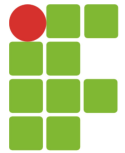
Mapeamentos

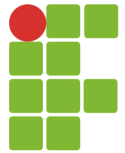
Introdução à Programação Competitiva

Prof. Daniel Saad Nogueira Nunes

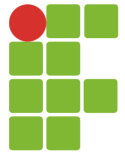
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

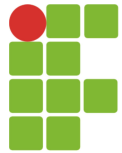


Sumário



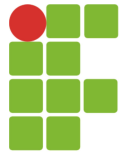
Mapeamento

- Um mapeamento é um tipo abstrato de dados que associa uma **chave** a um **valor**.
- Assemelha-se ao objeto matemático função, dado um x, obtém-se um $f(x)$ correspondente.
- Operações em mapeamentos envolvem:
  - Inserção, remoção e busca;

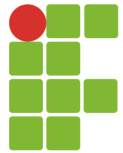


Mapeamentos

- Mapeamentos podem ser implementados de diversas formas em C++.
- A STL os implementa através de árvores autobalanceáveis ou tabelas de Hash.

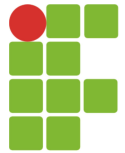


Sumário

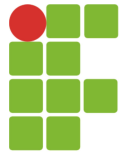


Mapeamentos

- A implementação `std::map` utiliza árvores rubro-negras para implementar mapeamentos.
- Cabeçalho: `<map>`.
- Essas árvores mantém os elementos de acordo com uma ordem e costumam oferecer tempo logarítmico em suas operações fundamentais.
- Conforme elementos são inseridos ou removidos, a árvore se autobalanceia para continuar fornecendo tempos competitivos.
- É possível recuperar todos os elementos em ordem crescente de acordo com a **chave**.

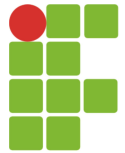


Sumário

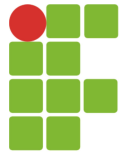


Declaração

- Para declarar um mapeamento em C++, utilizamos: `std::set<T,U> nome_variavel;`, em que `<T>` corresponde ao tipo da chave e `<U>` ao valor mapeado. Seria equivalente ao domínio e contradomínio de uma função.
- Exemplos:
  - `map<int,string> m1;`
  - `map<string,pessoa> m2;`
  - `map<int,pair<string,string>> m3;`
  - `map<int,vector<int>> m4;`

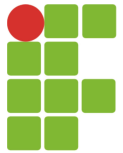


Sumário



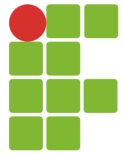
Inserção

- Para inserir em um mapeamento, utilizamos o método `insert`, em que passamos um par (chave,valor) a ser inserido.
- Ele retorna um par (iterador,booleano): `<iterator,bool>`.
  - Se o elemento não existe, o valor booleano é verdadeiro, e o iterador aponta para o elemento recém-inserido.
  - Se o elemento já existe, o valor booleano é falso, e o iterador aponta para o elemento já existente.
  - Tempo: $\Theta(\lg n)$.



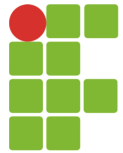
Inserção

- Adicionalmente, o `insert` aceita um iterador, além do elemento a ser inserido, como uma "dica".
- Esta dica tem como propósito aumentar a performance da inserção. Se o iterador apontar para o elemento que viria depois do elemento a ser inserido, a complexidade é $\Theta(1)$.
- Se a dica não apontar para o local citado, o desempenho da inserção não é melhorado.



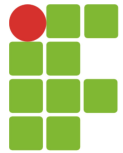
Inserção

- Também é possível inserir um intervalo de elementos apontados pelos iteradores de início e fim. Se o tamanho do intervalo é ℓ, a complexidade é $n \lg(\ell + n)$, em que n é o tamanho do mapeamento.



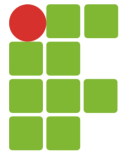
Inserção

```
1  #include <iostream>
2  #include <map>
3
4  int main() {
5      std::map<char, int> mymap;
6
7      // first insert function version (single parameter):
8      mymap.insert(std::pair<char, int>('a', 100));
9      mymap.insert(std::pair<char, int>('z', 200));
10
11     std::pair<std::map<char, int>::iterator, bool> ret;
12     ret = mymap.insert(std::pair<char, int>('z', 500));
13     if (ret.second == false) {
14         std::cout << "element 'z' already existed";
15         std::cout << " with a value of " << ret.first->second << '\n';
16     }
17
```



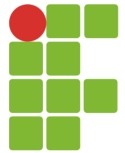
Inserção

```
18      // second insert function version (with hint position):
19      std::map<char, int>::iterator it = mymap.begin();
20      mymap.insert(it,
21                   std::pair<char, int>('b', 300)); // max efficiency inserting
22      mymap.insert(it,
23                   std::pair<char, int>('c', 400)); // no max efficiency inserting
24
```



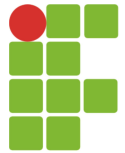
Inserção

```
25      // third insert function version (range insertion):
26      std::map<char, int> anothermap;
27      anothermap.insert(mymap.begin(), mymap.find('c'));
28
29      // showing contents:
30      std::cout << "mymap contains:\n";
31      for (it = mymap.begin(); it != mymap.end(); ++it)
32          std::cout << it->first << " => " << it->second << '\n';
33
34      std::cout << "anothermap contains:\n";
35      for (it = anothermap.begin(); it != anothermap.end(); ++it)
36          std::cout << it->first << " => " << it->second << '\n';
37
38      return 0;
39  }
```



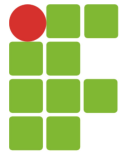
Inserção

- O método `insert_or_assign`, como o nome já diz, tem o propósito de inserir, caso o elemento não exista no mapeamento, ou modificar o elemento existente.



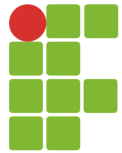
Inserção

```
1  #include <iostream>
2  #include <map>
3  #include <string>
4
5  auto print_node = [](const auto &node) {
6      std::cout << "[" << node.first << "] = " << node.second << '\n';
7  };
8
9  auto print_result = [](auto const &pair) {
10     std::cout << (pair.second ? "inserted: " : "assigned: ");
11     print_node(*pair.first);
12 };
13
```



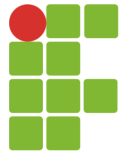
Inserção

```
14  int main() {
15      std::map<std::string, std::string> myMap;
16
17      print_result(myMap.insert_or_assign("a", "apple"));
18      print_result(myMap.insert_or_assign("b", "banana"));
19      print_result(myMap.insert_or_assign("c", "cherry"));
20      print_result(myMap.insert_or_assign("c", "clementine"));
21
22      for (const auto &node : myMap) {
23          print_node(node);
24      }
25  }
```



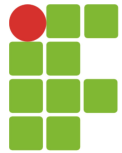
Inserção

- Também é possível utilizar o método `emplace`, que é similar ao `insert`, mas realiza a construção do objeto *in-place*.
- O `emplace_hint` funciona como o `emplace`, recebendo um parâmetro extra que é um iterador contendo a dica.
- Ambos estão disponíveis a partir do C++11.
- O método `try_emplace`, disponível a partir do C++17 funciona como o `emplace`, mas se o elemento estiver no mapeamento,ele não é alterado.



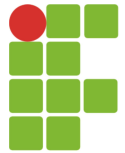
Inserção

- O operador `[]` pode ser utilizado para inserir ou acessar elementos de um mapeamento.

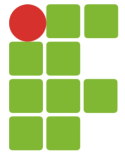


Inserção

```
1  #include <iostream>
2  #include <map>
3  #include <string>
4
5  int main() {
6      std::map<char, std::string> mymap;
7
8      mymap['a'] = "an element";
9      mymap['b'] = "another element";
10     mymap['c'] = mymap['b'];
11
12     std::cout << "mymap['a'] is " << mymap['a'] << '\n';
13     std::cout << "mymap['b'] is " << mymap['b'] << '\n';
14     std::cout << "mymap['c'] is " << mymap['c'] << '\n';
15     std::cout << "mymap['d'] is " << mymap['d'] << '\n';
16
17     std::cout << "mymap now contains " << mymap.size() << " elements.\n";
18     return 0;
19 }
```

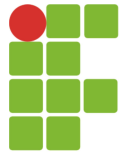


Sumário



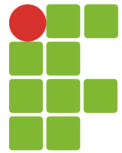
Busca

- Para buscar um elemento no mapeamento, utiliza-se o método `find`, que recebe a chave a ser buscada.
- Caso elemento esteja no mapeamento, o iterador para o elemento é retornado, caso contrário, retorna-se o fim do *container* (`end`).
- Tempo: $\Theta(\lg n)$.



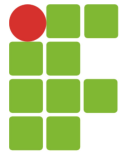
Busca

```
1  #include <iostream>
2  #include <map>
3
4  int main() {
5      std::map<int, double> mymap;
6
7      for (int i = 1; i <= 5; i++)
8          mymap.insert({i * 10, i * 2.5});
9
10     auto it = mymap.find(20);
11     if (it != mymap.end()) {
12         std::cout << "myset has key 20" << std::endl;
13     }
14     return 0;
15 }
```



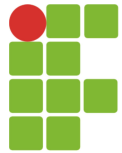
Busca

- Como mencionado anteriormente, o operador `[]` pode ser utilizado para acessar elementos.
- **Observação:** se o operador é utilizado sobre uma chave que não existe, o elemento é criado e inicializado com o valor padrão!



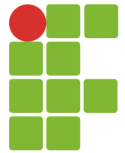
Busca

```
1  #include <iostream>
2  #include <map>
3  #include <string>
4
5  int main() {
6      std::map<char, std::string> mymap;
7
8      mymap['a'] = "an element";
9      mymap['b'] = "another element";
10     mymap['c'] = mymap['b'];
11
12     std::cout << "mymap['a'] is " << mymap['a'] << '\n';
13     std::cout << "mymap['b'] is " << mymap['b'] << '\n';
14     std::cout << "mymap['c'] is " << mymap['c'] << '\n';
15     std::cout << "mymap['d'] is " << mymap['d'] << '\n';
16
17     std::cout << "mymap now contains " << mymap.size() << " elements.\n";
18     return 0;
19 }
```



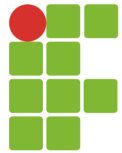
Busca

- A partir do C++20 é possível utilizar o método `contains` sobre uma chave para determinar se ela está ou não no mapeamento, isto é, o retorno é um `bool`.
- Tempo: $\Theta(\lg n)$.



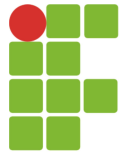
Busca

```cpp
#include <iostream>
#include <map>

int main() {
    std::map<int, char> example = {{1, 'a'}, {2, 'b'}};

    for (int x : {2, 5}) {
        if (example.contains(x)) {
            std::cout << x << ": Found\n";
        } else {
            std::cout << x << ": Not found\n";
        }
    }
}
```



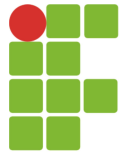
Busca

- Também é possível utilizar o `lower_bound`, `upper_bound` e `equal_range`, como feito sobre os *containers* do tipo `vector`.

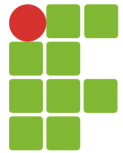


Busca

- O método `count` diz quantos elementos de determinada chave existem no mapeamento.
- Como o mapeamento não lida com repetições de chaves, a resposta sempre será 0 ou 1.
- Tempo $\Theta(\lg n)$.

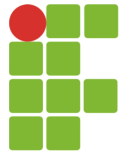


Sumário



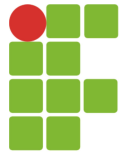
Remoção

- Para remover elementos do mapeamento, utiliza-se o método `erase`, que recebe o iterador para o elemento a ser removido ou a chave.
- Outra versão do `erase` recebe uma faixa de elementos a serem removidos através dos iteradores de início e fim (intervalo aberto no fim).
- No caso da versão que recebe o valor do elemento a ser removido, `erase` retorna 1 ou 0, isto é, o número de elementos removidos, a depender se o elemento estava ou não no mapeamento.



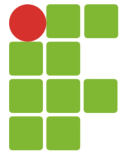
Remoção

- Remoção através de iterador: $\Theta(1)$ amortizado.
- Remoção por intervalo definido por dois iteradores: $\Theta(\lg n + s)$ em que s é o tamanho do intervalo.
- Remoção por valor: $\Theta(\lg n)$.

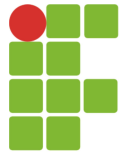


Remoção

```
1  #include <iostream>
2  #include <map>
3
4  int main() {
5      std::map<int, std::string> c = {{1, "one"},  {2, "two"},  {3, "three"},
6                                      {4, "four"}, {5, "five"}, {6, "six"}};
7
8      // erase all odd numbers from c
9      for (auto it = c.begin(); it != c.end();) {
10         if (it->first % 2 != 0)
11             it = c.erase(it);
12         else
13             ++it;
14     }
15
16     for (auto &p : c)
17         std::cout << p.second << ' ';
18 }
```

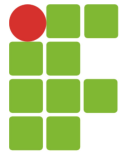


Sumário



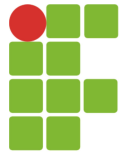
Alteração

- A partir do C++17 é possível, mediante o método `extract`, extrair um nó da árvore balanceada, modificá-lo e inseri-lo novamente no mapeamento.
- Tanto a chave quanto o valor mapeado podem ser modificados.
- O método funciona tanto através de um iterator para um nó, ou quanto por chave. No primeiro caso, leva-se tempo constante amortizado, enquanto no segundo, o tempo é $\Theta(\lg n)$.



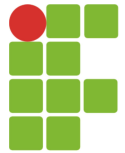
Alteração

```
1  #include <algorithm>
2  #include <iostream>
3  #include <map>
4  #include <string_view>
5
6  void print(std::string_view comment, const auto &data) {
7      std::cout << comment;
8      for (auto [k, v] : data)
9          std::cout << ' ' << k << '(' << v << ')';
10
11     std::cout << '\n';
12 }
13
14 int main() {
15     std::map<int, char> cont{{1, 'a'}, {2, 'b'}, {3, 'c'}};
16     print("Start:", cont);
17     // Extract node handle and change key
18     auto nh = cont.extract(1);
19     nh.key() = 4;
20     nh.mapped() = 'd';
21     print("After extract and before insert:", cont);
22     // Insert node handle back
23     cont.insert(std::move(nh));
24     print("End:", cont);
25 }
```

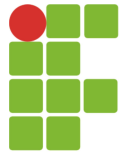


Alteração

```
1  #include <algorithm>
2  #include <iostream>
3  #include <map>
4  #include <string_view>
5
6  void print(std::string_view comment, const auto &data) {
7      std::cout << comment;
8      for (auto [k, v] : data)
9          std::cout << ' ' << k << '(' << v << ')';
10     std::cout << '\n';
11 }
12
13 int main() {
14     std::map<int, char> cont{{1, 'a'}, {2, 'b'}, {3, 'c'}};
15     print("Start:", cont);
16     // Extract node handle and change key
17     auto it = cont.begin();
18     it;
19     auto nh = cont.extract(it);
20     nh.key() = 4;
21     nh.mapped() = 'd';
22     print("After extract and before insert:", cont);
23     // Insert node handle back
24     cont.insert(std::move(nh));
25     print("End:", cont);
26 }
```

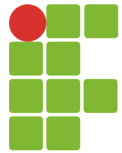


Sumário



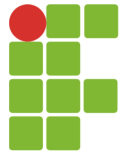
Limpeza

- O método `clear` é utilizado para limpar o mapeamento, tornando-o vazio.

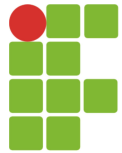


Limpeza

```
1  #include <algorithm>
2  #include <iostream>
3  #include <map>
4
5  int main() {
6      std::map<int, char> container{{1, 'x'}, {2, 'y'}, {3, 'z'}};
7
8      auto print = [](std::pair<const int, char> &n) {
9          std::cout << " " << n.first << '(' << n.second << ')';
10     };
11
12     std::cout << "Before clear:";
13     std::for_each(container.begin(), container.end(), print);
14     std::cout << "\nSize=" << container.size() << '\n';
15
16     std::cout << "Clear\n";
17     container.clear();
18
19     std::cout << "After clear:";
20     std::for_each(container.begin(), container.end(), print);
21     std::cout << "\nSize=" << container.size() << '\n';
22 }
```

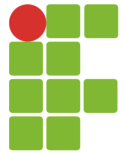


Sumário

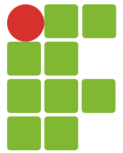


Métodos auxiliares

- `bool empty() const;` : retorna verdadeiro se o mapeamento está vazio.
- `size_t size() const;` : retorna o tamanho do mapeamento.

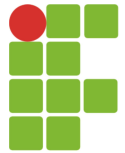


Sumário

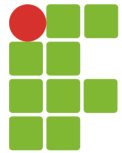


Multimapeamentos

- O C++ fornece ainda uma implementação para a abstração de multimapeamentos, em que elementos com chaves repetidas são permitidos.
- `std::multimap<T>`, em que `T` é o tipo.
- Os métodos são equivalentes, com algumas diferenças de tempo, como no `count`, que leva tempo proporcional à $\Theta(\lg n + occ)$ em que occ é a quantidade de elementos com o valor buscado.



Sumário



Referências

cplusplus, *cplusplus.com*, `https://cplusplus.com/`, Acessado em 12/2022.

cppreference, *cppreference.com*, `https://en.cppreference.com/`, Acessado em 12/2022.