ABC 070 解説

writer: Hec

2017 年 8 月 12 日

A: Palindromic Number

整数 N を文字列として入力を受け取ります。N は 3 桁の正整数であるため、100 の位の数字と 1 の位の数字が一致すれば、回文数となります。最後に、回文数である場合は「Yes」、そうでない場合は「No」を出力します。

C++のコード例

```
int main(void) {
    string N;
    cin >> N;
    if (N[0] == N[2])
        cout << "Yes" << endl;
    else
        cout << "No" << endl;
    return 0;
}
```

B:　Two Switches

Alice と Bob が、2 人ともスイッチを押している状況を考えてみます。そうすると、ロボットを動かし始めてから $\max(A, C)$ 秒後から $\min(B, D)$ 秒後までの間、2 人がスイッチを押していることが分かります。したがって、答えは次のようになります。

$$\begin{cases} \min(B, D) - \max(A, C) \text{ 秒} & (\max(A, C) \leq \min(B, D)) \\ 0 \text{ 秒} & (\max(A, C) > \min(B, D)) \end{cases}$$

ここで、$\min(\cdot)$ は最小値、$\max(\cdot)$ は最大値を表します。

なお、問題の制約条件が小さいので、ループを利用したシミュレーションでも通すことができます。

C++のコード例

```cpp
int main(void) {
    int A,B,C,D;
    cin >> A >> B >> C >> D;

    const int lower = max(A,C);
    const int upper = min(B,D);

    cout << max(0,upper-lower) << endl;

    return 0;
}
```

C: Multiple Clocks

まず、この問題で求めたい答えを A とおくと、A は $T_i(1 \le i \le N)$ で割り切れる最小の正の整数です。A をシミュレーションで求めようとすると、時間計算量は $O(NA)$ となるため TLE となります。

ここで、A について詳しく考えていきます。A は $T_i(1 \le i \le N)$ で割り切れることから $T_i(1 \le i \le N)$ の公倍数です。したがって、A は以下の式で求めることができます。

$$A = \mathrm{lcm}(T_1, \cdots, T_N) = \mathrm{lcm}(T_1, \mathrm{lcm}(T_2, \cdots \mathrm{lcm}(T_{N-1}, T_N) \cdots))$$

ここで、$\mathrm{lcm}(\cdot)$ は最小公倍数を表します。

2 つの正整数の最小公倍数を直接計算することはできないため、以下の公式を利用します[1]。

$$ab = \gcd(a, b)\mathrm{lcm}(a, b)$$

ここで、$\gcd(\cdot)$ は最大公約数を表します。

2 つの正整数の最大公約数はユークリッドの互除法を利用することで計算できます。ユークリッドの互除法は以下の再帰関数のような形で実装できます。

```
procedure GCD(a,b)
    if b = 0 then
        return a
    end if
    return GCD(b, a mod b)
end procedure
```

以上をまとめると、ユークリッドの互除法を利用して最大公約数を求め、公式を利用して最小公倍数を計算することで A が求まります。ユークリッドの互除法の計算量は $O(\log(\max T_i))$ であり、全体の時間計算量は $O(N \log(\max T_i))$ となるため間に合います。

なお、答えは 10^{18} 秒以内と保証されていますが、最小公倍数の計算の仕方によっては 64 bit 整数型のオーバーフローが発生するので、注意してください。

C++のコード例

```cpp
using ll = long long;

ll gcd(ll a,ll b){
    if(b == 0) return a;
    return gcd(b,a%b);
}

ll lcm(ll a,ll b){
    ll g = gcd(a,b);
    return a / g * b; // Be careful not to overflow
}

```

[1] 3 つ以上の正整数に対しては成り立たないので注意 $abc \neq \gcd(a, b, c)\mathrm{lcm}(a, b, c)$

```
int main(void) {
    int N;
    cin >> N;

    ll ans = 1LL;

    for (int i = 0; i < N; ++i) {
        ll T;
        cin >> T;
        ans = lcm(ans, T);
    }

    cout << ans << endl;
    return 0;
}
```

D:　Transit Tree Path

まず、与えられたグラフの性質について考えてみます。与えられたグラフは閉路がない連結グラフであるため、異なる任意の 2 頂点間の最短経路は 1 通りに定まります。次に、全ての質問クエリで求める最短経路は頂点 K を経由しています。したがって、各質問クエリの答えは 頂点 x_j から 頂点 K までの最短経路 + 頂点 K から 頂点 y_j までの最短経路となります。このことから、頂点 K から全ての頂点への最短経路を前計算することにより、各質問クエリを効率良く処理することが可能です。

頂点 K から全ての頂点への最短経路を前計算する方法として、頂点 K から DFS をするのが簡単です。実装としては、以下の再帰関数について DFS$(K, -1, 0)$ と呼ぶことで頂点 K から全ての頂点への最短経路を前計算できます。

---

procedure DFS(現在の頂点 v, v の親 p, 現在の距離 d)
　　頂点 K から 頂点 v までの距離は d
　　for 頂点 i: 頂点 v に隣接しているかつ未訪問 **do**
　　　　if $i = p$ の場合 **then**
　　　　　　continue
　　　　end if
　　　　DFS(i, v, $d+$ 頂点 v と 頂点 i の間の辺のコスト)
　　end for
　　return
end procedure

また、グラフの頂点数が $N \leq 10^5$ と大きいため、隣接行列でグラフを持つと空間計算量が $O(N^2)$ となり MLE します。そこで、隣接リストを利用して実装することにより、空間計算量が $O(N)$ に抑えることができます。実装は C++のコード例を参考にしてください。

以上をまとめると、グラフを隣接リストで管理して、頂点 K から DFS をすることにより、頂点 K から全ての頂点への最短経路を前計算します。そのあと、各質問クエリに対して、頂点 K から 頂点 x_j までの最短経路と頂点 K から 頂点 y_j までの最短経路の和を出力します。この解法の時間計算量は $O(N + Q)$ となり、十分間に合います。

C++のコード例（頂点番号を 0-indexed とする）

```cpp
using ll = long long;
const int limit = 100010;
using edge = struct {int to; ll cost;};
vector<edge> tree[limit];
ll depth[limit];

void dfs(int v, int p, ll d) {
    depth[v] = d;
    for (auto &e : tree[v]) {
        if (e.to == p) continue;
        dfs(e.to, v, d + e.cost);
    }
}
```

```cpp

int main(void) {
    int n;
    cin >> n;

    for (int i = 0; i < n - 1; ++i) {
        int a, b, c;
        cin >> a >> b >> c;
        a--, b--;
        tree[a].push_back({b, c});
        tree[b].push_back({a, c});
    }

    int q, k;
    cin >> q >> k;
    k--;

    dfs(k, -1, 0);
    for (int i = 0; i < q; ++i) {
        int x, y;
        cin >> x >> y;
        x--, y--;
        cout << depth[x] + depth[y] << endl;
    }

    return 0;
}
```