ESCOLA DE PRIMAVERA DA MARATONA DE PROGRAMAÇÃO

PROMOÇÃO:

CEFET-MG

UFMG

UFV

UNIFEI

USP Universidade de São Paulo

UFU

APOIO:

beecrowd

SBC Sociedade Brasileira de Computação

Grupo de Computação Competitiva

ESTRUTURA DE DADOS: SET

Por: *Wendell Reis Milani Matias*

CONTEÚDOS

01 – PROBLEMA MOTIVADOR - TROCA DE CARTAS

Alice e Beatriz colecionam cartas de Pokémon. As cartas são produzidas para um jogo que reproduz a batalha introduzida em um dos mais bem sucedidos jogos de videogame da história, mas Alice e Beatriz são muito pequenas para jogar, e estão interessadas apenas nas cartas propriamente ditas. Para facilitar, vamos considerar que cada carta possui um identificador único, que é um número inteiro.

Cada uma das duas meninas possui um conjunto de cartas e, como a maioria das garotas de sua idade, gostam de trocar entre si as cartas que têm. Elas obviamente não têm interesse em trocar cartas idênticas, que ambas possuem, e não querem receber cartas repetidas na troca.

01 – PROBLEMA MOTIVADOR - TROCA DE CARTAS

Além disso, as cartas serão trocadas em uma única operação de troca: Alice dá para Beatriz um sub-conjunto com N cartas distintas e recebe de volta um outro sub-conjunto com N cartas distintas.

As meninas querem saber qual é o número máximo de cartas que podem ser trocadas. Por exemplo, se Alice tem o conjunto de cartas {1, 1, 2, 3, 5, 7, 8, 8, 9, 15} e Beatriz o conjunto {2, 2, 2, 3, 4, 6, 10, 11, 11}, elas podem trocar entre si no máximo quatro cartas. Escreva um programa que, dados os conjuntos de cartas que Alice e Beatriz possuem, determine o número máximo de cartas que podem ser trocadas.

01 – PROBLEMA MOTIVADOR - TROCA DE CARTAS

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois números inteiros A e B, separados por um espaço em branco, indicando respectivamente o número de cartas que Alice e Beatriz possuem ($1 \leq A \leq 104$ e $1 \leq B \leq 104$). A segunda linha contém A números inteiros Xi, separados entre si por um espaço em branco, cada número indicando uma carta do conjunto de Alice ($1 \leq Xi \leq 105$). A terceira linha contém B números inteiros Yi, separados entre si por um espaço em branco, cada número indicando uma carta do conjunto de Beatriz ($1 \leq Yi \leq 105$). As cartas de Alice e Beatriz são apresentadas em ordem não decrescente.

O final da entrada é indicado por uma linha que contém apenas dois zeros, separados por um espaço em branco.

01 – PROBLEMA MOTIVADOR - TROCA DE CARTAS

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha, contendo um número inteiro, indicando o número máximo de cartas que Alice e Beatriz podem trocar entre si.

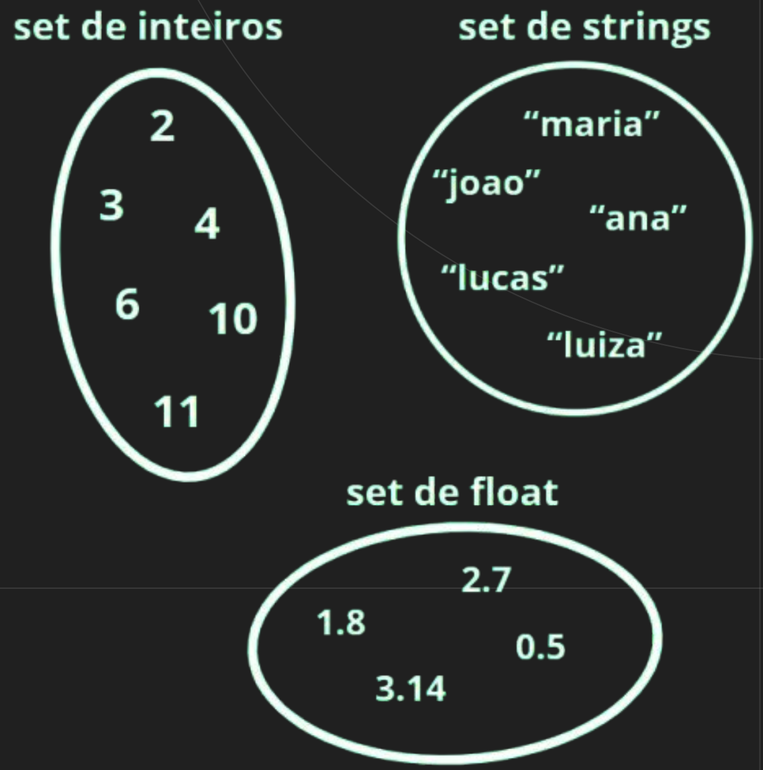
Sample Input	Sample Output
1 1 1000 1000 3 4 1 3 5 2 4 6 8 10 9 1 1 2 3 5 7 8 8 9 15 2 2 2 3 4 6 10 11 11 0 0	0 3 4

02 – INTRODUÇÃO

A classe de contêiner da Biblioteca Padrão C++ **set** oferecem uma estrutura que permitem armazenar elementos únicos (sem repetição) de forma ordenada. O valor de um elemento em um **set** não pode ser alterado diretamente. Em vez disso, você deve excluir valores antigos e inserir elementos com novos valores.

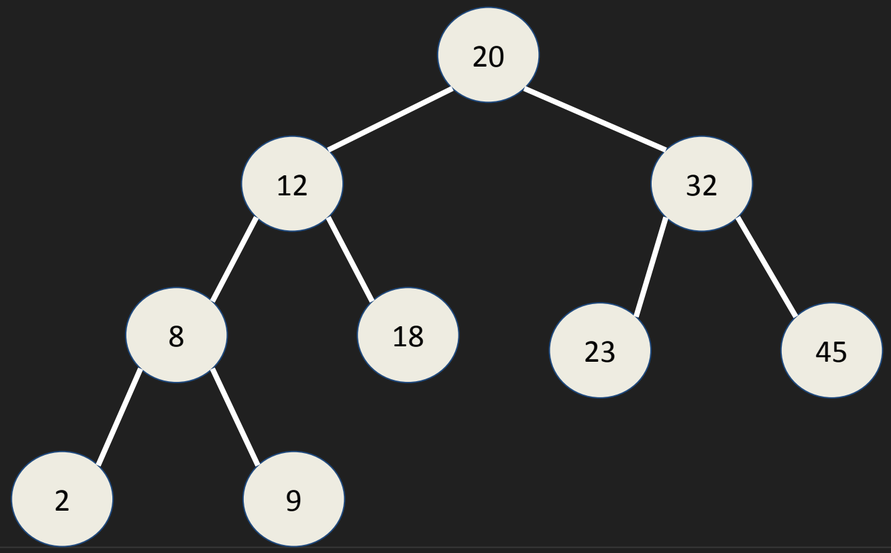
São propriedades dessa estrutura:

- Os elementos são referenciados por seus valores
- Elementos iguais não são armazenados
- Remoção, inserção e buscas com complexidade logarítmica



02 – INTRODUÇÃO

Sets são baseados em árvore binária de busca balanceada (AVL), o que permite operações de inserção, remoção e busca em tempo logarítmico. Esse fator de complexidade é a grande vantagem de se utilizar essa estrutura.



03 – DECLARAÇÃO

O template de declaração de um set é semelhante ao que ocorre com as estruturas vector, stack e queue e mostrado a seguir:

set.cpp > ...

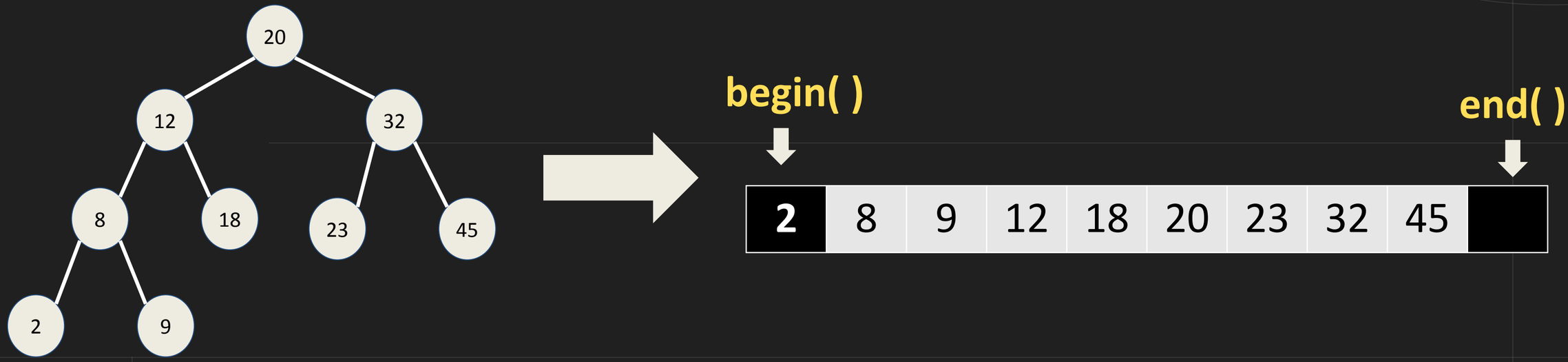
```cpp
#include <iostream>
#include <set>

using namespace std;

int main() {
    //set<tipo_do_dado> nome_do_set

    set<int> set_de_int;
    set<double> set_de_double;
    set<string> set_de_string;
    set<pair<int,int>> set_de_pair;

    return 0;
}
```

04 – ESTRUTURA INTERNA

Dentro de um set dois iteradores são importantes para o uso da estrutura:

begin(): iterador que referencia o primeiro elemento no set seguindo a ordenação;

end(): iterador que referencia a posição seguinte ao último elemento do set seguindo a ordenação.



03 – ESTRUTURA INTERNA

Assinatura	Parâmetros	Retorno	Complexidade
size()	-	Tamanho do vetor (int)	O(1)
insert()	Valor de um elemento	-	O(lg N)
find()	Valor de um elemento	Ponteiro de um elemento (set<type>iterator)	O(lg N)
erase()	Valor de um elemento ou iterador de um elemento	-	O(lg N) / O(1)
lower_bound()	Valor de um elemento	Ponteiro de um elemento (set<type>iterator)	O(lg N)
upper_bound()	Valor de um elemento	Ponteiro de um elemento (set<type>iterator)	O(lg N)

05 – INSERÇÃO

Elementos podem ser inseridos em um set utilizando a função de membro **insert()** passando o valor como parâmetro.

set.cpp > ...

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    set<int> meu_set;

    meu_set.insert(1);
    meu_set.insert(5);
    meu_set.insert(9);
    meu_set.insert(10);
    meu_set.insert(5);

    for(auto n : meu_set)
        cout << n << " ";
    cout << endl;

    return 0;
}
```

Saída:
1 5 9 10

06 – REMOÇÃO

Em um set é possível remover elementos utilizando a função **erase()** de duas formas:

- Passando o valor que deseja ser removido como parâmetro (O(lgN))
- Passando um iterador que referencia o elemento a ser removido (O(1))

A desvantagem de utilizarmos a remoção passando o valor do elemento é que é necessário realizar uma busca para encontrar o elemento a ser removido, o que não acontece quando passamos o iterador desse elemento.

06 – REMOÇÃO

Exemplo de remoção em um set:

set.cpp > ...

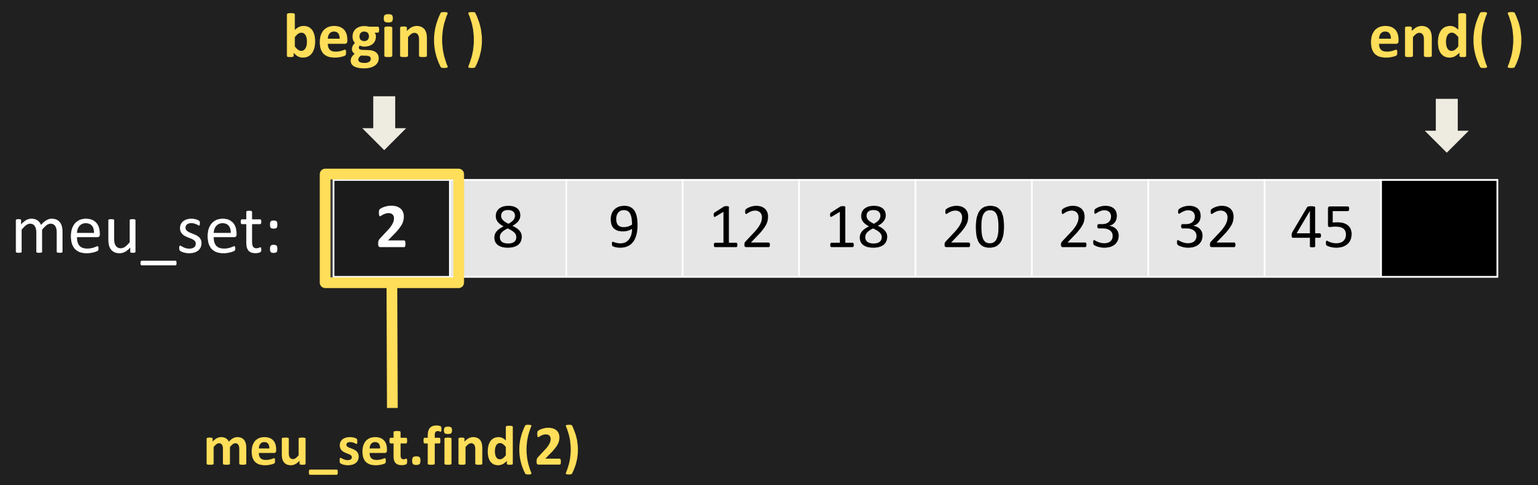
```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    set<int> meu_set;

    meu_set.insert(1);
    meu_set.insert(3);
    meu_set.insert(5);
    meu_set.insert(9);;
    meu_set.insert(10);

    meu_set.erase(meu_set.begin());
    meu_set.erase(10);

    for(auto n : meu_set)
        cout << n << " ";
    cout << endl;

    return 0;
}
```

Saída:
3 5 9

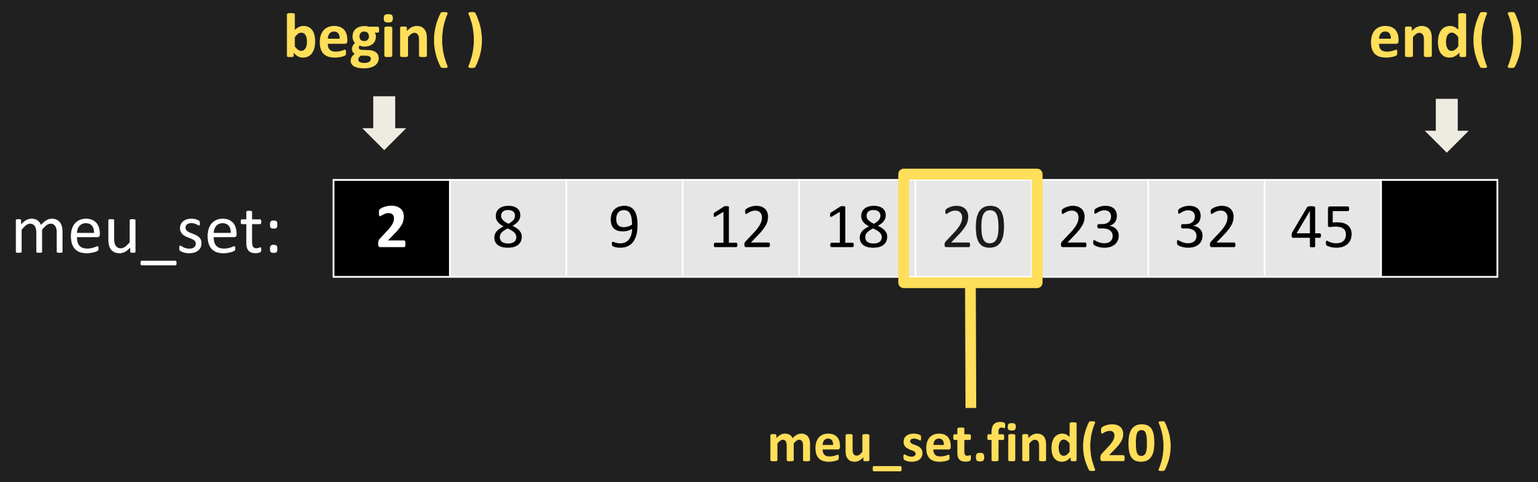
07 – BUSCA POR UM ELEMENTO

Para buscar um elemento dentro de um set utilizamos a função **find()**, que recebe o valor do elemento como parâmetro e retorna um **iterador** que referencia o elemento da estrutura. Caso o elemento não seja encontrado a função retorna um iterador apontando para o **end()**.

07 – BUSCA POR UM ELEMENTO



07 – BUSCA POR UM ELEMENTO



07 – BUSCA POR UM ELEMENTO

07 – BUSCA POR UM ELEMENTO

Segue um exemplo de busca:

```cpp
int main() {
    set<int> meu_set;

    meu_set.insert(1);
    meu_set.insert(3);
    meu_set.insert(5);

    auto res = meu_set.find(1);

    if(res == meu_set.end())
        cout << "Elemento nao encontrado!";
    else
        cout << "O elemento " << *res << " foi encontrado!";

    return 0;
}
```

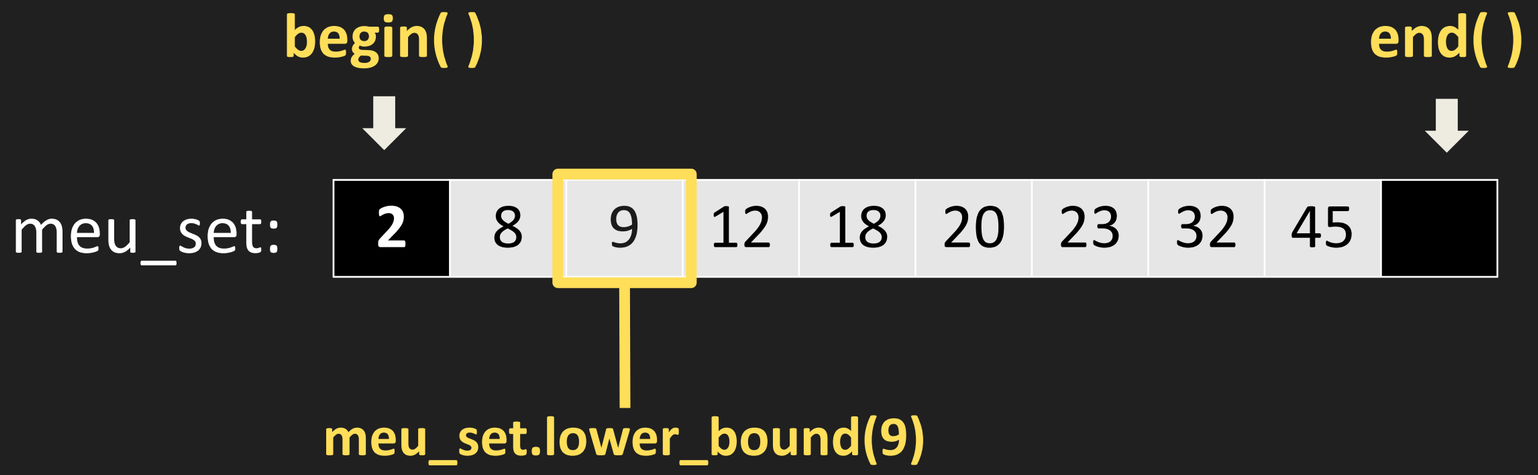
Saída:
O elemento 1 foi encontrado!

07 – BUSCA POR UM ELEMENTO

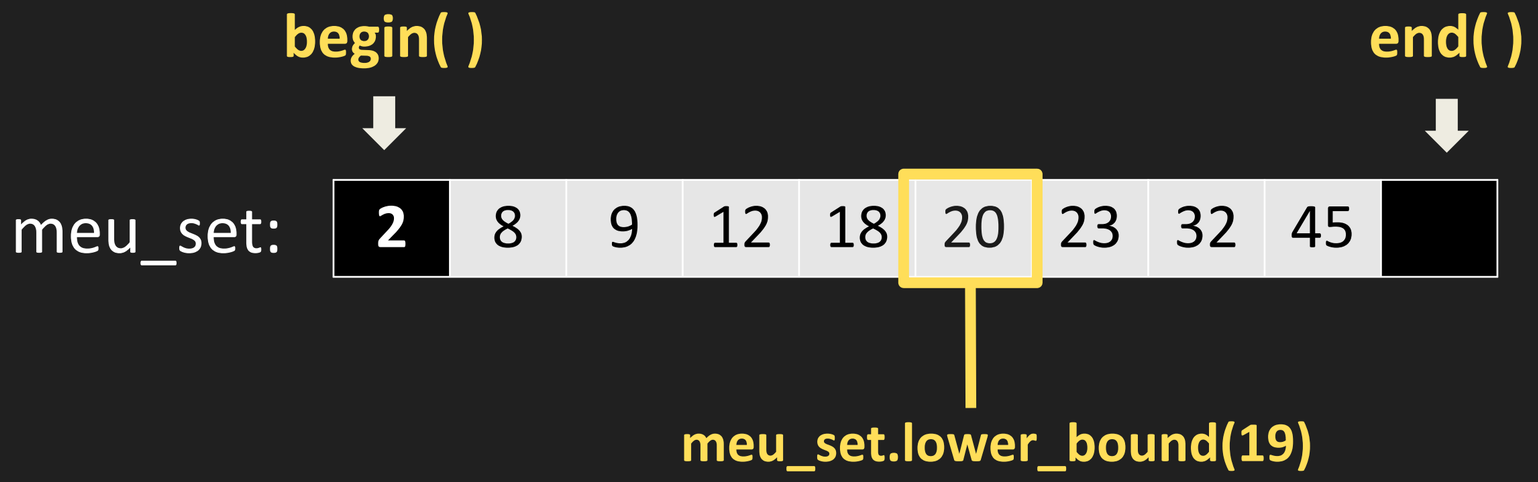
Outro função para resolver problemas é a função **lower_bound()**. Ela retorna um **iterador** que referencia o primeiro elemento do conjunto que não vem antes de um valor. Ou seja, ela informa o endereço do menor elemento que é menor ou igual ao valor passado como parâmetro.

Caso não haja nenhum elemento que seja menor ou igual ao valor passado como parâmetro, ela retorna um iterador **end()** da estrutura.

07 – BUSCA POR UM ELEMENTO



07 – BUSCA POR UM ELEMENTO



07 – BUSCA POR UM ELEMENTO

07 – BUSCA POR UM ELEMENTO

Segue um exemplo de busca com **lower_bound()**:

```cpp
int main() {
    set<int> meu_set;

    meu_set.insert(1);
    meu_set.insert(3);
    meu_set.insert(5);

    auto res = meu_set.lower_bound(4);

    if(res == meu_set.end())
        cout << "Todos os elementos sao menores que 4!\n";
    else
        cout << "Maior elemento que é menor ou igual a 4: " << *res << "!\n";

    res = meu_set.lower_bound(7);

    if(res == meu_set.end())
        cout << "Todos os elementos sao menores que 7!\n";
    else
        cout << "Maior elemento que é menor ou igual a 7: " << *res << "!\n";

    return 0;
}
```

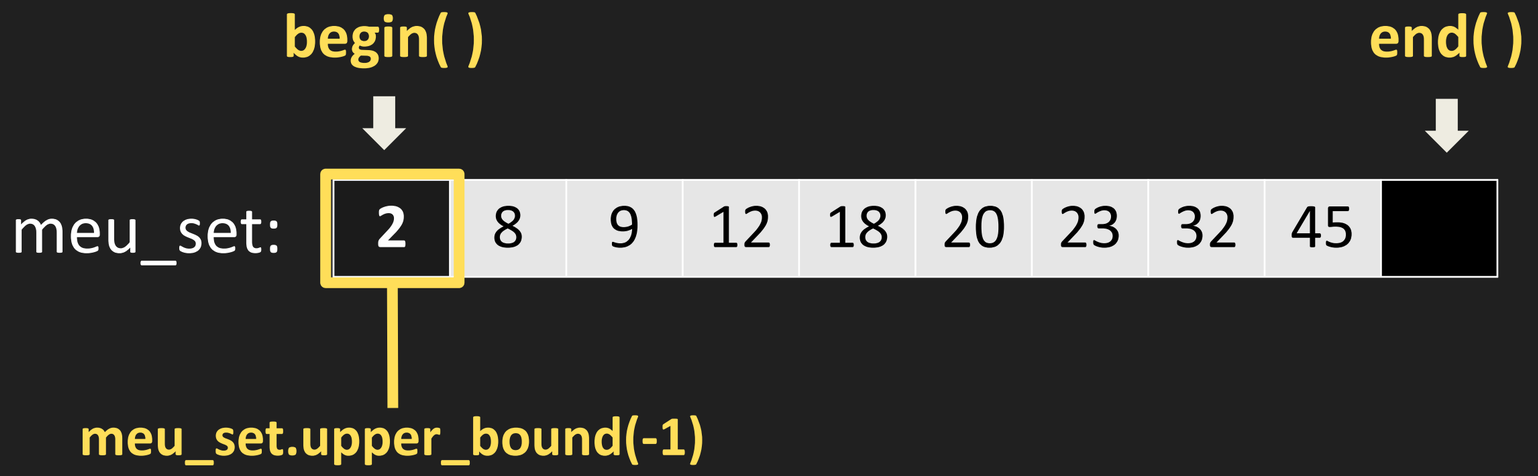
Saída:
Maior elemento que é menor ou igual a 4: 5!
Todos os elementos sao menores que 7!

07 – BUSCA POR UM ELEMENTO

De forma semelhante temos a função **upper_bound()** que retorna um **iterador** para o primeiro elemento que é considerado depois de um valor. Ou seja, ela referencia menor elemento que é maior que o valor passado como parâmetro.

Caso não haja nenhum valor que seja maior que o parâmetro informado a função retorna um iterador para o **end()** da estrutura.

07 – BUSCA POR UM ELEMENTO



07 – BUSCA POR UM ELEMENTO

07 – BUSCA POR UM ELEMENTO

07 – BUSCA POR UM ELEMENTO

Segue um exemplo de busca com **upper_bound()**:

```cpp
int main() {
    set<int> meu_set;

    meu_set.insert(1);
    meu_set.insert(3);
    meu_set.insert(5);

    auto res = meu_set.upper_bound(1);

    if(res == meu_set.end())
        cout << "Todos os elementos sao menores que 1!\n";
    else
        cout << "Menor elemento que é maior que 1: " << *res << "!\n";

    res = meu_set.upper_bound(5);

    if(res == meu_set.end())
        cout << "Todos os elementos sao menores que 5!\n";
    else
        cout << "Menor elemento que é maior que 5: " << *res << "!\n";

    return 0;
}
```

Saída:
Menor elemento que é maior que 1: 3!
Todos os elementos sao menores que 5!

08 – EXERCÍCIO – TROCA DE CARTAS

A ideia chave para resolver esse problema é entender que, dado os conjuntos de cartas, quais cartas podem ser trocadas. A resposta está enunciado: as meninas não querem trocar cartas idênticas, que ambas possuem, e não querem receber cartas repetidas na troca.

Em outras palavras, Ana pode trocar uma carta de valor A, por exemplo, somente se e Beatriz tiver uma outra carta de valor B que Ana também não tenha.

08 – EXERCÍCIO – TROCA DE CARTAS

A ideia chave para resolver esse problema é entender que, dado os conjuntos de cartas, quais cartas podem ser trocadas. A resposta está enunciado: as meninas não querem trocar cartas idênticas, que ambas possuem, e não querem receber cartas repetidas na troca.

Em outras palavras, Ana pode trocar uma carta de valor A, por exemplo, somente se e Beatriz tiver uma outra carta de valor B que Ana também não tenha.

08 – EXERCÍCIO – TROCA DE CARTAS

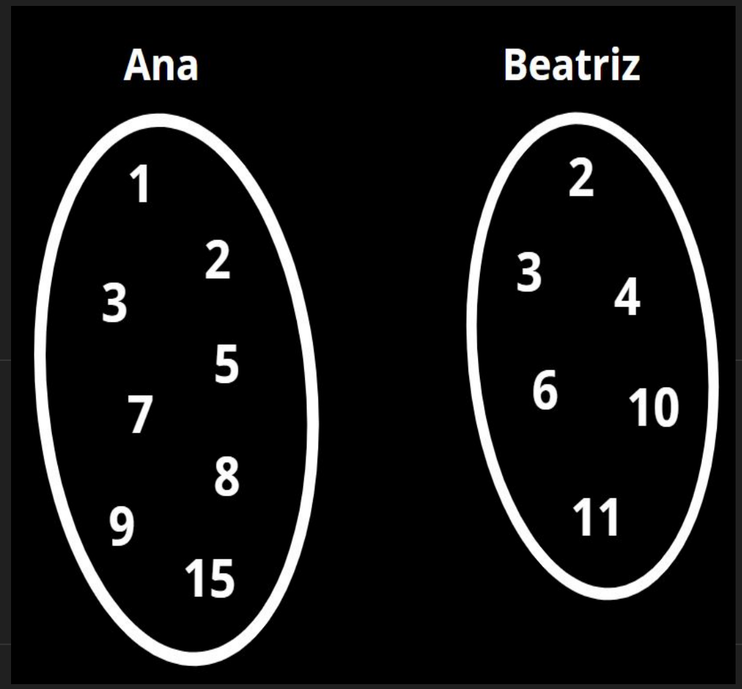
Sendo assim, podemos construir os conjuntos de cartas de cada uma das meninas. Nesse ponto, precisamos entender que o númetro máximo de cartas que podemos encontrar para trocar é no máximo igual a tamanho do menor conjunto.

Por exemplo, se Ana possui um conjunto com 5 cartas e Beatriz 10 cartas, quer dizer que poderemos encontrar um total de no máximo 5 cartas para trocar, pois é o máximo possível que Ana consegue trocar. Nesse caso, precisamos apenas verificar quais dessas 5 de Ana que Beatriz também possui. Cartas que ambas possuem não podem ser trocadas. O restante é o resultado que queremos obter!

08 – EXERCÍCIO – TROCA DE CARTAS

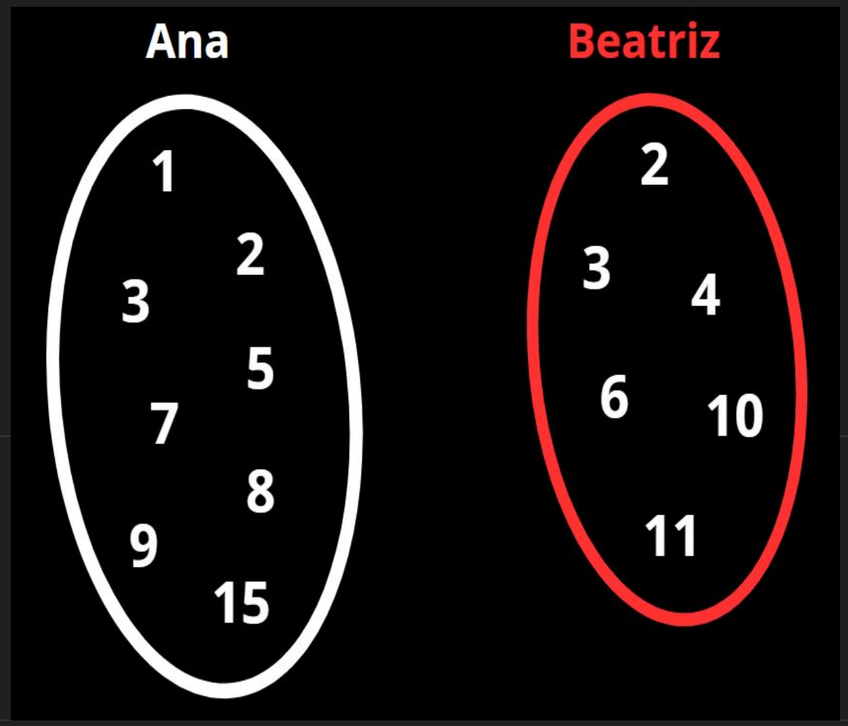
Assim, podemos estruturas nossa solução da seguinte forma:

1. Construir o conjunto de cartas de Ana e Beatriz



08 – EXERCÍCIO – TROCA DE CARTAS

2. Verificar qual o menor conjunto de cartas.



08 – EXERCÍCIO – TROCA DE CARTAS

3. Subtrair do tamanho do menor conjunto todas a quantidade de cartas que aparecem também no outro conjunto.

08 – EXERCÍCIO – TROCA DE CARTAS

4. Imprimir o resultado após as subtrações, ou seja, quantas cartas podem ser trocadas!

08 – EXERCÍCIO – TROCA DE CARTAS

- Primeiramente montamos a estrutura básica do código. Nesse caso, escrevemos a estrutura de repetição para ler os dados: lemos as variáveis a e b (tamanhos dos conjuntos) e, enquanto nenhuma das variáveis for igual a 0 executamos nosso código e lemos a e b novamente no final para podermos processar outro caso de teste.

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    int a,b,v; cin >> a >> b;

    while(a && b) {
        //codigo
        cin >> a >> b;
    }

    return 0;
}
```

08 – EXERCÍCIO – TROCA DE CARTAS

Em seguida criamos dois conjuntos de inteiros x e y, que são os conjuntos de cartas de Ana e Beatriz respectivamente.

Depois disso lemos os dados de suas cartas e os inserimos nos conjuntos.

```
int a,b,v; cin >> a >> b;

while(a && b) {
    set<int> x,y;
    for(int i = 0; i < a; i++){
        cin >> v;
        x.insert(v);
    }
    for(int i = 0; i < b; i++){
        cin >> v;
        y.insert(v);
    }

}
```

08 – EXERCÍCIO – TROCA DE CARTAS

Finalmente, primeiro escolhemos o conjunto de menor tamanho e verificamos quais cartas que não podemos trocar, ou seja, cartas que temos em um conjunto e que conseguimos encontrá-la no outro conjunto usando a função find().

No final do while() imprimimos a nossa variável max, que é o tamanho do menor conjunto menos a quantidade de elementos que encontramos do conjunto maior.

```
int max;
if(x.size() ⩽ y.size()){
    max = x.size();
    for (auto c : x)
        if(y.find(c) ≠ y.end())
            max--;
} else{
    max = y.size();
    for (auto c : y)
        if(x.find(c) ≠ x.end())
            max--;
}
cout << max << "\n";
cin >> a >> b;
```

09 – CONCLUSÃO

Os sets em C++ são estruturas de dados eficazes para problemas de busca e manipulação de conjuntos, pois oferecem operações de inserção, remoção e busca em tempo logarítmico, graças à sua implementação baseada em árvores balanceadas. Essa eficiência os torna ideais para busca binária e garante a ordem e unicidade dos elementos.

Além disso, os sets suportam iteração em ordem crescente e operações de conjunto, como união e interseção, facilitando a análise de dados e a resolução de problemas combinatórios. Dessa forma, sua utilização simplifica algoritmos e melhora o desempenho em tarefas que exigem busca e manipulação eficientes.

OBRIGADO PELA ATENÇÃO

Grupo de Computação Competitiva

GCC.CEFETMG

gcc-cefetmg.github.io/Site-GCC

@gcc.cefetmg