ভব না।

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

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

    int n;
    cin >> n;
    vector<vector<int>> adj(n + 1);
    for (int i = 1; i < n; i++)
    {
```

```
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<int> leaf, head;
    for (int i = 1; i <= n; i++)
    {
        if (adj[i].size() == 1)
            leaf.push_back(i);
        else if (adj[i].size() > 2)
            head.push_back(i);
    }

    if(head.empty())
    {
        cout << "Yes\n1\n";
        cout << leaf[0] << " " << leaf[1] << '\n';
    }
    else if(head.size() == 1)
    {
        cout << "Yes\n";
        cout << leaf.size() << '\n';
        for(int i = 0; i < leaf.size(); i++)
            cout << head[0] << " " << leaf[i] << '\n';
    }
    else
        cout << "No\n";

}
```