Undirected Graph Header

Generated by Doxygen 1.8.9.1

Fri Jun 19 2015 22:16:37

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

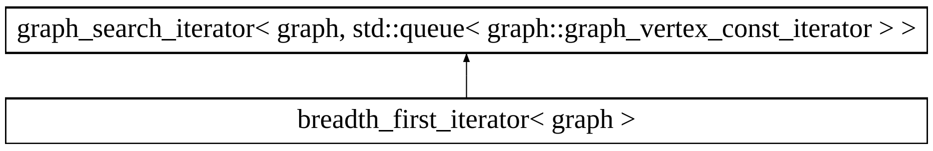
Class Documentation

3.1 breadth_first_iterator< graph > Class Template Reference

Breadth first iterator.

`#include <graph_search_iterator.h>`

Inheritance diagram for breadth_first_iterator< graph >:


graph_search_iterator< graph, std::queue< graph::graph_vertex_const_iterator > >
breadth_first_iterator< graph >

Public Member Functions

- breadth_first_iterator (const graph &g, typename graph::graph_vertex_const_iterator start)

  Constructs a new BFS iterator.
- graph::graph_vertex_const_iterator next ()

  Returns iterator to the next vertex.

Additional Inherited Members

3.1.1 Detailed Description

template<typename graph>class breadth_first_iterator< graph >

Breadth first iterator.

This iterator traverses an undirected_graph using the breadth first algortihm.

Template Parameters

graph	Type of the graph that should be traversed.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 template<typename graph > **breadth_first_iterator**< graph >::**breadth_first_iterator** (const graph & *g*, typename graph::graph_vertex_const_iterator *start*) `[inline]`

Constructs a new BFS iterator.

Constructs a breadth first search iterator object, initializing it with the supplied values.

Parameters

g	A reference to the graph that should be traversed.
start	Vertex iterator to the node that should be used as a starting point.

3.1.3 Member Function Documentation

3.1.3.1 template<typename graph > graph::graph_vertex_const_iterator **breadth_first_iterator**< graph >::next ( ) `[inline]`, `[virtual]`

Returns iterator to the next vertex.

This method increments the iterator using the BFS algorithm and returns an iterator to the next element.

Returns

Iterator to the next element

Implements graph_search_iterator< graph, std::queue< graph::graph_vertex_const_iterator > >.

The documentation for this class was generated from the following file:

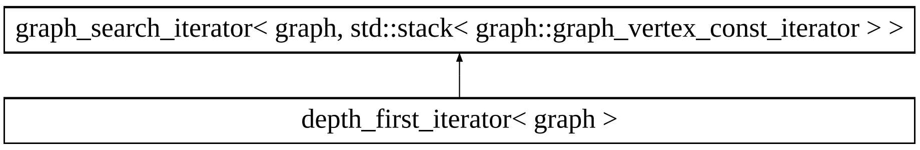
- undirected_graph/source/graph_search_iterator.h

3.2 depth_first_iterator< graph > Class Template Reference

Depth first iterator.

```
#include <graph_search_iterator.h>
```

Inheritance diagram for depth_first_iterator< graph >:


graph_search_iterator< graph, std::stack< graph::graph_vertex_const_iterator > >
depth_first_iterator< graph >

Public Member Functions

- depth_first_iterator (const graph &g, typename graph::graph_vertex_const_iterator start)

  Constructs a new DFS iterator.
- graph::graph_vertex_const_iterator next ()

  Returns iterator to the next vertex.

Additional Inherited Members

3.2.1 Detailed Description

template<typename graph>class depth_first_iterator< graph >

Depth first iterator.

This iterator traverses an undirected_graph using the depth first algortihm.

Template Parameters

graph	Type of the graph that should be traversed.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 template<typename graph > depth_first_iterator< graph >::depth_first_iterator (const graph & *g*, typename graph::graph_vertex_const_iterator *start*) `[inline]`

Constructs a new DFS iterator.

Constructs a depth first search iterator object, initializing it with the supplied values.

Parameters

g	A reference to the graph that should be traversed.
start	Vertex iterator to the node that should be used as a starting point.

3.2.3 Member Function Documentation

3.2.3.1 template<typename graph > graph::graph_vertex_const_iterator depth_first_iterator< graph >::next ( ) `[inline], [virtual]`

Returns iterator to the next vertex.

This method increments the iterator using the DFS algorithm and returns an iterator to the next element.

Returns

Iterator to the next element

Implements graph_search_iterator< graph, std::stack< graph::graph_vertex_const_iterator > >.

The documentation for this class was generated from the following file:

- undirected_graph/source/graph_search_iterator.h

3.3 graph_search_iterator< graph, waiting_container > Class Template Reference

Base class for search iterators in a graph.

`#include <graph_search_iterator.h>`

Public Member Functions

- graph_search_iterator (const graph &g, typename graph::graph_vertex_const_iterator start)
  Constructs a new graph search iterator.
- bool end () const
  Check whether search has finished.
- size_t discovered () const
  Returns number of discovered vertices.

- virtual graph::graph_vertex_const_iterator next ()=0

  Returns iterator to next vertex in the search order.

Protected Attributes

- waiting_container m_waiting

  Queue for waiting vertex iterators.
- std::unordered_set< typename graph::vertex_id_type > m_discovered

  The vertices that were already visited.
- const graph ∗ m_graph

  The graph this iterator traverses.

3.3.1 Detailed Description

template<typename graph, typename waiting_container>class graph_search_iterator< graph, waiting_container >

Base class for search iterators in a graph.

This class provides the base for search iterators on graphs with a container of currently waiting vertices (/vertex iterators) and a list of already visited vertices. Subclasses have to reimplement the next() function to increment the 'iterator'.

Template Parameters

graph	Type of the graph that should be traversed. Required to get the necessary iterator and data types.
waiting_container	Type of the waiting container that should be used. (stack/queue)

3.3.2 Constructor & Destructor Documentation

3.3.2.1 template<typename graph, typename waiting_container> graph_search_iterator< graph, waiting_container >::graph_search_iterator (const graph & *g*, typename graph::graph_vertex_const_iterator *start*) [inline]

Constructs a new graph search iterator.

Constructs a graph search iterator object, initializing it with the supplied values.

Parameters

g	A reference to the graph that should be traversed.
start	Vertex iterator to the node that should be used as a starting point.

3.3.3 Member Function Documentation

3.3.3.1 template<typename graph, typename waiting_container> size_t graph_search_iterator< graph, waiting_container >::discovered ( ) const [inline]

Returns number of discovered vertices.

Returns the number of vertices the search algorithm already traveresed.

Returns

Number of visited vertices.

3.3.3.2 template<typename graph, typename waiting_container> bool graph_search_iterator< graph, waiting_container >::end ( ) const [inline]

Check whether search has finished.

Returns whether the search alghorithms iterated through all accessible vertices.

Returns

true if the search finished.

3.3.3.3 template<typename graph, typename waiting_container> virtual graph::graph_vertex_const_iterator graph_search_iterator< graph, waiting_container >::next ( ) [pure virtual]

Returns iterator to next vertex in the search order.

This method should be reimplemented in subclasses to increment the search iterator to the next the element depending on the search algorithm

Returns

Iterator to the next element

Implemented in depth_first_iterator< graph >, and breadth_first_iterator< graph >.

The documentation for this class was generated from the following file:

- undirected_graph/source/graph_search_iterator.h

3.4 std::hash< undirected_pair< T > > Struct Template Reference

Hash functor for undirected_pair.

`#include <undirected_pair.h>`

3.4.1 Detailed Description

template<typename T>struct std::hash< undirected_pair< T > >

Hash functor for undirected_pair.

This functor provides a specialization of the std::hash functor for an undirected_pair to store them in containers like an unordered_map. To use this functor, the type T has to provide a std::hash specialization itself.

The documentation for this struct was generated from the following file:

- undirected_graph/source/undirected_pair.h

3.5 undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg > Class Template Reference

Undirected graph with vertices and edges.

`#include <undirected_graph.h>`

Public Types

- typedef vertex_container::iterator graph_vertex_iterator

    Iterator for vertices in the graph. It behaves like a std::map iterator with it->first being the vertex id and it->second being the mapped vertex data.
- typedef vertex_container::const_iterator graph_vertex_const_iterator

    Const iterator for vertices in the graph. It behaves like a std::map iterator with it->first being the vertex id and it->second being the mapped vertex data.
- typedef edge_container::iterator graph_edge_iterator

    Iterator for edges in the graph. It behaves like a std::map iterator with it->first being the edge id and it->second being the mapped edge data.
- typedef edge_container::const_iterator graph_edge_const_iterator

    Const iterator for edges in the graph. It behaves like a std::map iterator with it->first being the edge id and it->second being the mapped edge data.
- typedef adjacency_list::iterator graph_adjacency_iterator

    Iterator for adjacencent vertices in the graph. It behaves like a std::forward_list iterator dereferencing to the id of the adjacent vertex.
- typedef adjacency_list::const_iterator graph_adjacency_const_iterator

    Const iterator for adjacencent vertices in the graph. It behaves like a std::forward_list iterator dereferencing to the id of the adjacent vertex.

Public Member Functions

- bool empty () const

    Test whether graph is empty.
- size_t size_vertices () const

    Return vertex container size.
- size_t size_edges () const

    Return edge container size.
- vertex_data_type & at_vertex (const vertex_id_type &id)

    Access vertex.
- edge_data_type & at_edge (const edge_id_type &id)

    Access edge.
- graph_vertex_iterator find_vertex (const vertex_id_type &id)

    Get iterator to vertex.
- graph_edge_iterator find_edge (const edge_id_type &id)

    Get iterator to edge.
- graph_vertex_iterator begin_vertices ()

    Return iterator to beginning of vertices.
- graph_vertex_iterator end_vertices ()

    Return iterator to end of vertices.
- graph_edge_iterator begin_edges ()

    Return iterator to beginning of edges.
- graph_edge_iterator end_edges ()

    Return iterator to end of edges.
- graph_adjacency_iterator begin_adjacent (const vertex_id_type &vertex)

    Return iterator to beginning of adjacent vertices.
- graph_adjacency_iterator end_adjacent (const vertex_id_type &vertex)

    Return iterator to end of adjacent vertices.
- void clear ()

    Clear content.

- std::pair< graph_vertex_iterator, bool > insert_vertex (const vertex_id_type &vertex_id, const vertex_data↩_type &vertex_data)

  Insert vertex.
- bool erase_vertex (const vertex_id_type &vertex_id)

  Erase vertex.
- bool erase_edge (const edge_id_type &edge_id)

  Erase edge.
- std::pair< graph_edge_iterator, bool > insert_edge (const vertex_id_type &vertex_a, const vertex_id_type &vertex_b, edge_data_type edge_data)

  Add edge between two vertices.

Static Public Member Functions

- static edge_id_type make_edge_id (const vertex_id_type &a, const vertex_id_type &b)

  Make edge id.

Private Types

- typedef undirected_graph< vertex_data_type, vertex_id_type, edge_data_type, edge_id_type > graph_type

  Type of the current graph.
- typedef std::unordered_map< vertex_id_type, vertex_data_type > vertex_container

  Type for the container of vertex data in the graph.
- typedef std::unordered_map< edge_id_type, edge_data_type > edge_container

  Type for the container of edge data in the graph.
- typedef std::forward_list< vertex_id_type > adjacency_list

  Type for the adjacency lists of the graph.
- typedef std::unordered_map< vertex_id_type, adjacency_list > adjacency_container

  Type for the container of the adjacency lists in the graph.

Private Attributes

- vertex_container vertices
- edge_container edges
- adjacency_container adjacency

3.5.1 Detailed Description

template<typename vertex_data_type_arg, typename vertex_id_type_arg, typename edge_data_type_arg, typename edge_id_↩type_arg>class undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >

Undirected graph with vertices and edges.

A basic implementation of undirected graphs containing vertices connected by unique edges. There may be multiple vertices with the same value but only one edge between two vertices. The container stores data elements for the vertices and edges. The vertices and edges are identified and sorted by ids which have to be unique in the container.

Template Parameters

vertex_data_type	The type that should be used for the data elements at the vertices.

vertex_id_type	Type for the ids of vertices. Has to support the default comparison operators.
edge_data_type	The type that should be used for the data elments at the edges
edge_id_type	The type for the ids of edges. Has to provide a constructor taking two vertex ids as well as support of the default comparison operators where the order of the two vertices is not important. Furthermore it has to provide access to the two vertex ids with a public a and b member variable.

3.5.2 Member Function Documentation

3.5.2.1 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > edge_data_type& undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::at_edge (const edge_id_type & *id*) `[inline]`

Access edge.

Returns a reference to the mapped data of the edge identified with the specified id. If it does not match the id of any edge in the container, the function throws an out_of_range exception.

Parameters

id	The id of the edge whose mapped data is accessed.

Returns

A reference to the mapped data of the edge.

3.5.2.2 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > vertex_data_type& undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::at_vertex (const vertex_id_type & *id*) `[inline]`

Access vertex.

Returns a reference to the mapped data of the vertex identified with the specified id. If it does not match the id of any vertex in the container, the function throws an out_of_range exception.

Parameters

id	The id of the vertex whose mapped data is accessed.

Returns

A reference to the mapped data of the vertex.

3.5.2.3 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > graph_adjacency_iterator undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::begin_adjacent (const vertex_id_type & *vertex*) `[inline]`

Return iterator to beginning of adjacent vertices.

Returns an iterator referring to the first adjacent vertex of the specified vertex. If the adjacency list is empty, the returned iterator value shall not be dereferenced. It behaves like a forward_list iterator.

Returns

An iterator to the first adjacent vertex in the container.

3.5.2.4 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > graph_edge_iterator undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::begin_edges ( ) [inline]

Return iterator to beginning of edges.

Returns an iterator referring to the first edge in the graph container. If the container is empty, the returned iterator value shall not be dereferenced.

Returns

An iterator to the first edge in the container.

3.5.2.5 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > graph_vertex_iterator undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::begin_vertices ( ) [inline]

Return iterator to beginning of vertices.

Returns an iterator referring to the first vertex in the graph container. If the container is empty, the returned iterator value shall not be dereferenced.

Returns

An iterator to the first vertex in the container.

3.5.2.6 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > void undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::clear ( ) [inline]

Clear content.

Removes all elements from the graph container (which are destroyed), leaving the container with a size of 0.

3.5.2.7 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > bool undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::empty ( ) const [inline]

Test whether graph is empty.

Returns whether the graph is empty (i.e. whether there are no vertices). This function does not modify the graph in any way. To clear the content of a graph container, see undirected_graph::clear.

Returns

true if the there no vertices, false otherwise.

3.5.2.8 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > graph_adjacency_iterator undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::end_adjacent (const vertex_id_type & *vertex*) [inline]

Return iterator to end of adjacent vertices.

Returns an iterator referring to the past-the-end adjacent vertex to the specified vertex. It does not point to any element, and thus shall not be dereferenced. If the adjacency list is empty, this function returns the same as undirected_graph::begin_adjacent. It behaves like a forward_list iterator.

Returns

An iterator to the past-the-end adjacent vertex of the specified vertex.

3.5.2.9 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > graph_edge_iterator undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::end_edges ( ) [inline]

Return iterator to end of edges.

Returns an iterator referring to the past-the-end edges in the graph container. It does not point to any element, and thus shall not be dereferenced. If the container is empty, this function returns the same as undirected_graph↩ ::begin_edges.

Returns

An iterator to the past-the-end edge in the container.

3.5.2.10 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > graph_vertex_iterator undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::end_vertices ( ) [inline]

Return iterator to end of vertices.

Returns an iterator referring to the past-the-end vertex in the graph container. It does not point to any element, and thus shall not be dereferenced. If the container is empty, this function returns the same as undirected_graph↩ ::begin_vertices.

Returns

An iterator to the past-the-end vertex in the container.

3.5.2.11 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > bool undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::erase_edge (const edge_id_type & *edge_id*) [inline]

Erase edge.

Removes a single edge from the graph container. This effectively reduces the edge container size by one and the edge data is destroyed.

Parameters

edge_id	Id of the edge that should be removed.

Returns

Returns whether the edge was removed.

3.5.2.12 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > bool undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::erase_vertex (const vertex_id_type & *vertex_id*) [inline]

Erase vertex.

Removes a single vertex from the graph container. This effectively reduces the vertex container size by one and the vertex data is destroyed. Also all edge data connected to this vertex is destroyed.

Parameters

vertex_id	Id of the vertex that should be removed.

Returns

Returns whether the vertex was removed.

3.5.2.13 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > graph_edge_iterator undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::find_edge (const edge_id_type & *id*) [inline]

Get iterator to edge.

Searches the container for an edge with an id equivalent to the one specified and returns an iterator to it if found, otherwise it returns an iterator to undirected_graph::end_edges. Two ids are considered equivalent if the container's comparison object returns false reflexively (i.e., no matter the order in which the ids are passed as arguments).

Parameters

id	Id to be searched for.

Returns

An iterator to the edge, if an edge with specified id is found, or undirected_graph::end_edges otherwise.

3.5.2.14 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > graph_vertex_iterator undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::find_vertex (const vertex_id_type & *id*) [inline]

Get iterator to vertex.

Searches the container for a vertex with an id equivalent to the one specified and returns an iterator to it if found, otherwise it returns an iterator to undirected_graph::end_vertices. Two ids are considered equivalent if the container's comparison object returns false reflexively (i.e., no matter the order in which the ids are passed as arguments).

Parameters

id	Id to be searched for.

Returns

An iterator to the vertex, if a vertex with specified id is found, or undirected_graph::end_vertices otherwise.

3.5.2.15 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > std::pair<graph_edge_iterator, bool> undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::insert_edge (const vertex_id_type & *vertex_a*, const vertex_id_type & *vertex_b*, edge_data_type *edge_data*) [inline]

Add edge between two vertices.

This method inserts a new edge data element to the graph connecting two vertices. If the edge already exists the edge will not be modified.

Parameters

vertex_a	Id of the first vertex to connect.
vertex_b	Id of the second vertex to connect.
edge_data	The data element for the edge between the two vertices.

Returns

Returns a pair with an iterator to the inserted edge and a bool indicating whether the edge was inserted or not.

3.5.2.16 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > std::pair<graph_vertex_iterator, bool> undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::insert_vertex (const vertex_id_type & *vertex_id*, const vertex_data_type & *vertex_data*) [inline]

Insert vertex.

Inserts a new vertex to the graph, effectively increasing the container size by one. Multiple vertices with the same value may exist in one graph but ids have to be unique.

Parameters

vertex_id	The id of the vertex.
vertex_data	Value to be copied to the inserted vertex.

Returns

Returns a pair with an iterator to the inserted vertex and a bool indicating whether the vertex was newly inserted or not.

3.5.2.17 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > static edge_id_type undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::make_edge_id (const vertex_id_type & *a*, const vertex_id_type & *b*) [inline], [static]

Make edge id.

Creates the edge id for the edge between the two specified vertex ids. The two vertex ids don't have to be present in the graph.

Parameters

a	First vertex id.
b	Second vertex id.

Returns

Edge id for the edge between the two vertex ids.

3.5.2.18 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > size_t undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::size_edges ( ) const [inline]

Return edge container size.

Returns the number of edges in the undirected_graph container.

Returns

The number of edges in the graph.

3.5.2.19 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > size_t undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::size_vertices ( ) const `[inline]`

Return vertex container size.

Returns the number of vertices in the undirected_graph container.

Returns

The number of vertices in the graph.

3.5.3 Member Data Documentation

3.5.3.1 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > adjacency_container undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::adjacency `[private]`

Container for adjacency lists

3.5.3.2 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > edge_container undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::edges `[private]`

Container for the edge data

3.5.3.3 template<typename vertex_data_type_arg , typename vertex_id_type_arg , typename edge_data_type_arg , typename edge_id_type_arg > vertex_container undirected_graph< vertex_data_type_arg, vertex_id_type_arg, edge_data_type_arg, edge_id_type_arg >::vertices `[private]`

Container for the vertex data

The documentation for this class was generated from the following file:

- undirected_graph/source/undirected_graph.h

3.6 undirected_pair< T > Class Template Reference

Undirected pair.

```
#include <undirected_pair.h>
```

Public Member Functions

- undirected_pair ()=default

  Constructs an empty undirected pair.
- undirected_pair (T a_in, T b_in)

  Constructs an undirected pair with the specified objects as data.
- T & other_element (const T &compare)

  Returns a reference to the other data element if compare is the same as one of the elements in the undirected pair.

- const T & other_element (const T &compare) const

  Returns a const reference to the other data element if compare is the same as one of the elements in the undirected pair.
- T & smaller_element ()

  Returns a reference to the smaller of the two elements in the undirected pair or element a.
- const T & smaller_element () const

  Returns a const reference to the smaller of the two elements in the undirected pair or element a.
- T & bigger_element ()

  Returns a reference to the bigger of the two elements in the undirected pair or element b.
- const T & bigger_element () const

  Returns a const reference to the bigger of the two elements in the undirected pair or element b.
- bool operator== (const undirected_pair &rhs) const

  Returns whether the two undirected pairs contain the same data elements.
- bool operator!= (const undirected_pair &rhs) const

  Returns whether the two pairs contain different data elements.
- bool operator< (const undirected_pair &rhs) const

  Strict weak ordering for undirected pairs by the smallest element in the pairs.

Public Attributes

- T a
- T b

3.6.1 Detailed Description

template<typename T>class undirected_pair< T >

Undirected pair.

Pair whose comparison operators does not differentiate between pair(a,b) and pair(b,a). Suitable as the edge id type for an undirected graph

3.6.2 Member Data Documentation

3.6.2.1 template<typename T> T undirected_pair< T >::a

Element a of the undirected pair

3.6.2.2 template<typename T> T undirected_pair< T >::b

Element b of the undirected pair

The documentation for this class was generated from the following file:

- undirected_graph/source/undirected_pair.h

Index