Explicação dos Exercícios de Árvores

Exercícios A, D, F, E

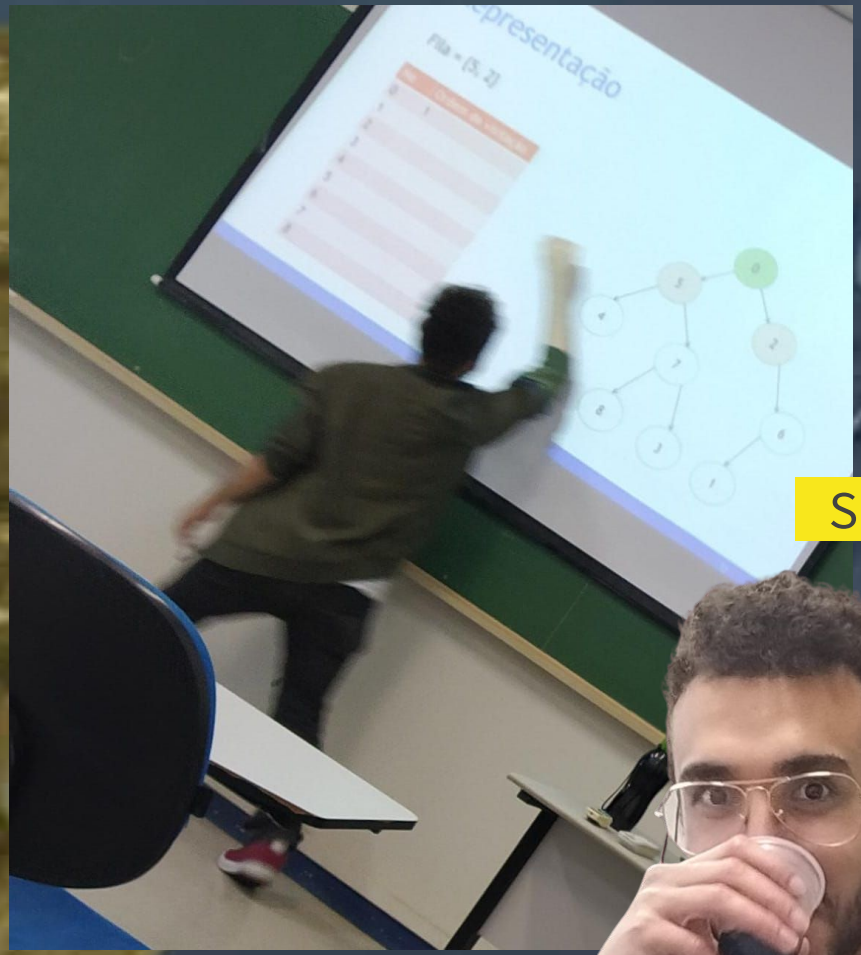
A - Cthulhu

Vamos supor que um Cthulhu consiga ser representado no espaço como sendo:

- Um grafo não direcionado que pode ser representado como um conjunto de uma árvore ou mais árvores que estão conectadas por um círculo simples

Na entrada é garantido que não existem múltiplas arestas nem *self-loops*

A - Cthulhu



Se liga aí na revisão!
heh

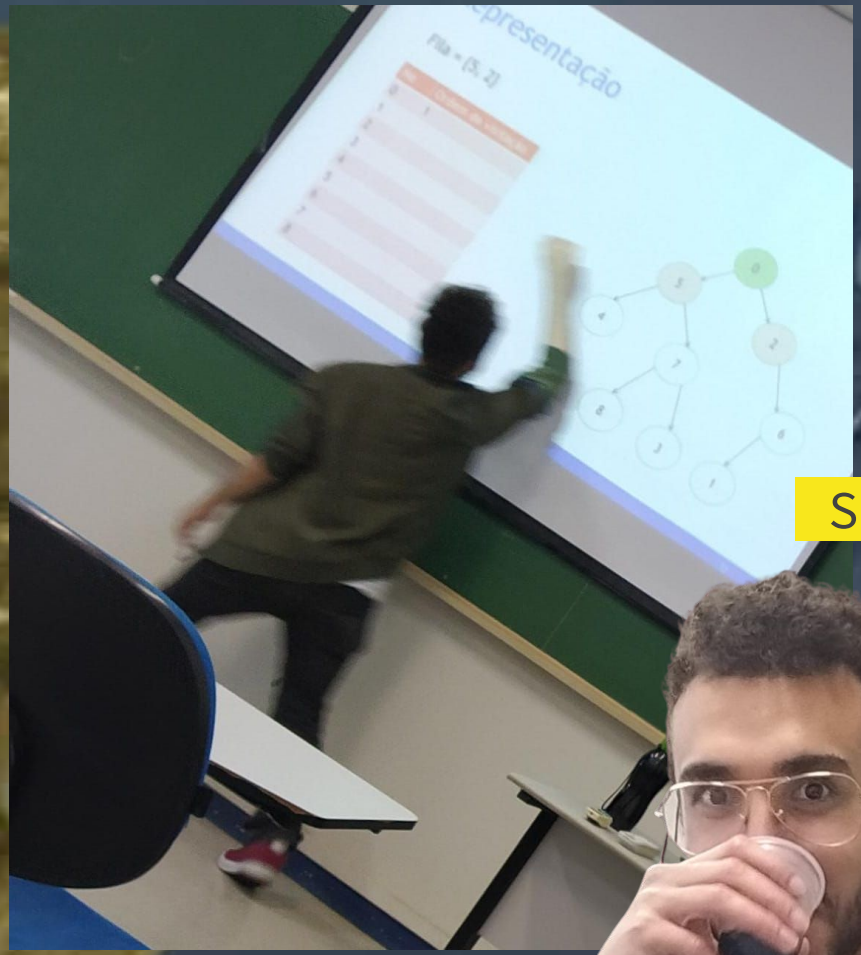
A - Cthulhu

Solução:

```
if (conexo == true && num_arestas == num_vertices)
        cout << “FHTAGN !”;
    else
        cout << “NO”;
```

Accepted!

D - Journey



Se liga aí na revisão! **heh**

D - Journey

- Encontrar o valor esperado de chegar à última cidade.
- N nós e N-1 arestas com todos os nós conectados
- O valor esperado tem relação com a probabilidade de se alcançar um nó.
- A definição de valor esperado vem da estatística

D - Journey

- Por exemplo, a probabilidade de tirar o número 2 em um dado é 1/6.
- Se fizermos 30 jogadas é esperado que tire o número 2 cinco vezes
- Se fizermos 40 jogadas o valor esperado para sair o dois no dado são 6.67 vezes.

$$\sum_{i=1}^{n} p_i(2)$$

$p_i(2)$ $->$ *probabilidade de sair dois na iésima jogada*

D - Journey

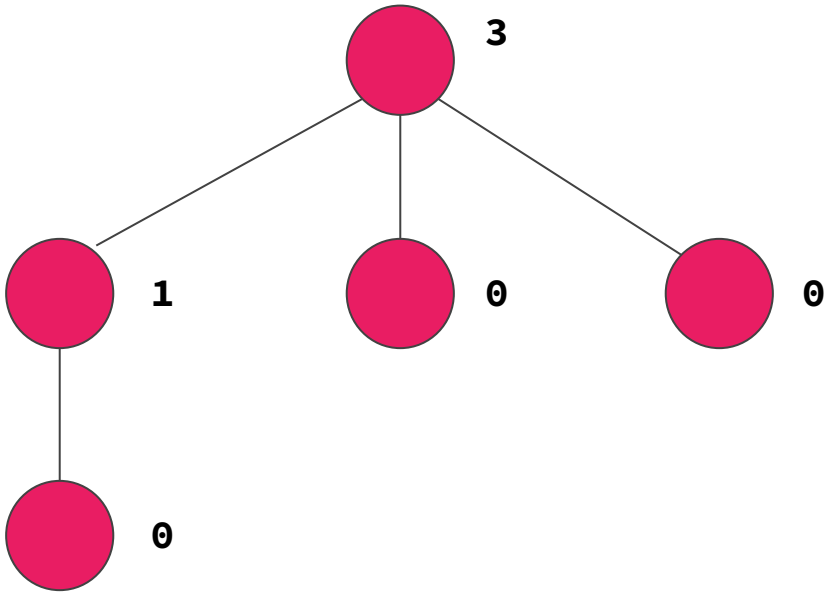
- O exercício quer o valor esperado para a distância percorrida durante a viagem

- Portanto, a esperança da distância da viagem é a soma das probabilidades de passar em cada aresta

$$\sum_{i=1}^{n} p_i(passar\ na\ aresta)$$

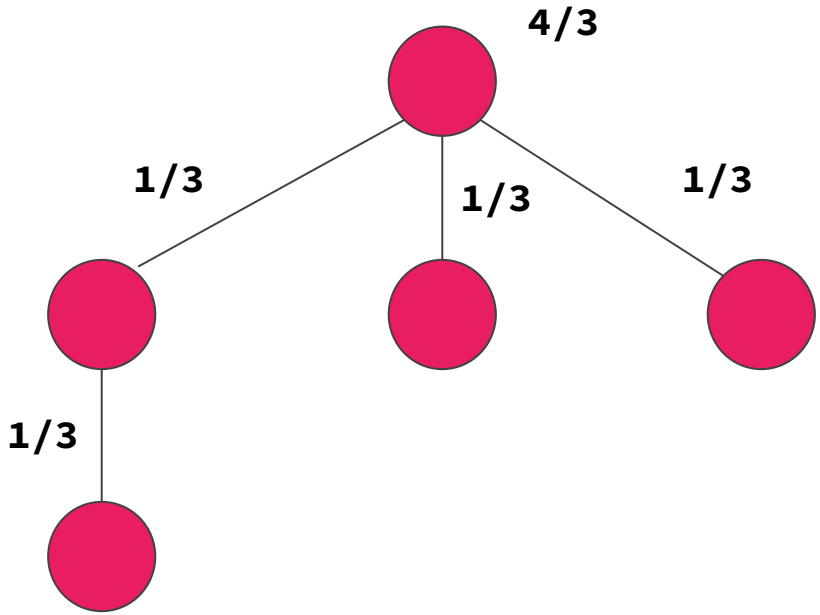
D - Journey

- Fazer uma dfs na árvore, e a probabilidade de acessar cada aresta será 1/num_filhos



D - Journey

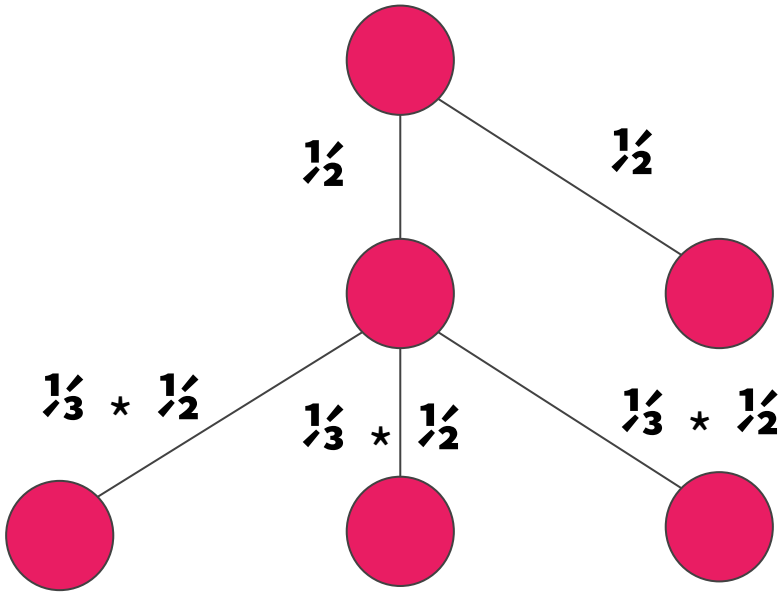
- Fazer uma dfs na árvore, e a probabilidade de acessar cada aresta será 1/num_filhos



Valor Esperado = 4/3

D - Journey

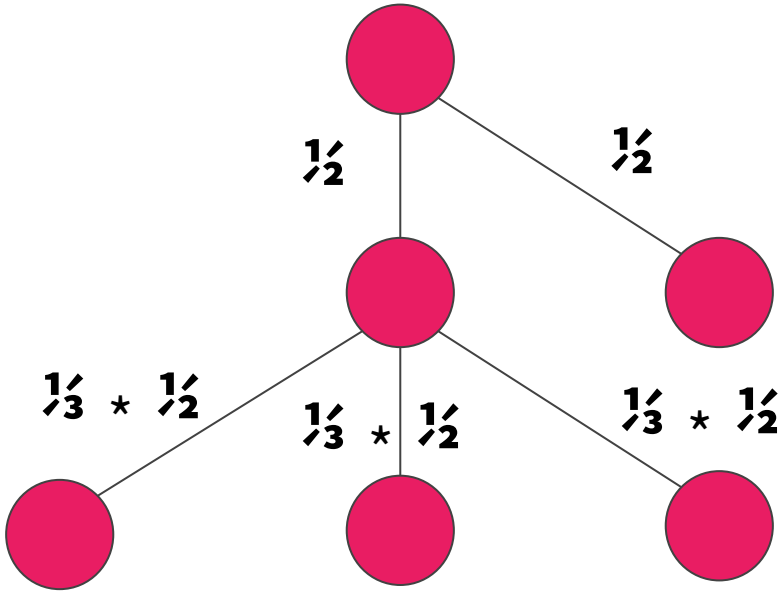
- Fazer uma dfs na árvore, e a probabilidade de acessar cada aresta será 1/num_filhos



Valor Esperado = 1.5

D - Journey

- Fazer uma dfs na árvore, e a probabilidade de acessar cada aresta será 1/num_filhos



Valor Esperado = 1.5

F - Longest Path on a Tree

F - Longest Path on a Tree

- Dada uma árvore não direcionada, queremos saber o **tamanho do maior caminho entre quaisquer dois nós** dela;

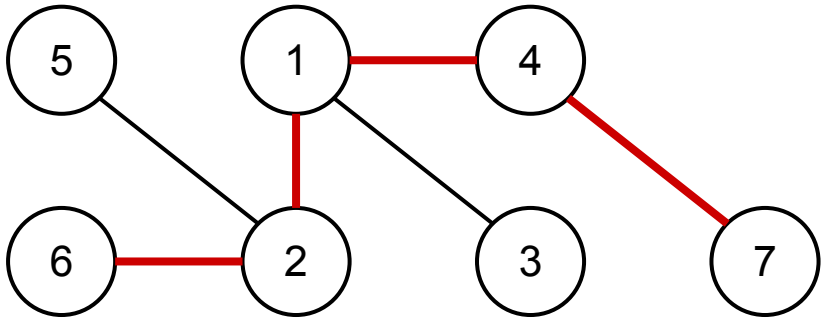
- Este tamanho é o **número de arestas percorridas** no caminho que **sai de um nó u** e **chega em um nó v**;

- Isto configura um problema de **cálculo de diâmetro** na árvore.

F - Longest Path on a Tree

- O **diâmetro** de uma árvore é o **maior comprimento de um caminho entre dois nós**;

- A árvore ao lado possui **diâmetro 4**, compreendendo o caminho **6 -> 2 -> 1 -> 4 -> 7**.



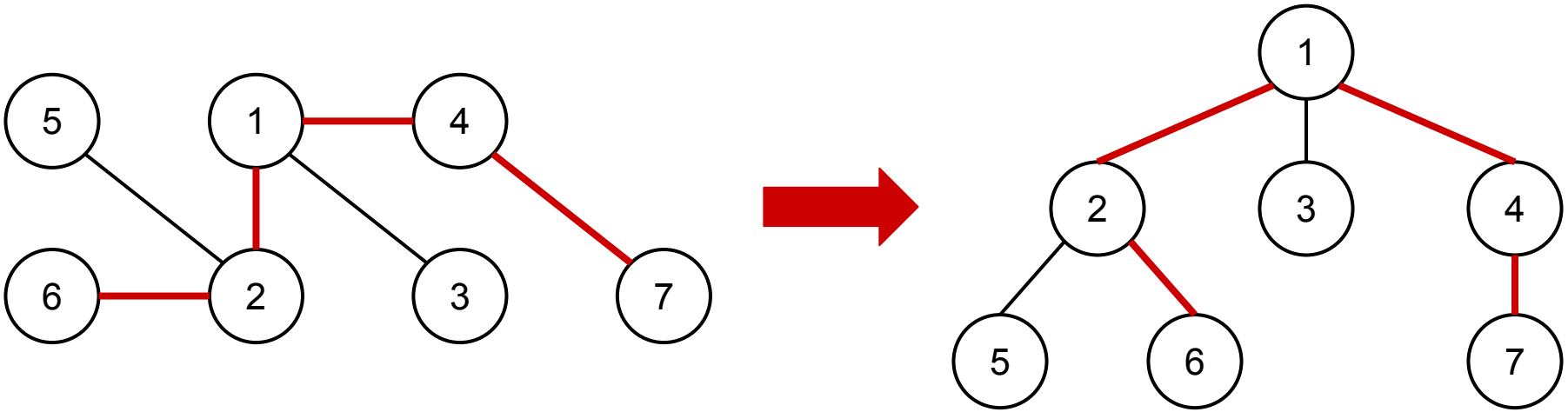
F - Longest Path on a Tree

- Mas como podemos calcular esse diâmetro máximo?

- Vamos utilizar **Programação Dinâmica!**

- Para tal, **escolhemos uma raiz para a árvore** a partir de um **nó arbitrário** e **resolvemos o problema separadamente para cada sub-árvore.**

F - Longest Path on a Tree



F - Longest Path on a Tree

- Com a configuração atual da árvore, podemos perceber que, para cada caminho em nossa árvore enraizada, temos um nó que é o **ponto mais alto**, as **raízes das sub-árvores**;

- Assim, podemos calcular, **para cada nó u**, o comprimento do maior caminho que tem como **ponto mais alto o nó u**.

F - Longest Path on a Tree

- Portanto, a estratégia é, sabendo a altura das sub-árvores de cada filho, basta **selecionar as duas maiores alturas**, **somá-las** e **guardar a soma máxima**, representando o diâmetro máximo;

- Nossa **PD** será um **vetor das alturas**, **height[n]**, com **height[u] = altura da sub-árvore u, ou seja, o comprimento máximo do nó u para qualquer folha.**

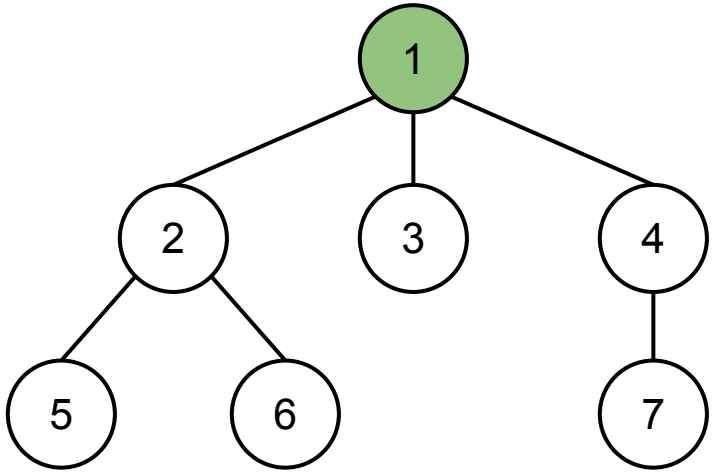
F - Longest Path on a Tree

- Recorrência:
    - Se u é nó **folha**:
        - height[u] = 1;
    - Senão, **∀ v | v é nó filho de u:**
        - height[u] = max(height[u], height[v] + 1)

F - Longest Path on a Tree

diâmetro = 0

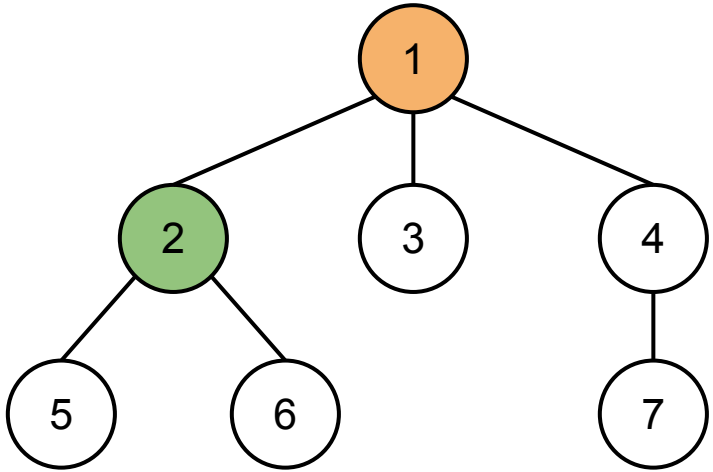
nó	height[nó]	h1	h2
1	1	0	0
2			
3			
4			
5			
6			
7			



F - Longest Path on a Tree

diâmetro = 0

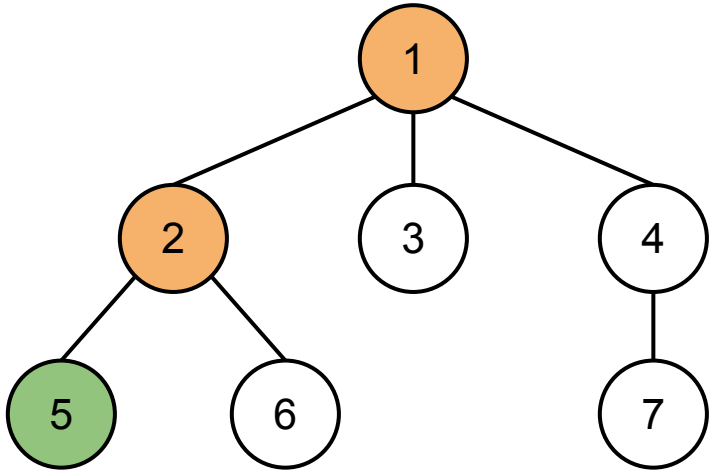
nó	height[nó]	h1	h2
1	1	0	0
2	1	0	0
3			
4			
5			
6			
7			



F - Longest Path on a Tree

diâmetro = 0

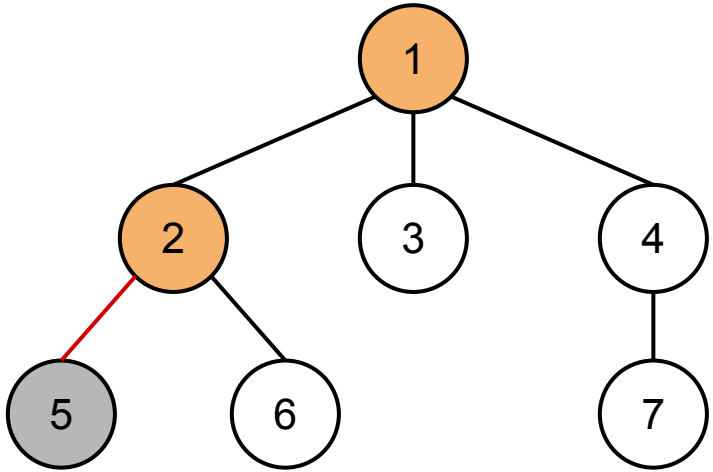
nó	height[nó]	h1	h2
1	1	0	0
2	1	0	0
3			
4			
5	1	0	0
6			
7			



F - Longest Path on a Tree

diâmetro = 0

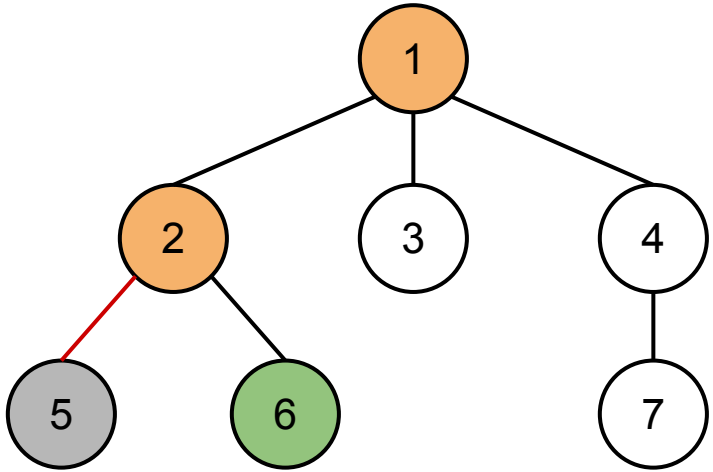
nó	height[nó]	h1	h2
1	1	0	0
2	1 + 1	1	0
3			
4			
5	1	0	0
6	1	0	0
7			



F - Longest Path on a Tree

diâmetro = 0

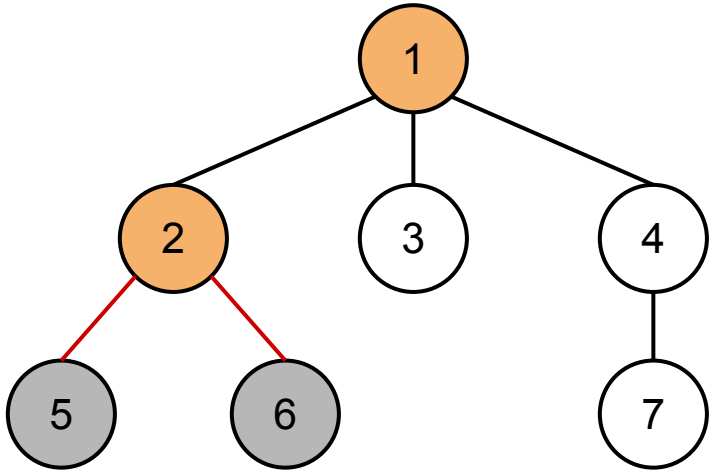
nó	height[nó]	h1	h2
1	1	0	0
2	2	1	0
3			
4			
5	1	0	0
6	1	0	0
7			



F - Longest Path on a Tree

diâmetro = 2

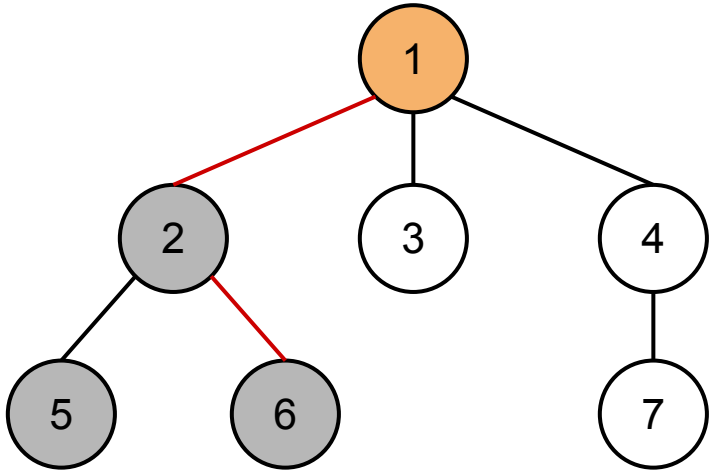
nó	height[nó]	h1	h2
1	1	0	0
2	2	1	0
3			
4			
5	1	0	0
6	1	0	0
7			



F - Longest Path on a Tree

diâmetro = 2

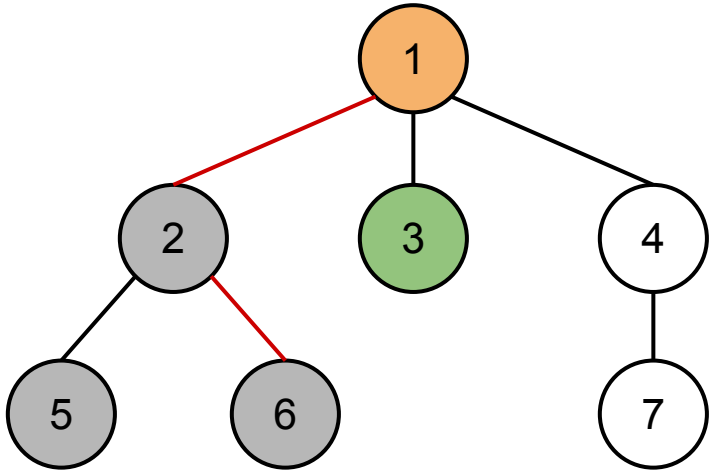
nó	height[nó]	h1	h2
1	3	2	0
2	2	1	1
3			
4			
5	1	0	0
6	1	0	0
7			



F - Longest Path on a Tree

diâmetro = 2

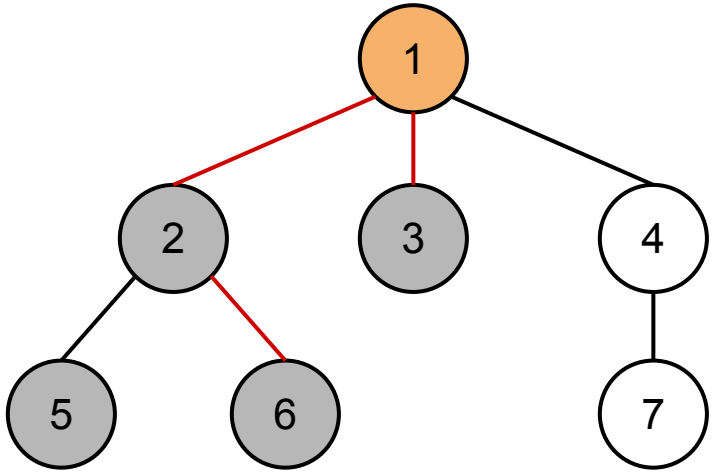
nó	height[nó]	h1	h2
1	3	2	0
2	2	1	1
3	1	0	0
4			
5	1	0	0
6	1	0	0
7			



F - Longest Path on a Tree

diâmetro = 3

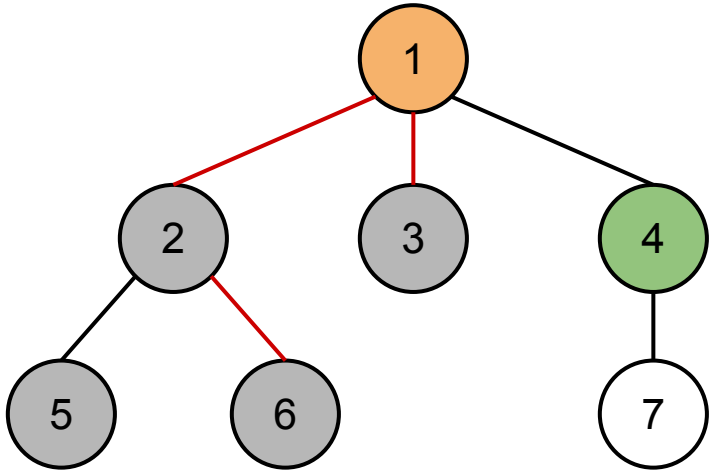
nó	height[nó]	h1	h2
1	3	2	1
2	2	1	1
3	1	0	0
4			
5	1	0	0
6	1	0	0
7			



F - Longest Path on a Tree

diâmetro = 3

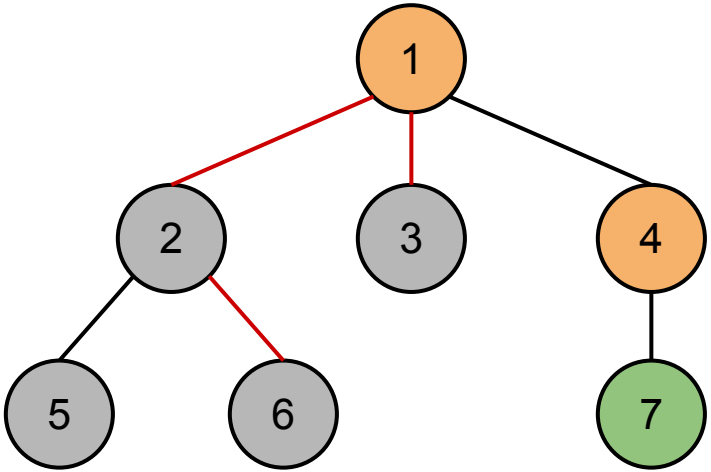
nó	height[nó]	h1	h2
1	3	2	1
2	2	1	1
3	1	0	0
4	1	0	0
5	1	0	0
6	1	0	0
7			



F - Longest Path on a Tree

diâmetro = 3

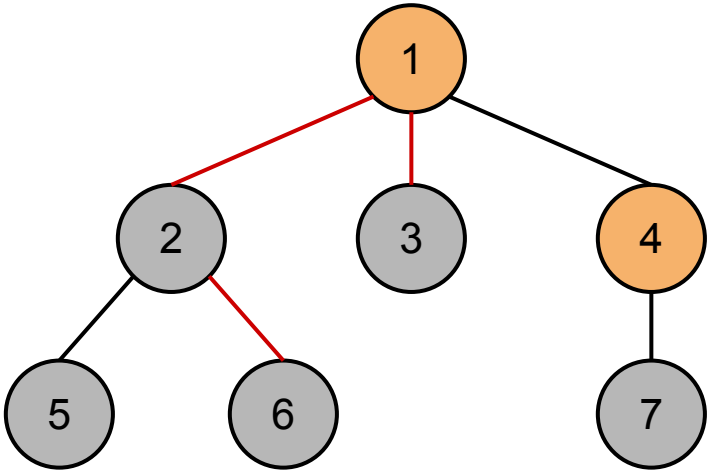
nó	height[nó]	h1	h2
1	3	2	1
2	2	1	1
3	1	0	0
4	1	0	0
5	1	0	0
6	1	0	0
7	1	0	0



F - Longest Path on a Tree

diâmetro = 3

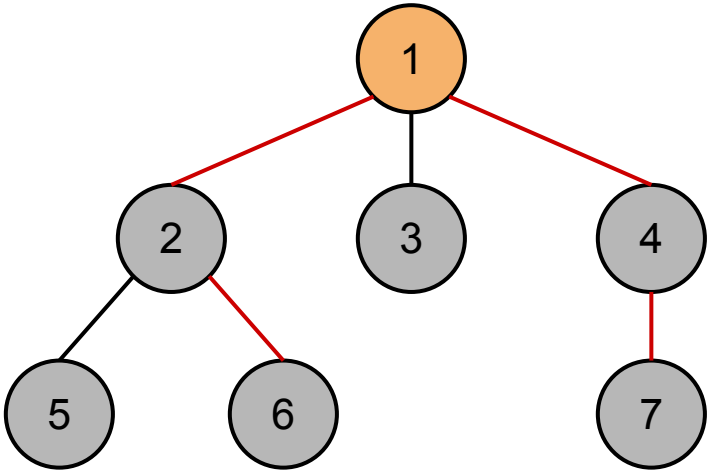
nó	height[nó]	h1	h2
1	3	2	1
2	2	1	1
3	1	0	0
4	2	1	0
5	1	0	0
6	1	0	0
7	1	0	0



F - Longest Path on a Tree

diâmetro = 4

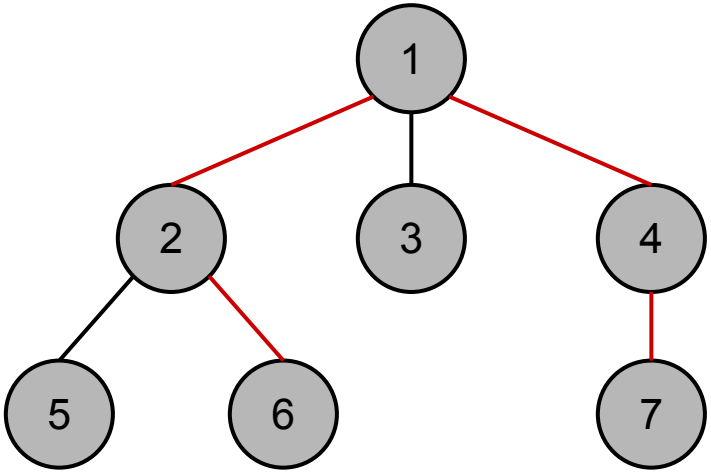
nó	height[nó]	h1	h2
1	3	2	2
2	2	1	1
3	1	0	0
4	2	1	0
5	1	0	0
6	1	0	0
7	1	0	0



F - Longest Path on a Tree

diâmetro = 4

nó	height[nó]	h1	h2
1	3	2	2
2	2	1	1
3	1	0	0
4	2	1	0
5	1	0	0
6	1	0	0
7	1	0	0



F - Longest path in a tree

```
int main() {
    cin >> n;
    adj = vector<vi>(n + 1);

    for (ll i = 0; i < n - 1; i++) {
        ll u, v;
        cin >> u >> v;
        add_edge(u, v);
    }

    height = vi(n + 1, 1);
    diameter = 0;

    dfs(1, -1);

    cout << diameter << "\n";

    return 0;
}
```

F - Longest path in a tree

```
int main() {
    cin >> n;
    adj = vector<vi>(n + 1);

    for (ll i = 0; i < n - 1; i++) {
        ll u, v;
        cin >> u >> v;
        add_edge(u, v);
    }

    height = vi(n + 1, 1);
    diameter = 0;

    dfs(1, -1);

    cout << diameter << "\n";

    return 0;
}
```

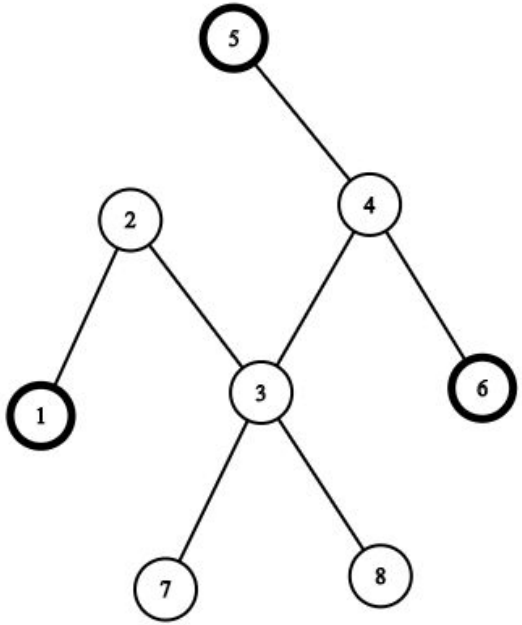
```
ll dfs(ll u, ll parent) {
    ll h1, h2;
    h1 = h2 = 0;

    for (auto v : adj[u]) {
        if (v != parent) {
            height[u] = max(height[u], dfs(v, u) +
1);

            if (height[v] > h2) {
                h2 = height[v];

                if (h2 > h1)
                    swap(h1, h2);
            }
        }
    }

    diameter = max(diameter, h1 + h2);

    return height[u];
}
```

F - Longest path in a tree

```
int main() {
    cin >> n;
    adj = vector<vi>(n + 1);

    for (ll i = 0; i < n - 1; i++) {
        ll u, v;
        cin >> u >> v;
        add_edge(u, v);
    }

    height = vi(n + 1, 1);
    diameter = 0;

    dfs(1, -1);

    cout << diameter << "\n";

    return 0;
}
```

```
ll dfs(ll u, ll parent) {
    ll h1, h2;
    h1 = h2 = 0;

    for (auto v : adj[u]) {
        if (v != parent) {
            height[u] = max(height[u], dfs(v, u) +
1);

            if (height[v] > h2) {
                h2 = height[v];

                if (h2 > h1)
                    swap(h1, h2);
            }
        }
    }

    diameter = max(diameter, h1 + h2);
    return height[u];
}
```

E - Three Paths on a Tree



Dada uma árvore, queremos encontrar três caminhos de forma a conseguir passar pelo maior número de vértices possíveis.

E - Three Paths on a Tree

Se liga aí na revisão! **heh**

E - Three Paths on a Tree

Pseudocódigo:

1. Calcular o diâmetro da árvore salvando as posições que fazer parte do mesmo
    a. Se o diâmetro for igual ao tamanho da árvore:

    cout << diâmetro << diam[0] << diam[1] << diam.back();

    b. Senão, passamos por cada um dos nós pertencentes ao diâmetro e calculamos a distância de cada uma delas em relação aos seus filhos, ou seja até a folha, salvando a folha

    cout << diametro + max_dist << diam[0] << max_folha << diam.back();

E - Three Paths on a Tree

Pseudocódigo:

1. Calcular o diâmetro da árvore salvando as posições que fazer parte do mesmo
    a. Se o diâmetro for igual ao tamanho da árvore:

    cout << diâmetro << diam[0] << diam[1] << diam.back();

    b. Senão, passamos por cada um dos nós pertencentes ao di
    calculamos a distância de cada uma delas em relação ac
    filhos, ou seja até a folha, salvando a folha

    cout << diametro + max_dist << diam[0] << max_folha << diam.