Biblioteca de Grafos - Manual do Usuário

class graph

Sumário
Classe que representa um grafo direto.

Sinopse
```
template <class Node, class Edge, class Compare = less<Node> >
class graph;
```

Descrição
Classe que representa um grafo direto. Possui métodos para inserir e remover nós e arcos, além de localizar um determinado nó ou um arco a partir de dois nós. Possui quatro *iterators*, (node_iterator, const_node_iterator, edge_iterator, const_edge_iterator) para trabalhar com os nós e arcos.

Interface
```
template <class Node, class Edge, class Compare = less<Node> >
class graph
{
public:
 // definitions
    class node_iterator;
    class const_node_iterator;
    class edge_iterator;
    class const_edge_iterator;
  // Constructors/Copy/Destructor
    graph(void);
    graph(const graph& x);
    virtual ~graph(void);
    graph& operator = (const graph& x);
  // iterators
    node_iterator begin(void);
    const_node_iterator begin(void) const;
    node_iterator end(void);
    const_node_iterator end(void) const;
  // Modifiers
    void clear(void);
    void erase(const Node& node);
    void erase_edge(const Node& n1, const Node& n2);
    node_iterator insert(const Node& node);
    edge_iterator insert(const Node& node1, const Node& node2,
        const Edge& edge);
    // Observers
    bool empty(void) const;
    unsigned long size(void) const;
    node_iterator find(const Node& node);
    const_node_iterator find(const Node& node) const;
    Edge& find_edge(const Node& no1, const Node& no2);
    const Edge& find_edge(const Node& no1, const Node& no2) const;
};
```

Construtores
graph(void);
Cria um grafo vazio, sem nos ou arcos.
graph(const graph& x);
Construtor de cópia. Cria uma cópia de x.

Destrutor

~graph(void);

Desaloca a memória alocada pelo próprio grafo.

Operador de atribuição

graph& **operator=**(const graph& x);

Copia x e retorna uma referência para o próprio grafo.

Iterators

node_iterator **begin**(void);

Retorna um iterator que aponta para o primeiro nó do grafo.

const_node_iterator **begin**(void) const;

Retorna um const_iterator que aponta para o primeiro nó do grafo.

node_iterator **end**(void);

Retorna um iterator que aponta para depois do último nó do grafo.

const_node_iterator **end**(void) const;

Retorna um const_iterator que aponta para depois do último nó do grafo.

Funções membro

void **clear**(void);

Remove todos os nós e arcos do grafo.

bool **empty**(void) const;

Retorna true se o grafo não possuir nós.

void **erase**(const Node& node);

Remove o nó *node*. Se o nó não existir, lança uma exceção.

void **erase_edge**(const Node& n1, const Node& n2);

Remove o arco entre os nós *node1* e *node2*. Se não existir arco, uma exceção é lançada.

node_iterator **insert**(const Node& node);

Insere o nó *node* e retorna um iterator que aponta para ele.

edge_iterator **insert**(const Node& node1, const Node& node2, const Edge& edge);

Insere o arco *edge* entre os nós *node1* e *node2* e retorna um iterator que aponta para ele.

node_iterator **find**(const Node& node);

Retorna um iterator que aponta para o nó *node*. Se o nó não for encontrado, retorna um iterator que aponta para depois do último nó do grafo.

const_node_iterator **find**(const Node& node) const;

Retorna um const_iterator que aponta para o nó *node*. Se o nó não for encontrado, retorna um const_iterator que aponta para depois do último nó do grafo.

Edge& **find_edge**(const Node& no1, const Node& no2);

const Edge& **find_edge**(const Node& no1, const Node& no2) const;

Retorna o arco entre os nós *node1* e *node2*. Se não existir arco, uma exceção é lançada.

unsigned long **size**(void) const;

Retorna o número de nós do grafo.

Exemplo

```
#include <iostream>
#include <string>
#include "graph.h"
using namespace std;
using namespace BBib;

void main(void)
{
  typedef graph<string, double>        Grafo;
  typedef Grafo::node_iterator         node_iterator;
  typedef node_iterator::edge_iterator edge_iterator;

  Grafo grafo;
  node_iterator no;
  edge_iterator arco;

  try
  {
    grafo.insert("SP");
    grafo.insert("Rio");
    grafo.insert("BH");
    grafo.insert("Porto Alegre");
    grafo.insert(400, "SP", "Rio");
    grafo.insert(1600, "Porto Alegre", "Rio");
    grafo.insert(600, "BH", "Rio");
    no = grafo.begin();
    while(no != grafo.end())
    {
      cout << *no << endl;
      arco = no.edges_from_begin();
      while(arco != no.edges_from_end())
      {
        cout << " " << (*arco) << " - vai para " << arco.node() << endl;
        arco++;
      }
      arco = no.edges_to_begin();
      while(arco != no.edges_to_end())
      {
        cout << " " << (*arco) << " - vem de " << arco.node() << endl;
        arco++;
      }
      no++;
    }
  }
  catch(BExcecao& e)
  {
    cout << e.Mensagem() << endl;
  }
}
```

Sumário
Iterator para representar um nó do grafo.

Sinopse
```
template <class Node, class Edge, class Compare>
class graph<Node, Edge, Compare>::node_iterator;
```

Descrição
Iterator para representar um nó do grafo. Pode-se obter todos os arcos incidentes, tanto deste nó quanto para ele.

Interface
```
template <class Node, class Edge, class Compare>
class graph<Node, Edge, Compare>::node_iterator
{
  public:
    node_iterator();
    node_iterator(const node_iterator& x);
    bool operator== (const node_iterator& x) const;
    bool operator!= (const node_iterator& x) const;
    Node& operator* () const;
    Node* operator-> () const;
    node_iterator& operator++ ();
    node_iterator operator++ (int);
    node_iterator& operator-- ();
    node_iterator operator-- (int);
    edge_iterator IncidentsFrom_begin(void) const;
    edge_iterator IncidentsFrom_end(void) const;
    edge_iterator IncidentsTo_begin(void) const;
    edge_iterator IncidentsTo_end(void) const;
};
```

Construtores
node_iterator(void);
Cria um objeto vazio.
node_iterator(const node_iterator& x);
Construtor de cópia. Cria uma cópia de x.

Operadores membro
bool **operator==** (const node_iterator& x) const;
Retorna true se o iterator for o mesmo que x.
bool **operator!=** (const node_iterator& x) const;
Retorna true se o iterator for diferente de x.
Node& **operator*** () const;
Retorna o nó apontado pelo iterator.
Node* **operator->** () const;
Retorna o endereço do nó apontado pelo iterator.
node_iterator& **operator++** ();
node_iterator **operator++** (int);
Aponta para o próximo nó.
node_iterator& **operator--** ();
node_iterator **operator--** (int);
Aponta para o nó anterior.

Funções membro

edge_iterator IncidentsFrom_begin(void) const;

Retorna um iterator para o primeiro dos arcos incidentes do nó.

edge_iterator IncidentsFrom_end(void) const;

Retorna um iterator para depois do último dos arcos incidentes do nó.

edge_iterator IncidentsTo_begin(void) const;

Retorna um iterator para o primeiro dos arcos que incidem no nó.

edge_iterator IncidentsTo_end(void) const;

Retorna um iterator para depois do último dos arcos que incidem no nó.

---

class const_node_iterator

Sumário

Iterator para representar um nó constante do grafo.

Sinopse

```
template <class Node, class Edge, class Compare>
class graph<Node, Edge, Compare>::const_node_iterator;
```

Descrição

Iterator para representar um nó constante do grafo. Pode-se obter todos os arcos incidentes, tanto deste nó quanto para ele.

Interface

```
template <class Node, class Edge, class Compare>
class graph<Node, Edge, Compare>::const_node_iterator
{
  public:
    const_node_iterator();
    const_node_iterator(const const_node_iterator& x);
    bool operator== (const const_node_iterator& x) const;
    bool operator!= (const const_node_iterator& x) const;
    const Node& operator* () const;
    const Node* operator-> () const;
    const_node_iterator& operator++ ();
    const_node_iterator operator++ (int);
    const_node_iterator& operator-- ();
    const_node_iterator operator-- (int);
    const_edge_iterator IncidentsFrom_begin(void) const;
    const_edge_iterator IncidentsFrom_end(void) const;
    const_edge_iterator IncidentsTo_begin(void) const;
    const_edge_iterator IncidentsTo_end(void) const;
};
```

Construtores

const_node_iterator(void);

Cria um objeto vazio.

const_node_iterator(const const_node_iterator& x);

Construtor de cópia. Cria uma cópia de x.

Operadores membro

bool **operator==** (const const_node_iterator& x) const;

Retorna true se o iterator for o mesmo que x.

bool **operator!=** (const const_node_iterator& x) const;

Retorna true se o iterator for diferente de x.

const Node& **operator*** () const;

Retorna o nó apontado pelo iterator.

const Node* **operator->** () const;
Retorna o endereço do nó apontado pelo iterator.
const_node_iterator& **operator++** ();
const_node_iterator **operator++** (int);
Aponta para o próximo nó.
const_node_iterator& **operator--** ();
const_node_iterator **operator--** (int);
Aponta para o nó anterior.

Funções membro
const_edge_iterator IncidentsFrom_begin(void) const;
Retorna um iterator para o primeiro dos arcos incidentes do nó.
const_edge_iterator IncidentsFrom_end(void) const;
Retorna um iterator para depois do último dos arcos incidentes do nó.
const_edge_iterator IncidentsTo_begin(void) const;
Retorna um iterator para o primeiro dos arcos que incidem no nó.
const_edge_iterator IncidentsTo_end(void) const;
Retorna um iterator para depois do último dos arcos que incidem no nó.

---

class edge_iterator

Sumário
Iterator para representar um arco do grafo.

Sinopse
```
template <class Node, class Edge, class Compare>
class graph<Node, Edge, Compare>::edge_iterator
```

Descrição
Iterator para representar um arco do grafo.

Interface
```
template <class Node, class Edge, class Compare>
class graph<Node, Edge, Compare>::edge_iterator
{
public:
    edge_iterator(void);
    edge_iterator(const edge_iterator& x);
    bool operator== (const edge_iterator& x) const;
    bool operator!= (const edge_iterator& x) const;
    Edge& operator* () const;
    Edge* operator-> () const;
    edge_iterator& operator++ ();
    edge_iterator operator++ (int);
    edge_iterator& operator-- ();
    edge_iterator operator-- (int);
    Node& incident_from() const;
    Node& incident_to() const;
};
```

Construtores
edge_iterator(void);
Cria um objeto vazio.
edge_iterator(const edge_iterator& x);
Construtor de cópia. Cria uma cópia de x.

Operadores membro

bool operator== (const edge_iterator& x) const;

Retorna true se o objeto for igual a x.

bool operator!= (const edge_iterator& x) const;

Retorna true se o objeto for diferente de x.

Edge& operator* () const;

Retorna o arco apontado pelo iterator.

Edge* operator-> () const;

Retorna um ponteiro para o arco apontado pelo iterator.

edge_iterator& operator++ ();

edge_iterator operator++ (int);

Aponta para o próximo nó.

edge_iterator& operator-- ();

edge_iterator operator-- (int);

Aponta para o nó anterior.

Funções membro

Node& incident_from() const;

Retorna o nó de onde o arco incide.

Node& incident_to() const;

Retorna o nó para onde o arco incide.

---

class const_edge_iterator

Sumário

Iterator constante para representar um arco do grafo.

Sinopse

```
template <class Node, class Edge, class Compare>
class graph<Node, Edge, Compare>::const_edge_iterator
```

Descrição

Iterator constante para representar um arco do grafo.

Interface

```
template <class Node, class Edge, class Compare>
class graph<Node, Edge, Compare>::const_edge_iterator
{
public:
    const_edge_iterator(void);
    const_edge_iterator(const const_edge_iterator& x);
    bool operator== (const const_edge_iterator& x) const;
    bool operator!= (const const_edge_iterator& x) const;
    const Edge& operator* () const;
    const Edge* operator-> () const;
    const_edge_iterator& operator++ ();
    const_edge_iterator operator++ (int);
    const_edge_iterator& operator-- ();
    const_edge_iterator operator-- (int);
    const Node& incident_from() const;
    const Node& incident_to() const;
};
```

Construtores

const_edge_iterator(void);

Cria um objeto vazio.

const_edge_iterator(const const_edge_iterator& x);

Construtor de cópia. Cria uma cópia de x.

Operadores membro

bool operator== (const const_edge_iterator& x) const;

Retorna true se o objeto for igual a x.

bool operator!= (const const_edge_iterator& x) const;

Retorna true se o objeto for diferente de x.

const Edge& operator* () const;

Retorna o arco apontado pelo iterator.

const Edge* operator-> () const;

Retorna um ponteiro para o arco apontado pelo iterator.

const_edge_iterator& operator++ ();

const_edge_iterator operator++ (int);

Aponta para o próximo nó.

edge_iterator& operator-- ();

const_edge_iterator operator-- (int);

Aponta para o nó anterior.

Funções membro

const Node& incident_from() const;

Retorna o nó de onde o arco incide.

const Node& incident_to() const;

Retorna o nó para onde o arco incide.