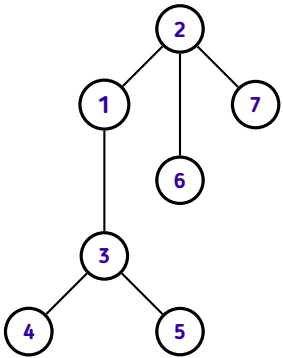
OJ 10459

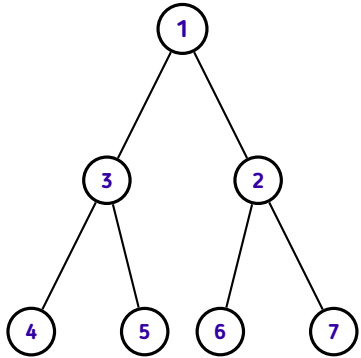
The Tree Root

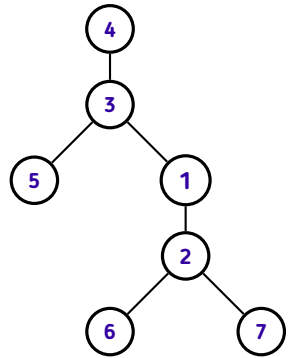
Prof. Edson Alves

Faculdade UnB Gama

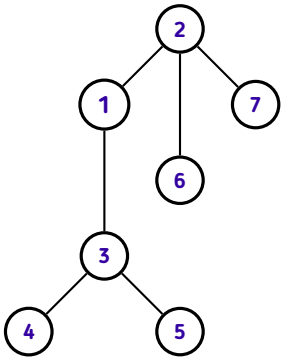
Tree is an important data structure. Searching is a basic operation in any data structure. In a tree searching mainly depends on its height. Consider the following three trees.

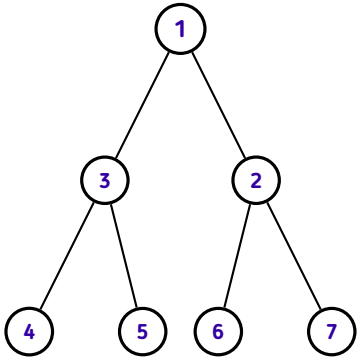


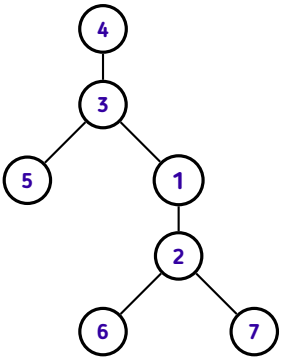




Árvore é uma importante estrutura de dados. Busca é uma operação básica em qualquer estrutura de dados. Em uma árvore a busca depende de sua altura. Considere as três árvores abaixo.







If you observe carefully, you will see that all trees are same except different nodes are used as roots. Here the height of the tree varies with the selection of the root. In the 1st tree root is '2' and height is 3. In 2nd one root is '1' and height is 2. And in last one root is '4' and height is 4. We will call '1' best root as it keeps the tree with the least possible height and '4' worst root for the opposite reason.

In this problem, you have to find out all best roots and worst roots for a given tree.

Se você observar atentamente, você verá que todas as três árvores são idênticas, exceto pelo fato de terem escolhidos nós diferentes como raízes. Aqui a altura da árvore varia de acordo com a escolha da raiz. Na primeira árvore a raiz é '2' e a altura é 3. Na segunda a raiz é '1' e a altura é 2. E na última a raiz é '4' e a altura é 4. Nos dizemos que '1' é uma raiz ótima no sentido que ela minimiza a altura máxima e '4' uma raiz péssima pela razão oposta.

Neste problema você deve encontrar todas as raízes ótimas e péssimas de uma dada árvore.

Input

Each dataset starts with a positive integer N $(3 \leq N \leq 5000)$, which is the number of nodes in the tree. Each node in the tree has a unique id from 1 to N. Then successively for each i'th node there will be a positive integer $K[i]$ following id of $K[i]$ nodes which are adjacent to i. Input is terminated by `EOF`.

Output

For each dataset print two lines. In the 1st line show all the best roots in ascending order and in next line show all worst roots in ascending order. See sample output for exact format.

Entrada

Cada caso de teste começa com um inteiro positivo N $(3 \leq N \leq 5000)$, o qual indica o número de nós na árvore. Cada nó da árvore tem um identificador único no intervalo de 1 a N. Então para cada i-ésimo nó haverá um inteiro positivo $K[i]$ seguido pelos identificadores dos $K[i]$ nós adjacentes a i. A entrada termina com `EOF`.

Output

Para cada caso de teste imprima duas linhas. Na primeira linha mostre todas as raízes ótimas, em ordem crescente, e na linha seguinte todas as raízes péssimas, em ordem crescente. Veja o exemplo para o formato exato da saída.

Exemplo de entrada e saída

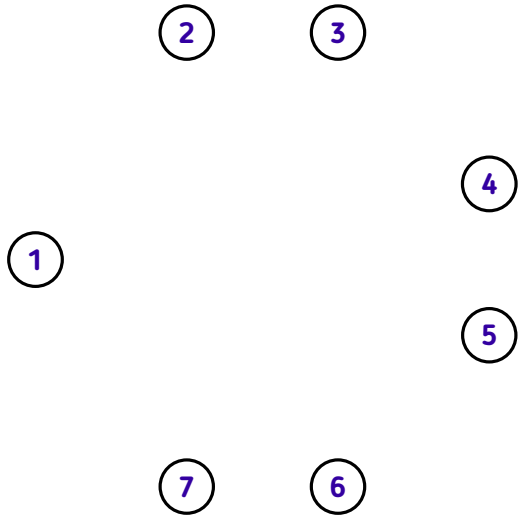
Exemplo de entrada e saída

7

Exemplo de entrada e saída

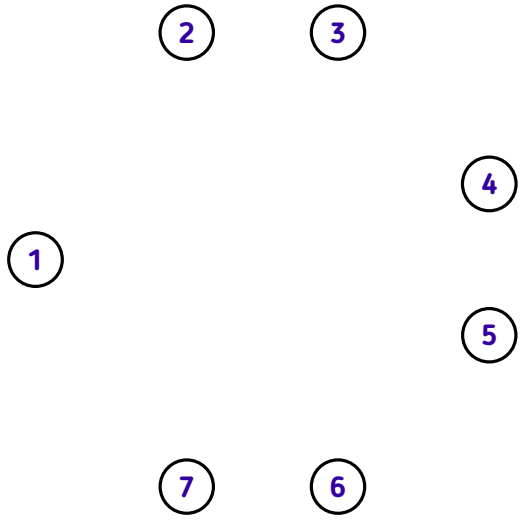
Exemplo de entrada e saída

7



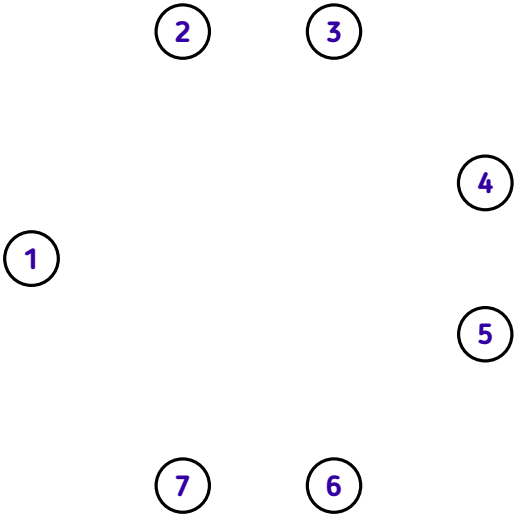
Exemplo de entrada e saída

```
7
2 2 3
```



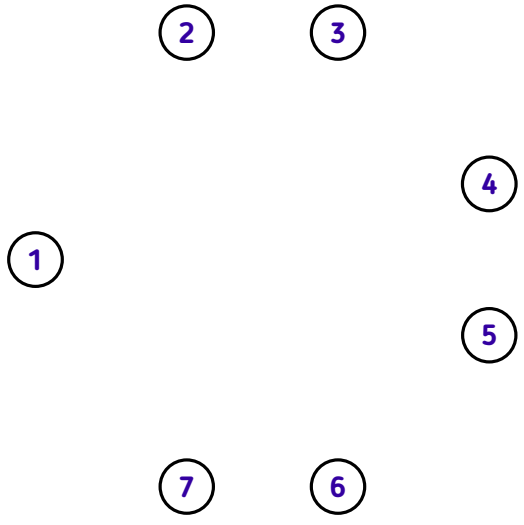
Exemplo de entrada e saída

7
2 2 3



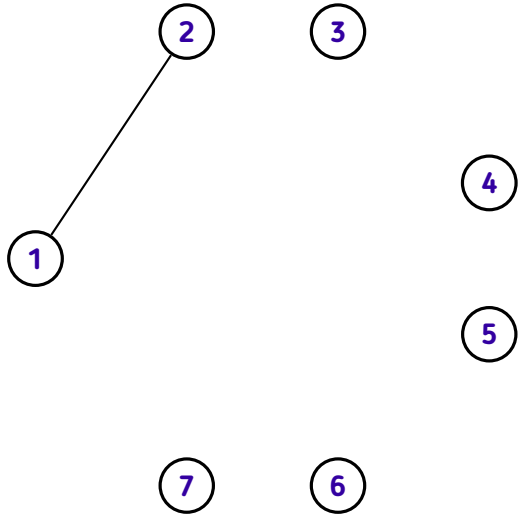
Exemplo de entrada e saída

7
2 2 3



Exemplo de entrada e saída

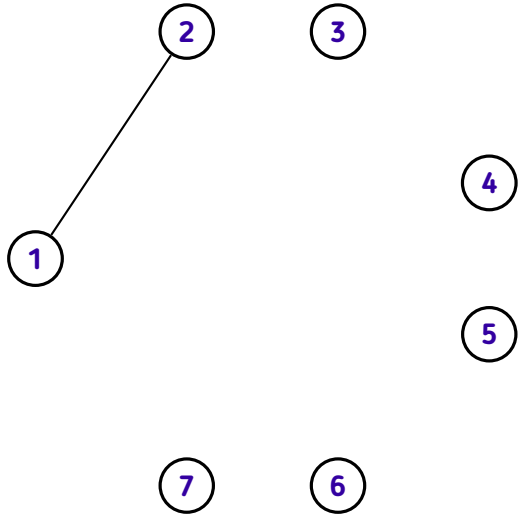
```
7
2 2 3
```



Exemplo de entrada e saída

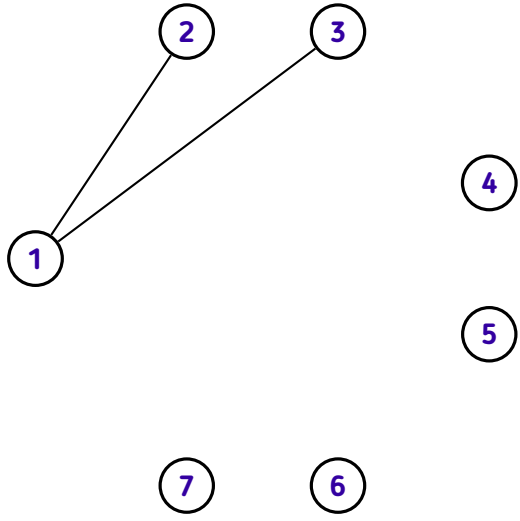
7
2 2 3

v_2



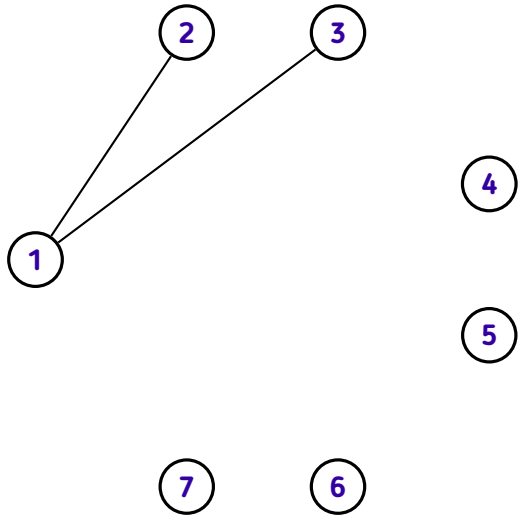
Exemplo de entrada e saída

```
7
2 2 3
```



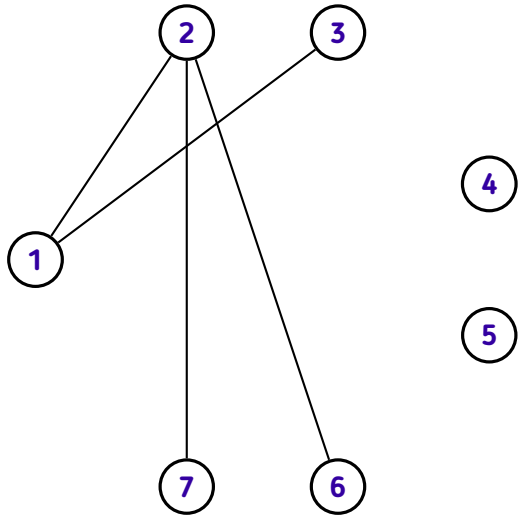
Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
```



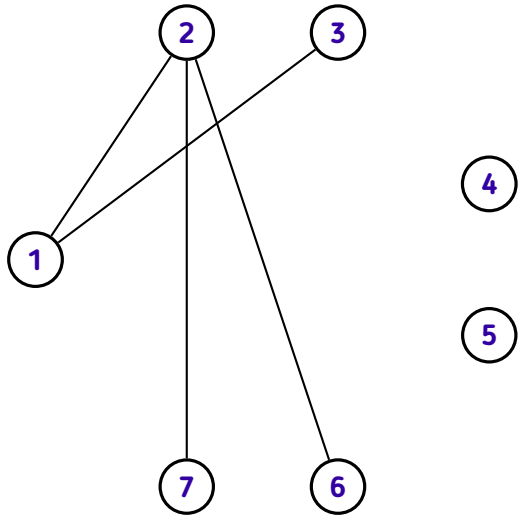
Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
```



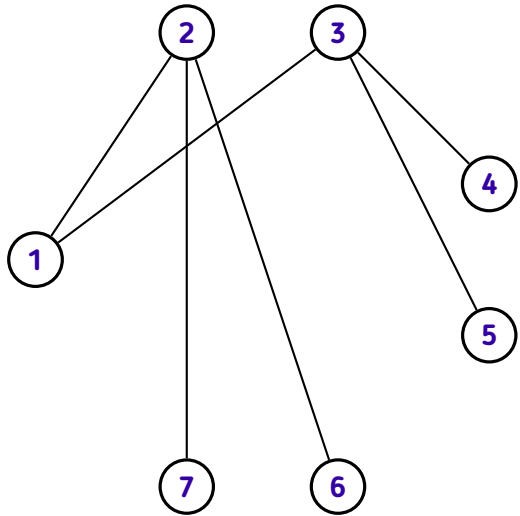
Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
3 1 4 5
```



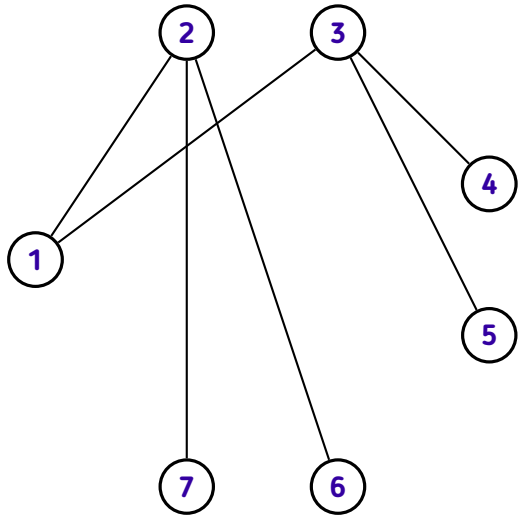
Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
3 1 4 5
```



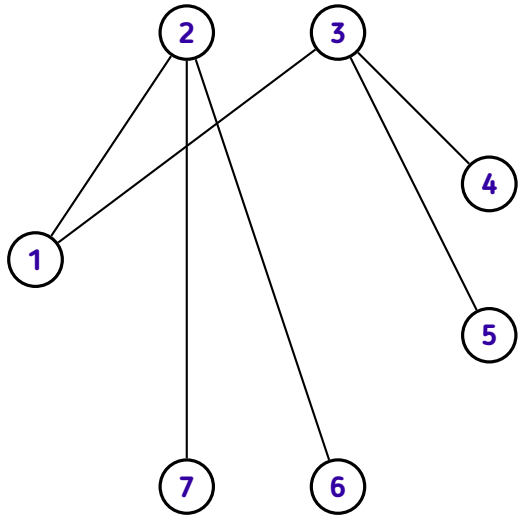
Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
```



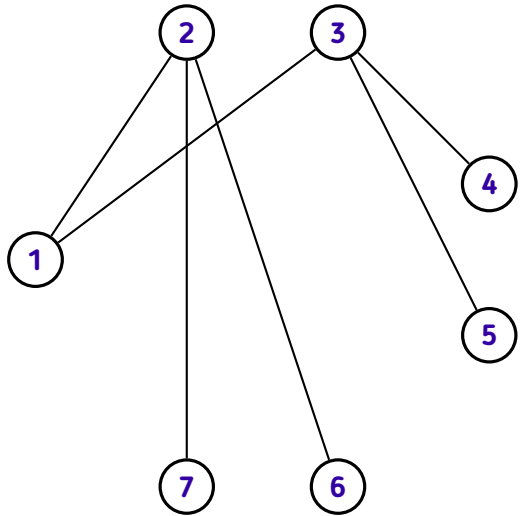
Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
```



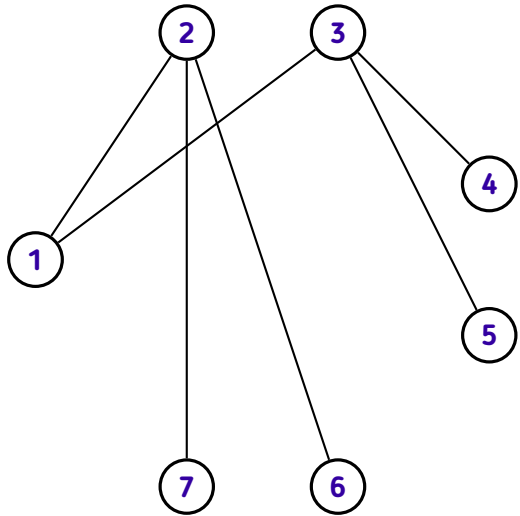
Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
```



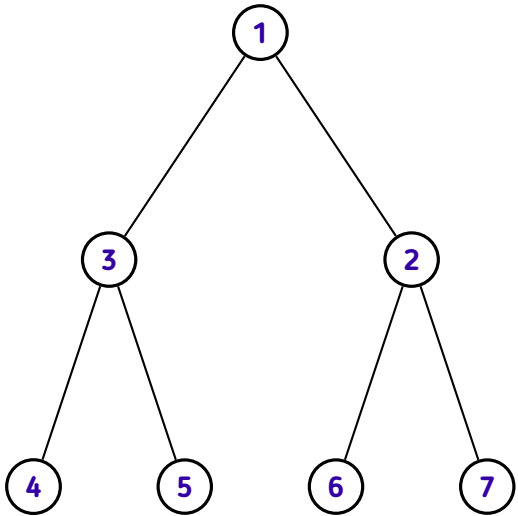
Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```



Exemplo de entrada e saída

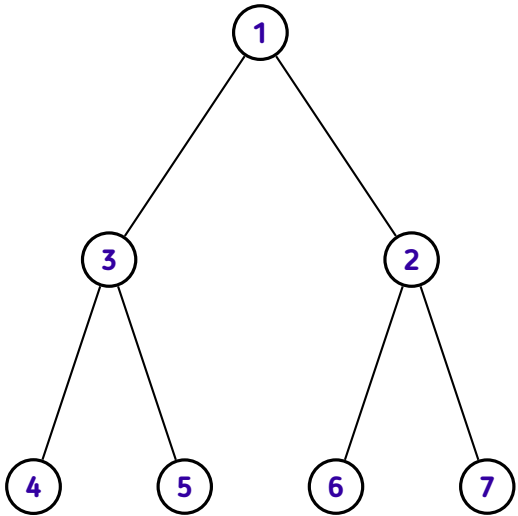
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```



Exemplo de entrada e saída

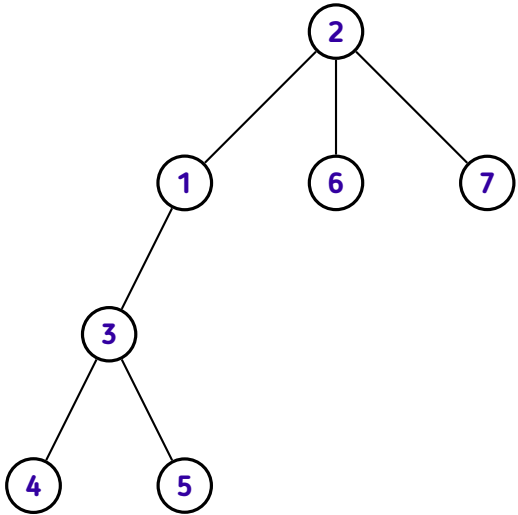
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```

$h = \mathbf{3}$



Exemplo de entrada e saída

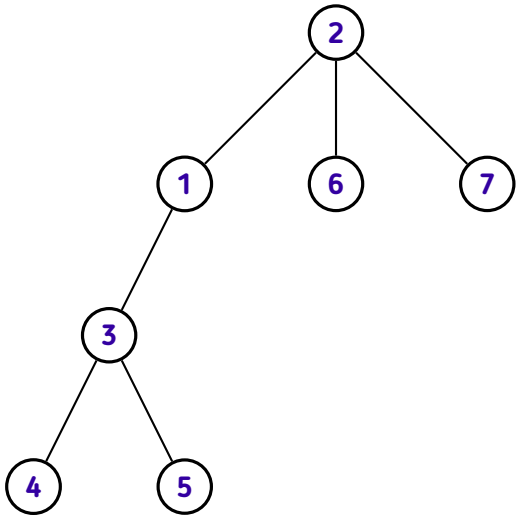
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```



Exemplo de entrada e saída

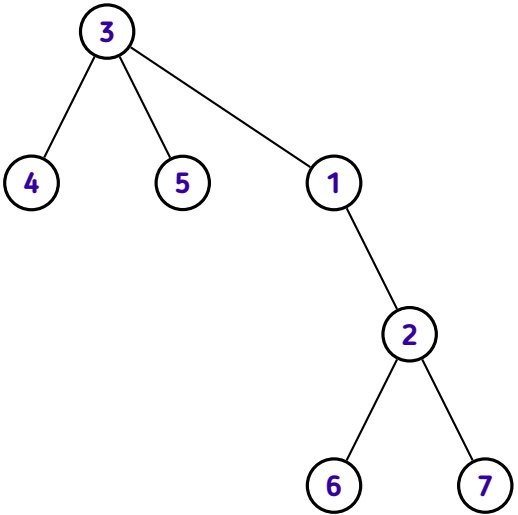
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```

$h = 4$



Exemplo de entrada e saída

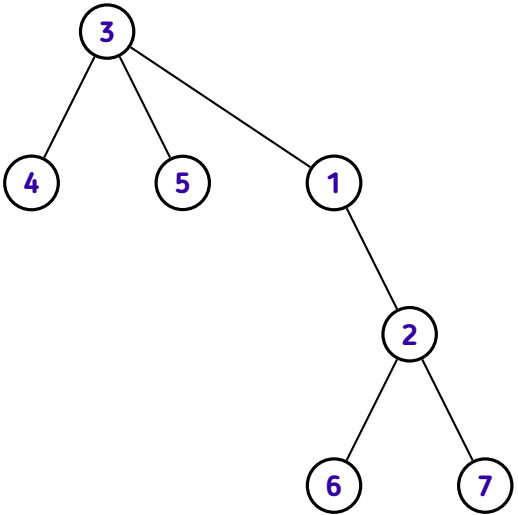
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```



Exemplo de entrada e saída

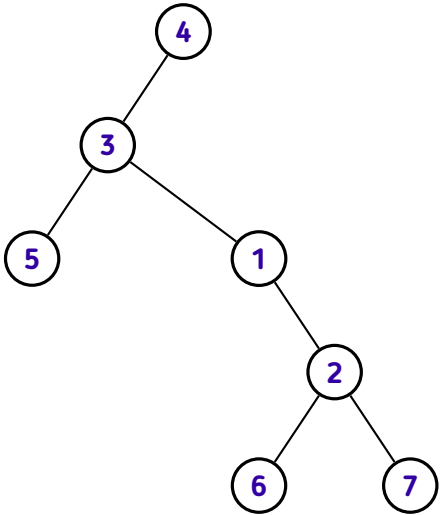
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```

$h = 4$



Exemplo de entrada e saída

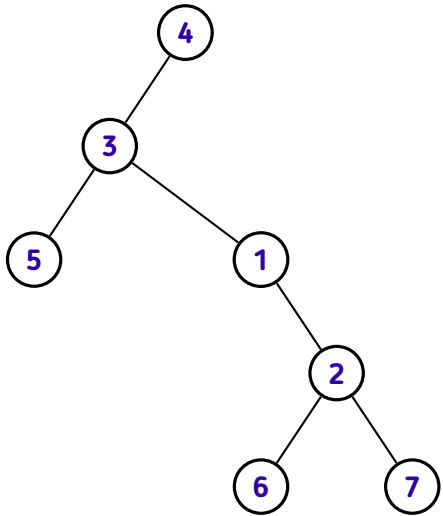
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```



Exemplo de entrada e saída

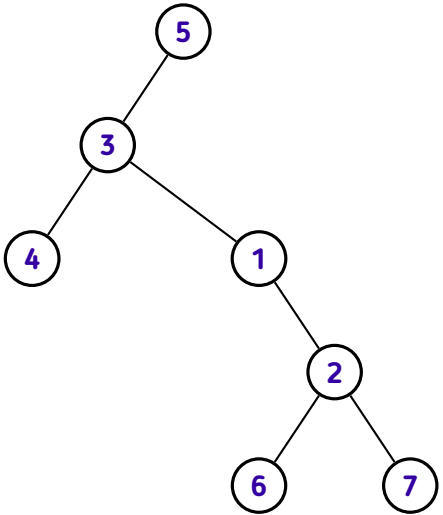
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```

$h = \mathbf{5}$



Exemplo de entrada e saída

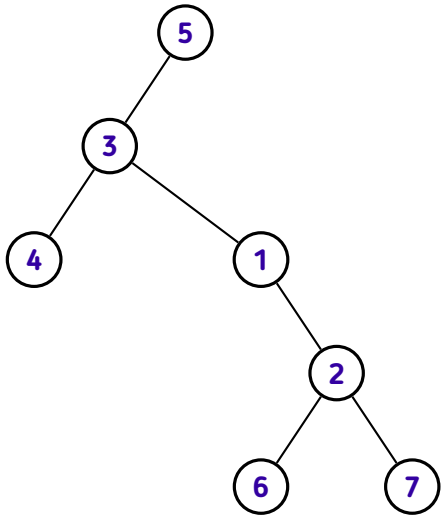
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```



Exemplo de entrada e saída

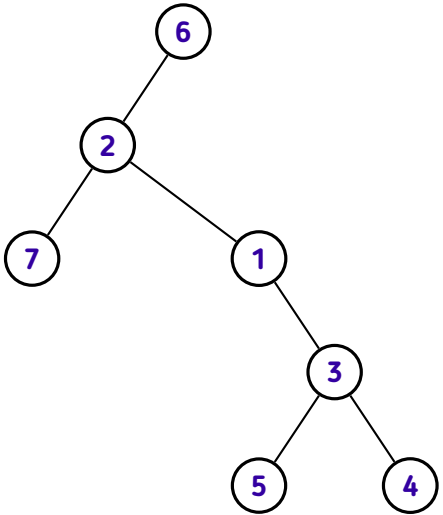
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```

$h = \mathbf{5}$



Exemplo de entrada e saída

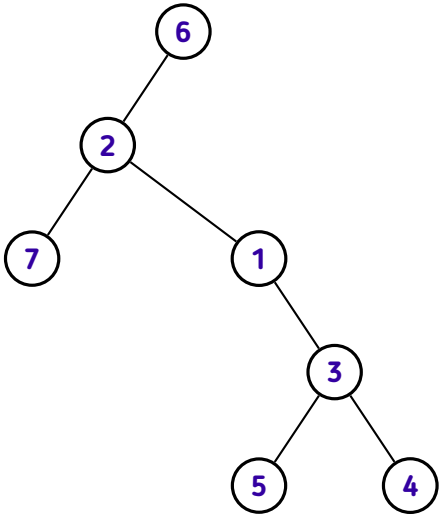
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```



Exemplo de entrada e saída

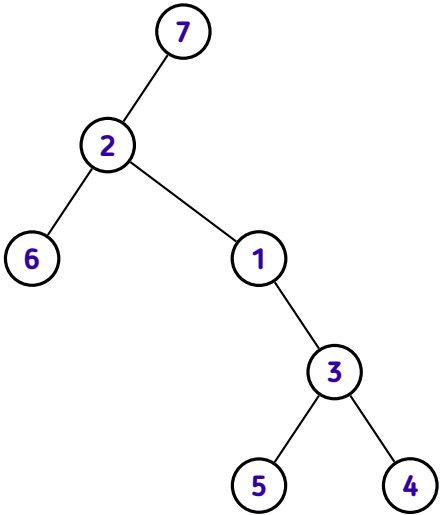
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```

$h = \mathbf{5}$



Exemplo de entrada e saída

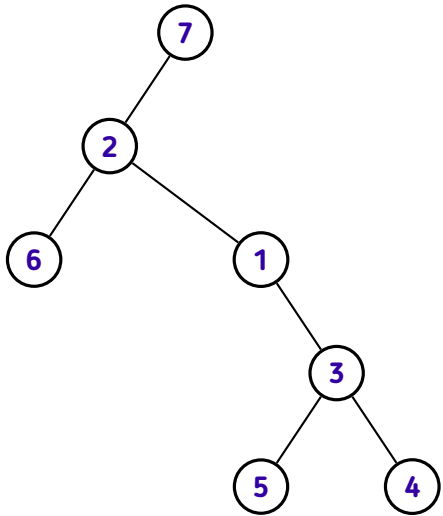
```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```



Exemplo de entrada e saída

```
7
2 2 3
3 1 6 7
3 1 4 5
1 3
1 3
1 2
1 2
```

$h = \mathbf{5}$



Solução

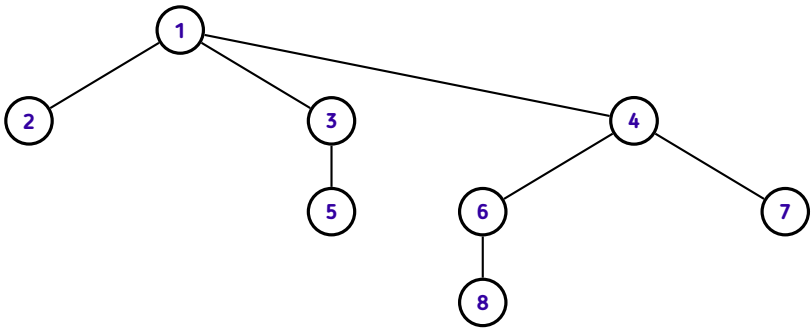
Solução

Aplicar a DFS N vezes para obter as alturas leva ao veredito TLE!

Solução

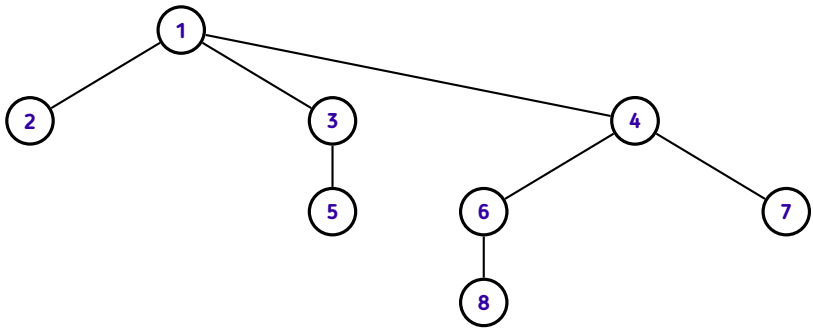
A ideia é usar a DFS em uma raiz em particular para extrair métricas e deduzir as alturas a partir destas métricas

Solução



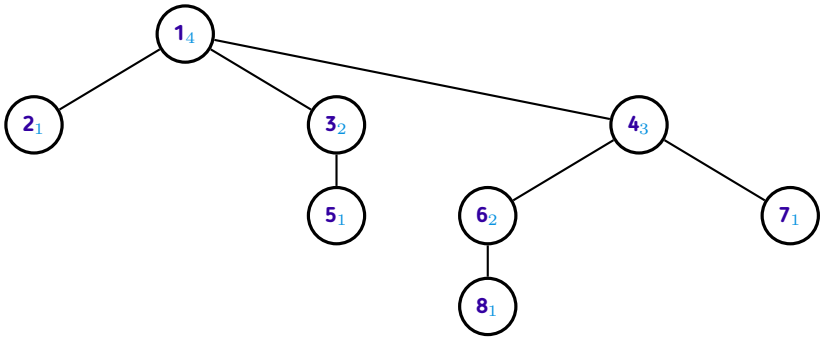
Solução

Métrica #1: $h(u) =$ **altura da subárvore cuja raiz é** u



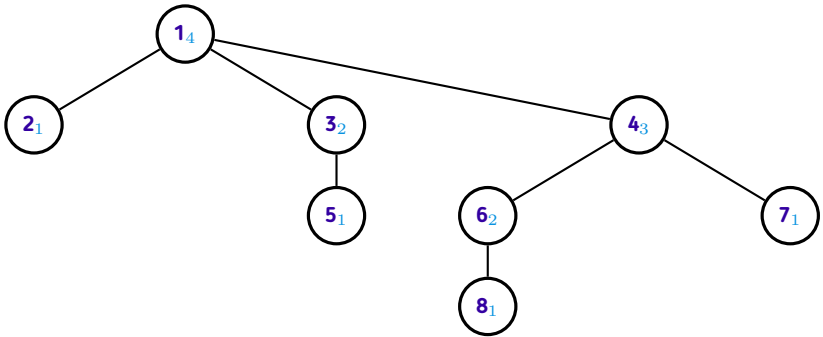
Solução

Métrica #1: $h(u) =$ **altura da subárvore cuja raiz é** u



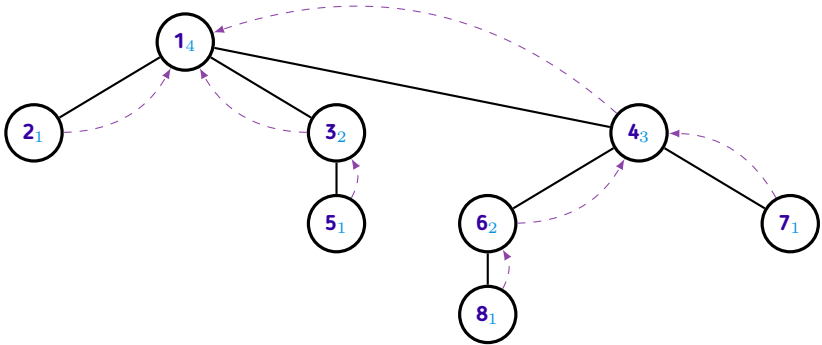
Solução

Métrica #2: $p(u) =$ pai do nó u, $p(r) = 0$ se r é a raiz



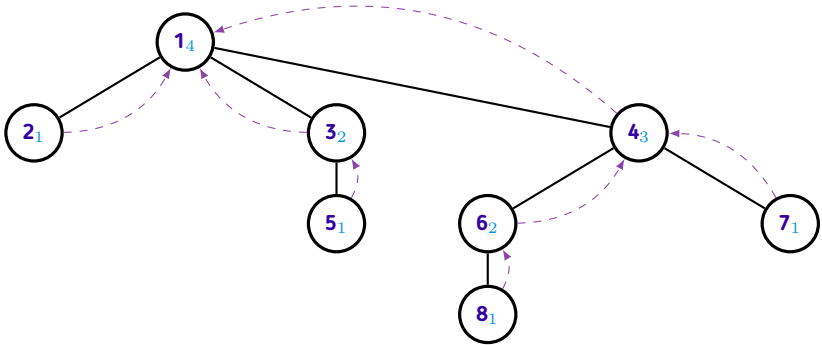
Solução

Métrica #2: $p(u) =$ pai do nó u, $p(r) = 0$ se r é a raiz



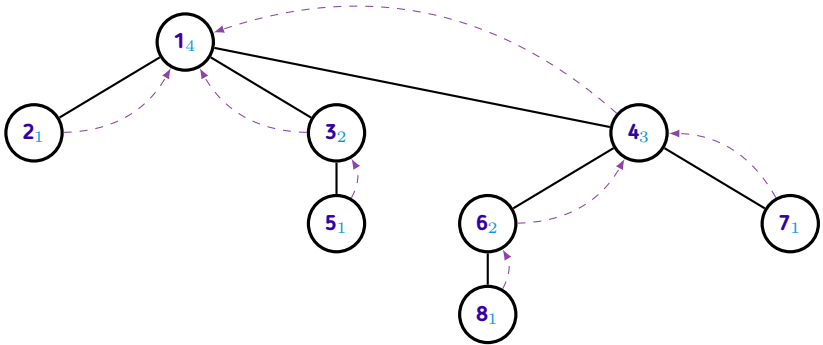
Solução

Métrica #3: $\delta(u, v) =$ **altura da árvore cuja raiz é** u **se** v **fosse o único filho de** u



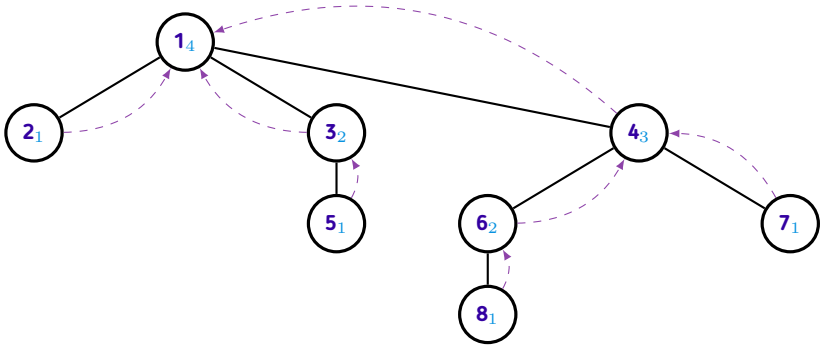
Solução

Para computar $\delta(u, v)$ é preciso processar os vértices na ordem da BFS



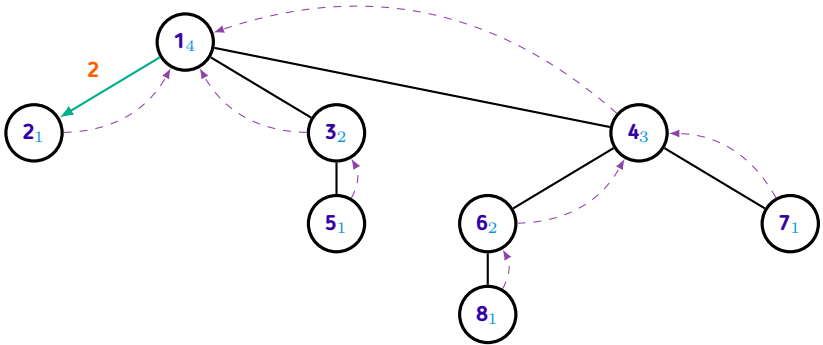
Solução

$\delta(r, v) = 1 + h(v)$, **se** r **é a raiz**



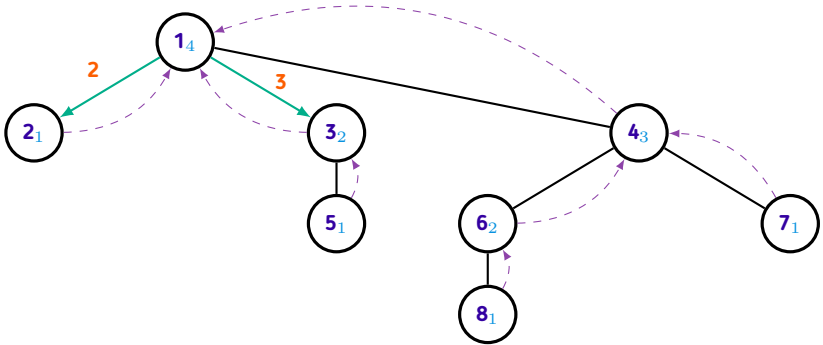
Solução

$\delta(r, v) = 1 + h(v)$, **se** r **é a raiz**



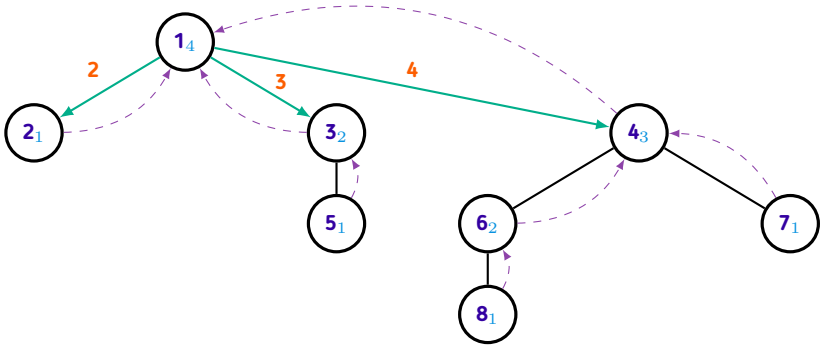
Solução

$\delta(r, v) = 1 + h(v)$, **se** r **é a raiz**



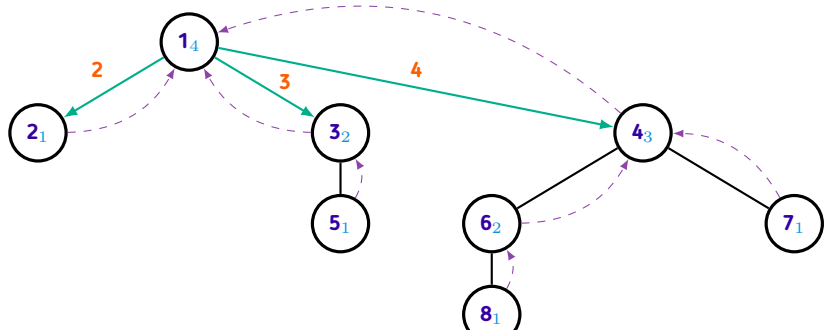
Solução

$\delta(r, v) = 1 + h(v)$, **se** r **é a raiz**



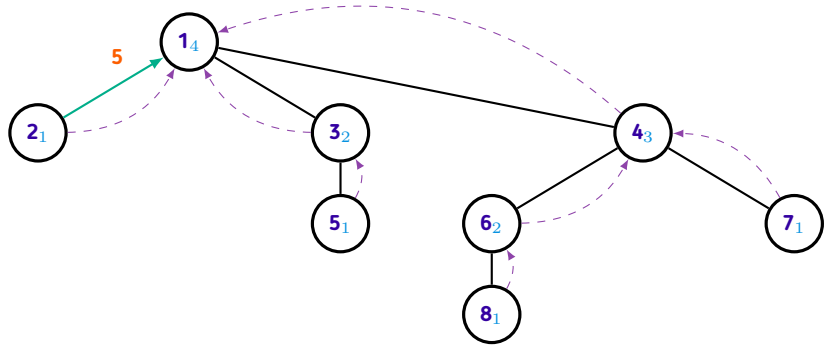
Solução

$$\delta(u,v) = \begin{cases} 1 + \max_{w \in \text{adj}[v], w \neq u}\{1, \delta(v,w)\}, & \text{se } v = p(u) \\ 1 + h(v), & \text{caso contrário} \end{cases}$$



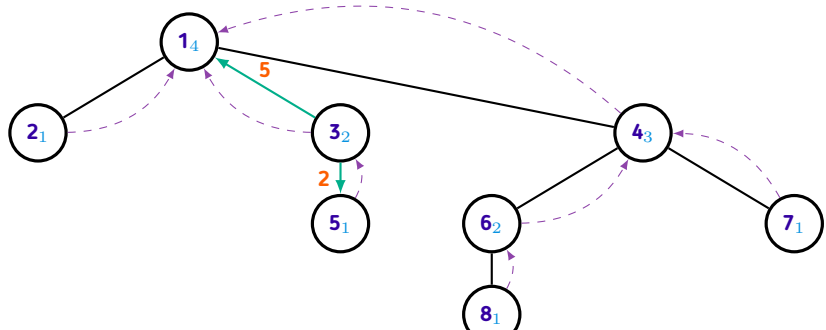
Solução

$$\delta(u,v) = \begin{cases} 1 + \max_{w \in \text{adj}[v], w \neq u}\{1, \delta(v,w)\}, & \text{se } v = p(u) \\ 1 + h(v), & \text{caso contrário} \end{cases}$$



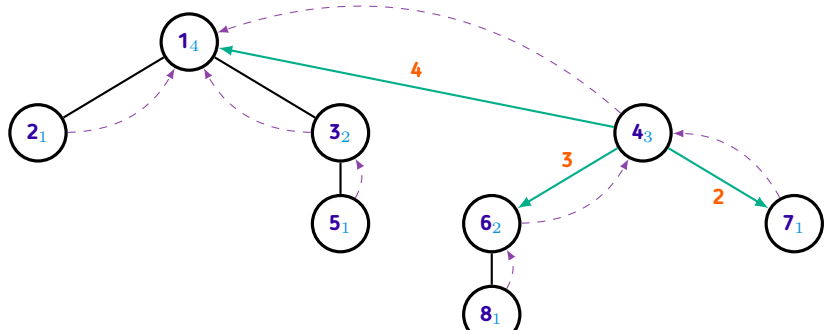
Solução

$$\delta(u,v) = \begin{cases} 1 + \max_{w \in \text{adj}[v], w \neq u}\{1, \delta(v,w)\}, & \text{se } v = p(u) \\ 1 + h(v), & \text{caso contrário} \end{cases}$$



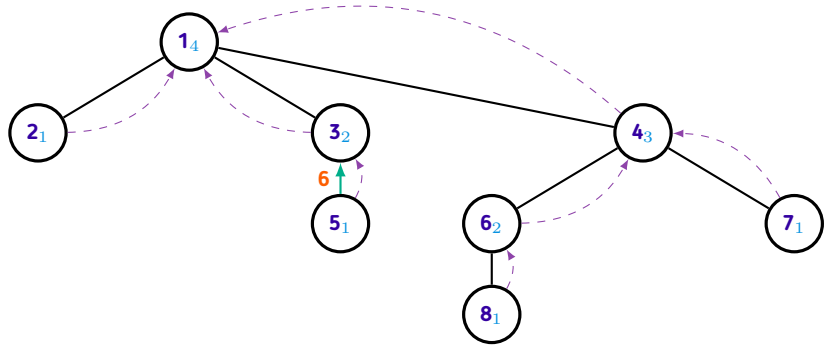
Solução

$$\delta(u,v) = \begin{cases} 1 + \max_{w \in \text{adj}[v], w \neq u}\{1, \delta(v,w)\}, & \text{se } v = p(u) \\ 1 + h(v), & \text{caso contrário} \end{cases}$$



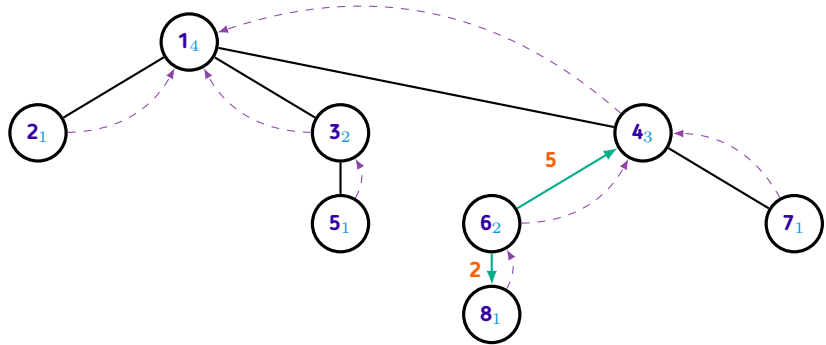
Solução

$$\delta(u,v) = \begin{cases} 1 + \max_{w \in \text{adj}[v], w \neq u}\{1, \delta(v,w)\}, & \text{se } v = p(u) \\ 1 + h(v), & \text{caso contrário} \end{cases}$$



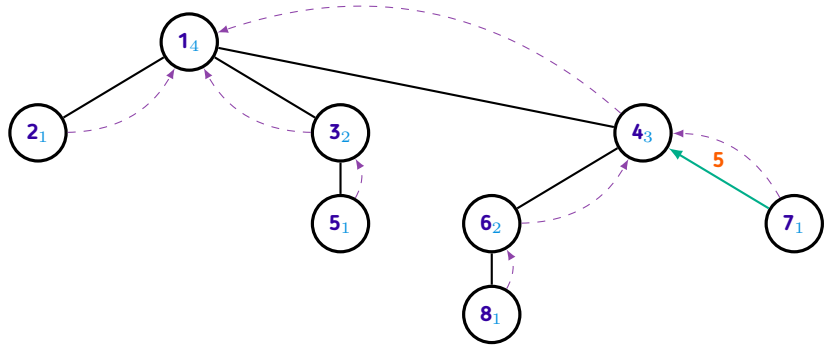
Solução

$$\delta(u,v) = \begin{cases} 1 + \max_{w \in \text{adj}[v], w \neq u}\{1, \delta(v,w)\}, & \text{se } v = p(u) \\ 1 + h(v), & \text{caso contrário} \end{cases}$$



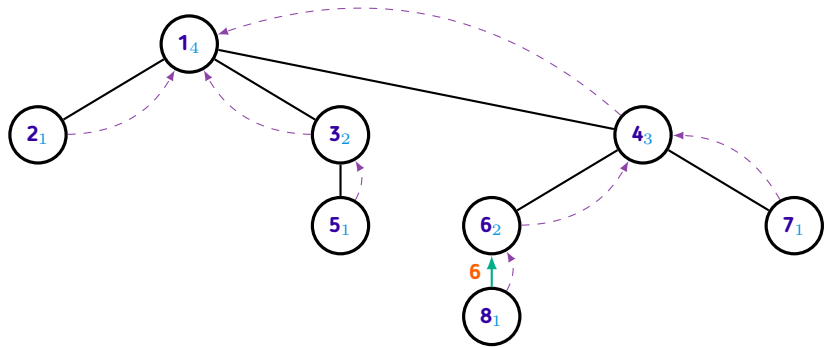
Solução

$$\delta(u,v) = \begin{cases} 1 + \max_{w \in \text{adj}[v], w \neq u}\{1, \delta(v,w)\}, & \text{se } v = p(u) \\ 1 + h(v), & \text{caso contrário} \end{cases}$$



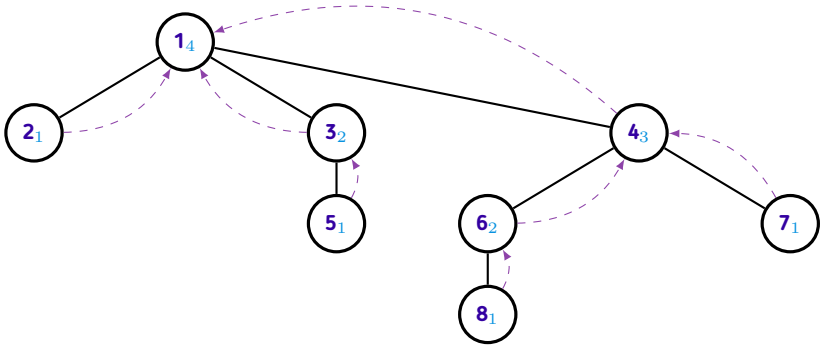
Solução

$$\delta(u,v) = \begin{cases} 1 + \max_{w \in \text{adj}[v], w \neq u}\{1, \delta(v,w)\}, & \text{se } v = p(u) \\ 1 + h(v), & \text{caso contrário} \end{cases}$$



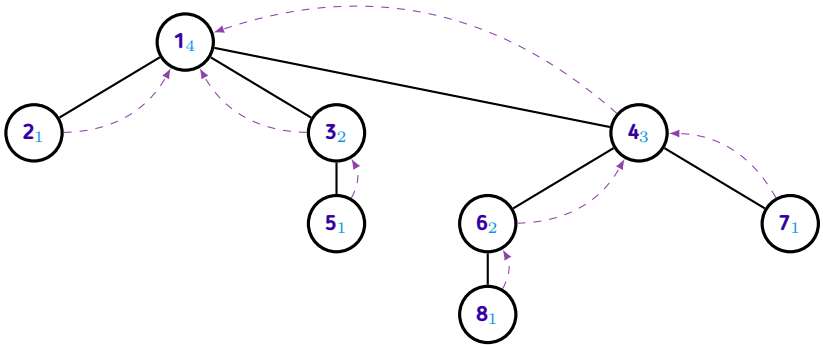
Solução

Por fim, seja $H(u)$ a altura da árvore cuja raiz é u



Solução

$$H(u) = \max_{v \in \mathsf{adj}[u]} \delta(u, v)$$



```cpp
int dfs(int u, int p, vector<int>& hs, vector<int>& ps)
{
    int h = 0;

    for (auto v : adj[u])
        if (v != p)
            h = max(h, dfs(v, u, hs, ps));

    ps[u] = p;

    return hs[u] = h + 1;
}
```

```
pair<vector<int>, vector<int>> solve(int N)
{
    vector<int> hs(N + 1), ps(N + 1);
    hs[1] = 1;

    dfs(1, 0, hs, ps);

    queue<int> q;
    map<ii, int> ys;
    map<int, vector<int>> ans;

    q.push(1);

    while (not q.empty())
    {
        auto u = q.front();
        q.pop();

        int H = 0;
```

```
        for (auto v : adj[u])
        {
            if (v != ps[u])
            {
                ys[ii(u, v)] = 1 + hs[v];
                q.push(v);
            } else
            {
                int y = 1;

                for (auto w : adj[v])
                    if (w != u)
                        y = max(y, ys[ii(v, w)]);

                ys[ii(u, v)] = 1 + y;
            }

            H = max(H, ys[ii(u, v)]);
        }
```

```
        ans[H].emplace_back(u);
    }

    auto best = ans.begin()->second;
    auto worst = ans.rbegin()->second;

    sort(best.begin(), best.end());
    sort(worst.begin(), worst.end());

    return { best, worst };
}
```