Aula de Map e Set

Samuel Alves Navarro

CIÊNCIA DA COMPUTAÇÃO
FC - UNESP BAURU

Sumário

Problema 1

Você está organizando um evento e precisa registrar o número de matrícula de cada aluno que chega. Para evitar que um mesmo aluno seja contado mais de uma vez, você precisa de um sistema que, ao final, informe exatamente quantos alunos distintos compareceram ao evento.

Reflexão

- Como você agruparia os votos?
- Como faria a contagem para cada nome?
- Como garantiria a ordem alfabética?

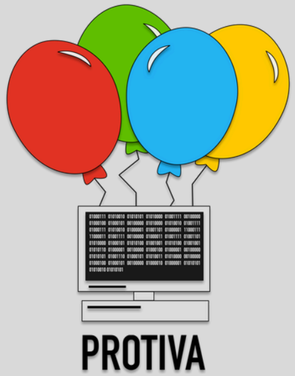
Problema 2

Você está trabalhando na apuração de votos para o representante de classe. Cada aluno votou escrevendo o nome de um candidato em um pedaço de papel. Sua tarefa é criar um programa que leia o nome de cada candidato votado e, ao final, exiba a contagem de votos de cada um. A ordem de exibição dos candidatos deve ser alfabética.

Reflexão

- Como você armazenaria as matrículas que já foram lidas?
- Como faria para ignorar uma matrícula que já apareceu antes?

Principais Funções



- insert(): Adiciona um elemento.
- erase(): Remove um elemento.
- find(): Busca um elemento e retorna um iterador para ele.
  Se não encontra, retorna um iterador para o final.
- count(): Retorna o número de ocorrências de um elemento.
  Para set e map, o valor será sempre 0 ou 1.

Set

- Set é uma estrutura de dados que armazena elementos distintos de maneira ordenada.

- As funções de inserção, remoção e busca possuem complexidade de O(log n)

- **Quando usar?** Ideal para quando você precisa armazenar uma coleção de itens únicos e verificar rapidamente se um item existe ou não.


```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){

    set<int> s;
    s.insert(5); //adiciona no vetor
    s.insert(2);
    s.insert(8);
    s.insert(1);

    auto b = s.find(5); //procura o elemento 5
    cout << *b << endl; //saída: 5
    if(b != s.end())
        cout << "Elemento 5 encontrado" << endl;

    s.erase(5); //remove o elemento 5
    for(auto it : s){
        cout << it << " "; //saída: 1 2 5 8
    }

    return 0;
}
```

Map

- O map é uma estrutura de dados que armazena pares de chave-valor de forma ordenada por meio da chave.

- Inserção, remoção e busca pela chave têm complexidade de O(log n).

- **Quando usar?** Perfeito para quando você precisa criar associações diretas entre dois tipos de dados.

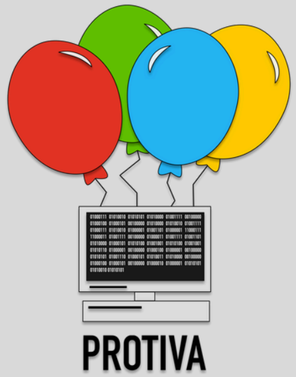

```cpp
int main(){
    // Declaração de um map
    map<int, string> m;
    m[1] = "um"; //Inserção no map
    m[2] = "dois";
    m[3] = "três";


    for(auto it : m){
        cout << it.first << " -> " << it.second << "\n";
        //Saída:
        /*
        1 -> um
        2 -> dois
        3 -> três
        */
    }

    // A função count() verifica se uma chave existe
    //Em um 'map', como as chaves são únicas, o resultado será sempre 1 (se existe) ou 0 (se não existe)
    if(m.count(2)){
        cout << "Resultado: A chave 2 foi encontrada no map.\n";
    } else {
        cout << "Resultado: A chave 2 NAO foi encontrada no map.\n";
    }

    if(m.count(5)){
        cout << "Resultado: A chave 5 foi encontrada no map.\n";
    } else {
        cout << "Resultado: A chave 5 NAO foi encontrada no map.\n";
    }

    for(auto i = m.begin(); i != m.end(); i++)
        cout << i->first << " : " << i->second << '\n';

    return 0;
}
```

MultiSet

- O multiset é muito parecido com o set. A diferença é que ele permite o armazenamento de elementos duplicados.

- Inserção, remoção e busca têm complexidade de O(log n)

- **Quando usar?** Ideal para quando você precisa manter uma coleção de itens ordenados e contar quantas vezes cada item aparece.




```cpp
int main(){
    //Declaração de um multiset de inteiros
    multiset<int> ms;

    ms.insert(10);
    ms.insert(30);
    ms.insert(20);
    ms.insert(10); // Inserindo um elemento duplicado

    for(int elemento : ms){
        cout << elemento << " "; // Saída: 10 10 20 30
    }
    cout << endl;

    // Contando ocorrências do número 10
    cout << "O número 10 aparece " << ms.count(10) << " vezes." << endl;
```

MultiMap

- O multimap é uma variação do map que permite chaves duplicadas.

- Inserção, remoção e busca têm complexidade de O(log n)

- **Quando usar?** Perfeito para quando você precisa agrupar vários valores sob uma mesma categoria (chave). Por exemplo, um dicionário onde uma palavra (chave) pode ter vários significados (valores).


```cpp
int main() {
    // Declaração de um multimap
    multimap<std::string, std::string> agenda;

    agenda.insert({"Maria", "9999-1111"});
    agenda.insert({"Joao", "9888-2222"});
    agenda.insert({"Maria", "9777-3333"}); // Mesma chave "Maria" com outro valor


    for(auto& par : agenda){
        cout << par.first << ": " << par.second << endl;
        // Saída:
        /*
            Joao: 9888-2222
            Maria: 9999-1111
            Maria: 9777-3333
        */
    }

    return 0;
}
```

UnordredMap

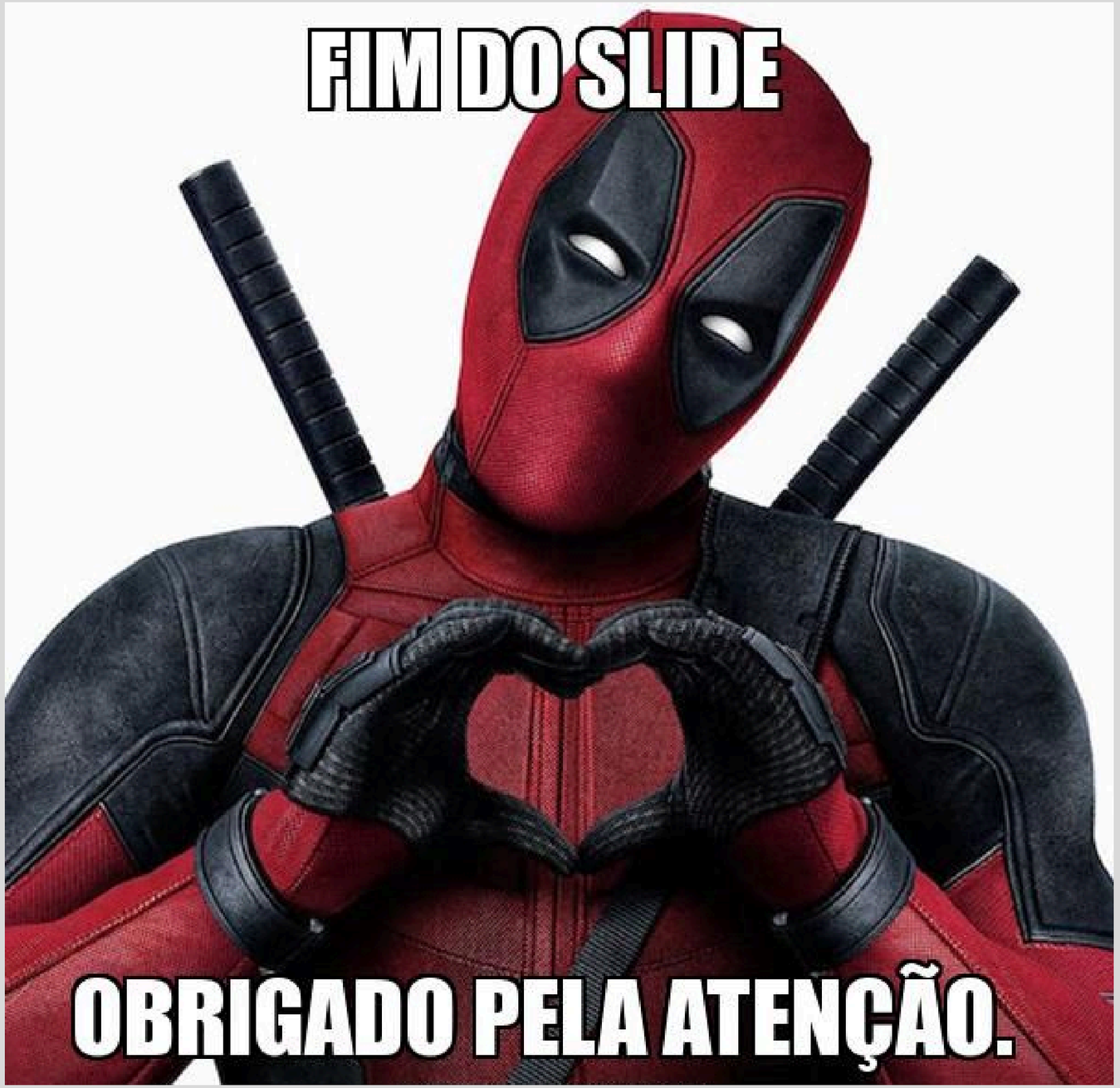
- O unordered_map é parecido com o map, diferindo na implementação e performance.

- Ele utiliza uma tabela de hash, o que torna as operações de inserção, remoção e busca mais rápidas. A desvantagem é que os elementos não são ordenados.

- A complexidade para inserção, remoção e busca é de **O(1)** (tempo constante).

- **Quando usar?** Sempre que você precisar de um map, mas não se importa com a ordem dos elementos e quer a máxima performance.


```cpp
int main() {
    //Declaração de um unordered_map
    unordered_map<string, int> contador_palavras;



    contador_palavras["gato"] = 1;
    contador_palavras["cachorro"] = 2;
    contador_palavras["gato"]++; //Incrementa o valor associado à chave "gato"

    cout << "Contagem de palavras:" << endl;
    for(auto par : contador_palavras){
        cout << par.first << ": " << par.second << endl;
    }

    return 0;
}
```



PROTIVA