AED III
Estruturas C++
Set
Multiset
Bitset
Map

Set

Um conjunto (set) é uma estrutura de dados que mantém uma coleção de elementos. As operações básicas dos conjuntos são a inserção, busca e remoção de elementos. A biblioteca padrão C++ contém duas implementações definidas: o set é baseado em uma árvore binária balanceada. A estrutura unordered_set usa hash.

A escolha da implementação definida para usar é muitas vezes uma questão de gosto. O benefício do set é que ele mantém a ordem dos elementos e fornece funções que não estão disponíveis no unordered_set. Por outro lado, unordered_set pode ser mais eficiente.

Set

O código a seguir cria um conjunto que contém números inteiros e mostra algumas das operações. A inserção de função adiciona um elemento ao conjunto, a contagem de função retorna o número de ocorrências de um elemento no conjunto e a função apaga remove um elemento do conjunto.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  set<int> s;
  s.insert(3);
  s.insert(2);
  s.insert(5);
  printf("%d\n", s.count(3));
  printf("%d\n", s.count(4));
  s.erase(3);
  s.insert(4);
  printf("%d\n", s.count(3));
  printf("%d\n", s.count(4));
  return 0;
}
```

Set

Um conjunto pode ser usado principalmente como um vetor, mas não é possível acessar os elementos usando a notação []. O código a seguir cria um conjunto, imprime o número de elementos e itera através de todos os elementos:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  set<int> s;
  set<int>:: iterator it;
  s.insert(2);
  s.insert(5);
  s.insert(6);
  s.insert(8);
  printf("Tamanho do Conjunto: %d\n", s.size());
  printf("s = {");
  for (it = s.begin(); it != s.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  return 0;
}
```

Set

Uma propriedade importante dos conjuntos é que todos os seus elementos são distintos. Assim, a contagem de função sempre retorna 0 (o elemento não está no conjunto) ou 1 (o elemento está no conjunto) e a inserção de função nunca adiciona um elemento ao conjunto se ele já estiver lá. O código a seguir ilustra isso:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  set<int> s;
  set<int>:: iterator it;
  s.insert(2);
  s.insert(5);
  s.insert(6);
  s.insert(8);
  printf("Tamanho do Conjunto: %d\n", s.size());
  printf("s = {");
  for (it = s.begin(); it != s.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  s.insert(5);
  s.insert(7);
  s.insert(8);
  printf("Tamanho do Conjunto: %d\n", s.size());
  printf("s = {");
  for (it = s.begin(); it != s.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  return 0;
}
```

Multiset

C++ também contém as estruturas multiset e unordered_multiset que, de outra forma, funcionam como set e unordered_set, mas podem conter várias instâncias de um elemento. Por exemplo, no seguinte código, as três instâncias do número 5 são adicionadas a um multiset:

```
multiset<int> s;
s.insert(5); s.insert(5); s.insert(5);
printf("Quantidade de 5: %d\n", s.count(5));
```

Multiset

A função apagar remove todas as instâncias de um elemento de um multiset:

```
s.erase(5);
printf("Quantidade de 5: %d\n", s.count(5));
```

Muitas vezes, apenas uma instância deve ser removida, o que pode ser feito da seguinte maneira:

```
s.erase(s.find(5));
printf("Quantidade de 5: %d\n", s.count(5));
```

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  multiset<int> s;
  multiset<int>:: iterator it;
  s.insert(5); s.insert(5); s.insert(5);
  printf("Quantidade de 5: %d\n", s.count(5));
  printf("s = {");
  for (it = s.begin(); it != s.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  s.erase(5);
  printf("Quantidade de 5: %d\n", s.count(5));
  printf("s = {");
  for (it = s.begin(); it != s.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  s.insert(5); s.insert(5); s.insert(5);
  s.erase(s.find(5));
  printf("Quantidade de 5: %d\n", s.count(5));
  printf("s = {");
  for (it = s.begin(); it != s.end(); ++it)
  {
    printf(" %d", *it);
  }
  printf("}\n");
  return 0;
}
```

Exercício

Implemente um algoritmo que tenha dois conjuntos, A e B, cujos elementos sejam previamente lidos, e permita que façam as seguintes operações:

1) Crie um conjunto C com a união entre A e B;

2) Crie um conjunto D com a intersecção entre A e B;

3) Crie um conjunto E com a subtração A - B

Bitset

Um bitset é um vetor cujo cada valor seja 0 ou 1.

O benefício do uso de bitsets é que eles exigem menos memória do que matrizes comuns, porque cada elemento em um bitset usa apenas um bit de memória. Por exemplo, se n bits forem armazenados em uma matriz int, serão utilizados 32n bits de memória, mas um bitetro correspondente apenas requer n bits de memória. Além disso, os valores de um bitset podem ser manipulados de forma eficiente usando operadores de bits, o que permite otimizar algoritmos usando conjuntos de bits.

O código a seguir mostra como manipular o bitset:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  bitset<10> s(string("0010011010"));
  // from right to left
  cout << "s = (" << s << ")\n";
  cout << "s[4] = " << s[4] << "\n";
  cout << "s[5] = " << s[5] << "\n";
  printf("Total = %d\n", s.count());
  bitset<10> a(string("0010110110"));
  bitset<10> b(string("1011011000"));
  cout << "a = (" << a << ")\n";
  cout << "b = (" << b << ")\n";
  cout << "a&b = (" << (a&b) << ")\n";
  cout << "a|b = (" << (a|b) << ")\n";
  cout << "a^b = (" << (a^b) << ")\n";
  return 0;
}
```

Exercício

Resolva as seguintes operações lógicas:

12 & 10

12 | 10

12 ^10

Map

Um mapa é uma matriz generalizada que consiste em pares de valores chave. Enquanto as teclas em uma matriz comum são sempre os inteiros consecutivos 0, 1, ... , N - 1, onde n é o tamanho da matriz, as chaves em um mapa podem ser de qualquer tipo de dados e não precisam ser valores consecutivos.

A biblioteca padrão C++ contém duas implementações de mapa que correspondem às implementações definidas: o mapa de estrutura é baseado em uma árvore binária balanceada, enquanto a estrutura unordered_map usa hash.

O código a seguir cria um mapa onde as chaves são strings e os valores são inteiros:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  map<string,int> m;
  m["primeiro"] = 4;
  m["segundo"] = 3;
  m["terceiro"] = 9;
  printf("primeiro: %d\n", m["primeiro"]);
  printf("segundo: %d\n", m["segundo"]);
  printf("terceiro: %d\n", m["terceiro"]);
  return 0;
}
```

Map

O código a seguir imprime todas as chaves e valores em um mapa e se o valor de uma chave for solicitado, mas o mapa não o contém, a chave é automaticamente adicionada ao mapa com um valor padrão. Por exemplo, no código a seguir, a chave “fulano" com valor 0 é adicionada ao mapa.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  map<string,int> m;
  map<string,int>:: iterator it;
  m["primeiro"] = 4;
  m["segundo"] = 3;
  m["terceiro"] = 9;
  printf("fulano - %d\n\n", m["fulano"]);
  for (it = m.begin(); it != m.end(); ++it)
  {
    cout << it->first << " - " << it->second << "\n";
  }
  return 0;
}
```

Exercício

Implemente um algoritmo, utilizando map, representando Nome e Salário de funcionários de uma empresa. Depois imprima o nome de quem tem o maior salário e o nome de quem tem o menor salário.