全方位木 DP

えびちゃん

2019 年 11 月 14 日

1　問題設定

n 頂点の木 (V, E) と，単位元 $e_\oplus$ を持つモノイド $(T, \oplus)$ および関数 $f : T \times T \to T$ に関する問題を考える．次のように型 T の値 $\mathrm{dp}_r[v]$ を定義する．

$$\mathrm{dp}_r[v] = f(\mathrm{dp}_r[c^r_1] \oplus \cdots \oplus \mathrm{dp}_r[c^r_k]).$$

ここで，根を r としたときの頂点 v の子を番号の昇順に $(c^r_1, \ldots, c^r_k)$ とする．また，v が葉のときは次のように定義する．

$$\mathrm{dp}_r[v] = f(e_\oplus).$$

このとき，$r = \{1, \ldots, n\}$ について $\mathrm{dp}_r[r]$ を求めたい．

2　考察

まず，以下が成り立つ．

$$\begin{aligned}\mathrm{dp}_r[v] &= f(\mathrm{dp}_r[c^r_1] \oplus \cdots \oplus \mathrm{dp}_r[c^r_k]) \\ &= f(\mathrm{dp}_v[c^r_1] \oplus \cdots \oplus \mathrm{dp}_v[c^r_k]).\end{aligned}$$

すなわち，各 r について $\mathrm{dp}_r[1], \ldots, \mathrm{dp}_r[n]$ を求める必要はなく，頂点の対 (u, v) について $\mathrm{dp}_u[v]$ を求める必要があるのは u と v が隣接する場合のみである．そのため，$|V|^2$ 個の要素を求める必要はなく，$2|E|$ 個の要素を求めれば十分であることがわかる．

さて，$\mathrm{dp}_r[v]$ を求めることを考える．根を r としたときの v の子は $(c^r_1, \ldots, c^r_k)$ であった．このとき，v と隣接する頂点を番号の昇順に $(u_1, \ldots, u_m)$ とすると $m = k + 1$ となる．さらに，根を r としたときの v の親を p_r とすると，ある $1 \leqslant m_r \leqslant m$ が存在して $u_{m_r} = p_r$ となる．つまり，$u_1, \ldots, u_m$ を m_r を境に二分でき，以下のように書ける．

$$\begin{aligned}\mathrm{dp}_v[c^r_1] \oplus \cdots \oplus \mathrm{dp}_v[c^r_k] &= (\mathrm{dp}_v[u_1] \oplus \cdots \oplus \mathrm{dp}_v[u_{m_r-1}]) \oplus (\mathrm{dp}_v[u_{m_r+1}] \oplus \cdots \oplus \mathrm{dp}_v[u_m]) \\ &= \left(\bigoplus_{i=1}^{u_{m_r}-1} \mathrm{dp}_v[c_i]\right) \oplus \left(\bigoplus_{i=u_{m_r}+1}^{m} \mathrm{dp}_v[c_i]\right).\end{aligned}$$

これを高速に求めるためには，dp_v の $\oplus$ に関する左右からの累積和を持っておけばよい．ここで，m_r 以外は r に依存していないことがうれしい．

さて，各 v について dp_v の $\oplus$ に左右からの累積和を求めたい．これには二回 DFS を行うとよい．まず，頂点 1 から DFS を行うことで 1 を根とする向きの値は求められる．すなわち，隣接する頂点対 (u, v) について，u の方が 1 に近いならば $\mathrm{dp}_u[v]$ はこの DFS で求められる．ここで，dp_1 の各値は求められているので，dp_1 の累積和も計算できる．

その後もう一度 DFS を行い，残りの $\mathrm{dp}_v[u]$ を求める．このとき，$\mathrm{dp}_v[u]$ を求めるために必要な $\mathrm{dp}_v[??]$ について，$\mathrm{dp}_v[u]$ 以外は一回目の DFS で求められている．一方で，（探索順から）$\mathrm{dp}_v[u]$ は dp_u の累積和は求められているので，これを適切に求めることで $\mathrm{dp}_v[u]$ も計算できる．

これらにより各 (u, v) について $\mathrm{dp}_v[u]$ を求めることができた．また，dp_v について累積和を求めるときに，全体の和を計算することで $\mathrm{dp}_v[v]$ を求められ，元の問題を解くことができた．

木の辺に型 U の重みがついているとき，f を $(T \times U) \times T \to T$ にするとよいかも．

$\langle$ 図が入る予定．$\rangle$

3 実装

$dp_0[v]$ では dp_v の左からの累積和を，$dp_1[v]$ では dp_v の右からの累積和を保持する．$dp_0[v]$ および $dp_1[v]$ は v の次数 $\delta(v)$ 要素の配列である．

3.1 擬似コード

Algorithm 1: 全方位木 DP

function DFS0(v, p)
  $x \leftarrow e_\oplus$
  foreach neighbor u_i **of** v **do**
    if $u_i = p$ **then**
      $\mathrm{par}[v] = i$ ▷ mark i^{th} neighbor as parent.
      continue
    $x' \leftarrow$ DFS0(u, v)
    $x \leftarrow x \oplus x'$
    $dp_0[v][i+1] \leftarrow dp_1[v][i] \leftarrow x'$
  return $f(x)$
function DFS1(v, p, p_v)
  if $v \neq 1$ **then**
    $x \leftarrow f(dp_0[p][p_i] \oplus dp_1[p][p_i+1])$
    $dp_0[v][\mathrm{par}[v]+1] \leftarrow dp_1[v][\mathrm{par}[v]] \leftarrow x$
  for $i \in (2, \ldots, \delta(v))$ **do** $dp_0[v][i] \leftarrow dp_0[v][i-1] \oplus dp_0[v][i]$
  for $i \in (\delta(v)-1, \ldots, 1)$ **do** $dp_1[v][i] \leftarrow dp_1[v][i] \oplus dp_1[v][i+1]$
  $dp[v] \leftarrow dp_1[v][0]$
  foreach neighbor u_i **of** v **do**
    if $u_i \neq p$ **then** DFS1(u_i, v, i)
DFS0(1, 0), DFS1(1, 0, 0)

3.2 C++

```
template <typename Monoid, typename UndirectedTree>
auto dp_on_tree(UndirectedTree const& g) {
  Monoid e{};
  size_t n = g.size();
  std::vector<size_t> parent(n, -1_zu);
```

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

make_fix_point([&](auto dfs0, size_t v, size_t p) -> Monoid {
    Monoid res = e;
    typename UndirectedTree::weight_type weight{};
    for (size_t i = 0; i < g[v].size(); ++i) {
      size_t u = g[v][i].target();
      if (u == p) {
        parent[v] = i;
        weight = g[v][i].weight();
        continue;
      }
      Monoid tmp = dfs0(u, v);
      res.append(tmp);
      dp0[v][i+1] = dp1[v][i] = tmp;
    }
    return res.f(weight);
})(0, -1_zu);

make_fix_point([&](auto dfs1, size_t v, size_t p, size_t pi) -> void {
    if (v != 0) {
      Monoid tmp = (dp0[p][pi] + dp1[p][pi+1]).f(g[p][pi].weight());
      dp0[v][parent[v]+1] = tmp;
      dp1[v][parent[v]] = tmp;
    }
    {
      for (size_t i = 1; i < dp0[v].size(); ++i)
        dp0[v][i].prepend(dp0[v][i-1]);
      for (size_t i = dp1[v].size()-1; i--;)
        dp1[v][i].append(dp1[v][i+1]);
      dp[v] = dp1[v][0];
    }
    for (size_t i = 0; i < g[v].size(); ++i) {
      size_t u = g[v][i].target();
```

```
      if (u != p) dfs1(u, v, i);
    }
  })(0, -1_zu, -1_zu);

  return dp;
}
```