MAC0345 - Desafios 2
Editorial Lista 3

Nathan Luiz e Enrique Junchaya

April 26, 2023

A. Gao on a tree.
Tópicos: HLD.

Este problema tem alguns jeitos de fazer. Vou descrever uma solução de HLD em que os caminhos **não** são segtree's.

Seja p_x a posição do nó x na ordem de visitação da dfs. Imagina que para cada cor c temos um `vector` ordenado que mantém todas as posições p_x dos nós x que tem cor c. Portanto, a subquery que queremos responder é:

- No caminho de x até h_x, onde h_x é a cabeça do caminho de x, quantos vértices tem cor c?

Nós queremos responder isso em $O(logN)$. Essa query pode ser convertida em:

- No `vector` da cor c tem posição de visitação da dfs no intervalo $[p_{h_x}, p_x]$?

Observe que essa conversão nós podemos fazer em $O(logN)$ com uma busca binária no `vector`. Portanto, a complexidade final é $O(Nlog^2N)$.

B. New Roads Queries.
Tópicos: Small to Large ou HLD.

Existem algumas abordagens diferentes. Vou descrever uma solução com DSU e Small to Large.

Será tudo feito offline. Inicialmente cada vértice é uma componente e vamos mergir quando adicionarmos as arestas na ordem correta. Para cada componente, teremos um `vector< pair<int, int> >` em que podemos salvar quais outros vértices queremos juntar, junto com o id da query. Portanto, se formos mergir dois vértices a e b, podemos iterar pelo `vector` de b e ver se o vértice que precisamos para completar a query está na componente de a. Se isso acontece, então temos a resposta.

```
void merge(int a, int b, int dia) {
    a = find_pai(a); b = find_pai(b);
    if(a == b) return;
    if(M[a].size() < M[b].size()) swap(a, b);

    for(auto [u, id_query] : M[b]) {
        if(find_pai(u) == a) ans[id_query] = dia;
        else M[a].pb(mp(u, id_query));
    }
    pai[b] = a;
}
```

A parte do Small to Large serve só para amortizar o merge. Portanto, a complexidade final fica $O(NlogN)$, supondo que a complexidade do seu DSU é $O(1)$.

C. Water Tree.
Tópicos: HLD.

Este problema é aplicação direta de HLD. Basta que a sua estrutura suporte update em range (ex segtree com lazy propagation). Operação do tipo 1 é só atualizar uma subárvore, do tipo 2 é atualizar todo mundo no caminho até a raiz, e do tipo 3 é uma query em um nó.

A complexidade final é $O(Nlog^2N)$.

D. XOR Tree.
Tópicos: Small to Large.

Não acho que consigo explicar melhor que o editorial. A simplificação que ele usou do peso do caminho ser $b_x \oplus b_y \oplus a_{LCA(x,y)}$ é bem legal e melhora bastante a implementação. Se os pesos estivesse nas arestas o caminho teria tamanho $b_x \oplus b_y$. Porém, também é possível resolver sem isso.

E. Lomsat gelral.
Tópicos: Small to Large.

Esse problema é bem parecido com o de descobrir quantas cores distintas tenho em cada subárvore, que foi resolvido em aula.

Para cada subárvore, basta salvarmos:

- um `map` com as cores e a frequência de cada cor;
- a frequência máxima que aparece na subárvore;
- a resposta.

A transição deve ficar algo do tipo:

```
ans[x] = a[x];
mx_freq[x] = 1;
freq[x][a[x]]++;

if (adj[x][0] != p) {
    swap (freq[x], freq[adj[x][0]]);
    swap (ans[x], ans[adj[x][0]]);
    swap (mx_freq[x], mx_freq[adj[x][0]]);

    for (auto u : adj[x]) if (u != p) {
        for (auto [cor, val] : freq[u]) {
            freq[x][cor] += val;

            if (freq[x][cor] > mx_freq[x]) {
                ans[x] = cor;
                mx_freq[x] = freq[x][cor];
            }
            else if (freq[x][cor] == mx_freq[x]) {
                ans[x] += cor;
            }
        }
    }
}
```

A complexidade final é $O(Nlog^2N)$.

F. Moonwalk challenge.
Tópicos : HLD, Hash.

Se temos uma consulta u, v, S, podemos responder ela como as ocorrências de S no caminho de u a $lca(u, v)$ + as ocorrências de `reverse`(S) no caminho de v a $lca(u, v)$ + as ocorrências de S que atravessam o $lca(u, v)$ com uma parte subindo e a outra descendo pela árvore. A terceira parte pode ser respondida em $O(|S|)$ apõs construir a string de tamanho $2|S| - 2$ no caminho de u a v centrada em $lca(u, v)$ (o que pode ser feito calculando os ancestrais adequados de u e v em $O(\lg n)$). Assim, vamos nos focar em como responder a consulta de contar as ocorrências de S no caminho de u a w, onde w é um ancestral de u.

Podemos resolver essa consulta com HLD. Para cada cadeia da HLD vamos manter um `map` cujas chaves serão os hashes de todas as strings da cadeia com tamamho menor ou igual a 100. Para cada um desses hashes, salvaremos no `map` uma lista ordenada das profundidades na árvore da posições finais da palavra que representam. Com isso podemos responder os trechos da consulta que estão completamente contidos numa cadeia. Basta olhar a lista salvada no `map` e fazer buscas binárias para delimitar o subarray dela que contém as aparições que estão dentro da consulta.

Desta forma, só falta contar as ocorrências de S que vão de uma cadeia a outra, o que também pode ser feito em $O(|S|)$ criando a string que vai de uma cadeia para outra(s) e que não contém as ocorrências que já foram encontradas. Vamos fazer isso apenas $O(\lg n)$ vezes pois há no máximo essa quantidade de cadeias no caminho de u até w.

Complexidade da solução: $O(n \lg n + q \lg n(\lg n + |S|))$ tempo e $O(n)$ espaço.