HASHING NA STL

Prof. Juliano Foleis

22 de maio, 2025

INTRODUÇÃO

- Vamos ver como usar as implementações de hashing da STL
- Containers da STL baseados em **tabelas de dispersão (hash tables)**
- Operações em tempo constante (na média):
  - inserção
  - busca
  - remoção

UNORDERED_SET

- Conjunto não ordenado
- Armazena apenas chaves
- Armazena elementos únicos
- Implementado como uma tabela *hash*
- Suporta iteradores
  - Permite percorrer os elementos com *range-based for loops*

UNORDERED_SET: INSTANCIAÇÃO

```cpp
#include <unordered_set>
#include <iostream>
std::unordered_set<int> numeros;
```

- Tipo do elemento: `int`
- `unordered_set` é um template
- Pode ser usado com qualquer tipo que suporte operações de comparação

UNORDERED_SET: INSERÇÃO

```cpp
#include <unordered_set>
#include <iostream>
std::unordered_set<int> numeros;
numeros.insert(42);
numeros.insert(7);
numeros.insert(42); // Ignorado
```

- Cada elemento é único
- `insert` ignora duplicatas

UNORDERED_SET: BUSCA

```cpp
if (numeros.find(42) != numeros.end()) {
    std::cout << "42 está no conjunto\n";
}
```

- `find` retorna um iterador
- Se não encontrar, retorna `end()`

UNORDERED_SET: REMOÇÃO

```
numeros.erase(7);
```

- Remove o valor se estiver presente
- Não faz nada se o valor não existir

UNORDERED_SET: CONTAGEM

```cpp
if (numeros.count(42) > 0) {
    std::cout << "42 está presente\n";
}
```

- `count` retorna 0 ou 1
- Útil para verificações rápidas

UNORDERED_SET: ITERAÇÃO

```cpp
for (const auto& num : numeros) {
    std::cout << num << " ";
}
```

- Itera sobre os elementos
- Ordem não garantida

UNORDERED_MAP

- Mapeamento chave-valor
- Chaves únicas
- Implementado como uma tabela *hash*
- Suporta iteradores
  - Permite percorrer os pares chave-valor

UNORDERED_MAP: INSTANCIAÇÃO

```cpp
#include <unordered_map>
#include <iostream>
std::unordered_map<std::string, int> idade;
```

- Tipo da Chave: `std::string`
- Tipo do Valor: `int`
- `unordered_map` é um template
- Pode ser usado com qualquer tipo de chave que suporte operações de comparação

UNORDERED_MAP: INSERÇÃO

```cpp
#include <unordered_map>

std::unordered_map<std::string, int> idade;
idade.insert({"Alice", 30});
idade.insert({"Bob", 25});
```

- Chave associada a um valor
- Duplicatas de chave são ignoradas

UNORDERED_MAP: OPERADOR []

```
idade["Carol"] = 20;
```

- Cria nova entrada se a chave não existir
- Atualiza valor se a chave já existir

UNORDERED_MAP: ACESSO SEGURO

```
std::cout << idade.at("Bob") << "\n";
```

- `at` lança exceção se a chave não existir
- Útil para evitar inserção acidental

UNORDERED_MAP: REMOÇÃO

```
idade.erase("Alice");
```

- Remove a entrada pela chave
- Ignora se a chave não estiver presente

UNORDERED_MAP: BUSCA

```
if (idade.find("Carol") != idade.end()) {
    std::cout << "Carol está no mapa\n";
}
```

- `find` retorna iterador ou `end()`

UNORDERED_MAP: CONTAGEM

```cpp
if (idade.count("Bob") > 0) {
    std::cout << "Bob encontrado\n";
}
```

- `count` retorna 0 ou 1 (para map)

UNORDERED_MAP: ITERAÇÃO

```cpp
for (const auto& par : idade) {
    std::cout << par.first << ": " << par.second << "\n";
}
```

- Itera sobre os pares chave-valor
- Ordem não garantida
- `par.first` é a chave
- `par.second` é o valor

UNORDERED_MAP: ITERAÇÃO COM *STRUCTURED BINDINGS*

```cpp
for (const auto& [chave, valor] : idade) {
    std::cout << chave << ": " << valor << "\n";
}
```

- Usando *structured bindings* do C++17
- Mais legível
- `chave` e `valor` são variáveis locais ao for

CONCLUSÃO

- `unordered_set` e `unordered_map` oferecem acesso eficiente
- Ideal para buscas rápidas por chave
- Baseados em hashing
- Operações principais:
  - `insert`, `erase`, `find`, `count`, `[]`, `at`

PERGUNTAS?