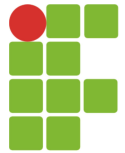
Conjuntos

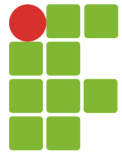
Introdução à Programação Competitiva

Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga

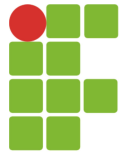


Sumário



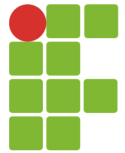
Conjuntos

- Um conjunto é um tipo abstrato de dados que armazena elementos únicos.
- Operações em conjuntos envolvem:
  - Inserção, remoção e busca;
  - União, interseção, diferença, diferença simétrica.

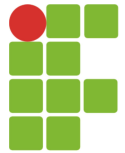


Conjuntos

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

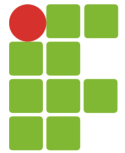


Sumário

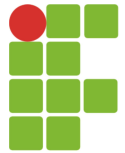


Conjuntos

- A implementação `std::set` utiliza árvores rubro-negras para implementar conjuntos.
- 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.

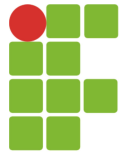


Sumário

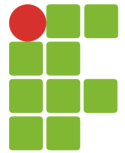


Declaração

- Para declarar um conjunto em C++, utilizamos: `std::set<T> nome_variavel;`, em que `<T>` corresponde ao tipo desejado.
- Exemplos:
  - `set<int> set_int;`
  - `set<pessoa> set_pessoa;`
  - `set<pair<int,string>> set_par;`
  - `set<vector<int>>> set_vetor;`

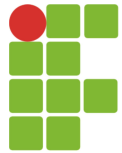


Sumário



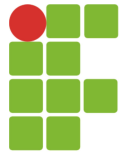
Inserção

- Para inserir em um conjunto, utilizamos o método `insert`.
- 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: $\Theta(\lg n)$.



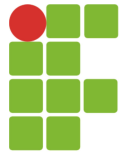
Inserção

```
1  #include <cassert>
2  #include <iostream>
3  #include <set>
4
5  int main() {
6      std::set<int> set;
7      auto result_1 = set.insert(3);
8      assert(result_1.first != set.end());
9      assert(*result_1.first == 3);
10     if (result_1.second)
11         std::cout << "insert done\n";
12     auto result_2 = set.insert(3);
13     assert(result_2.first == result_1.first);
14     assert(*result_2.first == 3);
15     if (!result_2.second)
16         std::cout << "no insertion\n";
17     return 0;
18 }
```

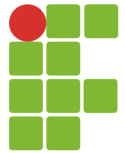


Inserção

- Também é possível utilizar o método `emplace`, que é similar ao `insert`, mas realiza a construção do objeto *in-place*.

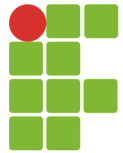


Sumário



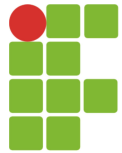
Busca

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



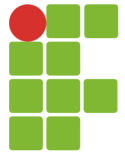
Busca

```
1  #include <iostream>
2  #include <set>
3
4  int main() {
5      std::set<int> myset;
6      std::set<int>::iterator it;
7
8      // set some initial values:
9      for (int i = 1; i <= 5; i++)
10         myset.insert(i * 10); // set: 10 20 30 40 50
11
12     it = myset.find(20);
13     if (it != myset.end()) {
14         std::cout << "myset has 20" << std::endl;
15     }
16     return 0;
17 }
```



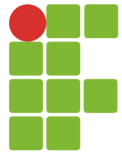
Busca

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



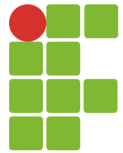
Busca

```cpp
#include <iostream>
#include <set>

int main() {
    std::set<int> example = {1, 2, 3, 4};

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



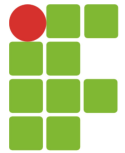
Busca

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

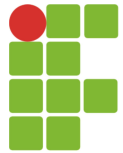


Busca

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

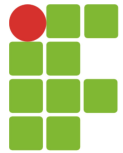


Sumário



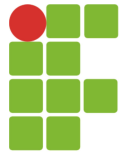
Remoção

- Para remover elementos do conjunto, utiliza-se o método `erase`, que recebe o iterador para o elemento a ser removido ou o valor do elemento a ser removido.
- 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 conjunto.



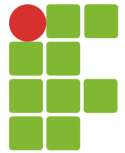
Remoção

- 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: $\Theta(\lg n)$.

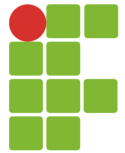


Remoção

```
1  #include <iostream>
2  #include <set>
3
4  int main() {
5      std::set<int> myset;
6      std::set<int>::iterator it;
7
8      // insert some values:
9      for (int i = 1; i < 10; i++)
10         myset.insert(i * 10); // 10 20 30 40 50 60 70 80 90
11
12     it = myset.begin();
13     ++it; // "it" points now to 20
14
15     myset.erase(it);
16
17     myset.erase(40);
18
19     it = myset.find(60);
20     myset.erase(it, myset.end());
21
22     std::cout << "myset contains:";
23     for (it = myset.begin(); it != myset.end(); ++it)
24         std::cout << ' ' << *it;
25     std::cout << '\n';
26
27     return 0;
28 }
```

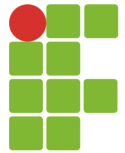


Sumário



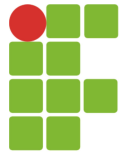
Alteração

- Até antes do C++17, não era possível alterar os dados de um conjunto, a não ser que o elemento fosse retirado, modificado e inserido novamente.
- A partir dessa versão é possível mediante o método `extract` extrair um nó da árvore balanceada, modificá-lo e inseri-lo novamente no conjunto.
- O método funciona tanto através de um iterator para um nó, ou quanto por valor. No primeiro caso, leva-se tempo constante amortizado, enquanto no segundo, o tempo é $\Theta(\lg n)$.



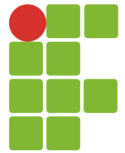
Alteração

```
1  #include <algorithm>
2  #include <iostream>
3  #include <set>
4
5  void print_set(std::set<int>& cont){
6      for(auto x: cont){
7          std::cout << x << " ";
8      }
9      std::cout << std::endl;
10 }
11
12
13 int main() {
14     std::set<int> cont{1, 2, 3};
15     print_set(cont);
16     // Extract node handle and change key
17     auto nh = cont.extract(1);
18     nh.value() = 4;
19
20     // Insert node handle back
21     cont.insert(std::move(nh));
22     print_set(cont);
23     return 0;
24 }
```

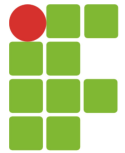


Alteração

```cpp
#include <algorithm>
#include <iostream>
#include <set>

void print_set(std::set<int>& cont){
    for(auto x: cont){
        std::cout << x << " ";
    }
    std::cout << std::endl;
}


int main() {
    std::set<int> cont{1, 2, 3};
    print_set(cont);
    // Extract node handle and change key
    auto it = cont.begin();
    it++;
    auto nh = cont.extract(it);
    nh.value() = 4;

    // Insert node handle back
    cont.insert(std::move(nh));
    print_set(cont);
    return 0;
}
```

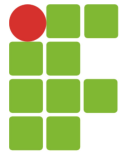


Sumário



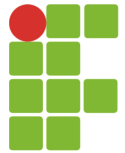
Limpeza

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

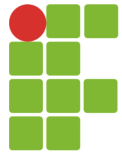


Limpeza

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

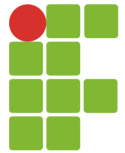


Sumário

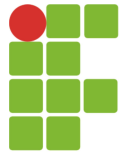


Métodos auxiliares

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

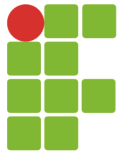


Sumário

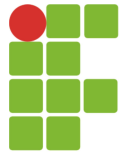


Multiconjuntos

- O C++ fornece ainda uma implementação para a abstração de multiconjuntos, em que elementos repetidos são permitidos.
- `std::multiset<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.