

# 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

---

```
1 int main(void) {
2     string N;
3     cin >> N;
4     if (N[0] == N[2])
5         cout << "Yes" << endl;
6     else
7         cout << "No" << endl;
8     return 0;
9 }
```

---

## 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

---

```
1 int main(void) {
2     int A,B,C,D;
3     cin >> A >> B >> C >> D;
4
5     const int lower = max(A,C);
6     const int upper = min(B,D);
7
8     cout << max(0,upper-lower) << endl;
9
10    return 0;
11 }
```

---

## C: Multiple Clocks

まず、この問題で求めたい答えを  $A$  とおくと、 $A$  は  $T_i (1 \leq i \leq N)$  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、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  $T_i (1 \leq i \leq N)$  で割り切れることから  $T_i (1 \leq i \leq N)$  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。

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

ここで、 $\text{lcm}(\cdot)$  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<sup>1</sup>。

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

ここで、 $\text{gcd}(\cdot)$  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。

---

```
1: procedure GCD( $a, b$ )
2:   if  $b = 0$  then
3:     return  $a$ 
4:   end if
5:   return GCD( $b, a \bmod b$ )
6: 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

---

```
1 using ll = long long;
2
3 ll gcd(ll a, ll b){
4     if(b == 0) return a;
5     return gcd(b, a%b);
6 }
7
8 ll lcm(ll a, ll b){
9     ll g = gcd(a, b);
10    return a / g * b; // Be careful not to overflow
11 }
12
```

---

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

```
13 int main(void) {
14     int N;
15     cin >> N;
16
17     ll ans = 1LL;
18
19     for (int i = 0; i < N; ++i) {
20         ll T;
21         cin >> T;
22         ans = lcm(ans, T);
23     }
24
25     cout << ans << endl;
26     return 0;
27 }
```

---

## 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  $\text{DFS}(K, -1, 0)$  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。

---

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

---

また、グラフの頂点数が  $N \leq 10^5$  と大きいと、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)

---

```
1 using ll = long long;
2 const int limit = 100010;
3 using edge = struct {int to; ll cost;};
4 vector<edge> tree[limit];
5 ll depth[limit];
6
7 void dfs(int v, int p, ll d) {
8     depth[v] = d;
9     for (auto &e : tree[v]) {
10         if (e.to == p) continue;
11         dfs(e.to, v, d + e.cost);
12     }
13 }
```

```

14
15 int main(void) {
16     int n;
17     cin >> n;
18
19     for (int i = 0; i < n - 1; ++i) {
20         int a, b, c;
21         cin >> a >> b >> c;
22         a--, b--;
23         tree[a].push_back({b, c});
24         tree[b].push_back({a, c});
25     }
26
27     int q, k;
28     cin >> q >> k;
29     k--;
30
31     dfs(k, -1, 0);
32     for (int i = 0; i < q; ++i) {
33         int x, y;
34         cin >> x >> y;
35         x--, y--;
36         cout << depth[x] + depth[y] << endl;
37     }
38
39     return 0;
40 }

```

---