1　全方位木 DP

n 頂点の木が与えられる．ある頂点 r を根として以下のような DP を考える．

- 頂点 i が葉のとき $dp[i]$ は既知である．
- 頂点 i が内部節点で $(c_{i,1}, \ldots, c_{i,m})$ を子に持つとき $dp[i]$ は $(dp[c_{i,1}], \ldots, dp[c_{i,m}])$ から計算できる．
- 最終的に求めたい答えは $dp[r]$ である．

このアルゴリズムは，上の DP で $r = 1, \ldots, n$ としたときそれぞれについての $dp[r]$ を $O(n)$ で求める[*1]．

いま，適当な関数 f と演算 $\oplus$ が定義されて $dp[i]$ が次のように計算できるとする．

$$dp[i] = f(dp[c_{i,1}]) \oplus \cdots \oplus f(dp[c_{i,m}])$$

頂点 i を根として見たとき，i のある子 $c_{i,k}$ 以下の部分木がない木における $dp[i]$ を $dp[i]_k$ と書くことにすると，これは次のようになっているはずである．

$$dp[i]_k = f(dp[c_{i,1}]) \oplus \cdots \oplus f(dp[c_{i,k-1}]) \oplus f(dp[c_{i,k+1}]) \oplus \cdots \oplus f(dp[c_{i,m}])$$

このアルゴリズムではこれを高速に求められる必要があるので，左右からの累積和を用いて次のように書けるとする[*2]．

$$\begin{aligned} dp_L[i][j] &= f(dp[c_{i,1}]) \oplus \cdots \oplus f(dp[c_{i,j-1}]) \\ dp_R[i][j] &= f(dp[c_{i,j}]) \oplus \cdots \oplus f(dp[c_{i,m}]) \\ dp[i]_k &= dp_L[i][k] \oplus dp_R[i][k+1] \end{aligned}$$

全体としては，DFS を 2 回行う．最初の DFS では頂点 1 を根としたときの DP を求め，次の DFS ではその結果を元に各頂点を根としたときの DP を求める．

木を隣接リスト表現で管理しているとする．このとき，i に関する隣接リストの j 番目の要素は，対象の木を i を根として見たときの j 番目の子として考えることができる．また，頂点 i に関する隣接リストの j 番目の要素が，頂点 1 を根としたときの i の親であるとき，$p[i] = j$ とする．

各頂点 i に対し，i の次数が $\delta(i)$ のとき，$dp_L[i]$ および $dp_R[i]$ を要素数 $\delta(i) + 1$ の配列としておき，各要素を $\oplus$ の単位元で初期化しておく．$dp[i]$ を i を根としたときの問題の答えとする．

[*1] ここ正確ではなくて，各マージの際の単位演算を $O(n)$ 回行うみたいなことが言いたい．
[*2] 適当に単位元を番兵としておいておく．たぶん葉での値が単位元になると思う．

Algorithm 1: 最初の DFS

function DFS0(i, p)
- $x \leftarrow d$ ▷ 答えを DP の初期値で初期化
- **for** $j \in \{1, \ldots, n\}$ **do**
  - $u \leftarrow g[i][j]$
  - **if** $u = p$ **then**
    - $p[i] \leftarrow j$
    - **continue**
  - $y \leftarrow f(\text{DFS0}(u, i))$
  - $x \leftarrow x \oplus y$
  - $dp_L[i][j+1] \leftarrow dp_R[i][j] \leftarrow y$
- **return** x

Algorithm 2: 二回目の DFS

function DFS1(i, p, p_i)
- **if** $i \neq 1$ **then**
  - $y \leftarrow f(d \oplus dp_L[p][p_i] \oplus dp_R[p][p_i + 1])$
  - $dp_L[i][p[i]+1] \leftarrow dp_R[i][p[i]] \leftarrow y$
- **for** $j \in \{2, \ldots, \delta(i)\}$ **do**
  - $dp_L[i][j] \leftarrow dp_L[i][j] \oplus dp_L[i][j-1]$
- **for** $j \in \{\delta(i)-1, \ldots, 1\}$ **do**
  - $dp_R[i][j] \leftarrow dp_R[i][j] \oplus dp_R[i][j+1]$
- $dp[i] = (d \oplus dp_R[i][1])$
- **for** $j \in \{1, \ldots, \delta(i)\}$ **do**
  - $u \leftarrow g[i][j]$
  - **if** $i \neq p$ **then** DFS1(u, i, j)

DFS0$(1, 0)$ および DFS1$(1, 0, 0)$ を呼び出すとよい.

```
std::vector<intmax_t> dp_on_tree(const tree& g) {
  size_t n = g.size();
  std::vector<size_t> parent(n, -1);

  std::pair<intmax_t, intmax_t> const leaf(1, 1);
  std::pair<intmax_t, intmax_t> const unit(0, 0);
  auto f = [](std::pair<intmax_t, intmax_t> const& c) {
```

```
    return std::make_pair(c.first, c.first+c.second);
};

std::vector<std::vector<std::pair<intmax_t, intmax_t>>> dp0(n), dp1(n);
std::vector<intmax_t> dp(n);
for (size_t i = 0; i < n; ++i) {
  dp0[i].resize(g[i].size()+1, unit);
  dp1[i].resize(g[i].size()+1, unit);
}

make_fix_point([&](auto dfs0, size_t v, size_t p) -> std::pair<intmax_t, intmax_t> {
    std::pair<intmax_t, intmax_t> res = leaf;
    for (size_t i = 0; i < g[v].size(); ++i) {
      size_t u = g[v][i];
      if (u == p) {
        parent[v] = i;
        continue;
      }
      std::pair<intmax_t, intmax_t> tmp = f(dfs0(u, v));
      res += tmp;
      dp0[v][i+1] = dp1[v][i] = tmp;
    }
    return res;
})(0, -1);

make_fix_point([&](auto dfs1, size_t v, size_t p, size_t pi) -> void {
    if (v != 0) {
      std::pair<intmax_t, intmax_t> tmp = f(leaf + dp0[p][pi] + dp1[p][pi+1]);
      dp0[v][parent[v]+1] = tmp;
      dp1[v][parent[v]] = tmp;
    }
    {
      for (size_t i = 1; i < dp0[v].size(); ++i)
        dp0[v][i] += dp0[v][i-1];
      for (size_t i = dp1[v].size()-1; i--;)
        dp1[v][i] += dp1[v][i+1];

      dp[v] = (leaf + dp0[v][0] + dp1[v][0]).second;
    }
    for (size_t i = 0; i < g[v].size(); ++i) {
```

```
        size_t u = g[v][i];
        if (u != p) dfs1(u, v, i);
      }
  })(0, -1, -1);

  return dp;
}
```