

# «Map» e «Unordered\_Map»

**Kaio Christaldo**  
**Fabricio Matsunaga**

**Template** ◀Map▶

# Apresentação Problema Motivador

beecrowd | 1260

## Espécies de Madeira

Por Gordon V. Cormack 🇨🇦 Canadá

Timelimit: 3

Coníferas e folhosas (softwoods e hardwoods) são dois grandes grupos de vegetais produtores de madeira. As folhosas são aquele grupo de árvores que têm folhas largas, produzem uma fruta ou castanha e geralmente ficam dormentes no inverno.

Os climas temperados da América produzem florestas com centenas de espécies de madeira de lei - árvores que compartilham certas características biológicas. Embora o carvalho, bordo e cereja sejam tipos de árvores de madeira de lei, são espécies diferentes. Juntas, todas as espécies de madeira folhosas representam 40 por cento das árvores nos Estados Unidos.



Por outro lado, as madeiras macias (Softwoods) ou coníferas, chamadas "cone-bearing", são resinosas amplamente disponíveis EUA. Incluem cedro, abeto, cicuta, pinho, abeto vermelho e cipreste. Em uma casa, os resinosos são utilizados principalmente como madeira de viga estrutural, mas também podem ser utilizadas em algumas aplicações decorativas.

Usando tecnologia de imagem por satélite, o Departamento de Recursos Naturais elaborou um inventário de todas as árvores de um local específico em um determinado dia. Você deverá calcular a fração da população de cada árvore representada por cada uma das espécies.

### Entrada

A entrada possui vários casos de teste. A primeira linha de entrada contém um inteiro **N** que indica o número de casos de teste, seguido por uma linha em branco. Cada caso de teste consiste de uma lista com a espécie de cada árvore observada pelo satélite, uma árvore por linha. Nenhum nome de espécie é superior a 30 caracteres. Não existem mais de 10.000 espécies e não mais de 1.000.000 árvores. Há uma linha em branco entre cada caso de teste consecutivo.

### Saída

Para cada caso de teste imprima o nome de cada espécie representada na população, em ordem alfabética, seguida pelo percentual da população que representa, com 4 casas decimais. Imprima uma linha em branco entre dois conjuntos de dados consecutivos.

**1260 – Espécies de Madeira**

# Template <Map>

- **Definição:** `map` é um **container associativo da STL** que armazena elementos únicos em **ordem ordenada**.
- **Implementação:** Árvore Red-Black (**RB-Tree**) – operações em  **$O(\log n)$** .
- **Cabeçalho necessário:**



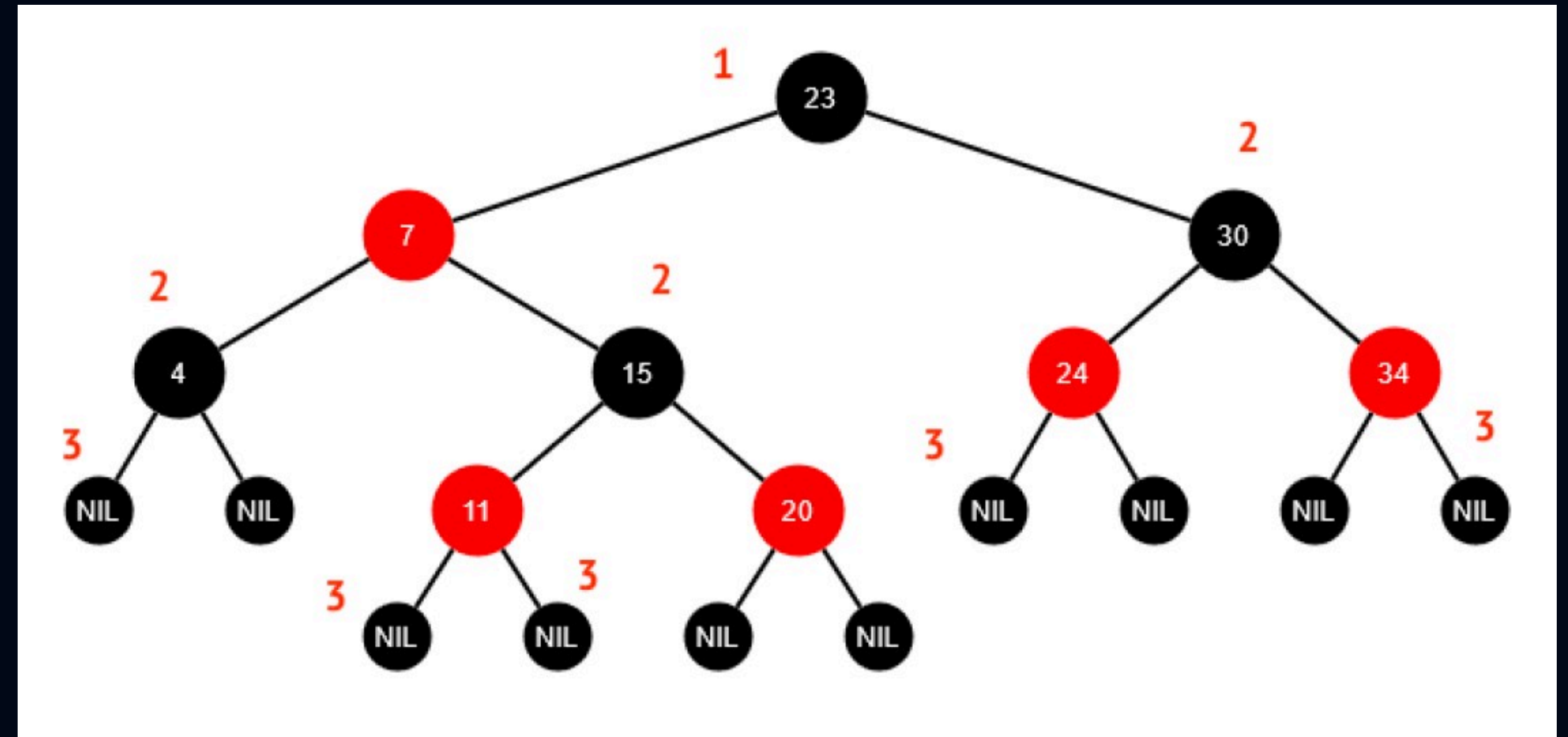
```
1 #include <map>
2 using namespace std;
```

# Template <Map>

- **Árvore Red-Black (RB-Tree)**

Segue cinco regras para manter o balanceamento:

- Todo nó é **vermelho** ou **preto**.
- A **raiz** sempre é **preta**.
- Nós **vermelhos** não podem ter **filhos vermelhos** (não há nós consecutivos vermelhos).
- **Todos** os **caminhos** da **raiz** até as **folhas** têm o **mesmo número** de nós **pretos**.
- **Novos nós** são sempre **inseridos** como **vermelhos**.



# Template <Map>

## Operações no Map

Principais Operações Usadas:

- **`m[key] = value`** – Insere ou atualiza o valor associado à chave –  **$O(\log n)$**
- **`m.at(key)`** – Acessa o valor da chave (lança exceção se não existir) –  **$O(\log n)$**
- **`m.find(key)`** – Retorna um iterador para a chave, ou `m.end()` se não existir –  **$O(\log n)$**
- **`m.count(key)`** – Retorna 1 se a chave existe, senão 0 –  **$O(\log n)$**
- **`m.insert({k, v})`** – Insere um par chave-valor (não sobrescreve se já existir) –  **$O(\log n)$**
- **`m.erase(key)`** – Remove a chave (e seu valor) do map –  **$O(\log n)$**
- **`m.size()`** – Retorna o número de elementos no map –  **$O(1)$**

# Template <Map>

## Operações no Map

Principais Operações Usadas:

- **m.clear()** – Remove todos os elementos do map –  **$O(n)$**
- **m.empty()** – Retorna true se o map estiver vazio –  **$O(1)$**
- **m.begin() / m.end()** – Retorna um iterador para a chave, ou m.end() se não existir –  **$O(1)$**
- **m.lower\_bound(key)** – Retorna iterador para o primeiro elemento com chave  $\geq$  key –  **$O(\log n)$**
- **m.upper\_bound(key)** – Retorna iterador para o primeiro elemento com chave  $>$  key –  **$O(\log n)$**
- **<greater>** – Map em ordem decrescente
- **Iteração com for** – Percorre todos os pares em ordem crescente das chaves –  **$O(n)$**



# Template <Map>

- **m[key] = value**



```
1 map<string, int> nome_idade; // Declarando map de string e inteiro
2
3 nome_idade["Lucas"] = 20; // Inserindo - Ordem Crescente
4 nome_idade["Ana"] = 19;
5
6 for (auto [nome, idade] : nome_idade) // percorrendo map
7     cout << nome << " " << idade << "\n"; // imprimindo nome e idade
8
```



# Template <Map>

- `at(key)` ou `m[key]`



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"Ana", 30}, {"Beto", 26};;
3
4 cout << idade.at("Ana") << "\n"; // saida: 30
5 cout << idade["Beto"] << "\n"; // saida: 26
6
```

# Template <Map>

- find(key)



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"Ana", 30}, {"Beto", 26}};
3
4 auto busca = idade.find("Kaio"); // Retorna iterador
5
6 cout << (busca != idade.end() // if Ternário
7         ? (busca->first + " - " + to_string(busca->second) + " anos, Encontrado!")
8         : "Não encontrado!")
9 << "\n";
```

# Template <Map>

- **count(key)**



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"Ana", 30}, {"Beto", 26};;
3
4 cout << (idade.count("Ana") // if Ternário
5      ? "Encontrado!" : "Não encontrado!") << "\n";
```

# Template <Map>

- insert({key, value})



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"Ana", 30}, {"Beto", 26}};
3
4 idade.insert({"Ana", 29}); // Não sobrescreve (não será inserido)
5 idade.insert({"João", 20}); // será inserido (Não existe João no map)
6
7 for (auto [n, i] : idade)
8     cout << n << " " << i << "\n";
```

# Template <Map>

- `erase(key)`



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"Ana", 30}, {"Beto", 26}};
3
4 idade.erase("Ana"); // Removeu a Ana
5
6 for (auto [n, i] : idade)
7     cout << n << " " << i << "\n";
```

# Template <Map>

- size()



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"Ana", 30}, {"Beto", 26};;
3
4 cout << idade.size() << "\n"; // Número de elementos
5
6 for (auto [n, i] : idade)
7     cout << n << " " << i << "\n";
```

# Template <Map>

- `clear()`



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"Ana", 30}, {"Beto", 26}};
3
4 idade.clear(); // Remove todos os elementos
5
6 for (auto [n, i] : idade)
7     cout << n << " " << i << "\n";
8
```



# Template <Map>

- `empty()`



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"Ana", 30}, {"Beto", 26};;
3
4 if (idade.empty()) cout << "Map Vazio!\n";
5 else cout << "Map não Vazio!\n";
```

# Template <Map>

- `begin()` e `end()` e outros...



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"João", 70}, {"Ana", 30}, {"Beto", 26};;
3
4 cout << "Primeiro Elemento: " << idade.begin()->first << " " << idade.begin()->second << "\n";
5
6 // usar --idade.end() ou prev(idade.end())
7
8 cout << "Ultimo Elemento: " << (--idade.end())->first << " " << (--idade.end())->second << "\n";
```

# Template <Map>


- `begin()` e `end()` e outros...



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"João", 70}, {"Ana", 30}, {"Beto", 26};;
3
4 cout << "Primeiro Elemento: " << idade.begin()->first << " " << idade.begin()->second << "\n";
5
6 // usar --idade.end() ou prev(idade.end())
7
8 cout << "Ultimo Elemento: " << (--idade.end())->first << " " << (--idade.end())->second << "\n";
```

# Template <Map>

- `lower_bound()` e `upper_bound()`



```
1 // Declarando map e Preenchendo valores
2 map<string, int> idade = {"João", 70}, {"Ana", 30}, {"Beto", 26};;
3
4 auto busca_low = idade.lower_bound("João"); // Primeiro elemento >= "João"
5 auto busca_upp = idade.upper_bound("João"); // Primeiro elemento > "João"
6
7 if (busca_low != idade.end()) cout << "lower_bound: " << busca_low->first << " - " << busca_low->second << "\n";
8 else cout << "lower_bound não encontrou nada\n";
9
10 if (busca_upp != idade.end()) cout << "upper_bound: " << busca_low->first << " - " << busca_low->second << "\n";
11 else cout << "upper_bound não encontrou nada\n";
```

# Template <Map>


- <greater>



```
1 // Declarando map e Preenchendo valores em ordem decrescente
2 map<string, int, greater<string>> idade = {
3     {"João", 70},
4     {"Ana", 30},
5     {"Beto", 26}
6 };
7
8 for (auto [nome, id] : idade) {
9     cout << nome << " " << id << "\n";
10 }
11
```

# Template <Map>

- Percorrendo map



```
1 // Declarando map e Preenchendo valores em ordem decrescente
2 map<string, int> idade = {"João", 70}, {"Ana", 30}, {"Beto", 26}};
3
4 // Percorrendo com Iterator
5 for (auto it = idade.begin(); it != idade.end(); ++it) {
6     cout << it->first << " " << it->second << "\n";
7 }
8
9 // For Each
10 for (auto [nome, id] : idade) {
11     cout << nome << " " << id << "\n";
12 }
```

# Template <Map>

## Vantagens

- Não precisa se preocupar com duplicatas
- Elementos ordenados automaticamente
- Complexidade  $O(\log n)$

## Desvantagens

- Mais lento que `unordered_map`
- Maior uso de memória que um vetor
- Inserção lenta em massa



# Template `<Map>`

## Alternativas

- `unordered_map` – Quando não precisa de ordenação e quer mais performance ( $O(1)$  médio).
- `vector<pair<>>` – Para listas pequenas com pesquisa linear e ordenação manual.
- `multimap` – Quando precisa **repetir chaves**.

# Resolução do Problema Motivador

## 1260 – Espécies de Madeira

### Dicas:

- Usar um `map<string, int>` para guardar o Nome e quantidade da especie
- Pegar a linha usando um `getline()`
- Tratar linhas Vazias
- Imprimir Resultado, Nome da Especia, Quantidade da especie / quantidade total de especies lidas nos casos de teste
- use `setPrecision()` e `fixed`


A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

**Template** ◀ **Unordered\_Map** ▶

# Apresentação Problema Motivador

beecrowd | 1218

Getline Three - Calçados

Por Neilor Tonin, URI  Brasil

Timelimit: 1

Agora que Mangojata resolveu alguns problemas que utilizavam getline, acha que está apta a dar um passo adiante. Ela está prestes a fazer um novo programa para auxiliar a sua irmã, Overlaine. Overlaine é vendedora de calçados e por um acidente, misturou todos os pares de calçados que tinha para vender. Ela quer informar um número qualquer  $N$  e contar quantos calçados de uma determinada caixa são deste tamanho ( $N$ ). O problema é que Overlaine não tem a menor idéia de quantos calçados existem em cada caixa. A única coisa que sabe é que cada calçado pode ter numeração de 20 a 44, podendo ser masculino ou feminino.

### Entrada

A entrada contém vários casos de teste e termina com EOF (Fim de Arquivo). Cada caso de teste consiste de duas linhas de entrada. A primeira linha contém uma numeração  $N$  ( $20 \leq N \leq 44$ ) de calçado que Overlaine informa e a segunda linha contém o número de cada par que está dentro da caixa seguido de **M** ou **F** indicando se o par é de calçado Masculino ou Feminino.

### Saída

Para cada caso de teste imprima quatro linhas, conforme exemplo abaixo. A primeira linha deve apresentar a mensagem "Caso n:", onde  $n$  é o número do caso de teste. A segunda linha deve informar quantos pares da caixa de calçados são iguais ao número que Overlaine quer encontrar, com mensagem correspondente. Seguem duas linhas com a quantidade respectiva de calçados Femininos (F) e Masculinos (M), com mensagem correspondente.

Imprima uma linha em branco **entre** as saídas de dois casos de teste consecutivos.

## Exemplo de Entrada

```
23
23 F 28 M 23 F 40 M 36 F 23 M 23 F 24 M 23 M
28
22 M 23 F 28 M 32 F
```

## Exemplo de Saída

```
Caso 1:
Pares Iguais: 5
F: 3
M: 2

Caso 2:
Pares Iguais: 1
F: 0
M: 1
```

## 1218 – Getline Three – Calçados

## **Template** <Unordered\_Map>

**Em C++, the STL unordered\_map é um container associativo desordenado que provê as funcionalidades de uma estrutura de mapa desordenado ou dicionário de dados. (armazena elementos em uma combinação de elementos chave e elementos mapeados)**

**Em contraste com o map, a ordem de chaves no map desordenado é indefinida.**

# Template <Unordened\_Map> Declaração



```
1  #include <unordered_map>
```



```
1  unordered_map<tipo_chave, tipo_valor> ump;
```



```
1  // create an unordered_map with integer key and value
2  unordered_map<int, int> ump_integer;
3
4  // create an unordered_map with string key and int value
5  unordered_map<string, int> ump_string;
6
```

# Template <Unordered\_Map> Inicialização



```
1 // Initializer List
2 unordered_map<string, int> unordered_map1 = {
3     {"One", 1},
4     {"Two", 2},
5     {"Three", 3}
6 };
7
8 // loop across the unordered map
9 // display the key-value pairs
10 for(const auto& key_value: unordered_map1) {
11     string key = key_value.first;
12     int value = key_value.second;
13
14     cout << key << " - " << value << endl;
15 }
```



# Template **Unordered\_Map** Metodos

**insert()**-insere um ou mais pares de chave-valor.

**count()**-retorna 1 se a chave existe e 0 se não.

**find()**-retorna o iterator para o elemento com a chave especificada.

**at()**-retorna o elemento na chave especificada.

**size()**-retorna o numero de elementos.

**empty()**-retorna true se o unordered map está vazio.

**erase()**-remove elementos com a chave especificada.

**clear()**-remove todos os elementos.

# Template `<Unordered_Map>` Inserção de Elementos



```
1 // insert key-value pair {"One", 1}
2 unordered_map1["One"] = 1;
3
4 // insert a pair {"Two", 2}
5 unordered_map1.insert({"Two", 2});
6
7 // insert two pairs {"Three", 3}, {"Four", 4}
8 unordered_map1.insert({"Three", 3}, {"Four", 4});
9
```

# Template `<Unordered_Map>` Acessando Valores



```
1 unordered_map<string, string> capital_city {  
2     {"Nepal", "Kathmandu"},  
3     {"India", "New Delhi"},  
4     {"Australia", "Canberra"}  
5 };  
6  
7 cout << "Capital of Nepal is " << capital_city["Nepal"] << endl;  
8 cout << "Capital of Australia is " << capital_city.at("Australia");
```

# Template <Unordered\_Map> Alterando Valores

```
1 unordered_map<string, string> capital_city {
2     {"India", "Calcutta"},
3     {"Pakistan", "Karachi"},
4 };
5
6 cout << "Old Capitals:" << endl;
7 cout << "India : " << capital_city["India"] << endl;
8 cout << "Pakistan : " << capital_city["Pakistan"];
9
10 // change value for "India" using []
11 capital_city["India"] = "New Delhi";
12
13 // change value for "Pakistan" using at()
14 capital_city.at("Pakistan") = "Islamabad";
15
16
17 cout << "\n\nNew Capitals:" << endl;
18 cout << "India : " << capital_city["India"] << endl;
19 cout << "Pakistan : " << capital_city["Pakistan"];
```

# Template <Unordered\_Map> Removendo Valores

```
1 unordered_map<int, string> student {
2     {111, "John"},
3     {132, "Mark"},
4     {143, "Chris"}
5 };
6
7 cout << "Initial Unordered Map:\n";
8 display_unordered_map(student);
9
10 // remove element with key: 143
11 student.erase(143);
12
13
14 cout << "\nFinal Unordered Map: \n";
15 display_unordered_map(student);
16
```

```
1 // utility function to print unordered_map elements
2 void display_unordered_map(const unordered_map<int, string> &umap){
3
4     for(const auto& key_value: umap) {
5         int key = key_value.first;
6         string value = key_value.second;
7
8         cout << key << " - " << value << endl;
9     }
10 }
```

# Template <Unordered\_Map> Verifica se Existe Chave



```
1 unordered_map<int, string> student{
2     {111, "John"},
3     {132, "Mark"},
4     {143, "Chris"}
5 };
```



```
1 cout << "Using count():" << endl;
2 cout << "Does id " << 1433 << " exist? ";
3
4 // count() returns 0 if the key doesn't exist
5 if (student.count(1433)) {
6
7     cout << "Yes";
8 }
9 else {
10     cout << "No";
11 }
```



```
1 cout << "Using find():" << endl;
2 cout << "Does id " << 143 << " exist? ";
3
4 // find() returns student.end() if the key is not found
5 if (student.find(143) != student.end()) {
6
7     cout << "Yes";
8 }
9 else {
10     cout << "No";
11 }
```

# Resolução do Problema Motivador

A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊



# Lista de Exercícios

1260 – Espécies de Madeira

1218 – Getline Three – Calçados

1609 – Contando Carneirinho

2951 – O Retorno do Rei

1255 – Frequência de Letras

1551 – Frase Completa

1318 – Bilhetes Falsos



Se tiver alguma dúvida ou dificuldade na resolução de algum exercício, sinta-se à vontade para perguntar! 😊

# Referências

**[1] JAVA RUSH. Diagrama de Red-Black Tree** [imagem]. Disponível em: <https://cdn.javarush.com/images/article/9a5b5d15-c32b-4b6f-9f8e-b1d12908379c/1080.jpeg>. Acesso em: 03 abr. 2025.

**[2] PROGRAMIZ. C++ Unordered Set.** Disponível em: <https://www.programiz.com/cpp-programming/unordered-set>. Acesso em: 03 abr. 2025.

**[3] PROGRAMIZ. Hash Table.** Disponível em: <https://www.programiz.com/dsa/hash-table>. Acesso em: 03 abr. 2025.

**[4] GLASIELLY DEMORI. Árvore Rubro-Negra (Aula 11) – Inserção.** YouTube, 10 out. 2023. Disponível em: <https://www.youtube.com/watch?v=vSAE4O2zpkY>. Acesso em: 4 abr. 2025.