

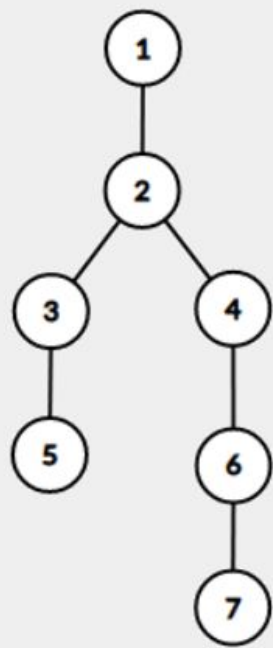
LCA - Lowest Common Ancestor

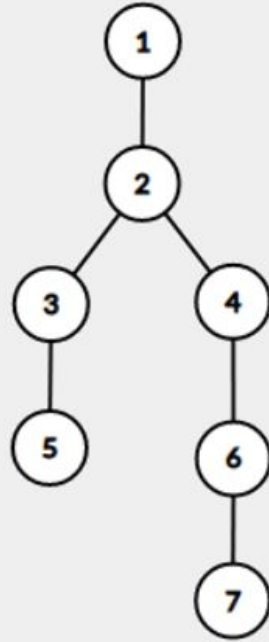
Ancestral comum mais próximo

Pedro Bortolli
MaratonUSP

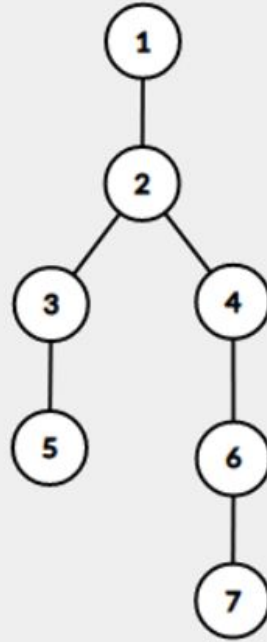
Problema

- Sejam **a** e **b** dois vértices de uma árvore enraizada
- Determinar o ancestral comum entre eles que possui a maior profundidade na árvore



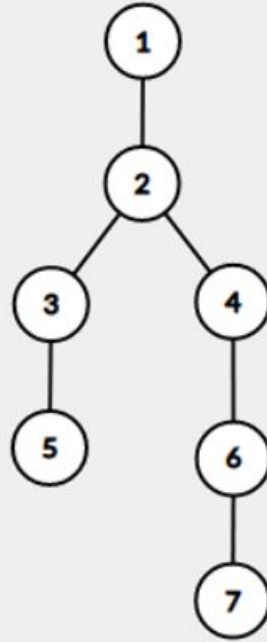


Ancestrais de 5: $\{3, 2, 1\}$



Ancestrais de 5: {3, 2, 1}

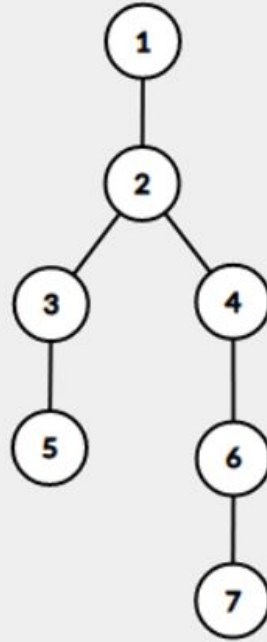
Ancestrais de 7: {6, 4, 2, 1}



Ancestrais de 5: {3, 2, 1}

Ancestrais de 7: {6, 4, 2, 1}

Ancestrais em comum: {2, 1}



Ancestrais de 5: {3, 2, 1}

Ancestrais de 7: {6, 4, 2, 1}

Ancestrais em comum: {2, 1}

LCA: 2

Algoritmo básico

- Fazer com que **a** e **b** estejam no mesmo nível (mesma profundidade)
- Escolha o vértice mais profundo e caminhe com ele em direção a raiz da árvore até que sua profundidade seja igual a do outro nó
- Agora, caminhe com os dois vértices simultaneamente para a raiz até que se encontrem em um vértice **c** - este é o LCA

Código

```
1  se profundidade[a] < profundidade[b]:  
2      troca(a, b)  
3  enquanto profundidade[a] > profundidade[b]:  
4      a = pai[a]  
5  enquanto a != b:  
6      a = pai[a]  
7      b = pai[b]  
8  devolve a
```

Código

```
1  se profundidade[a] < profundidade[b]:  
2      troca(a, b)  
3  enquanto profundidade[a] > profundidade[b]:  
4      a = pai[a]  
5  enquanto a != b:  
6      a = pai[a]  
7      b = pai[b]  
8  devolve a
```

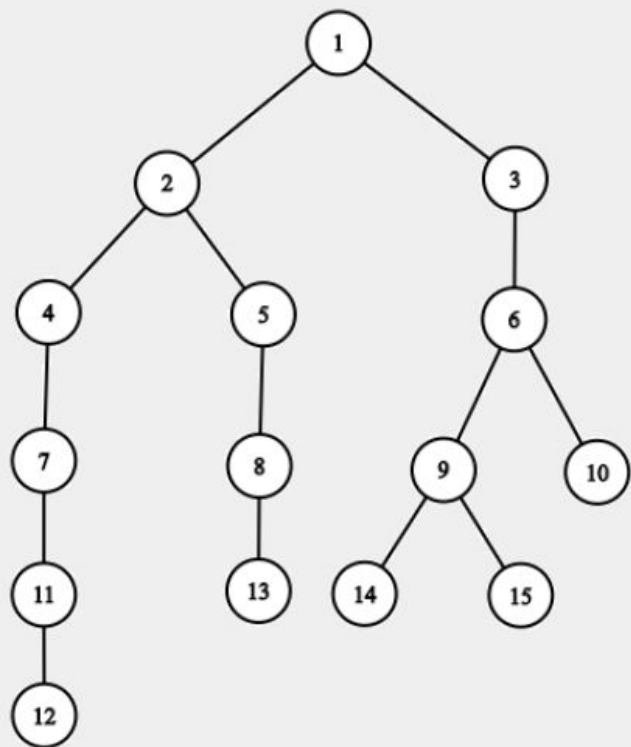
Complexidade de tempo: **$O(n)$**

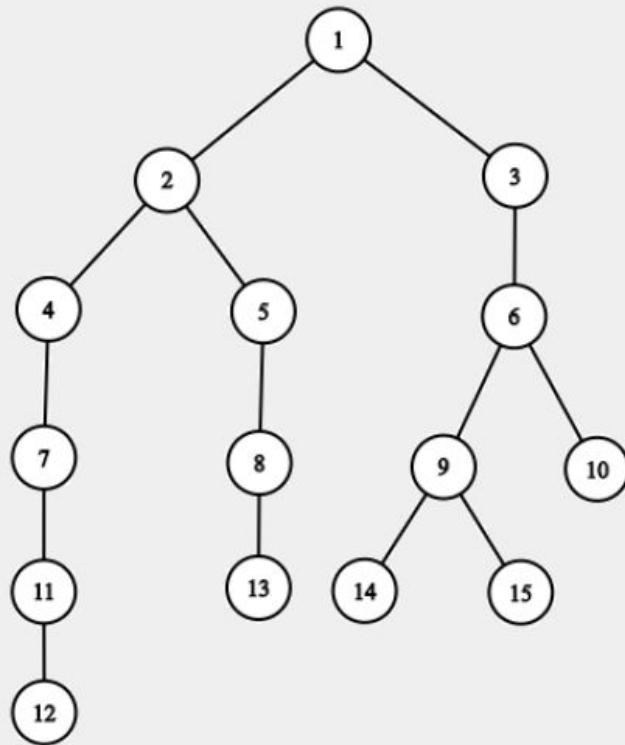
Solução com programação dinâmica

- Mesma ideia: vamos caminhar em direção à raiz com os vértices **a** e **b**
- Não vamos pular de “pai em pai”, podendo visitar todos os **n** vértices da árvore até encontrar o LCA
- Pulos de tamanho 2^i
- Usaremos o fato de que qualquer número pode ser representado por uma soma de potências de dois

Pré processamento

- Seja **ancestral(2^i)** de **x** o ancestral situado **2^i** níveis acima do vértice **x**
- Seja **L** o maior inteiro tal que **$2^L < \text{altura da árvore}$**
- Para cada vértice da árvore, calcularemos quem é seu **ancestral(2^i)** para todo **i** entre 0 e **L**



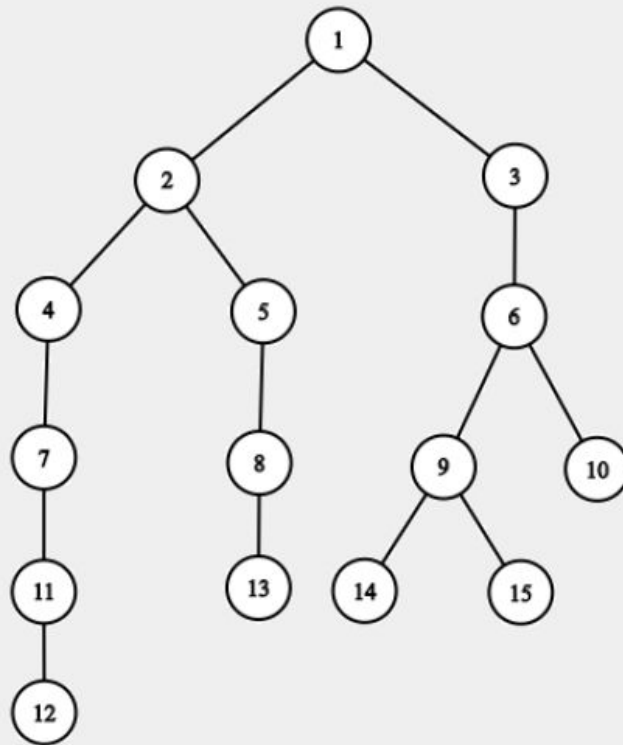


Ancestrais de 12:

$$\text{anc}(2^0) = 11$$

$$\text{anc}(2^1) = 7$$

$$\text{anc}(2^2) = 2$$



Ancestrais de 12:

$$\text{anc}(2^0) = 11$$

$$\text{anc}(2^1) = 7$$

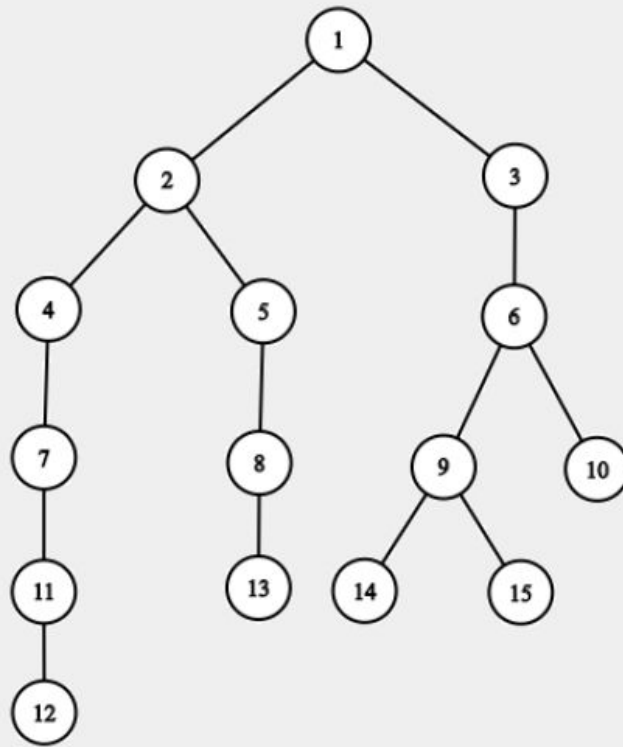
$$\text{anc}(2^2) = 2$$

Ancestrais de 11:

$$\text{anc}(2^0) = 7$$

$$\text{anc}(2^1) = 4$$

$$\text{anc}(2^2) = 1$$



Ancestrais de 12:	$\text{anc}(2^0) = 11$	$\text{anc}(2^1) = 7$	$\text{anc}(2^2) = 2$
Ancestrais de 11:	$\text{anc}(2^0) = 7$	$\text{anc}(2^1) = 4$	$\text{anc}(2^2) = 1$
Ancestrais de 15:	$\text{anc}(2^0) = 9$	$\text{anc}(2^1) = 6$	$\text{anc}(2^2) = 1$

Código para computar os ancestrais

Queremos computar a seguinte recorrência, para todo vértice v :

$\text{anc}(2^i)$ de $v = \text{pai}[v]$, caso $i = 0$
 $\text{anc}(2^i)$ de $v = \text{anc}(2^{i-1})$ do $\text{anc}(2^{i-1})$ de v , caso contrário

```
1  Para todo vértice  $v$ :  
2       $\text{Anc}[v][0] = \text{pai}[v]$   
3  Para cada inteiro  $i$  de 1 até  $M$ :  
4      Para todo vértice  $v$ :  
5           $\text{Anc}[v][i] = \text{anc}[\text{anc}[v][i-1]][i-1]$ 
```

Código para computar os ancestrais

Queremos computar a seguinte recorrência, para todo vértice v :

$\text{anc}(2^i)$ de $v = \text{pai}[v]$, caso $i = 0$
 $\text{anc}(2^i)$ de $v = \text{anc}(2^{(i-1)})$ do $\text{anc}(2^{(i-1)})$ de v , caso contrário

```
1  Para todo vértice  $v$ :  
2       $\text{Anc}[v][0] = \text{pai}[v]$   
3  Para cada inteiro  $i$  de 1 até  $L$ :  
4      Para todo vértice  $v$ :  
5           $\text{Anc}[v][i] = \text{anc}[\text{anc}[v][i-1]][i-1]$ 
```

Como $L = \log(n)$, então...

Complexidade (tempo/espço): $O(n \log(n))$

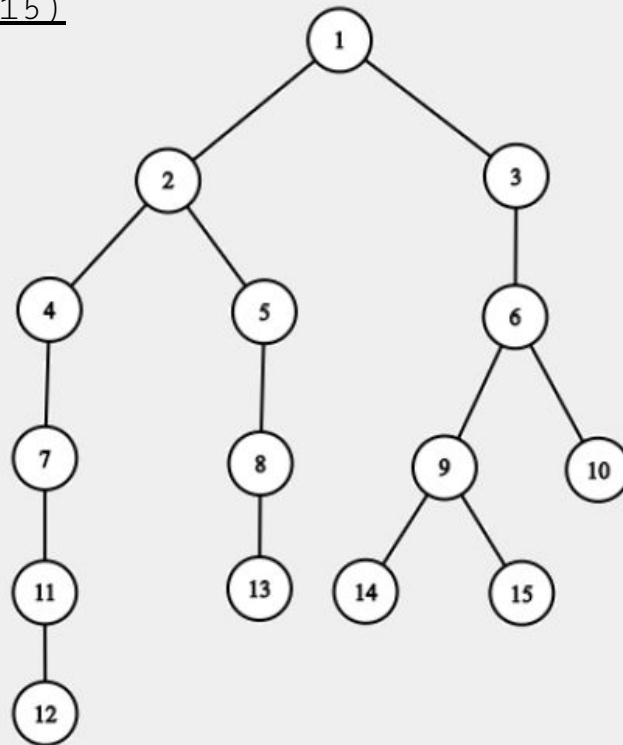
Algoritmo

- Faça com que a profundidade dos dois vértices **a** e **b** sejam iguais
- Determinar qual é o ancestral **mais** profundo de **a** cujo pai direto é ancestral de ambos **a** e **b**: este pai direto é o LCA

Voltando ao exemplo

Queremos o LCA(12, 15)

- 12 é 1 nível mais profundo do que 15
- Então subimos para o seu pai: 11
- Tentamos subir $2^2 = 4$ níveis
 - $\text{anc}(2^2)$ de 11 = 1
 - $\text{anc}(2^2)$ de 15 = 1
 - 1 **pode ou não** ser o LCA. Seguimos.
- Tentamos subir $2^1 = 2$ níveis
 - $\text{anc}(2^1)$ de 11 = 4
 - $\text{anc}(2^1)$ de 15 = 6
 - Como $4 \neq 6$, subimos para estes nós
- Tentamos subir $2^0 = 1$ nível
 - $\text{anc}(2^0)$ de 4 = 2
 - $\text{anc}(2^0)$ de 6 = 3
 - Como $2 \neq 3$, subimos para estes nós
- **Pai de 2 (ou pai de 3) = 1 é o LCA**



Código para encontrar o LCA

```
1  se profundidade[a] < profundidade[b]:
2      troca(a, b)
3  para cada inteiro i entre M e 0:
4      se profundidade[b] - 2i >= profundidade[a]:
5          b = anc[b][i]
6  se a == b:
7      devolve a
8  para cada inteiro i entre M e 0:
9      se anc[a][i] != anc[b][i]:
10         a = anc[a][i]
11         b = anc[b][i]
12  devolve pai[a]
```

Código para encontrar o LCA

```
1  se profundidade[a] < profundidade[b]:
2      troca(a, b)
3  para cada inteiro i entre M e 0:
4      se profundidade[b] - 2^i >= profundidade[a]:
5          b = anc[b][i]
6  se a == b:
7      devolve a
8  para cada inteiro i entre M e 0:
9      se anc[a][i] != anc[b][i]:
10         a = anc[a][i]
11         b = anc[b][i]
12  devolve pai[a]
```

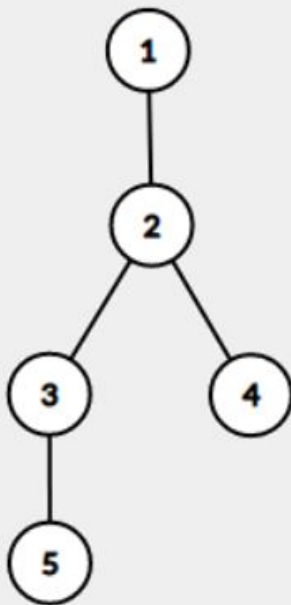
Complexidade de tempo por consulta: $O(M) = O(\log(n))$

Otimização com árvore de segmentos

- Imagine que temos todos vértices do caminho que parte do vértice **a** e chega em **b**
- Este caminho passa pelo LCA, que é o vértice com menor profundidade dentre os nós
- Assim, a busca pelo LCA pode se resumir à busca do menor elemento numa sequência de números: **árvore de segmentos!**

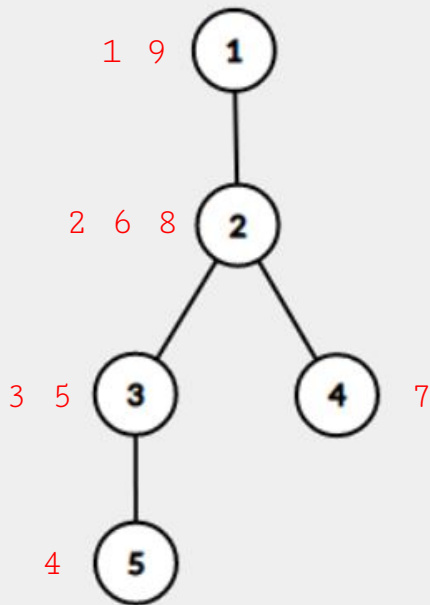
Passeio de Euler

- **DFS:** para cada vértice, anotamos o tempo (qual iteração está sendo feita) em que visitamos cada um de seus filhos e o tempo em que a chamada da DFS de um vértice é encerrada



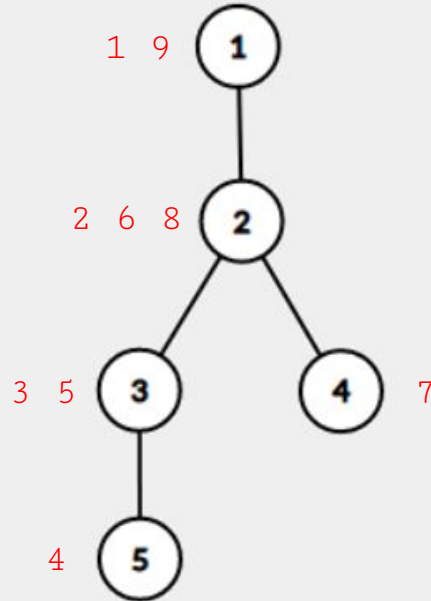
Passeio de Euler

- **DFS:** para cada vértice, anotamos o tempo (qual iteração está sendo feita) em que visitamos cada um de seus filhos e o tempo em que a chamada da DFS de um vértice é encerrada



Passeio de Euler

Índice	1	2	3	4	5	6	7	8	9
Vértice	1	2	3	5	3	2	4	2	1
Profundidade	1	2	3	4	3	2	3	2	1



Passeio de Euler

Índice	1	2	3	4	5	6	7	8	9
Vértice	1	2	3	5	3	2	4	2	1
Profundidade	1	2	3	4	3	2	3	2	1

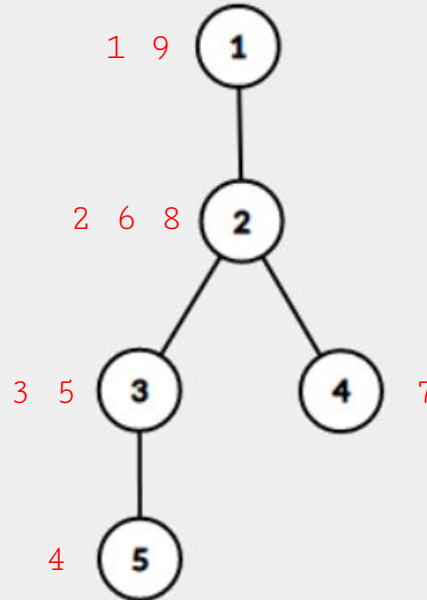
Para encontrar o LCA entre **a** e **b** agora, basta:

- Obter **qualquer** índice de **a** e **qualquer** índice de **b**
- Consultar o elemento de menor profundidade presente no intervalo contínuo **[índice(a), índice(b)]**

Passeio de Euler

Índice	1	2	3	4	5	6	7	8	9
Vértice	1	2	3	5	3	2	4	2	1
Profundidade	1	2	3	4	3	2	3	2	1

LCA(5, 4)

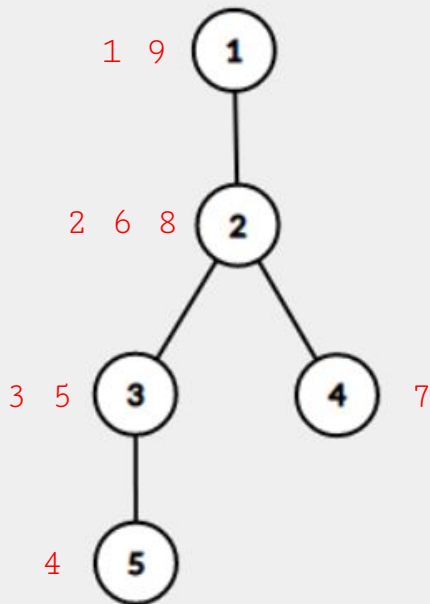


Passeio de Euler

Índice	1	2	3	4	5	6	7	8	9
Vértice	1	2	3	5	3	2	4	2	1
Profundidade	1	2	3	4	3	2	3	2	1

LCA(5, 4)

- Vértice 5: índice 4
- Vértice 4: índice 7
- Profundidades:
 - [4, 3, 2, 3]
- Menor profundidade = 2
- Correspondente ao vértice **2** = **LCA**



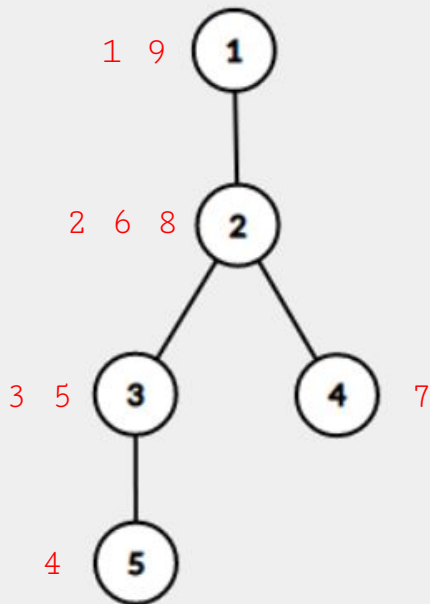
Passeio de Euler

Índice	1	2	3	4	5	6	7	8	9
Vértice	1	2	3	5	3	2	4	2	1
Profundidade	1	2	3	4	3	2	3	2	1

$LCA(5, 4)$

$LCA(1, 3)$

- Vértice 5: índice 4
- Vértice 4: índice 7
- Profundidades:
 - [4, 3, 2, 3]
- Menor profundidade = 2
- Correspondente ao vértice 2 = **LCA**

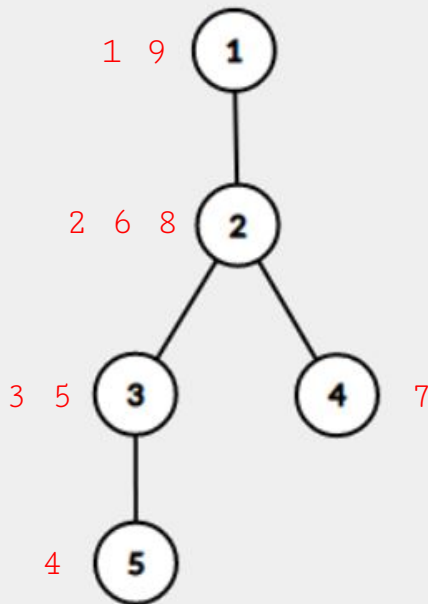


Passeio de Euler

Índice	1	2	3	4	5	6	7	8	9
Vértice	1	2	3	5	3	2	4	2	1
Profundidade	1	2	3	4	3	2	3	2	1

LCA(5, 4)

- Vértice 5: índice 4
- Vértice 4: índice 7
- Profundidades:
 - [4, 3, 2, 3]
- Menor profundidade = 2
- Correspondente ao vértice **2** = **LCA**



LCA(1, 3)

- Vértice 1: índice 9
- Vértice 3: índice 5
- Profundidades:
 - [3, 2, 3, 2, 1]
- Menor profundidade = 1
- Correspondente ao vértice **1** = **LCA**

Código do Passeio de Euler

- Precisamos de uma variável global **contador** inicializada com 1

```
1  função Euler(vertice):  
2  para cada filho de vertice:  
3      arvoreSeg[contador] = {profundidade[vertice], vertice}  
4      contador += 1  
5      Euler(filho)  
6  indice[vertice] = contador  
7  arvoreSeg[contador] = {profundidade[vertice], vertice}  
8  contador += 1
```

Código do Passeio de Euler

- Precisamos de uma variável global **contador** inicializada com 1

```
1  função Euler(vertice):
2  para cada filho de vertice:
3      arvoreSeg[contador] = {profundidade[vertice], vertice}
4      contador += 1
5      Euler(filho)
6  indice[vertice] = contador
7  arvoreSeg[contador] = {profundidade[vertice], vertice}
8  contador += 1
```

Complexidade de tempo/espço: **$O(n+m)$**

Código para encontrar o LCA

- Assumimos que temos uma função de construção da árvore de segmentos pronta, além de uma função que consulta qual é o elemento mínimo em um intervalo.

```
1  função LCA(a, b):  
2      esquerda = indice[a]  
3      direita = indice[b]  
4      se esquerda > direita:  
5          troca(esquerda, direita)  
6  devolve consulta(esquerda, direita)
```

Código para encontrar o LCA

- Assumimos que temos uma função de construção da árvore de segmentos pronta, além de uma função que consulta qual é o elemento mínimo em um intervalo.

```
1  função LCA(a, b):  
2      esquerda = indice[a]  
3      direita = indice[b]  
4      se esquerda > direita:  
5          troca(esquerda, direita)  
6  devolve consulta(esquerda, direita)
```

Complexidade de tempo por consulta: $O(\log n)$

Obrigado!

Monografia sobre o problema do LCA:

<https://linux.ime.usp.br/~bortolli/mac0499/>