

«Set» e «Unordered_Set»

Kaio Christaldo
Fabricio Matsunaga

Template <Set>

Apresentação Problema Motivador

beecrowd | 1861

O Hall dos Assassinos

Por Leandro Zatesko, UFFS  Brazil

Timelimit: 1

Quem matou Meryn Trant? Quem matou Syrio Forel, se é que ele de fato morreu? Quem matou Stannis Baratheon? Quem matou Myrcella Baratheon? Quem matou Aerys II Targaryen? Quem vai matar (**alerta de spoiler!**) Jaime Lannister? Para algumas destas perguntas já sabíamos a resposta. Para outras, tínhamos apenas especulações. No entanto, recebemos de um correspondente anônimo uma lista descrevendo vários assassinatos, que já aconteceram ou que estão para acontecer, revelando tanto o nome dos assassinos quanto dos assassinados. Mas os assassinatos não estão em ordem lexicográfica, nem mesmo em ordem cronológica, e fica difícil contar quantas pessoas cada assassino matou. Você pode nos ajudar?

Entrada

Cada linha da entrada descreve um assassinato informando o nome do assassino seguido pelo nome do assassinado. Cada nome é composto por no mínimo um e no máximo 10 caracteres, sendo o primeiro sempre uma letra maiúscula e os demais sempre letras minúsculas. A entrada consiste de no mínimo uma e no máximo 10^5 linhas e é encerrada em *fim de arquivo*.



Saída

A primeira linha da saída deve consistir da frase “HALL OF MURDERERS”, sem as aspas. Cada uma das linhas seguintes deve conter um nome de um assassino seguido do número de pessoas que ele matou. A lista de assassinos deve obedecer a ordem lexicográfica. Se um assassino também acabou sendo assassinado, ele não deve figurar na lista.

Exemplo de Entrada

```
Arya Meryn
Meryn Syrio
Brienne Stannis
Ellaria Myrcella
Jaime Aerys
Brienne Jaime
```

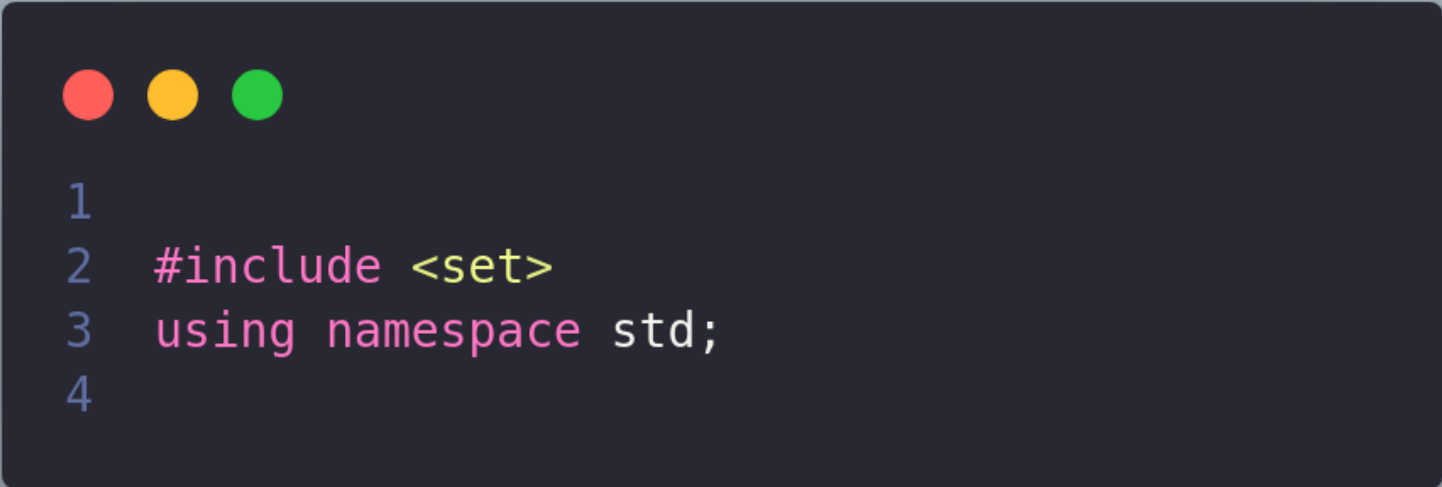
Exemplo de Saída

```
HALL OF MURDERERS
Arya 1
Brienne 2
Ellaria 1
```

1861 – O
Hall dos
Assassinos

Template <Set>

- **Definição:** set é um **container associativo da STL** que armazena elementos únicos em **ordem ordenada**.
- **Implementação:** Árvore Red-Black (**RB-Tree**) – operações em **$O(\log n)$** .
- **Cabeçalho necessário:**



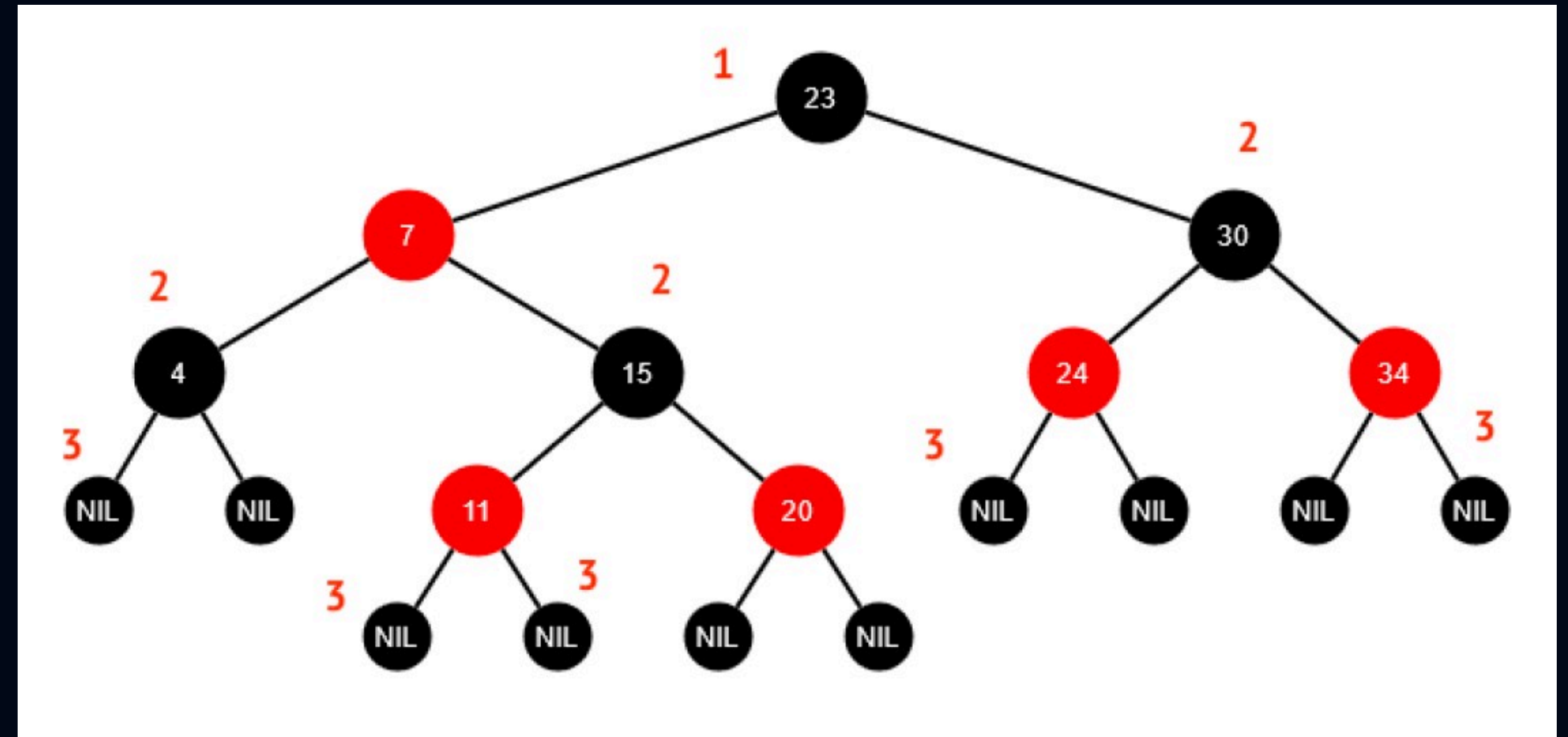
```
1
2 #include <set>
3 using namespace std;
4
```

Template <Set>

- **Árvore Red-Black (RB-Tree)**

Segue cinco regras para manter o balanceamento:

- Todo nó é **vermelho** ou **preto**.
- A **raiz** sempre é **preta**.
- Nós **vermelhos** não podem ter **filhos vermelhos** (não há nós consecutivos vermelhos).
- **Todos** os **caminhos** da **raiz** até as **folhas** têm o **mesmo número** de nós **pretos**.
- **Novos nós** são sempre **inseridos** como **vermelhos**.



Template <Set>

Operações no Set

Principais Operações Usadas:

- **insert(x)** – Insere **x** no conjunto – **$O(\log n)$**
- **erase(x)** – Remove **x** (se existir) – **$O(\log n)$**
- **find(x)** – Retorna **iterador** para **x**, ou **s.end()** se não existir – **$O(\log n)$**
- **count(x)** – Retorna **1** se **x** existir, senão **0** – **$O(\log n)$**
- **size(x)** – Retorna o número de elementos – **$O(1)$**
- **empty(x)** – Retorna **true** se o conjunto estiver vazio – **$O(1)$**
- **clear(x)** – Remove todos os elementos – **$O(n)$**

Template <Set>

- Insert(x)



```
1 set<int> s; // Template instanciado com int
2 s.insert(10);
3 s.insert(5);
4 s.insert(15);
5 s.insert(5); // Duplicata é ignorada
6
7 for (int x : s) cout << x << " "; // Saída: 5 10 15
```

Template <Set>

- `erase(x)`



```
1  s.erase(10); // Remoção do 10
2
3  for (int x : s) cout << x << " "; // Saída: 5 15
```


Template <Set>

- **find(x)**



```
1  set<int> s = {1,2,3,4,5}; // Preechendo conjunto
2
3  auto busca = s.find(9); // Buscando valor 9
4
5  string msg = (busca != s.end()) ? "Encontrado" : "Não Encontrado";
6
7  cout << "Elemento " << msg << "\n"; // Elemento Não Encontrado
8
```

Template <Set>

- **count(x)**



```
1  set<int> s = {1,2,3,4,5}; // Prenchendo Conjunto
2  int busca = 9;
3
4  string msg = (s.count(busca)) ? "Sim" : "Não";
5
6  cout << busca << " está no conjunto? " << msg << "\n"; // 9 está no conjunto? Não
```

Template <Set>

- size(x)



```
1  set<int> s = {1,2,3,4,5}; // Prenchendo Conjunto
2
3  cout << "Tamanho do conjunto: " << s.size() << "\n"; // Tamanho do conjunto: 5
```

Template <Set>

- **empty(x)**



```
1 set<int> s = {1,2,3,4,5}; // Prenchendo Conjunto
2
3 cout << "Conjunto está vazio? " << (s.empty() ? "Sim" : "Não") << "\n"; // Conjuntoi está vazio? Não
```

Template <Set>

- **clear(x)**



```
1  set<int> s = {1,2,3,4,5}; // Prenchendo Conjunto
2
3  s.clear(); // Conjunto S vazio
4
```

Template <Set>

Operações no Set – Bônus

+ Operações:

- **lower_bound(x)** – Retorna o primeiro elemento $\geq x$ – $O(\log n)$
- **upper_bound(x)** – Retorna o primeiro elemento $> x$ – $O(\log n)$
- **multiset**
- **Iterator** – `begin()`, `end()`, `rbegin()`, `rend()`, etc...
- **greater<T>** – altera a ordem de ordenação usando um **comparador**

Template <Set>

Operações no Set – Bônus

- **Lower_bound()** e **Upper_bound()**



```
1  set<int> s = {1,2,3,4,5};
2
3  auto it = s.lower_bound(4); // Primeiro elemento >= 4
4  if (it != s.end()) cout << "Lower Bound de 4: " << *it << '\n';
5
6  it = s.upper_bound(4); // Primeiro elemento > 4
7  if (it != s.end()) cout << "Upper Bound de 4: " << *it << '\n';
8
9  for (auto i : s) cout << i << " "; cout << "\n";
```

Template <Set>

Operações no Set – Bônus

- **Multiset<int>**




```
1 multiset<int> ms;  
2 ms.insert(5);  
3 ms.insert(5);  
4 ms.insert(3);  
5  
6 for (int x : ms) cout << x << " "; // Saída: 3 5 5
```


Template <Set>

Operações no Set – Bônus

- **Iterator** – `begin()`, `end()`, `rbegin()`, `rend()`, etc...




```
1  set<int> s = {1,2,3,4,5};
2
3  cout << "Menor: " << *s.begin() << '\n'; // Menor: 1
4  cout << "Maior: " << *s.rbegin() << '\n'; // Maior: 5
5
6  for (auto it = s.begin(); it != s.end(); ++it) {
7      cout << *it << " ";
8  } cout << "\n";
9
10 // 1 2 3 4 5
```

Template <Set>

Operações no Set – Bônus

- **greater<T>**



```
1  set<int, greater<int>> s;  // Ordenação decrescente
2
3  s.insert(10);
4  s.insert(5);
5  s.insert(20);
6
7  cout << "Elementos em ordem decrescente:\n";
8  for (int x : s) cout << x << " "; // 20 10 5
9  cout << "\n";
```

Template <Set>

Vantagens

- Não precisa se preocupar com duplicatas
- Elementos ordenados automaticamente
- Complexidade $O(\log n)$

Desvantagens

- Mais lento que `unordered_set`
- Maior uso de memória que um vetor
- Inserção lenta em massa

Resolução do Problema Motivador

1861 – O Hall dos Assassinos

Dicas:

- Usar um Map para guardar Nome e Numero de Assassinatos
- Usar um Set para guardar as Vitimas
- Fazer verificação se tem algum Assassino foi Morto
 - Imprimir na tela o Map

A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

Template ◀ **Unordered_Set** ▶

Apresentação Problema Motivador

beecrowd | 1609

Contando Carneirinhos

Por Bruno Adami, Universidade de São Paulo - São Carlos 🇧🇷 Brazil

Timelimit: 1

Para dormir você resolveu contar carneirinhos. O sono está demorando muito para vir e você percebeu que alguns carneirinhos estão se repetindo! Cada um deles é identificado por um número inteiro único, desta forma você vai evitar contar os repetidos.

Dado a sequência dos carneirinhos, imprima quantos de verdade você contou, ou seja, imprima o número de carneirinhos distintos.

Entrada

Na primeira linha você terá um inteiro **T** ($T = 100^*$) indicando o número de casos de teste.

Na primeira linha de cada caso teremos o número inteiro **N** ($1 \leq N \leq 100^*$ ou $1 \leq N \leq 10^4^{**}$), indicando o número de carneirinhos. Na próxima linha teremos **N** inteiros separados por espaço indicando a sequência de carneirinhos.

Os identificadores dos carneiros irão de 0 até 10^9 , inclusive.

*Ocorre em aproximadamente 90% dos casos de teste;

**Ocorre nos demais casos de teste.

```
3
3
1 2 3
3
1 2 1
5
100 1 1 0 0
```

```
3
2
3
```

Saída

Imprima o número de carneirinhos distintos para cada caso.

1609 – Contando
Carneirinhos

Template <Unordened_Set>

A STL unordered_set é um container associativo que prove funcionalidades de um estrutura de dados set desordenado.

Em contraste com o set normal, a ordem de valores de um unordered set é indefinida. Tambem, o unordered set é implementado como uma estrutura de dado de tabela hash enquanto o set normal é implementado como uma estrutura de dado de árvore binária de busca.

Tabela de Espalhamento < Tabela Hash >

A estrutura da tabela de espalhamento armazena elemento em pares de chave-valor onde:

Chave- inteiros unicos usado para indexar os valer

Valor - dados que sao associados com as chaves.

Em lugar de organizar a tabela segundo o valor relativo de cada chave em relação às demais, a tabela de dispersão leva em conta somente o seu valor absoluto, interpretado como um valor numérico

Tabela de < Tabela Hash >

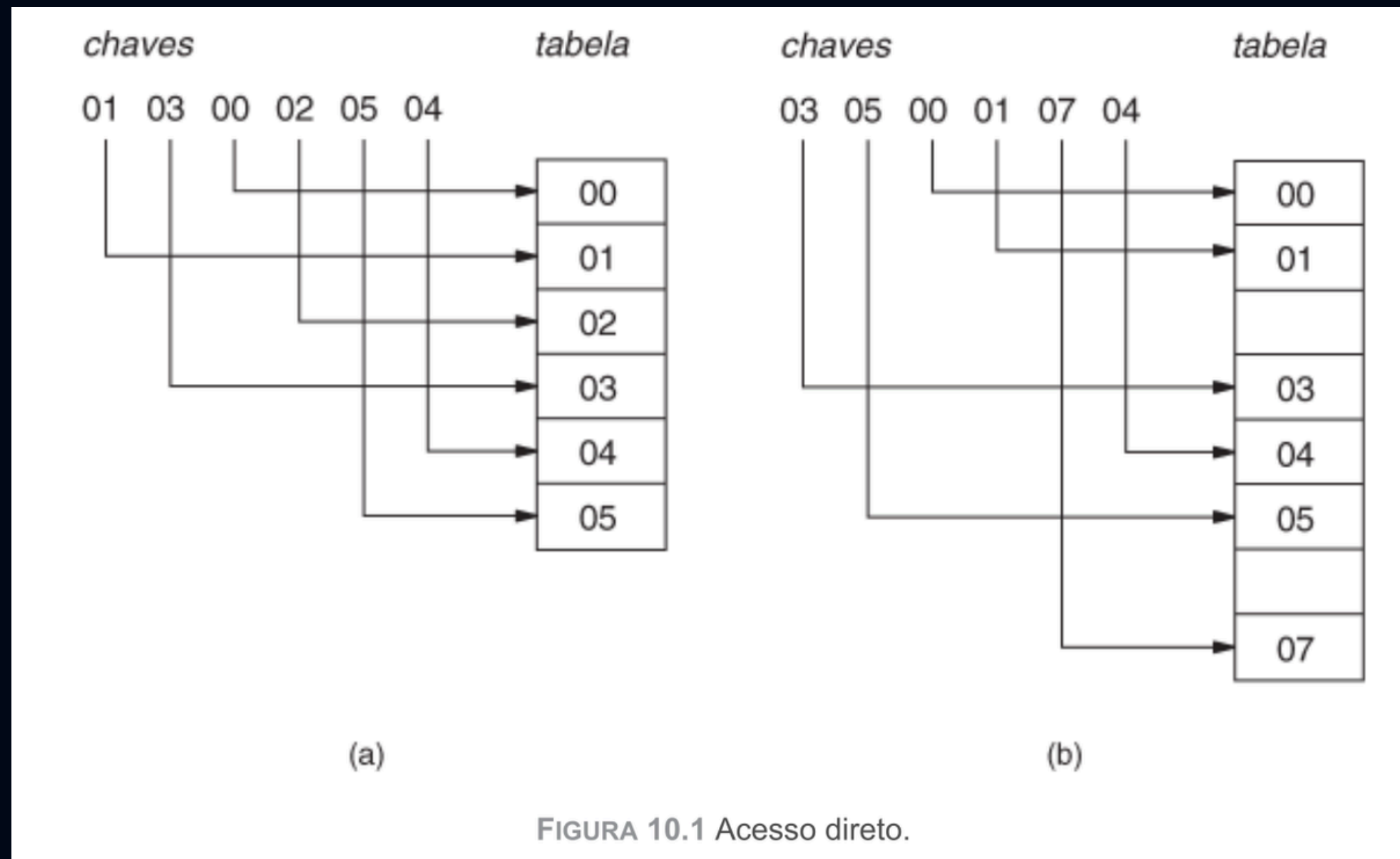


Tabela de < Tabela Hash > – Funções de Dispersão

Uma função de dispersão h transforma uma chave x em um endereço-base $h(x)$ da tabela de dispersão. Idealmente, uma função de dispersão deve satisfazer às seguintes condições:

- produzir um número baixo de colisões;**
- ser facilmente computável;**
- ser uniforme.**

Tabela de < Tabela Hash >

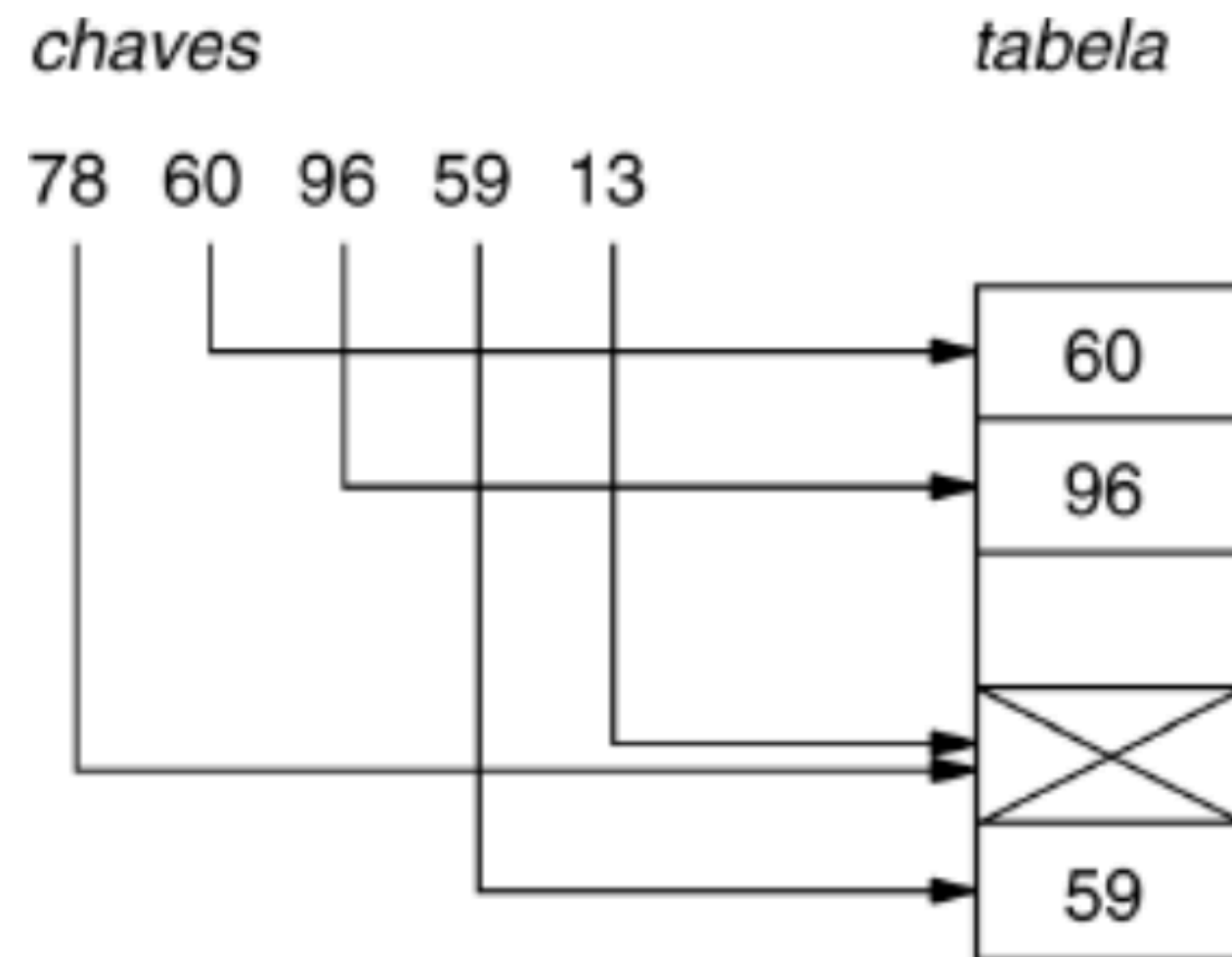


FIGURA 10.2 Exemplo de colisão.

Template <Unordened_Set> – Declaração



```
1  #include <unordered_set>
2
3  unordered_set<data_type> ust;
4
5  // create an unordered_set of integers
6  unordered_set<int> ust_integer;
7
8  // create an unordered_set of strings
9  unordered_set<string> ust_string;
```

Template `<Unordened_Set>` – Inicialização do Unordered Set



```
1 // Initializer List
2 unordered_set<int> unordered_set1 = {1, 100, 2, 9};
3
4 // Uniform Initialization
5 unordered_set<int> unordered_set2 {1, 100, 2, 9};
```

Template <Unordened_Set> – Metodos

insert()

count()

find()

size()

empty()

erase()

clear()

Template <Unordered_Set> – Insert()



```
1  #include <iostream>
2  #include <unordered_set>
3  using namespace std;
4
5  int main() {
6      unordered_set<string> countries;
7
8      // inserts "Nepal" into countries
9      countries.insert("Nepal");
10
11     // inserts "Nepal" , "India", "USA", "Korea" into countries
12     countries.insert({"Nepal", "India", "USA", "Korea"});
13
14     cout << "Countries = ";
15     // display elements of countries
16     for(const auto& country: countries) {
17         cout << country << ", ";
18     }
19     return 0;
20 }
```

Template <Unordered_Set> – erase()

```
1  #include <iostream>
2  #include <unordered_set>
3  using namespace std;
4
5  // function prototype for display_unordered_set()
6  void display_unordered_set(const unordered_set<string> &);
7
8  int main() {
9
10     unordered_set<string> languages {"C++", "Python", "Java", "PHP"};
11
12     cout << "Initial unordered set:\n";
13     display_unordered_set(languages);
14
15     // remove element with value: "Python"
16     languages.erase("Python");
17
18     cout << "\n\nFinal unordered set: \n";
19     display_unordered_set(languages);
20
21     return 0;
22 }
23
24 // utility function to print unordered_set elements
25 void display_unordered_set(const unordered_set<string> &uset) {
26     for(const auto& value: uset) {
27         cout << value << ", ";
28     }
29 }
```


Template `<Unordered_Set>` – `find()` e `count()`

```
1 #include <iostream>
2 #include <unordered_set>
3 using namespace std;
4
5 int main() {
6
7     unordered_set<int> primes {2, 3, 5, 7, 11, 13};
8
9     cout << "Using find():" << endl;
10    cout << "Does number " << 12 << " exist? ";
11
12    // find() returns primes.end() if the value is not found
13    if (primes.find(12) != primes.end()) {
14        cout << "Yes";
15    }
16    else {
17        cout << "No";
18    }
19
20    cout << "\n\nUsing count():" << endl;
21    cout << "Does number " << 7 << " exist? ";
22
23    // count() returns 0 if the value doesn't exist
24    if (primes.count(7)) {
25        cout << "Yes";
26    }
27    else {
28        cout << "No";
29    }
30
31    return 0;
32 }
```

Template <Unordered_Set> – size()



```
1  #include <iostream>
2  #include <unordered_set>
3  using namespace std;
4
5  int main() {
6
7      unordered_set<int> numbers {1, 100, 10, 70, 100};
8
9      cout << "Size: " << numbers.size();
10
11     return 0;
12 }
```

Resolução do Problema Motivador

```
#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

int main(int argc, char const *argv[])
{
    int numeroCasos, numOvelhas, ovelha;
    //vector<int> vectorOvelhas;
    unordered_set<int> ovelhas;
    vector<int> respostas;

    cin >> numeroCasos;
```

```
    for (int i = 0; i < numeroCasos; i++)
    {
        ovelhas.clear();
        cin >> numOvelhas;
        //vectorOvelhas.push_back(numOvelhas);

        for (int j = 0; j < numOvelhas; j++)
        {
            cin >> ovelha;
            ovelhas.insert(ovelha);
        }
        respostas.push_back(ovelhas.size());
    }

    for (int i = 0; i < (int)respostas.size(); i++)
    {
        cout << respostas.at(i) << "\n";
    }

    return 0;
}
```

A resolução estará disponível no Drive. Tente resolver por conta própria e, se precisar, compare com a solução! 😊

Lista de Exercícios

1861 – O Hall dos Assassinos

1921 – Guilherme e suas Pipas

2493 – Jogo do Operador

2653 – Dijkstra

1318 – Bilhetes Falsos

1104 – Troca de Cartas

2456 – Cartas



Se tiver alguma dúvida ou dificuldade na resolução de algum exercício, sinta-se à vontade para perguntar! 😊

Referências

[1] JAVA RUSH. Diagrama de Red-Black Tree [imagem]. Disponível em: <https://cdn.javarush.com/images/article/9a5b5d15-c32b-4b6f-9f8e-b1d12908379c/1080.jpeg>. Acesso em: 03 abr. 2025.

[2] PROGRAMIZ. C++ Unordered Set. Disponível em: <https://www.programiz.com/cpp-programming/unordered-set>. Acesso em: 03 abr. 2025.

[3] PROGRAMIZ. Hash Table. Disponível em: <https://www.programiz.com/dsa/hash-table>. Acesso em: 03 abr. 2025.

[4] GLASIELLY DEMORI. Árvore Rubro-Negra (Aula 11) – Inserção. YouTube, 10 out. 2023. Disponível em: <https://www.youtube.com/watch?v=vSAE4O2zpkY>. Acesso em: 4 abr. 2025.